



Internet of Things for Industry and Human Applications
VOLUME 2
MODELLING AND DEVELOPMENT



Internet of Things for Industry and Human Applications

Volume 2
Modelling and Development



Funded by the
Erasmus+ Programme
of the European Union

Ministry of Education and Science of Ukraine
National Aerospace University “Kharkiv Aviation Institute”

**Internet of Things
for
Industry and Human Applications**

Volume 2

Modelling and Development

Edited by V. S. Kharchenko

Project ERASMUS+ ALIOT
“Internet of Things:
Emerging Curriculum for Industry and Human Applications”
(573818-EPP-1-2016-1-UK-EPPKA2-CBHE-JP)

2019

UDC 62:004=111

I73

Reviewers: Dr. Mario Fusani, ISTI-CNR, Pisa, Italy

Dr. Olga Kordas, KTH University, Stockholm, Sweden

Viktor Kordas, KTH University, Stockholm, Sweden

I73 Internet of Things for Industry and Human Application. In Volumes 1-3. Volume 2. Modelling and Development /V. S. Kharchenko (ed.) - Ministry of Education and Science of Ukraine, National Aerospace University KhAI, 2019. - 547p.

ISBN 978-617-7361-80-9

ISBN 978-617-7361-82-3

Three-volume book contains theoretical materials for lectures and training modules developed in frameworks of project “Internet of Things: Emerging Curriculum for Industry and Human Applications /ALIOT” (Project Number: 573818-EPP-1-2016-1-UK-EPPKA2-CBHE-JP, 2016-2019) funded by EU Program ERASMUS+. Volume 2 describes models, simulation and development techniques for Internet of Things (IoT). The book consists of 4 parts for corresponding PhD courses: modelling of IoT based systems (sections 16-19), software defined networks and IoT (sections 20-23), dependability and security of IoT (sections 24-27), development and implementation of IoT based systems (sections 28-31). The book prepared by Ukrainian university teams with support of EU academic colleagues of the ALIOT consortium.

The book is intended for MSc and PhD students studying IoT technologies, software and computer engineering and science, cyber security. It could be useful for lecturers of universities and training centers, researchers and developers of IoT systems.

Fig.: 158. Ref.: 430. Tables: 45.

Approved by Academic Council of National Aerospace University “Kharkiv Aviation Institute” (record № 4, December 19, 2018).

UDC 62:004=111

ISBN 978-617-7361-82-3

© O.V.Drozd, O.O.Illiashenko, V.S.Kharchenko, M.O.Kolisnyk, G.V.Kondratenko, Yu.P.Kondratenko, O.Yu.Maevskaya, D.A.Maevsky, O.M.Martyniuk, D.S.Mazur, M.V.Nesterov, A.P.Plakhteyev, V.V.Shkarupylo, Ie.V.Sidenko, I.S.Skarga-Bandurova, V.V.Sklyar, G.V.Tabunshchik, M.O.Taranov, A.Y.Velykzhanin, D.D.Uzun, Y.O.Uzun, N.G.Yatskiy, V.V.Yatskiy, H.A.Zemlianko

This work is subject to copyright. All rights are reserved by the authors, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms, or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed

Міністерство освіти і науки України
Національний аерокосмічний університет
ім. М. С. Жуковського «Харківський Авіаційний Інститут»

**Інтернет речей
для
індустріальних і гуманітарних застосунків**

Том 2

Моделювання і розроблення

Редактор Харченко В.С.

Проект ERASMUS+ ALIOT
“Інтернет речей: нова освітня програма для потреб
промисловості та суспільства”
(573818-EPP-1-2016-1-UK-EPPKA2-CBHE-JP)

2019

УДК 62:004=111

173

Рецензенти: Др. Маріо Фузані, ISTI-CNR, Піза, Італія
Др. Ольга Кордас, KTH University, Стокгольм, Швеція
Віктор Кордас, KTH University, Stockholm, Sweden

173 Інтернет речей для індустріальних і гуманітарних застосунків. У трьох томах. Том 2. Моделювання і розроблення / За ред. В. С. Харченка. – Міністерство освіти і науки України, Національний аерокосмічний університет ХАІ, 2019. – 547 с.

ISBN 978-617-7361-80-9

ISBN 978-617-7361-82-3

Книга, що складається з трьох томів, містить теоретичні матеріали для лекцій та тренінгів, розроблених в рамках проекту Internet of Things: Emerging Curriculum for Industry and Human Applications / ALIOT, 573818-EPP-1-2016-1-UK-EPPKA2-SBHE-JP, 2016-2019, що фінансується програмою ЄС ERASMUS+. Том 2 описує моделі, методи моделювання та розробки для Інтернету речей (IoT). Книга складається з 4 частин для відповідних докторантських курсів: моделювання систем на основі IoT (розділи 16-19), програмно-визначувані мережі і IoT (розділи 20-23), надійність і безпека IoT (розділи 24-27), розроблення і впровадження систем на основі IoT (розділи 28-31).

Книга підготовлена українськими університетськими командами за підтримки колег з академічних закладів країн ЄС, що входять до консорціуму проекту ALIOT.

Книга призначена для магістрантів і аспірантів, які вивчають технології IoT, програмну і комп'ютерну інженерію, комп'ютерні науки. Може бути корисною для викладачів університетів і навчальних центрів, дослідників і розробників систем IoT.

Рис.: 158. Посилань: 430. Таблиць: 45.

Рекомендовано до видання вченою радою Національного аерокосмічного університету імені М.Є. Жуковського «Харківський авіаційний інститут» (протокол № 4 від 19 грудня 2018 г.).

УДК 62:004=111

ISBN 978-617-7361-82-3

© О.В.Дрозд, О.О.Лляшенко, В.С.Харченко, М.О.Колісник, Г.В.Кондратенко, Ю.П.Кондратенко, О.Ю.Маєвська, Д.А.Маєвській, О.М.Мартинюк, Д.С.Мазур, М.В.Нестеров, А.П.Плахтєв, В.В.Шкарапило, Є.В.Сіденко, І.С.Скарга-Бандурова, В.В.Скляр, Г.В.Табунщик, М.О.Таранов, А.Ю.Велижканін, Д.Д.Узун, Ю.О.Узун, Н.Г.Яцків, В.В.Яцків, Г.А.Землянко

Ця робота захищена авторським правом. Всі права зарезервовані авторами, незалежно від того, чи стосується це всього матеріалу або його частини, зокрема права на переклади на інші мови, перевидання, повторне використання ілюстрацій, декламацію, трансляцію, відтворення на мікрофільмах або будь-яким іншим фізичним способом, а також передачу, зберігання та електронну адаптацію за допомогою комп'ютерного програмного забезпечення в будь-якому вигляді, або ж аналогічним або іншим відомим способом, або ж таким, який буде розроблений в майбутньому.

CONTENTS

PREFACE.....	7
PART V. SIMULATION OF IoT AND IoE-BASED SYSTEMS.....	14
16. PROGRAM TOOLS FOR THE SMART SYSTEMS SIMULATION	14
16.1 Basic principles of IoT simulations	16
16.2 Simulation IoT devices based on Arduino platform.....	25
16.3 Software development. Arduino C/C++ sketch.....	31
16.4 Work related analysis	36
17. THREE-LEVEL SIMULATION OF IOT/IOE BASED SYSTEMS WITH THE USE OF UML DIAGRAMS, PETRI NETS AND TEMPORAL LOGIC	41
17.1. Simulation and verification in architecture of IoT and IoE-based systems with the use of visual UML diagrams	43
17.2 Simulation and verification in behavior of IoT and IoE systems on the basis of the Queuing Systems and Petri Nets	53
17.3 Simulation and verification of synchronization processes in IoT and IoE-based systems on the basis of temporal logic	64
17.4 Work related analysis	67
18. MARKOV’S MODELLING OF IOT SYSTEMS	76
18.1 Features of Markov’s modeling of IoT systems	78
18.2 Markov’s modeling of IoT systems reliability and availability....	85
18.3 Markov’s modeling of IoT systems cyber security and availability	93
18.4 Semi Markov’s modeling of IoT systems.....	99
18.5 Work related analysis	105
19. INTERACTION SIMULATION FOR IOT SYSTEMS	110
19.1 Interaction in IoT systems	112
19.2 Interaction Flow Modelling Language	118
19.3. Case Study	119
19.4 Work related analysis	131
PART VI. SOFTWARE DEFINED NETWORKS AND IOT	135
20. SOFTWARE DEFINED NETWORKS BASICS	135

20.1 SDN architecture. Fundamental notions, principles and concepts	137
20.2 An in-depth look at the aspects of implementation. Differentiation between Control and Data Planes	142
20.3 OpenFlow protocol. The basics, peculiarities and limitations....	150
20.4 Work related analysis	161
21. SDN PROGRAMMING AND SIMULATION OF SDN COMPOSING, CONFIGURING AND SCALING	165
21.1 On the peculiarities of SDN switches and controllers functioning and implementation	167
21.2 Network programming and testing	172
21.3 SDN programming and Python scripting	178
21.4 Work related analysis	186
22. ALGORITHMS AND APPLICATIONS FOR UTILIZATION OF SDN TECHNOLOGY TO IOT	194
22.1 Managing the IoT with SDN	196
22.2 Smart routing and scheduling	198
22.3 Optimization of SDN Traffic Flow for IoT	206
22.4 SDN Performance prediction.....	219
22.5 Work related analysis	234
23. SDN IN CONTEXT OF DEVOPS TECHNOLOGY	241
23.1 DevOps technology overview	243
23.2 DevSecOpS.....	255
23.3 SDN and DevOpS.....	260
23.4 DevOpS and IoT	271
23.5 Work related analysis	278
PART VII. DEPENDABLITY AND SECURITY OF IOT	283
24. DEPENDABILITY AND SECURITY MODELS OF IOT	283
24.1. Dependability and security concepts for IoT.....	285
24.2 Dependability and safety models for IoT	290
24.3 Security models for IoT.....	302
24.4 Work related analysis	312
25. SAFETY AND SECURITY MANAGEMENT OF IOT	317
25.1 Safety and security management requirements to IoT	319
25.2 Safety and security life cycle for IoT	329

25.3 Review, analysis and testing techniques for IoT	334
25.4 Work related analysis	337
26. ASSURANCE CASE FOR IOT	341
26.1. Assurance Case fundamentals	343
26.2. Safety and security techniques and measures for IoT.....	347
26.3. Security informed and energy efficiency informed Assurance Case for IoT	357
26.4 Work related analysis	363
27. SECURITY OF IOT BASED BLOCKCHAIN TECHNOLOGY	368
27.1. Bases of blockchain technology and examples of application ..	370
27.2 Consensus algorithms in blockchain technology	377
27.3 Blockchain technology for the IoT security	384
27.4 Work related analysis	394
PART VIII. DEVELOPMENT AND IMPLEMENTATION OF IOT-BASED SYSTEMS	403
28. BASIC CONCEPTS AND APPROACHES TO DEVELOPMENT AND IMPLEMENTATION OF IOT SYSTEMS	403
28.1 IoT-based system development process	405
28.2 Strategies to planning IoT architectures	413
28.3 The base components of the IoT systems	419
28.4 The IoT development boards and platforms for prototyping.....	426
28.5 The IoT platforms: types and selection criteria	429
28.6 Work related analysis	431
29. MODELS FOR IOT-BASED DEVICES AND TECHNOLOGIES FOR DATA PROCESSING AND TRANSFER	436
29.1 IoT-based devices: models and network communication protocols	438
29.2 Technologies for data processing in IoT-based systems	447
29.3 Protocols and standards for data transfer between IoT-based devices	456
29.4 Work related analysis	464
30. INTELLIGENT METHODS AND APPROACHES FOR MANAGEMENT AND LEARNING OF IOT-BASED SYSTEMS	470
30.1 Management systems and IoT platforms	472

30.2 Multi-agent approach for development and management of IoT systems	482
30.3 Methods and approaches for learning of IoT-based systems.....	490
30.4 Work related analysis	497
31. PROTOTYPING AND RAPID DEVELOPMENT OF IOT SYSTEMS	503
31.1 IoT devices	505
31.2 Prototyping and rapid development principles	511
31.3 Cases of IoT systems rapid development	519
31.4 Work related analysis	530
Анотації	536
Аннотации	542

PREFACE

ALIOT ERASMUS+ project. Three-volume book contains material for lectures and training modules developed during carrying out of project “**Internet of Things: Emerging Curriculum for Industry and Human Applications /ALIOT¹**” 1(Project Number: 573818-EPP-1-2016-1-UK-EPPKA2-CBHE-JP, 2016-2019) funded by EU Program ERASMUS+. Main ALIOT project objectives are development and transfer of innovative Internet of Things (IoT) and Internet of Everything (IoE) related research ideas and practices between the academic and industrial sectors and for society as whole.

The tasks of the ALIOT project are the following:

1) to introduce a Multi-domain and Integrated Internet of Things (IoT) programme and develop 4 courses for MSc students:

- MC1 Fundamentals of IoT and IoE,
- MC2 Data science for IoT and IoE,
- MC3 Mobile and hybrid IoT-based computing,
- MC4 IoT technologies for cyber physical systems;

2) to introduce a Multi-Domain and Integrated IoT programme and develop 4 courses for doctoral students:

- PC1 Simulation of IoT and IoE-based systems,
- PC2 Software defined networks and IoT,
- PC3 Dependability and security of IoT,
- PC4 Development and implementation of IoT-based systems;

3) to establish multi-domain IoT cluster network and develop 6 training courses for human and industry applications:

- ITM1 IoT for smart energy grid,
- ITM 2 IoT for smart building and city,
- ITM 3 IoT for intelligent transport systems,
- ITM 4 IoT for health systems,
- ITM 5 IoT for ecology monitoring systems,
- ITM 6 IoT for industrial systems.

¹ *The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.*

The tasks of the project have been solved by ALIOT consortium of Ukraine and EU countries universities and organizations:

- Newcastle University (NU), United Kingdom (grant holder and EU coordinator);
- National Aerospace University "Kharkiv Aviation Institute" (KhAI), Ukraine (national coordinator);
- Leeds Beckett University (LBU), United Kingdom;
- Coimbra University (CU), Portugal;
- University KTH, Stockholm, Sweden;
- Institute of Information Science and Technologies ISTI-CNR, Pisa, Italy;
- Chernivtsi National University (ChNU), Ukraine;
- East Ukraine National University (EANU), Ukraine;
- Odesa National Polytechnic University (ONPU), Ukraine;
- Ternopil National Economic University (TNEU), Ukraine;
- Petro Mohyla Black Sea National University (PMBSNU), Mykolaiv, Ukraine;
- Zaporizhzhya National Technical University (ZNTU), Ukraine;
- Pukhov Institute for Modelling in Energy Engineering (IPME), National Academy of Science of Ukraine, Kyiv, Ukraine;
- IT-Alliance (ITA), Ukraine;
- Smart.ME company (SM), Ukraine.

ALIOT books. To assure the ALIOT courses the following books are edited:

- Three volume multi-book "Internet of Things for Industry and Human Applications" for theoretical/lecture part of courses:
 - Volume 1. Fundamentals and Technologies (MSc study),
 - Volume 2. Modelling and Development (PhD study),
 - Volume 3. Assessment and Implementation (training modules);
- 4 practicum books for MSc courses;
- 4 practicum books for PhD courses;
- 6 books for domain oriented training modules.

The volumes consists of 14 parts according with list of MSc (Parts I-IV), PhD (Parts V-VIII) and training (Parts IX-XIV) courses. Parts are called according with corresponding courses (Parts I-IV as MC1-MC4, Part V-VIII as PC1-PC14, Parts IX-XIV as ITM1-ITM6).

Parts consist of the sections 1-56 (4 sections for courses MC1-MC2, MC4, PC1-PC4, ITM1-ITM5; 3 sections MC3, 5 sections for ITM6). Section 0 introduces into the multi-book.

Contents and authors of the Volume 2. Volume 2 consists of parts V-VIII, sections 16-31.

PART V. SIMULATION OF IOT AND IOE-BASED SYSTEMS.

Section 16 is devoted to the description of the general principles of functioning of the Arduino board and the simulation of its work. The differences between physical and computer simulation are shown. The simulation methods that can be applied to the ARDUINO boards are described. A comparative analysis of various software tools that can be used for simulation is given. The operation with the PROTEUS software package is described in detail.

Author of the section 16 are Assoc. Prof., Dr. O. Yu. Maevskaya (ONPU), Prof., DrS D. A. Maevsky.

Three-level simulation of IoT/IoE based systems in their structure, behavior and processes of synchronization is considered in section 17. Visual modeling, simulation and verification of architectures, functionality and temporal features of IoT/IoE based systems and their components in static and dynamic modes with the use of UML diagrams, Petri nets, temporal logic, corresponding methods and tools. The features of the simulation based on evolutionary genetic and multi-agent technologies, techniques and tools is suggested.

Authors of the section 17 are Assoc. Prof., Dr. O. M. Martynyuk, Prof., DrS. O. V. Drozd (ONPU).

Section 18 describes the features of development of Markov and semi-Markov models for research of the Internet of Things operation and assessment of availability, cyber security and dependability. Models are developed for typical IoT devices (switches, hubs, UBS) and system (Smart Business Centre) as a whole. Markov models have been used to assess SBC cyber security under DoS/DDoS attacks.

Authors of the section 18 are Assoc. Prof., Dr. M. O. Kolisnyk, Prof., DrS. V. S. Kharchenko (KhAI)

Section 19 is devoted to the interaction simulation in IoT systems. In the chapter the common architecture of the IoT systems are considered as well as patterns of the simulation of the interactions. As for simulation of the different interactions could be used variety of

techniques and tools, authors suggested several use cases. The use case with remote laboratory GOLDi demonstrate the usage of the FSM models and Kripke model, in the use case with smart campus there is considered implementation of the IFML models for modelling interactions with the users, for simulation of the cyber-physical systems could be used digital twins, which was shown in the examples with the ISTR system.

Author of the section 19 is Prof., Dr. G.V. Tabunshchyk (ZNTU).

PART VI. SOFTWARE DEFINED NETWORKS AND IOT.

Section 20 considers the fundamentals of software-defined networking (SDN), principles of composition and functioning, technologies, architectures. The accent is also put on the features of technology, historical premises that have prompted the emerging of SDN paradigm. OpenFlow specification evolution process, forming the basis for granting the unified mechanism of communication between the controller and switches have been analysed.

Authors of the sections 20 are Dr. V. V. Shkarupylo, MSc student D. S. Mazur (ZNTU).

In section 21, the principles of software defined networks programming and simulation are considered. The aspects of programming have been covered on the basis of Python programming language. Basic commands for network topology configuration are given, the commands for resolving the automation tasks in particular. The tips on Mininet environment and corresponding MiniEdit graphical tool usage have been provided.

Authors of the section 21 is Dr. V. V. Shkarupylo (ZNTU).

Section 22 deals with a series of research problems related to the implementing specific QoS models over SDN by developing and implementing algorithms and approaches supplying efficient operation of SDN in IoT. Recent trends in algorithms utilization for SDN technology were analyzed in terms of their suitability for establishing and maintenance large-scale backbone SDN/OpenFlow networks within IoT infrastructure. Perspectives on SDN performance prediction using data fusion technique are discussed.

Authors of the section 22 are Prof., DrS. I. S. Skarga-Bandurova, PhD student M. V. Nesterov, PhD student A. Y. Velykzhanin (EUNU).

Section 23 focuses on DevOps principles and practices supported on the well-known platforms, like AWS, MS Azure, Google Cloud, etc.

A brief introduction to the origins of methodology DevOps sets the scene and explains how and why DevOps has evolved. Interconnection of DevOps, Software Defined Networks (SDN) and IoT is analysed.

Authors of the section 23 are, Assoc. Prof., Dr. D. D. Uzun, Y.O. Uzun, Prof., DrS. V. S. Kharchenko (KhAI).

PART VII. DEPENDABILITY AND SECURITY OF IOT.

Dependability and security models for IoT systems are considered in the section 24. In frame of dependability and security concept we propose the taxonomy of safety and security requirements, after to represent dependability, safety and security attributes and risks analysis fundamentals. Dependability and safety models are mostly quantitative based on probabilistic analysis of indicators values. Security models are mostly qualitative based on threats analysis and the attacks scenario.

Authors of the section 24 are Prof., DrS. V. V. Sklyar, Prof., DrS. V. S. Kharchenko (KhAI).

Section 25 considers safety and security management requirements including human resource management, configuration management, tools selection and evaluation, documentation management, and safety and security assessment. Also V-shape Safety and Security Life Cycle is represented in details including requirements tracing. Finally, the main issues of verification techniques including documents review, static code analysis, functional and structural testing are considered.

The Assurance Case methodology is considered in section 26 as an integral approach to integrate safety and security requirements and artefacts. For that, the Assurance Case fundamentals as well as concept and history are represented. For graphical representation of the Assurance Case, semi-formal notations such as Claim, Argument and Evidence (CAE) and Goal Structuring Notation (GSN) are used. Security informed and energy efficiency informed Assurance Case consists features appropriated to IoT systems.

Author of the sections 25, 26 is Prof., DrS. V. V. Sklyar (KhAI).

Section 27 considers the basics of blockchain technology and examples of implementation in the Internet of things. The consensus algorithms used in the blockchain technology and the principles of ensuring the Internet of things safety and security using the blockchain technology are discussed. The advantages and the existing problems of the blockchain technology integration in the Internet of things are highlighted. Resolving the security problem at different levels of IoT

application is a more complex issue due to the lack of performance and high heterogeneity of devices.

Authors of the section 27 are Prof., DrS. V. V. Yatskiv, Ass. Prof., Dr. N. G. Yatskiv (TNEU).

PART VIII. DEVELOPMENT AND IMPLEMENTATION OF IOT-BASED SYSTEMS.

Section 28 deals with a series of research problems related to the developing IoT architectures, device architectures, and IoT-based system integration. Efficient strategies and approaches to overcome essential challenges in the development and implementation of an efficient IoT solution are considered. The base components of the IoT systems, phases and deliverables of an IoT technical strategy as well as selection criteria for IoT platforms deployment, are discussed.

Authors of the section 28 are Prof., DrS. I. S. Skarga-Bandurova, PhD student A. Y. Velykzhanin (EUNU).

Models for IoT-based devices and technologies for data processing and transfer are considered in the section 29. This section discusses the basic principles of constructing information models of IoT-based devices and tools for their creation, in particular Eclipse Vorto. Also analyzed network communication protocols for IoT-based devices. In addition, an important component of the IoT network is the choice of data processing technologies for IoT based systems and methods of management and forecasting. The protocols and standards for data transfer between IoT-based nodes and their cybersecurity are discussed.

Intelligent methods and approaches for management and learning of IoT-based systems are considered in the section 30. It discusses the types and capabilities of IoT platforms, multi-criteria approach and soft computing for choosing the IoT platform. Also analyzed the concept of multi-agent approach in IoT, in particular, types and characteristics of agents, communication agents with the external environment and data transfer techniques between agents. In addition, an important component of the IoT network is the choice of methods and approaches for learning of IoT-based systems. Also considered general principles of M2M learning, self-learning systems and neural networks.

Authors of the sections 29 and 30 are Prof., DrS. Yu. P. Kondratenko, Ass. Prof., Dr. G. V. Kondratenko, Ass. Prof., Dr. Ie. V. Sidenko, PhD Student M. O. Taranov (PMBSNU).

In Section 31 models of information exchange of elements of the IoT systems are considered. The order of development and fast prototyping of devices is given. Standard solutions for creation of the IoT systems, use of virtual devices for software development are shown. Examples of development and prototyping of the channel of measurements on the basis of low-resource microcontrollers are given. Acceleration of development of the IoT device with use of modern open platforms and libraries of high-level functions is shown.

Author of the section 31 is Assoc. Prof., Dr A. P. Plakhteyev, MSC student H. Zemlianko (KhAI).

Volumes 1-3 edited by Prof., DrS. V. S. Kharchenko (KhAI). Camera-ready versions of Volumes 1-3 were prepared by Dr. O. O. Illiashenko (KhAI).

Acknowledgements. The editor and authors would like to express their appreciation and gratitude to all colleagues from partner universities and organizations for discussion, advises and support.

We thank colleagues who develop the project ERASMUS+ ALIOT “Internet of Things: Emerging Curriculum for Industry and Human Applications” <http://aliot.eu.org/> and participate in discussions of topics related to IoT during a few meetings and schools in Sweden (Stockholm, December 2016), Ukraine (February 2017, 2018, Chernivtsi; May 2017, Mykolaiv; May 2018, Kyiv; February 2019, Ternopil; May 2019, Zaporizhzhya), Portugal (Coimbra, October 2017), United Kingdom (Newcastle-Leeds, July 2018).

We thank participants of International Workshops on Cyber Physical Systems and Internet of Things Dependability (WS CyberIoT-DESSERT) at the conferences IDAACS (September 2017, Bucharest, Romania), DESSERT (May 2018, Kyiv, Ukraine) and monthly Seminar on Critical Computer Technologies and Systems (CriCTechS, KhAI, 2017-2019) at the Department of Computer Systems, Networks and Cybersecurity for discussion of preliminary project results in point of view research, development and education issues.

We would like to thank reviewers of the multi-book:

- Dr. Mario Fusani (ISTI-CNR, Pisa, Italy);
- Dr. Olga Kordas (KTH University, Stockholm, Sweden)
- Senior Project Manager Viktor Kordas (KTH University, Stockholm, Sweden)

for very helpful advises and valuable recommendations.

PART V. SIMULATION OF IoT AND IoE-BASED SYSTEMS

16. PROGRAM TOOLS FOR THE SMART SYSTEMS SIMULATION

Assoc. Prof., Dr O. Yu. Maevskaya, Prof., DrS D. A. Maevsky (ONPU)

Contents

PREFACE.....	7
Abbreviations	14
16.1 Basic principles of IoT simulations	16
16.1.1 Classification and terminology. Real-world objects and kinds of its simulation	18
16.1.2 Physical and computer simulations	20
16.1.3 Virtual simulation. Common user interaction systems for virtual simulations.....	22
16.2 Simulation IoT devices based on Arduino platform.....	25
16.2.1 General information about the Arduino platform.....	26
16.2.2 Arduino and Arduino-compatible boards.....	27
16.2.3 Technical characteristics Arduino Mega	28
16.2.4 Inputs and outputs of Arduino Mega 2560.....	29
16.3 Software development. Arduino C/C++ sketch.....	31
16.3.1 General methodology of Arduino sketch working	31
16.3.2 Arduino sketch example	34
16.4 Work related analysis	36
Conclusions and questions.....	37
References	38

Abbreviations

AC – Alternating Current

DC – Direct Current

GUI – Graphical User Interface

IoT – Internet of Things

MQTT – Message Queuing Telemetry Transport

OPC – Open Platform Communications

OPC UA – Open Platform Communications Unified Architecture

REST – REpresentational State Transfer

XML – eXtensible Markup Language

In this section, we will consider software tools for modeling smart IOT systems. The terms "modeling" and "model" we use so often in everyday life that we even do not think what they mean. Everything seems pretty intuitive and clear. In the headlines of scientific works on Engineering Sciences, the phrase "models and methods" is used almost more often than all other. A popular online resource <https://ieeexplore.ieee.org/> produces so more than 1 million 200 thousand results if you specify "Modeling" in the search machine. This word is the most often found in the titles of the reports at conferences (more than 900 thousand) and in the titles of articles in magazine (more than 200 thousand). A part of these publications is about 25% of the total number of publications posted on this Internet resource.

However, not everyone knows that modeling as a process of scientific search has its own laws, rules and varieties. These laws and regulations are the subject of a separate branch of science called "Theory of Modeling and Simulation" [1, 2, 3, 4]. The main provisions of this science bring to the exact science elements of the philosophical thought and try to answer the question, how the world around us with its models relate to each other. This is a difficult question, but, without an answer, we risk losing the connection between the real object and its model. Nobody needs simulation results when the model is not correct, is not it? Therefore, firstly we will try to understand the terminology and model types, as well as the differences between physical and computer modeling.

16.1 Basic principles of IoT simulations

First of all, it should be noted that there are two separate terms in English – "Modeling" (or the American version - "modelling", with two letters "l") and "Simulation". These terms are not synonymous in English and are used to indicate two different processes. According to the English Wikipedia [5], "Modeling" is the process of creating a certain model, for instance, mathematical. In this process, for any natural phenomenon, based on known physical laws, a model is created. Often this model does not describe the phenomenon as a whole, but reflects the law of variation of a certain characteristic (parameter) of this phenomenon. For example, for the process of launching an artificial satellite vehicle into orbit, such characteristics may be the law of speed variation of the launch vehicle over time, or the law of altitude

variation over time. Two separate models are created for these two parameters. Another example is the transient simulation in electrical circuits. Here also this process is not modeled completely, to the contrary we find the change laws over time of certain currents or voltages acting in this circle.

The result of the process, which in English is referred to "Modeling", is a certain model. In IoT, it is often a mathematical model, that is, simply put, a certain formula, equation, or set of equations. The word "Simulation" in English means the process of practical application of the developed model. Here mathematical model is forced to work as a rule, it is done by means of computer facilities. The mathematical model is implemented in computer software. During the simulation, the model is substituted with certain values, and the program, according to the given mathematical equation, so that the result is calculated. In Russian, the model creation and its application are also formally different and the corresponding English terms are used for them. However, usually, the most common is the term "Modeling", which refers to the process of using the already developed model. While using "Modeling" in English sense, it is often about "model creation".

Nevertheless, science does not like discrepancies. Therefore, in this section we will use the terms "Modeling" and "Simulation" in their conventional sense – modeling is the process of a model creation, while simulation is the process of its use.

Internet of things systems are very complex systems that combine electronic and mechanical devices, computers and information transfer devices. It combines advanced technologies for creating electromechanical devices and advanced information technology. This leads to the fact that the construction of a mathematical model of IoT devices (modeling) is difficult and, in many cases, impossible task. The only solution of this problem is a development of independent mathematical models for individual IoT devices and a storage in the specialized databases. Simulation of IoT systems is performed in several stages.

Firstly, the user creates a block diagram of a particular system, which shows, how the individual devices of the system are connected and interact with each other.

In the second stage of the simulation, the user selects the specific types and characteristics of all elements of the IoT system. Element

types are references to specific, pre-prepared mathematical models that are stored in a database. The characteristics of the selected elements are the parameters of these models, which adjust them to specific types of operation.

In the third stage, which begins after the simulation started, a sequential simulation process is performed for each model included in the overall IoT system. The results that are obtained for each model are passed to the associated models. Afterwards, the processes are simulated for each of these models.

At a cyclic performance the simulation of work of all system of the Internet of things in is carried out in general.

Now, let us take a closer look at the modeling and simulation processes.

16.1.1 Classification and terminology. Real-world objects and kinds of its simulation

More than two centuries the scientists are aware with the fact that the differential equations, same in a form, describe the phenomena, various by the nature. Such similarity of mathematical equations for various phenomena is called isomorphism [6]. It allows us to use a certain mathematical model to build a model of almost any similar object, phenomenon or process. Studying of model allows better understanding of nature of the phenomenon, which is modelled. Therefore, modeling can be considered one of the main tools of science.

Phenomena described by isomorphic equations can be similar. It means that between them the one-to-one correlation can be established, what makes it possible to extend the conclusions obtained in the study of one phenomenon to another.

Sir Isaac Newton gave the first scientific justifications of conditions of similarity as well as specification of this concept in relation to mechanical motion at the end of the 17th century. Now the similarity law strictly proved by mathematical apparatus of the similarity theory is based on three theorems.

The first theorem of similarity (it is also called the "direct" theorem) for the first time is intuitively formulated by Isaac Newton in 1686, and is proved nearly two hundred years later, in 1848 by the member of the French academy of Sciences Bertrán. According to this theorem, the similarity of systems can always be found such

dimensionless complexes of quantities, which for such points of these systems are the same. Thus, such systems, phenomena or processes are characterized by numerically equal values, which are called similarity criteria. It means that if the phenomena are similar, the similarity criteria can be found for them. To the contrary, for two systems, the phenomena or processes in case of obtaining criteria of similarity, identical in magnitude, this fact may be the basis for considering these systems, phenomena or processes. However, the first theorem does not specify how to establish similarity and how to implement it. It only forms the necessary conditions for the existence of similarity (the same similarity criteria).

The second theorem of similarity called p - the theorem, claims: "The full equation of the physical process, written in a linear system of units, can be represented by the dependence between the similarity criteria, i.e., the dependence connecting dimensionless quantities, obtained in a certain way from the existing parameters in the process." It follows from the second theorem that if the functional dependence of a phenomenon is known, that is, the parameters (factors) are known, but its mathematical description is unknown, then the similarity criteria can be obtained. The second theorem, as well as the first one, does not indicate ways to identify similarity and ways to implement similarity.

The third similarity theorem determines the necessary and sufficient conditions for the similarity of physical phenomena. The third similarity theorem states: "the necessary and sufficient similarity conditions are the proportionality of such parameters included in the conditions of uniqueness, and the equality of the similarity criteria of the phenomenon under study." Unambiguity conditions are the conditions defining specific features of the studied phenomenon, for example, terminal or boundary conditions. These conditions do not depend on the mechanism of studied phenomenon.

Based on these three theorems, there was a special science about model and modeling which is called "the theory of modeling and similarity". This theory distinguishes between three types of models: heuristic, natural (physical) and mathematical.

Generally, heuristic models represent the images drawn in imagination of the person. Their description is conducted by words of natural language, and therefore are ambiguous and subjective. These models are not formalized, it means, they are not described by formal-

logical and mathematical expressions, although they are based on real processes and phenomena.

16.1.2 Physical and computer simulations

Natural (physical) models exist physically, so that they are quite material. They differ from their prototypes in size, materials and number of constituent elements. Numerous models of planes and ships, which sets for production can be bought in specialized shops or in toy stores, are known to all. These models are typical examples of physical models. As a rule, these models perform the same function as the prototypes – models of aircraft can usually fly, and models of ships can swim. It is possible with them to carry out physical experiments and to define as they behave in various conditions. These experiments are usually cheaper, easier and safer than similar experiments with prototypes. However, based on the similarity theorem, we can understand how prototypes will behave under certain conditions. Mathematical models are typically mathematical expressions (formulas or equations) that define the relationship between the basic parameters of a process or phenomenon. Mathematical models are not material. Experiments with them (simulation) do not demand any equipment, except the computing device. It is not necessarily that such device should be a powerful modern computer. If you know how to count orally, then you can use yourself as a device. The drawback of mathematical models is that they usually do not take into account all the parameters of the phenomenon or process and not all the connections between these parameters. It means while applying mathematical models, there is always some error that occurs because of the inevitable errors of calculations as well as it can be incorporated in the mathematical model itself.

However, today mathematical models are the most used in the simulation of devices of the Internet of things. Advantages of computer modeling are the reason:

1. No physical objects need to be created to perform the simulation. Computer modeling does not demand expenses of materials and costs of production of physical model, that is, this type of modeling is the cheapest.

2. Simulation by mathematical model allows to study the behavior of the object in such conditions that are difficult or impossible to

implement in the experiment, such as ultrahigh temperatures or pressures.

3. Mathematical models of objects can be combined among themselves. Thus, it is possible to simulate a system of objects at once, even those that can not be combined in real life.

By the way, IoT devices almost never function separately from each other. They are always integrated into the system. It is the main feature of their operation. The Internet in term "Internet of Things" serves only as a binding element of a system. Things are basic here, not what unites them.

Finally, one more thought. Mathematical models, and simulation, which is carried out using them, are the main source of information for today about the behavior of the things that we use. Mathematical models are built based on laws by which a particular object or phenomenon of nature function. There is a question that worries the most powerful minds of humanity such as the physicist Einstein and the mathematician David Gilbert. The issue concerns why generally Nature (with a capital letter) has to submit the mathematical equation. Mathematics was born at the beginning of time as a mean for calculations of what the person dealt with. These calculations were reduced to integers and the operations of addition or subtraction. How many Buffalo do I have now? Three ones. One more was born. How many? Four. One fell into the gorge. How many lefts? Three again. There are basics of modern mathematics. Anything else - fractional numbers, multiplication and division - is only a consequence of our imagination. Roots, logarithms, derivatives and integrals – where do they exist in Nature? The same capital letters? Nowhere. If someone remembers about well-known Pythagorean`s theorem in which the root appears – and it is also not in nature. In Nature, there is a relationship between the legs and the hypotenuse. Why do mathematical equations describe the behavior of Nature?

There is still no response to this question. Even so, every day we use mathematics to predict something in our lives. Even so, the answer can be very simple. May be laws, which, as we suppose, the Nature follows, are also imagination of our mind, as well as mathematics, which describes these laws?

However, whatever the case with laws, we are able based on these laws to do mathematical models and to predict behavior of systems. We do this through software. Let us consider these means.

16.1.3 Virtual simulation. Common user interaction systems for virtual simulations

Currently, in the Internet you can find a lot of specialized software for simulation of objects, devices and systems of the Internet of things. Use of IOT simulators is the first step in creating and testing smart home and smart city devices. In this preliminary development step, a computer model of the IoT system is created. This computer model is a simulation, which aims to make sure the system efficiency and identify possible problems in its operation. At this step takes place approbation of future system and clarification of its working capacity.

Use of simulators is much cheaper than installing the system on site and testing it in the real world. Let us consider some of the most used computer simulation systems.

Simulator IoTIFY. It is a virtual online laboratory, that allows a user, who has access to the Internet to create a workspace where he can quickly assemble his own virtual hardware, download the desired operating system or build a firmware of his own choice. This tool allows to simulate large-scale IoT installations in virtual IoT lab. User traffic can be created from thousands of virtual endpoints and platform can be tested for scale, security, and reliability to identify and resolve issues before deploying the end product in a real environment. Heavy network traffic can be simulated to see how network latency affects overall system performance. Thus, this system allows [7]:

Select an existing hardware and touch combination or create own.

- Choose an operating system that meets certain requirements or create firmware for the microcontroller.

- Develop software on virtual hardware with the language for choice.

- Share the project with colleagues and collaborate with other IoT projects.

IoTIFY only allows network access to the resources of the simulation. Only project work files are stored on the user's computer.

NetSim. NetSim is a powerful network simulator that can be used to model IoT systems. [8] It can be used to test the performance of real-world applications over a virtual network. If you are creating a new IoT network or expanding an existing one, NetSim can be used to predict how the relevant network will work.

NetSim can be used in three versions. The first version, NetSim Pro, as its name suggests, is the most powerful. It can be used by corporate users and allows:

- Create network scripts using the NetSim GUI or using XML configuration files.
- To add devices, links, software applications, etc. in environment using a graphical interface NetSim.
- Model large and complex networks using an XML configuration file that comes with automatic validation
- Animate the flow of packets through wired and wireless links.
- To see performance metrics at different levels-networks, subnets, links, queues, applications, and so on.
- Explore a variety of metrics such as bandwidth, latency, loss, packet error, link usage and so on.
- To interpret the indices using the built-in scopes and schedules.
- Export packages and files to software packages such as Excel, Notepad for processing and statistical analysis.

The external interface allows NetSim to transfer instantly simulation results to other programs such as MATLAB. This program can start its computing process and send the information to NetSim. Then, NetSim can use this information to carry out their procedures for modeling networks.

This simulator supports multiple sources of information and can be scaled to hundreds of nodes. You can simulate a wide range of situations using “What-if” scenarios and test metrics such as loss, latency, errors, quality of service, and more.

MATLAB. MATLAB has a powerful IoT module that allows you to develop and test smart devices as well as collect and analyze IoT data in the cloud. It is possible to use MATLAB to create a prototype of IoT systems. In particular, it is possible to develop algorithms in Simulink and then deploy them on embedded hardware [9]. MATLAB can simulate IoT systems built on Arduino and Raspberry Pi processors.

MATLAB allows:

- Access streaming data and pre-archived data through built-in interfaces for cloud storage, relational and non-relational databases, and protocols such as REST, MQTT, and OPC UA.
- Create your own IOT algorithms using thousands of proven, out-of-the-box functions for processes such as data cleaning, machine and deep learning, computer vision, management and optimization.
- Develop data-driven physical models to understand, control, and optimize the behavior of IoT systems.
- To use the platform ThingSpeak, IoT, Analytics MATLAB for prototyping and operation of small systems.

BevyWise IoT Simulator. This IOT Simulator is an easy to use, but powerful simulation tool that allows the simulation of thousands IoT devices. Intuitively clear interface allows you to create and add the necessary devices in the shortest possible time. You can customize the simulated IoT devices. IoT Simulator can store simulation data in FLAT or MySQL and SQLite files. The tool supports thousands of IoT devices in Windows 7 and later versions.

IoT Simulator allows to create templates and dynamic networks for devices. The simulation is performed using the same code signals as real IoT devices. Simulation of IoT systems behavior is performed using templates, which allows you to create thousands of IoT devices in a few minutes.

IoT Simulator supports storage of simulated network in MySQL database. It is possible to store multiple modeling environments and reuse them.

ANSYS IoT Simulator. This simulator is a powerful IoT engineering simulation software system. To create it, the experience of the world's leading companies that create IoT systems in various industries is used.

ANSYS IoT Simulator allows developing reliable electronic devices for industrial IOT. The solutions cover the development and optimal placement of sensors, antennas, electronic control and their power system required to connect intelligent machines and their ecosystems. In addition, the virtual system ANSYS helps to record and certify the embedded code that manages the smart devices and provides a reliable signal without interference in the working industrial installation.

Interaction with other software products of ANSYS family allows performing modeling and technical development at all stages of IoT systems creation, from primary design to development solutions.

IBM Watson IoT Platform. IBM's Watson IoT Platform is an innovative cloud platform that allows IoT developers to create their own platform, even if they do not have physical devices. Built-in tools allow tracking and analyzing IoT data and then use it to create and optimize own programs. The tool supports a wide range of functions for manipulating data, storing it and even interacting with social media.

IBM Watson to IBM IoT Cloud Platform provides a comprehensive set of tools, which includes gateway devices, control devices and access programs. With the Watson IoT platform, you can collect data from connected devices and perform real-time data Analytics.

Proteus IoT Builder. IoT Builder is the world's first software product that provides a complete workflow for designing IoT devices on Arduino or Raspberry Pi hardware. It can be added either Visual Designer for Arduino, or Visual Designer for Raspberry Pi product to allow development of remote interfaces of embedded devices.

Simulation in Proteus IoT Builder system has many features and at the same time it is intuitive. It starts with designing the hardware of the IoT device on the layout. There is the possibility of adding electronic screens, sensors, dials, buttons and many other devices using the gallery controls.

Due to its simplicity and the presence of the student version of the program, Proteus IoT Builder today is the most common simulation system. At the same time, the Arduino platform is the most common platform for building real IoT systems. Therefore, the further process of IoT systems simulation will be considered on the example of this software.

16.2 Simulation IoT devices based on Arduino platform

In this section, we will get acquainted with the basic concept and principle of building the Arduino platform. The year of Arduino birth can be considered as 2005, when a team of young designers from the Italian town of Ivrea created the first prototype of Arduino Board. Starting the work, the development team immediately set a goal to make a microprocessor device, the price of which would be suitable for a student's pocket - \$ 30. They also wanted to make it with the capacity

to build, like a normal children's designer. At the same time, Arduino is even more than a regular constructor. Arduino is a platform based on which a variety of complex things can be created. Today, there are many interesting Arduino-based developments such as breathalyzers, led cubes, home automation systems, Twitter notification displays, and even DNA analysis kits! Already there were whole clubs and communities of Arduino fans. Google has recently released an Arduino-based development kit for its Android smartphone [13].

Let us take a closer look at what it is – Arduino.

16.2.1 General information about the Arduino platform

Arduino is a tool for designing electronic devices (electronic designer) more tightly interacting with the physical environment than standard personal computers, which actually work only with the data stored in it [14]. It is a platform designed to build open source cyber-physical systems, based on a simple printed circuit Board with a modern environment for writing software.

Arduino is used to create electronic devices with the ability to receive signals from various digital and analog sensors that can be connected to it, and control various actuators (Fig. 16.1).

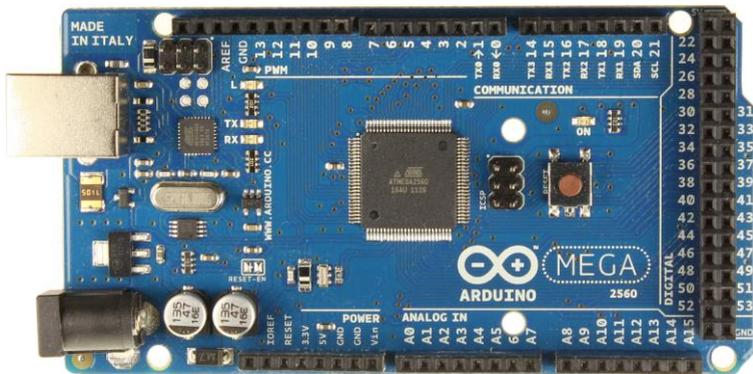


Fig. 16-1 Arduino Board

The device projects, based on Arduino, can work independently or interact with the software on computer (e.g. Flash, Processing, MaxMSP). Boards can be assembled by the user himself or purchased

in a set. The open source software development environment is available for free download.

There are many microcontrollers and platforms for creating cyber-physical systems, such as Parallax Basic Stamp [15], Netmedia's BX-24 [16], Phidgets [17], MIT's Handyboard [18], and many others that offer similar functionality. All of these devices even look similar to the Arduino. They integrate separated programming information into an easy-to-use Assembly. Arduino also simplifies the work process with microcontrollers, but has a number of advantages compared to other devices for teachers, students and amateurs:

- Low-cost of Arduino boards are relatively cheap compared to other platforms. The most inexpensive version of the Arduino module can be assembled by hand, and some even ready-made modules cost less than \$ 50.

- Cross-platform - the Arduino software runs on Windows, Macintosh OSX, and Linux. Most microcontrollers are limited to Windows.

- Simple and clear programming environment - Arduino environment is suitable for both novice and experienced users. Arduino is based on the programming environment Processing, which is very convenient for teachers, as students who work with this environment will be familiar with Arduino.

Extensible and open source Arduino software is released as a tool that can be supplemented by other users. It can be supplemented with C++ libraries. Users, who want to understand the technical nuances, have the opportunity to switch to the language AVR C which is based on C++. Accordingly, it is possible to add code from the AVR-C environment to the Arduino program.

16.2.2 Arduino and Arduino-compatible boards

Arduino is a set of electronic unit and software storage. An electronic unit here is a printed circuit board with a set microcontroller and a minimum of elements, which are necessary for its functioning. The software is needed to create control programs. It consists of a simple development environment the Arduino IDE and programming language, namely the C/C++ version for microcontrollers, supplemented by certain functions for controlling inputs/outputs on the contacts of board.

In fact, the electronic unit Arduino is analogous to the motherboard of the modern computer. Arduino Mega 2560 (Fig. 1) – this is a device, which based at microcontroller ATmega2560 (datasheet). The board itself consists of:

- 54 digital inputs/outputs (15 can be used as PWM-outputs);
- 16 analog inputs;
- 4 UART (hardware receivers for the implementation of serial in-terfaces);
- quartz resonator at 16 MHz;
- USB connector;
- power connector;
- ICSP connector for internal circuit programming;
- reset button.

16.2.3 Technical characteristics Arduino Mega

The main characteristics of the Arduino Mega 2560 are shown in the table 1.1.

Table 1.1 - Characteristics of the Arduino Mega 2560

Operating voltage	5 V
Power supply (Recommended)	7-12 V
Power supply (Maximum)	6-20 V
Digital inputs / outputs	54
Analogue inputs	16
Maximum current of 5 V output	40 mA
Maximum current of 3.3 V output	50 mA
Clock frequency	16 MHz

The board Arduino Mega 2560 can powered by two methods, namely:

- from a computer by using a USB cable;
- from an AC / DC adapter or from battery.

When using an external power supply, it is necessary to select its value in the range of 6 to 20 V. But when using a power source with a voltage below 7 V, the voltage at the output 5 V decreases, which leads

to unstable function of the board. Using a power supply with a voltage higher than 12V leads to overheating of the voltage regulator and to breaking down of board. Considering this, it is recommended to use a power source with a voltage value in the range of 7 to 12 V.

16.2.4 Inputs and outputs of Arduino Mega 2560.

The main inputs and outputs that are located on the Arduino Mega 2560 (Fig. 16-2) are designed to connect of contacts board with sensors or actuators. Pins on the board Arduino Mega 2560:

- **VIN** – the input voltage to the board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V** – this pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V).
- **3.3V** – this pin comes in 3.3 V from the voltage regulator on the board. Maximum current is 50 mA;
- **GND** – 2 ground pins;

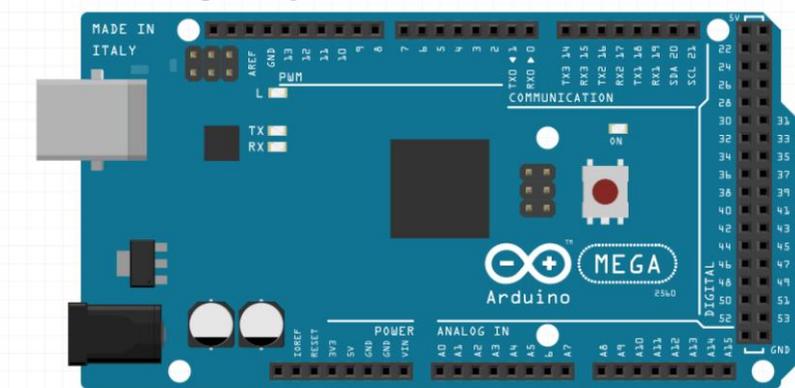


Fig. 16.2 – The pins location of the board Arduino Mega 2560

- **IOREF** – this pin on the board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source

or enable voltage translators on the outputs for working with the 5V or 3.3V.

The Arduino Mega 2560 is located 54 digital pins. Each digital contact can be used as an input and output, using functions *pinMode()*, *digitalWrite()* and *digitalRead()*. They work at 5 V voltage. Each pin can provide or receive 20 mA in the recommended operating mode and has an internal load resistor (default disabled) with a nominal impedance of 20-50 k Ω . The maximum permissible current value is 40 mA - this value should not be exceeded to avoid damage to the microcontroller. In addition, some foams have specialized functions:

Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX). Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip.

External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2). These pins can be configured to trigger an interrupt on a low level, a rising or falling edge, or a change in level. See the *attachInterrupt()* function for details.

PWM: від 2 до 13 і від 44 до 46 Provides 8-bit PWM output using the *analogWrite()* function PWM is a pulse-width modulation, an operation for obtaining a variable analog value using digital devices. By outputting a signal that consists of high and low levels, the voltage is modeled between the maximum (5 V) and the minimum (0 V) values. The duration of switching on the maximum value is called the pulse width. It changes to get different analog values.

SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS). These foams support the SPI connection using the SPI library. SPI is a serial peripheral interface. SPI is a synchronous interface in which each transmission is synchronized with a clock signal generated by the master device (microcontroller).

LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

TWI: 20 (SDA) and 21 (SCL). Support TWI communication using the *wire* library

The Mega 2560 has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure

from ground to 5 V, though is it possible to change the upper end of their range using the AREF pin and `analogReference()` function.

There are a couple of other pins on the board:

AREF: Reference voltage for the analog inputs. Used with `analogReference()`.

Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

16.3 Software development. Arduino C/C++ sketch

The development of programs to perform certain tasks with the help of the Arduino boards is carried out in the official programming environment of the Arduino IDE. This environment is intended for writing, compiling and downloading created programs in the memory of the microcontroller.

A program written in the Arduino IDE programming environment is called sketch. It is written in a program code editor. When saving and exporting a project in the notification area appear explanations and error information. The text output window shows Arduino messages, which show full error reports and other important information. The toolbar buttons allow you to check and write the program, create, open and save the sketch; open the monitor for the serial bus.

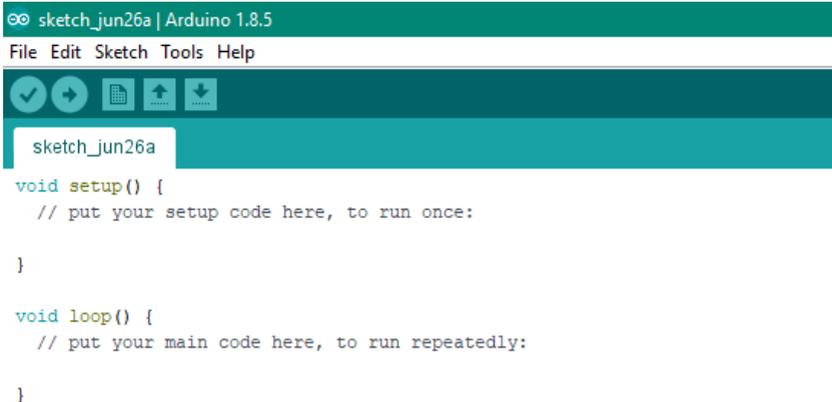
Developed sketches can receive additional functions through libraries, which are a specially designed code. It helps to realize some of the opportunities that can be added to the project. There are many specialized libraries. Usually, libraries are written to simplify the decision of a task.

16.3.1 General methodology of Arduino sketch working

Sketches for Arduino are written in a language very similar to the C programming language. However, if in the standard C language, the function `main()` is necessary, then it is missing in the sketches. But here two functions are necessary – `setup()` and `loop()`.

When enabled, the Arduino firmware calls the `setup()` function. The `setup()` function is called only once, each time the board is started. This place is ideal for initialization (setting initial values) of variables, setting pin modes (input or output), setting correspondence between

connected sensors, servo drives or other pins. After the *setup()* function is executed, the *loop()* function is looped (i.e., immediately after exiting the setup function, the loop function is executed, after exiting it, it is called again. This process continues until the power is turned off (Fig. 16.3).



```

sketch_jun26a | Arduino 1.8.5
File Edit Sketch Tools Help
sketch_jun26a
void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}

```

Fig. 16.3 – Basic elements of the Arduino IDE programming environment

After the functions *setup()* and *loop()* in the sketch can be placed other functions that are written by the user. These functions perform the actions provided by the algorithm of work and at least one of them must be called from the function *loop()*. The syntax of writing custom functions completely coincides with the syntax of the C language.

A standard example of the simplest sketch is a sketch that makes the LED blink. Consider all the steps for writing and running such a sketch.

Step 1. Connect with the USB cable the appropriate outputs of the multi-function unit and computer to work with the board Arduino Mega 2560, which is located inside the multifunction unit.

Step 2. Check for the Arduino IDE environment development required on your computer to work with the board Arduino Mega 2560. In the absence of this program - download it from the official site [19].

Step 3. Run on the computer the program of development environment Arduino IDE. Select the board Arduino Mega 2560 by: «Tools/Board/Arduino Mega 2560» (Fig 16.4).

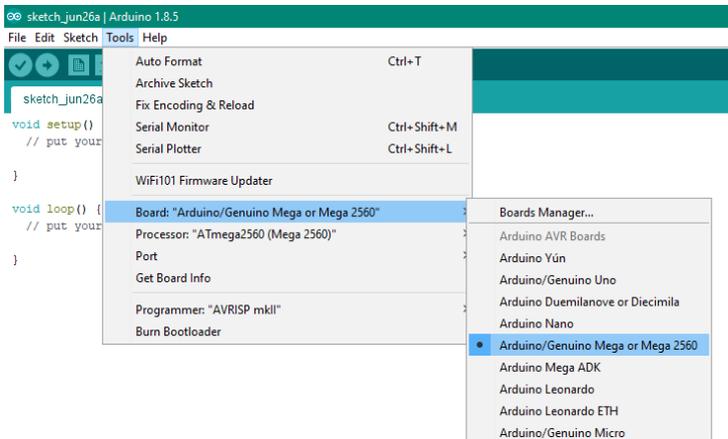


Fig. 16.4 – Choice of board Arduino Mega 2560

Step 4 is to select the appropriate port by «Tools/Port/COM1 (Arduino Mega 2560)» (Fig. 16.5).

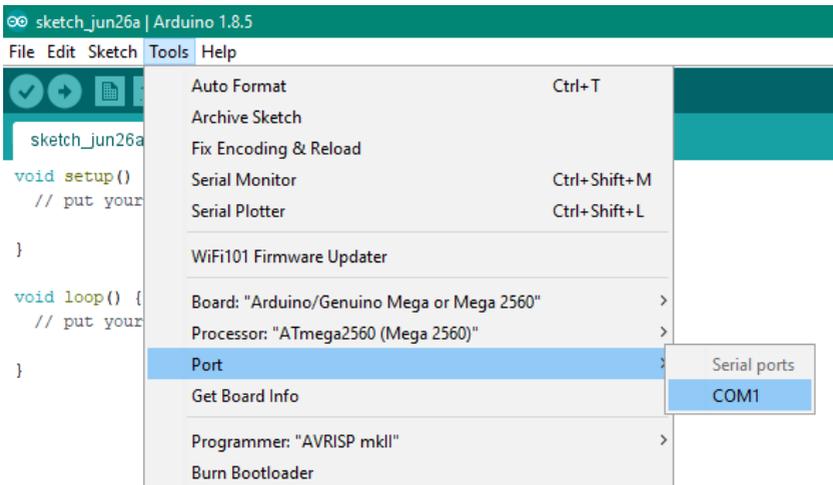


Fig 16.5 – Choice of port

Consider working with a sketch on a simple example. This example has become traditional for illustration work with sketches.

16.3.2 Arduino sketch example

In this example, we will make the LED blink at intervals 1 second.

The LED connection is made as follows:

1) Cathode connects to any GND in foam panel of the multifunction unit;

2) Anode connected to pin 22 in foam panel of the multifunction unit (this pin is digital).

Next you need to open the program Arduino IDE, create a new sketch (in the toolbar, click “File / New” (Fig. 16.6)

and load the next sketch there:

```
int led = D22;
void setup()
{
pinMode(led, OUTPUT);
}
void loop()
{
digitalWrite(led, HIGH);
delay(1000);
digitalWrite(led, LOW);
delay(1000);
}
```

Next, need to check the sketch text for errors: in the toolbar, click “Sketch / Verify” (Fig. 16.7). After that, at the status window will display information about the structure of the sketch, as well as the presence or absence of pillocks.

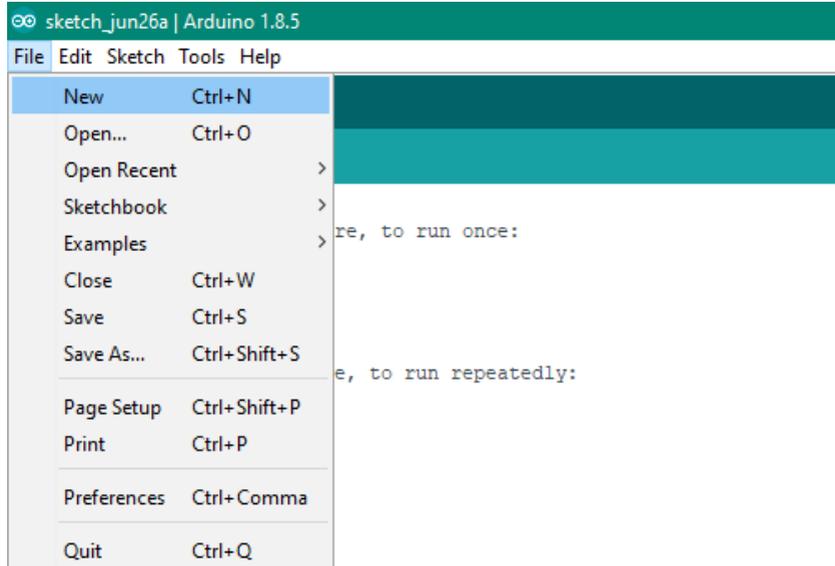


Fig. 16.6 – Create a new sketch

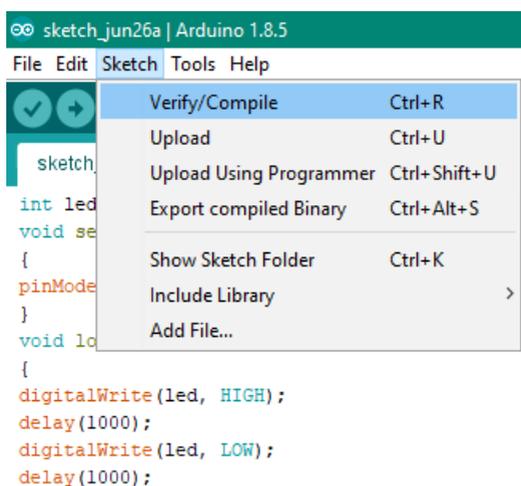


Fig 16.7 – Function of verify a sketch

If the program does not detect any inequalities in the code, man can download sketch to the microprocessor. To do this, click : “Sketch / Upload” (Fig. 16.8). After this, the LED should be measured once a second.

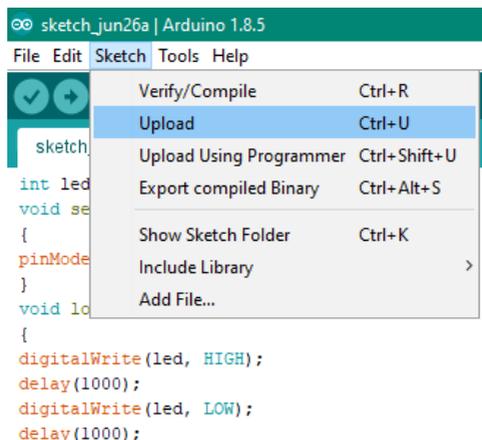


Fig. 16.8 – Function of upload a sketch

16.4 Work related analysis

The simulation of the Internet of things is a relatively young area of research. Therefore, there are few publications on this topic. However, European universities conduct research and modeling of the Internet of things and individual devices. In addition, a separate direction of modeling and simulation is the modeling of the communication component. So, in the article [20] S. Forbacha and C. Pattinson simulated an energy-aware mobile agent (MA) NMS in OPNET which could operate efficiently in a power saving environment. He created an infrastructur wireless network with scenarios which represented non-power saving and power saving and then compared the impact of power saving on both centralized and optimized paradigms.

This topic continues the publication [21] in which the authors model self-configuring wireless network in which each node could act as a router, as well as a data source or sink. Its application areas include

battlefields and vehicular and disaster areas. It is shown that any techniques applied to infrastructure-based networks are less effective than this self-configuring wireless network.

A series of publications of researchers from Royal Institute of Technology (KTH, Stockholm, Sweden). In paper [22] authors model an emerging network architecture based on edge computers. This network can be protected even under situations such as network failures or denial-of-service (DoS) attacks. This model allows IoT devices to migrate to other local authorization entities served in trusted edge computers when their authorization entity becomes unavailable.

Conclusions and questions

This chapter describes the methods and tools for modeling and simulating Internet of things systems and individual devices. The main attention is paid to the modeling of smart home systems, which are built on the basis of microprocessor boards ARDUINO. A review of the main software tools used to simulate the Internet of Things systems has been completed. The PROTEUS software system is considered in detail, which allows simulating the operation of ARDUINO devices in a convenient form.

To control the assimilation of the material we recommend that you answer the following questions.

1. What is modeling?
2. How does computer modelling differ from computer simulation?
3. What types of models do you know?
4. What is a physical model?
5. What is a verbal model?
6. What is a mathematical model?
7. Can Einstein's famous formula be considered a mathematical model?
8. Why do I need to simulate the work of the Internet of things?
9. What software for simulating the Internet of things do you know?
10. What are the possibilities of the simulation program IoTIFY?
11. What are the possibilities of the simulation program NetSim?

12. What are the possibilities of the simulation program MATLAB?
13. What are the possibilities of the simulation program BevyWise IoT Simulator?
14. What are the possibilities of the simulation program ANSYS IoT Simulator?
15. What are the possibilities of the simulation program IBM Watson IoT Platform?
16. What are the possibilities of the simulation program PROTEUS?
17. List the technical characteristics of the Arduino Mega.
18. List the components of the ARDUINO Input-Output System.
19. What is ARDUINO sketch?
20. Why in the sketch function Setup() is needed?
21. Why in the sketch function Loop() is needed?
22. What is the methodology of the sketch?
23. In what language sketches are written for ARDUINO?
24. List the main research directions in the field of simulation of the Internet of things.
25. Download the PROTEUS program and repeat the example given in this section.

References

1. Bernard P. Zeigler, Tag Gon Kim, and Herbert Praehofer. 2000. Theory of Modeling and Simulation (2nd ed.). Academic Press, Inc., Orlando, FL, USA.
2. John H. Holland, "Theory and Models: General Principles," in Signals and Boundaries: Building Blocks for Complex Adaptive Systems , 1, MIT Press, 2012, pp.35-56
3. Sumit Ghosh; Tony Lee, "Principles of Modeling Complex Processes," in Modeling and Asynchronous Distributed Simulation: Analyzing Complex Systems, 1, Wiley-IEEE Press, 2000, pp.332. doi: 10.1109/9780470545300.ch3
4. Rothmaler, P., 2000, Introduction to Model Theory, Amsterdam: Gordon and Breach.

5. "Modeling and simulation", En.wikipedia.org, 2019. Available: https://en.wikipedia.org/wiki/Modeling_and_simulation. [Accessed: 01- Jan-2019].
6. M. Hazewinkel, Encyclopaedia of mathematics. Dordrecht: Kulver academic, 1995.
7. "IoTIFY- Develop full stack IoT Application with virtual device simulation", Iotify.io, 2019. Available: <https://iotify.io/>. [Accessed: 06- Jan-2019].
8. "NetSim-Network Simulator & Emulator | Emulator", Tetcos.com, 2019. Available: <https://www.tetcos.com/emulator.html>. [Accessed: 06- Jan-2019].
9. "Internet of Things", Mathworks.com, 2019. Available: <https://www.mathworks.com/solutions/internet-of-things.html>. [Accessed: 06- Jan- 2019].
10. "IoT Simulator, Simulate IoT Devices - Bevywise Networks", Bevywise Networks | IoT Platform & IoT Solutions, 2019. Available: <https://www.bevywise.com/iot-simulator/>. [Accessed: 06- Jan- 2019].
11. "IoT - Industrial Equipment and Asset Management | ANSYS", Ansys.com, 2019. Available: <https://www.ansys.com/Campaigns/internet-of-things/industrial-equipment-and-asset-management>. [Accessed: 06- Jan-2019].
12. "IBM Cloud Docs", Console.bluemix.net, 2019. Available: <https://console.bluemix.net/docs/services/IoT/index.html#gettingstartedtemplate>. [Accessed: 06- Jan- 2019].
13. Arduino Project Hub. (2019). Google Assistant Robot Using Arduino. Available at: <https://create.arduino.cc/projecthub/jithinsanal1610/google-assistant-robot-using-arduino-0c70d6> [Accessed 9 Jan. 2019].
14. Arduino.cc. (2019). *Arduino - Home*. Available at: <https://www.arduino.cc/> [Accessed 9 Jan. 2019].
15. Parallax.com. (2019). Getting Started | Parallax Inc. [online] Available at: <https://www.parallax.com/getting-started> [Accessed 9 Jan. 2019].
16. Basicx.com. (2019). *BasicX by NetMedia Inc.*. Available at: <http://www.basicx.com/> [Accessed 9 Jan. 2019].
17. Phidgets.com. (2019). Phidgets Inc. - Products for USB Sensing and Control. [online] Available at: <https://www.phidgets.com/?> [Accessed 9 Jan. 2019].
18. Handyboard.com. (2019). THE HANDY BOARD AND THE SUPER CRICKET. [online] Available at: <http://www.handyboard.com/> [Accessed 9 Jan. 2019].
19. Arduino.cc. (2019). Arduino - Software. Available at: <https://www.arduino.cc/en/Main/Software> [Accessed 13 Jan. 2019].

20. Forbacha, S. and Pattinson, C. (2011). Simulation of Energy-Aware Mobile Agent Based Network Management System. 2011 Fifth Asia Modelling Symposium.

21. Pullin, A., Pattinson, C. and Kor, A. (2018). Building Realistic Mobility Models for Mobile Ad Hoc Networks. *Informatics*, 5(2), p.22.

22. Kim, H., Kang, E., Broman, D. and Lee, E. (2017). An Architectural Mechanism for Resilient IoT Services. Proceedings of the 1st ACM Workshop on the Internet of Safe Things - SafeThings'17.

17. THREE-LEVEL SIMULATION OF IOT/IOE BASED SYSTEMS WITH THE USE OF UML DIAGRAMS, PETRI NETS AND TEMPORAL LOGIC

Dr. Ass. Prof. . O. Martynyuk, DrS. Prof. O. V. Drozd (ONPU)

Contents

Abbreviations 42
Introduction 43
17.1. Simulation and verification in architecture of IoT and IoE-based systems with the use of visual UML diagrams 43
17.1.1 Introduction to representation of architecture of IoT and IoE-based systems with the use of visual UML diagrams (precedents, components, classes, interaction, activities, sequences, states) 44
17.1.2 Static visual UML diagrams for the description of architecture of IoT and IoE-based systems and their analysis 47
17.1.3 Dynamic visual UML diagrams for the description of architecture of IoT and IoE-based systems and their analysis 50
17.2 Simulation and verification in behavior of IoT and IoE systems on the basis of the Queuing Systems and Petri Nets 53
17.2.1 Introduction to the general description of the operation of IoT and IoE systems at the level of resource and functional mode presentation 54
17.2.2 Resource imitating modeling and simulation on base of functioning of IoT and IoE systems and their components using QS . 59
17.2.3 Behavior imitating modeling of features for functioning of IoT and IoE systems and their components using Petri Nets 61
17.3 Simulation and verification of synchronization processes in IoT and IoE-based systems on the basis of temporal logic 64
17.3.1 Introduction to specification of synchronization process in IoT and IoE-based systems by using of temporal logic 64
17.3.2 Simulation and verification of IoT and IoE-based systems at the level of temporal logic 66
17.3.3 Special temporal simulation and verification of IoT and IoE-based systems 66
17.4 Work related analysis 67
Conclusions and questions 69
References 72

Abbreviations

CPN – Colored Petri Nets

CTL – Concurrent Temporal Logic

DFD – Data-flow Diagram

EGS – Evolutionary-Genetic System

ERD – Entity-Relationship Diagram

GPSS – General Purpose Simulation System

IDE - Integrated development environment

IoE – Internet of Everything

IoT – Internet of Things

LTL – Linear Temporal Logic

MAS – Multi-Agent Systems

QS – Queuing System

TINA – TIme Petri Net Analyzer

UML – Universal Modeling Language

UML-CSAS – UML-diagrams of Communications, Sequences, Activity, States

Introduction

The purpose of section is formation of complete idea of mathematical base, formal the specification and modeling of Internet of Things (IoT) and Internet of Everything (IoE) [1 – 4] of systems on structural, functional and event – time levels with the use of the corresponding Universal Modeling Language (UML) diagrams [5 – 7], Queuing Systems (QS) [8, 9], automata [10, 11] and Petri nets [12 – 14], temporal logic [15, 16] and also development of skills in the use of the gained knowledge in practice of the general system for the specification and modeling of IoT and IoE.

Objective is to understand formal structural, component, object, behavioral and temporary models of the architecture for systems of IoT and IoE in the form of UML diagrams, QS, Petri nets, expressions and conclusions of temporal logic.

Subjects are formal processes of modeling and the verification of the structural, component, object, behavioral and temporary models' architectures for systems of IoT and IoE.

Tasks solved:

1) Visual modeling, simulation and verification of architectures for systems of IoT and IoE on the basis of visual UML diagrams.

2) Resource modeling, simulation and verification of architectures for systems of IoT and IoE on the basis of QS.

3) Behavioral modeling, simulation, analysis of correctness, verification, on-line testing and testing check of architectures and processes for IoT and IoE systems on the basis of QS, expanded finite automata and Petri nets.

4) Modeling and verification in synchronization of processes for systems of IoT and IoE on the basis of temporal logic.

17.1. Simulation and verification in architecture of IoT and IoE-based systems with the use of visual UML diagrams

The following objects are taken as input:

1) Specifications of the technical description for the architecture of components, subsystems IoT and IoE-based systems, as well as such systems, in future – as a whole in analytic-text, tabular, graphical

representations, defining the structure of topological relationships, functions, information objects, interfaces of topological interactions (in future – format, dimension, type, conditions of transmission over the topological connections of the objects to be sent – parameters, data, methods and their compositions), the temporal behavior of functions and scenarios (in future – time diagrams, graph, automaton, algorithmic representations).

2) The previously prepared static and dynamic UML diagrams, which in UML standards define the architecture of components, IoT and IoE-based systems subsystems, and also such systems as a whole in analytic-text, tabular, graphical representations, for which system-wide interactive-visual analysis, simulation and verification, in particular, in the Star UML tool environment is necessary.

The following objects are considered as output objects: the resulting correct and verified UML static and dynamic diagrams, which in UML standards represent the architecture of components, IoT and IoE-based systems, as well as such systems in general, for which in the corresponding instrumental environment, particular, Star UML, system-wide interactive-visual analysis, simulation and verification, special conditions obtained, parameters and scenarios of such analysis, simulations, verifications, special results of fulfillment of conditions, application of parameters and scenarios.

17.1.1 Introduction to representation of architecture of IoT and IoE-based systems with the use of visual UML diagrams (precedents, components, classes, interaction, activities, sequences, states)

In the construction of both specifications and models on the basis of UML (also on QS, automata, Petri nets, temporal logic) in the analysis and synthesis of the architecture of systems and processes in IoT and IoE, the following basic stages of early system engineering technology are usually defined:

Stage 1. Building the structure of system and determining the composition of the components.

Stage 2. Definition of syntax for system and component operations and functions.

Stage 3. Determining the structure and format of information objects and classes for system, components and functions.

Stage 4. Building system intercomponent interfaces (structure links) with their formats, dimensionality of values, binding conditions and events on both sides.

Stage 5. Determining the own logical time, conditions and events both for the system as a whole, and for components, functions, information objects and classes, interfaces.

Stage 6. Building the relationship of temporal interactions, synchronization, processing of conditions and events for functions, information objects and classes, interfaces.

Stage 7. Construction of timing diagrams, procedures and algorithms for executing system scripts, component functions, methods and handlers of classes and objects.

Universal Modeling Language UML [5 – 7] and appropriate tool environments, for example, Star UML, MS Visual Studio (UML) [5, 6, 17], are used for IoT architectural analysis and design, for system structural, functional, object, component, event-time characteristics of objects and relationships of IoT, their visual simulation and verification.

In UML, all objects can be divided into the following basic types: a) structural or static; b) behavioral or dynamic; c) grouping; d) annotational.

In accordance with stages 1 – 7 of early system design technology, the procedure for constructing and analyzing of the UML diagrams, depending on the objects of the systems and components of IoT is determined by the use of 7 types of the static UML diagrams at the stages 1 – 4 for describing of precedents, components, objects, classes, packages, layout, structure and by using of the 6 types of the dynamic UML diagrams at the stages 5 – 7 for describing of synchronization, activity, communication, sequences, automata and state machines, interaction.

To determine the relationships of objects and entities of all types of UML diagrams, also depending on the objects of the systems and components of IoT, four types of basic relations are used, such as:

1) Dependence indicates that change of independent essence somehow influences dependent essence. Graphically the relation of

dependence is represented in the form of a dashed line with the arrow directed from dependent essence to independent.

2) Generalization as the relation between two entities, one of which is a special (specialized) case another. Graphically generalization is represented in the form of the line with the triangular not painted over arrow on the end directed from private (subclass) to general (super class).

3) Association points that one essence is directly connected with another (or with others – the association can be not only binary). Graphically, the association is represented in the form of the continuous line with various additions connecting the connected entities.

4) Realization specifies that one essence is realization another. Graphically realization is represented in the form of a dashed line with the triangular not painted over arrow on the end directed from the realizing essence by realized.

The tool environments (frameworks) with their operation bases, such as Star UML, MS Visual.NET (UML) [5, 6, 17], are used at specification, analysis and verification of visual UML diagrams.

System modeling requires the description of several models, because it is not enough to describe the system from a single point of view. A model is a description of any aspect of systems, such as structure, behavior, requirements, etc. The model can be presented in a text-analytical, tabular or visual-graphic form.

The model element is a building block of a model. Diagram is a visual graphical symbolic representation of a model. A model can be represented in one or more diagrams with different aspects. For example, a diagram can focus on class hierarchical structure while another diagram can focus on interaction between objects.

Diagrams consists of graphical elements, which are visual representations of a model element.

Star UML possesses some features. So, Star UML is an open source project to develop fast, flexible, extensible, featureful, and freely-available UML/MDA platform. The goal of the Star UML project is to build a software modeling tool. Star UML is a sophisticated software modeler aimed to support *agile* and *concise* modeling. The key features of Star UML are: Multi-platform support (MacOS, Windows and Linux); UML 2.x standard compliant; Entity-Relationship diagram (ERD); Data-flow diagram (DFD); Flowchart

diagram; Multiple windows; Modern UX; Dark and light themes; Retina (High-DPI) display support; Model-driven development; Open APIs; Various third-party extensions; Asynchronous model validation; Export to HTML docs; Automatic updates.

MS Visual.NET (UML) possesses some own features too. MS Visual.NET create UML models at different levels of detail throughout the application lifecycle as part of development process. Track requirements, tasks, test cases, bugs, and other work associated with models by linking model elements to Team Foundation Server work items and development plan. The key features of MS Visual.NET for UML are: visualize code; describe and communicate user requirements; define the architecture; validate system with the requirements and intended design; share models, diagrams, and code maps using Team Foundation version control; customize models and diagrams; generate text using T4 templates.

17.1.2 Static visual UML diagrams for the description of architecture of IoT and IoE-based systems and their analysis

The analysis of static and dynamic visual UML diagrams allows to define and verify the description of architecture of IoT and IoE-based systems. Static visual UML diagrams [5 – 7] serve to describe the static part of architecture of the IoT and IoE-based systems in representation of structure, their components, component functions and information objects, intercomponent interfaces with their formats, conditions, events and means of processing, in particular, hardware, software, information objects of data collection, transfer and storage, sensory and executing nodes, brokers, servers, administrative stations.

The seven types of the static UML diagrams are applicable for describing of the precedents, components, objects, classes, packages, placements and structures.

Precedents are the subjects and objects of IoT and the IoE systems, their components and interactions. Components are the structural components in systems of IoT and IoE. Objects are the program objects, using the frames, variables, structures, functions, methods from systems of IoT and IoE, their components. Classes are the program classes, using the frames, variables, structures, functions, methods from systems of IoT and IoE, their components. Packages are the program

packages, including program objects, components and classes from systems of IoT and IoE, their components. Placements are objects of physical positioning / allocation for components, objects, classes, packages in IoT and IoE systems. Structures are the directional and nondirectional point-to-point, peer tire and star, treelike, hierarchical and cluster topological structures of relational and interaction of component in systems of IoT and IoE.

Static visual UML diagrams are used to describe the static part of architecture of complex subsystems, in particular, on base of IoT and IoE. The main applications of static UML diagrams for describing structures and systems of IoT and IoE include a number of approaches, cases, stages and steps described below.

General static, spatial structures, described by static UML diagrams, usually are formed in the natural sequence of the following procedure of construction:

- Step 1. Construction of diagrams of the precedents.
- Step 2. Construction of diagrams of the components.
- Step 3. Construction of diagrams of the classes.
- Step 4. Construction of diagrams of the packages.
- Step 5. Construction of diagrams of the objects.
- Step 6. Construction of diagrams of the placements.
- Step 7. Construction of diagrams of the composite structures.

Special spatial architecture of static network environment of IoT and IoE by static UML diagrams is formed in the sequence of the procedure of visual determination, construction, placement, simulation and verification of the above special static UML diagrams for/into the next network hosting and designing environment, further – the UML-design procedure*, namely for/into:

Step 1. Existing product/resource service network for modification of monitoring and control (UML-diagrams of components, classes, structure (UML-CCS*) for end-points, network, control environment – lighting, power supply, temperature/heating/air conditioning, ventilation, humidity, ionization).

Step 2. Existing computer network consisting of wired and wireless network environment (UML-CCS* for end-points, network, control environment – administrative and end-user stations, routers/access-points, servers).

Step 3. Developed external end-points of IoT and IoE subsystem, (UML-CCS* for developed end-point environment – end-point sensors/actuators, end-point controllers, controller buffers, controller intercomponent interfaces).

Step 4. Developed topology structure and internal node components of IoT and IoE subsystem (UML-CCS* for – developed topology, nodes – brokers/routers/access-points, broker/router/access-point buffers, broker/router/ /access-point intercomponent interfaces).

Step 5. Extended developed topology structure and internal servers of IoT and IoE subsystem (UML-CCS* for – extended developed topology, servers, server buffers, server intercomponent interfaces).

Step 6. Extended developed topology structure and external terminals of IoT and IoE subsystem (UML-CCS* for – extended developed topology, end-user/administrative terminals, terminal buffers, terminal intercomponent interfaces).

Step 7. Composition these partial developed static UML-CCS* into general developed system of static diagrams and their structure for existing product/resource service network, existing computer network and developed IoT and IoE subsystem.

Spatial structure for special resource properties of dynamic transport/service data flows by static UML diagrams is formed in the sequence of the following UML-design procedure* for/into graph structures (nodes, paths, trees, hammocks, cycles) of dynamic processes through the developed general spatial structure of components of subsystem of IoT and IoE (with all its components – sensors/actuators, end-point controllers, buffers, intercomponent interfaces, brokers/routers, servers, terminals), namely for:

Step 1. Computational, memory, communicative metric units, min/max tensions and capacity, capacity of buffers, its values (UML-CCS* with properties and methods and their placement for all trivial graph nodes of dynamic processes, as components of transport/service data flows).

Step 2. The same units and their values and properties but for all nontrivial graph structures of dynamic processes, as components of transport/service data flows).

Step 3. Selected technological standards, interface and communication protocols of computer networks and IoT and IoE for

the all above static UML diagrams of the developed resource, transport/service and computer subsystem of IoT and IoE.

Special spatial resource and transport/service evolutionary-genetic system (EGS), their EGS-models – genes, chromosomes, individuals, populations, signatures of operations and relationships [18, 21 – 23] is formed by static UML diagrams into life cycles of IoT and IoE in the sequence of the following technologies of construction – defining, analysis, modeling, simulation and verification of next stages (each of which includes own special evolutionary steps):

Stage 1. Developed slow spatial evolution by static UML diagrams into life cycles of static general spatial topology system structure of IoT and IoE.

Stage 2. Developed fast spatial evolution by static UML diagrams into life cycles of dynamic transport/service data flows for spatial system structure of IoT and IoE.

Stage 3. Developed coevolution as composition of slow spatial evolution for spatial system structure and fast spatial evolution for transport/service data flows by static UML diagrams.

Special spatial resource and transport/service multiagent systems (MAS), with their MAS-models – agents, properties, signatures of operations and relationships, properties autonomy, mobility, intellectuality, cooperativeness [19, 20] is formed by static UML diagrams into life cycle of IoT and IoE with the use the same sequence of technologies and stages as it was considered for EGS.

17.1.3 Dynamic visual UML diagrams for the description of architecture of IoT and IoE-based systems and their analysis

Dynamic visual UML diagrams [5 – 7] serve to describe the dynamic part of architecture of the IoT and IoE-based systems in ordered, temporary representation of processes, their synchronization and interactions of conditions, events and means of processing, in particular for main, component and interface functions and methods of hardware, software, information objects of data collection, transfer and storage, sensory and executing nodes, brokers, servers, administrative stations.

The 6 types of dynamic UML diagrams can be applicable for describing of the communications, survey interaction, the sequences, synchronization, activity and states (automata).

Communications are the interactions of IoT and IoE systems according to intercomponent network interfaces in structure of IoT and IoE. Survey interaction is the special case of communications limited to component functions and communications, essential to the general external consideration of IoT and IoE. The sequences are logic-time diagrams of scenarios of work of IoT and IoE, their components. Synchronization is the special case of the sequences limited to accounting interconnected only. Activity is a kind of graphic schemes of algorithms for functions and methods of processes for IoT and IoE. States (automata) are kinds of graphic schemes of machines of states (Kripke's structures) and automata models for functions and methods of processes for IoT and IoE.

Dynamic visual UML diagrams are used to describe the dynamic part of architecture of complex subsystems, in particular, on base IoT and IoE. The main applications of dynamic UML diagrams for describing structures and systems of IoT and IoE include a number of approaches, cases and steps described below.

General dynamic, spatial and temporary structure, described by dynamic UML diagrams, are formed in the possible sequence of the following procedure of construction:

Step 1. Construction of diagrams of communications, in particular, survey interaction.

Step 2. Construction of diagrams of sequences, in particular, synchronization.

Step 3. Construction of diagrams of activity.

Step 4. Construction of diagrams of states.

Special temporal individual functional architecture of static network environment of IoT and IoE by dynamic UML diagrams is formed in the sequence of the UML-design procedure*, namely for/into:

Step 1. Service network* for modification of monitoring and control (UML-diagrams of communications, sequences, activity, states (UML-CSAS*) for functions of end-points, network, control environment – lighting, power supply, temperature/heating/air conditioning, ventilation, humidity, ionization).

Step 2. Computer network* consisting of wired and wireless network environment (UML-CSAS* for functions of end-points, network, control environment – administrative and end-user stations, routers/access-points, servers).

Step 3. End-point subsystem*, (UML-CSAS* for functions of – developed end-point sensors/actuators, end-point controllers, controller buffers, controller intercomponent interfaces).

Step 4. Node subsystem*, (UML-CSAS* for functions of – developed topology, nodes – brokers/routers/access-points, broker/router/access-point buffers, broker/ /router/access-point intercomponent interfaces).

Step 5. Server subsystem*, (UML-CSAS* for functions of – extended developed topology, servers, server buffers, server intercomponent interfaces).

Step 6. Terminal subsystem*, (UML-CSAS* for functions of – extended developed topology, end-user/administrative terminals, terminal buffers, terminal intercomponent interfaces).

Step 7. Composition these partial developed dynamic UML-CSAS* into general developed system of dynamic diagrams, further – dynamic system*, and their general structure for functions of existing product/resource service network, existing computer network and developed IoT and IoE subsystem.

Temporal structure for special resource properties of dynamic transport/service data flows by dynamic UML diagrams is formed in the sequence of the following UML-design procedure* for/into graph structures (nodes, paths, trees, hammocks, cycles) of dynamic processes through the general temporal hierarchical structure of functions of components of subsystem of IoT and IoE (with functions of all its components), namely for:

Step 1. Function and lows of distribution of values of computational, memory, communicative metric unit, min/max tensions and capacity, capacity of buffers (simple UML-CSAS* and their placement for all trivial graph nodes of dynamic processes, as components of transport/service data flows).

Step 2. The same function and lows of distribution in case of complex UML-CSAS* and their placement for all nontrivial graph structures of dynamic processes.

Step 3. Selected technological standards, interface and communication protocols of computer networks and IoT and IoE for the all above dynamic UML diagrams of the developed resource, transport/service and computer subsystem of IoT and IoE.

Special spatial and temporal resource and transport/service EGS for optimization analyze, their EGS-models is formed by dynamic UML diagrams of IoT and IoE life cycle in the sequence of the corresponding technology from determination to verification and the following stages (each of which includes own special evolutionary steps):

Stage 1. Developed slow evolution by dynamic UML diagrams into life cycles of static general temporal system structure of IoT and IoE.

Stage 2. Developed fast evolution by dynamic UML diagrams into life cycles of dynamic transport/service data flows for temporal hierarchical system structure of IoT and IoE.

Stage 3. Developed coevolution as composition of slow temporal evolution for spatial system structure and fast temporal evolution for transport/service data flows by dynamic UML diagrams.

Special spatial and temporal resource and transport/service MAS for distribution, with their MAS-models is formed by dynamic UML diagrams into life cycle of IoT and IoE in the sequence of the corresponding technology from visual determination to verification and with the use of the same stages as it was believed for EGS.

17.2 Simulation and verification in behavior of IoT and IoE systems on the basis of the Queuing Systems and Petri Nets

The following objects are taken as input for QS:

1) The specifications of the technical description of the architecture of components, subsystems IoT and IoE-based systems, as well as such systems, defining the structure of topological relationships, functions, information objects, interfaces of topological interactions, the temporal behavior of functions and scenarios.

2) The previously prepared QS networks, that define in QS standards, and automata/Petri nets, that define in Petri nets standards, set accordingly resource and behavioral models of process, components, IoT and IoE-based systems, as well as such systems as a whole in analytic-text, tabular, graphical representations, for which

system resource and behavior analysis and simulation and verification are needed, in particular, in the ExtendSim Demo tool environment.

The following objects are considered as output objects for QS and Petri Nets: the obtained correct, verified QS and Petri nets, representing resource and functional models of process, components, subsystems of IoT and IoE-based systems, as well as such systems in general, for which in the corresponding tool environment, in particular, ExtendSim Demo and CPN Tools, system resource and automata analysis, simulation and verification, obtained special conditions, parameters and scenarios of such analysis, simulation, verification, and also special the results of the fulfillment of conditions, the application of parameters and scenarios.

17.2.1 Introduction to the general description of the operation of IoT and IoE systems at the level of resource and functional mode presentation

General description of the operation of IoT and IoE systems at the level of the resource mode presentation is provided by queue systems – QS and on their basis networks of QS [8, 9]. As is known, basic QS includes two main objects – queues and service devices, which process the streams of requests for some service. Queues and service devices are characterized by own special working laws.

The main classification of QS is performed on the basis of structural and functional properties. Accordingly, QS can be: single-channel and multichannel; with expenses (refusal) and without expenses; with expectation and without expectation; with a limited length of turn and not limited length of turn; with limited waiting time and not limited waiting time; with priority and without priority; single-phase and multiphase; opened and closed; Markov and non-Markov; QS compositions and also feature of the use of various QS for imitating modeling of IoT and IoE-based systems, their components and processes.

Except, classification of the QS is executed basrd on the streams of requests, namely: uniform and non-uniform; regular and irregular; recurrent and not recurrent; stationary, ordinary and extraordinary and also, on the basis of features of the use of various streams for imitating

modeling of IoT and IoE-based systems, their components and processes.

Various instrumental environments – frameworks ExtendSim Demo, GPSS World, OMNet ++ [24–26] can be applied for simulating of QS and networks of QS, analyzing the loading of resources of the IoT and IoE and their components.

As it is known, simulation is the imitation of the operation of a real-world process or system over time. The act of simulating something first requires that a model be developed; this model represents the key characteristics or behaviors/functions of the selected physical or abstract system or process.

The model represents the system itself, whereas the simulation represents the operation of the system over time. Verification and validation are independent procedures that are used together for checking that service or system meets requirements and specifications and that it fulfills its intended purpose.

Specifically, specifications, resource modeling and simulation, features of validation and verification of the QS-models of IoT and IoE systems and their components can be performed, in particular, in frameworks ExtendSim Demo, GPSS World, OMNet ++.

The built-in compiler is a computer program that transforms source code, written in a programming language, into another computer language, with the latter often having a binary form, known as object code. Frameworks ExtendSim Demo, GPSS World, OMNet ++ have the following features.

ExtendSim is a proven simulation environment capable of modeling of IoT and IoE. ExtendSim is used to model continuous, discrete event, discrete rate, and agent-based systems. ExtendSim's design facilitates every phase of the simulation project, from creating, validating, and verifying the model, to the construction of a user interface that allows others to analyze the system. Simulation tool developers can use ExtendSim's built-in, compiled language, ModL, to create reusable modeling components. All of this is done within a single, self-contained software program that does not require external interfaces, compilers, or code generators.

GPSSWorld presents a graphical interface with an embedded text editor that allows the definition of the model inside the tool itself. The user can also find in a window all the defined GPSS blocks

implemented in the tool, simplifying the modeling process. GPSSWorld presents the capability to represent the movement of the transactions over the different elements of the model. Language Plus allows to define the more complex behavior (the Plus syntax). GPSS World brings all the simulation primitives up to the user interface, and makes it easy to visualize and manipulate simulations. The result is that simulations can be developed, tested, and understood more quickly than before. There is more to GPSS World than just the GPSS language. Since all the blocks have a graphical representation, the definition of a GPSS process can be represented graphically.

OMNET++ has a domain-specific functionality such as support for sensor networks, wireless ad-hoc networks, Internet protocols, performance modeling, photonic networks, etc., is provided by model frameworks, developed as independent projects. OMNeT++ provides component architecture for models. Components (modules) are programmed in C++, and then assembled into larger components and models using a high-level language (NED). OMNeT++ IDE makes it possible to run simulations directly from the integrated environment. It is possible to run a simulation as a normal C/C++ application and perform C++ source-level debugging on it. The user can also run it as a standalone application or run batches of simulations where runs differ in module parameter settings or random number seeds. **OMNET++ SENSOR NETWORK** helps to communicate among them using radio signals, and deployed in quantity to sense, monitor and understand the physical world. Wireless Sensor nodes are called motes. Wireless Sensor Network is a self-configuring network.

Models and methods for analyzing the functioning of the automaton class, in particular, extended automata [10, 11] and Petri nets [12 – 14], can be used in processes of specification, modeling, simulation, verification and check of the various aspects of behavior of the IoT and IoE systems and their components. The features of such models assume the following classifications and cases for:

1) Automata models, namely, synchronous, asynchronous, temporary, nondeterministic, indistinct, contextual, predicate automata, automata networks and hierarchies and also features of the use of various automata models for behavioral modeling of the IoT and IoE-based systems, their components and processes.

2) Petri nets, namely, simple, temporary, nondeterministic, indistinct, contextual, predicate, painted Petri nets, compositions and hierarchies of Petri nets and also features of the use of various Petri nets for behavioral modeling of IoT and IoE systems, their components and processes.

3) Features in analyze (specification, modeling, simulation and check) of processes in functioning of the IoT and IoE systems, their components on the basis of their representation by:

- synchronous, monoprocessing automata and their compositions;
- asynchronous, multiprocessing Petri nets and their compositions.

For behavior analysis of the IoT and IoE systems and their components frameworks TINA, CPN Tools [27] can be applied.

Simulation is the imitation of the operation of a real-world process or system over time. The act of simulating something first requires that a model be developed, this model represents the key characteristics or behaviors/functions of the selected physical or abstract system or process. The model represents the system itself, whereas the simulation represents the operation of the system over time. As before, verification and validation are independent procedures that are used together for checking that service or system meets requirements/specifications and that it fulfills its intended purpose. The behavioral operating and testing check the conformity of the behavior of the system under check to the behavior of the reference system, in the mode, respectively, for the first, basic operating functioning and, for the second, specific testing functioning.

Specifications, behavioral modeling and simulation, features of validation and verification, elements of operating an testing check of the extended automata [10, 11] and Petri net [12 – 14] models of IoT and IoE systems and their components can be performed, in particular, in frameworks TINA, CPN Tools. Frameworks TINA, CPN Tools have the following features:

TINA (TIme petri Net Analyzer) [27] is a toolbox for the editing and analyzing Petri nets and Time Petri nets, with possibly inhibitor and read arcs, Time Petri Nets, with possibly priorities and stopwatches, and an extension of Time Petri Nets with data handling called Time Transition Systems. The toolbox includes an editor for graphical or textual description of Petri nets and Time Petri nets. TINA can perform construction of reachability graphs, perform structural and path analysis. It also has a conversion tool that translates among several

Petri net formats. TINA accepts as input, descriptions of a Petri net or Time Petri net in textual (.net, .pnml, .tpn formats) or graphical form (.ndr format of files produced by nd, .pnml with graphics). Here “nd” is the textual/graphical editor of TINA called NetDraw. TINA provides a number of options for output formats depending on the flags selected. Options those were useful my thesis were a textual format printing full results and a textual format printing a summary of results. TINA also outputs graphs for various available model checkers and equivalence checkers.

CPN Tools [27] is a tool for editing, simulating and analyzing Colored Petri Nets. The GUI is based on advanced interaction techniques, such as toolglasses, marking menus, and bi-manual interaction. Feedback facilities provide contextual error messages and indicate dependency relationships between net elements. The tool features incremental syntax checking and code generation which take place while a net is being constructed. A fast simulator efficiently handles both untimed and timed nets. Full and partial state spaces can be generated and analyzed, and a standard state space report contains information such as boundedness properties and liveness properties. The functionality of the simulation engine and state space facilities are similar to the corresponding components in Design/CPN, which is a widespread tool for Colored Petri Nets.

Thus, these features of TINA, CPN Tools allow the following stages of behavioral analysis: Stage 1. Specifications/modeling of IoT and IoE of systems and their components in modes: synchronous; asynchronous/event; sequential; parallel. Stage 2. Simulation in modes: step by step; automatic; graph of attainable states and markings. Stage 3. Analysis of correctness for steps: unattainable states and markings; dead-end states and markings; infinite cycles; multiplying markings. Stage 4. Compositional verification for adjacent subsystems of steps: spatial network compositions; spatial hierarchical compositions; temporal network compositions; temporal hierarchical compositions. Stage 5. Operating and testing check for: neighborhoods of states, positions, transition; Rabin-Scott automata for identifiers; operating and testing primitives and fragments; recognizing and check experiments.

17.2.2 Resource imitating modeling and simulation on base of functioning of IoT and IoE systems and their components using QS

Resource imitating modeling, simulation and features of verification of functioning of IoT and IoE systems and their components (in future – end-point sensors/actuators, end-point controllers, zone brokers/routers/access-points, system servers, end-user/administrator terminals and their special subcomponents – buffers, interfaces, memory, control, analyze), that using QS is performed taking into account the basic architecture of IoT and IoE, its components, topological intercomponent structure of interactions (two-point, one-way, star, tree, hierarchical and cluster).

As noted, the structural-functional features of IoT and IoE and their components and also the processes of loading their resources affect the classification of QS with laws for queues, service devices and streams for IoT and IoE. Basic classes of QS for IoT and IoE systems can be represented as simple and difficult [8, 9].

Simple classes – single-channel, without expenses and expectation, with an unlimited length of turn and unlimited waiting time and also with streams of simple classes including uniform, regular, recurrent, stationary and ordinary classes and difficult classes: non-uniform, irregular, not recurrent, non-stationary, extraordinary.

Difficult classes – multichannel with expenses, expectation, a limited length of turn, limited waiting time, Markov and non-Markov – with streams of simple and difficult classes described above.

The QS is possible and appropriate to apply also with more detailed descriptions of processes for resources and their consumption in IoT and IoE systems, their components and topological interaction structure.

In particular, QS can consider the peculiarities of their architecture, multi-level, advanced technologies, including IoT and IoE. These special approaches, architectures, structures and their properties, defining the features of models and methods of IoT and IoE for QS, determine the technology of construction – defining, analysis, modeling, simulation and verification, which contains the following 4 stages.

Stage 1 is executed for construction of components of the general static, spatial topological structure for subsystems of IoT and IoE by models of some type of QS and their objects and components by steps for: end-point sensors/actuators (step 1), end-point controllers – decision making units (step 2), brokers/routers/ access points (step 3), servers (step 4), end-user/administration terminals/work stations (step 5), buffers/memory/storages for all components (step 6), intercomponent interfaces (transport units) for all components (step 7).

Stage 2 provides designing of the general static spatial topological structure of subsystem of IoT and IoE through the intercomponent interfaces by interconnection of models of some type of QS and their objects and topological properties by steps for: end-point sensors/actuators – end-point controllers connection, as point-to-point, star (step 1), end-point controllers – brokers/routers/access points, as point-to-point, star (step 2), brokers/ routers/access points – brokers/routers/ /access points, as point-to-point, one-way, star, tree, hierarchical, hammocks, cycles, mesh and cluster (step 3) brokers/routers/access points – servers, as point-to-point, star (step 4), brokers/routers/access points – end-user/ /administration terminals/work stations, as point-to-point, star (step 5), servers – end-user/administration terminals/ /work stations, as point-to-point, star (step 6), servers – servers, as point-to-point, one-way, star, tree, hierarchical, hammocks, cycles, mesh and cluster (step 7), end-user/administration terminals/work stations – end-user/administration terminals/work stations, as point-to-point, one-way, star, tree, hierarchical, mesh and cluster.

Stage 3 is performed for construction of the general static spatial structure of data flows for the developed resource, transport/service and computer subsystem of IoT and IoE by models of service flows for some type of QS and their objects and properties – bandwidth, queue length, transmission time, delay, transmission errors, its values for individual queues and service devices – individual graph nodes of service flows (step 1), fragments (graph structures) of service flows and service flows generally (step 2), preselected technological standards, interface and communication protocols of computer networks and IoT and IoE for the all above QS models (step 3).

Stage 4 provides designing of the general dynamic temporal structure of data flows for the developed resource, transport/service and computer subsystem of IoT and IoE by models of service flows for

some type of QS and their objects – queues and service devices by steps of defining, analysis, modeling, simulation and verification of laws of distribution of values of service flows for QS-properties on base types of service flows – uniform/non-uniform, regular/irregular, recurrent/not-recurrent, stationary/non-steady, /ordinary/extraordinary, continuity/discreteness – for individual queues and service devices – individual graph nodes of service flows (step 1), fragments (graph structures) of service flows and service flows generally (step 2), post selected technological standards, interface and communication protocols of computer networks and IoT and IoE, based on analysis all above QS-models (step 3).

Special spatial and temporal resource and transport/service EGS or/and MAS with special EGS- and MAS-models is formed by QS-models (in life cycle of IoT and IoE) in the sequence of the corresponding technologies of construction and the following stages:

Stage 1. Developed slow evolution by QS-models of static spatial structure of EGS or/and MAS into life cycles of static general spatial topology system structure of IoT and IoE.

Stage 2. Developed fast evolution by QS-models of dynamic transport/service flows of EGS or/and MAS into life cycles of dynamic transport/service data flows for temporal hierarchical system structure of IoT and IoE.

Stage 3. Developed static and dynamic coevolution of EGS's or/and composition of MAC's for slow spatial system structures and fast transport/service data flows by QS-models.

17.2.3 Behavior imitating modeling of features for functioning of IoT and IoE systems and their components using Petri Nets

Behavior imitating modeling is performed taking into account scenarios and functions of the basic architecture, functions of all components of architecture IoT and IoE, functions of interactions for ports and interfaces into topological intercomponent structure (two-point, one-way, star, tree, hierarchical, mesh and cluster).

The general functional features of IoT and IoE and their components and also their processes affect the classification of automata and Petri Nets with special functions input/output, storage, processing for IoT and IoE.

Special behavioral analysis of IoT and IoE systems and their components is focused on the tasks of the modeling, simulation, analyzing correctness, verification, operating and testing check of the basic, component, interface and subsystem functions presented at the system level of the IoT and IoE architecture.

This special modeling can essentially rely on evolutionary and multiagent models and methods.

Thus, these tasks are defined as follows actions for correctness, verification and also behavioral operating and testing check [28]:

1) The stages and steps for correctness:

Stage 1 confirms that the model has the following properties by steps: The absence of static locks (Step 1). Completeness (Step 2). Unambiguous correspondence of states (Step 3). Lack of redundancy (Step 4). Limitedness (Step 5). Lack of dynamic locks (Step 6). Self-synchronization (Step 7). Partial correctness (Step 8). Complete correctness (Step 9). Security (Step 10). Liveliness (Step 11).

Stage 2 selects the basis of the methods for analyzing the correctness, these methods of check is performed of: Analysis of achievable states and mapping in classical and improved version (the method of dialog matrices). Phase diagrams. Adjacent states. Joint paths that preliminarily identifies static locks.

Stage 3 lowers the dimension of the model of achievable states due steps: Structural and functional decomposition (Step 1). Previously created "equivalent" states (Step 2). Limiting the number of parameters and detectable errors (Step 3).

2) Verification, that includes the following stages and steps:

Stage 1 proves that the specification of the service objects of some verifiable level, together with: The specification of the lower level, that are used by these service objects, is consistent with the description provided by the checked level (Step 1). The specification of the higher level, that use these service objects, is consistent with the description provided by the checked level (Step 2).

Stage 2 is performed on the basis of methods of: analysis of the achievable states and markings in a version extended in comparison with the analysis of correctness (Step 1); logical induction by the number of events based on axioms and verification rules (Step 2); time logic method with confirmation of safety and liveliness (Step 3).

Stage 3 achieves using the methods of: combining, that is based on methods of analysis of achievable states and logical induction, where the development of the system is reflected in the states it achieves, and the requirements for the system (service) in statements (Step 1); structural induction, that is on the basis of abstract data types, with the proof that low-level specification implements high-level specification; axiomatic, that is in the specifications of formal languages (Step 2).

3) The stages and steps of behavioral operating and testing check:

Stage 1 consists in verifying, that the behavior of the system on the conceptual boundary with the environment corresponds to the intended one.

Stage 2 allows to get test scenarios, like recognizing and checking experiments, in terms of abstract service primitives and data elements of the system;

Stage 3 consists in passive recognizing experiment of the automata class by behavioral on-line testing. This is based on a formal method of recognizing behavioral check in the external flow of the system's operational functioning based on the identification of reference states (Step 1) and it establishes the conformity of the reference and verified models by searching for recognizing primitives and fragments in a fixed working behavior of the system (Step 2).

Stage 4 executes an active checking experiment of the automaton class by behavioral testing check. This is based on a formal method of constructing behavioral checks in the internal specially formed stream of test functioning of the system based on the identification of reference states (Step 1) and it establishes the correspondence of the reference and verified models, for which it embeds checking primitives and fragments in the test behavior of the system (Step 2).

Special EGS with their genes, chromosomes, individuals, populations, signatures of operations, relationships, evolutions and special MAS with their agents, signatures of operations and relationships are used for consideration of the special behavior of automata and Petri nets. In particular, they examine correctness by verification, on-line testing and testing check in life cycle of IoT and IoE of SBC (in properties: autonomy; mobility; intellectuality; cooperativeness) for stages: behavior hosting analyzes of stationary and mobile, static communication network environment, as special slowly developing EGS or MAS (stage 1); behavior analyze of processes of

dynamic transport service flows, as special highly developing EGS or MAS (stage 2); cooperation of behavior analyze of static communication and dynamic transport evolution systems or MAS's (stage 3).

17.3 Simulation and verification of synchronization processes in IoT and IoE-based systems on the basis of temporal logic

The following objects are taken as input:

1. The specifications of the technical description of the architecture of components, subsystems IoT and IoE-based systems, as well as such systems, defining the structure of topological relationships, functions, information objects, interfaces of topological interactions, the temporal behavior of functions and scenarios.

2. The previously prepared abstract-temporal models that define in the LTL and CTL time logic standards the properties of mutual ordering and synchronization of abstract-time conditions, events and process relationships for components, IoT and IoE-based systems, as well as such systems as a whole in analytic-text, tabular, graphical representations for which it is necessary to perform system time analysis, simulation and verification, in particular, in the XSPIN tool environment.

The following objects are considered as output objects: The obtained correct, verified abstract-temporal models, representing in LTL and CTL temporal logic standards, the properties of mutual ordering and synchronization of abstract-temporal conditions, events and process relations for components, IoT and IoE-based systems, as well as such systems, for which system time analysis, simulation and verification, special conditions obtained, parameter, are performed in the appropriate tool environment, in particular, XSPIN. s and scenarios for the organization of such analyzes, simulations, verifications, special results of condition fulfillment, application of parameters and scenarios.

17.3.1 Introduction to specification of synchronization process in IoT and IoE-based systems by using of temporal logic

The temporal logic of Linear Temporal Logic (LTL) and Concurrent Temporal Logic (CTL) [15, 16] can be used for analytical analysis and synthesis of temporal expressions, proof of the conclusions

in the synchronization of processes in IoT and IoE systems and their components. Temporal analysis involves the following parts:

1) Temporal classification of objects, properties, conditions, events, relations, operations, laws of temporal logic of processes of functioning of IoT and IoE and their components.

2) Special models and methods – logical conclusion, machine of states, Kripke's structures, Buchy automata, Promela specifications.

3) Distinctions of objects, properties, the relations, operations, laws, a logical conclusion of temporal logic of CTL from LTL for processes of functioning of the IoT and IoE systems and their components.

4) Features of temporary specification, analytical analysis and synthesis of expressions, proofs of conclusions at synchronization of conditions and events in processes of functioning of IoT and IoE and their components on the basis of their representation by expressions and conclusions of temporal logic.

Various instrumental environments and linguistic tools for analytical time analysis and synthesis of LTL and CTL time-logic expressions and outputs [15, 16] are used to synchronize of processes of IoT and IoE and their components. Among such environments are, in particular, Promela language, SPIN and XSPIN framework [27].

These tools allow to solve the following tasks:

1) Description of expressions and conclusions of temporal logic in Promela language for synchronization of conditions and events of processes in functioning of IoT and IoE and their components.

2) Specifications of expressions and conclusions of temporal logic for synchronization of conditions and events of processes in functioning of IoT and IoE and their components in the tool environments SPIN and XSPIN.

3) Temporary asynchronous and event interpretation of expressions and conclusions temporal logic, as analytical temporary models of processes in functioning of IoT and IoE and their components, in the tool environments SPIN and XSPIN.

4) Features of the analysis and synthesis of expressions, proofs of conclusions of temporal logic, as analytical temporary models of processes in functioning of IoT and IoE and their components, in the tool environments SPIN and XSPIN.

17.3.2 Simulation and verification of IoT and IoE-based systems at the level of temporal logic

Temporal simulation and verification of IoT and IoE-based systems at the level of Temporal Logic (LTL) [15, 16] have their own peculiarities for synchronizing the processes and interactions of IoT and IoE and their components, depending on the task being solved and the model.

Temporal specifications and modeling of processes synchronization of the main component and interface functions of IoT and IoE, in particular, temporary expressions; Kripke's structures; Buchy automata; Promela language, include stages and steps [29, 30]:

Stage 1. Temporary analysis of conditions and events at synchronization on the basis of temporary expressions, Kripke's structures, Promela language, in particular, for: general dynamic, spatial and temporary structures of IoT and IoE on base of the UML (Step 1); special structures of dynamic transport service flows on base of the QS (Step 2); processes of functioning of IoT and IoE, their components on the basis of their representation by automata and the Petri nets (Step 3).

Stage 2. Temporary transformation, conclusion and proof of expressions for conditions and events at synchronization by repeating a sequence of stages which are considered for temporary analysis.

Stage 3. Temporary verification, optional working and testing check for functioning of IoT and IoE and their components by repeating a sequence of stages for temporary analysis and transformation.

17.3.3 Special temporal simulation and verification of IoT and IoE-based systems

Special temporary analyzes of EGS properties for optimization or/and MAS properties for distribution are used for temporary simulation and verification at the following stages:

Stage 1. Temporary EGS- or MAS-analysis of special dynamic, spatial and temporary entities, relations and properties of components, subsystems and SBC on base of the UML-evolution or UML-MAS.

Stage 2. Temporary EGS- or MAS-analysis of special structures of dynamic transport service flows through queues and service devices of

components, subsystems and SBC on base of the QS-evolution or the QS-MAS.

Stage 3. Temporary EGS- or MAS-analysis of special processes of transformation of states, conditions, events, actions, markers in functioning of components, subsystems and SBC on the base of Petri-net-evolution or Petri-net-MAS.

17.4 Work related analysis

Development of IoT and IoE systems was prepared by improvement of formal methods, techniques and tools for designing and the analysis of the distributed computer systems, Internet technologies, technologies of Green computing and the communications directed to resource-saving, functional safety and information security, having business and social components in the scientific, industrial and educational sphere.

Three-level system, behavioural, temporary modeling and simulation of IoT and IoE systems widely uses the formalism of UML diagrams, foundations of QS and Petri nets, elements of temporal logic for verification of properties in processes of functioning of the components and systems in general.

Formation and development of the tools of UML diagrams for the description of static and dynamic processes of interaction of components found reflection in works of Grady Booch, James Rumbaugh, Ivar Jacobson. UML became one of new paradigms of the object-oriented methods on the basis of development in fundamental elements of the object model, such as abstraction with focus on interface and separate consideration of behaviour and the implementation, hierarchy as a way of ordering abstractions, encapsulation like complementary to abstraction and modularity based on a common “Divide and conquer” approach [5, 6].

I. Sander, J. Obergr from KHT, Sweden, presented the connection between a framework dedicated to the enrichment of UML with formal semantics, a framework based on formal models of computation supporting validation by simulation, and a system synthesis tool targeting a flexible platform with well-defined execution services [31].

UML enhances focus on modeling, and the UML diagrams gain development for check of nonfunctional properties, including requirements to performance and the dependability [7].

Queueing Theory is represented for common case of its implementation in IoT and IoE systems during their life cycle by using of the QS-models. A number of elementary queueing models with attention to methods for the analysis of these models, and also to their applications, including production systems, transportation and stocking systems, communication systems and information processing systems important for IoT and IoE area is considered. Queueing models are particularly useful for the design of these system in terms of layout, capacities, control and verification [8].

Except elements of queueing theory are applied to providing of the healthiness for modelling and availability assessment of mobile healthcare IoT. Exponentially growing technology – Internet of Things (IoT) in the field of healthcare is spoken about the networked healthcare and medical architecture. Networked medical and healthcare devices and their applications create an Internet of Medical Things for better health monitoring and preventive care with the use of tree analysis and queueing theory [9].

The use of state machines and the Petri nets for the analyses of basic, component and interface functions, represented at the system and structural-behavioral levels of the IoT and IoE architecture is considered for verification of its behavioral models [10, 11].

Extended automata and Petri nets, which support models and methods in analyzing the functioning of the distributed systems are important for research in behavior of the IoT and IoE systems and their components concerning the various aspects including processes of specification, modeling, simulation and verification are represented in [12, 13].

A. Yakovlev from New-Castle University, UK. considered problematic circuit behaviour, such as potential hazards and deadlocks, in a reasonable amount of time a technique. It is required which would avoid exhaustive exploration of the state space of the system, this paper presents a special type of Petri nets to represent circuits. An algorithm for automatic conversion of a circuit netlist into a behaviourally equivalent Petri net is proposed. Once the circuit Petri net is constructed and composed with the provided environment specification,

the presence and reachability of troublesome states is verified by using methods based on finite prefixes of Petri net unfoldings. The shortest trace leading to a deadlock or a hazard in the circuit Petri net is mapped back onto the gate-level representation of the circuit, thus assisting a designer in solving the problem. The method has been automated and compared against previously existing circuit verification tools [14].

Luca Ferrucci, Marcello M. Bersani and Manuel Mazzara describe a business workflow case study with abnormal behavior management and demonstrate how temporal logics and model checking can provide a methodology to iteratively revise the design and obtain a correct-by construction system [32].

Multi-agent and evolutionary approaches, genetic algorithms applied for research in behavior of the IoT and IoE systems and their components considered are examined in a point of the life cycle. Genetic algorithms, the best-known technique in the area of evolutionary computations, numerical optimization and various applications of evolutions programs important for IoT and IoE systems are represented in [20 – 23].

The impact of the IoT and IoE systems on society, a problem of dependable IoT development for human and industry, techniques of modelling and the assessment for dependable and secure IoT, implementation of IoT for smart cities, business and industry application are considered in [2, 30].

Conclusions and questions

The formation of a knowledge system of a formalized description, analysis and synthesis of Internet of Things systems is becoming an important part of the process of training specialists in the field of computer science. Such formalization presupposes a formal study of the models of the components of IoT and IoT as a whole, the modeling and verification of their properties and the process of functioning.

At the system behavioral level of IoT, research is carried out for components, subsystems and IoT as a whole, taking into account their structural, functional, informational, and interface features.

In this section, a three-level system, behavioral, and temporal modeling and simulation of IoT systems is considered using UML diagrams, QS and Petri nets, and time logic, respectively.

Visual modeling, simulation and verification of architectures for IoT, IoE and IoT SBC systems based on UML visual diagrams are based on static diagrams that describe the static part of the architecture in representing the structure, their components, component functions and information objects, and intercomponent interfaces with their formats conditions, events and means of processing, as well as on dynamic diagrams, which are used to describe the dynamic part of the architecture in an ordered, temporal representation of processes, their synchronization and interaction of conditions, events and means of processing.

Resource models of building, resource simulation and verification of architectures for IoT, IoE and IoT SBC systems based on queuing systems (QS) and queuing nets QS during the operation of IoT and IoE systems and their components, which is performed basic architecture, its components, structure of topological interaction. In QS, it is possible and appropriate to use a detailed description of the processes for resources and their consumption in the IoT and IoE systems, their components and the structure of the topological interaction.

Behavioral models, simulation, correctness analysis, verification and testing of architectures and processes for IoT, IoE and IoT SBC systems based on advanced state machines and Petri nets, that are focused on the analyses of basic, component and interface functions, presented at the system and structural-behavioral levels of the IoT and IoE architecture.

In the process of such analysis, special evolutionary genetic systems with evolutions, their genes, individuals, populations, signatures of operations and relationships, and also multi-agent systems with their agents, signatures of operations and relationships, properties of autonomy, mobility, intellectuality, cooperativity are used for the special behavior of automata and Petri nets in the life cycle of IoT and IoE SBC.

The construction of temporal models, temporal simulation, synchronization check and verification of IoT, IoE and IoT SBC systems are based on LTL and CTL temporal logic taking into account

time delays and synchronization features of IoT and IoE processes and their components depending on the tasks they solve.

1. What can describe and model UML diagrams, SMS resource models, automata and Petri nets, temporary logic?
2. What features distinguish tool environments Star UML, MS Visual.NET (UML), ExtendSim Demo, CPN Tools, SPIN?
3. What and how to simulate UML diagrams?
4. What are the features of verification UML diagrams?
5. What are the features of static UML diagrams, which static diagrams are used for visual analysis of IoT, IoE and IoT SBC?
6. What are the features of dynamic UML diagrams, which dynamic diagrams are used for visual analysis of IoT, IoE and IoT SBC?
7. What components and QS types are applicable for IoT, IoE and IoT SBC analysis?
8. What and how can QS be modeled in IoT, IoE and IoT SBC systems?
9. What are the features of QS verification for IoT, IoE and IoT SBC systems?
10. What are the features of QS application at different levels of IoT, IoE and IoT SBC systems?
11. What and how do they allow to simulate automatons and Petri nets in IoT, IoE and IoT SBC systems?
12. What are the features of verification and testing of machines and Petri QS networks for IoT, IoE and IoT SBC systems?
13. What is the difference between evolutionary and multi-agent modeling and verification of automatics and Petri QS networks for IoT, IoE and IoT SBC systems?
14. What and how do you model temporal models based on LTL and CTL temporal logic in IoT, IoE and IoT SBC systems?
15. What are the features of synchronization and verification of temporal models based on LTL and CTL temporal logic for IoT, IoE and IoT SBC systems?

References

1. Pallavi Sethi and Smruti R. Sarangi Internet of Things: Architectures, Protocols, and Applications // Journal of Electrical and Computer Engineering Volume 2017, Article ID 9324035, 25 p. <https://doi.org/10.1155/2017/9324035>
2. Dependable IoT for Human and Industry: Modeling, Architecting, Implementation, Vyacheslav Kharchenko, Ah Lian Kor, Andrzej Rucinski (Eds), River Publishers Series in Information Science and Technology, 2018, 450 p.
3. Tara Salman Networking Protocols and Standards for Internet of Things. – https://www.cse.wustl.edu/~jain/cse570-15/ftp/iot_prot/index.html
4. Saber Talari, Miadreza Shafie-khah, Pierluigi Siano, Vincenzo Loia, Aurelio Tommasetti and João P. S. Catalão. A Review of Smart Cities Based on the Internet of Things Concept, *Energies* 2017, 10, 421. 23 p. <http://www.mdpi.com/1996-1073/10/4/421/pdf>
5. Rumbaugh James The unified modeling language reference manual – 2-nd edition / James Rumbaugh, Ivar Jacobson, Grady Booch. Addison-Wesley on Web: <http://www.awprofessional.com> Available from: https://www.utdallas.edu/~chung/Fujitsu/UML_2.0/Rumbaugh--UML_2.0_Reference_CD.pdf.
6. Grady Booch James Rumbaugh Ivar Jacobson The Unified Modeling Language User Guide. Addison-Wesley Longman Inc., 1999. 391 p. Available from: <https://pdfs.semanticscholar.org/fc51/1dcebd3dae76133d5dbbda4250bebd0fb5e3.pdf>
7. Toledo Rodríguez F., Lonetti F., Bertolino A., Polo Usaola M., Pérez L. B. Extending UML testing profile towards non-functional test modeling Second International Conference on Model-Driven Engineering and Software Development, pp. 488–497, Lisbon, 7 - 9 January 2014.
8. Ivo Adan, Jacques Resing, Queueing Systems. Department of Mathematics and Computing Science Eindhoven University of Technology, March 26, 2015. 182 p. Available from: <https://www.win.tue.nl/~iadan/queueing.pdf>
9. A. A. Strielkina, D. D. Uzun, V. S. Kharchenko, A. H. Tetskyi. Modelling and Availability Assessment of Mobile Healthcare IoT Using Tree Analysis and Queueing Theory. In book: Dependable IoT for Human and Industry: Modeling, Architecting, Implementation, Vyacheslav Kharchenko, Ah Lian Kor, Andrzej Rucinski (Eds.), River Publishers Series in Information Science and Technology, 2018.
10. Javier Esparza Automata theory. An algorithmic approach. Lecture Notes. August 26, 2017. 321 p. Available from: <https://www7.in.tum.de/~esparza/autoskript.pdf>

11. Y. Kondratenko, O. Kozlov, A. Topalov, O. Korobko, O. Gerasin. Automation of Control Processes in Specialized Pyrolysis Complexes Based on Industrial Internet of Things. In book: Dependable IoT for Human and Industry: Modeling, Architecting, Implementation, Vyacheslav Kharchenko, Ah Lian Kor, Andrzej Rucinski (Eds.), River Publishers Series in Information Science and Technology, 2018.

12. Jorg Desel, Javier Esparza Free Choice Petri Nets. Cambridge University Press, Cambridge, 1995. 256 p. Available from: <https://www7.in.tum.de/~esparza/fcbook-middle.pdf>

13. J. Kleijn and A. Yakovlev (Eds). Petri nets and Other Models of Concurrency – ICATPN 2007, Lecture Notes in Computer Science, vol. 4546, ISBN 978-3-54073093-4, Springer-Verlag, 2007, 515 p.

14. I. Poliakov, A. Mokhov, A. Rafiev, D. Sokolov and A. Yakovlev. Automated Verification of Asynchronous Circuits Using Circuit Petri Nets, Proceedings of the 14th IEEE International Symposium on Asynchronous Circuits and Systems, Newcastle upon Tyne, UK, April 2008, pp. 161-170. DOI: 10.1109/ASYNC.2008.18

15. Daniel Shahaf Temporal Logics I: Theory. Tel-Aviv University November 2007. P. 155. Available from: http://www.cs.tau.ac.il/~annaz/teaching/TAU_winter08/Seminar/daniel.pdf

16. Patricia Bouyer Model-Checking Timed Temporal Logics. LSV – CNRS & ENS de Cachan – France. 142 p. Available from: <http://www.lsv.fr/~bouyer/files/tfit08.pdf>

17. For Programmer (UML diagrams in Visual Studio Feature Pack) <http://cc.ee.ntu.edu.tw/~farn/courses/BCC/NTUEE/2012.spring/vs.uml.instruction.pdf>

18. *Dan Simon Evolutionary Optimization Algorithms*. Biologically-Inspired and Population-Based Approaches to Computer Intelligence. Wiley, Cleveland State University, 2013. 727 p. https://books.google.com.ua/books?hl=en&lr=&id=gwUwIEPqk30C&oi=fnd&pg=PP1&dq=computer+evolutionary-genetic+systems+pdf&ots=Glm3DqUag2&sig=UeVaj6EE41SAdXKgeuMMQ6LtUyM&redir_esc=y#v=onepage&q=computer%20evolutionary-genetic%20systems%20pdf&f=false

19. Yoav Shoham, Kevin Leyton-Brown Multiagent Systems. Algorithmic, Game-Theoretic, and Logical Foundations. Revision 1.1. Shoham and Leyton-Brown, 2010. 532 p. <http://www.masfoundations.org/mas.pdf>

20. A. Sugak, O. Martynyuk, O. Drozd. The Hybrid Agent Model of Behavioral Testing, International Journal of Computing, 2015, Volume 14, Issue 4, Ternopil, pp. 232-244.

21. Zbigniew Michalewicz Genetic Algorithms + Data Structures = Evolution Programs. Third Edition. / Springer, 1996. 388 p. <http://web.ist.utl.pt/adriano.simoese/tese/referencias/Michalewicz%20Z.%20Genetic%20Algorithms%20+%20Data%20Structures%20=%20Evolution%20Programs%20%283ed%29.PDF>

22. A. Sugak, O. Martynyuk, O. Drozd. Models of the Mutation and Immunity in Test Behavioral Evolution, Proceedings of the 2015 8th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2015, Warsaw, Poland, pp. 790-795.

23. O. Martynyuk, A. Sugak, D. Martynyuk, O. Drozd. Evolutionary Network of Testing of the Distributed Information Systems, Proceedings of the 2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2017, Bucharest, Romania, pp. 888-893. <https://ieeexplore.ieee.org/document/8095215>

24. ExtendSim. User Guide / Imagine That Inc. 2007. P. 808. http://www.edgestone-it.com/papers/ExtendSim7_Manual.pdf

25. I. Skarga-Bandurova, M. Derkach, A. Velykzhanin, A Framework for Real-Time Public Transport Information Acquisition and Arrival Time Prediction Based on GPS Data. In book: Dependable IoT for Human and Industry: Modeling, Architecting, Implementation, Vyacheslav Kharchenko, Ah Lian Kor, Andrzej Rucinski (Eds.), River Publishers Series in Information Science and Technology, 2018.

26. OMNeT++. Simulation Manual. Version 5.4.1 / András Varga and OpenSim Ltd. 2016. 538 p. <https://www.omnetpp.org/doc/omnetpp/SimulationManual.pdf>

27. Michael Westergaard CPN Tools / Eindhoven, 2010. – P. 46. <https://westergaard.eu/wp-content/uploads/2010/09/CPN-Tools.pdf>

28. Anduo Wang Formal Analysis of Network Protocols. University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-10-16. 2010. 32 p. https://repository.upenn.edu/cgi/viewcontent.cgi?article=1970&context=cis_reports

29. Daniel Câmara Formal Verification of Communication Protocols for Wireless Networks. Belo Horizonte, 2009. 136 p. http://www.eurecom.fr/~camara/files/ThesisCamara_FormalVerification.pdf

30. A. Boyarchuk, V. Kharchenko, O. Illiashenko, D. Maevsky, C. Phillips, A. Plakhteev, L. Vystorobskaya. Internet of Things for Human and Industry Applications: ALIOT Based Curriculum. In book: Dependable IoT for Human and Industry: Modeling, Architecting, Implementation, Vyacheslav

Kharchenko, Ah Lian Kor, Andrzej Rucinski (Eds.), River Publishers Series in Information Science and Technology, 2018.

31. P. I. Diallo, S.-H. Attarzadeh-Niaki, F. Robino, I. Sander, J. Champeau, J. Oberg. A formal, model-driven design flow for system simulation and multi-core implementation. 10th IEEE International Symposium on Industrial Embedded Systems (SIES), 2015.

32. L. Ferrucci, M. M. Bersani, M. Mazzara. An LTL semantics of business workflows with recovery. 9th International Conference on Software Paradigm Trends (ICSOFT-PT), 2014.

18. MARKOV'S MODELLING OF IOT SYSTEMS

DrS. Prof. V. S. Kharchenko, Dr., Ass. Prof. M. O. Kolisnyk (KhAI)

Contents

Abbreviations	77
18.1 Features of Markov's modeling of IoT systems	78
18.1.1 Principles of Markov's and semi Markov's modeling.....	78
18.1.2 Features and assumptions for functionality modeling	79
18.1.3 Features and assumptions for availability modeling	81
18.1.4 Metrics and indicators	83
18.2 Markov's modeling of IoT systems reliability and availability...	85
18.2.1 Technique	86
18.2.2 Development of models.....	88
18.2.3 Research of models.....	90
18.3 Markov's modeling of IoT systems cyber security and availability	93
18.3.1 Technique	93
18.3.2 Development of Markov model of IoT system functioning	94
18.3.3 Research of models.....	96
18.4 Semi Markov's modeling of IoT systems.....	99
18.4.1 Technique	99
18.5 Work related analysis	105
Conclusions and questions.....	106
References	107

Abbreviations

DoS – Denial-of-Service
DDoS – Distributed Denial-of-Service
HMM – Hidden Markov Model
HSMM - Hidden Semi-Markov Model
IoT – Internet of Things
MTBF - Mean Time Between Failures
MTTF - Mean Time to Failure
MTTR - Mean Time to Recover
NLP - Natural Language Processing
RUL - Remaining Useful Lifetime
SA - Service Available
SMR - Service May Recover
SMNR - Service May Not Recover
SNA - Service Not Available
TTF - Time to Failure
TTR - Time to Recover

18.1 Features of Markov's modeling of IoT systems

This section describes the features of Markov modeling in the Internet of Things systems. This book is intended for MSc-, PhD-students and engineers, who will be involved in design and development of such integrated projects, so we will provide an overview of the modeling of the function process of IoT system with use of Markov models. This chapter covers the following topics:

- Principles of Markov's and semi Markov's modeling.
- Features and assumptions for functionality modeling.
- Features and assumptions for availability modeling.
- Metrics and indicators.

To start with, we will consider characteristics of the Big Data and try to highlight the most important from the IoT point of view ones.

18.1.1 Principles of Markov's and semi Markov's modeling

The question of the expediency of using the theory of Markov processes for solving one or another practical problem is determined, first of all, by its content and the possibility of constructing for it a Markov model, on the one hand, not very complicated, and on the other - that adequately reflects the regularities that are inherent in the task. The correct justification for the fundamental possibility of using the Markov model is the first and very crucial stage in solving the problem. The second stage, no less responsible, is to decide on the specific type of Markov model to use and with which parameters. Of course, one can formulate the problem differently - both as a continuous one and as a discrete one. Models based on Markov chains are simpler and clearer than models that use discrete or continuous Markov processes. In addition, they are easier to model with the use of computer technology. Therefore, if there are no compelling reasons to use other models, Markov chains should be preferred. An important feature of the discrete Markov process is the property of singularity, which means, in this case, that the probability of transition to any new state for a short time Δt is significantly less than the probability that the state remains unchanged.

18.1.2 Features and assumptions for functionality modeling

A Markov model [1] is a stochastic method for randomly changing systems where it is assumed that future states do not depend on past states. These models show all possible states as well as the transitions, rate of transitions and probabilities between them.

Markov models are often used to model the probabilities of different states and the rates of transitions among them. The method is generally used to model systems. Markov models can also be used to recognize patterns, make predictions and to learn the statistics of sequential data.

There are four types of Markov models that are used situationally [2]:

- Markov chain - used by systems that are autonomous and have fully observable states.
- Hidden Markov model - used by systems that are autonomous where the state is partially observable.
- Markov decision processes - used by controlled systems with a fully observable state.
- Partially observable Markov decision processes - used by controlled systems where the state is partially observable.

Markov models can be expressed in equations or in graphical models. Graphic Markov models typically use circles (each containing states) and directional arrows to indicate possible transitional changes between them. The directional arrows are labeled with the rate or the variable one for the rate. Applications of Markov modeling include modeling languages, natural language processing (NLP), image processing, bioinformatics, speech recognition and modeling computer hardware and software systems.

Consider the random process $X(t)$ in which the region T of definition of an argument is a continuous set of points $t \in T$, and the space of states is a discrete set of points S , $\Theta_l \in S$, $l=1\dots L$. At any random moments of time $t_0 < t_1 < \dots$ changes in the state may occur. Such a process is a discrete random function.

Definition [2]. The discrete Markov process is called a discrete random function for which the one-dimensional distribution function

$$F_1(x_N; t_N / x_0, \dots, x_{N-1}; t_0, \dots, t_{N-1}) = F_1(x_N; t_N / x_{N-1}; t_{N-1}).$$

Note: From the above equality it follows that the probability that a random variable

$$P\{x_N; t_N / x_0, \dots, x_{N-1}; t_0, \dots, t_{N-1}\}$$

will be taken value, provided that the random variables take values $X(t_N)$, is determined by the equality $x_N \in S$ if random values $X(t_0), \dots, X(t_{N-1})$.

Semi-Markov [3-7] is called a discrete random process $X(t)$, one-step transitions of which from the state Θ_j ($j=1 \dots L$) to the state Θ_k ($j=1 \dots L$) are described by the matrix Π of the probabilities of one-step transitions with the elements π_{jk} , and the time of stay in the state T_{jk} until the transition Θ_j to the state Θ_k by matrix $F(t)$ of probability density with elements $f_{jk}(t)$ (fig. 18.1).

Remark 1. For the complete probabilistic description of the semi-Markov process, in addition to the matrices Π , we must set the initial conditions, namely the state $F(t)$ i Θ_j at the instant of time t_0 .

Remark 2. A characteristic feature of the semi-Markov process is that the matrices Π and $F(t)$ do not depend on the behavior of the process outside the considered steps.

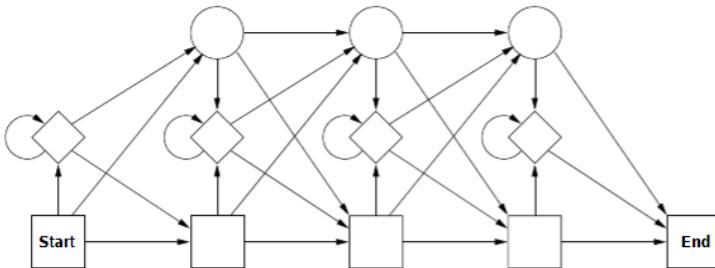


Fig. 18.1 – Semi-Markov model

Remark 3. If the term of stay in the states is a value T_c (with $f_{jk}(t) = \delta(t-T_c)$), then the semi-Markov process at time points $t=0, T_c, 2T_c$ is a

homogeneous Markov chain, which is called an embedded Markov chain.

Remark 4. In general, the semi-Markov process is not Markov, but it is proved that, in a particular case, the probability density $f_{jk}(t)$ ($j=1\dots L$) does not depend on the states Θ_j and Θ_k and are described by the exponential function, $f_{jk}(t) = v \cdot \exp(-vt)$, v is a constant; the semi-Markov process is a discrete Markov process.

18.1.3 Features and assumptions for availability modeling

IoT systems combined with their high-availability requirements means that these systems are more at risk of unintended, non-malicious downtime [8-13]. When designing IoT system, it is necessary to provide the security of the operation and the reliability of hardware and software components of the system. Understanding new communication protocols, hardware types, and obscure operating systems is difficult, making IoT security an incredible challenge.

In the network equipment that used for the organization of IoT system, according to statistics, more and more vulnerabilities found in software code. When exposed to hacker attacks via these vulnerabilities can be stolen proprietary information of the company, and making failure of the software and hardware components of network devices and servers. Manufacturers proposes decisions on the release of patch, redundancy of components to reduce the risks of vulnerabilities of network equipment in IoT. However, vulnerabilities are discovered again and again, and the attacks translates them inoperable technical condition. In order to provide network dependability of IoT, which includes providing a high reliability and high safety at the required level, it is necessary to develop a mathematical model for a more accurate quantification.

Assumptions in the development of the model [14]:

- stream hardware failures of the system obeys Poisson distribution;
- the flow of failures of subsystems is subject to Poisson for-grabs, as the results of monitoring and diagnostics, anti-virus software testing corrected secondary error (the result of the accumulation of the effects of primary errors and defects, bookmarks), and to fix a malfunction or failure of software, eliminating or the consequences of

software bookmarks and code vulnerabilities, DoS- and DDoS-attacks, the number of primary software defects permanently. Therefore, the assumption is true, that the flow of software failures obeys Poisson distribution, the failure rate is constant;

- the model does not take into account that eliminating software vulnerabilities and design faults changes the parameters of the flow of failures (and recovering). To investigate the IoT system dependability use the theory of Markov models, as the failure rates of hardware and software and the availability of software vulnerabilities is constant.

Fig. 18.2 is a Markov graph of functioning of the main subsystems of IoT system, λ - the rate of failure or attack, μ - the rate of the recovery system.

The basic state of the system [14]:

- 1) Normal condition (up-state) system.
- 2) Failure due to faulty feeder from the stationary power supply (220 V).
- 3) Failure due to a malfunction of the second feeder (a solar battery).
- 4) Failure of the battery in the UPS.
- 5) Reconfigure the power subsystem;
- 6) Failure of the cable connecting the Router and Server.
- 7) Failure of the cable connecting the UPS and Switch, and / or the Router, and / or Server.
- 8) Failure of the cable connecting the Router and Switch.
- 9) Firewall Denial.
- 10) Refusal Server due to a fault server components, or exposure to attacks on the code server system software with vulnerabilities.
- 11) Failure Router as a result of failure of the router components, or the impact of the attacks on the code of the router operating system vulnerabilities.
- 12) Switch Failure due to a fault switch components, or exposure to an attack on the system software code switch with the presence of vulnerabilities.
- 13) Partial failure of the system due to the failure of cable connecting any or multiple sensors and IP cameras.
- 14) Partial failure of the system due to the failure of any one or more sensors and IP cameras.

15) Failure of the system.

IoT system availability function $AC(t)$ is defined as the sum of the probabilities of staying the system in an up-states:

$$AC(t) = P_1(t) + P_5(t).$$

Solving the system of Kolmogorov-Chapman equations, can get the value of the availability function components and SBS network, the number of network failures due to software vulnerabilities, and how and with what intensity the system is restored after such failures. It follows that service availability, service continuity, cyber security, data integrity, resilience and high dependability of software and hardware should be inherent in IoT networks.

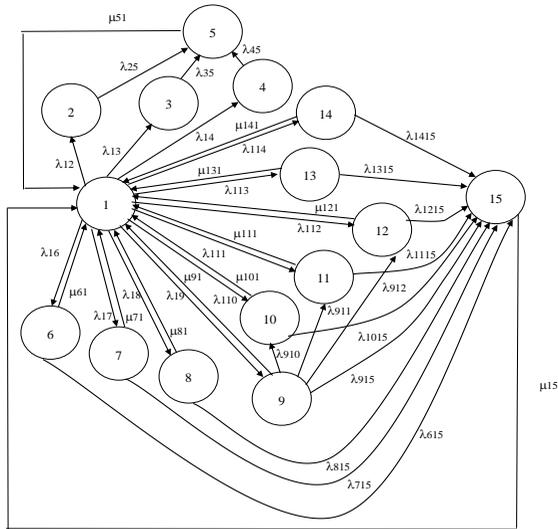


Fig. 18.2 – Markov's graph of IoT system's states

18.1.4 Metrics and indicators

Using a unified fail to recovery model that assumes time to failure (TTF) and time to recover (TTR) are exponentially distributed for all the three cases. Suppose once the system becomes operational, it takes certain time to fail again. The average time it takes the system to fail is called MTTF (mean time to failure). Once the system fails it takes

certain time to recover from failure and return to operational state. The average time it takes for the system to recover is called MTTR (mean time to recover). The average time between failures is called MTBF (mean time between failures) and can be written as

$$MTBF = MTTF + MTTR, \quad (18.1)$$

shown in table 18.1.

Availability is defined as the fraction of time that a component is operational [15].

Table 18.1 – Metrics for IoT systems maintenance

Category	Parameter	Metrics
Design	Inherent Reliability	MTBF
	Fault Detection & Isolation	CND RTOK
	Scheduled Maintenance	MTBPM M_{pt}
	Maintainability	MTTR
Maintenance Infrastructure	Maintenance Workforce	No. of Maintainers
	Spares Availability	MALDT
	Administration Time	
Operation	Remote Maintenance	P_{remote}

Metrics: Service Available (SA), Service May Recover (SMR), Service May Not Recover (SMNR), Service Not Available (SNA) use for description of cloud availability in IoT infrastructure.

In all such cases the service requests may encounter unavailable web service. But it may happen that in next interval some of the services may be available after QoS satisfaction. Hence two more status is introduced known as Service may recover and Service may not recover.

Service Available (SA): This status indicates that the service is running stable and no invocation failure has happened, for these requests. Service May Recover (SMR): This status indicates that the service is not currently available, but chances are there to recover it,

because this unavailability is not due to failure but it is due to incompliance of QoS metrics by the controller. Service May Not Recover (SMNR): This status indicates that the service is not currently available, but chances are less for recovery.

Service Not Available (SNA): this status indicates that service is down due to a specified reason. In this approach the metrics computation is based on invocation of records, the model is simple, and in this model the short term down is further divided in two sub categories SMR and SMNR.

Just as inherent reliability can impact operational performance, maintainability metrics can also have a large impact. These metrics can include Mean Time to Repair (MTTR), Mean Time to Fault Isolate, Mean Administrative Logistics Delay Time (MALDT), and wait times. Maintainability issues can be addressed in a fielded system, whereas inherent reliability is typically a design function and subject to engineering improvements only in extreme cases of substandard performance. Maintainability can be improved by increasing maintenance resources such as manpower, spares, and repair locations and by improving the maintenance concept and maintenance decisions. Unlike design and production efforts to improve inherent reliability, each of these comes at a significant annual recurring cost.

The metric estimation - it's a three step approach [15]:

- 1) Calculate the success percentage for each sequence.
- 2) Calculate the weighted average of success rates for status SMR and SMNR.
- 3) Calculate the time percentage for each status.

18.2 Markov's modeling of IoT systems reliability and availability

When designing the Internet of things system, it is necessary to consider and ensure its reliability and cyber security. To assess the reliability indicators, the section discusses the features of the application of the Markov models mathematical apparatus. Chapter consists of such topics:

- 1) Technique.
- 2) Development of models.
- 3) Research of models.

18.2.1 Technique

Graphical Markov models provide a method of representing possibly complicated multivariate dependencies in such a way that the general qualitative features can be understood, that statistical independencies are highlighted, and that some properties can be derived directly. Variables are represented by the nodes of a graph. Pairs of nodes may be joined by an edge. Edges are directed if one variable is a response to the other variable considered as explanatory, but are undirected if the variables are on an equal footing. Absence of an edge typically implies statistical independence, conditional, or marginal depending on the kind of graph. The need for a number of types of graph arises because it is helpful to represent a number of different kinds of dependence structures. Of special importance are chain graphs in which variables are arranged in a sequence or chain of blocks, the variables in any one block being on an equal footing, some being possibly joint responses to variables in the past and some being jointly explanatory to variables in the future of the block considered. Some main properties of such systems are outlined, and recent research results are sketched. Suggestions for further reading are given. As an illustrative example, some analysis of data on the treatment of chronic pain is presented.

Types of Markov models [2]:

1) Homogeneous CTMCs (Fig. 18.3).

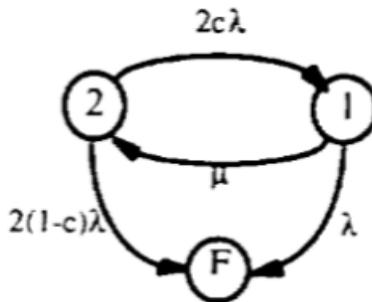


Fig. 18.3 – Homogeneous Markov model

- Simplest, most commonly used.
 - Markov property always holds.
 - Transition rates are constant.
 - State holding times are exponentially distributed.
 - "Memoryless Property" - time to next transition is not influenced by the time already spent in the state.
- 2) Non-homogeneous CTMC (Fig. 18.4).

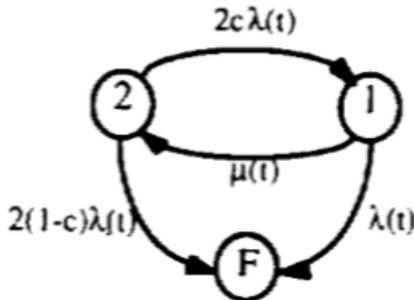


Fig. 18.4 – Non-homogeneous Markov model

- more complex;
- Markov property always holds;
- transition rates are generalized to be functions of time - dependent on a "global clock".

The control system example may again be used to illustrate the difference between a semi-Markov model and the previous two types of Markov models. Assume that the failure rate of a processor is again constant. Now, however, assume that the repair duration is a function of the time $f(t)$ that the processor has been under repair. The state-transition diagram for the resulting semi-Markov model is shown in the slide. It is identical to that for the homogeneous CTMC case except that the repair transition rate is a function of the time z that the processor has been under repair (i.e. the time that the system has been in state [3-7]). Semi-Markov models require the most computational effort of all the Markov model types to solve. They are often produced when detailed fault/error handling is included in a Markov model. This is the case because non-constant transitions between states that model fault handling often depend on the time elapsed since the fault occurred and handling/recovery commenced rather than on the elapsed mission time.

18.2.2 Development of models

Based on the analysis of standard solutions for the implementation of IoT system is proposed the wired architecture of the network. Using for IoT system Internet wire network devices are: router with Ethernet-ports and wireless access ability, softswitch the second layer, firewall, power block, server with control software, IP-camera, sensors, cables [8-13]. The system can operate as a standalone or with Internet connection.

Assumptions for the developed Markov model of IoT system availability are the following [16]:

- the flow of hardware system failures obeys the Poisson distribution law;

- there is reserve of the server and the router;

- failures caused by software design faults of IoT system subsystems obeys Poisson distribution, as on the results of monitoring and diagnostics, testing corrected secondary error (the result of the accumulation of the effects of primary errors and defects, software backdoors) to fix a malfunction or failure of the software, remove of impacts on software vulnerabilities, DoS- and DDoS-attacks, the number of primary defects in the software permanently;

- the process, which occurs in the system, it is a process without aftereffect, every time in the future behavior of the system depends only on the state of the system at this time and does not depend on how the system arrived at that state.

Therefore, the process has the Markov property. The mode of the server when software system shutdown and startup cycles in this model S4 is absent, because in this mode it is impossible to manage the server remotely.

A Markov model of IoT system subsystems functioning represented on fig.18.2.3, considering DDoS-attacks and energy modes of server and router, which has the following states [17-24]: good-working state (1); the server is fully used with high power consumption state (2); the server is fully used, the hardware, that are not used, can enter the low-power mode S1 (3); sleep mode of the server with low power consumption, a computer can wake up from a keyboard input, a LAN network or USB device S2 (4); server appears off, power consumption is reduced to the lowest level S3 (5); server failure (6);

switching to the backup server device after the server failure (7); restarting the server software after the software fault (8); successful DDoS-attack on the server after the firewall failure (9); firewall software or hardware failure (10); attack on the power supply system after the firewall failure, that lead the failure of general power system of IoT system (11); technical condition of switch from the general power system after its failure on the alternative energy sources (solar, diesel generator, wind turbine) (12); router status active - sending packages with high power consumption (13); DDoS- successful attack on the router (14); good-working state of the router without transmitting packets - Normal Idle (15); good-working state of the router without packet transmission Low-Power Idle (16); router software or hardware failure (17); server software or hardware fault (18); router hardware or software fail (20); switching to the backup router device after the router failure (21); restarting the router software after the router software fault (22).

A system of linear differential equations of the Kolmogorov-Chapman composed and solved in the paper with the initial conditions:

$$\sum_{i=1}^{22} P_i(t) = 1, i = 1 \dots 22, P_i(0) = 1. \quad (18.2)$$

An important indicator of IoT system dependability under the influence of different kinds of DDoS-attacks is the availability factor. As an index of IoT system reliability we choose availability function $AC(t)$, that is defined as the sum of the probabilities of staying the system in an up-states. Availability function $AC(t)$ is determined from equation:

$$AC(t) = P_1(t) + P_2(t) + P_3(t) + P_4(t) + P_5(t) + P_{12}(t) + P_{13}(t) + P_{15}(t) + P_{16}(t), \quad (18.3)$$

where $P_i(t)$ – probability of good condition IoT system components.

Solving the system of Kolmogorov-Chapman equations, we can get the value of the availability function components and IoT system after successful DDoS-attacks and with considering energy modes of the server and the router. It follows that service availability, service continuity, cyber security, data integrity, resilience and high

dependability of software and hardware should be inherent in IoT networks.

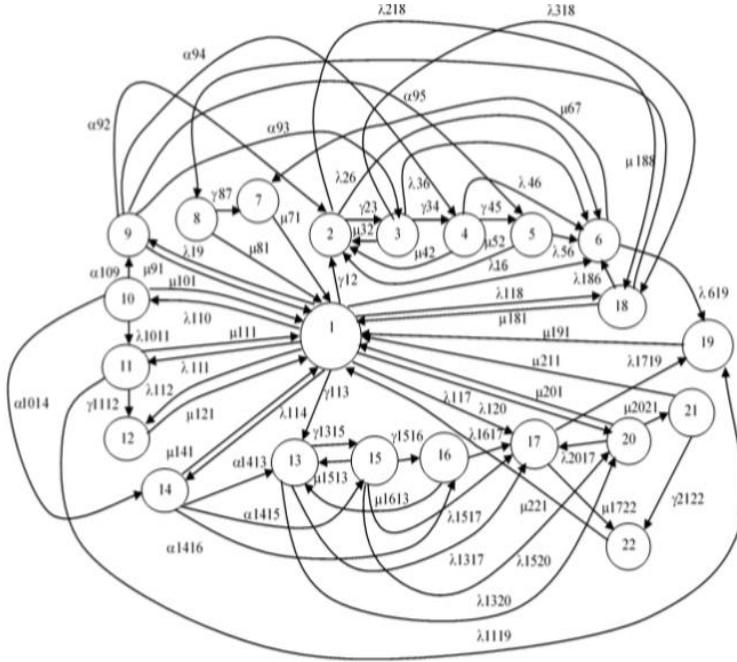


Fig. 18.5 – A Markov model of IoT system's general subsystems functioning

18.2.3 Research of models

On the basis of the analysis of statistical data we assess the main indicators of dependability - AC and built a graph shown in Fig. 18.2.4-18.2.6. As an example, we give graphical dependencies for different technical states of the server. We constructed the dependence of the system availability function (we denote it AC) from the transitions rates to different states (λ_{ij} , α_{ij} , γ_{ij} , where $i = 1 \dots 22$ $j = 1 \dots 22$), which depend on events occurrence time. The analysis of the Markov's model simulation results shows decreases the value of SBC availability function AC with increase of:

- the transition rate λ_{218} from an active-power mode of the server 2 to a state of the server fail 18 (Fig. 18.8);
- the transition rate λ_{1317} from active-power mode of the router 13 to a state of the router failure 17 (Fig. 18.6);
- the transition rate λ_{26} from server's active-power mode 2 to a state of the server failure 6 and the transition rate λ_{36} from server's low-power mode 3 to a state of the server failure 6 (Fig. 18.7).

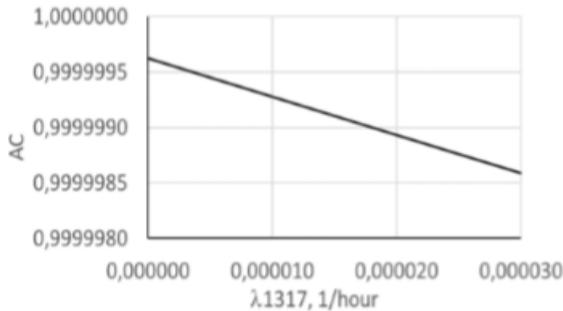


Fig. 18.6 – Graph of dependence of SBC AC on the transition rate λ_{1317} from active power state of the router 13 to a state of the router failure 17

Increase the transition rate from a good state of a server with full power consumption 2 to a server failure state 6 (λ_{26}); from a good state of a server with a reduced power consumption 3, to the server's failure state 6 (λ_{36}) results to AC decrease. With an increase of the transition rate from a good state 1 to a state with full power consumption 2 (λ_{12}), increase the nominal value of AC(t).

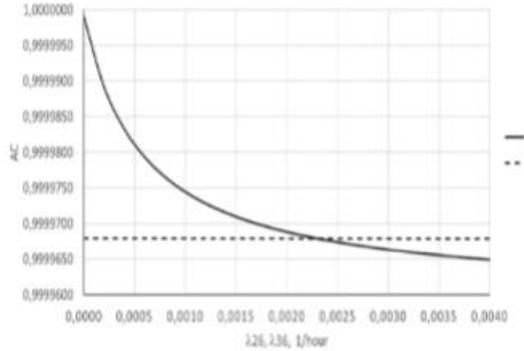


Fig. 18.7 – Graph of dependence of SBC AC on the transition rate λ_{26} from active power state of the server 2 to a state of the server failure 6 and the transition rate λ_{36} from server's low-power mode 3 to a state of the server failure 6 if $\lambda_{12}=30$ 1/hour; $\mu_{61}=0,02083$ 1/hour; $\mu_{67}=60$ 1/hour; $\mu_{71}=20$ 1/hour

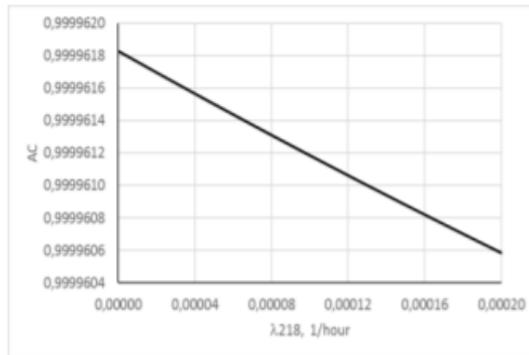


Fig. 18.8 – Graph of dependence of SBC AC on the transition rate λ_{218} from active power state of the server 2 to a state of the server fail 18

Moreover, at a high rate of the transition from the failure state of the server 6 to the working state 1 (μ_{61}), and also to the reconfiguration state 7 (μ_{67}), a smoother change in the availability function is observed than values of μ_{61} , μ_{67} are low. Moreover, at a high transition rate from the server failure state 6 to the working state 1 (μ_{61}), and also to the reconfiguration state 7 (μ_{67}), a smoother change in the availability

function is observed than at low values of μ_{61} , μ_{67} . With the transitions rates $\lambda_{12}=30$ 1/hour; $\mu_{61}=0,02083$ 1/hour; $\mu_{67}=60$ 1/hour; $\mu_{71}=20$ 1/hour – the value of AC with $\lambda_{26}=0,004$ 1/hour is about equal to 0,9999340. If $\lambda_{12}=100000$ 1/hour; $\mu_{61}=20$ 1/hour; $\mu_{67}=1000$ 1/hour; $\mu_{71}=50$ 1/hour availability function value with $\lambda_{26}=0,004$ 1/hour is equal to 0,9999650. Therefore, it is necessary to choose such values of SBC parameters at which the availability factor of the proposed system for any changes in parameters taking into account the power consumption modes and under states of DoS- and DDoS-attacks will not change significantly. Reducing the availability function when increasing the transition rate from a good state with a high power consumption of the server into a software fail mode occurs due to the impact of external influences (DoS- and DDoS-attacks), and because of internal causes associated with defects in the software and/or hardware of the server. The initial value of the AC is less than 1 when the transition rate from state 9 to state 2 (λ_{92}) changes (by the DoS- and DDoS-attacks influence on the state of the server with high power consumption if there is a vulnerability in the server firewall), because the AC is influenced both by external influences (attack), and internal causes (defects of software and/or hardware). With the increase in the attack flow to the server through the firewall vulnerability, it is perceived as a simple increase in the flow of data to the server, which leads to the server's transition into a good state of high energy consumption. With a further increase in λ_{92} , the change in AC function. Under the influence of DDoS-attacks, the server, which is in one of the energy saving modes, will switch to the mode of increased power consumption. The practical significance of the results is the following. They allow to assess the availability factor and to develop recommendations for the IoT system design for reduce the vulnerabilities of the software.

18.3 Markov's modeling of IoT systems cyber security and availability

18.3.1 Technique

When organizing IoT system, it is necessary to take into account the security, reliability of software and hardware of its components and their energy consumption modes [16-24]. The Markov model proposed

in [16], describes the process of IoT system functioning taking into account the attack on the system and the various power modes of the server and the router. Assumptions taken to construct and research the advanced IoT system availability model assume a Poisson flow of failure distribution of hardware and software of the IoT system components and allow the apparatus of Markov random processes to be used to estimate its availability. The means of control and diagnostics, as well as the means of switching to backup units, are considered ideal (they correctly identify the failed units and perform the switching to serviceable ones).

18.3.2 Development of Markov model of IoT system functioning

The description of the states of the improved model (Fig. 18.9) is similar to [16]. The improved Markov model of IoT system availability presented in Fig. 18.9 [25] takes into account the possibility of successful attacks on the router, the transition of the server and the router to different energy modes, and the installation of software patches on the vulnerability of the router's firewall without new vulnerabilities.

For the Markov model, systems of Kolmogorov-Chapman equations with initial conditions [19]:

$$PI(0) = 1. \tag{18.4}$$

The sum of the probabilities of finding the system in each of the states is 1.

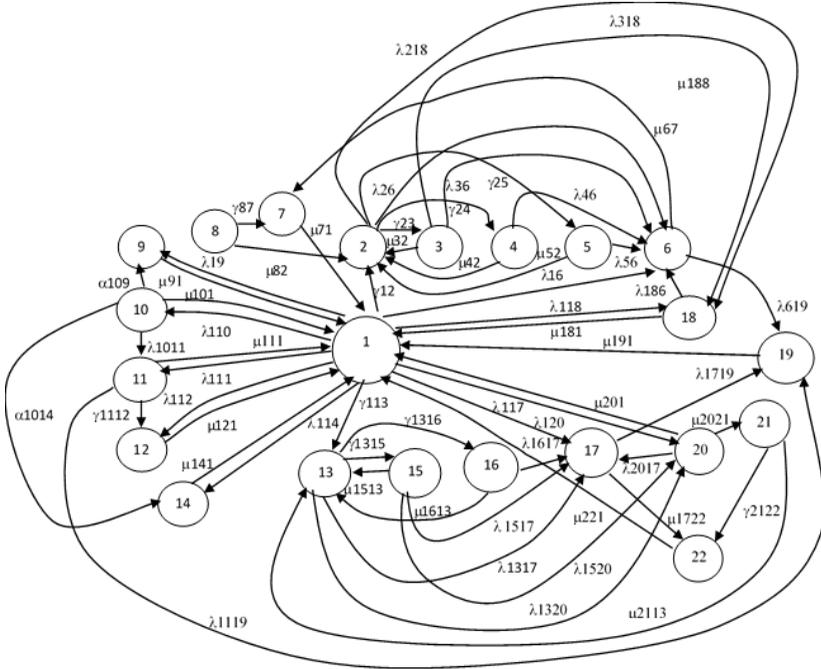


Fig. 18.9 – A graph of a Markov model of SBC systems functioning when installed patches on the router firewall and server firewall

To assess the IoT system availability with the conditions of external factors, such as various attacks on the router, the availability factor AC was chosen, the value of which for IoT system is defined as the sum of the probabilities of such systems being in good working states:

$$AC = P1(t)+P2(t)+P3(t)+P4(t)+P5(t)+P12(t)+ P13(t)+P15(t)+ P16(t)+P21(t). \tag{18.5}$$

$Pi(t)$ – probability of operable IoT system states.

18.3.3 Research of models

Fig. 18.10 shows the graphical dependence of the ΔAC with a change in the rate transition 1317, which is determined by the difference in the values of AC IoT system using a patch on the router's firewall and the AC IoT system without a patch. Fig. 18.11 shows the graphical dependencies of $\Delta AC1$, determined by the difference in the values of AC IoT system if λ_{1517} changing with installation of patch on router firewall software and AC IoT system without a patch, and $\Delta AC2$, determined by the difference in the values of IoT system's AC if λ_{1617} changing with installation of patch on router firewall software and IoT system's AC without a patch (Fig. 18.12). Analysis of changes in IoT system's AC when installing a patch on the vulnerabilities of the router software firewall showed: the router's transition rate from the Active state (13) to the router's failure state (17) varies slightly $\Delta AC=6 \cdot 10^{-9}$ 1/hour (Fig. 18.10, Fig. 18.11). This is explain by loading the router if impacts an attack without a software firewall patch at low transition rates $\lambda_{1317}=0 \dots 2 \cdot 10^{-6}$ 1/hour is close to loading the router in the active mode if there is a patch, since attacks impacts on the router gradually, first simulating the active mode of the router; the transition rate from the Normal (15) state to the router's failure state (17) varies from $\Delta AC1=2,90525 \cdot 10^{-5} \dots 2,90448 \cdot 10^{-5}$ if $\lambda_{1517} = 0 \dots 0,0001$ 1/hour (fig. 18.3.5); the transition rate from the Low (16) state to the router failure state (17) varies from $\Delta AC2=2,9048 \cdot 10^{-5} \dots 2,9027 \cdot 10^{-5}$ if $\lambda_{1617}=0 \dots 0,0001$ 1/hour (fig. 18.13).

The research showed that the timely establishment of a patch on the vulnerability of the router's firewall makes it possible to increase the value of IoT system's AC.

To assess IoT system availability, the Markov model was improved and researched, taking into account the impact of successful cyber-attacks on the IoT system, failures and fails of hardware and software components IoT system, the transition of the router in modes of reduced power consumption, patching the vulnerabilities of the router firewall software.

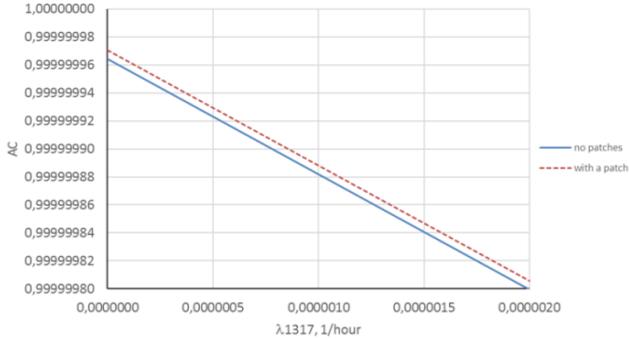


Fig. 18.10 – Dependencies of IoT system's AC from the transition rate λ_{1317} with the installation of patches on the router firewall and without patches

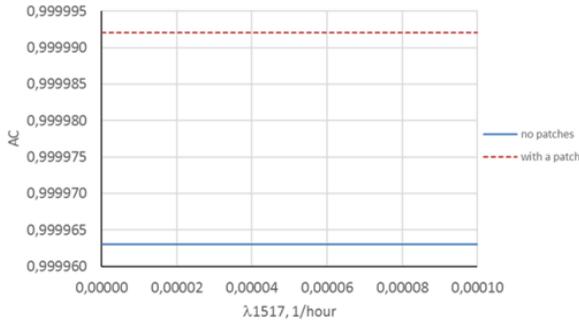


Fig. 18.11 – Dependencies of IoT system's AC from the transition rate λ_{1517} with the installation of patches on the router firewall and without patches

The study showed that installing the patch on the router's firewall when it is operating in the Active mode has little effect on changing of the IoT system AC with the patch installed. When the router is operating in low power mode, the IoT system availability increases by an order of magnitude when installing the patch on the router firewall software vulnerabilities, compared to the IoT system AC value without a patch.

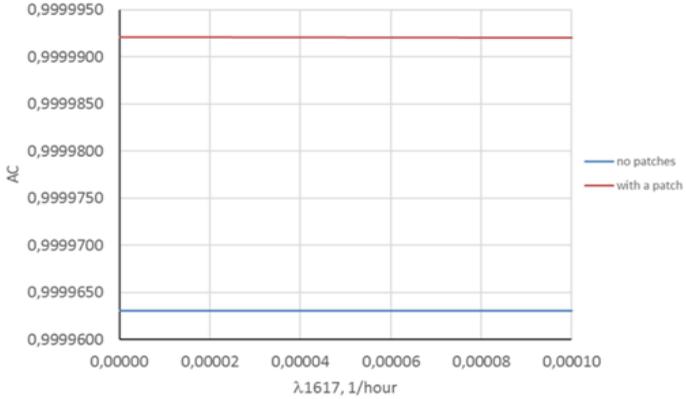


Fig. 18.12 – Dependencies of IoT system's AC from the transition rate $\lambda.1617$ from LP_IDLE state in the failure state with the installation of patches on the router firewall and without patches

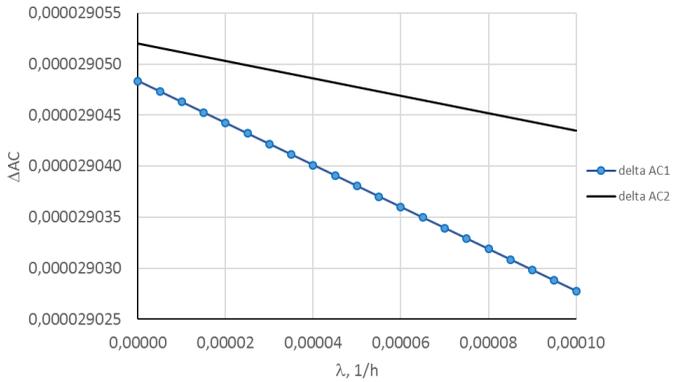


Fig. 18.13 – Dependencies of IoT system's $\Delta AC1$ and $\Delta AC2$

18.4 Semi Markov's modeling of IoT systems

18.4.1 Technique

The semi-Markov processes were introduced independently and almost simultaneously by P. Levy, W. L. Smith and L. Takacs in 1954–1955. The essential developments of semi-Markov processes theory were proposed by R.Pyke, E. Cinlar, Koroluk, Turbin, N. Limnios and G. Oprisan, D. C. Silvestrov. We present only semi-Markov processes with a discrete state space. A semi-Markov process (Fig.18.14) is constructed by the Markov renewal process which is defined by the renewal kernel and the initial distribution or by another characteristics which are equivalent to the renewal kernel [2-7].

Suppose that $\mathbb{N} = \{1,2,\dots\}$, $\mathbb{N}_0 = \{0,1,2,\dots\}$, $\mathbb{R}_+ = [0,\infty)$ and S is a discrete (finite or countable) state space. Let ξ_n be a discrete random variable taking values on S and let ϑ_n be a continuous random variable with values in the set \mathbb{R}_+ .

Definition 1. A two-dimensional sequence of random variables $\{(\xi_n, \vartheta_n): n \in \mathbb{N}_0\}$ is said to be a Markov Renewal Process (MRP) if: 1) for all $n \in \mathbb{N}_0$, $j \in S$, $t \in \mathbb{R}_+$ $P(\xi_{n+1} = j, \vartheta_{n+1} \leq t \mid \xi_n = i, \vartheta_n, \dots, \xi_0, \vartheta_0) = P(\xi_{n+1} = j, \vartheta_{n+1} \leq t \mid \xi_n = i)$ (1) with probability 1; 2) for all $i, j \in S$, $P(\xi_0 = i, \vartheta_0 = 0) = P(\xi_0 = i)$. (2) From the definition 1 it follows, that MRP is a homogeneous two-dimensional Markov chain such that its transition probabilities depend only on the discrete component (they do not depend on the second component). A matrix $Q(t) = [Q_{ij}(t): i, j \in S]$; (3) $Q_{ij}(t) = P(\xi_{n+1} = j, \vartheta_{n+1} \leq t \mid \xi_n = i)$ is called a renewal matrix. A vector $p = [p_i: i \in S]$, where $p_i = P\{\xi_0 = i\}$ defines an initial distribution of the Markov renewal process. It follows from the definition 1 that the Markov renewal matrix satisfies the following conditions:

1. The functions $Q_{ij}(t)$, $t \geq 0$, $(i, j) \in S \times S$ are not decreasing and right-hand continuous.
2. For each pair $(i, j) \in S \times S$, $Q_{ij}(0) = 0$ and $Q_{ij}(t) \leq 1$ for $t \in \mathbb{R}_+$.
3. For each $i \in S$, $\lim_{t \rightarrow \infty} \sum_{j \in S} Q_{ij}(t) = 1$. One can prove that a function matrix $Q(t) = [Q_{ij}(t): i, j \in S]$ satisfying the above mentioned conditions and a vector $p_0 = [p_i(0): i \in S]$ such that $\sum_{i \in S} p_i(0) = 1$ define some Markov renewal process. From definition of the renewal matrix it follows that $Q = [Q_{ij}(t): i, j \in S]$, $p_{ij} = \lim_{t \rightarrow \infty} Q_{ij}(t)$ (4) is a

stochastic matrix. It means that for each pair $(i, j) \in S \times S$ $p_{ij} \geq 0$ and for each $i \in S$, $\sum_{j \in S} p_{ij} = 1$.

It is easy to notice that for each $i \in S$

$$Q_{ij}(t) = \sum_{j \in S} Q_{ij}(t) \quad (18.6)$$

is a probability cumulative distribution function (CDF) on \mathbb{R}_+ . The definition 1 leads to the interesting and important conclusions $(\vartheta_0 = 0) = 1$. For a Markov Renewal Process with an initial distribution p_0 and a renewal kernel $Q(t)$, $t \geq 0$ a following equality is satisfied

$$P(\xi_0 = i_0, \xi_1 = i_1, \vartheta_1 \leq t_1, \dots, \xi_n = i_n, \vartheta_n \leq t_n) = p_{i_0} Q_{i_0 i_1}(t_1) Q_{i_1 i_2}(t_2) \dots Q_{i_{n-1} i_n}(t_n). \quad (18.7)$$

For $t_1 \rightarrow \infty, \dots, t_n \rightarrow \infty$, we obtain

$$P(\xi_0 = i_0, \xi_1 = i_1, \dots, \xi_n = i_n) = p_{i_0} p_{i_0 i_1} p_{i_1 i_2} \dots p_{i_{n-1} i_n}. \quad (18.8)$$

It means that a sequence $\{\xi_n: n \in \mathbb{N}_0\}$ is a homogeneous Markov chain with the discrete state space S , defined by the initial distribution $p = [p_{i_0}: i_0 \in S]$ and the transition matrix

$$P = [P_{ij}: i, j \in S], \text{ where } p_{ij} = \lim_{t \rightarrow \infty} Q_{ij}(t). \quad (18.9)$$

The random variables $\vartheta_1, \dots, \vartheta_n$ are conditionally independent if a trajectory of the Markov chain $\{\xi_n: n \in \mathbb{N}_0\}$ is given. It means that

$$P(\vartheta_1 \leq t_1, \vartheta_2 \leq t_2, \dots, \vartheta_n \leq t_n \mid \xi_0 = i_0, \xi_1 = i_1, \dots, \xi_n = i_n) = \prod_{k=1}^n P(\vartheta_k \leq t_k \mid \xi_k = i_k, \xi_{k-1} = i_{k-1}). \quad (18.10)$$

The Markov renewal matrix $G(t) = [G_{ij}(t): i, j \in S]$ is called continuous if each row of the matrix contains at least one element having continuous component in the Lebesgue decomposition of the probability distribution. The matrix $G(t) = [G_{ij}(t): i, j \in S]$ with elements $G_{ij}(t) = p_{ij} G_{ij}(t)$, $i, j \in S$, where

$$G_{ij}(t) = c [1, \infty)(t) + (1 - c) \int_0^t h_{ij}(u) du, \quad c \in (0, 1), \quad p_{ij} \geq 0,$$

$$\sum_{j \in S} p_{ij} = 1$$

and $h_i(\cdot)$ is a continuous probability density function, is an example of the continuous Markov renewal matrix.

The Markov renewal matrix $Q(t) = [Q_{ij}(t): i, j \in S]$ with elements $Q_{ij}(t) = p_{ij}I_{[1, \infty)}(t)$, $i \in S$, where $p_{ij} \geq 0$, $\sum_{j \in S} p_{ij} = 1$ is not continuous Markov renewal matrix. Moreover, in the whole paper we will assume that the Markov renewal matrix $(t) = [(t): i, j \in S]$ is continuous. Let $0 = \vartheta_0$, $\tau_n = \vartheta_1 + \vartheta_2 + \dots + \vartheta_n$, $n \in \mathbb{N}_0$, (10) $\tau_\infty = \lim_{n \rightarrow \infty} \tau_n = \sup\{\tau_n: n \in \mathbb{N}_0\}$. The sequence $\{(\xi_n, \tau_n): n \in \mathbb{N}_0\}$ is two-dimensional Markov chain with transition probabilities $P(\xi_{n+1} = j, \tau_{n+1} \leq t \mid \xi_n = i, \tau_n = h) = Q_{ij}(t - h)$, $i, j \in S$ (11) and it is also called Markov Renewal Process (MRP) Koroluk.

Hidden Markov Models (HMMs) basically represent Bayesian networks triggered to collect contextual information based on scanty approach data in order to recognize possible threats or conducts that may turn abusive. HMMs in other words can be described as a doubly stochastic embedded network for identifying threats based on visual appearances and verbal conversations (Fig. 18.1).

Predictive models that are able to estimate the current condition and the Remaining Useful Lifetime of an industrial equipment are of high interest, especially for manufacturing companies, which can optimize their maintenance strategies.

If we consider that the costs derived from maintenance are one of the largest parts of the operational costs and that often the maintenance and operations departments comprise about 30% of the manpower, it is not difficult to estimate the economic advantages that such innovative techniques can bring to industry. Moreover, predictive maintenance, where in real time the Remaining Useful Lifetime (RUL) of the machine is calculated, has been proven to significantly outperforms other maintenance strategies, such as corrective maintenance. In this work, RUL is defined as the time, from the current moment, that the systems will fail. Failure, in this context, is defined as a deviation of the delivered output of a machine from the specified service requirements that necessitate maintenance [3-7].

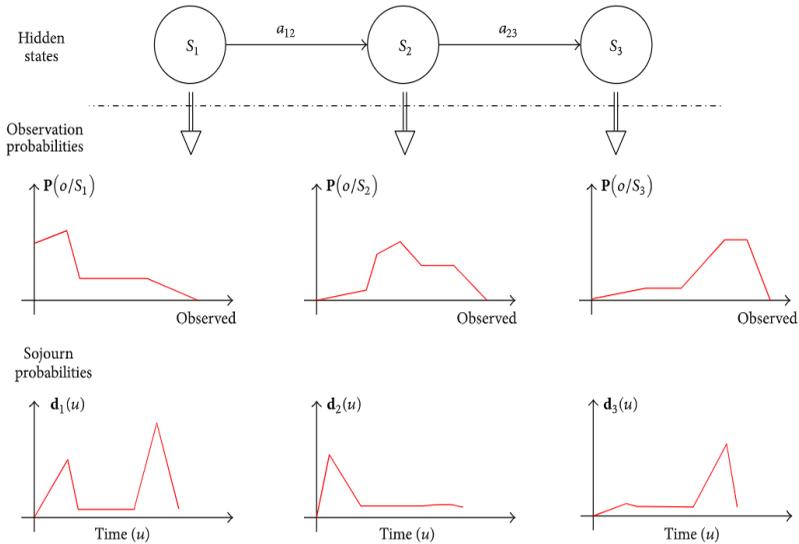


Fig. 18.14 – Hidden Semi-Markov model

Models like Support Vector Machines, Dynamic Bayesian Networks, clustering techniques, and data mining approaches have been successfully applied to condition monitoring, RUL estimation, and predictive maintenance problems. State space models, like Hidden Markov Models (HMMs), are particularly suitable to be used in industrial applications, due to their ability to model the latent state which represents the health condition of the machine.

Classical HMMs have been applied to condition assessment; however, their usage in predictive maintenance has not been effective due to their intrinsic modeling of the state duration as a geometric distribution.

To overcome this drawback, a modified version of HMM, which takes into account an estimate of the duration in each state, has been proposed in the works of Tobon-Mejia et al. Thanks to the explicit state sojourn time modeling, it has been shown that it is possible to effectively estimate the RUL for industrial equipment. However, the drawback of their proposed HMM model is that the state duration is

always assumed as Gaussian distributed and the duration parameters are estimated empirically from the Viterbi path of the HMM.

A complete specification of a duration model together with a set of learning and inference algorithms has been given firstly by Ferguson. In his work, Ferguson allowed the underlying stochastic process of the state to be a semi-Markov chain, instead of a simple Markov chain of a HMM. Such model is referred to as Hidden Semi-Markov Model (HSMM). HSMMs and explicit duration models have been proven beneficial for many applications. A complete overview of different duration model classes has been made by Yu [3]. Most state duration models, used in the literature, are nonparametric discrete distributions [4-7]. As a consequence, the number of parameters that describe the model and that have to be estimated is high, and consequently the learning procedure can be computationally expensive for real complex applications. Moreover, it is necessary to specify a priori the maximum duration allowed in each state.

To alleviate the high dimensionality of the parameter space, parametric duration models have been proposed. For example, Salfner proposed a generic parametric continuous distribution to model the state sojourn time. However, in their model, the observation has been assumed to be discrete and applied to recognize failure-prone observation sequence. Using continuous observation, Azimi et al. specified an HSMM with parametric duration distribution belonging to the Gamma family and modeled the observation process by a Gaussian.

Inspired by the latter two approaches, in this work we propose a generic specification of a parametric HSMM, in which no constraints are made on the model of the state duration and on the observation processes. In our approach, the state duration is modeled as a generic parametric density function. On the other hand, the observations can be modeled either as a discrete stochastic process or as continuous mixture of Gaussians. The latter has been shown to approximate, arbitrarily closely, any finite, continuous density function. The proposed model can be generally used in a wide range of applications and types of data. Moreover, in this paper we introduce a new and more effective estimator of the time spent by the system in a determinate state prior to the current time. To the best of our knowledge, a part from the above referred works, the literature on HSMMs applied to prognosis and predictive maintenance for industrial machines is limited. Hence, the

present work aims to show the effectiveness of the proposed duration model in solving condition monitoring and RUL estimation problems.

Dealing with state space models, and in particular of HSMMs, one should define the number of states and correct family of duration density, and in case of continuous observations, the adequate number of Gaussian mixtures. Such parameters play a prominent role, since the right model configuration is essential to enable an accurate modeling of the dynamic pattern and the covariance structure of the observed time series. The estimation of a satisfactory model configuration is referred to as model selection in literature.

While several state-of-the-art approaches use expert knowledge to get insight on the model structure an automated methodology for model selection is often required. In the literature, model selection has been deeply studied for a wide range of models [2-7]. Among the existing methodologies, information based techniques have been extensively analyzed in literature with satisfactory results. Although Bayesian Information Criterion (BIC) is particularly appropriate to be used in finite mixture models, Akaike Information Criterion (AIC) has been demonstrated to outperform BIC when applied to more complex models and when the sample size is limited, which is the case of the target application of this paper.

In this work AIC is used to estimate the correct model configuration, with the final goal of an automated HSMMs model selection, which exploits only the information available in the input data. While model selection techniques have been extensively used in the framework of Hidden Markov Models, to the best of our knowledge, the present work is the first that proposes their appliance to duration models and in particular to HSMMs [4-7].

In summary, the present work contributes to condition monitoring, predictive maintenance, and RUL estimation problems by (i) proposing a general Hidden Semi-Markov Model applicable for continuous or discrete observations and with no constraints on the density function used to model the state duration; (ii) proposing a more effective estimator of the state duration variable, that is, the time spent by the system in the next state, prior to current time; (iii) adapting the learning, inference and prediction algorithms considering the defined HSMM parameters and the proposed estimator; (iv) using the Akaike Information Criterion for automatic model selection.

Hidden Semi-Markov Models (HSMMs) introduce the concept of variable duration, which results in a more accurate modeling power if the system being modeled shows a dependence on time.

In this section we give the specification of the proposed HSMM, for which we model the state duration with a parametric state-dependent distribution. Compared to nonparametric modeling, this approach has two main advantages: (i) the model is specified by a limited number of parameters; as a consequence, the learning procedure is computationally less expensive; (ii) the model does not require the a priori knowledge of the maximum sojourn time allowed in each state, being inherently learnt through the duration distribution parameters.

A Hidden Semi-Markov Model is a doubly embedded stochastic model with an underlying stochastic process that is not observable (hidden) but can only be observed through another set of stochastic processes that produce the sequence of observations. HSMM allows the underlying process to be a semi-Markov chain with a variable duration or sojourn time for each state. The key concept of HSMMs is that the semi-Markov property holds for this model: while in HMMs the Markov property implies that the value of the hidden state at time depends exclusively on its value of time, in HSMMs the probability of transition from state to state at time depends on the duration spent in state prior to time.

18.5 Work related analysis

Nowadays there are many projects, which describes and researches IoT systems.

Smart Vehicle features shown by the Volvo Car Group company, working on new car projects - exchange of information about the dangers on the road through the "cloud" and control of the driver's state [17].

Toyota Motor Corp. and Panasonic jointly develop a service that will connect cars and home appliances through the IoT [18].

The project PRORETA [19] is a research in the area of the cooperative HMIs. The research object is the prototype of the cooperative automobile HMI that implements the scenarios of preventing collisions at the cross-roads.

The PRORETA HMI system implements a huge number of use scenarios, it does not complicate or irritate and ensures the multimode support.

The HMI provides 4 support levels – information messages, warnings, actions recommendations, automatic intervention.

A lot of EU universities including ALIOT project partners conduct research and implement education MSc and PhD programs in the Internet of Things application for transport and other domains. Development of cooperative HMI for cloud and IoT systems based on analysis of these programs and providing some of the educational topics and research directions.

In particular, the following courses and programs have been considered:

- Coimbra University, Portugal: IoT course for MSc [20]. The courses represents a new stage in the digital evolution and focuses on the Internet of Things for smart transport and cities, and the development of tools to transform city infrastructure;

- KTH University, Sweden: three MSc programs including:

- a) IoT related topics in Information and Network Engineering [21],

- b) Communication Systems [22],

- c) Embedded Systems [23];

- Newcastle University, United Kingdom: MSc Program on Embedded Systems and Internet of Things (ES-IoT) MSc [24].

Conclusions and questions

The section presents an analysis of Markov and semi-Markov models of the Internet of things systems functioning, conducted their study from the point of view of reliability and cyber security.

The research of Markov models showed that the IoT system, even with the required high AC value, is highly dependent on the correct failure-free operation of the firewalls. Analysis of the graphical dependencies obtained for the developed models, taking into account the rearrangement in case of appearance and installation of the patch on the vulnerability of the firewall software, showed that AC SBC is most sensitive to patching the firewall software of the router and the network firewall. When the patch is set, the AC remains high (0.9999925), even with a transition rate to a failure state of 0.001 1/h. The hypothesis is

confirmed that the establishment of the patch significantly increases the AC value even at clearly high values of the transition rates to the failure state. The practical importance of the results allows to assess the SBC availability and to develop recommendations to reduce the vulnerability of its software from the impact of DDoS attacks, as well as reduce its power consumption.

Questions:

1. Define the requirements to the Markov process.
2. Explain the difference between Homogeneous and Non-Homogeneous Markov model.
3. Give the definition of semi-Markov process.
4. Give the classification of Markov models.
5. What are the basic availability metrics of the Internet of things?
6. Explain what is the difference between the functional graph of the Markov model and the state model of the Internet of things system?
7. Explain, please, the principle of constructing state graphs of Markov models.
8. Explain, please, principles of Markov's modeling.
9. Explain, please, principles of Semi Markov's modeling.

References

1. Margaret Rouse. Markov model. [<https://whatis.techtarget.com/definition/Markov-model>].
2. Charles M. Grinstead, J.Lauri Snell. Probability. The CHANCE Project1. Version dated 4 July 2006. Markov Chains. 518 p. [https://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/Chapter11.pdf].
3. Grabski F. Semi-Markov models of reliability and operation, IBS PAN, Warsaw, 2002 [in Polish].
4. Grabski F. Semi-Markov Processes: Applications in Systems Reliability and Maintenance, Elsevier, Amsterdam, Boston, Heidelberg, London, New York, Oxford, Paris, San Diego, San Francisco, Sydney, 2014.
5. Franciszek Grabski. Semi-markov reliability model of the cold standby system. Zeszyty Naukowe AMW — Scientific Journal of PNA. International Symposium on Stochastic Models in Reliability Engineering, Life Sciences and Operations Management (SMRLO'10).

6. Limnios N., Oprisan G., *Semi-Markov Processes and Reliability*, Birkhauser, Boston 2001.

7. Grabski Franciszek. Semi-Markov reliability model of system composed of main subsystem, cold backup component and switch. *Summer Safety and Reliability Seminars*, Vol. 8, Number 1, 2017, pp. 47-53.

8. Internet of Things. IoT Governance, Privacy and Security Issues. European Research Cluster on the Internet of Things. Ovidiu Vermesan, Peter Friess, Coordinators of IERC Cluster. January, 2015. 128 p.

9. Delivering on the IoT customer experience. Business white paper. Hewlett Packard Enterprise. Available at: [<http://h20195.www2.hp.com/v2/GetDocument.aspx?docname=4AA6-5128ENW>] (accepted at 5.08.2016). 8 p.

10. Internet of Things and its future. Available at: [http://www.huawei.com/ilink/en/about-huawei/newsroom/pressrelease/HW_080993?dInID=23407&relatedID=19881&relatedName=HW_076569&dInDocName=HW_076557] (access date: 20.11.2017)].

11. IETF Standardization in the Field of the Internet of Things (IoT): A Survey. Isam Ishaq, David Carels, Girum K. Teklemariam, Jeroen Hoebeke, Floris Van den Abeele, Eli De Poorter, Ingrid Moerman and Piet Demeester. *J. Sens. Actuator Netw.* 2013, 2, 235-287; doi:10.3390/jsan2020235. *Journal of Sensor and Actuator Networks* ISSN 2224-2708. Available at: [<http://www.mdpi.com/journal/jsan/>].

12. ISO/IEC 27000 family – Information security management systems. [<https://www.iso.org/isoiec-27001-informationsecurity.html>].

13. Internet Architecture Board (IAB). RFC 7452 “Architectural Considerations in Smart Object Networking”. [<https://www.rfceditor.org/pdf/rfc/rfc7452.txt.pdf>].

14. Kharchenko Vyacheslav, Kolisnyk Maryna, Piskachova Iryna. Reliability and Security Issues for IoT-Based Smart Business Center: Architecture and Markov Model. *IEEE; Computer of science, MCSI 2016, Greece, Chania, 2016*. Paper ID: 4564699.

15. Gerrod Andresen, Zachary Williams. Metrics, key performance indicators, and modeling of long range aircraft availability and readiness. *NATO, RTO-MP-AVT-144*. 12 p.

16. Maryna Kolisnyk, Iryna Piskachova, Vyacheslav Kharchenko. Patching the Firewall Software to Improve the Availability and Security: Markov Models for Internet of Things Based Smart Business Center. *CEUR-WS, Workshop Thermit 2018*, 13 p.

17. Lynn Walford, Volvo New Connected Car Features-Magnets, Real-Time Cloud Road Data & Driver Sensing [<http://www.autoconnectedcar.com>]

[/2014/03/volvo-new-connected-car-features-magnets-real-time-cloud-road-data-driver-sensing/](#), 2014.

18. Toyota and Panasonic develop cloud service to connect cars and household appliances [http://panasonic.ru/press_center/news/detail/464204], 2014.

19. Bauer, E. PRORETA 3: An Integrated Approach to Collision Avoidance and Vehicle Automation / E. Bauer, F. Lotz, M. Pfromm // At - Automatisierungstechnik. – 2012. – № 12. – P. 755-765.

20. Internet Of Things Course - Immersive Program Master in City and Technology [<https://apps.uc.pt/search?q=Internet+of+Things>].

21. Master's program in Information and Network Engineering [<https://www.kth.se/en/studies/master/information-and-network-engineering/master-s-programme-in-information-and-network-engineering-1.673817>]

22. Master's program in Communication Systems [<https://www.kth.se/en/studies/master/communication-systems/description-1.25691>]

23. Master's program in Embedded Systems [<https://www.kth.se/en/studies/master/embedded-systems/description-1.70455/>].

24. Related Programs to Embedded Systems and Internet of Things (ES-IoT) MSc [<https://www.ncl.ac.uk/postgraduate/courses/degrees/embedded-systems-internet-of-things-msc/relateddegrees.html>].

19. INTERACTION SIMULATION FOR IOT SYSTEMS

DrS. Prof. G.V. Tabunshchyk (ZNTU)

Contents

Abbreviations	111
19.1 Interaction in IoT systems	112
19.1.1 Introduction into infrastructure in the IoT systems	113
19.1.2 Patterns for designing interactions for IoT	116
19.2 Interaction Flow Modelling Language	118
19.3. Case Study	119
19.3.1 Usage of the Remote Laboratory GOLDi for interaction modelling.....	119
19.3.2 Simulation of the interaction for Smart Campus.	123
19.3.3 Interaction Simulation for e-Health systems.	124
19.3.4 Interaction Simulation for the Remote Laboratories	127
19.3.5 Interaction Simulation for Intelligent transport.	128
19.4 Work related analysis	131
Conclusions and questions.....	131
References	132

Abbreviations

BLE – Bluetooth Low Energy

BPMN - Business Process Model and Notation

GOLDi – Grid of Online Lab Devices

IoTIM – IoT Integration Middleware

IFML – Flow Modeling Language

ISRT - Interactive platform for Embedded
Software Development

RFID - Radio-frequency identification

SoaML - Service-oriented architecture Modeling Language

SysML - Systems Modeling Language

UI – User Interface

UML - Unified Modeling Language

The “Internet of Things” (IoT) refers to the growing range of everyday objects acquiring connectivity, sensing abilities, and increased computing power. In consumer terms, some common categories currently include [1,2]:

- connected home technology (such as thermostats, lighting, and energy monitoring);
- wearables medical/wellness devices (such as “smart” watches and blood pressure monitors);
- artificial intelligent implants;
- connected cars (which may provide access to onboard services, environment, car maintenance and connection to smart grid);
- urban systems (such as air quality sensors, city rental bikes, and parking meters/sensors).

Designing these systems raises challenges with the maturity of the technology you are working, complex use of user expectations of the system and the complexity of the services, provided by the system.

Implementation of the IoT technologies gave great impact on the types and ways of interactions. On the one side it influences greatly on the way how users interact with everything, on the other sides it changes the way of interactions inside on IoT systems. This chapter will be devoted to the simulation of the interactions between users and IoT systems.

19.1 Interaction in IoT systems

Approaches for designing of the IoT systems should unite data, interactions and the physical world. Interaction coupled with data transcends the laptop and mobile device, and it becomes literally embedded into any object, infrastructure or interaction.

For example in-store interactions simultaneously generating data might include entering the store; checking in on a mobile device; connecting to Wi-Fi or passing by a beacon; scanning an RFID tag, quick response code or other sensor indicating interest in an object or promotion; interacting with an employee stylist equipped with a tablet or other scanning device; trying on an item using a "smart mirror," in which one could search for various sizes, prints, colors or accessories; or even digitally overlay products on themselves using augmented reality. [2].

Purchases, redemption of coupons and digital receipts, among other interactions, can now all be integrated with a shopper's online profile, thereby connecting "brick" (in-store) and "click" (online) interactions. Such interactions can also signal inventory and supply chain transactions and even inform store layout, merchandizing, labor allocation and a host of other operational decisions, many of which are entirely invisible to the customer [**Error! Reference source not found.**].

19.1.1 Introduction into infrastructure in the IoT systems

In essence, IoT architecture is the system of numerous elements: sensors, protocols, actuators, cloud services, and layers.

In the simplest way the IoT architecture contains only three layers [3] :

1. The client side (IoT Device Layer)
2. Operators on the server side (IoT Getaway Layer)
3. A pathway for connecting clients and operators (IoT Platform Layer)

But the number if to include all elements which are included by IoT architecture [**Error! Reference source not found.**] the structure ill be much more difficult.

Let's consider the basic elements of IoT systems [6,7], which are main construction blocks of the IoT platforms.

Things. A "thing" is an object equipped with sensors that gather data which will be transferred over a network and actuators that allow things to act (for example, to switch on or off the light, to open or close a door, to increase or decrease engine rotation speed and more). This concept includes fridges, street lamps, buildings, vehicles, production machinery, rehabilitation equipment and everything else imaginable. Sensors are not in all cases physically attached to the things: sensors may need to monitor, for example, what happens in the closest environment to a thing.

Gateways. Data goes from things to the cloud and vice versa through the gateways. A gateway provides connectivity between things and the cloud part of the IoT solution, enables data preprocessing and filtering before moving it to the cloud (to reduce the volume of data for detailed processing and storing) and transmits control commands going

from the cloud to things. Things then execute commands using their actuators.

Cloud gateway facilitates data compression and secure data transmission between field gateways and cloud IoT servers. It also ensures compatibility with various protocols and communicates with field gateways using different protocols depending on what protocol is supported by gateways.

Streaming data processor ensures effective transition of input data to a data lake and control applications. No data can be occasionally lost or corrupted.

Data lake. A data lake is used for storing the data generated by connected devices in its natural format. Big data comes in "batches" or in "streams". When the data is needed for meaningful insights it's extracted from a data lake and loaded to a big data warehouse.

Big data warehouse. Filtered and preprocessed data needed for meaningful insights is extracted from a data lake to a big data warehouse. A big data warehouse contains only cleaned, structured and matched data (compared to a data lake which contains all sorts of data generated by sensors). Also, data warehouse stores context information about things and sensors (for example, where sensors are installed) and the commands control applications send to things.

Data analytics. Data analysts can use data from the big data warehouse to find trends and gain actionable insights. When analyzed (and in many cases – visualized in schemes, diagrams, infographics) big data show, for example, the performance of devices, help identify inefficiencies and work out the ways to improve an IoT system (make it more reliable, more customer-oriented). Also, the correlations and patterns found manually can further contribute to creating algorithms for control applications.

Machine learning and the models ML generates. With machine learning, there is an opportunity to create more precise and more efficient models for control applications. Models are regularly updated (for example, once in a week or once in a month) based on the historical data accumulated in a big data warehouse. When the applicability and efficiency of new models are tested and approved by data analysts, new models are used by control applications.

Control applications send automatic commands and alerts to actuators, for example:

Windows of a smart home can receive an automatic command to open or close depending on the forecasts taken from the weather service.

When sensors show that the soil is dry, watering systems get an automatic command to water plants.

Sensors help monitor the state of industrial equipment, and in case of a pre-failure situation, an IoT system generates and sends automatic notifications to field engineers.

The commands sent by control apps to actuators can be also additionally stored in a big data warehouse. This may help investigate problematic cases (for example, a control app sends commands, but they are not performed by actuators – then connectivity, gateways and actuators need to be checked). On the other side, storing commands from control apps may contribute to security, as an IoT system can identify that some commands are too strange or come in too big amounts which may evidence security breaches (as well as other problems which need investigation and corrective measures).

Control applications can be either rule-based or machine-learning based. In the first case, control apps work according to the rules stated by specialists. In the second case, control apps are using models which are regularly updated (once in a week, once in a month depending on the specifics of an IoT system) with the historical data stored in a big data warehouse.

Although control apps ensure better automation of an IoT system, there should be always an option for users to influence the behavior of such applications (for example, in cases of emergency or when it turns out that an IoT system is badly tuned to perform certain actions).

The IoT Integration Middleware (IoTIM) serves as an integration layer for different kinds of Sensors, Actuators, Devices, and Applications. It is responsible for receiving data from the connected Devices, processing the received data, providing the received data to connected Applications, and controlling Devices. An example for processing is to evaluate condition-action rules and sending commands to Actuators based on this evaluation.

User applications are a software component of an IoT system which enables the connection of users to an IoT system and gives the options to monitor and control their smart things (while they are

connected to a network of similar things, for example, homes or cars and controlled by a central system). With a mobile or web app, users can monitor the state of their things, send commands to control applications, set the options of automatic behavior (automatic notifications and actions when certain data comes from sensors).

In [4] there is done detailed analysis of the existing as open source as proprietary platforms such as are FIWARE, OpenMTC, SiteWhere, Webinos, AWS IoT2, IBM's Watson IoT Platform10, Microsoft's Azure IoT Hub11, and Samsung's SmartThings5.

And these research shows the great variety of existing solutions that's why the main criteria in the selection of the platform should be the user requirements to the interactions within the final systems.

19.1.2 Patterns for designing interactions for IoT

Design Patterns provide well known ways to solve design problems commonly encountered in a particular discipline or problem domain.

Interaction design is closely aligned to user interface (UI) design in the sense that the two are usually done in tandem and often by the same people. But interaction design is primarily concerned with behaviors and actions, whereas UI/visual design is concerned with layout and aesthetics. (Just to confuse matters, some people use UI design as a shorthand term to include both interaction design and visual design.) Typical outputs for interaction design might include user flows, low-medium fidelity interactive prototypes, and for a visual UI, screen wireframes.



Fig. 19.1 – Example of inter-usability[2]

IoT devices come in a wide variety of form factors with varying input and output capabilities. That's why it's important to consider not just the usability of individual UIs but interusability: distributed user experience across multiple devices [**Error! Reference source not found.**].

Design patterns for Application Programming describe ways that software and interfaces are created, managed, deployed, and used in IoT applications [**Error! Reference source not found.**]:

1. *REST Objects*: Mapping of REST API resources onto program objects in the application language, using libraries.

2. *Event handler*, *onEvent*: Application code that responds to asynchronous events.

3. *Event driven flow*: A set of application handlers that operate in an event driven graph containing series cascade and parallel constructs.

4. *State Machine*: A logic construct where a next state depends on a set of inputs and the current state, evaluated by a set of logic rules associated with each state.

5. *State Externalization*: The ability to create stateless application software by mapping application state onto external resources.

6. *Rule oriented programming*: Using a set of rules or rule language to program state machine logic.

7. *Abstraction of applications*: Stateless application software uses application templates for reusability.

8. *Application templates*: Abstract application components with well defined interfaces.

9. *Modular applications*: Applications consisting of one or more reusable components.

Applications run anywhere, location independent applications: Application components can run anywhere, in devices, on local network servers, in gateways, in edge servers, in cloud, on user devices.

Discovery and Linking: Integrates resources into applications by resolving resource links, sets attributes in application objects

Object Constructor: Creates application software objects from metadata models.

From the point of view of users we can define interactions
[**Error! Reference source not found.**]:

- configuration of the access and permissions;
- interaction with devices;
- managing devices;
- managing wait for signal;
- managing notifications;
- searching devices;
- storing information;
- retrieving stored information;
- getting information from devices;
- information visualization;
- sharing information.

So from above the user interface design patterns for the IoT systems can be grouped into three categories: Set Patterns, Get Patterns, and Event-based Patterns.

In [6] for the smart spaces application there are suggested such patterns as: greeting pattern, farewell pattern, action reaction pattern, conversation pattern and exploration pattern.

For the development of user-machine interactions there are obviously exists such challenges – complexity and manual design. Flow Modeling Language (IFML) is one of the solutions for the designing front-end of the IoT applications.

19.2 Interaction Flow Modelling Language

The standard Interaction Flow Modeling Language (IFML) is designed for expressing the content, user interaction and control behavior of the front-end of software applications [6]. Its metamodel uses the basic data types from the UML metamodel, specializes a number of UML metaclasses as the basis for IFML metaclasses, and presumes that the IFML Domain Model is represented in UML.

IFML is used for expressing (fig.19.2):

- content visualized in the user interfaces
- navigation paths
- user events and interaction
- binding to business logic
- binding to persistence layer.

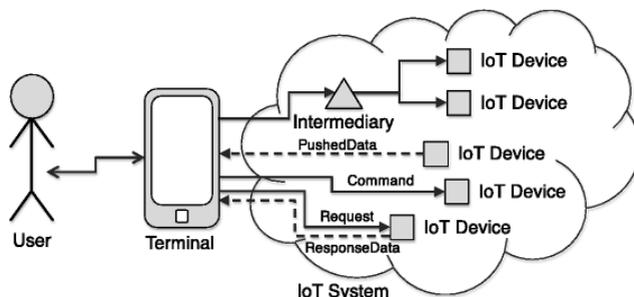


Fig. 19.2 – Example of IFML [10]

It is strongly integrated with such modelling languages as UML, BPMN, SysML, SoaML.

There is also possible to use online open source tool for developing and editing IFML[9].

It allows user to create and edit applications with the IFMF language, edit database while the application is running, generate and run and Web as Mobile prototypes.

19.3. Case Study

19.3.1 Usage of the Remote Laboratory GOLDi for interaction modelling

Cyber-physical systems are the systems that provide the integration of computing, physical processes and networks, or as systems where software and physical subsystems are closely bounded, each of which works in a variety of temporal and spatial dimensions, demonstrating clear and multiple behavioral patterns, and interacts in a variety of ways [11]. Modern trends in productivity and complexity of requirements for systems use require fundamentally new design approaches in which cybernetic and physical components are integrated at different stages.

In general, the qualitative properties of cyber-physical systems can be classified into the following two broad categories:

- reachability or guarantee properties that raise the question of whether a system can achieve a configuration that satisfies a particular property;

- security properties that raise the question of whether the system can remain forever in configurations that satisfy a particular property.

The main properties of cyber-physical systems include following [12]: high degree of automation, reorganization / reconfiguration of the dynamics, cybernetic capabilities in each physical component, the ability of networks to work on multiple scales, integration on different time and spatial scales.

The behavior of cyber-physical systems is described in terms of sequences of events distributed in time. So-called temporal logics are often used for the specification of requirements for cyber-physical systems. Temporal logics are formal languages that allow to define the interrelationships of events in time: causal relationships, restrictions on the relative order, the magnitude of delays between events, etc. The following examples can be cited as temporal properties: the system always works without freezing; two users cannot simultaneously access shared data; a request with a higher weight will be processed before constipation with a lower weight.

Integrated Communication Systems Group at the Ilmenau University of Technology has many years of experience in integrated hard- and software systems and over 10 years of experience in dealing with Internet-supported teaching in the field of digital system design [13]. Grid of Online Lab Devices Ilmenau (GOLDi) gives the students the possibility to work on real physical systems without the need to stand in line at a lab or the need to take care of opening hours and offers the students a working environment that is as close as possible to a real world laboratory. Under real laboratory conditions disturbances can appear and lead to failures of the control algorithm that cannot be detected under virtual lab conditions.

Online laboratories offer various features like visualization and animation, which allows to observe and to test all the properties of the design. In connection with formal design techniques, simulation and prototyping are used to establish a foundation for the development of a reliable system design. To check the functionality of the whole design, some special simulation and validation features are included as integral part of the GOLDI system. This offers various possibilities for the execution of simulations [14], such as:

- usage of simulation models of the physical system for visual prototyping,

- step by step and parallel execution of these prototypes,
- visualization of the simulation process with the tools also used for specification,
- features for test pattern generation and
- code generation for hardware and software synthesis.

GOLDI offers a Web-based environment supporting the above mentioned features to generate and execute a design by using simulation models.

As an example of modeling it was decided to create Kripke structure of the elevator which is located in the GOLDi [15]. This elevator has ability to move upwards and downwards from floor to floor and open or close its door.

The atomic propositions for the Kripke structure representing the elevator are as follows:

1st – elevator is located at the 1st floor;

2nd – elevator is located at the 2nd floor;

DO – door is open;

MU – elevator is moving in the upward direction;

MD – elevator is moving in the downward direction.

For clarity, each state is labeled with both the atomic propositions that are true in the state and the negations of the propositions that are false in the state. The labels on the arcs indicate the actions that cause transitions and are not part of the Kripke structure. Kripke structure of the elevator can be seen at Fig.19.3.

This model can be used for further formal verification. For example one might want to determine that “door of the elevator is closed and it is moving upwards”.

So, $p = \neg 1st \wedge \neg 2nd \wedge \neg DO \wedge MU \wedge \neg MD$. Using Kripke structure this can be determined.

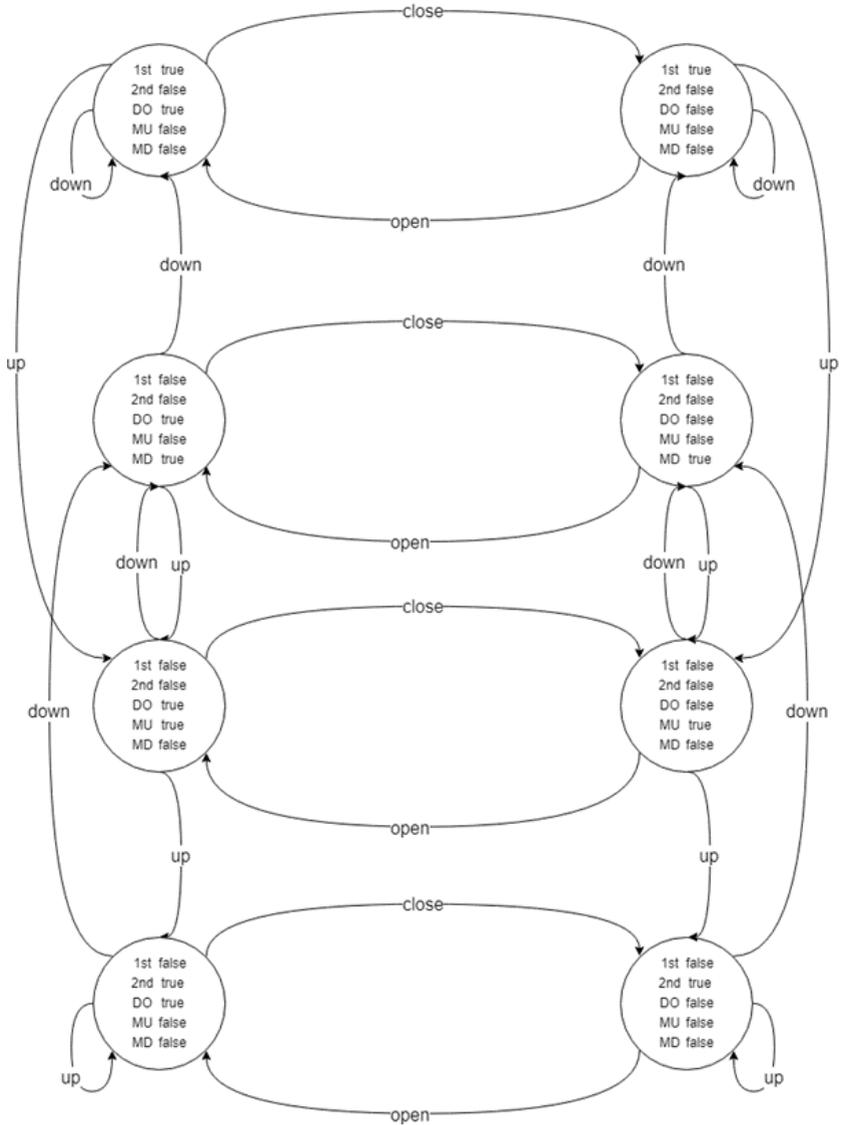


Fig. 19.3 – Kripke structure of the elevator in GOLDi

19.3.2 Simulation of the interaction for Smart Campus.

The idea of a Smart Campus for universities is that the campus talks to you. Individual information for students, teachers and visitors is delivered, depending on their profile and time of day [16-18].

Smart Campus Application consists from three main parts: Mobile application for different operational systems iOS, Android; CMS for updating advertisement information, administration system, which consists from different components aimed to adjust hardware characteristics.

Smart Campus Mobile Application provides users a variety of functionality, allowing working both in on-line mode as in off-line mode detecting buzz from the beacons.

The CMS is providing managing of maps development and storage it in various ways:

- there is possibility to support diversity of media content attributed to one beacon;
- the mobile application provides the search option to find the optimal path to the selected beacon location;
- the mobile application provides an intellectual interface, which allows selecting information based on user preferences.

After analyzing the applications, the main characteristics that should have a voice navigator have been highlighted. The voice navigator, for integration into the Smart-Campus must have the following features:

- to record a voice sentence to get an audience that the user is looking for;
- to recognize vocal sentences and convert them to text;
- to formulate a response to the user;
- to issue a voice message about the user's request;
- to determine the location of the user;
- to build a route from the current position of the user to the required body;
- to display the schedule of occupations of the user;
- to add classes to the schedule;
- to enter the name of the class not only through the virtual keyboard but also through speech recognition;
- to edit or delete selected classes from the schedule;

- to get the route to the chosen lesson;
- to display the schedule for the current day;
- to display the list of recent queries.

The interaction diagram for audio navigation is shown at fig.19.4.

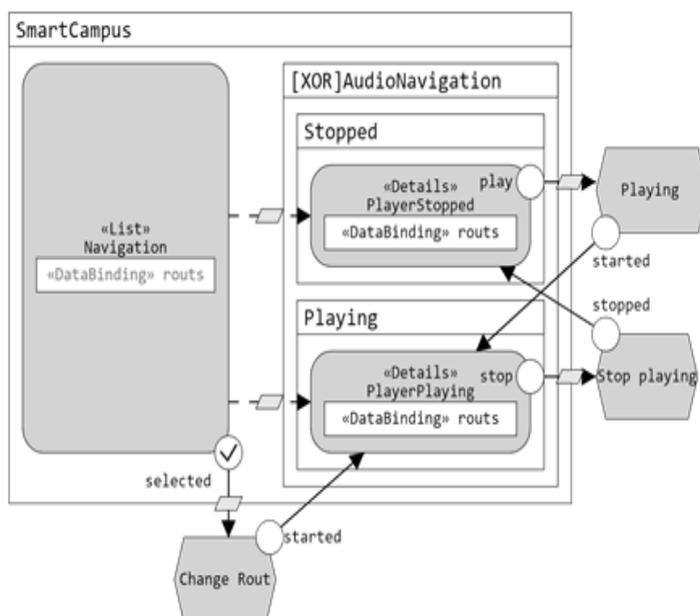


Fig. 19.4 – Interactions in the voice navigator

19.3.3 Interaction Simulation for e-Health systems

The usage of microcontrollers in biomedical applications is an ongoing process and will only increase in the next few years, where the added intelligence augments the possibilities for adaptive therapy and increase knowledge in the healing process or where these systems actually take over body functions. As technology advances microcontrollers are becoming increasingly small and low-power, while their computing power increases rapidly. There are well known

implants such as cardiac pacemakers, implantable cardioverter defibrillators, deep brain stimulation, epidural spinal cord stimulation, cochlear implants and others.

The advances made in technology, such as advanced and smart materials innovations, surgical techniques, robotic surgery and methods of fixations and sterilization facilitated hip implants undergoing multiple design revolutions to seek the least problematic implants and a longer survivorship[2]. As a consequences, a large number of hip endoprosthesis models are available, with different designs, different materials from which they are made and the needed method of fixation. Nevertheless, there are still problems that affect prostheses functionality and longevity.

The average life of the endoprosthesis is 15-20 years, which is caused with a number of physiological changes in bone tissue associated with both age-related changes and with features of transfer of load from the endoprosthesis to the bone.

Implantable medical electronics for hip endoprosthesis are a part of cyber-physical system aimed for continuous monitoring of health and implant state. Different physical parameters can be measured with these systems. Medical staff could receive feedback on the stress and strain in cables connected to the hipbone, torque in the endoprosthesis, the angle of the knee implant.

According to the common architecture of cyber-physical system our Smart Hip Endoprosthesis System consists of three components: implant, contained medical electronics, external remote mobile control system and external cognitive operator [21] .

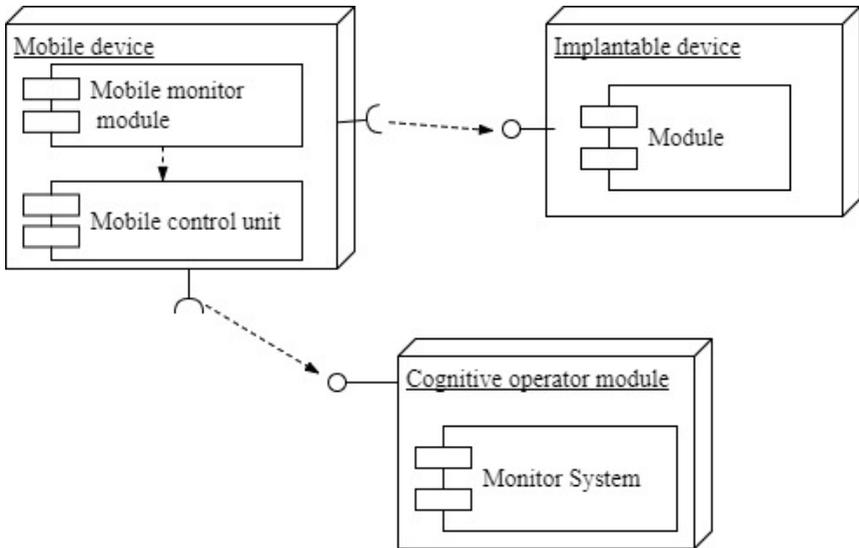


Fig. 19.5 – Common system architecture

The intelligent implant should be able to measure the physical parameters envisaged, possibly log them locally and communicate this data wirelessly. It should also be able to operate autonomously for a long period, have a battery on board for inductive powering. A very important property of the endo-microcontroller system is its reliability over decades as repair is not desirable and often not even possible.

The exo-system is there to receive the data and power the endo-system. This data is delivered wirelessly. Possibly a battery pack can be worn by the patient as a continuous power supply.

The last system is to visualize the data in an easy to understand manner for the paramedics to make their medical analysis on the internal state of the patient and the implant.

The complete system should be designed in a multidisciplinary manner, keeping both the properties of the implant and the intelligent system in mind. The tasks which should be solved are divided into material sciences, aimed to keep the properties of the hip endoprosthesis, and bio-engineering, aimed to develop the common system including hardware and software development.

Within development of the hip endoprosthesis requirements to the biocompatibility with human tissues and sufficient strength characteristics of the material should be considered. The requirement for high strength is due to the fact that during the operation the implant is subjected to extreme external loads of various kinds to fixate the prosthesis, which can lead to fracture of the stem. Thus, the strength characteristics of the endoprosthesis material that determine the strength stocks and the mass of the endoprosthesis play a key role in the design and choice of the implant material.

In the case of the installation of the intellectual system, it becomes necessary to form a blind hole in the endoprosthesis femoral stem. As a consequence, the change in the design of the system and the introduction of a voltage concentration can lead to a redistribution of stresses during the operation of the implant and its destruction. On the other hand, the materials used can have its effect on the possibility of inductive powering and wireless communication.

19.3.4 Interaction Simulation for the Remote Laboratories

The Interactive platform for Embedded Software Development (ISRT) [17,22] consists of a number of smaller dedicated experiments which allow students to study and experiment on different aspects of embedded systems and communication tools over the internet. The aim is to prepare students for IoT. The series of experiments include experiments on the manipulation of components (LED-lights, stepper motors), on communication (mobile phone manipulation), on security (face detection through image detection) and programming (in C++, Python).

The ISRT was built to let students of bachelor and master studies in software development experiment and self-study the different aspect of programming and controlling. After the self-study, students get a project assignment in which they use the different skills they acquired using the ISRT. Evaluation of the learning outcomes was done on the project results.

Tasks include transformation of data, connecting and using different sensors for physical parameters (temperature, light intensity, luminosity, distance), image recognition, detecting time-delays in the execution of programs, access to remote working systems with different protocols like Wi-Fi, Bluetooth Low Energy and GSM. The goal of the

predefined tasks is that students later on will work on an own-defined project in which they combine and use the knowledge to make a physical remote sensing device for some physical status (e.g. ecological measurements, climate control measurements).

19.3.5 Interaction Simulation for Intelligent transport

There are several technologies which are developing in the field of connected cars: Vehicle-to-Vehicle (V2V), Vehicle-to-Infrastructure (V2I), Vehicle-to-Device (V2D), Vehicle-to-Pedestrian (V2P), Vehicle-to-Home (V2H), Vehicle-to-Device (V2D), Vehicle-to-Grid (V2G) solutions [23,24].

Electric vehicles (EV) can be thought of as being a part of the global smart grid. As it is known, the term smart grid is defined primarily by its ability to integrate Information and Communications Technology with large-scale energy networks aimed to increase environmental friendliness of generation, transmission and distribution of electricity and efficiency of the system and has following major components: mass production, transfer, distribution, consumers, service providers, operations, markets [11].

For the reducing complexity of the IoT system it is important to teach students of Software Engineering specialities to simulate and model interaction. For this case a prototype has been developed to mimic an electrical vehical charging station which allows to charge Li-ion batteries with a dedicated charging module as a part of ISRT.

Several tasks need to be fulfilled in the charging station:

- security and identity control of the client who wants to recharge his vehicle
- checking of charging level of the attached batteries
- payment of charging: upfront for limited amount of charging or over subscription
- monitoring of charging time
- pricing of charging energy
- communication with the user
- end of charging when charge reached the full level.

Manipulation is possible locally through RFID access as well as online. The hardware prototype is in fig 2. For physical access a RFID card (ISO 14443A) is used or a dongle which is distributed to the subscribed users of the system.

Remote charging is organised with the usage of Raspberry Pi3, connected to the Internet (either Ethernet, or WiFi or 3G). Clients can use different methods of access and control (laptop, tablet, smartphone..) (see Fig. 3). Software is developed on Python, and connected to the remote server of the ISRT by get request. Server send request to the hardware prototype (charging station) as Start and Stop commands and request for the time of charging in seconds.

Remote control is only possible for the subscription users of the ISRT. At the online panel there is also presented the price of the electricity (Fig.19.6).

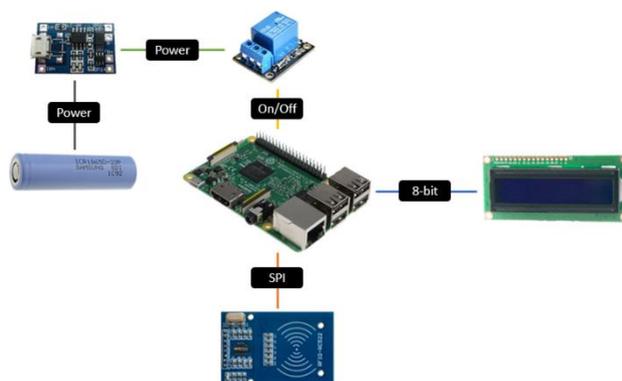


Fig.19.6 – Hardware prototype of the simulated charging station.

In this setup, engineering students need to perform all tasks which will be available in commercial setups, and they can experiments with different approaches and different human-machine interfaces. The multitude of addressed communication platforms makes it a real-life setup.

The ISRT-server is a platform in Zaporizhzhia Polytechnic National University for remote laboratories, in order to train students in IoT-task [22]. It is ideal for the scalability of the charging station case. A ready made black-box charging station module is available in the ISRT so that students can work on the communication and client layer. Next they can also develop their own charging station hardware and eventually put in more functionality and hardware is necessary. In combination they make the complete system. Extra features could be

different methods of fast and slow charging, multiple car charging, payment locally for single users without subscription or dongle, helpdesk access from the charging station over messages or chat to the service provider in case of problems, notification of charging clients that the charging session ended by sms.. Opportunities and extended possibilities are trivial [25].

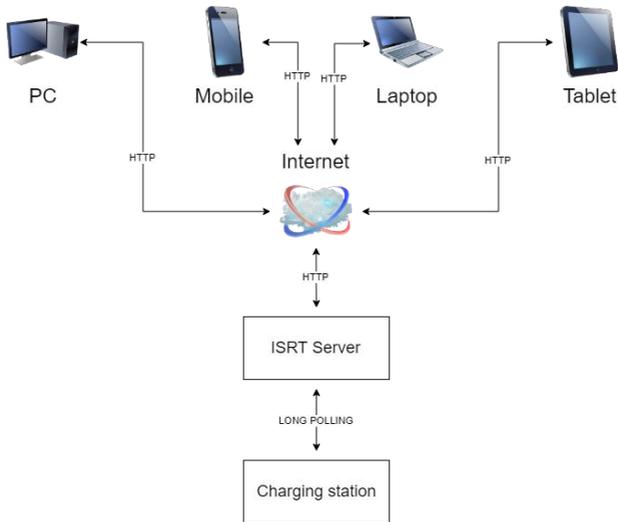


Fig.19.7 – Software system layout

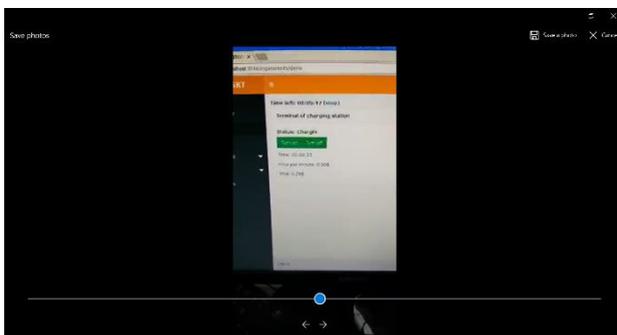


Fig.19.8 – Online Charging Terminal

Flexibility of ISRT infrastructure allows to increase existing prototype into grid what will the next step of the research.

19.4 Work related analysis

Thus, given work has been devoted to the description of the possibilities for the simulation of the interactions in the IoT systems. Also we paid attention to the practical examples of the interaction simulation.

Notation and examples of implementation of the Interaction Flow Modeling Language are described in [7-10]. Implementation of the FSM models are described in [13-15,17].

The basic theory on formal methods, model checking and Kripke models are analysed in [26-31]. Addition material with in-depth descriptions could be found in the literature given below and based on analysis of education programs of Newcastle University [32] and other ALIOT consortium universities.

Conclusions and questions

In order to better understand and assimilate the educational material that is presented in this section, we invite you to answer the following questions:

1. Which types of interactions should be designed in the IoT systems?
2. From which layers consists IoT systems architecture?
3. Which layer of IoT system architecture serves as an integration layer.
4. Which patterns could be used for designing different types of interaction with IoT systems?
5. Which types of interactions from the point of view of the user could be defined?
6. Which types of interactions could be described with Interaction Flow Modeling Language?
7. Which types of interactions could be described with UML?
8. Which types of interactions could be simulated with FSM?

9. Which types of interactions could be simulated with Kripke models?

References

1. C. Rowland, "What's different about user experience design for the Internet of Things?", O'Reilly Media, 2019. Available: <https://www.oreilly.com/learning/whats-different-about-user-experience-design-for-the-internet-of-things>. [Accessed: 28- Jul- 2019]. What's different about user experience design for the Internet of Things? [Online Access]: <https://www.oreilly.com/learning/whats-different-about-user-experience-design-for-the-internet-of-things>

2. "Designing Connected Products", O'Reilly | Safari, 2019. Available: <https://www.oreilly.com/library/view/designing-connected-products/9781449372682/>. [Accessed: 28- Jul- 2019]. J. Groopma Product manufacturers: It's time to rethink the IoT user interface [Online Access]: <https://internetofthingsagenda.techtarget.com/feature/Product-manufacturers-Its-time-to-rethink-the-IoT-user-interface>

3. M. Brambilla, E. Umuhoza and R. Acerbis, "Model-driven development of user interfaces for IoT systems via domain-specific components and patterns", Journal of Internet Services and Applications, vol. 8, no. 1, 2017. Available: 10.1186/s13174-017-0064-1 [Accessed 28 July 2019]. P. Strouks, 4 Stages of IoT architecture explained in simple words [Online Access]: <https://medium.com/datadriveninvestor/4-stages-of-iot-architecture-explained-in-simple-words-b2ea8b4f777f>

4. "IoT Architecture Explained: Building Blocks and How They Work", Scnsoft.com, 2019. Available: <https://www.scnsoft.com/blog/iot-architecture-in-a-nutshell-and-how-it-works>. [Accessed: 28- Jul- 2019].

5. Guth J. et al. (2018) A Detailed Analysis of IoT Platform Architectures: Concepts, Similarities, and Differences. In: Di Martino B., Li KC., Yang L., Esposito A. (eds) Internet of Everything. Internet of Things (Technology, Communications and Computing). Springer, Singapore

6. "Design Patterns for the Internet of Things", Community.arm.com, 2019. Available: <https://community.arm.com/iot/b/blog/posts/design-patterns-for-an-internet-of-things>. [Accessed: 28- Jul- 2019].

7. M. Vega-Barbas, I. Pau, J. C. Augusto and F. Seoane, "Interaction Patterns for Smart Spaces: A Confident Interaction Design Solution for Pervasive Sensitive IoT Services," in IEEE Access, vol. 6, P. 1126-1136, 2018.

8. "IFML: The Interaction Flow Modeling Language | The OMG standard for front-end design", Ifml.org, 2019. Available: <https://www.ifml.org/>. [Accessed: 28- Jul- 2019].

9. "IFML online tool" Available: <http://www.ifmledit.org/>

10. M. Bambilo, "Interaction Flow Modeling Language in the IIoT context". [Online] Available: <https://www.omg.org/news/meetings/tc/ma-15/special-events/iiot-pdf/Brambilla.pdf>

11. Korotunov, S., Tabunshchik, G., Wolff, C.: Cyber-Physical Systems Architectures and Modeling Methods Analysis for Smart Grids. 2018 IEEE 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT). (2018).

12. Miclea, L., Sanislav, T. "About dependability in cyber-physical systems". 2011 9th East-West Design & Test Symposium (EWDTS). (2011).

13. Remote and virtual tools in engineering: student textbook /general editorship Dr.Ing.Karsten Henke. – Zaporizhzhya: Dike Pole, 2016. – pp. 250.

14. Poliakov M. Hybrid Models of Studied Objects Using Remote Laboratories for Teaching Design of Control Systems/ M. Poliakov, T.Larionova, G. Tabunshchik, A. Parkhomenko and Karsten Henke//International Journal of Online Engineering (iJOE), Vol.9(2016), Vienna,IAOE, P. 7-13. <http://dx.doi.org/10.3991/ijoe.v12i09.6128>.

15. S. Korotunov, G.Tabunshchik, K. Henke, D. Wuttke, Analysis of the Verification Approaches for the CyberPhysical Systems. Proceedings of the Second International Workshop on Computer Modeling and Intelligent Systems (CMIS-2019), Zaporizhzhia, Ukraine, April 15-19, 2019. –PP. 950-961 CEUR-WS.org, online <http://ceur-ws.org/Vol-2353/paper75.pdf>

16. Tabunshchik G. Flexible Technologies for Smart Campus/ D. Van Merode, G. Tabunshchik, K. Patrakhalko, Y. Goncharov // Proceedings of XIII International Conference on Remote Engineering and Virtual Instrumentation (REV2016) (24-26 February, 2016, Madrid, Spain) UNED: P. 58-62.

17. Project Oriented Teaching Approaches for E-learning Environment /P. Arras, D. Van Merode, G. Tabunshchik // IEEE 9th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2017. -P.317-320. DOI: 10.1109/IDAACS.2017.8095097

18. Tabunshchik G. Intellectual Flexible Platform for Smart Beacons/G. Tabunshchik, D. Van Merode// In book: Edit by M. Auer, D. Zhutin Online Engineering and Internet of Things, Springer International Publishing, P. 895-900. https://doi.org/10.1007/978-3-319-64352-6_83

19. Tabunshchik G. Interactive platform for Embedded Software Development Study / G. Tabunshchik, D. Van Merode, P. Arras, K. Henke, V. Okhmak// In book: Edit by M. Auer, D. Zhutin Online Engineering and Internet of Things, Springer International Publishing, P. 315-321. DOI 10.1009/978-3-319-64352-6_30.

20. G. Tabunshchik, O. Petrova and P. Arras, "Implementation of Audio Navigation for Smart Campus." Proceedings of the Second International

Workshop on Computer Modeling and Intelligent Systems (CMIS-2019), Zaporizhzhia, Ukraine, April 15-19, 2019, P. 267-276.

21. Engineering Education for HealthCare Purposes: A Ukrainian Perspective // Galyna Tabunshchik, Anzhelika Parkhomenko, Serhij Morshchavka, David Luengo / Conf. proc. of the XIVth International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH), Lviv, Polyana, 18-21 April, -PP. 245 – 249.

22. G. Tabunshchik, P. Arras, T. Kapliienko, Sustainability of the Remote Laboratories based on Systems with Limited Resources // In book: Smart Industry & Smart Education P. 197-224.

23. F. Granda, L. Azpilicueta, C. Vargas-Rosales, R. Lopez-Iturri, E. Aguirre, J. Javier-Astrain, J. Villandangos, F. Falcone, “Spatial characterization of radio propagation channel in urban vehicle-to-infrastructure environments to support WSNs deployment”, in Sensors 2017, vol 17, 1313; doi:10.3390/s17061313,2017.

24. K. C. Dey, A. Rayamajhi, M. Chowdhury, P. Bhavsar, J. Martin, “Vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication in a heterogeneous wireless network – Performance evaluation”, Elsevier, 2016, <https://doi.org/10.1016/J.TRC.2016.03.008>

25. G. Tabunshchik, D. Van Merode, P. Arras, K. Henke, V. Okhmak, “Interactive platform for Embedded Software Development Study”, in: Edit by M. Auer, D. Zhutin Online Engineering and Internet of Things, Springer International Publishing, P. 315-321. DOI 10.1009/978-3-319-64352-6_30

26. P. Grant, “Elementary Computability, Formal Languages and Automata”. Software & Microsystems. 1, 171 (1982).

27. D. Gabbay, Saul A. Kripke, “Semantical considerations for modal logics”. Proceedings of a Colloquium on Modal and Many-valued Logics, Helsinki, 23-26 August, 1962, Acta Philosophica Fennica 1963, P. 83–94. The Journal of Symbolic Logic. 34, 501 (1969).

28. M. Müller-Olm, D. Schmidt, B. Steffen, “Model-Checking. Static Analysis”. P. 330-354 (1999).

29. A. Pnueli, The temporal logic of programs. 18th Annual Symposium on Foundations of Computer Science (sfcS 1977). (1977).

30. E. Clarke, E. Emerson, A. Sistla, “Automatic verification of finite-state concurrent systems using temporal logic specifications”. ACM Transactions on Programming Languages and Systems. 8, P. 244-263 (1986).

31. S. Tonetta, “Linear-time Temporal Logic with Event Freezing Functions”. Electronic Proceedings in Theoretical Computer Science. 256, 195-209 (2017).

32. <https://www.ncl.ac.uk/postgraduate/courses/degrees/advanced-computer-science-msc/#profile>

PART VI. SOFTWARE DEFINED NETWORKS AND IOT**20. SOFTWARE DEFINED NETWORKS BASICS**

Dr. V. V. Shkarupylo, M.Sc. D. Mazur (ZNTU)

Contents

Abbreviations	136
20.1 SDN architecture. Fundamental notions, principles and concepts	137
20.1.1 The evolution of networks, switches and control planes	137
20.1.2 SDN architecture	138
20.1.3 SDN predecessors.....	139
20.1.4 Network virtualization	141
20.2 An in-depth look at the aspects of implementation. Differentiation between Control and Data Planes	142
20.2.1 Fundamental characteristics of SDN. Plane separation	142
20.2.2 SDN operations.....	143
20.2.3 SDN switches	146
20.2.4 SDN controller. Existing SDN controller implementations and their comparison	146
20.3 OpenFlow protocol. The basics, peculiarities and limitations...	150
20.3.1 OpenFlow specification overview	150
20.3.2 OpenFlow switch	151
20.3.3 OpenFlow controller	153
20.3.4 OpenFlow protocol.....	154
20.3.5 OpenFlow v1.0 specification	155
20.3.6 OpenFlow v1.1 specification	156
20.3.7 OpenFlow v1.2 specification	158
20.3.8 OpenFlow v1.3 specification	158
20.4 Work related analysis	161
Conclusion and questions	162
References	163

Abbreviations

ACL – Access Control List

API – Application Programming Interface

ASIC - Application-specific Integrated Circuit

ATM – Asynchronous Transfer Mode

DARPA – Defense Advanced Research Projects Agency

IDS – Intrusion Detection System

IP – Internet Protocol

MAC – Media Access Control

NEMs – Network Equipment Manufacturers

OpenSig – Open Signaling

OVS – Open vSwitch

QoS – Quality of Services

REST – Representational State Transfer

SDN – Software-defined Network

TCAM – Ternary Content Addressable Memory

TCP – Transmission Control Protocol

UDP – User Datagram Protocol

VLAN – Virtual Local Area Network

20.1 SDN architecture. Fundamental notions, principles and concepts

20.1.1 The evolution of networks, switches and control planes

Since the first network was created in 1969, almost everything other than the physical layer (layer one) was implemented in software. Even the simplest tasks, such as MAC-level decisions, were used by software inside the devices. This remained true even through the early days of the commercialized Internet in the early 1990s. But step by step networks have changed, and practically everything has been implemented at the physical level through a short time. In the Fig. 20.1 we can see the evolutions of the networks [1].

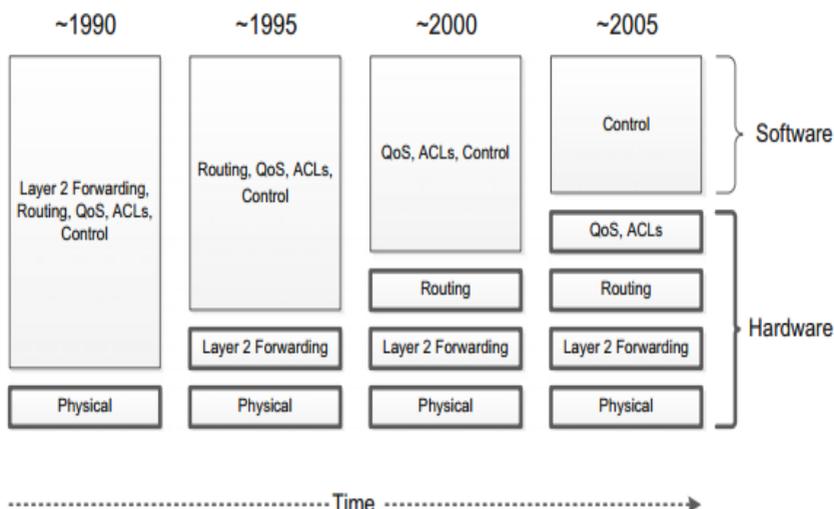


Fig. 20.1 – Migration of layers into the hardware

The network device evolution we have recounted thus far has yielded the following current situation:

- bridging (layer two forwarding). Basic layer two MAC forwarding of packets is handled in the hardware tables;
- routing (layer three forwarding). To keep up with today's high-speed links and to route packets at link speeds, layer three forwarding functionality is also implemented in hardware tables;

- advanced filtering and prioritization. General traffic management rules such as ACLs, which filter, forward, and prioritize packets, are handled via hardware tables located in the hardware (e.g., in TCAMs) and accessed through low-level software;

- control. The control software used to make broader routing decisions and to interact with other devices in order to converge on topologies and routing paths is implemented in software that runs autonomously inside the devices. Since the current control plane software in networking devices lacks the ability to distribute policy information about such things as security, QoS, and ACLs, these features must still be provisioned through relatively primitive configuration and management interfaces.

Given this landscape of layer two and layer three hardware handling most forwarding tasks, software in the device providing control plane functionality, and policy implemented via configuration and management interfaces, an opportunity presents itself to simplify networking devices and move forward to the next generation of networking [1].

20.1.2 SDN architecture

SDN is about moving that control software off the device and into a centrally located compute resource that is capable of seeing the entire network and making decisions that are optimal, given a complete understanding of the situation. According to this, we can define 3 layers of SDN architecture (Fig. 20.2):

- forwarding. Forwarding responsibilities, implemented in hardware tables, remain on the device. In addition, features such as filtering based on ACLs and traffic prioritization are enforced locally on the device as well;

- control. All needed control software moved from devices to centralized controller, which has complete view of the network. That's means that controller will manage the network, will provide routes between devices, will make rules for the network and so on;

- application. Above the controller is where the network applications run, implementing higher-level functions and, additionally, participating in decisions about how best to manage and control packet forwarding and distribution within the network.

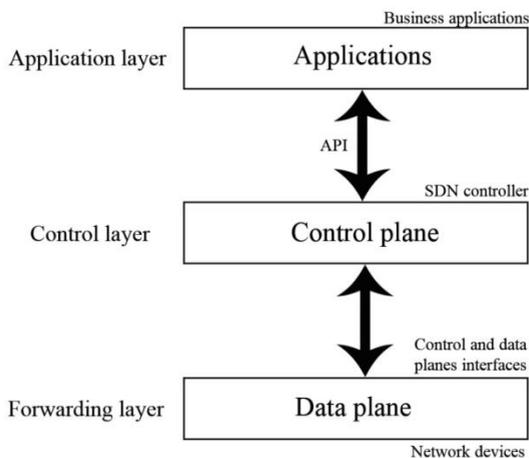


Fig. 20.2 – SDN architecture

20.1.3 SDN predecessors

Concept of programmable networks appeared in the mid-90s, when the Internet was starting experience widespread success. Because of it networks started to growing, connecting a huge number of devices. But some time later, appeared a problem of managing the network infrastructure. Network devices were used as black boxes designed to support specific protocols essential for the operation of the network, without even guaranteeing vendor interoperability. Therefore, modifying the control logic of such devices was not an option, severely restricting network evolution. To remedy this situation, various efforts focused on finding novel solutions for creating more open, extensible and programmable networks [2].

Two of the most significant early ideas proposing ways of separating the control software from the underlying hardware and providing open interfaces for management and control were of the Open Signaling (OpenSig) working group and from the Active Networking initiative [3, 4].

OpenSig. The Open Signaling working group appeared in 1995 and focused on applying the concept of programmability in ATM networks. The main idea was the separation of the control and data

plane of networks, with the signaling between the planes performed through an open interface. As a result, it would be possible to control and program ATM switches remotely, essentially turning the whole network into a distributed platform, greatly simplifying the process of deploying new services. The ideas advocated by the OpenSig community for open signaling interfaces acted as motivation for further research. Towards this direction, the Tempest framework [5], based on the OpenSig philosophy, allowed multiple switch controllers to manage multiple partitions of the switch simultaneously and consequently to run multiple control architectures over the same physical ATM network. This approach gave more freedom to network operators, as they were no longer forced to define a single unified control architecture satisfying the control requirements of all future network services.

Active Networking. The Active Networking initiative appeared in the mid-90s and was mainly supported by DARPA [6]. Like OpenSig, its main goal was the creation of programmable networks which would promote network innovations. The main idea behind active networking is that resources of network nodes are exposed through a network API, allowing network operators to actively control the nodes as they desire by executing arbitrary code. Therefore, contrary to the static functionality offered by OpenSig networks, active networking allowed the rapid deployment of customized services and the dynamic configuration of networks at run-time.

The general architecture of active networks defines a three-layer stack on active nodes. At the bottom layer sits an operating system (NodeOS) multiplexing the node's communication, memory and computational resources among the packet flows traversing the node. Various projects proposing different implementations of the NodeOS exist, with some prominent examples being the NodeOS project and Bowman. At the next layer exist one or more execution environments providing a model for writing active networking applications, including ANTS and PLAN. Finally, at the top layer are the active applications themselves, i.e. the code developed by network operators.

Two programming models fall within the work of the active networking community; the capsule model, in which the code to be executed is included in regular data packets; and the programmable router/switch model, in which the code to be executed at network nodes is established through out-of-band mechanisms. Out of the two, the capsule

model came to be the most innovative and most closely associated with active networking. The reason is that it offered a radically different approach to network management, providing a simple method of installing new data plane functionality across network paths. However, both models had a significant impact and left an important legacy, since many of the concepts met in SDN (separation of the control and data plane, network APIs etc.) come directly from the efforts of the active networking community [2].

20.1.4 Network virtualization

The urgency for automation, multitenancy, and multipathing has increased as a result of the scale and fluidity introduced by server and storage virtualization. The general idea of virtualization is that you create a higher-level abstraction that runs on top of the actual physical entity you are abstracting. The growth of compute and storage server virtualization has created demand for network virtualization. This means having a virtual abstraction of a network running on top of the actual physical network. With virtualization, the network administrator should be able to create a network anytime and anywhere he chooses, as well as expanding and contracting networks that already exist. Intelligent virtualization software should be capable of this task without requiring the upper virtualized layer to be aware of what is occurring at the physical layer [1].

Server virtualization has caused the scale of networks to increase as well, and this increased scale has put pressure on layer two and layer three networks as they exist today. Some of these pressures can be alleviated to some degree by tunnels and other types of technologies, but fundamental network issues remain, even in those situations. Consequently, the degree of network virtualization required to keep pace with data center expansion and innovation is not possible with the network technology that is available today [1].

To summarize, advances in data center technology have caused weaknesses in the current networking technology to become more apparent. This situation has spurred demand for better ways to construct and manage networks [7], and that demand has driven innovation around SDN [8].

20.2 An in-depth look at the aspects of implementation. Differentiation between Control and Data Planes

20.2.1 Fundamental characteristics of SDN. Plane separation

Software Defined Networking, as it evolved from prior proposals, standards, and implementations such as ForCES, 4D, and Ethane, is characterized by five fundamental traits: plane separation, a simplified device, centralized control, network automation and virtualization, and openness.

The first fundamental characteristic of SDN is the separation of the forwarding and control planes. Forwarding functionality, including the logic and tables for choosing how to deal with incoming packets based on characteristics such as MAC address, IP address, and VLAN ID, resides in the forwarding plane. The fundamental actions performed by the forwarding plane can be described by the way it dispenses with arriving packets. It may forward, drop, consume, or replicate an incoming packet. For basic forwarding, the device determines the correct output port by performing a lookup in the address table in the hardware ASIC. A packet may be dropped due to buffer overflow conditions or due to specific filtering resulting from a QoS rate-limiting function, for example. Special-case packets that require processing by the control or management planes are consumed and passed to the appropriate plane. Finally, a special case of forwarding pertains to multicast, where the incoming packet must be replicated before forwarding the various copies out different output ports.

The protocols, logic, and algorithms that are used to program the forwarding plane reside in the control plane. Many of these protocols and algorithms require global knowledge of the network. The control plane determines how the forwarding tables and logic in the data plane should be programmed or configured. Since in a traditional network each device has its own control plane, the primary task of that control plane is to run routing or switching protocols so that all the distributed forwarding tables on the devices throughout the network stay synchronized. The most basic outcome of this synchronization is the prevention of loops.

Although these planes have traditionally been considered logically separate, they co-reside in legacy Internet switches. In SDN, the control plane is moved off the switching device and onto a centralized controller [1].

20.2.2 SDN operations

At a conceptual level, the behavior and operation of a Software Defined Network is straightforward. In Fig. 20.3 we provide a graphical depiction of the operation of the basic components of SDN: the SDN devices, the controller, and the applications. The easiest way to understand the operation is to look at it from the bottom up, starting with the SDN device. As shown in Fig. 20.3, the SDN devices contain forwarding functionality for deciding what to do with each incoming packet. The devices also contain the data that drives those forwarding decisions. The data itself is actually represented by the flows defined by the controller, as depicted in the upper-left portion of each device.

A flow describes a set of packets transferred from one network endpoint (or set of endpoints) to another endpoint (or set of endpoints).

The endpoints may be defined as IP address TCP/UDP port pairs, VLAN endpoints, layer three tunnel endpoints, and input ports, among other things. One set of rules describes the forwarding actions that the device should take for all packets belonging to that flow. A flow is unidirectional in that packets flowing between the same two endpoints in the opposite direction could each constitute a separate flow. Flows are represented on a device as a flow entry.

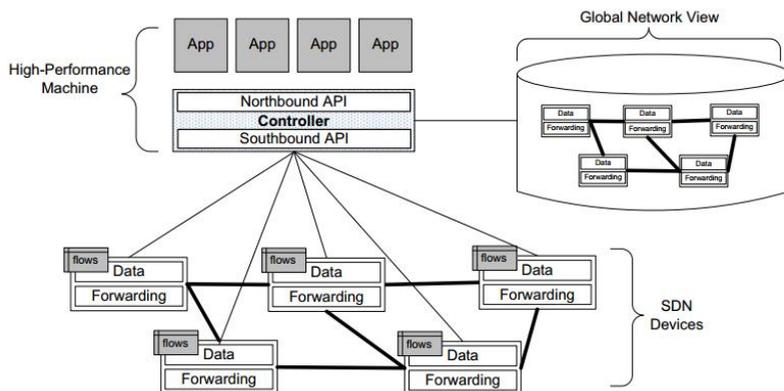


Fig. 20.3 – SDN operations overview

A flow table resides on the network device and consists of a series of flow entries and the actions to perform when a packet matching that flow arrives at the device. When the SDN device receives a packet, it consults its flow tables in search of a match. These flow tables had been constructed previously when the controller downloaded appropriate flow rules to the device. If the SDN device finds a match, it takes the appropriate configured action, which usually entails forwarding the packet. If it does not find a match, the switch can either drop the packet or pass it to the controller, depending on the version of OpenFlow and the configuration of the switch.

The definition of a flow is a relatively simple programming expression of what may be a very complex control plane calculation previously performed by the controller. For the reader who is less familiar with traditional switching hardware architecture, it is important to understand that this complexity is such that it simply cannot be performed at line rates and instead must be digested by the control plane and reduced to simple rules that can be processed at that speed. In Open SDN, this digested form is the flow entry.

The SDN controller is responsible for abstracting the network of SDN devices it controls and presenting an abstraction of these network resources to the SDN applications running above. The controller allows the SDN application to define flows on devices and to help the application respond to packets that are forwarded to the controller by the SDN devices. In Fig. 20.3 we see on the right side of the controller that it maintains a view of the entire network that it controls. This permits it to calculate optimal forwarding solutions for the network in a deterministic, predictable manner. Since one controller can control a large number of network devices, these calculations are normally performed on a high-performance machine with an order-of-magnitude performance advantage over the CPU and memory capacity than is typically afforded to the network devices themselves. For example, a controller might be implemented on an eight-core, 2-GHz CPU versus the single-core, 1-GHz CPU that is more typical on a switch.

SDN applications are built on top of the controller. These applications should not be confused with the application layer defined in the seven-layer OSI model of computer networking. Since SDN applications are really part of network layers two and three, this concept is orthogonal to that of applications in the tight hierarchy of OSI protocol layers. The SDN application interfaces with the controller, using it to set proactive flows on the devices and to receive packets that have been forwarded to the controller.

Proactive flows are established by the application; typically, the application will set these flows when the application starts up, and the flows will persist until some configuration change is made. This kind of proactive flow is known as a static flow. Another kind of proactive flow is where the controller decides to modify a flow based on the traffic load currently being driven through a network device.

In addition to flows defined proactively by the application, some flows are defined in response to a packet forwarded to the controller. Upon receipt of incoming packets that have been forwarded to the controller, the SDN application will instruct the controller as to how to respond to the packet and, if appropriate, will establish new flows on the device in order to allow that device to respond locally the next time it sees a packet belonging to that flow. Such flows are called reactive flows. In this way, it is now possible to write software applications that implement forwarding, routing, overlay, multipath, and access control functions, among others.

There are also reactive flows that are defined or modified as a result of stimuli from sources other than packets from the controller. For example, the controller can insert flows reactively in response to other data sources such as intrusion detection systems (IDS) or the NetFlow traffic analyzer [9].

An OpenFlow protocol as the mean of communication between the controller and the device is depicted in Fig. 20.4 [1].

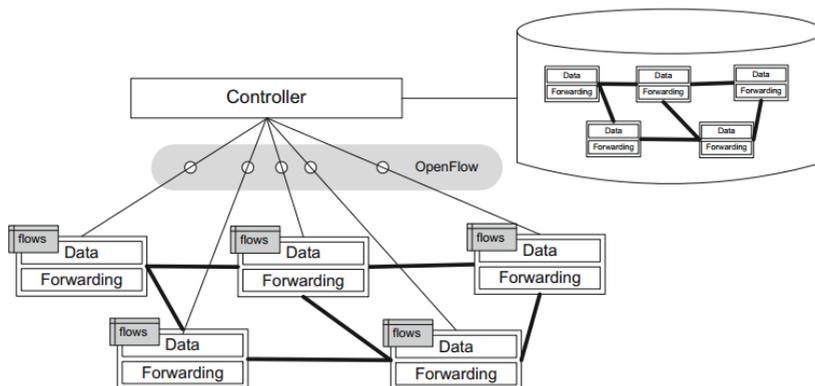


Fig. 20.4 – Controller-to-device communication

20.2.3 SDN switches

First of all, to create a SDN, you need special switches. A number of SDN switches implementations are available today, both commercial and open source. Software SDN devices are predominantly open source. Currently, two main alternatives are available: Open vSwitch (OVS) from Nicira and Indigo from Big Switch [10, 11]. Incumbent network equipment manufacturers (NEMs), such as Cisco, HP, NEC, IBM, Juniper, and Extreme, have added OpenFlow support to some of their legacy switches. Generally, these switches may operate in both legacy mode as well as OpenFlow mode. There is also a new class of devices called white-box switches, which are minimalist in that they are built primarily from merchant silicon switching chips and a commodity CPU and memory by a low-cost original device manufacturer (ODM) lacking a well-known brand name. One of the premises of SDN is that the physical switching infrastructure may be built from OpenFlow-enabled white-box switches at far less direct cost than switches from established NEMs. Most legacy control plane software is absent from these devices, since this functionality is largely expected to be provided by a centralized controller. Such white-box devices often use the open source OVS or Indigo switch code for the OpenFlow logic, then map the packet-processing part of those switch implementations to their particular hardware [1].

20.2.4 SDN controller. Existing SDN controller implementations and their comparison

The controller maintains a view of the entire network, implements policy decisions, controls all the SDN devices that comprise the network infrastructure, and provides a northbound API for applications. When we have said that the controller implements policy decisions regarding routing, forwarding, redirecting, load balancing, and the like, these statements referred to both the controller and the applications that make use of that controller. Controllers often come with their own set of common application modules, such as a learning switch, a router, a basic firewall, and a simple load balancer. These are really SDN applications, but they are often bundled with the controller. Here we focus strictly on the controller.

The anatomy of an SDN controller is represented in Fig. 20.5. The modules that provide the controller’s core functionality, both a northbound and a southbound API, and a few sample applications that might use the controller are depicted in Fig. 20.5. As we described earlier, the southbound API is used to interface with the SDN devices.

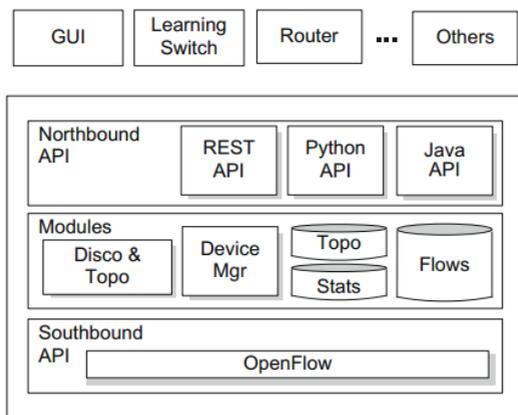


Fig. 20.5 – SDN controller components

This API is OpenFlow in the case of Open SDN or some proprietary alternative in other SDN solutions. It is worth noting that in some product offerings, both OpenFlow and alternatives coexist on the same controller. Early work on the southbound API has resulted in more maturity of that interface with respect to its definition and standardization. [1] OpenFlow itself is the best example of this maturity, but de facto standards such as the Cisco CLI and SNMP also represent standardization in the southbound-facing interface. OpenFlow’s companion protocol, OF-Config [12], and Nicira’s Open vSwitch Database Management Protocol (OVSDB) [13] are both open protocols for the southbound interface, though these are limited to configuration roles.

Unfortunately, there is currently no northbound counterpart to the southbound OpenFlow standard or even the de facto legacy standards. This lack of a standard for the controller-to-application interface is considered a current deficiency in SDN, and some bodies are

developing proposals to standardize it. The absence of a standard notwithstanding, northbound interfaces have been implemented in a number of disparate forms. For example, the Floodlight controller includes a Java API and a Representational State Transfer (RESTful) API [14]. The OpenDaylight controller provides a RESTful API for applications running on separate machines [15]. The northbound API represents an outstanding opportunity for innovation and collaboration among vendors and the open source community.

Requirements to SDN controllers can be divided into 2 main characteristics:

- performance: throughput (about 10M events per second), delay (us);
- programmability: functionality (applications and services), programming Interface.

There are many different controllers existing. All of them have different characteristics. The main controllers and their general characteristics are listed in the table 20.1.

Table 20.1 – The main controllers and their general characteristics

	POX	Ryu	Trema	FloodLight	OpenDaylight
Interfaces	SB(<u>OpenFlow</u>)	SB(<u>OpenFlow</u>) +SB Management (OVSDB JSON)	SB(<u>OpenFlow</u>)	SB(<u>OpenFlow</u>) NB (Java & REST)	SB(<u>OpenFlow</u>) NB (Java & REST)
Virtualization	<u>Mininet&Open vSwitch</u>	<u>Mininet&Open vSwitch</u>	Buil-in Emulation Virtual tool	<u>Mininet&Open vSwitch</u>	<u>Mininet&Open vSwitch</u>
GUI	Yes	Yes	No	<u>WebUI</u>	Yes
Rest API	No	Yes	No	Yes	Yes
Productivity	Medium	Medium	High	Medium	Medium
Open Source	Yes	Yes	Yes	Yes	Yes
Documentation	Poor	Medium	Medium	Good	Medium
Language support	Python	Python- Specific + Message Passing Reference	Ruby/C	Java + Any languages that uses REST	Java
Modularity	Medium	Medium	Medium	High	High
Platform support	Linux, Mac OS, Windows	Most Supported on Linux	Linux only	Linux, Mac OS, Windows	Linux
TLS support	Yes	Yes	Yes	Yes	Yes
OpenFlow support	OF v1.0	OF v1.0, OF v1.2, OF v1.3	OF v1.0	OF v1.0	OF v1.0 OF v1.3
OpenStack support (quantum)	NO	Strong	Weak	Medium	Medium

In the Fig. 20.6 the comparison of controllers is provided. The dependence between flows per second and the number of threads is shown. The increase of threads number stipulates the increase of flows per second. More flows per second gives more performance.

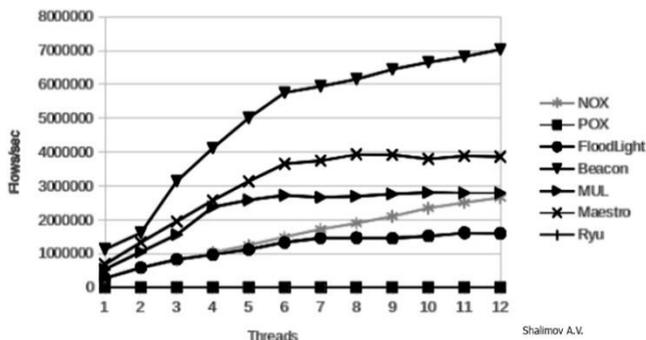


Fig. 20.6 – Comparison of controllers

In addition to the above controllers, there is a controller with very high performance. It is called In-kernel controller. The comparison between In-kernel controller and others is given in Fig. 20.7.

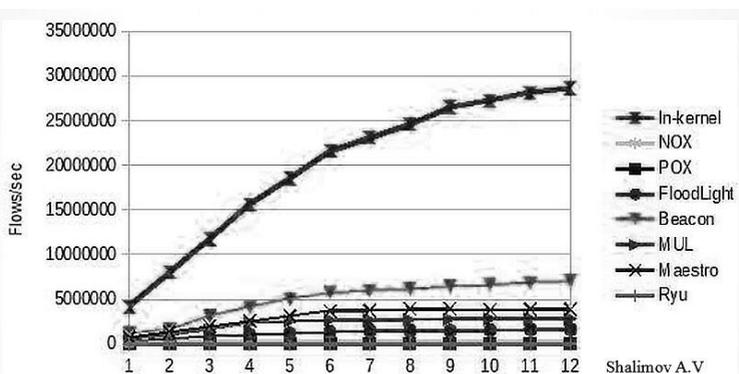


Fig. 20.7 – Comparison between In-kernel controller and others

As it can be seen in Fig. 20.7, the number of flows per second of this controller is significantly higher than others.

High performance of this controller is provided by the implementation in the Linux kernel. In-kernel controller is super productive, because of less time to work with virtual memory, but it has limitations as well. First of all, it's very difficult to develop your applications because of low-level programming language. Also it has limited number of libraries and debugging tools. To provide high performance, hardware of controller must be very powerful. Otherwise, there is a high risk of crashing the whole system.

20.3 OpenFlow protocol. The basics, peculiarities and limitations

20.3.1 OpenFlow specification overview

The OpenFlow specification has been evolving for a number of years. The nonprofit Internet organization openflow.org was created in 2008 as a mooring to promote and support OpenFlow. Though openflow.org existed formally on the Internet, in the early years the physical organization was really just a group of people that met informally at Stanford University. From its inception OpenFlow was intended to “belong” to the research community to serve as a platform for open network switching experimentation, with an eye on commercial use through commercial implementations of this public specification. The first release, Version 1.0.0, appeared on December 31, 2009, though numerous point prereleases existed before then and were made available for experimental purposes as the specification evolved. At this point and continuing up through release 1.1.0, development and management of the specification were performed under the auspices of openflow.org. On March 21, 2011, the Open Network Foundation (ONF) was created for the express purpose of accelerating the delivery and commercialization of SDN. There are a number of proponents of SDN that offer SDN solutions that are not based on OpenFlow. For the ONF, however, OpenFlow remains at the core of its SDN vision for the future. For this reason, the ONF has become the responsible entity for the evolving OpenFlow specification.

Starting after the release of V.1.1, revisions to the OpenFlow specification have been released and managed by the ONF.

One could get the impression from the fanfare surrounding OpenFlow that the advent of this technology has been accompanied by concomitant innovation in switching hardware. The reality is a bit more complicated. The OpenFlow designers realized a number of years ago that many switches were really built around ASICs controlled by rules encoded in tables that could be programmed. Over time, fewer homegrown versions of these switching chips were being developed, and there was greater consolidation in the semiconductor industry. More manufacturers' switches were based on ever-consolidating switching architecture and programmability, with ever-increasing use of programmable switching chips from a relatively small number of merchant silicon vendors. OpenFlow is an attempt to allow the programming, in a generic way, of the various implementations of switches that conform to this new paradigm. OpenFlow attempts to exploit the table-driven design extant in many of the current silicon solutions. As the number of silicon vendors consolidates, there should be a greater possibility for alignment with future OpenFlow versions.

It is worth pausing here to remark on the fact that we are talking a lot about ASICs for a technology called Software Defined Networking. Yet hardware must be part of the discussion, since it is necessary to use this specialized silicon in order to switch packets at high line rates. What is really meant by the word software in the name SDN, then, is that the SDN devices are fully programmable, not that everything is done using software running on a traditional CPU [1].

20.3.2 OpenFlow switch

The basic functions on an OpenFlow V.1.0 switch and its relationship to a controller are depicted in Fig. 20.8. As would be expected in a packet switch, we see that the core function is to take packets that arrive on one port (path X on port 2 in Fig. 20.8) and forward it through another port (port N in Fig. 20.8), making any necessary packet modifications along the way. A unique aspect of the OpenFlow switch is embodied in the packet-matching function shown in Fig. 20.8. The wide, gray, double arrow in Fig. 20.8 starts in the decision logic, shows a match with a particular entry in that table, and

directs the now-matched packet to an action box on the right. This action box has three fundamental options for the disposition of this arriving packet:

- forward the packet out a local port, possibly modifying certain header fields first;
- drop the packet;
- pass the packet to the controller.

These three fundamental packet paths are illustrated in Fig. 20.8.

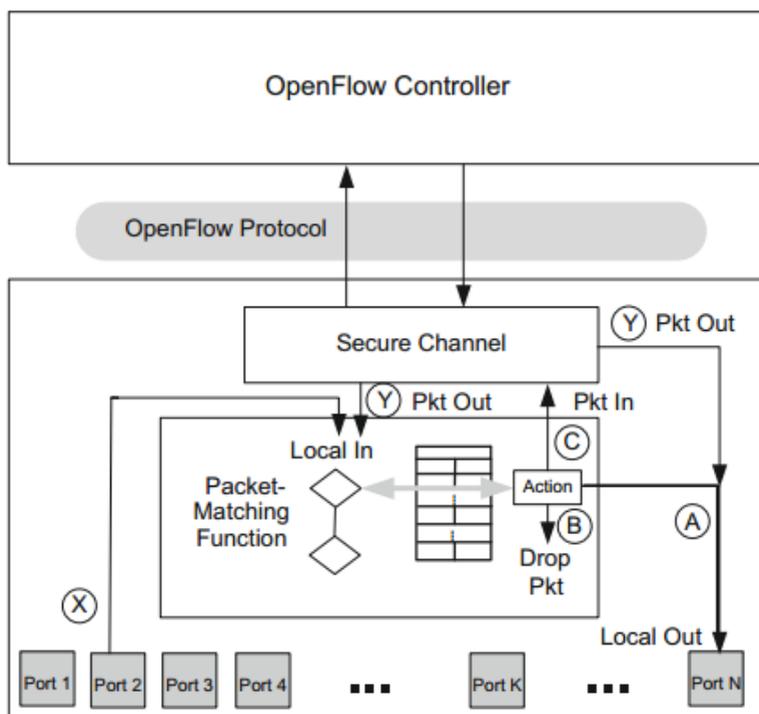


Fig. 20.8 – OpenFlow v1.0 compatible switch

In Fig. 20.8, in the case of path C, the packet is passed to the controller over the secure channel shown in Fig. 20.8. If the controller has either a control message or a data packet to give to the switch, the controller uses this same secure channel in the reverse direction. When

the controller has a data packet to forward out through the switch, it uses the OpenFlow PACKET_OUT message. We see in Fig. 20.8 that such a data packet coming from the controller may take two different paths through the OpenFlow logic, both denoted with Y.

In the rightmost case, the controller directly specifies the output port and the packet is passed to that port N in the example. In the leftmost path Y case, the controller indicates that it wants to defer the forwarding decision to the packet-matching logic. A given OpenFlow switch implementation is either OpenFlow-only or OpenFlow-hybrid. An OpenFlow-only switch is one that forwards packet solely according to the OpenFlow logic described above. An OpenFlow hybrid is a switch that can also switch packets in its legacy mode as an Ethernet switch or IP router. One can view the hybrid case as an OpenFlow switch residing next to a completely independent traditional switch. Such a hybrid switch requires a preprocessing classification mechanism that directs packets to either OpenFlow processing or the traditional packet processing. It is probable that hybrid switches will be the norm during the migration to pure OpenFlow implementations. Note that we use the term OpenFlow switch in this chapter instead of the term OpenFlow device we customarily use. This is because switch is the term used in the OpenFlow specification. In general, though, we opt to use the term device, since there are already non switch devices being controlled by OpenFlow controllers, such as wireless access points [1].

20.3.3 OpenFlow controller

Modern Internet switches make millions of decisions per second about whether or not to forward an incoming packet, to what set of output ports it should be forwarded, and what header fields in the packet may need to be modified, added, or removed. This is a very complex task. The fact that this can be carried out at line rates on multigigabit media is a technological wonder. The switching industry has long understood that not all functions on the switching data path can be carried out at line rates, so there has long been the notion of splitting the pure data plane from the control plane. The data plane matches headers, modifies packets, and forwards them based on a set of forwarding tables and associated logic, and it does this very, very fast. The rate of decisions being made as packets stream into a switch

through a 100 Gbps interface is astoundingly high. The control plane runs routing and switching protocols and other logic to determine what the forwarding tables and logic in the data plane should be. This process is very complex and cannot be done at line rates as the packets are being processed, and it is for this reason we have seen the control plane separated from the data plane, even in legacy network switches.

The OpenFlow control plane differs from the legacy control plane in three key ways. First, it can program different data plane elements with a common, standard language, OpenFlow. Second, it exists on a separate hardware device than the forwarding plane, unlike traditional switches, where the control plane and data plane are instantiated in the same physical box. This separation is made possible because the controller can program the data plane elements remotely over the Internet. Third, the controller can program multiple data plane elements from a single control plane instance.

The OpenFlow controller is responsible for programming all the packet-matching and forwarding rules in the switch. Whereas a traditional router would run routing algorithms to determine how to program its forwarding table, that function or an equivalent replacement to it is now performed by the controller. Any changes that result in recomputing routes will be programmed onto the switch by the controller [1].

20.3.4 OpenFlow Protocol

As shown in Fig. 20.8, the OpenFlow protocol defines the communication between an OpenFlow controller and an OpenFlow switch. This protocol is what most uniquely identifies OpenFlow technology. At its essence, the protocol consists of a set of messages that are sent from the controller to the switch and a corresponding set of messages that are sent in the opposite direction. Collectively the messages allow the controller to program the switch so as to allow fine-grained control over the switching of user traffic. The most basic programming defines, modifies, and deletes flows. The endpoints may be defined as IP address-TCP/UDP port pairs, VLAN endpoints, layer three tunnel endpoints, or input ports, among other things. One set of rules describes the forwarding actions that the device should take for all packets belonging to that flow. When the controller defines a flow, it is

providing the switch with the information it needs to know how to treat incoming packets that match that flow. The possibilities for treatment have grown more complex as the OpenFlow protocol has evolved, but the most basic prescriptions for treatment of an incoming packet are denoted by paths A, B, and C in Fig. 20.8. These three options are to forward the packet out one or more output ports, drop the packet, or pass the packet to the controller for exception handling.

The OpenFlow protocol has evolved significantly with each version of OpenFlow, so we cover the detailed messages of the protocol in the version-specific sections that follow. The specification has evolved from development point release 0.2.0 on March 28, 2008, through release V.1.3.0, released in 2012. Numerous point releases over the intervening years have addressed problems with earlier releases and added incremental functionality. OpenFlow was viewed primarily as an experimental platform in its early years. For that reason, there was little concern on the part of the development community in advancing this standard to provide for interoperability between releases. As OpenFlow began to see more widespread commercial deployment, backward compatibility has become an increasingly important issue. Many features, however, were introduced in earlier versions of OpenFlow that are no longer present in the current version [16].

20.3.5 OpenFlow v1.0 specification

OpenFlow v1.0 was released in December 2009 [17]. This version supports multiple queues per output port. Queues support the ability to provide minimum bandwidth guarantees; the bandwidth allocated to each queue is configurable. The name slicing is derived from the ability to provide a slice of the available network bandwidth to each queue.

Flows have been extended to include an opaque identifier, referred to as a cookie. The cookie is specified by the controller when the flow is installed; the cookie will be returned as part of each flow stats and flow expired message.

The OFPST_DESC (switch description) reply in v1.0 includes a datapath description field. This is a user specifiable field that allows a switch to return a string specified by the switch owner to describe the switch.

The reference implementation in this version can match on IP fields inside ARP packets. The source and destination protocol address are mapped to the `nw_src` and `nw_dst` fields respectively, and the opcode is mapped to the `nw_proto` field.

Flow durations in stats and expiry messages in v1.0 expresses with nanosecond resolution. Note that the accuracy of flow durations in the reference implementation is on the order of milliseconds. The actual accuracy particularly depends on kernel parameters.

20.3.6 OpenFlow v1.1 specification

OpenFlow v1.1 was released in February 2011 [18].

Prior versions of the OpenFlow specification did expose to the controller the abstraction of a single table. The OpenFlow pipeline could internally be mapped to multiple tables, such as having a separate wildcard and exact match table, but those tables would always act logically as a single table.

OpenFlow 1.1 introduces a more flexible pipeline with multiple tables. Exposing multiple tables has many advantages. The first advantage is that many hardware have multiple tables internally (for example L2 table, L3 table, multiple TCAM lookups), and the multiple table support of OpenFlow may enable to expose this hardware with greater efficiency and flexibility. The second advantage is that many network deployments combine orthogonal processing of packets (for example ACL, QoS and routing), forcing all those processing in a single table creates huge ruleset due to the cross product of individual rules, multiple tables may decouple properly those processing.

The new OpenFlow pipeline with multiple tables is quite different from the simple pipeline of prior OpenFlow versions. The new OpenFlow pipeline expose a set of completely generic tables, supporting the full match and full set of actions. It's difficult to build a pipeline abstraction that represent accurately all possible hardware, therefore OpenFlow 1.1 is based on a generic and flexible pipeline that may be mapped to the hardware. Some limited table capabilities are available to denote what each table is capable of supporting.

Packets are processed through the pipeline, they are matched and processed in the first table, and may be matched and processed in other tables. As it goes through the pipeline, a packet is associated with an

action set, accumulating action, and a generic metadata register. The action set is resolved at the end of the pipeline and applied to the packet. The metadata can be matched and written at each table and enables to carry state between tables.

OpenFlow introduces a new protocol object called instruction to control pipeline processing. Actions which were directly attached to flows in previous versions are now encapsulated in instructions. Instructions may apply those actions between tables or accumulate them in the packet action set. Instructions can also change the metadata, or direct packet to another table.

The new group abstraction enables OpenFlow to represent a set of ports as a single entity for forwarding packets. Different types of groups are provided, to represent different abstractions such as multicasting or multipathing. Each group is composed of a set group buckets, each group bucket contains the set of actions to be applied before forwarding to the port. Groups buckets can also forward to other groups, enabling to chain groups together.

Prior versions of the OpenFlow specification had limited VLAN support, it only supported a single level of VLAN tagging with ambiguous semantic. The new tagging support has explicit actions to add, modify and remove VLAN tags, and can support multiple level of VLAN tagging. It also adds similar support the MPLS shim headers.

Prior versions of the OpenFlow specification assumed that all the ports of the OpenFlow switch were physical ports. Version 1.1 adds support for virtual ports, which can represent complex forwarding abstractions such as LAGs or tunnels.

Prior versions of the OpenFlow specification introduced the emergency flow cache as a way to deal with the loss of connectivity with the controller. The emergency flow cache feature was removed in this version of the specification, due to the lack of adoption, the complexity to implement it and other issues with the feature semantic.

This version of the specification adds two simpler modes to deal with the loss of connectivity with the controller. In fail secure mode, the switch continues operating in OpenFlow mode, until it reconnects to a controller. In fail standalone mode, the switch reverts to using normal processing (Ethernet switching).

20.3.7 OpenFlow v1.2 specification

This version of OpenFlow was released in December 2011 [19].

Prior versions of the OpenFlow specification used a static fixed length structure to specify `ofp_match`, which prevents flexible expression of matches and prevents inclusion of new match fields. The `ofp_match` has been changed to a TLV structure, called OpenFlow Extensible Match (OXM), which dramatically increases flexibility. The match fields themselves have been reorganised. In the previous static structure, many fields were overloaded; for instance, `tcp.src_port`, `udp.src_port`, and `icmp.code` were using the same field entry. Now, every logical field has its own unique type.

Prior versions of the OpenFlow specification were using hand-crafted actions to rewrite header fields. The Extensible `set_field` action reuses the OXM encoding defined for matches, and enables to rewrite any header field in a single action (EXT-13). This allows any new match field, including experimenter fields, to be available for rewrite. This makes the specification cleaner and eases cost of introducing new fields.

OpenFlow v1.2 supports IPv6. Added support for matching on IPv6 source address, destination address, protocol number, traffic class, ICMPv6 type, ICMPv6 code and IPv6 neighbor discovery header fields (EXT-1). Added support for matching on IPv6 flow label (EXT-36).

Since version 1.2, OpenFlow started to support multiple controllers for failover (EXT-39). This scheme is entirely driven by the controllers, so switches only need to remember the role of each controller to help the controller election mechanism.

20.3.8 OpenFlow v1.3 specification

OpenFlow v1.3 was released in 2012 [20].

Prior versions of the OpenFlow specification included limited expression of the capabilities of an OpenFlow switch. OpenFlow 1.3 include a more flexible framework to express capabilities (EXT-123). The main change is the improved description of table capabilities. Those capabilities have been moved out of the table statistics structure in its own request/reply message, and encoded using a flexible TLV format. This enables the additions of next-table capabilities, table-miss

flow entry capabilities and experimenter capabilities. Other changes include renaming the 'stats' framework into the 'multipart' framework to reflect the fact that it is now used for both statistics and capabilities, and the move of port descriptions into its own multipart message to enable support of a greater number of ports.

Prior versions of the OpenFlow specification included table configuration flags to select one of three 3 behaviour for handling table-misses (packet not matching any flows in the table). OpenFlow 1.3 replace those limited flags with the table-miss flow entry, a special flow entry describing the behaviour on table miss (EXT-108). The table-miss flow entry uses standard OpenFlow instructions and actions to process table-miss packets, this enables to use the full flexibility of OpenFlow in processing those packets. All previous behaviour expressed by the table-miss config flags can be expressed using the table-miss flow entry. Many new way of handling table-miss, such as processing table-miss with normal, can now trivially be described by the OpenFlow protocol.

Add support for per-flow meters (EXT-14). Per-flow meters can be attached to flow entries and can measure and control the rate of packets. One of the main applications of per-flow meters is to rate limit packets sent to the controller. The per-flow meter feature is based on a new flexible meter framework, which includes the ability to describe complex meters through the use of multiple metering bands, metering statistics and capabilities. Currently, only simple rate-limiter meters are defined over this framework. Support for color-aware meters, which support Diff-Serv style operation and are tightly integrated in the pipeline, was postponed to a later release.

Previous version of the specification introduced the ability for a switch to connect to multiple controllers for fault tolerance and load balancing. Per connection event filtering improves the multi-controller support by enabling each controller to filter events from the switch it does not want (EXT-120).

A new set of OpenFlow messages enables a controller to configure an event filter on its own connection to the switch. Asynchronous messages can be filtered by type and reason. This event filter comes in addition to other existing mechanisms that enable or disable asynchronous messages, for example the generation of flow-removed

events can be configured per flow. Each controller can have a separate filter for the slave role and the master/equal role.

In previous version of the specification, the channel between the switch and the controller is exclusively made of a single TCP connection, which does not allow exploiting the parallelism available in most switch implementations. OpenFlow 1.3 enables a switch to create auxiliary connections to supplement the main connection between the switch and the controller (EXT-114). Auxiliary connections are mostly useful to carry packet-in and packet-out messages.

In previous version of the specification, the final order of tags in a packet was statically specified. For example, a MPLS shim header was always inserted after all VLAN tags in the packet. OpenFlow 1.3 removes this restriction, the final order of tags in a packet is dictated by the order of the tagging operations, and each tagging operation adds its tag in the outermost position (EXT-121).

The logical port abstraction enables OpenFlow to support a wide variety of encapsulations. The tunnel-id metadata `OXM_OF_TUNNEL_ID` is a new OXM field that expose to the OpenFlow pipeline metadata associated with the logical port, most commonly the demultiplexing field from the encapsulation header (EXT-107). For example, if the logical port performs GRE encapsulation, the tunnel-id field would map to the GRE key field from the GRE header. After decapsulation, OpenFlow would be able to match the GRE key in the tunnel-id match field. Similarly, by setting the tunnel-id, OpenFlow would be able to set the GRE key in an encapsulated packet.

A cookie field was added to the packet-in message (EXT-7). This field takes its value from the flow the sends the packet to the controller. If the packet was not sent by a flow, this field is set to `0xffffffffffffff`. Having the cookie in the packet-in enables the controller to more efficiently classify packet-in, rather than having to match the packet against the full flow table.

In September 2012, an OpenFlow v1.3.1 was released. Prior versions of the OpenFlow specification included a simple scheme for version negotiation, picking the lowest of the highest version supported by each side. Unfortunately this scheme does not work properly in all cases, if both implementations don't implement all versions up to their

highest version, the scheme can fail to negotiate a version they have in common (EXT-157).

The main change is adding a bitmap of version numbers in the Hello messages using during negotiation. By having the full list of version numbers, negotiation can always negotiate the appropriate version if one is available. This version bitmap is encoded in a flexible TLV format to retain future extensibility of the Hello message.

OpenFlow v1.3.2 was released in April 2013. Changes, implemented in this version [16]:

- mandate in OXM that 0-bits in mask must be 0-bits in value (EXT-238);
- allow connection initiated from one of the controllers (EXT-252);
- add clause on frame misordering to spec (EXT-259);
- set table features doesn't generate flow removed messages (EXT-266);
- fix description of set table features error response (EXT-267);
- define use of generation_id in role reply messages (EXT-272);
- switch with only one flow table are not mandated to implement goto (EXT-280).

20.4 Work related analysis

The section is based on the analysis of work and publications of specialists from both industrial and scientific fields, e.g. Cisco [1], The University of Edinburgh (Scotland) [2]. These publications are aimed particularly at bringing in the idea of programmable network (University of Cambridge [5]), describing its architecture, building blocks, defining features, evolutionary aspects (Georgia Institute of Technology, Princeton University [6]) etc.

An insight about the predecessors of Software-defined Networking and enabling technologies has been build, in particular, on the basis of publications of The University of Toronto (Canada) [3], Massachusetts Institute of Technology (MIT, Cambridge) [4].

The publications by the fellows of the aforementioned institutions and organizations helped to build up the comprehensive picture on the topic of Software defined Networking basics.

Conclusion and questions

Thus, the following topics have been covered in given section:

- the architectural part of SDN – its predecessors and network virtualization aspects;
- an in-depth look at the peculiarities of SDN implementation has been taken – an accent on the differentiation between the control and data planes has been put – its defining role has been demonstrated;
- the versions of OpenFlow protocol specification have been considered.

1. Describe the trend that appeared during the evolution of networking.
2. Describe the layered structure of SDN architecture.
3. What is the reason for the emergence of programmable networks?
4. What are the main predecessors of SDN technology?
5. What is the general idea of virtualization? What are the benefits of a virtual network?
6. Name the fundamental characteristic of SDN.
7. Give the definition of the flow table.
8. How do the SDN-devices interact with each other using the flow table?
9. List existing software and hardware SDN devices.
10. Describe the concept of SDN controller. Characterize the structure of SDN controller.
11. List general characteristics of SDN controller.
12. Name some SDN controllers that you are aware of. Give a list of existing controllers.
13. What are the basic requirements for SDN controllers?
14. Why in-kernel controller provides better performance than analogues?
15. When did the first version of OpenFlow protocol appear?
16. List three fundamental packet paths inside the OpenFlow v1.0 switch.
17. Describe the difference between the OpenFlow control plane and the legacy control plane.
18. What does OpenFlow protocol consist of?

19. List and describe the characteristics of OpenFlow versions.
20. Name the first version of OpenFlow protocol specification with IPv6 protocol support.

References

1. P. Goransson and C. Black, *Software defined networks: A Comprehensive Approach*. Waltham, MA: Elsevier, 2014.
2. X. Foukas, M. K. Marina, and K. Kontovasilis, "Software Defined Networking Concepts," P.1-33, 2014. Available: <https://homepages.inf.ed.ac.uk/mmarina/papers/sdn-chapter.pdf>. [Accessed: Nov. 22, 2018].
3. A. T. Campbell, I. Katzela, K. Miki, and J. Vicente, "Open Signaling for ATM, INTERNET and Mobile Networks (OPENSIG'98)," *ACM SIGOPS Operating Systems Review*, vol. 33, no. 2, P. 15-28, 1999.
4. D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research," *IEEE Communications*, vol. 35, no. 1, P. 80-86, 1997.
5. J.E. van der Merwe, S. Rooney, I. Leslie, and S. Crosby, "The tempest-a practical framework for network programmability," *IEEE Network*, vol. 12, no. 3, P. 20-28, 1998.
6. N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, P. 87-98, 2014.
7. Problem Statement: Overlays for Network Virtualization, RFC 7364, 2014. Available: <https://tools.ietf.org/html/rfc7364>. [Accessed: Nov. 22, 2018].
8. J. Brodtkin, "Data Center Startups Emerging to Solve Virtualization and Cloud Problems," *Network World*, 2011. Available: https://www.pcworld.com/article/230297/Data_Center_Startups_Emerging_to_Solve_Virtualization_and_Cloud_Problems.html. [Accessed: Nov. 22, 2018].
9. "NetFlow Traffic Analyzer: NetFlow analyzer and bandwidth monitoring software," 2018. Available: <https://www.solarwinds.com/netflow-traffic-analyzer>. [Accessed: Nov. 22, 2018].
10. "Production Quality, Multilayer Open Virtual Switch," 2018. Available: <https://www.openvswitch.org/>. [Accessed: Nov. 22, 2018].
11. "Open thin switching, open for business," June 27, 2013. Available: <https://www.bigswitch.com/topics/introduction-of-indigo-virtual-switch-and-switch-light-beta>. [Accessed: Nov. 22, 2018].
12. OpenFlow Management and Configuration Protocol (OF-Config 1.1.1), Version 1.1.1, March 23, 2013. Available:

<https://www.opennetworking.org/wp-content/uploads/2013/02/of-config-1-1-1.pdf>. [Accessed: Nov. 22, 2018].

13. The Open vSwitch Database Management Protocol, RFC 7047, December 2013. Available: <https://tools.ietf.org/html/rfc7047>. [Accessed: Nov. 22, 2018].

14. “Learn REST: a RESTful tutorial,” 2018. Available: <https://www.restapitutorial.com/>. [Accessed: Nov. 22, 2018].

15. “Open Daylight: technical overview,” 2015. Available: <http://archive15.opendaylight.org/project/technical-overview>. [Accessed: Nov. 22, 2018].

16. OpenFlow Switch Specification, Version 1.3.2, April 25, 2013. Available: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.2.pdf>. [Accessed: Nov. 22, 2018].

17. OpenFlow Switch Specification, Version 1.0.0, December 31, 2009. Available: <https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf>. [Accessed: Nov. 22, 2018].

18. OpenFlow Switch Specification, Version 1.1.0 Implemented, February 28, 2011. Available: <https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/openflow-spec-v1.1.0.pdf>. [Accessed: Nov. 22, 2018].

19. OpenFlow Switch Specification, Version 1.2, December 5, 2011. Available: <https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.2.pdf>. [Accessed: Nov. 22, 2018].

20. OpenFlow Switch Specification, Version 1.3.0, June 25, 2012. Available: <http://www.cs.yale.edu/homes/yu-minlan/teach/csci599-fall12/papers/openflow-spec-v1.3.0.pdf>. [Accessed: Nov. 22, 2018].

21. SDN PROGRAMMING AND SIMULATION OF SDN COMPOSING, CONFIGURING AND SCALING

Dr. V. V. Shkarupylo (ZNTU)

Contents

Abbreviations	166
21.1 On the peculiarities of SDN switches and controllers functioning and implementation	167
21.1.1 Considering SDN as a system. Key components: controllers, switches, hosts	167
21.1.2 SDN infrastructure simulation and emulation. Network configuration and scaling.....	170
21.1.3 Network orchestration and virtualization. Simulation of data flows	170
21.2 Network programming and testing	172
21.2.1 Setting up the configuration of SDN network in Mininet environment.....	172
21.2.2 Testing the soundness and consistency of SDN infrastructure	174
21.2.3 Dataflow orchestration. SDN reconfiguration and scaling	178
21.3 SDN programming and Python scripting	178
21.4 Work related analysis	186
Conclusion and questions	187
References	189

Abbreviations

- AHP – Analytic Hierarchy Process
- API – Application Programming Interface
- CLI – Command-line Interface
- CPU – Central Processing Unit
- DISCO – Distributed SDN Control plane
- HTML – HyperText Markup Language
- MDSE – Model-Driven Software Engineering
- NFV – Network Functions Virtualization
- ODL – OpenDayLight
- ONF – Open Networking Foundation
- SDN – Software-defined Networking
- WG – Working Group

21.1 On the peculiarities of SDN switches and controllers functioning and implementation

These days the state of global networking can be characterized as being on the verge of global refinement [1]. Considering the number of nodes involved (billions), the questions of the convenience of network management, monitoring, control, scaling and reconfiguring are becoming more and more topical. In this context the paradigm of Software Defined Networking (SDN) can be considered as a plausible solution.

20.1.1 Considering SDN as a System. Key components: controllers, switches, hosts

Prior considering the SDN as a system, the fundamental principles of the SDN should be encompassed [2]:

- separation between the control and data planes – a single software control program manages multiple data planes;
- usage of lightweight switches;
- utilization of the concept of controller – to coordinate the lightweight switches in a centralized manner.

One of the main purposes of utilizing the SDN technology is the potential opportunity to foster the effectiveness of available network resources utilization to meet the rapidly changing requirements of the environment, e.g., the ad-hoc requirements taking place within business processes. This is achievable through software-oriented configuration and management of the network.

Classical switches are based on proprietary software, which fosters the interoperability problem, which is potentially resolvable by way of standardization [2]. Moreover, the cost and complexity of network management and support increase significantly. To this end, the industry has also embraced the SDN as the strategy to increase the functionality of the network while reducing the costs and hardware complexity [4].

Conventional network solutions utilize dedicated devices to control and monitor the data flow, e.g., Application Specific Integrated

Circuits (ASICs), designed for performing specific tasks [5]. The wider the range of rules the packages can be treated, the more expensive these devices become. The questions about scalability, security and reliability are still open though. It has been stated that current networks lack the flexibility to deal with different types of packets with various content [5]. Promising solution is in programmable manner of network configuration and control. Such approach allows create the agile solutions meeting permanently changing requirements in a sufficient manner. The available computing resources then getting more effectively allocated and distributed. This prompts the increase of the economic effect from the utilization of available resources. Thus, the company, which has adopted the SDN technology, is expected to be more relevant in modern highly competitive business-environment.

Nevertheless, the concept of programmable network is not the novelty of SDN – the experimentations have started with Active Networking in 1990s, and the concept of control plane has been introduced in 2000s by the IETF ForCES Working Group (WG) [7]. Those concepts haven't been implemented widely though.

It is stated that SDN concepts are grounded upon the ideas of telephony networks, where the differentiation between the data and control planes has also been applied.

Previous attempts to bring in the concepts of programmable networks are generalized in table 21.1 [10].

Table 21.1 – Previous technologies encapsulating the concepts of programmable networks

No.	Technology	Year	Key features
1	DCAN (Devolved Control of ATM Networks)	1990s	Scalable control and management of ATM networks. The idea – to decouple control and management functions from the devices (ATM switches) and assign them to a dedicated devices.
2	Active Networking	1990s	Is supposed to be a programmable infrastructure to provide customizable services, e.g., the concept of user programmable switches etc.
3	OPENSIG (Open Signaling)	1995	The idea – to make ATM, Internet and mobile networks more open, extensible, and programmable. It is proposed to achieve that through the open programmable interfaces.
4	4D Project	2004	Four planes have been distinguished: a “decision” plane preserving the global perception of network, “dissemination” plane and “discovery” plane devoted to manage the “data” plane. The later one is devoted to data transfer.
5	NETCONF	2006	The technology has been proposed by IETF Network Configuration Working Group as a protocol for network devices configuring through corresponding API (Application Programming Interface).
6	Ethane	2006	A predecessor of OpenFlow protocol. The concept of controller to manage network security and policies in a centralized manner has been utilized. Like in SDN, the concepts of controller and lightweight switch have been distinguished.

To sum up, grounding on the content of table 21.1, it can be stated that building blocks of SDN technology have been created previously, and the predecessors of OpenFlow protocol, centralized controller and lightweight switch can be seen in NETCONF and Ethane technologies.

21.1.2 SDN infrastructure simulation and emulation. Network configuration and scaling

Taking into consideration the scale and the complexity of SDN solutions, to be confident that certain SDN-devoted application is going to be functioning as supposed, the preliminary simulation needs to be conducted. It can be done with corresponding tools. One of such tools is “fs-sdn”, devoted to do the prototyping and evaluation of SDN-applications at a large scale [23].

Because of the fact that SDN technology is relatively new, it is commonly relatively difficult to work with such network directly. The solution can be found in different emulators usage. The emulator typically is a set of software and hardware means to represent SDN network within virtual environment. SDN software is based on Linux platform. Here are some examples of such emulators: Mininet [28], EstiNet [29], OpenNet [30], ns-3 [31]. Each of these solutions has its advantages and drawbacks. The Mininet emulator though is being frequently considered to be an exemplar to be compared to.

Mininet environment is devoted to be the mean for SDN-network emulation, particularly by creating virtual hosts, switches, controllers and connections between these components. Named components and connections between them form the topology of network.

21.1.3 Network orchestration and virtualization. Simulation of data flows

Certain controller of SDN is typically considered with respect to corresponding domain of applicability. Thus, to leverage the advantages of programmable networking on inter-domain level, the need for a more global concept arises. Here comes the concept

orchestrator, encompassing the topology of SDN network on inter-domain level.

It has been stated that there are plenty of different challenges to provide an orchestration (centralized control) of SDN networks over multiple domains, e.g., heterogeneous control planes, diverse transport technologies and communication protocols inside the domain [27]. The orchestrator is supposed to be supplied with an abstract view of inter-domain SDN-network, and each domain-specific controller is assumed to operate with the services of the following types [27]: provisioning (e.g., connections modification), topology discovery (export topology information), monitoring of the created connections, path computation. The controller is supposed to calculate the paths within the corresponding domain. On contrary, the orchestrator perceives the global inter-domain view, obtaining the required data from the aforementioned services of domain-related controllers.

The concept of orchestration is tightly bound with a concept of virtualization. Network virtualization is devoted to leverage the opportunity for the bodies to use the own controller and manage the available virtual resources [25].

To cope with a constant increase of data traffic and the number of different intercommunicating applications generating this traffic, the concept of Network Functions Virtualization (NFV) has arisen [9]. Named concept is devoted to foster the easiness of network management, granting the required level of QoS-parameters to all the bodies involved in an ad-hoc manner. To implement such an approach, the Cloud infrastructure is supposed to be utilized. Moreover, the NFV technology is considered to be not a replacement, but a complementary one to the SDN technology [9].

To increase network flexibility and programmability, the concepts of NFV and SDN are expected to be used in conjunction. To leverage the advantages of both NFV and SDN technologies, the HyperFlex architecture has been proposed: it provides the virtualization of control plane, by adding a control-plane isolation function [24]. Moreover, an SDN/NFV-enabled edge node for IoT services has been proposed [26].

21.2 Network programming and testing

A widespread and open implementation of SDN controller is known as ODL (OpenDayLight) built on MDSE (Model-Driven Software Engineering) principles [11], [12]. It is stated that ODL has already widely been deployed (over one hundred deployments) around the world, e.g., by Orange, China Mobile, AT&T, T-Mobile companies etc. [13].

Another well-known solution is known as Beacon [15]. The key features of this controller implementation are the following: cross-platform, open source. The advanced study on SDN controllers has been conducted in [16]. There are plenty of different other SDN/OpenFlow controllers, e.g., Floodlight [17], Maestro [19], Ryu [22] etc. The Floodlight controller has been used as the basis for open distributed SDN operating system – ONOS (Open Network Operating System) [18]. Moreover, Floodlight is stated to be the first implementation of SDN controller gaining the attention of both research and industry [18].

Nevertheless, the centralized nature of SDN provokes the scalability and reliability issues. To this end, the distributed SDN controllers have been proposed [14]. It is stated that centralized SDN controller provokes SDN network to be vulnerable to disruptions and attacks [20]. To diminish this drawback, an open Distributed SDN Control plane (DISCO) for multi-domain SDN networks has been proposed [20]. The multi-criteria decision making method – AHP (Analytic Hierarchy Process) – has been proposed to choose the best suitable SDN controller [21]. It has been stated that, on the basis of the research conducted, the best suitable SDN controller has been found to be Ryu [22].

21.2.1 Setting up the configuration of SDN network in Mininet environment

The Mininet as an emulator provides the means for controller testing. Mininet environment provides the means to conduct the development, investigation, testing and software configuring of SDN systems, etc.

Mininet provides in particular the following abilities:

- can be used as testbed for SDN applications development;
- brings to the table the ability of different developers to jointly work on network topology;
- includes the means of complex topology testing;

- provides specialized Application Programming Interface (API), oriented on Python programming language usage;

Comparing to typical approaches to virtualization, Mininet provides the following advantages:

- easiness of installation;
- quick boot time;
- easiness of system reconfiguration.

As a drawback the difficulties during the work with graphical environment of Mininet on Windows platform and also the limitation of network configuration by hardware resources available for virtual machine can be pointed out [28].

The tasks to be accomplished during the laboratory work:

- Mininet Linux-environment installation on Windows platform by way of VirtualBox usage;
- virtual machine network interfaces configuration;
- get in touch with basic console commands of Mininet emulator, particularly to create the networks with different topologies.

The presentation of accomplished tasks has to be conducted by one of two ways:

- one-by-one;
- after all the tasks have been accomplished.

Obtained results have to be properly represented in the report to be defended.

To set up SDN network configuration in Mininet environment, the following commands can be used:

- ```
> sudo mn --test pingall --topo single,3
> sudo mn --test pingall --topo linear,4
> sudo mn --topo tree,depth=1,fanout=2 --test pingall
```

First command creates the network with three hosts. Second command creates the liner topology network with four hosts. The final one creates the tree-like topology.

The `--topo tree` parameter sets tree topology itself. The `depth` attribute sets the amount of switches layers (one in our case, represented with single element (top) of switches tree): on the potential second layer there will be a pair of switches, on the third – four, and so on. The `fanout` attribute defines the number of connections to each switch. In our case `fanout=2`. This means that, taking into consideration that `depth=1` (there are no other

layers with switches and there are no other switches at all), both connections are the direct connections to hosts.

For instance, if we had depth=2, there would be one switch from the first layer connected to a pair of switches from the second layer, and those switches from the second layer would be directly connected to a pair of hosts each. That means that there would be three switches and four hosts in total.

The `--test pingall` parameter means that, after the creation of network with specified topology, each host should ping all other hosts to test network consistency.

The procedure of such network creation and testing is a pretty time consuming process, which will take place about 5 sec and will be shown in console log.

### ***21.2.2 Testing the soundness and consistency of SDN infrastructure***

To conduct the testing of SDN infrastructure, or a controller in particular, the Mininet environment is typically used.

Mininet network consists of the following components [47]:

- isolated hosts – each host is emulated as a group of user-level processes; each emulated host has its virtual Ethernet interface;
- emulated links – as each emulated host has its virtual Ethernet interface, the link is the representation of connection between the Ethernet interfaces of two hosts;
- emulated switches – can run in the kernel or in user space.

Mininet environment provides plenty of tools to test the soundness and consistency of SDN infrastructure. To this end, the concepts of controller switches and nodes are used. Some of named tools are given in Table 21.2.

As a result of last command execution, the HTML-code of webpage, obtained by client, will be shown on the console.

Table 21.2 – The tools (commands) to test the configuration of SDN network

| No. | Command                                                   | Description                                                                                                                                                                   |
|-----|-----------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1   | > nodes                                                   | See information about all network nodes (there are should be four nodes in total – controller (c0), switch (s1) and pair of hosts (h1, h2)) the nodes command should be used. |
| 2   | > net                                                     | See the information about nodes' interfaces.                                                                                                                                  |
| 3   | > dump                                                    | See the information about nodes configuration.                                                                                                                                |
| 4   | > h1 ifconfig -a                                          | See information about network interfaces of specified node. For instance, for h1 node the given command should be executed.                                                   |
| 5   | > s1 ps -a                                                | Check the information about processes executed on nodes. For instance, for s1 node the following command should be executed.                                                  |
| 6   | > h1 ping -c 1 h2                                         | Check the connections between two given hosts.                                                                                                                                |
| 7   | > pingall                                                 | Check the connections between all pairs of hosts.                                                                                                                             |
| 8   | > h1 python -m SimpleHTTPServer 80 &<br>> h2 wget -O - h1 | Launch web server and appropriate client on hosts. Web server will be launched on h1 node, client – on h2 node.                                                               |

To expand the demonstrativeness of resulting solutions and to simplify the process of network creation, configuration and testing, the corresponding MiniEdit graphical interface can be used [47].

The snapshot of MiniEdit workspace is given in Fig. 21.1.

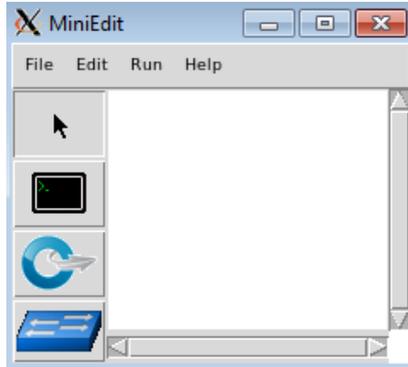


Fig. 21.1 – MiniEdit workspace

The representation of SDN network created in MiniEdi environment is given in Fig. 21.2.

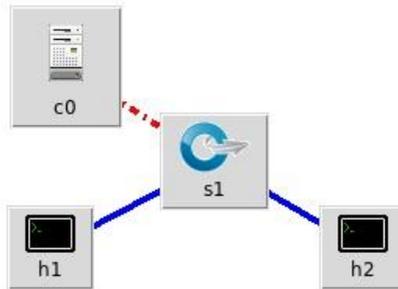


Fig. 21.2 – SDN network

In Fig. 21.2 the network with one controller (c0), one switch (s1) and pair of hosts (h1 and h2) is represented.

The configuration details of c0 are given in Fig. 21.3.

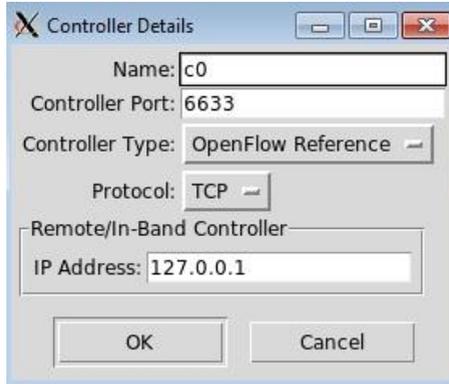


Fig. 21.3 – Controller configuration

The version of OpenFlow protocol to be used is assigned in network preferences (Fig. 21.4).

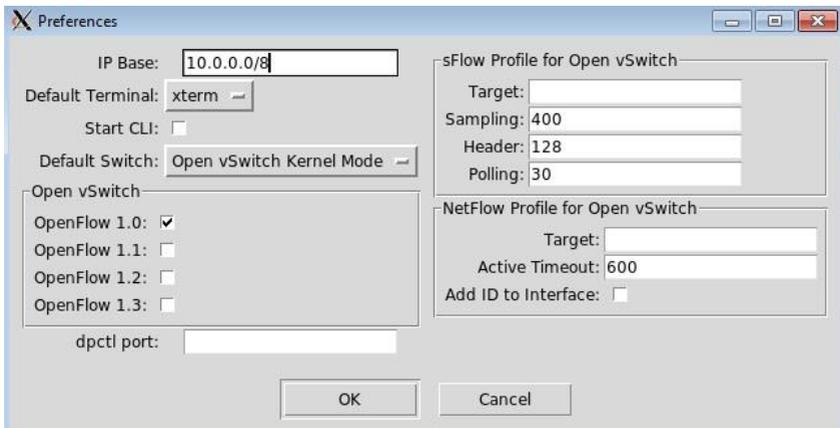


Fig. 21.4 – Network preferences

In Fig. 21.4, it is assigned that OpenFlow 1.0 is used.

### ***21.2.3 Dataflow orchestration. SDN reconfiguration and scaling***

Modern data centers usage scenarios are commonly based on virtual machines and virtual resources utilization in cloud environment. To improve the resources utilization and data exchange in cloud environment, different strategies for dynamic balancing of dataflow are applied. The need for a dataflow orchestration arises here. Diverse approaches have been proposed to date. One of those is all about the architectural design of SDN-based orchestrator for dynamic communication and computing resources chaining [33]. It provides a coordinated chaining of network and computing data centers' services, fostering the increase of allocated resources utilization. Moreover, SDN and NFV technologies are considered to be the enabling mechanisms to bring into life the integration of cloud and network resources. With respect to 5G services usage domain, the ADRENALINE testbed (placed in Barcelona, Spain) has been utilized to demonstrate the soundness of SDN orchestration as feasible and scalable solution for providing the end-to-end connectivity between heterogeneous networks and cloud systems [34].

With respect to heterogeneous wireless networks (DenseNets), the requirement to scale and reconfigure the existing network infrastructure to fulfill the dynamically changing traffic requirements arises, as energy consumption and signaling overhead increase. In order to maximize the number of devices involved to be served simultaneously and, at the same time, to minimize the total energy consumption while reducing the costs for service providers, the CROWD architecture has been proposed [35].

### **21.3 SDN programming and Python scripting**

Covering the aspects of SDN programming, the underlying API needs to be taken into consideration. Nowadays, this API is represented with OpenFlow specification, based on Ethernet switch with an internal flow table, has been proposed [3]. To this end, it is essential to discuss the aspects of OpenFlow protocol first. Not to mention that OpenFlow protocol has been originally devoted to allow the researchers run the experiments on heterogeneous switches in a uniform way. An OpenFlow switch includes one or more tables of packet-handling rules.

Each particular rule is aimed at certain portion of traffic. Depending on traffic properties, the package-related actions can either be dropping, forwarding or flooding. Depending on the rule, imposed by controller software, the switches can act in a different manner – as a router, switch, firewall etc. [2].

### ***21.3.1 An in-depth look at SDN-related programming approaches, principles and concepts***

Prior covering the programming peculiarities, to bring to the table the possibility of OpenFlow-based communication of controller with switches, the following idea has to be previously exploited: the majority of modern Ethernet switches and routers utilize the flow tables. The OpenFlow protocol is intended to provide an interface to program the heterogeneous switches and routers [3].

An OpenFlow protocol describes the rules of SDN-compatible switches intercommunication. The protocol is described within an OpenFlow specification hosted by an Open Networking Foundation (ONF) [6].

An OpenFlow switch is built from the following constituents [3]:

- Flow Table, coupled with the actions associated with each flow entry telling the switch the mechanism of flow processing;
- Secure Channel connecting the switch with a remote control process – the controller;
- OpenFlow Protocol – an open standard for switch-controller interaction.

Controller-related applications are devoted to run on a network operating system. The fundamental idea of SDN is to shift the computations consuming routing tasks from the hardware layer to a software-based controller [4].

Utilization of the OpenFlow gives the following opportunities [4]:

- ability to manage multiple switches from a single controller;
- capability to analyze traffic statistics;
- forwarding information can be updated dynamically.

A brief list of OpenFlow-compliant switches is given in table 21.3.

Table 21.3 – OpenFlow-compliant switches

| Company | Switch series                                                                                                                                                                                        |
|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| HP      | FlexFabric 12900, FlexFabric 11900, 8200 zl, 5920, HP FlexFabric 5700, 5500 EI, 5400 zl, 3800, 2920, 12500, 10500, HP FlexFabric 5930, 5900, 5500 HI, HP 5400R zl2, HP 5130 EI, HP 3500 and 3500 yl; |
| Cisco   | Cisco Catalyst 2960X/XR, Nexus 3000, Nexus 9000 etc.;                                                                                                                                                |
| Dell    | S4810, S4820T, S5000, S6000, Z9000, Z9500, MXL;                                                                                                                                                      |
| NEC     | PF5240, PF5820, PF6800, PF5248, PF5340, QX-S5200, QX-S4100.                                                                                                                                          |

Despite of representatives, given in Table 21.3, there are plenty of other proprietary solutions though.

Since its debut in 2009, an OpenFlow specification has changed significantly. The latest OpenFlow version is 1.5.1 [6]. The evolution of OpenFlow releases is given in table 21.4 [8].

Table 21.4 – Evolution of OpenFlow protocol

| No | Version | Distinctive Feature | Goal                                            |
|----|---------|---------------------|-------------------------------------------------|
| 1  | 1.0-1.1 | Multiple table      | Avoid flow entry explosion.                     |
|    |         | Group table         | Enable actions applying to the groups of flows. |
| 2  | 1.1-1.2 | Multiple controller | Load balancing, scalability.                    |
|    |         | OXM Match           | Extend matching flexibility.                    |
| 3  | 1.2-1.3 | Table miss entry    | Provide flexibility.                            |
|    |         | Meter table         | Add QoS and DiffServ capability.                |
| 4  | 1.3-1.4 | Bundle              | Enhance switch synchronization.                 |
|    |         | Synchronized table  | Enhance table scalability.                      |
| 5  | 1.4-1.5 | Scheduled bundle    | Enhance switch synchronization.                 |
|    |         | Egress Table        | Enabling processing to be done in output port.  |

The timeline of OpenFlow evolution is given in Fig. 21.5.

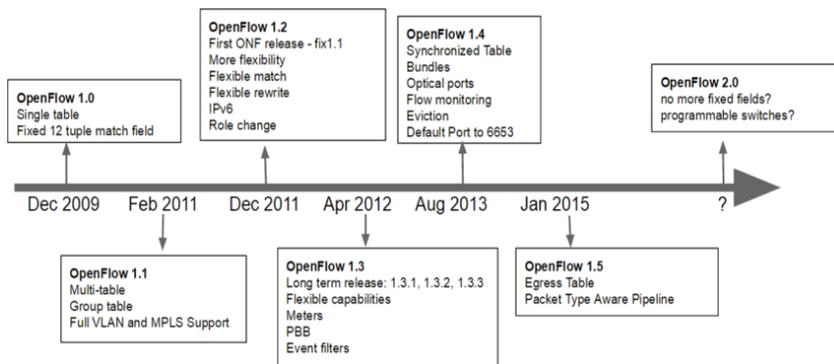


Fig. 21.5 – A timeline of OpenFlow evolution

In Fig. 21.5, in the initial OpenFlow 1.0 version (back in 2009) there has been only a single flow table. This fact stipulated the lack of flexibility due to the limited matching capabilities. Corresponding OpenFlow 1.0-compliant switches was able to perform only a single operation during the packet forwarding. This caused the flow entry explosion problem [8]. To this end, in the OpenFlow 1.1 version, the multiple tables and a group table have been introduced, and so forth.

The logical structure of OpenFlow-compliant switch:

- one or more flow tables;
- group table;
- OpenFlow channels to an external controller.

Flow tables and a group table are devoted to lookup and forward the packets.

Group table consists of the group entries, which can be grouped as follows [8]:

- general – execute all action buckets in the group;
- selecting – execute one action bucket in the group;
- indirect – execute only the defined action bucket in the group;
- failover – execute first live action bucket.

The structure of an OpenFlow-compliant switch with respect to specification is given in Fig. 21.6.

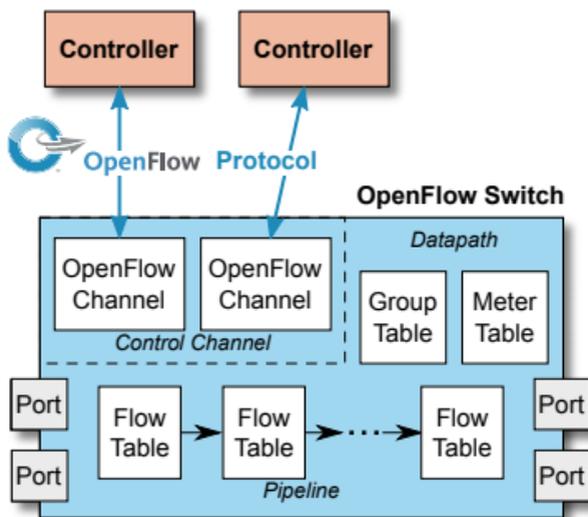


Fig. 21.6 – The structure of OpenFlow-compliant switch

The “general” group provides multicasting – packets are forwarded to multiple ports.

The “selecting” group is characterized as allowing the load balancing and link aggregation.

The “indirect” group fosters the scalability aspects by categorizing the flows into froups to increase the efficiency of default routing in particular [8].

The “failover” group detects thelive action bucket to execute, aiming the aspect of high availability.

With respect to an OpenFlow protocol, the adding, updating and deleting of flow entries is conducted by a controller.

Each flow table in the switch contains a set of flow entries; each flow entry consists of match fields, counters, and a set of instructions to apply to matching packets [6]. Matching starts at first flow table and may proceed to additional flow tables, as shown in Fig. 21.7 [6].

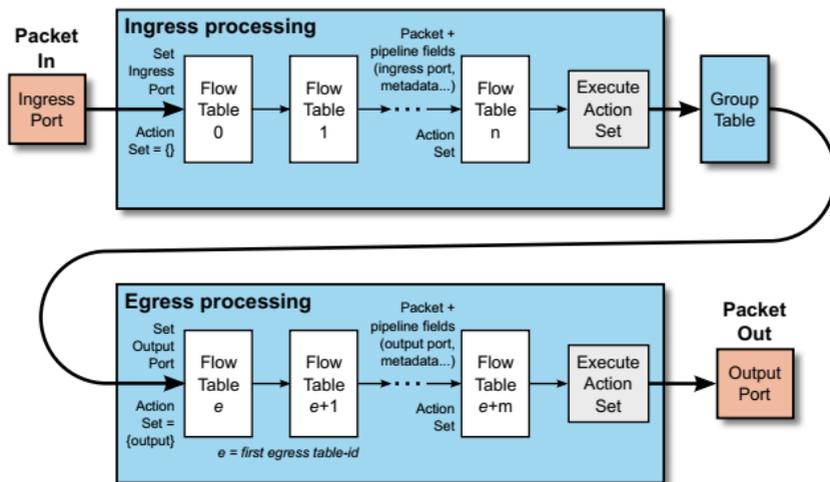


Fig. 21.7 – Packet processing pipeline

The OpenFlow switch protocol is grounded upon a set of structures and implemented on the basis of messages transferred through an OpenFlow channels. All OpenFlow messages are sent in big-endian format [6].

All OpenFlow-messages begin with a header, given in Table 21.5.

Table 21.5 – The structure of OpenFlow message header

| No. | Code line                                                                                                                            |
|-----|--------------------------------------------------------------------------------------------------------------------------------------|
| 1   | struct ofp_header {                                                                                                                  |
| 2   | uint8_t version; /* OFP_VERSION. */                                                                                                  |
| 3   | uint8_t type; /* One of the OFPT_constants. */                                                                                       |
| 4   | uint16_t length; /* Length including this ofp_header. */                                                                             |
| 5   | uint32_t xid; /* Transaction id associated with this packet. Replies use the same id as was in the request to facilitate pairing. */ |
| 6   | };                                                                                                                                   |
| 7   | OFP_ASSERT(sizeof(struct ofp_header) == 8);                                                                                          |

In Table 21.5 the length field contains the length of message.

Covering the approaches to SDN-programming, it is essential to point out first that existing SDN controllers offer programmers low-level programming interfaces [36]. Taking into consideration the complexity of such networks, there is a need for highly abstract and modular solutions, simplifying the programming of complex systems. To this end, modular approach to SDN programming on the basis of Pyretic language has been proposed [36]. This language allows write highly abstract policies for packets routing.

Another approach is to conduct the programming in an algorithmic manner. To this end, the Maple system has been proposed [37]. It allows the programmer to manipulate with the behaviors of entire network in an algorithmic manner. Maple also includes efficient multi-core scheduler, scaling efficiently to the controllers with multiple cores.

The evolutionary aspects of OpenFlow protocol are covered in P4 language [38]. Developers define the following goals to be reached:

- programmers should be able to change the way switches process the packets once deployed;
- programmers should be able to describe packets processing functionality independently of underlying hardware;
- switches shouldn't be bound with any specific network protocol.

There are plenty of different other SDN programming languages, e.g., Flog (logic programming approach) [39], Procera (a language for high-level reactive network control) [40] etc.

Moreover, an exhaustive survey on SDN programming languages has already been conducted [41].

A bit different story is the programming of SDN controllers. A comparative analysis of open-source OpenFlow-compliant controllers has been conducted in [16]:

- NOX – multi-threaded C++ based controller [42];
- POX – single-threaded Python based controller [43];
- OpenDaylight – written in Java [11];
- Beacon – multi-threaded Java-based controller [44];
- etc.

When considering the aspects of SDN configuration, with automation in mind, the Python scripting is applicable.

### 21.3.2 Setting up SDN configuration by way of python scripting

The forthcoming text corresponds to Python programming in Mininet environment. To this end, corresponding Mininet Python API has been created [45]. The interface is based on “topo” namespace, providing the means to set up or reconfigure network topology.

Moreover, Python scripting can be successfully used to expand the existing CLI (Command-line Interface) of Mininet. Corresponding script is given in Table 21.6 [46].

Table 21.6 – Python script

| No. | Python code                                                 |
|-----|-------------------------------------------------------------|
| 1   | def mycmd( self, line ):                                    |
| 2   | "mycmd is an example command to extend the Mininet CLI"     |
| 3   | net = self.mn                                               |
| 4   | output( 'mycmd invoked for', net, 'with line', line, '\n' ) |
| 5   | CLI.do_mycmd = mycmd                                        |

The commands, adding created mycmd command to Mininet CLI, are given in Table 21.7.

Table 21.7 – Adding of created command to Mininet CLI

| No. | Command line code                                                                      |
|-----|----------------------------------------------------------------------------------------|
| 1   | sudo mn --custom mycmd.py -v output                                                    |
| 2   | mininet> help mycmd                                                                    |
| 3   | output: mycmd is an example command to extend the Mininet CLI                          |
| 4   | mininet> mycmd bar                                                                     |
| 5   | output: mycmd invoked for <mininet.net.Mininet object at 0x7fd7235fb9d8> with line bar |

More sophisticated tips on Python scripting are given below.

### 21.3.3 Sophisticating the Python scripting. Bringing in the automation

A fragment of Python-script, setting up the performance parameters of SDN network, is given in Table 21.8.

Table 21.8 – A sample of configurational Python script

| No. | Python code                                                                                                                                                                               |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1   | <code>#!/usr/bin/python</code>                                                                                                                                                            |
| 2   | <code>from mininet.topo import Topo</code>                                                                                                                                                |
| 3   | <code>from mininet.net import Mininet</code>                                                                                                                                              |
| 4   | <code>from mininet.node import CPULimitedHost</code>                                                                                                                                      |
| 5   | <code>from mininet.link import TCLink</code>                                                                                                                                              |
| 6   | <code>from mininet.util import dumpNodeConnections</code>                                                                                                                                 |
| 7   | <code>from mininet.log import setLogLevel</code>                                                                                                                                          |
| 8   | <code>class SingleSwitchTopo( Topo ):</code>                                                                                                                                              |
| 9   | <code>"Single switch connected to n hosts."</code>                                                                                                                                        |
| 10  | <code>def build( self, n=2 ):</code>                                                                                                                                                      |
| 11  | <code>switch = self.addSwitch( 's1' )</code>                                                                                                                                              |
| 12  | <code>for h in range(n):</code>                                                                                                                                                           |
| 13  | <code># Each host gets 50%/n of system CPU</code><br><code>host = self.addHost( 'h%s' % (h + 1), cpu=.5/n )</code>                                                                        |
| 14  | <code># 10 Mbps, 5ms delay, 2% loss, 1000 packet queue</code><br><code>self.addLink( host, switch, bw=10, delay='5ms', loss=2,</code><br><code>max_queue_size=1000, use_htb=True )</code> |
| 15  | <code>...</code>                                                                                                                                                                          |

In Table 21.8, the method `addHost` is used to allocate available CPU resource to the virtual host. The `addLink` method is used to set up a bidirectional link with a specified bandwidth, delay, packets loss and max queue size characteristics.

## 21.4 Work related analysis

The section is based on the analysis of work and publications of fellows from Georgia Institute of Technology and Princeton University to provide the background on network programmability [2].

To characterize SDN controllers, the work of the fellows from Stanford University [15], Moscow State University [16] etc., has been analyzed. To differentiate between the controllers, the results of the comparative analysis, conducted by the fellows from the Fraunhofer

Institute for Secure Information Technology (Darmstadt, Germany), have been utilized [21].

An information on open distributed SDN operating system has been provided on the basis of work of the fellows from Open Networking Laboratory [18].

### ***Conclusion and questions***

Thus, in given lecture material, the following topics have been covered:

- peculiarities of SDN switches and controllers functioning and implementation – the aspects of SDN infrastructure simulation and emulation; the aspects of network orchestration and virtualization;
- SDN controller programming and testing, e.g., setting up the configuration of SDN network in Mininet environment;
- SDN programming and Python scripting. The aspects of automation have been described.

1. Briefly describe current state of the global network.
2. Characterize existing drawbacks of regular network – in terms of management, reconfiguration and interoperability.
3. Characterize the concept of interoperability. What are the solutions to achieve the latter?
4. Describe the fundamental principles of Software Defined Networking.
5. What is the use of differentiation between the control and data planes.
6. The purpose of OpenFlow protocol usage.
7. The constituents of OpenFlow-enabled switch.
8. Describe the outcomes of OpenFlow utilization.
9. Name a couple of OpenFlow-compliant switches.
10. Characterize the evolution of OpenFlow specification. Point out the distinctive features of releases.
11. Provide a brief overview of SDN predecessors.
12. Name the key concepts of DCAN technology.
13. Name the key concepts of Active Networking technology.
14. Name the key concepts of OPENSIG technology.

15. Name the key concepts of 4D Project.
16. The purpose of NETCONF protocol usage.
17. Describe the main idea of Ethane protocol.
18. Describe the peculiarities of OpenDaylight protocol.
19. Give a brief review of popular SDN controllers.
20. Explain the advantages of distributed SDN controllers usage.
21. Explain the drawbacks of the centralized SDN controllers.
22. Describe the concept of Network Function Virtualization (NFV).
23. Describe the idea of HyperFlex architecture.
24. Describe the advantages the virtualization brings in.
25. Characterize the challenges to providing the orchestration in SDN environment.
26. Differentiate between the concepts of controller and orchestrator – in terms of domains.
27. Substantiate the use of “depth” and “fanout” parameters during the creation of network with tree topology. Characterize the impact of these parameters values on total number of network nodes.
28. Provide a brief list of SDN emulators.
29. Characterize the use of Mininet emulator.
30. Describe the use of “--topo tree” parameter during network configuration.
31. Describe the use of “--test pingall” parameter during network configuration.
32. Describe the use of “depth=1” parameter during network configuration.
33. Name and briefly characterize the commands for checking the soundness and consistency of SDN infrastructure.
34. Substantiate the use of “nodes”, “net” and “dump” commands.
35. Describe the expediency of dataflow orchestration.
36. Name the spheres of dataflow orchestration applicability.
37. Substantiate the expediency of SDN networks reconfiguration and scaling.
38. Briefly characterize the approaches to SDN programming.
39. Describe the use of Pyretic programming language.
40. Give the idea of Maple system.

41. Name the evolutional perspectives of SDN-compliant switches programming.
42. Provide a brief review of the approaches to SDN programming.
43. Name and briefly characterize a couple of languages for SDN programming.
44. Briefly characterize open-source OpenFlow-compliant controllers.
45. Describe the scenarios of Python scripting language applicability in Mininet environment.
46. Describe the use of MiniEdit graphical interface.
47. Name the components of Mininet network.

### ***References***

1. V. Shkarupylo, S. Skrupsky, A. Oliinyk, and T. Kolpakova, "Development of stratified approach to software defined networks simulation," *Eastern-European Journal of Enterprise Technologies. Information and controlling systems*, vol. 5 no. 9, P. 67–73, 2017.
2. N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44 no. 2, P. 87–98, 2014.
3. N. McKeown et al., "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communications Review*, vol. 38, no. 2, P. 69–74, 2008.
4. A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using OpenFlow: a survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, 493–512, 2014
5. F. Hu, Q. Hao, and K. Bao, "A survey on Software-Defined Network and OpenFlow: from concept to implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, 2181–2206, 2014.
6. Open Networking Foundation. OpenFlow Switch Specification, Version 1.5.1, 2015. Available: <https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf>. [Accessed: 11 Nov. 2018].
7. J. Tourrilhes, P. Sharma, S. Banerjee, and J. Pettit, "SDN and openflow evolution: A standards perspective," *Computer*, vol. 47, no. 11, P. 22–29, Nov. 2014.
8. C. Ching-Hao and Y. Lin, OpenFlow Version Roadmap, 2015. Available: [http://speed.cis.nctu.edu.tw/~ydlin/miscpub/indep\\_frank.pdf](http://speed.cis.nctu.edu.tw/~ydlin/miscpub/indep_frank.pdf). [Accessed: 11 Nov. 2018].

9. J. Costa-Requena et al., “SDN and NFV integration in generalized mobile network architecture,” in 2015 European Conference on Networks and Communications (EuCNC), Paris, France, 29 June–2 July 2015, P. 154–158.

10. M. Bindhu and G. Ramesh, “The journey to SDN: a peek into the history of programmable networks,” *International Journal of Computer Science and Engineering Communications*, vol. 2, no. 5, P. 500–506, 2014.

11. J. Medved, R. Varga, A. Tkacik, and K. Gray, “OpenDaylight: Towards a Model-Driven SDN Controller architecture,” in 2014 IEEE 15th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM) (WOWMOM), Sydney, Australia, 19 June 2014, P. 1–6.

12. Z. K. Khattak, M. Awais, and A. Iqbal, “Performance evaluation of OpenDaylight SDN controller,” in 2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), Hsinchu, Taiwan, 16–19 Dec. 2014, P. 671–676.

13. J. H. Cox et al., “Advancing Software-Defined Networks: a survey,” *IEEE Access*, vol. 5, P. 25487–25526, 2017.

14. A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, “Towards an elastic distributed SDN controller,” in Second ACM SIGCOMM workshop on Hot topics in software defined networking (HotSDN '13), Hong Kong, China, 16 August 2013, P. 7–12.

15. D. Erickson, “What is Beacon?,” 2013. Available: <https://openflow.stanford.edu/display/Beacon/Home>. [Accessed: 11 Nov. 2018].

16. A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, “Advanced study of SDN/OpenFlow controllers,” in 9th Central & Eastern European Software Engineering Conference in Russia, Moscow, Russia, 24–25 October 2013, P. 1–6.

17. Project Floodlight: Open Source Software for Building Software-Defined Networks, 2018. Available: <http://www.projectfloodlight.org/>. [Accessed: 11 Nov. 2018].

18. P. Berde et al., “ONOS: towards an open, distributed SDN OS,” in Third workshop on Hot topics in software defined networking (HotSDN 2014), Chicago, Illinois, USA, 22 August 2014, P. 1–6.

19. C. Zheng, “Maestro: Achieving scalability and coordination in centralized network control plane,” 2012. Available: <https://scholarship.rice.edu/handle/1911/70214>. [Accessed: 11 Nov. 2018].

20. K. Phemius, M. Bouet, and J. Leguay, “DISCO: Distributed multi-domain SDN controllers,” in 2014 IEEE Network Operations and Management Symposium (NOMS 2014), Krakow, Poland, 5–9 May 2014, P. 1–4.

21. R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou, “Feature-based comparison and selection of Software Defined Networking (SDN) controllers,”

in 2014 World Congress on Computer Applications and Information Systems (WCCAIS 2014), Hammamet, Tunisia, 17–19 Jan. 2014, P. 1–7.

22. M. Monaco, O. Michel, and E. Keller, “Applying operating system principles to SDN controller design,” in Twelfth ACM Workshop on Hot Topics in Networks, College Park, Maryland, 21–22 November 2013, P. 1–7.

23. M. Gupta, J. Sommers, and P. Barford, “Fast, accurate simulation for SDN prototyping,” in The second ACM SIGCOMM workshop on Hot topics in software defined networking, Hong Kong, China, 16 August 2013, P. 31–36.

24. A. Blenk, A. Basta, and W. Kellerer, “Hyperflex: an SDN virtualization architecture with flexible hypervisor function allocation,” in IFIP/IEEE International Symposium on Integrated Network Management (IM), Ottawa, ON, Canada, 11–15 May 2015, P. 397–405.

25. A. Basta, A. Blenk, H. B. Hassine, and W. Kellerer, “Towards a dynamic SDN virtualization layer: control path migration protocol,” in 2015 11th International Conference on Network and Service Management (CNSM), Barcelona, Spain, 9–13 Nov. 2015, P. 354–359.

26. R. Vilalta et al., “End-to-end SDN orchestration of IoT services using an SDN/NFV-enabled edge node,” in 2016 Optical Fiber Communications Conference and Exhibition (OFC), Anaheim, CA, USA, 20–24 March 2016, P. 1–3.

27. V. Lopez et al., “Demonstration of SDN orchestration in optical multi-vendor scenarios,” in 2015 Optical Fiber Communications Conference and Exhibition (OFC), Los Angeles, CA, USA, 22–26 March 2015, P. 1–3.

28. F. Ketikci and S. Askar, “Emulation of Software Defined Networks using Mininet in different simulation environments,” in 2015 6th International Conference on Intelligent Systems, Modelling and Simulation, Kuala Lumpur, Malaysia, 9–12 February 2015, P. 205–210.

29. S.-Y. Wang, “Comparison of SDN OpenFlow network simulator and emulators: EstiNet vs. Mininet,” in 2014 IEEE Symposium on Computers and Communication (ISCC), Funchal, Portugal, 23–26 June 2014, P. 1–6.

30. M.-C. Chan et al., “OpenNet: A simulator for software-defined wireless local area network,” in 2014 IEEE Wireless Communications and Networking Conference, Istanbul, Turkey, 6–9 April 2014, P. 3332–3336.

31. J. Ivey, H. Yang, C. Zhang, and G. Riley, “Comparing a Scalable SDN simulation framework built on ns-3 and DCE with existing SDN simulators and emulators,” in 2016 annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation, Banff, Alberta, Canada, 15–18 May 2016, P. 153–164.

32. A. Wang, Y. Guo, F. Hao, T. V. Lakshman, and S. Chen, “Scotch: Elastically Scaling up SDN Control-Plane using vSwitch based Overlay,” in 10th ACM International Conference on emerging Networking Experiments and Technologies, Sydney, Australia, 2–5 Dec. 2014, P. 403–414.

33. B. Martini et al., “An SDN orchestrator for resources chaining in Cloud data centers,” in 2014 European Conference on Networks and Communications (EuCNC), Bologna, Italy, 23–26 June. 2014, P. 1–5.

34. R. Vilalta, A. Mayoral, R. Casellas, R. Martinez, and R. Muoz, “Experimental demonstration of distributed multi-tenant cloud/fog and heterogeneous sdn/nfv orchestration for 5g services,” in 2016 European Conference on Networks and Communications (EuCNC), Athens, Greece, 27–30 June 2016, P. 52–56.

35. S. Aurox, M. Draxler, A. Morelli, and V. Mancuso, “Dynamic network reconfiguration in wireless densenets with the crowd sdn architecture,” in 2015 European Conference on Networks and Communications (EuCNC), Paris, France, 29 June–2 July 2015, P. 144–148.

36. J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, “Modular SDN programming with Pyretic,” *Programming*, vol. 38, no. 5, P. 40–47, 2013.

37. A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, “Maple: simplifying SDN programming using algorithmic policies,” *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, P. 87–98, 2013.

38. P. Bosshart et al., “P4: programming protocol-independent packet processors,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, P. 87–95, 2014.

39. N. P. Katta, J. Rexford, and D. Walker, “Logic Programming for Software-Defined Networks,” in *First International Workshop on Cross-model Language Design and Implementation*, Copenhagen, Denmark, 9 September 2012, P. 1–3.

40. A. Voellmy, H. Kim, and N. Feamster, “Procera: a language for high-level reactive network control,” in *The first workshop on Hot topics in software defined networks (HotSDN '12)*, ACM, New York, NY, USA, 13 August 2012, P. 43–48.

41. C. Trois, M. D. Del Fabro, L. C. E. de Bona, and M. Martinello, “A survey on SDN programming languages: toward a taxonomy,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, 2687–2712, 2016.

42. N. Gude et al. “NOX: towards an operating system for networks,” *SIGCOMM Computer Communication Review*, vol. 38, no. 3, P. 105–110, 2008.

43. L. R. Prete, A. A. Shinoda, C. M. Schweitzer, and R. L. S. de Oliveira, “Simulation in an SDN network scenario using the POX controller,” in 2014 IEEE Colombian Conference on Communications and Computing (COLCOM), Bogota, Colombia, 4–6 June 2014, P. 1–6.

44. D. Erickson, “The beacon openflow controller,” in *Second ACM SIGCOMM workshop on Hot topics in software defined networking (HotSDN '13)*, ACM, New York, NY, USA, Hong Kong, China, 16 August 2013, P. 13–18.

45. Mininet Python API reference manual, 27 Aug. 2018. Available: <http://mininet.org/api/index.html>. [Accessed: 11 Nov. 2018].

46. Introduction to Mininet, 26 Sep. 2018. Available: <https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#custom>. [Accessed: 11 Nov. 2018].

47. How to use MiniEdit, Mininet's graphical user interface, 2 April 2015. Available: <http://www.brianlinkletter.com/how-to-use-miniedit-mininets-graphical-user-interface/>. [Accessed: 11 Nov. 2018].

## **22. ALGORITHMS AND APPLICATIONS FOR UTILIZATION OF SDN TECHNOLOGY TO IOT**

DrS., Prof. I. S. Skarga-Bandurova, Ph.D. student M. V. Nesterov,  
PhD student A. Y. Velykzhanin (V. Dahl EUNU)

### ***Contents***

|                                                                                        |     |
|----------------------------------------------------------------------------------------|-----|
| Abbreviations .....                                                                    | 195 |
| 22.1 Managing the IoT with SDN .....                                                   | 196 |
| 22.1.1 SLA management .....                                                            | 197 |
| 22.1.2 Metrics .....                                                                   | 197 |
| 22.2 Smart routing and scheduling .....                                                | 198 |
| 22.2.1 Data streaming over SDN .....                                                   | 201 |
| 22.2.2 Metrics for evaluation performance of QoS routing algorithms<br>.....           | 202 |
| 22.2.3 QoS routing algorithms applicable to large-scale SDN .....                      | 203 |
| 22.2.4 Traffic scheduling algorithms .....                                             | 205 |
| 22.3 Optimization of SDN Traffic Flow for IoT .....                                    | 206 |
| 22.3.1 Algorithms for calculating the optimal position of the SDN-<br>controller ..... | 206 |
| 22.3.2 Balancing algorithms in IoT-based software defined networks<br>.....            | 210 |
| 22.3.3 Algorithms for finding the optimal path in SDN networks ....                    | 218 |
| 22.4 SDN Performance prediction .....                                                  | 219 |
| 22.4.1 Algorithms performance metrics .....                                            | 219 |
| 22.4.2 An overall approach to detect and diagnose failures in SDN .                    | 220 |
| 22.4.3 Case study .....                                                                | 223 |
| 22.5 Work related analysis .....                                                       | 234 |
| Conclusions and questions .....                                                        | 235 |
| References .....                                                                       | 237 |

### ***Abbreviations***

BWP – Band Width Proportion  
DORA – Dynamic Online Routing Algorithm  
FIBs – Forwarding Information Base  
IoT – Internet of things  
I/O – input / output  
ISP – Internet Service Provider  
ML – machine learning  
MHA – Minimum Hop Algorithm  
MIRA – Minimum Interference Routing Algorithm  
NFV – Network Function Virtualization  
PPV – Path Potential Values  
QoE – Quality of Experience  
QoS – Quality of Service  
OSPF – Open Shortest Path First  
RIP – Routing Information Protocol  
ROADM – Reconfigurable Optical Add Drop Multiplexer  
SDN – Software Defined Networks  
SD – source-destination  
SLA – Service Level Agreements  
SLO – Service Level Objectives  
SPF – Shortest path first  
SWP – Shortest Widest Path algorithm  
TE – Traffic Engineering  
VM – virtual machine  
WAN – Wide Area Network  
WSP – Widest Shortest Path algorithm

SDN technology play a vital role in configuration, reconfiguration, resource allocation and even the pattern of inter communication in IoT ecosystem. In [1] Lin L. *et al.* discussed three properties of SDN:

*The service guarantee property* that means the system performance bounds (e.g. delay bound, backlog bound, etc.) are derived under the given traffic model and server model.

*The concatenation property* means that a series of servers can be considered as one single server and represented using the same server model.

*End-to-end property* is a parameter that describes the network performance. High end-to-end latency adversely affects the performance of time-sensitive applications, such as SDN recovering.

Talking about the algorithms applicable in SDN, they can be used mainly for the following tasks:

- SLA (Service Level Agreements) management;
- smart routing and optimal virtual machine (VM) placement;
- solving controller placement problem;
- load balancing;
- performance prediction;
- intrusion detection and prevention.

In this chapter, we briefly review the existent algorithms and approaches to utilizing SDN technology in IoT and take a look at perspectives on SDN performance prediction using data fusion technique.

### **22.1 Managing the IoT with SDN**

As can be seen from the previous chapters, SDN can cost-effectively virtualize IoT networks providing automatic device reconfiguration and bandwidth allocation to boost performance and conserve bandwidth. This technology simplifies network management for even the most complex networks by offering plug-and-play device setup and deployment, ensures security and improved access control with the benefit of greater traffic transparency at the network's edge.

According to major service and network providers, 70% of deployed networks will rely on cloud infrastructures and multi-domain

SDN controllers as far back as in 2020 [2]. This is provided by the following features:

- (1) much faster deployment (from months down to minutes);
- (2) continuous provisioning in line with up and downscaling;
- (3) end-to-end orchestration;
- (4) service assurance for fault and performance management.

### ***22.1.1 SLA management***

SLA agreement generally comprises parameters describing the service functional and non-functional properties such as the minimum acceptable QoS values (referred to as SLOs). In this context, SLO can be seen as a range of values (i.e. lower or upper thresholds) that guarantee a certain level of quality with respect to a specific service and to a specific set of variables (or aggregates, i.e. mean value or percentiles).

SLA management can be classified into three broad categories,

- (1) resource monitoring;
- (2) SLA management including SLA violation prediction;
- (3) mapping from low level monitored metrics to SLA;
- (4) mapping of SLA between SaaS, PaaS, IaaS cloud layers.

### ***22.1.2 Metrics***

As it mentioned in [3], SLA management for efficient joint use of SDN and clouds has not been developed yet and new approaches to meet these new SLA and SLO management are in demand. To do it the following metrics could be used:

- Service Availability ratio;
- Response time ratio;
- Capacity, downlink bandwidth ratio.

In [3], each SLO is defined as a multi-step function expressed as a combination of at least two metrics and thresholds, for example, response time with respect to the workload. In this connection, metrics are defined as average over a certain period of time (see Fig. 22.1).

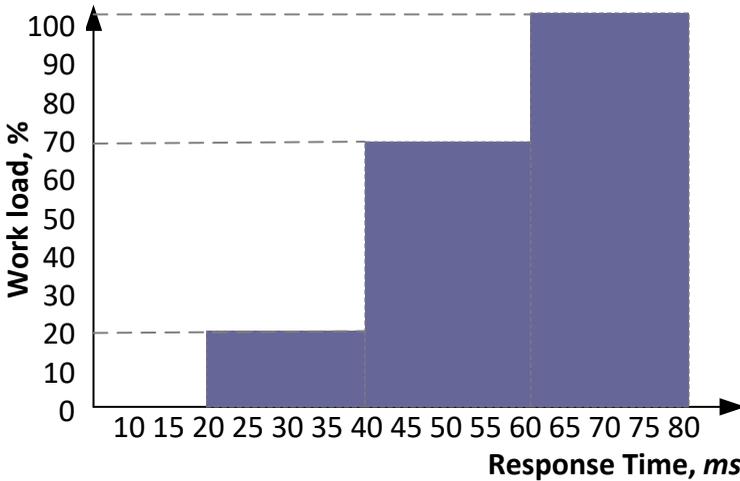


Fig. 22.1 – Example of an SLO

As shown in Fig. 22.1, for each workload interval, a specific threshold for the response time is set up. When the workload is between 0 and 20% the minimum response time is 20 *ms*; if the workload is between 20% and 70% then the threshold is 40 *ms*, etc.

## 22.2 Smart routing and scheduling

Distributed cloud systems typically consist of distributed, interconnected data centers that use virtualization technology to provide computing and storage services for each request on demand. As soon as the request arrives, several virtual machines (VMs) are created in one or more server nodes (which can be located in the same or different data centers) to satisfy the request. However, server-side crashes caused by hardware crashes, such as hard disk or memory module crashes, as well as program problems, such as program errors or configuration errors, may result in the loss of virtual machines hosted on it, and thus the entire service can not be guaranteed. An effective way to overcome this problem is to create and place more replicas of virtual machines, but this approach must also take into account the availability of nodes. For example, if all virtual machines, along with their replicas, are

hosted at sites with a high probability of failure, then proper servicing can not be guaranteed. Therefore, the availability of a virtual machine placement, a value from 0 to 1, is important and relates to the probability that at least one set of all requested virtual machine clients is in working order throughout the requested service life. Additionally, if two or more virtual machines are hosted on different hosts, we must also ensure a reliable connection between these virtual machines. In fact, one unprotected path fails if one of its related links fails. To improve the reliability of data transfer from the source to your destination, you need to protect the path (or survivability).

Figure 22.2 shows the different levels of routing in the network. IP channels are routed through the ROADM level, while Multi-Protocol Label Switching - Traffic Engineering (MPLS-TE) tunnels that carry end-to-end traffic are routed through the IP layer.

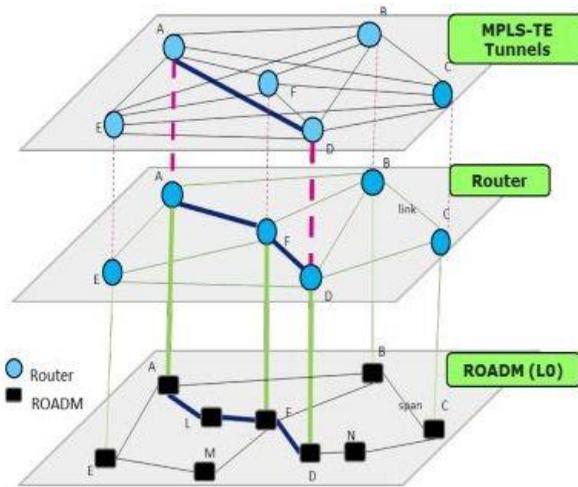


Fig. 22.2 – Different Layers of Routing (adapted from [4])

If there are  $N$  finite points of traffic and  $K$  classes of QoS, then in the traffic matrix there are  $T$  elements

$$T = K \cdot N (N-1).$$

For example, when  $K = 2$  and  $N = 100$ , then  $T = 19800$ . It is assumed also that each element of the traffic matrix is routed over the packet network as a TE tunnel. As a rule, the traffic to the TE-tunnel in a large ISP network has complex nonlinear fluctuations and the seasonal frequency at different time scales that reflect the use of the network by users. Traffic in most active tunnels contains strong daytime hesitations, less noticeable weekly fluctuations (reflecting different patterns of use on weekends), as well as sharp jumps that correspond to dynamically moving network traffic between tunnels after changing IP topology (sharp jumps can not be directly predicted by the prediction model, but the model is configurable to a new level of data immediately at a later point in time after observing the jump). As far as the changes in the long-term topology of the IP and the links routing changes are known then we can pass this information to the forecasting model. An example of the total volume of traffic and volume of traffic in a particular TE-tunnel is shown in units of the overall bandwidth in Fig. 22.3.

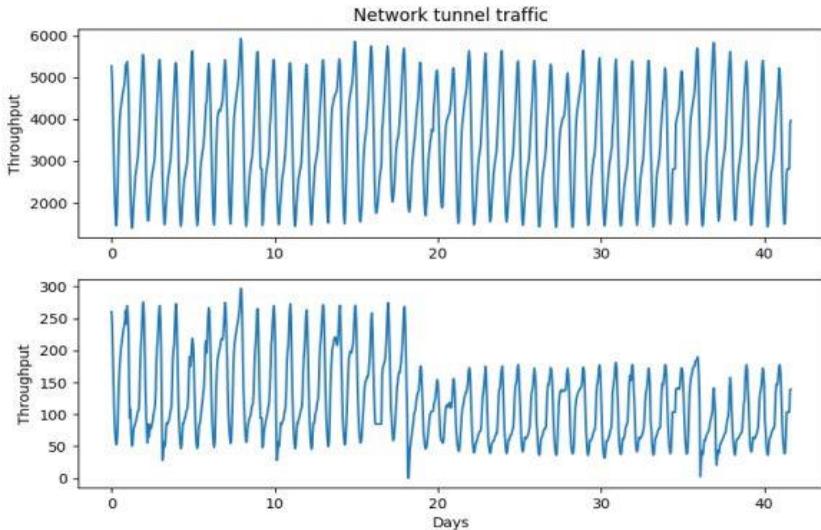


Fig. 22.3 – The tunnel traffic volume across the entire network (top) and an individual TE tunnel (bottom) using generic bandwidth units [4].

This knowledge enables to utilize different regression models to optimize many different failure scenarios and joint global optimization

of IP layers and perform capacity planning. The knowledge of near-term traffic pattern can significantly improve the feasibility and efficiency of offering SDN service.

### ***22.2.1 Data streaming over SDN***

The current trend of introducing complex software is the division of the system into several independent components or micro-services. Because components communicate through well-defined APIs, each of them can be developed separately and reused between services. Parts of the programs can be scaled separately. In IoT, software often takes the form of a circuit, where each component processes the sensor data and transfers it to processing by the next component of the circuit. Movement can be moved in both directions of the chain.

The amount of data produced by an IoT device is significantly different: some types of sensor devices wake up periodically (for example, once per hour) to report the measurement value, while other types of devices, such as video devices or complex machines, constantly generate significant amounts of data. Typically, there is a high degree of redundancy in the sensor data, for example, a multiple measurement value with low variation. Therefore, it makes sense to filter and compress data at the source before transmitting it on the uplink from the gateway to the cloud. This can be implemented as a separate data reduction component, the implementation of which is very specific to the specific use case. Moreover, many IoT applications include control loops: data produced by sensors is analyzed or supplied to a control process that runs commands about the commands sent to the device. For this type of closed loop control, low and estimated latency is crucial.

There are several proposals for implementing new routing algorithms for video streaming applications running over SDN. The purpose of developing new routing algorithms is to increase the quality of experience (QoE), a measure of client experience with a video streaming application. Commonly used QoE metrics are the received bitrate, percentage of lost packets, outage duration, number of quality changes and startup delay.

Another important component of networks is the overall performance of the connection, which is called Quality of Service

(QoS). QoS contains requirements for all major aspects of data transmission, such as response time, shudder, interrupts, etc.

### ***22.2.2 Metrics for evaluation performance of QoS routing algorithms***

To date, there are two basic QoS architectures available, they are IntServ and DiffServ [5]. However, none of them is implemented globally due to their inherent weaknesses. IntServ provides end-to-end QoS guarantees at connection level using resource backup methods. QoS requests are sent in the shortest path that is defined by traditional routing protocols. If this path is overloaded, the request will be rejected, even if some other way to the same destination has sufficient bandwidth. Architecture DiffServ uses a different approach. Instead of reserving resources for each traffic stream, DiffServ indicates the packets on the network entry and classifies them for the finite number of traffic classes. Although this solution is more scalable than IntServ, it provides only relative performance assurances. DiffServ also does not include new routing mechanisms. Multiprotocol Switching Labels (MPLS) provides a partial solution with the possibility of TE. However, TE mechanisms are not implemented in the networks of modern service providers because of the inflexibility of basic protocols that do not allow reconfiguring the network in real-time [6].

QoS routing algorithms can be analyzed in terms of their suitability for establishing traffic tunnels in large-scale backbone SDN / OpenFlow networks [7]. Besides providing the required QoS level to service providers who rent resources of the backbone network, it is desirable that algorithm maximizes utilization of the network resources, since that is the main interest of the infrastructure provider. Regarding this, there are several metrics to evaluate algorithms performance:

- (1) Bandwidth Rejection Ratio (BRR);
- (2) Average Route Length (ARL);
- (3) Delay;
- (4) Loss Ratio.

Most algorithms proposed in literature are dominantly focused on the first metrics and is a bandwidth guarantees. Second metrics is applicable because in WAN (Wide Area Networks) longer paths usually entail higher delay.

*Routing problem definition.* Let us assume that there is a network topology with  $n$  nodes and  $m$  links. Each link has its own capacity and residual bandwidth at a given time.

The routing task, which requires a path with a certain bandwidth from the input node to the output node, is processed by the routing algorithm. The algorithm consistently handles requirements with the assumption that the network conditions are available, such as topology, channel bandwidth, and input / output (I/O) pair. However, routing requirements are not known in prior.

The task of the TE routing algorithm is to send as many requests as possible, provided that each set route stores several bandwidth resources for a certain period of time (i.e. the bandwidth for each route is guaranteed). Since the I/O pair has integral commodity flows, the TE routing problem is NP-hard.

In general, there are two categories of routing algorithms, namely proactive algorithms and reactive algorithms. Most of reactive routing algorithms first calculate the weight of the channels based on the network states, and then utilize the shortest path algorithms (e.g., Dijkstra, SPF, Bellman-Ford, etc.) to select the least weighed route. As a side note, Dijkstra algorithm is used to compute a weight optimized feasible path for QoS request.

### **22.2.3 QoS routing algorithms applicable to large-scale SDN**

The most widely used bandwidth-constrained routing algorithm is *Minimum Hop Algorithm (MHA)* [8]. MHA grounded on static selection scheme that maintains information about available bandwidth on each of the links, and only those that have enough resources to satisfy user's requirement are taken into account for routing. This means that for each I/O pair, one and the same shortest path is selected, until at least one of its channels can not meet the requirements of the bandwidth. Although MHA is very simple, it could quickly create a bottleneck for future requests, leading to poor utilization of network resources.

*Widest Shortest Path (WSP) algorithm* is a modified version of MHA, as it attempts to load-balance the network traffic. In this way the algorithm tries to make tradeoff between two conflicting requirements: load balancing and resource consumption. WSP has the same

drawbacks as MHA since the path selection is performed among the shortest feasible paths which are used until saturation before switching to other feasible paths.

*Shortest Widest Path (SWP) algorithm* is a further improvement of the previous ones. In this algorithm, the first criterion is taken to be the path with the maximum residual bandwidth and if more than one path is selected then the one with the smallest number of hops is chosen.

These three algorithms use information about network topology and available bandwidth, but do not use information about source-destination (SD) node pairs to find a feasible route. To minimize "interference" on routes that may be critical to future demands in SDN, the following algorithms that do not use a priori knowledge of traffic scheme can be useful.

*Minimum Interference Routing (MIRA) algorithm* uses knowledge of the I/O label switching router, which are potential source-destination pairs of traffic. MIRA makes an on-demand routing decision based on the level of "interference" it will have upon a future request from another receiving source. This level of interference is used as the line weight to calculate the shortest path for new demand. The novelty of this algorithm leads to less selected critical references to other source-destination pairs. However, it has two major drawbacks. First, it is the difficulty to calculate the maximum flow between any source-destination pairs and the weight of all links.

*Dynamic Online Routing (DORA) algorithm* works offline and online. In an offline phase, an array of Path Potential Values (PPV) is calculated for each SD pair. The elements of the PPV array correspond to the network connection and reflect their importance for other SD pairs. First, all PPV values are set to zero. Then, for the corresponding SD pair, the set of shortest non-overlapping paths is calculated. The values of PPV links included in these paths are reduced by 1. Finally, each link is checked for the non-overlapping paths of other SD pairs. If it is found there, its PPV value is increased by 1. PPV values are determined for each SD pair individually, but these calculations are performed only when the network is initialized or with some change in the topology. At the PPV stage, the bandwidth of each channel is combined to form the weight of the channel. The effect of the residual bandwidth is controlled by the BWP parameter (Band Width Proportion):

$$weight = (1 - BWP) \cdot PPV + BWP \cdot \frac{1}{residual\_bandwidth}, 0 \leq BWP \leq 1$$

### 22.2.4 Traffic scheduling algorithms

Traffic scheduling in traditional network is generally based on IP and MPLS-TE network, using the SPF algorithm (such as OSPF, ECMP) to finish the route calculation. In physical networks routing problem can be solved by using several algorithms such as Open Shortest Path First (OSPF), or Routing Information Protocol (RIP) and nowadays, it is not an actual challenge. However, when SDN appeared, the way of understanding the network operation radically changed.

As the traditional traffic engineering can not adjust the traffic allocation dynamically, the traffic scheduling has the difficulty to maximize the network traffic while the balance over paths is achieved.

Most of the traffic engineering equilibrium models can be classified into two categories [9]:

- (1) Minimize the maximum link utilization;
- (2) Minimize the link cost.

The first category is introduced by Kennington *et al.* [10] where the following objective function is used to minimize the maximum link utilization:

$$\max_{e \in E} (BU_e), \quad (22.1)$$

where  $BU$  denotes a bandwidth utilization of link  $e$ .

The model that minimizes the link cost is described in [11] and can be denoted by the cost function  $\phi$ :

$$\min \phi = \sum_{e \in E} \varphi(BU_e), \quad (22.2)$$

where  $\phi$  is a function of the link utilization. The objective function is defined as a piecewise linear convex function  $\varphi$ .

The literature [12] indicated that traffic scheduling based on SDN has three main directions, they are the traffic scheduling of data layer, the traffic scheduling of control layer and the traffic scheduling virtualization. Currently, the issue of routing in SDN should be again considered in order to know how these networks work.

## 22.3 Optimization of SDN Traffic Flow for IoT

### 22.3.1 Algorithms for calculating the optimal position of the SDN-controller

In general case it is assumed that for a good controller placement it is necessary to minimize the latencies between nodes and controllers in the network. However, looking only at delays is not sufficient. A controller placement should also fulfill certain resilience constraints [13].

*The Controller Placement Problem formulation.* Suppose there are  $M$  SDN-compliant switches connected to form a network representing one or more logical / physical domains. For simplicity of presentation, we assume homogeneity among switches. Controllers are transmitted by switches when they receive new threads so that they can update their forwarding rules or the Forwarding Information Base (FIB). Controllers should regularly update FIB switches and provide QoS on networks.

Let  $M$  switches receive new streams, randomly generating an uneven network load scenario at any time  $T$ . Assuming that the switch  $i$  receives the  $l_i$  number of new flows and the average load that can be processed by one controller is equal to  $C$ , the minimum number of required controllers is [14]:

$$\lceil k \rceil = \frac{\sum_{i=1}^M l_i}{C}. \quad (22.3)$$

The above argument is justified if the load on the network/switches is known a priori, which in practice is not possible. In addition, with dynamic load changes, the value of  $k$  also changes

dynamically. The goal is to get the optimal value  $k$  for dynamically and optimally displaying  $k$  controllers (location) on the  $M$  switchers SDN.

To solve this problem, we assume that each controller can operate in master mode, in slave mode or in both modes (master for one set of switches and/or slave for another set of switches). In subordinate mode, the controller is able to listen to the switching of switches without any action. Both the master and subordinate controllers can communicate with each other using the communication protocol between the SDN. As the network load increases, one or more new controllers can be added to handle the load, which leads to a change in the placement of existing controllers and the change of the base state to the subordinate or vice versa. However, if the network load decreases, one active (master/subordinate) controller can be deleted, which leads to a change in the placement of the remaining controllers and the change of state from the main to the subordinate and vice versa. This process of adding/removing a controller is fixed by the following optimization problem:

$$\begin{aligned} \min f(k, c), & \quad (22.4) \\ \text{s.t.}, \Delta_i \leq \Delta_{th}, \forall_i \in I, \\ U_\alpha \leq U_i \leq U_{th}, \forall_i \in I \end{aligned}$$

where  $f$  denotes a nonlinear function of the number of controllers  $k$  and cost  $c$  associated with each controller. Note that  $k$  and  $c$  are interrelated, and  $c$  may be a  $k$  function.  $U_i$  is the usage index (CPU, memory, or stream),  $\Delta_i$  is the delay (processing combination and delay in the path) associated with the  $i$ -th controller at time  $T$ , and  $I$  is the set of all active controllers who work online. Limitations of delay and use are such that the delay associated with the controller should be less than the predefined limit value (to support QoS), and use must be within the minimum and maximum thresholds ( $U_\alpha$  and  $U_{th}$ ); the minimum threshold for cost reduction and the maximum threshold to meet the sudden increase in network traffic, respectively.

Equation (22.4) represents a global optimization problem in which the purpose and the constraints contradict each other.

The solution of equation (22.4) should be such that the number of controllers used ( $k$ ) is unique and optimal. In addition, the display of switches on controllers should provide requirements for delay and use. However, when changing the load, obtaining a unique  $k$  is impossible, which can be used for all load conditions. Moreover, a centralized solution is not recommended due to problems of scalability, controllability and fault-tolerance. Therefore, it is necessary to solve following equation using distributed individual optimization as follows

$$\begin{aligned} \min c_i, & \quad (22.5) \\ \text{s.t.}, \Delta_i & \leq \Delta_{th} \\ U_\alpha & \leq U_i \leq U_{th} \end{aligned}$$

where  $c_i$  is the value associated with the  $i$ -th controller.

### *Placement metrics*

The following metrics can be used to evaluate the position of the SDN-controller [14]:

(1) Average-case Latency.

For a network graph  $G(V, E)$  where edge weights represent propagation latencies, where  $d(v, s)$  is the shortest path from node  $v \in V$  to  $s \in V$ , and the number of nodes  $n = |V|$ , the average propagation latency for a placement of controllers  $S'$  is:

$$L_{avg}(S') = \frac{1}{n} \sum_{v \in V} \min_{(s \in S')} d(v, s) \quad (22.6)$$

In the corresponding optimization problem, the goal is to find the placement  $S'$  from the set of all possible controller placements  $S$ , such that  $|S'| = k$  and  $L_{avg}(S')$  is minimum. For an overview of the approaches to solving this problem, along with extensions, refer to [15].

(2) Worst-case latency.

An alternative metric is worst-case latency, defined as the maximum node-to-controller propagation delay:

$$L_{wc}(S') = \max_{(v \in V)} \min_{(s \in S')} d(v, s) \quad (22.7)$$

where again we seek the minimum  $S' \subseteq S$ . The related optimization problem here is finding minimum  $k$ -center [16].

(3) Latency bound.

Instead of minimizing the average or worst case, we could place controllers to maximize the number of nodes within a delay.

The general version of this problem of arbitrary overlapping sets is called maximal coverage [17]. An instance of this problem includes the number  $k$  and set of sets  $S = S_1, S_2, \dots, S_m$ , where  $S_i \subseteq v_1, v_2, \dots, v_n$ .

The objective is to find a subset  $S' \subseteq S$  of sets, such that  $\left| \bigcup_{S_i \in S'} S_i \right|$  is maximized and  $|S'| = k$ . Each set  $S_i$  comprises all nodes within a latency bound from a single node.

### *SDN-controller placement algorithms*

*K-medoids algorithm.* This is a clustering algorithm which chose the center first and take an approach of minimizing the sum of dissimilarity between the points and marked to be in a cluster and a data point chosen to be the center of that cluster.

Steps:

initial gauss for center  $C_1 \dots C_k$

Repeat:

1. Minimize over  $C$ : for each  $i=1 \dots n$  find the cluster center  $C_K$  closest to  $P_i$

2. Minimize over  $C_1 \dots C_k$  : for each  $k=1 \dots K$

3. Stop until inter-cluster variation doesn't change.

*K-center algorithm.* This is another clustering algorithm. The goal of this algorithm is to select  $K$  points from the given data points which minimizes the maximum distance from the controller to the switches.

1. Require:  $(N \times N)$  Shortest Path Matrix. and Required delay  $(r)$

2.  $k \leftarrow$  Select randomly a node
3. While there are nodes not belonging to the cluster do
4.  $Cluster_k \leftarrow$  Find the nodes  $v$  that satisfy  $d(k;v) \leq r$ , where  $v \notin$  Cluster
5. For each node  $v \in cluster_k$  do:
6. Evaluate  $\max(\min(d(v, cluster_k)))$
7. End for
8. Choose the node as controller  $s$  which minimizes the  $d(v,s)$
9. Find the furthest node  $k$  from Cluster
10. End while

*Pareto-Optimal Controller Placement (POCO) algorithm.* POCO is proposed in [13] and is a failure tolerant controller placement approach. POCO does not provide recommendations for a specific SDN controller placement, but returns a set of placements that are optimal for Pareto, which allows network operators to choose the location that best suits their needs. In particular, they can also decide how much controller failure should be covered by an elastic placement.

### **22.3.2 Balancing algorithms in IoT-based software defined networks**

As the topology SDN grows, it is necessary to manage an increasing number of switches and handle more and more threads. As stated in the OpenFlow standard, current solutions are based on the message packet header (or first packet) of each new flow of revenue to a centralized controller that reactively sets forwarding rules on switches. If there is only one controller in the control plane, it can become a bottleneck for the SDN, which will significantly degrade user interaction. To reduce the load on the controller, some researchers suggest using the default paths for all threads. When the thread enters the switch, the switch can find the appropriate rule for this thread and direct the stream directly.

However, in many practical applications, network operators must specify detailed (or for each thread) policies that determine how base switches send, reject, and measure traffic. Because substitution rules provide only rough flow management, deploying the default path for all

threads is not attractive. Therefore, in order to avoid such overload / failure of one controller, the control plane is usually implemented as a distributed system with a cluster of controllers, also called a distributed control plane.

One of the key issues in the distributed control plane is the potential load imbalance of the controller caused by the traffic dynamics. In particular, the controller may be overloaded if a large number of threads are fed to switches that are connected to this controller while another controller may not be used sufficiently. In practical networks, the traffic dynamics will occur if some applications generate streams from certain parts of the network, or some switches serve a large number of threads compared to other switches. To do this, eliminate the load imbalance of the controller is necessary.

To overcome this problem, one way is to allow each switch to dynamically change its connected controller from the source to the target, also called switching migration.

#### *Formulation of the balancing problem*

As SDNs provide the centralized control capability with the global view of network status, we address the load-balancing of control traffic to minimize the link transmission delay via an optimization approach. Specifically, the traffic assignment matrix  $\mathbf{x} = [x_{ij}]_{i \in V, j \in J}$ , where  $x_{ij}$  denotes the amount of control traffic that the  $i$ th switch contributes to the  $j$ th link, is obtained with respect to minimizing the average network delay over the network.

To achieve load balancing, multi-path routing is adopted, where given  $P_i$  as a set of available paths for the  $i$ th switch and  $i \in V$ , this switch can forward the control messages to the controller via  $|P_i|$  available paths. To characterize possible multi-path routings of control flows, for the flow from the  $i$ th switch, we define a topology matrix  $\mathbf{T}_i$  of size  $|J| \times |P_i|$  as follows:

$$\mathbf{T}_i[j, p] = \begin{cases} 1, & \text{if the } j_{\text{th}} \text{ link lies on the } p_{\text{th}} \text{ path;} \\ 0, & \text{otherwise.} \end{cases} \quad (22.8)$$

The matrix  $\mathbf{T}_i$  maps the traffic from paths to links and should always be full column-rank to avoid redundant paths. Its left-inverse matrix  $\mathbf{T}_i^{-1} = [t_{i1}, t_{i2}, \dots, t_{i|J|}]$  exists and has the size  $|P_i| \times |J|$ , where  $t_{ij}$  is the column vector that maps the  $j$ th link to all possible paths of the  $i$ th switch's flow.  $t_{ij}$  is obtained by multiplying  $\mathbf{T}_i^{-1}$  with the  $j$ th standard basis  $e_j$ , i.e.  $t_{ij} = \mathbf{T}_i^{-1} e_j$ . While each switch  $i$  brings a control flow with the mean value  $\sigma_i$ , the switch  $i^*$ , where the controller is directly attached, can send its flow to controller without going through the network (i.e.,  $x_{i^*j} = 0, \forall j \in J$ ). We set up the equalities for the control flow conservation of switches as  $\mathbf{T}_i^{-1} [x_{i1}, \dots, x_{i|J|}]_1^T = \sigma_i, \forall i \in \tilde{V} := V \setminus \{i^*\}$ , where  $\|\cdot\|^T$  and  $\|\cdot\|_1$  denote the transpose and 1-norm of vector, respectively. Let  $d_{ij} = \mathbf{T}_i^{-1} e_{j1}$ , such equalities can be further simplified as

$$\sum_{j \in J} d_{ij} x_{ij} = \sigma_i \quad \forall i \in V, \quad (22.9)$$

which is the flow conservation constraint, implying that the control flow initiated by each switch is split into multiple outgoing flows on the selected transmission links. Furthermore, with the aid of Little's law [7], the average network delay  $D$  over the network for the control messages is obtained as

$$D = \frac{1}{\sum_{i \in \tilde{V}} \sigma_i + \sum_{j \in J} \lambda_j} \sum_{j \in J} \frac{\sum_{i \in \tilde{V}} x_{ij} + \lambda_j}{\mu_j - (\sum_{i \in \tilde{V}} x_{ij} + \lambda_j)}. \quad (22.10)$$

In particular, for link  $j \in J$ , new packets arrive with rate  $\left( \sum_{i \in \tilde{V}} x_{ij} + \lambda_j \right)$  and stay an average time of  $1 / \left[ \mu_j - \left( \sum_{i \in \tilde{V}} x_{ij} + \lambda_j \right) \right]$ . Summing queue backlogs over all links, the average network delay is thus yielded, as the total external arrivals of control and data traffic into the network are  $\left( \sum_{i \in \tilde{V}} \sigma_i + \sum_{j \in J} \lambda_j \right)$ . In addition, to balance the traffic loads among all links, every link should have finite transmission delay. From the formulation in (3), such finite link delay conditions are equivalent to

$$\sum_{i \in \tilde{V}} x_{ij} < \mu_j - \lambda_j \quad \forall j \in J, \quad (22.11)$$

which ensure the incoming traffic rates are less than the link service rates and link delays remain nonnegative. Therefore, with the above accomplishments, we define the Control Traffic Load-Balancing Problem as follows.

Given a SDN modeled by  $G = (V, J)$  with the controller location  $i^* \in V$ , control traffic arrival rates  $\sigma_i$ , a set of topology matrices  $\mathbf{T}_i$ ,  $\forall i \in V$ , data traffic rates  $\lambda_j$ , and link serving rates  $\mu_j$ ,  $\forall j \in J$ , the load-balancing optimization problem to be solved by the controller is

$$D \left( \begin{array}{l} \forall i \in \tilde{V} = V \{i^*\} \\ x_{ij} \\ \forall j \in J \end{array} \right) \rightarrow \min \text{ subject to (22.9) and (22.11).}$$

### *Load balance strategies and algorithms*

Many researches have been proposed on load balance in traditional multipath network. There are two load balance strategies have been widely used in multipath network at present: (1) Equal-Cost MultiPath (ECMP) and (2) Valiant Load Balance (VLB). The core idea of ECMP is to evenly distribute data-flow to next-hop switches, and VLB

distributes traffic among all available paths and randomly picks the next-hop switch. ECMP is a simple routing scheme with load balancing. Instead of having one "better" way (measured in some metric, for example, by the number of hops) to a specific destination, ECMP enables to use several "best" paths where possible. This provides some form of load distribution, since we can, if necessary, distribute traffic in all ways.

These two strategies both use fixed methods and cannot pick transmission path adaptively to the path load condition.

*A dynamic load balance algorithm*, known as DLB, has been proposed in [18]. The DLB algorithm simply applies greedy selection strategy to pick next-hop link which transmits least data load. Although these algorithm implements load balance on multipath SDN, this routing strategy is only decided by link load of every next-hop without combining the superiority of global network view in SDN. Hence, this routing strategy may not find the best transmission path in global view so that may not achieve the best load balance effect.

*Hash-Based ECMP Flow Forwarding* [19]: a hash-based Equal-Cost Multi-Path (ECMP) [20] is a load balancing scheme for distributing flows to the available paths using stream hashing methods. The main limitation of ECMP is that two or more large long-lived streams can collide in their hash and share the same output port, thereby creating a bottleneck in the network. This static mapping of flows in the path is not associated with the current use of the network, nor with the size of the flow, which leads to collisions that can overload the switch buffers and degrade the overall use of the network. To avoid the constraints of ECMP, a significant number of large (ivory) streams can be detected on edge switches or end hosts [21] and then the central controller can calculate the appropriate paths for them, while small (mouse) streams are forwarded using ECMP routing on the switches. However, such a solution can cause high bandwidth and processing overhead on switches or hosts.

*Wildcard Rule Flow Forwarding*: OF switches use flowmatch wildcards to aggregate traffic flows [20]. OF is a great concept that simplifies network and traffic management by providing switch level control at the switch level and providing a global view of the network. However, centralized management and a global view of all flows require the controller to configure all flows for a critical path

throughout the network, which is not scalable enough and leads to both bottlenecks and delays. To reduce the number of interactions between the controller and the switches, the SDN TE approaches implement OF substitution rules on the switches, and the switches can make local routing decisions that process mouse flows to avoid controller involvement, while the controller maintains control over only elephant target streams, especially for flows important to quality of service (QoS). In another approach, authoritative switches are used to process all data packets without involving a controller in order to reduce the control costs on the control plane.

*VLB algorithm* is a randomized load distribution or two-phase routing algorithm. This is decentralized, so each node makes local decisions. It also makes the scheme scalable. VLB does not depend on the traffic matrix, because randomness erases the traffic pattern, and different traffic matrices can lead to the same load on the channels.

Consider a network of  $N$  nodes, each with capacity  $r$ , i.e., a node can initiate traffic at the maximum rate of  $r$ , and can receive traffic at the same maximum rate. We assume that the network traffic satisfies such node aggregate constraint, because otherwise there is no way to avoid congestion. A logical link of capacity  $2r/N$  is established between every pair of nodes over the physical links, as shown in Figure 22.4.

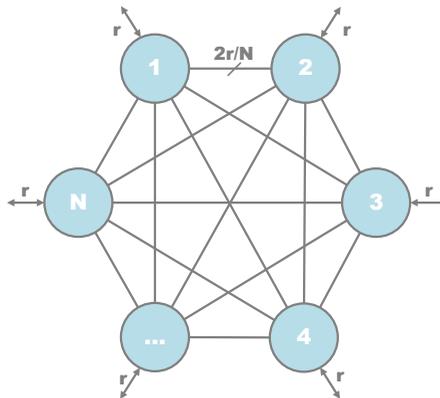


Fig.22.4 – VLB in a network of  $N$  identical nodes each having capacity  $r$ .

We use the convention that a *flow* in the network is defined by the source node and the destination node, unless further specified.

Every flow entering the network is equally split across  $N$  two-hop paths between input and output nodes, i.e., a packet is forwarded twice in the network: In the first hop, an input node uniformly distributes each of its incoming flows to all the  $N$  nodes, regardless of a full mesh of logical links of capacity  $2r/N$  connect the nodes the destinations. In the second hop, all packets are sent to the final destinations by the intermediate nodes. Load-balancing can be done packet-by-packet, or flow-byflow at the application flow level. The splitting of traffic can be random (e.g., to a randomly picked intermediate node) or deterministic (e.g., round-robin).

Assume we can achieve perfect load-balancing, i.e., can split traffic at the exact proportions we desire, then each node receives exactly  $1/N$  of every flow after first-hop routing. This means, all the  $N$  nodes equally share the burden of forwarding traffic as the intermediate node. When the intermediate node happens to be the input or output node, the flow actually traverses one hop (the direct link between input and output) in the network. Hence,  $2/N$  of every flow traverses the corresponding one-hop path.

Such uniform load-balancing can guarantee to support all traffic matrices in this network. Since the incoming traffic rate to each node is at most  $r$ , and the traffic is evenly load-balanced to  $N$  nodes, the actual traffic on each link due to the first-hop routing is at most  $r/N$ . The second-hop routing is the dual of the first-hop routing. Since each node can receive traffic at a maximum rate of  $r$  and receives  $1/N$  of the traffic from every node, the actual traffic on each link due to the second-hop routing is also at most  $r/N$ . Therefore, a full-mesh network where each link has capacity  $2r/N$  is sufficient to support all traffic matrices in a network of  $N$  nodes of capacity  $r$ .

This is perhaps a surprising result – a network where any two nodes are connected with a link of capacity  $2r/N$  can support traffic matrices where a node can send traffic to another node at rate  $r$ . It shows the power of load-balancing. In VLB, each flow is carried by  $N$  paths, and each link carries a fraction of many flows; therefore any large flow is averaged out by other small flows. In a static full-mesh network, if all the traffic were to be sent through direct paths, we would

need a full-mesh network of link capacity  $r$  to support all possible traffic matrices; therefore, load-balancing is  $N/2$  times more efficient than direct routing.

*Rounding-Based Multi-Area Routing (RDMAR)* algorithm is used for link/controller load balancing in an SDN. To solve LBR-C problem, the algorithm constructs a linear program as its relaxation. More specifically, LBR-LC assumes that the traffic of each flow should be forwarded through only one path. By relaxing this assumption, the traffic of each flow  $f$  is permitted to be forwarded through a set of feasible paths  $P_f$ . We formulate the following program  $LP_1$ :

$$\left\{ \begin{array}{ll} \sum_{f \in \Gamma_1} \sum_{e \in p, p \in P_f} y_f^p s(f) \leq \lambda_j & \forall e \in E_j \\ \sum_{f \in \Gamma_1^2} \sum_{p \in P_f, p \cap E_l \neq \emptyset} y_f^p s(f) \leq \varphi_{jl}(t) & l \in A_j \\ \sum_{f \in \Gamma_2} \sum_{p \in P_f, p \cap E_l \neq \emptyset} y_f^p \leq \phi_{kl}(t) & l \in A_j \\ \sum_{p \in P_f} y_f^p = 1 & \forall f \in \Gamma \\ y_f^p \in [0,1] & \forall f \in \Gamma, p \in P_f \end{array} \right. \quad (22.12)$$

The main difference of variable  $y_f^p$  with Eq. (22.12) is fractional instead of integral. Since  $LP_1$  is a linear program, it can be solved in polynomial time with a linear program solver. We assume that the optimal solution is denoted by  $\tilde{\lambda}_j$ . Using randomized rounding method, we obtain an integral solution  $\hat{y}$ . More specifically,  $\hat{y}_f^p$  is set as 1 with the probability  $\tilde{y}_f^p$  while satisfying  $\sum_{p \in P_f} \tilde{y}_f^p = 1, \forall f \in \Gamma$ . By

this way, we have determined the path for flow  $f$ .

The RDMAR algorithm is formally described as follows.

Algorithm RDMAR on Controller  $u_j$

- 1: Step 1: Solving the relaxed LBR-C Problem
- 2: Construct a linear program in Eq. (22.12) as relaxed LBR-LC

- 3: Obtain the optimal solution  $\tilde{y}_f^p$
- 4: Step 2: Flow route Selection for Load Balancing
- 5: Drive an integer solution  $\hat{y}_f^p$  for each flow by randomized rounding
- 6: for each flow  $f \in \Gamma$  do
- 7:   for each feasible path  $p \in P_f$  do
- 8:     if  $\hat{y}_f^p = 1$  then
- 9:       Appoint a feasible path  $p$  for flow  $f$

### ***22.3.3 Algorithms for finding the optimal path in SDN networks***

In networking systems, data can be disseminated (from source to destination) either through a single or multi path(s). In single path, data should be routed from origin to destination through a unique path, which has to meet some of predefined constraints. While the other solution for traffic distribution is by routing the traffic through a number of existing paths between the origin and destination. According to the literature, for each of these routing strategies there are some different methods with common principles such as using the well-known shortest path algorithms like Dijkstra, Bellman-Ford, etc. The major routing schemes are usually focusing on how to select the most optimal path (single path) in order to disseminate the data packets, while various of single-path algorithms have been reported and classified based on their QoS metrics. For all of the proposed algorithms there is a set of common metrics such as bandwidth, delay, jitter, packet loss and hop count. The concept of disjoint path, when

$$Path1 \cap Path2 = \emptyset ,$$

can be utilized in both data transmission scenarios, for instance it can be employed as a backup for the single path methods or a primary/backup for the multi ones. Disjoint paths have a significant benefit over many aspects and it always more preferable to be used in the context of enhancing the network performance like link/node failure, load balance improvement and for better network resource utilization.

The path computation is the core work for the controller after it learns the network topology. Dijkstra's and Floyd-Warshall algorithms are commonly used for the shortest path computation. Dijkstra's algorithm computes all the shortest paths between a single-source to all possible destinations; while Floyd-Warshall algorithm computes the shortest paths for all possible source/destination pairs. In a legacy network, each network node computes the shortest path from the node to all destinations, so the Dijkstra's algorithm is the best option. In SDN networks, the controller knows the complete network topology and is in charge of setting up paths for all possible source/destination pairs, thus Floyd-Warshall algorithm is the good choice.

## **22.4 SDN Performance prediction**

Regarding to the performance of the SDN and OpenFlow, there are not much research focused on this topic, yet. The most common approach to evaluate the performance of SDN is benchmark tools, namely OFLOPS [22], OFCBenchmark [23], etc. OFLOPS is used to measure the performance of OpenFlow-enabled hardware and software switches on the controller side. OFCBenchmark is a benchmark tool designed to create a set of message-generating virtual switches. Each switch can be configured independently from each other to simulate a specific scenario, at the same time keeping its own statistics. From other hand, there are a lot of papers devoted do machine learning (ML) algorithms that have been successfully applied to a wide variety of problem. The application of ML to SDN communication and networking is still in its infancy

### ***22.4.1 Algorithms performance metrics***

The comprehensive overview of Metrics for Analyzing, Developing and Managing Telecommunication Networks is given in [24]. In this paragraph, we will address to the metrics applicable to ML. When applying ML to a classification problem, a common approach to evaluate the ML-algorithm performance is to show its classification accuracy and performance to overcome complexity.

Comparison of ML algorithms and some performance metrics is shown in Table 22.1.

Table 22.1 – Different use cases at network layers, metrics and algorithms (adopted from [25])

| Use Case                            | Metrics                                                                       | Adopted algorithms                                                   | Ref. |
|-------------------------------------|-------------------------------------------------------------------------------|----------------------------------------------------------------------|------|
| QoT estimation (BER classification) | Accuracy, false positives                                                     | Naive Bayes, Decision tree, RF, J4.8 tree, CBR                       | [26] |
|                                     | Accuracy, AUC, running time                                                   | KNN, RF                                                              | [27] |
|                                     | Accuracy, Confusion Matrix, ROC curves                                        | KNN, RF, SVM                                                         | [28] |
| MF recognition in Stokes space      | Running time, minimum OSNR to achieve 95% accuracy                            | K-means, EM, DBSCAN, OPTICS, spectral clustering, Maximum-likelihood | [29] |
| Failure Management                  | Confusion Matrix                                                              | Bayesian Inference, EM                                               | [30] |
|                                     | Accuracy versus model parameters (BER sampling time, amount of BER data etc.) | NN, RF, SVM                                                          | [31] |
| Flow / Loss Classification          | Misclassification probability (similar to FPR)                                | HMM, EM                                                              | [32] |

#### 22.4.2 An overall approach to detect and diagnose failures in SDN

A framework to identification performance problems and failure detector based on troubleshooting and performance tuning methodology for multi-database was proposed in [33], [34]. In this paper, we will show the capability of our data fusion technique to ensure early detection of SDN performance issues that arise as a result of cumulative

effects (overloading requests, exceeding execution time, etc.) under competing hypotheses. It is suggested that the SDN is in a critical state when there is slow processing or some resource is heavily loaded to respond in the normal state.

The methodology involves the use of data monitoring parameters and metrics, obtained in real time and includes the following six stages, (1) data preprocessing and normalization, (2) time-series forecasting, (3) computing residuals for every node, (4) computing BPA for every node, (5) fusion BPA, (6) decision making regarding future performance issues.

At the first stage, the preliminary processing of the received data and their normalization is carried out. This stage is necessary for correct fusion since the parameters that are used to assess the state of the database are measured in unequal units. At the next stage, data is predicted using the ARIMA (Autoregressive Integrated Moving Average) model. At the third stage, the obtained predicted values of each parameter are used to calculate the prediction error residues, as  $\eta_t$ , the forecast errors. These values will be used for fusion and assess deviations.

$$\eta_t = y_t - \hat{y}_t, \quad (22.13)$$

where  $y_t$  - real value,  $\hat{y}_t$  - predicted value.

At the fourth stage, the basic probability assessments (BPA)  $m(X)$  of the normal  $m_i(\{N\})$  and critical state  $m_i(\{C\})$  of the database system are performed. For computing the BPA residuals obtained on the previous stage are used. The main probability distribution function of the normal state of the database system can be defined as follows.

$$m(\{N\}) = P(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}, \quad (22.14)$$

where  $x$  denotes the remainder of the parameter at a given time stamp;  $\mu$  is the average value of the balance of the state of the database system;  $\sigma$  is the standard deviation of the residual of the state parameter of the database system.

In compliance with DS theory of evidence for full SDN set states  $\Omega = \{N = \text{"normal"}, C = \text{"critical"}\}$ ,  $N \cap C = \Omega$ , the probability of the critical state  $m(\{C\})$  can be determined using (22.14):

$$m(\{C\}) = 1 - m(\{N\}). \quad (22.15)$$

At the fifth stage, the fusion of the BPA for normal  $m_i(\{N\})$  and critical state  $m_i(\{C\})$  is performed.

Since the number of evaluated parameters is more than two, the hybrid model is used as the base model within the concept of combining Dezert-Smarandache (DSmT) [35], which is an extension of the Dempster-Shafer theory [36] with the following combination rule  $m^{PCR}(X)$ , proposed by Martin and Osswald in [37] as a PCR6:

$$m^{PCR}(X) = m_{123\dots s}(X) + \sum_{\substack{X_1, X_2, \dots, X_s \in \Theta \setminus \{\emptyset\} \\ X_1 \cap X_2 \cap \dots \cap X_s = \emptyset}} \frac{m_1(X_1) \cdot m_2(X_2) \cdot \dots \cdot m_s(X_s)}{m_1(X_1) + m_2(X_2) + \dots + m_s(X_s)}, \quad (22.16)$$

where  $m_{123\dots s}(X) \cap m(X)$  corresponds to the conjunction of consensus on  $X$  between  $s > 2$  parameters,  $m_{123\dots s}(X)$  is the conjunctive rule given by the equation

$$m_c(X) = \sum_{Y_1 \cap \dots \cap Y_M = X} \prod_{j=1}^M m_j(Y_j). \quad (22.17)$$

At the last stage, a model correction is carried out, a report is generated for the ISP, and a decision is made to create an additional report on the critical state of the database.

For two consecutive-time combined estimates of  $m^{PCR1}$  and  $m^{PCR2}$  containing  $n$  mutually exclusive and exhaustive hypotheses, the distance  $d$  between  $m^{PCR1}$  and  $m^{PCR2}$  is calculated as (22.18):

$$d(m^{PCR1}, m^{PCR2}) = \sqrt{\frac{1}{2} \left( \|m^{PCR1}\|^2 + \|m^{PCR2}\|^2 - 2 \langle m^{PCR1}, m^{PCR2} \rangle \right)}. \quad (22.18)$$

The resulting value is used as the source of the decision about the similarity of the indicators and, as the degree of belief for the predicted state. Thus, a conflict of hypotheses regarding the existing problems associated with the performance of the SDN can be determined the probability of a critical overload of the system. The next section contains an example of the implementation of technology for obtaining predictions about the performance of the SDN.

### 22.4.3 Case study

The study uses the time series of 2006 observations of the following parameters CPUW, ASES, and IOPS. Table 22.2 provides quantitative metrics for different pieces of data.

Table 22.2 – A Fragment of the Initial Data, Sample #1

| ID  | SDN metrics |          |          |
|-----|-------------|----------|----------|
|     | CPUW        | IOPS     | LoadProc |
| 403 | 12,636      | 57,6812  | 1583     |
| 404 | 14,569      | 57,6812  | 1559     |
| 405 | 18,345      | 57,6812  | 1585     |
| 406 | 45,604      | 44,87961 | 1604     |
| 407 | 34,266      | 44,87961 | 1591     |
| 408 | 25,791      | 44,87961 | 1579     |
| 409 | 15,362      | 47,92667 | 1481     |

Figures 22.5, 22.6, and 22.7 show the different time series of observations of CPUW, ASES, and IOPS with a time step of observations equal to 5 minutes.

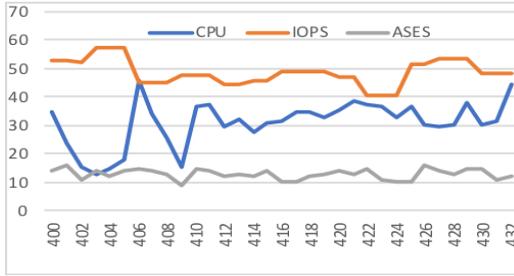


Fig. 22.5 – Sample #1

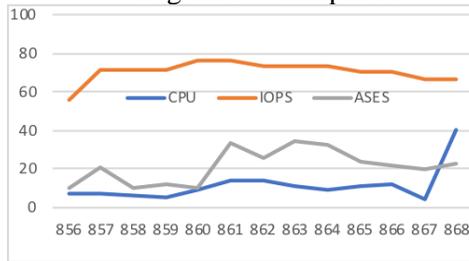


Fig. 22.6 – Sample #2

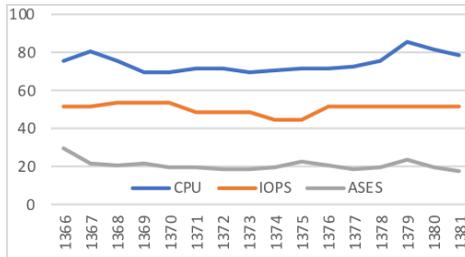


Fig. 22.7 – Sample #3

*Data normalization.* Data normalization was carried out using the calculation of the percentage of approximation of the current values of monitored parameters to their preset limits.

The maximum values of selected metrics are taken as 100%, and the values of real data are estimated, relative to their approximation to the limit values.

The examples of normalized data are presented in Tables 22.3-22.5.

Table 22.3 – A fragment of the normalized data, sample #1

| ID  | Metrics |          |       |
|-----|---------|----------|-------|
|     | CPUW'   | IOPS'    | ASES' |
| 403 | 12,636  | 57,6812  | 14    |
| 404 | 14,569  | 57,6812  | 12    |
| 405 | 18,345  | 57,6812  | 14    |
| 406 | 45,604  | 44,87961 | 15    |
| 407 | 34,266  | 44,87961 | 14    |
| 408 | 25,791  | 44,87961 | 13    |
| 409 | 15,362  | 47,92667 | 9     |

Let us assume that CPUW, IOPS parameters are obtained as a percentage of the available system resources and do not require additional normalization (100% CPUW or IOPS corresponds the worst state of the SDN). LoadProc is taken from the host operation system (OS) and requires normalization since the maximum number of processes that can be started may vary depending on the system.

Table 22.4 – A fragment of the normalized data, sample #2

| ID  | SDN metrics |          |       |
|-----|-------------|----------|-------|
|     | CPUW'       | IOPS'    | ASES' |
| 857 | 6,796       | 71,59825 | 21    |
| 858 | 5,966       | 71,59825 | 10    |
| 859 | 4,940       | 71,59825 | 12    |
| 860 | 8,807       | 76,54575 | 10    |
| 861 | 13,563      | 76,54575 | 33    |
| 862 | 13,725      | 73,60774 | 26    |
| 863 | 11,178      | 73,60774 | 34    |

Table 22.5 – A fragment of the normalized data, sample #3

| ID   | SDN metrics |          |       |
|------|-------------|----------|-------|
|      | CPUW'       | IOPS'    | ASES' |
| 1366 | 75,083      | 51,83118 | 30    |
| 1367 | 80,318      | 51,83118 | 22    |

|      |        |          |    |
|------|--------|----------|----|
| 1368 | 75,815 | 53,57843 | 21 |
| 1369 | 70,036 | 53,57843 | 22 |
| 1370 | 69,131 | 53,57843 | 20 |
| 1371 | 71,127 | 48,31402 | 20 |
| 1372 | 71,340 | 48,31402 | 19 |

Next, we determine the minimum indicator of the system operating in which there are no external connections to the database, only the processes of the OS system and the database. In our case, it corresponds to 66%. This number is taken as the initial or zero state. So, for a system with an established maximum of 2,000 processes, 1,583 processes account for 80%.

### ***Performance prediction***

Prediction of the parameter values was performed using the Autoregressive Integrated Moving Average (ARIMA) model.

$$\hat{y}_t = \mu + \phi_1 y_{t-1} + \dots + \phi_p y_{t-p} - \theta_1 e_{t-1} - \dots - \theta_q e_{t-q}, \quad (22.19)$$

where  $\hat{y}_t$  denotes the predicted value,  $\mu$ ,  $\phi_k$ ,  $\theta_l$  are the parameters of the model,  $p$  is the order of autoregression,  $q$  is the order of the moving average,  $e_t$  denotes the random noise at the time.

The prediction of state of the database was performed for three metrics (CPU, ASES, IOPS) for 1373 time steps.

To assess the quality of the forecasting model, the Bayes Information Criterion (*BIC*) was used.

$$BIC = -2 \ln(L) + k \ln(n). \quad (22.20)$$

As a result, the ARIMA (1,1,1) model with the minimum value of BIC was chosen.

The predicted and actual values for CPU, ASES and IOPS are plotted in Fig. 22.8, 22.9 and 22.10 respectively.

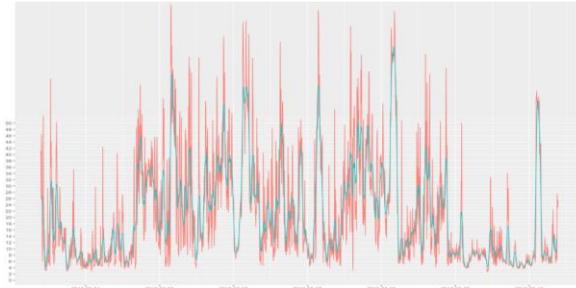


Fig. 22.8 – Predicted and actual CPU values



Fig. 22.9 – Predicted and actual ASES values

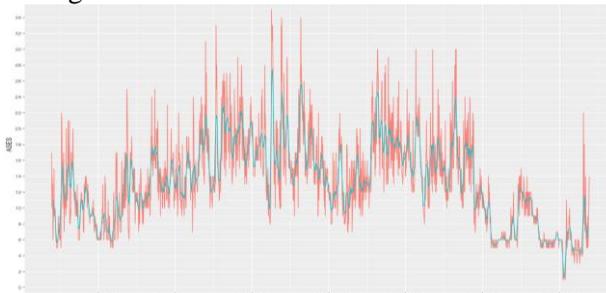


Fig. 22.10 – Predicted and actual IOPS values

As can be seen from the figures, the predicted values closely correlate with the actual data values. However, as mentioned above, there is a set of competing hypotheses argued for CPU, IOPS and ASES parameters in different time steps that complicate the detection of database performance issues.

**Residuals of the ARIMA model**

The residuals for three fragments of normalized data metrics of the database are presented in Tables 22.6-22.8 respectively.

Table 22.6 – Residuals fragment of the sample #1

| ID  | Residuals         |                   |                   |
|-----|-------------------|-------------------|-------------------|
|     | R <sub>CPUW</sub> | R <sub>IOPS</sub> | R <sub>ASES</sub> |
| 403 | 0,51302           | -0,29273          | 0,12161           |
| 404 | -0,46444          | 0,10807           | 0,305274          |
| 405 | -0,10301          | -0,09519          | 0,00709           |

Table 22.7 – Residuals fragment of the sample #2

| ID  | Residuals         |                   |                   |
|-----|-------------------|-------------------|-------------------|
|     | R <sub>CPUW</sub> | R <sub>IOPS</sub> | R <sub>ASES</sub> |
| 860 | -0,43145          | -0,10342          | 1,29507           |
| 861 | -0,10942          | 0,03312           | 3,06829           |
| 862 | -0,21710          | 0,20911           | 1,20827           |

Table 22.8 – Residuals fragment of the sample #3

| ID   | Residuals         |                   |                   |
|------|-------------------|-------------------|-------------------|
|      | R <sub>CPUW</sub> | R <sub>IOPS</sub> | R <sub>ASES</sub> |
| 1366 | 1,36751           | 0,39218           | -0,85509          |
| 1367 | 0,05604           | -0,19447          | 0,30656           |
| 1368 | -1,4567           | -0,20819          | -0,57332          |

In the absence of noise, these residues usually show smooth fluctuations, which can be observed in the residuals plots. Then the residuals of the three sensitivity parameters are used as sources of evidence for using the DS<sub>m</sub>T fusion method and the results of fusion is the probability of a critical state of the database system.

**Basic probability assignment**

BPA is calculated by the equation (22.14) for each parameter of the database state for each epoch. The fragments of the results of the calculation of BPA for the normal (*N*) and critical (*C*) state of the

database is presented in Tables 22.9-22.11. As it can be seen from the tables, there are six partial conflicts between data.

Table 22.9 –A Fragment of the Calculation of BPA for the Sample #1

| ID  | Basic probability assessment |                         |                          |                          |                          |                          |
|-----|------------------------------|-------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
|     | m <sub>CPU</sub><br>(N)      | m <sub>CPU</sub><br>(C) | m <sub>IOPS</sub><br>(N) | m <sub>IOPS</sub><br>(C) | m <sub>ASES</sub><br>(N) | m <sub>ASES</sub><br>(C) |
| 403 | 0,40605                      | 0,59395                 | 0,66328                  | 0,33671                  | 0,39207                  | 0,60793                  |
| 404 | 0,45162                      | 0,54837                 | 0,77489                  | 0,22511                  | 0,36874                  | 0,63126                  |
| 405 | 0,52881                      | 0,47119                 | 0,77490                  | 0,22510                  | 0,40030                  | 0,59970                  |

Table 22.10 – A Fragment of the Calculation of BPA for the Sample #2

| ID  | Basic probability assessment |                         |                          |                          |                          |                          |
|-----|------------------------------|-------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
|     | m <sub>CPU</sub><br>(N)      | m <sub>CPU</sub><br>(C) | m <sub>IOPS</sub><br>(N) | m <sub>IOPS</sub><br>(C) | m <sub>ASES</sub><br>(N) | m <sub>ASES</sub><br>(C) |
| 860 | 0,46259                      | 0,53741                 | 0,77225                  | 0,22775                  | 0,14637                  | 0,85363                  |
| 861 | 0,52839                      | 0,47161                 | 0,78967                  | 0,21033                  | 0,00229                  | 0,99771                  |
| 862 | 0,51584                      | 0,48416                 | 0,72947                  | 0,27053                  | 0,16521                  | 0,83479                  |

Table 22.11 – A Fragment of the Calculation of BPA for the Sample #3

| ID   | Basic probability assessment |                         |                          |                          |                          |                          |
|------|------------------------------|-------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
|      | m <sub>CPU</sub><br>(N)      | m <sub>CPU</sub><br>(C) | m <sub>IOPS</sub><br>(N) | m <sub>IOPS</sub><br>(C) | m <sub>ASES</sub><br>(N) | m <sub>ASES</sub><br>(C) |
| 1366 | 0,09254                      | <b>0,90746</b>          | 0,59034                  | <b>0,40966</b>           | 0,30445                  | <b>0,69555</b>           |
| 1367 | <b>0,52677</b>               | 0,47323                 | <b>0,73050</b>           | 0,26950                  | <b>0,36854</b>           | 0,63146                  |
| 1368 | <b>0,08917</b>               | 0,91083                 | <b>0,72236</b>           | 0,27764                  | <b>0,36195</b>           | 0,63805                  |

### *Data fusion*

In such manner we get all BPA and apply DS<sub>m</sub>T combination rule for each partial conflict using formula (22.16), as demonstrated below:

$$m_{CPU}^{1366}(N)m_{ASES}^{1366}(C)m_{IOPS}^{1366}(C) = 0,09254 \cdot 0,69555 \cdot 0,40966 \approx 0,0264. \quad \square$$

$$\frac{x_N^{PCR}}{0,09254} = \frac{x_{C,2}^{PCR}}{0,69555} = \frac{x_{C,3}^{PCR}}{0,40966} = \frac{0,0264}{1,1977} \approx 0,022.$$

$$\begin{cases} x_N^{PCR} = 0,09254 \cdot 0,0220 \approx 0,002, \\ x_{C,2}^{PCR} = 0,69555 \cdot 0,0220 \approx 0,0153, \\ x_{C,3}^{PCR} = 0,40966 \cdot 0,0220 \approx 0,009, \end{cases}$$

and therefore with  $m^{PCR}(X)$ , the following redistributions to  $N$  and  $C$  states are obtained:

$$\begin{cases} x_{1N}^{PCR} \approx 0,002, \\ x_{1C}^{PCR} = x_{C,2}^{PCR} + x_{C,3}^{PCR} = 0,0153 + 0,009 = 0,0243. \end{cases}$$

A fragment of the results of the calculation partial conflicts for the normal and critical state of the database is presented in Table 22.12 and Table 22.13.

Table 22.12 – The partial conflicts  $X_j(N)$  for the sample #3

| ID   | Partial conflicts $X_i(N)$ |          |          |          |          |          |
|------|----------------------------|----------|----------|----------|----------|----------|
|      | $X_1(N)$                   | $X_2(N)$ | $X_3(N)$ | $X_4(N)$ | $X_5(N)$ | $X_6(N)$ |
| 1366 | 0,0020                     | 0,0212   | 0,1003   | 0,0810   | 0,0188   | 0,0057   |
| 1367 | 0,0331                     | 0,0156   | 0,0869   | 0,0891   | 0,1618   | 0,0402   |
| 1368 | 0,0014                     | 0,0214   | 0,1335   | 0,1294   | 0,0230   | 0,0055   |

Table 22.13 – The partial conflicts  $X_j(C)$  for the sample #3

| ID   | Partial conflicts $X_i(C)$ |          |          |          |          |          |
|------|----------------------------|----------|----------|----------|----------|----------|
|      | $X_1(C)$                   | $X_2(C)$ | $X_3(C)$ | $X_4(C)$ | $X_5(C)$ | $X_6(C)$ |
| 1366 | 0,0243                     | 0,0919   | 0,2723   | 0,0821   | 0,0192   | 0,0059   |
| 1367 | 0,0566                     | 0,0314   | 0,1314   | 0,0383   | 0,0812   | 0,0121   |
| 1368 | 0,0144                     | 0,0702   | 0,2863   | 0,1087   | 0,0181   | 0,0034   |

Therefore, with PCR one finally gets

$$m_{CPU,ASES,IOPS}^{PCR}(N) = 0,016632 + \sum_{i=1}^6 x_i(N) \approx 0,2456.$$

$$m_{CPU,ASES,IOPS}^{PCR}(C) = 0,258571 + \sum_{i=1}^6 x_i(C) \approx 0,7544.$$

A fragment of the results of the calculation  $m_{CPU,ASES,IOPS}^{PCR}$  for normal and critical state of the database is presented in Table 22.14.

Table 22.14 – The Results of Data Fusion for sample #3

| ID   | $m_{CPU,ASES,IOPS}^{PCR}(N)$ | $m_{CPU,ASES,IOPS}^{PCR}(C)$ |
|------|------------------------------|------------------------------|
| 1366 | 0,0243                       | 0,0919                       |
| 1367 | 0,0566                       | 0,0314                       |
| 1368 | 0,0144                       | 0,0702                       |

Figure 22.11 summarizes the experimental results for fusion CPU, IOPS and ASES on time steps 1366-1381 as a two lines for normal (green dash line) and critical states (red dash line).

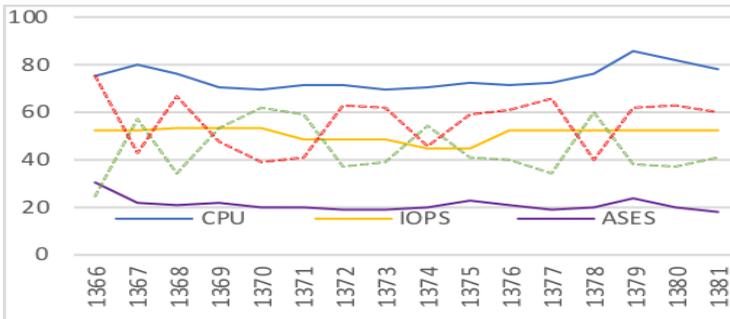


Fig. 22.11 – Data fusion results for Sample#3

As can be seen from Fig. 22.8, the calculated values at the time-steps #1367, 1369, 1370, 1371, 1378 show that SDN state is normal, but

the real values of CPUW and ASES are not normal, that could mean failure propagation. In this situation, the correction of the model is needed.

Applying equation (22.18) and calculating the distance between two BPA enables to correct values between two time periods and allows making a decision concerning the state of SDN.

The results of the distance calculation for Saple#3 are presented in Table 22.15.

As is seen from Fig. 22.11, on time-step #1366 the values point on the critical state, at #1367 they are normal, so we calculate the distance, at #1368 a second assumption about critical state appears.

Table 22.15 – A fragment of the calculation of BPA for the sample #3

| ID   | $m_{CPU}$<br>(N) | $m_{CPU}$<br>(C) | $m_{ASES}$<br>(N) | $m_{ASES}$<br>(C) | $d_{bpa}$<br>( $m_1, m_2$ ) | $d_{bpa}$<br>( $m_1, m_2$ ) |
|------|------------------|------------------|-------------------|-------------------|-----------------------------|-----------------------------|
| 1366 | 0,093            | 0,907            | 0,304             | 0,696             |                             |                             |
| 1367 | 0,091            | 0,473            | 0,369             | 0,631             | 0,307                       | -                           |
| 1368 | 0,089            | 0,911            | 0,362             | 0,638             | 0,722                       | 0,002                       |
| 1369 | 0,092            | 0,908            | 0,380             | 0,620             | 0,207                       | 0,205                       |
| 1370 | 0,178            | 0,822            | 0,401             | 0,599             | 0,306                       | 0,308                       |
| 1371 | 0,0923           | 0,908            | 0,367             | 0,633             | 0,002                       | 0,311                       |
| 1372 | 0,134            | 0,866            | 0,370             | 0,630             |                             |                             |
| 1373 | 0,092            | 0,908            | 0,114             | 0,886             |                             |                             |

Distance between previous critical state is equal to 0,024 so critical state are similar that correcting the value #1367 to 0,091. The system alert from 3 critical state has passed, at #1369 there are two critical signals from CPU and ASES that are above 0,5. After this step we correct value to 0,092. Steps #1370, 1371 are passed as normal. At step #1372 two BPA(C) of CPU and ASES are above 0,5 than correcting the values #1370, 1371, 1372 to 0,178; 0,0923; 0,134. Final step #1373 has two BPA(C) of CPU and ASES are above 0,5 than correcting the value to 0,0923.

The corrected results are shown in Fig. 22.12 by two additional lines for probabilities of normal (green bold line) and critical states (red bold line) that most closely approximate the real state of the system.

The prediction models can be used to increase the ability of the ISP to respond on different performance issues. Grounded on history monitoring data, Network I/O, query complexity etc. for various data sizes the performance prediction models evaluate the query execution time and inform ISP about upcoming trouble event. This makes it possible to identify risks and prepare mitigating plan.

Thus, using the proposed methodology, in conditions of conflict of probabilities of values and analysis of time series, system is able to compute the probability of occurrence of performance issues and critical state.

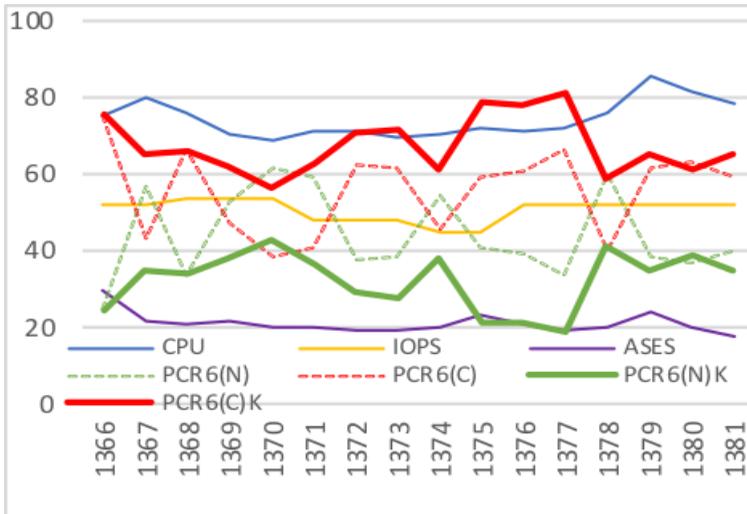


Fig. 22.12 – Correcting the model values for Sample#3

When troubleshooting forecast or alarm is triggered, the ISP will obtain automatic notify (via e-mail, web-page, etc.) about the recent developments to take appropriate action. This approach allows users to maintain a stable yet optimal performance of their business-critical systems.

## 22.5 Work related analysis

There are different approaches to enforce service level agreements in SDN and virtualized network functions. Bendriss *et al.* [3] proposed a technique to prediction of service level objectives breaches for streaming services deploying on NFV and SDN. An analytical model to measure performance of SDN deployment based on stochastic network calculus is presented in [1]. Research on Load Balance Method in SDN is presented in [38]. Hani et al. [39] raise the issue of predicting violations of SLA. They define SLA violation as deviations from the conditions agreed on in the SLA. They use a Support Vector Machine (SVM) adapted for regression, termed SVR, for time series forecasting. They identify two SLOs, namely, bandwidth and response time in cloud database. The final evaluation shows a minimum accuracy of more than 80% for 10 days look ahead.

Another interesting issue not discussed in this chapter is Intrusion detection in SDN. It has a big potential to further research and deployment in SDN. Thus, Kokila et al. [40] proposed a method for detection of DDoS attacks on the SDN controller.

Our colleagues from University of Leeds [41] developed a Deep Neural Network (DNN) model for an intrusion detection system and trained it with the NSL-KDD Dataset. In this work, they used six basic features (that can be easily obtained in an SDN environment) taken from the fortyone features of NSL-KDD Dataset. Through experiments, they confirmed that the deep learning approach shows strong potential to be used for flow-based anomaly detection in SDN environments. The key difference between their work and other papers is that they uses simplex preprocessing and features extraction in the SDN context. A comprehensive survey of recent works that apply SDN to security, and identify promising future directions that can be addressed by such research is prepared by multinational research team from Newcastle University (UK), university of New South Wales (Australia) and Cisco Systems (Australia) [42].

The course IK3619 Software Defined Networking (SDN) and Network Functions Virtualization from KTH Royal Institute of Technology [43] is targeted on a deep understanding of two important, emerging network technologies: Software Defined Networking (SDN)

and Network Functions Virtualization (NFV). This course consists of 4 hours of lectures and 16 hours of discussion of research papers. The teachers present the basic material and then assign selected research papers to be presented by the students in class. Each student will be required to read and write a paper summary (evaluation) of each of the assigned papers before presenting the paper and participating in a discussion of the paper during class. The course will also include assignments in the form of small projects.

### *Conclusions and questions*

In this section, the materials for module PC2 of PhD course “Software Defined Networks and IoT” are presented. They can be used for preparation to lectures and self-learning. The materials were developed by Prof. I.S. Skarga-Bandurova, Ph.D. student M.V. Nesterov and Ph.D. student A.Y. Velykhanin.

Due to SDN is a recently emerging paradigm, there are only a limited number of studies on implementing specific QoS models over SDN. In this chapter, we review the existent algorithms and approaches to utilizing SDN technology in IoT and take a look at perspectives on SDN performance prediction using data fusion technique. Traditional and novel algorithms applicable in SDN can be used for the following tasks: SLA management; smart routing and optimal VM placement; solving controller placement problem; load balancing; performance prediction; intrusion detection and prevention. Some of them are tested and implemented well; others need to be improved that gives a wide corridor for new research and innovations.

In order to better understand and assimilate the course content that is presented in this section, we encourage you to answer the following questions.

1. What categories of algorithms are used applicable to SDN and IoT?
2. What parameters SLA agreement generally comprises?
3. Categories of SLA management.
4. SLA metrics.
5. How to overcome problems with server-side crashes caused by hardware crashes, such as hard disk or memory module crashes and program problems, such as program errors or configuration errors?

6. How QoS routing algorithms can be analyzed?
7. What metrics can be used to evaluate algorithms performance?
8. What is the tasks of the TE routing algorithm?
9. What QoS routing algorithms can be used for large-scale SDN?
10. Traffic scheduling algorithms.
11. What tasks are solved in traffic engineering for SDN?
12. What means good controller placement?
13. What metrics can be used to evaluate the position of the SDN-controller?
14. SDN-controller placement algorithms.
15. How reduce the load on the controller?
16. Load balance strategies.
17. What algorithms can be used to finding the optimal path in SDN networks?

### **References**

1. C. Lin, C. Wu, M. Huang, Z. Wen, Q. Zheng, "Performance Evaluation for SDN Deployment: an Approach based on Stochastic Network Calculus", *Wireless Communication over ZigBee for Automotive Inclination Measurement. China Communications*, vol. 13(1), pp. 98-106, 2016. DOI: 10.1109/CC.0.7560881.
2. S. Dixit, "Future of IMT Systems: Wireless World Vision 2020" *Reseach Forum*, 2013. [Online]. Available: [https://www.itu.int/en/ITU-D/Technology/Documents/Events2013/RegionalForumIMT\\_ARB\\_Tunis\\_May2013/Presentations/RegForumIMT\\_2013\\_ARB\\_Tunis\\_May13\\_Presentation\\_SDixit\\_2.pdf](https://www.itu.int/en/ITU-D/Technology/Documents/Events2013/RegionalForumIMT_ARB_Tunis_May2013/Presentations/RegForumIMT_2013_ARB_Tunis_May13_Presentation_SDixit_2.pdf) [Accessed: 25 December 2017].
3. J. Bendriss, I.G. Ben Yahia, D. Zeghlache, "Forecasting and anticipating SLO breaches in programmable networks", *20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, March 2017. doi:10.1109/icin.2017.7899402.
4. G. Choudhury, D. Lynch, G. Thakur, S. Tse "Two Use Cases of Machine Learning for SDN-Enabled IP/Optical Networks: Traffic Matrix Prediction and Optical Path Performance Prediction", *Arxiv.org*, 2019. [Online]. Available: <https://arxiv.org/abs/1804.07433> [Accessed: 23- Feb-2019].
5. S. Tomovic, I. Radusinovic, N. Prasad, "Performance comparison of QoS routing algorithms applicable to large-scale SDN networks", *IEEE EUROCON 2015 - International Conference on Computer as a Tool (EUROCON)*, September 2017, DOI: 10.1109/EUROCON.2015.7313698.

6. S. Das, Y. Yiakoumis, G. Parulkar, N. McKeown, P. Singh, D. Getachew, P.D. Desai, "Application-aware aggregation and traffic engineering in a converged packet-circuit network", *Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC)*, March 2011, pp.1-3.
7. R. Yanggratoke et al., "Predicting service metrics for cluster-based services using real-time analytics", *11th International Conference on Network and Service Management (CNSM)*, November 2015, pp. 135–143. DOI: 10.1109/CNSM.2015.7367349
8. D.O. Awduche, L. Berger, D. Gain, T. Li, G. Swallow, and V. Srinivasan. "Extensions to RSVP for LSP Tunnels", Internet Draft draftietf-mpls-rsvp-lsp-tunnel-04.txt, September 1999.
9. Ren, H., Li, X., Geng, J., & Yan, J. (2016). *A SDN-Based Dynamic Traffic Scheduling Algorithm*. 2016 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC).doi:10.1109/cyberc.2016.103
10. Kennington, J. and A. Madhavan. "Optimization Models and Algorithms for Minimizing the Maximum Link Utilization in Ospf Data Networks." "Technical report, <http://lyle.smu.edu/jlk/>. 2007.
11. Resende M and Pardalos P. Handbook of Optimization in Telecommunications[M]. New York, Springer Science +Business Media, 2006: 679-700.
12. Fortz B and Thorup M. Internet traffic engineering by optimizing ospf weights[C]. IEEE Infocom Proceedings, Tel Aviv, Israel, Aug, 2000, 2: 518-528.
13. Hock, D., Hartmann, M., Gebert, S., Jarschel, M., Zinner, T., & Tran-Gia, P. (2013). *Pareto-optimal resilient controller placement in SDN-based core networks*. *Proceedings of the 2013 25th International Teletraffic Congress (ITC)*.doi:10.1109/itc.2013.6662939
14. B. Heller, R. Sherwood, and N. McKeown, "The Controller Placement Problem," in Proc. of ACM HotSDN'12, pp. 7–12, August 2012.
15. M. Shindler. Approximation algorithms for the metric k-median problem. Written Qualifying Exam Paper, University of California, Los Angeles. Cited on, page 44.
16. V. Vazirani. Approximation algorithms. Springer Verlag, 2001.
17. D. Hochba. Approximation algorithms for np-hard problems. ACM SIGACT News, 28(2):40–52, 1997.
18. Y. Li, D. Pan, "OpenFlow based load balancing for Fat-Tree networks with multipath support", *12th IEEE International Conference on Communications (ICC'13)*, 2013, pp. 1-5.

19. I.F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "Research Challenges for Traffic Engineering in Software Defined Networks", [Online]. Available: <https://bwn.ece.gatech.edu/papers/2016/TrEnggSDN.pdf> [Accessed: 25 December 2017].
20. I.F. Akyildiz et al., "A Roadmap for Traffic Engineering in Software Defined Networks", *Computer Networks*, vol. 71, Oct. 2014, pp. 1–30.
21. M. Al-Fares et al., "Hedera: Dynamic Flow Scheduling for Data Center Networks" *Symposium on Networked Systems Design and Implementation*, vol. 10, April 2010, p. 19.
22. C. Rotsos, N. Sarrar, S. Uhlig, et al. "OFLOPS: An open framework for OpenFlow switch evaluation", *Passive and Active Measurement*. Springer Berlin Heidelberg, 2012, pp. 85-95.
23. M. Jarschel, F. Lehrieder, Z. Magyari, et al. "A flexible OpenFlow controller benchmark", *IEEE European Workshop on Software Defined Networking (EWSDN)*, 2012, pp. 48-53.
24. S.M. Al-Shehri, P. Loskot, and M. Mert, "Common Metrics for Analyzing, Developing and Managing Telecommunication Networks", *Arxiv.org*, 2019. [Online]. Available: <https://arxiv.org/ftp/arxiv/papers/1707/1707.03290.pdf> [Accessed: 2-May-2019].
25. F. Musumeci, C. Rottondi, A. Nag, I. Macaluso, D. Zibar, M. Ruffini, and M. Tornatore, "An Overview on Application of Machine Learning Techniques in Optical Networks" *Arxiv.org*, 2019. [Online]. Available: <https://arxiv.org/pdf/1803.07976.pdf> [Accessed: 2-May-2019].
26. I. de Miguel, R.J. Duran, T. Jimenez, N. Fernandez, J.C. Aguado, R.M. Lorenzo, A. Caballero, I.T. Monroy, Y. Ye, A. Tymecki et al., "Cognitive dynamic optical networks", *IEEE/OSA Journal of Optical Communications and Networking*, vol. 5, no. 10, pp. A107–A118, Oct. 2013.
27. C. Rottondi, L. Barletta, A. Giusti, and M. Tornatore, "Machinelearning method for quality of transmission prediction of unestablished lightpaths," *IEEE/OSA Journal of Optical Communications and Networking*, 2018, vol. 10, no. 2, pp. A286–A297.
28. S. Aladin and C. Tremblay, "Cognitive Tool for Estimating the QoT of New Lightpaths", in *Optical Fiber Communications Conference (OFC)* 2018, Mar. 2018.
29. R. Boada, R. Borkowski, and I. T. Monroy, "Clustering algorithms for Stokes space modulation format recognition", *Optics Express*, 2015, vol. 23, no. 12, pp. 15521–15531.
30. S. Gosselin, J. L. Courant, S. R. Tembo, and S. Vaton, "Application of probabilistic modeling and machine learning to the diagnosis of FTTH

GPON networks", *International Conference on Optical Network Design and Modeling (ONDM) 2017*, May 2017, pp. 1–3.

31. S. Shahkarami, F. Musumeci, F. Cugini, and M. Tornatore, "MachineLearning-Based Soft-Failure Detection and Identification in Optical Networks", *Optical Fiber Communications Conference (OFC) 2018*, Mar. 2018.

32. A. Jayaraj, T. Venkatesh, and C. S. Murthy, "Loss Classification in Optical Burst Switching Networks Using Machine Learning Techniques: Improving the Performance of TCP", *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 6, pp. 45–54, Aug. 2008.

33. M. Nesterov, I. Skarga-Bandurova "Troubleshooting and Performance Methodology for Business Critical Systems", *The 9th IEEE International Conference on Dependable Systems, Services and Technologies (DESSERT2018)*, May 2018, p. 551-555.

34. I. Skarga-Bandurova, M. Nesterov, T. Biloborodova, G. Krivoulya, I. Kotsiuba, O. Biloborodov, "Data Fusion Technique to Predicting Database Performance Issues", *Conf. Proceedings of 2019 IEEE 10th International Conference on Dependable Systems, Services and Technologies (DESSERT2019)*, UK, Leeds, June, 2019. In press.

35. J. Dezert, An introduction to DSMT. <http://fs.unm.edu/IntroductionToDSMT.pdf>. [Accessed 12 January 2019].

36. G. Shafer, "A Mathematical Theory of Evidence", Princeton Univ. Press, Princeton, NJ, 1976.

37. "Advances and applications of DSMT for information fusion" F. Smarandache, J. Dezert (Editors), American Research Press, Rehoboth, NM, U.S.A., Vol. 1–3, 2004–2009. [Online]. Available: <http://fs.gallup.unm.edu/DSMT.htm>. [Accessed 12 January 2019].

38. C. Chen-xiao, X. Ya-bin, "Research on Load Balance Method in SDN", *International Journal of Grid and Distributed Computing*, 2016, vol. 9(1), pp. 25–36. DOI:10.14257/ijgcd.2016.9.1.03.

39. A.F. Hani, I.V. Papatungan, M.F. Hassan, "Support Vector regression for Service Level Agreement violation prediction", *International Conference on Computer, Control, Informatics and Its Applications (IC3INA)*, November 2013. DOI: 10.1109/IC3INA.2013.6819192.

40. R.T. Kokila, S.T. Selvi, K. Govindarajan, "DDoS detection and analysis in SDNbased environment using support vector machine classifier", *IEEE Sixth International Conference on Advanced Computing (ICoAC)*, Chennai, India, December 2014, pp. 205-210.

41. T.A. Tang, L. Mhamdi, D. McLernon, et al. "Deep Learning Approach for Network Intrusion Detection in Software Defined Networking", *The International Conference on Wireless Networks and Mobile*

Communications (WINCOM'16), October 2016, Fez, Morocco. ISBN 978-1-5090-3837-4

42. S. Ali, V. Sivaraman, A. Radford and S. Jha, "A Survey of Securing Networks Using Software Defined Networking", *IEEE Transactions on Reliability*, vol. 64, no. 3, pp. 1086-1097, 2015. Available: 10.1109/tr.2015.2421391 [Accessed 28 July 2019].'

43. KTH | IK2220", *Kth.se*, 2019. [Online]. Available: <https://www.kth.se/student/kurser/kurs/IK2220?l=en>. [Accessed: 28- Jul-2019].

## 23. SDN IN CONTEXT OF DEVOPS TECHNOLOGY

Dr, Associated Prof. D. D. Uzun, Y.O. Uzun, DrS, Prof. V. S. Kharchenko

### *Contents*

|                                                   |     |
|---------------------------------------------------|-----|
| Abbreviations .....                               | 242 |
| 23.1 DevOps technology overview .....             | 243 |
| 23.1.1 Basic concepts and principles .....        | 243 |
| 23.1.2 Techniques and tools .....                 | 248 |
| 23.2 DevSecOpS.....                               | 256 |
| 23.2.1 Features and purposes.....                 | 256 |
| 23.2.2 Approaches .....                           | 258 |
| 23.3 SDN and DevOpS.....                          | 262 |
| 23.3.1 SDN and DevOpS interconnection.....        | 262 |
| 23.3.2 Leading practices for SDN and DevOps ..... | 266 |
| 23.4 DevOpS and IoT.....                          | 272 |
| 23.4.1 General .....                              | 272 |
| 23.4.2 Reasons DevOps matter in IoT.....          | 274 |
| 23.5 Work related analysis .....                  | 279 |
| Conclusion and questions .....                    | 279 |
| References .....                                  | 281 |

## **Abbreviations**

AWS – Amazon Web Service  
CI/CD - Continuous Integration and Continuous Delivery  
CDN – Content Delivery Network  
COTS – Commercial off the Shelf  
CSP – Communication Service Provider  
DAST – Dynamic Application Security Testing  
DevOps – Development and Operations  
DevSecOps – Development and Security Operations  
EPC – Evolved Packet Core  
IMS – IP Multimedia System  
LAN – Local Area Network  
MANO – Management and Orchestration  
MS – Microsoft  
NFV – Network Function Virtualization  
OLA – Organization Level Agreement  
OTT – Over-The-Top  
OWASP – Open Web Application Security Project  
SAST – Static Application Security Testing  
SCM – Source Control Management  
SDLC – Software Development Lifecycle  
SDN – Software Defined Networking  
SLA – Service Level Agreement  
SQM – Service Quality Management  
VNF – Virtual Network Functions  
WAN – Wide Area Network

## **23.1 DevOps technology overview**

As innovation accelerates and customer needs rapidly evolve, businesses must become increasingly agile. Time to market is key, and to facilitate overall business goals, IT departments need to be agile. Over the years software development lifecycles moved from waterfall to agile models of development. These improvements are moving downstream toward IT operations with the evolution of methodology Development and Operations (DevOps).

In order to meet the demands of an agile business, IT operations need to deploy applications in a consistent, repeatable, and reliable manner. This can only be fully achieved with the adoption of automation.

Widespread platforms, like AWS, MS Azure, Google Cloud, etc. support numerous DevOps principles and practices that IT departments can capitalize on to improve business agility.

This section focuses on DevOps principles and practices supported on the well-known platforms, like AWS, MS Azure, Google Cloud, etc. A brief introduction to the origins of DevOps sets the scene and explains how and why DevOps has evolved. Interconnection of DevOps, Software Defined Networks (SDN) and IoT is analysed.

### ***23.1.1 Basic concepts and principles***

DevOps is a new term that primarily focuses on improved collaboration, communication, and integration between software developers and IT operations. It's an umbrella term that some describe as a philosophy, cultural change, and paradigm shift.

Historically many organizations have been vertically structured with poor integration among development, infrastructure, security and support teams. Frequently the groups report into different organizational structures with different corporate goals and philosophies.

Deploying software has predominately been the role of the IT operations group. Fundamentally developers like to build software and change things quickly, whereas IT operations focus on stability and reliability. This mismatch of goals can lead to conflict, and ultimately the business may suffer.

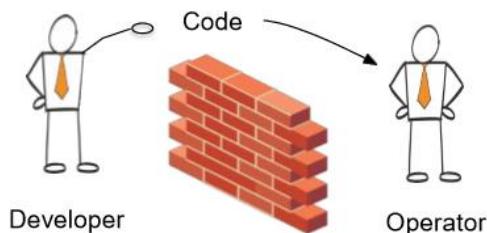


Figure 23.1 - Code transmission process

Today, these old divisions are breaking down, with the IT and developer roles merging and following a series of systematic principles:

- Continuous Integration/Continuous Delivery
- Infrastructure as code
- Continuous deployment
- Automation
- Monitoring
- Security

An examination of each of these principles reveals a close connection to the offerings available from Amazon Web Services.

*Agile Evolution to DevOps.* To fully appreciate DevOps principles, it is helpful to understand the context in which they evolved. The story begins with agile software development, which became popular over a decade ago and was seen as better approach to building software. Prior to agile, the dominant waterfall development methodology was based on a sequence starting with a requirements phase where 100% of the system under development was defined up front. The approach has shown itself to be inflexible and monolithic.

The agile model brought the concept of new and improved collaboration between business users and developers. Software development began to focus on iterations of working software that would evolve over time, delivering value along the way. Agile is a disciplined engineering process, and numerous tools now support it. For developers, such tools include IDEs, unit test frameworks, and code optimizers. As developers become more productive, the business becomes more agile and can respond to their customer requests more quickly and efficiently.

Over the last few years, the agile software development evolution has started to move downstream towards infrastructure under the label DevOps. Whereas agile software development primarily focuses on the collaboration between the business and its developers, DevOps focuses on the collaboration between developers, IT operations and security teams. IT operations include system administrators, database administrators, network engineers, infrastructure architects, and support personnel. Whereas agile software development provides business agility, DevOps provides IT agility, enabling the deployment of applications that are more reliable, predictable, and efficient.

DevOps practices vary with the task: With application development, DevOps focuses on code building, code coverage, unit testing, packaging, and deployment. With infrastructure, on the other hand, DevOps focuses on provisioning, configuration, orchestration, and deployment. But in each area the underlying principles of version management, deployment, roll back, roll forward, and testing remain the same.

*Continuous Integration.* Continuous integration is a software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

*Continuous Delivery.* Continuous delivery is a software development practice where code changes are automatically built, tested, and prepared for a release to production. It expands upon continuous integration by deploying all code changes to a testing environment and/or a production environment after the build stage. When continuous delivery is implemented properly, developers will always have a deployment-ready build artifact that has passed through a standardized Gecko Test process.

*Infrastructure as Code.* A fundamental principle of DevOps is to treat infrastructure the same way developers treat code. Application code has a defined format and syntax. If the code is not written according to the rules of the programming language, applications cannot be created. Code is stored in a version-management system that logs a history of code development, changes, and bug fixes. When code is compiled (built) into applications, we expect a consistent application

to be created. That is to say, the build is repeatable and reliable.

Practicing “infrastructure as code” means applying the same rigor of application code development to infrastructure provisioning. All configurations should be defined in a declarative way and stored in a version management system, just like application code. Infrastructure provisioning, orchestration, and deployment should support the use of the “infrastructure code.”

Until recently the rigor applied to application code development has not necessarily been applied to infrastructure. Frequently infrastructure is provisioned using manual processes. Scripts developed during the provisioning may not be stored in version control systems and the creation of environments is not always repeatable, reliable, or consistent.

In contrast, widespread platforms, like AWS, MS Azure, Google Cloud, etc. provide a DevOps-focused way of creating and maintaining infrastructure. Similar to the way software developers write application code, AWS and others provide similar services that enable the creation, deployment and maintenance of infrastructure in a programmatic, descriptive, and declarative way. These services provide rigor, clarity, and reliability. These services discussed in this paper are core to a DevOps strategy and form the underpinnings of numerous higher level cloud provider platform DevOps principles and practices.

*Continuous Deployment.* Continuous deployment is another core concept in a DevOps strategy. Its primary goal is to enable the automated deployment of production-ready application code.

Sometimes continuous deployment is referred to as continuous delivery. The only difference is that continuous deployment usually refers to production deployments.

By using continuous delivery practices and tools, software can be deployed rapidly, repeatedly, and reliably. If a deployment fails, it can be automatically rolled back to previous version.

*Blue–Green Deployment.* Blue–green deployment is a DevOps deployment practice that uses domain name services (DNS) to make application deployments. The strategy involves starting with an existing (blue) environment while testing a new (green) one. When the new environment has passed all the necessary tests and is ready to go live, you simply redirect traffic from the old environment to the new one via DNS.

The ability to create and dispose of identical environments easily in the cloud provider services makes DevOps practices like blue–green deployment feasible.

The blue–green deployment can be also used for back-end services like database deployment and failover.

*Automation.* Another core philosophy and practice of DevOps is automation. Automation focuses on the setup, configuration, deployment, and support of infrastructure and the applications that run on it. By using automation, you can set up environments more rapidly in a standardized and repeatable manner. The removal of manual processes is a key to a successful DevOps strategy. Historically, server configuration and application deployment have been predominantly a manual process. Environments become nonstandard, and reproducing an environment when issues arise is difficult.

The use of automation is critical to realizing the full benefits of the cloud. Widespread platforms, like AWS, MS Azure, Google Cloud, etc. relies heavily on automation to provide the core features of elasticity and scalability. Manual processes are error prone, unreliable, and inadequate to support an agile business. Frequently an organization may tie up highly skilled resources to provide manual configuration. Time could be better spent supporting other, more critical and higher value activities within the business.

Modern operating environments commonly rely on full automation to eliminate manual intervention or access to production environments. This includes all software releasing, machine configuration, operating system patching, troubleshooting, or bug fixing. Many levels of automation practices can be used together to provide a higher level end-to-end automated process.

Automation has many benefits:

- Rapid changes
- Improved productivity
- Repeatable configurations
- Reproducible environments
- Leveraged elasticity
- Leveraged auto scaling
- Automated testing

Automation is a cornerstone of cloud provider services and should be internally supported in all services, features, and offerings.

*Monitoring.* Communication and collaboration is fundamental in a DevOps strategy. To facilitate this, feedback is critical. Such core services should provide a robust monitoring, alerting, and auditing infrastructure so developers and operations teams can work together closely and transparently.

*Security.* In a DevOps enabled environment, focus on security is still of paramount importance. Infrastructure and company assets need to be protected, and when issues arise they need to be rapidly and effectively addressed.

### **23.1.2 Techniques and tools**

As already been said, DevOps, like agile, has evolved to encompass many different disciplines, but most people will agree on a few things: DevOps is a software development practice or a software development lifecycle (SDLC) and its central tenet is cultural change, where developers and non-developers all breathe in an environment where formerly manual things are automated; everyone does what they are best at; the number of deployments per period increases; throughput increases; and flexibility improves.

While having the right software tools is not the only thing you need to achieve a DevOps environment, some tools are necessary. A key one is continuous integration and continuous deployment (CI/CD). This pipeline is where the environments have different stages (e.g., DEV, INT, TST, QA, UAT, STG, PROD), manual things are automated, and developers can achieve high-quality code, flexibility, and numerous deployments.

This subsection describes a five-step approach to creating a DevOps pipeline, like the one in the following diagram, using open source tools.

**Step 1: CI/CD framework.** The first thing you need is a CI/CD tool. Jenkins, an open source, Java-based CI/CD tool based on the MIT License, is the tool that popularized the DevOps movement and has become the de facto standard.

Jenkins is an universal remote control tool that can manipulate with many different services and tools and orchestrate them. On its own, a CI/CD tool like Jenkins is useless, but it becomes more powerful as it plugs into different tools and services. Jenkins is just one

of many open source CI/CD tools that you can leverage to build a DevOps pipeline. Other open source CI/CD tools shown in Table 23.1.

Table 23.1 - Open source CI/CD tools

| Name                 | License                  |
|----------------------|--------------------------|
| <u>Jenkins</u>       | Creative Commons and MIT |
| <u>Travis CI</u>     | MIT                      |
| <u>CruiseControl</u> | BSD                      |
| <u>Buildbot</u>      | GPL                      |
| <u>Apache Gump</u>   | Apache 2.0               |
| <u>Cabie</u>         | GNU                      |

**Step 2: Source control management.** The best (and probably the easiest) way to verify that your CI/CD tool can perform some experience is by integrating with a source control management (SCM) tool. Why do you need source control? Suppose you are developing an application. Whenever you build an application, you are programming—whether you are using Java, Python, C++, Go, Ruby, JavaScript, or any of the gazillion programming languages out there.

The programming codes you write are called source codes. In the beginning, especially when you are working alone, it's probably OK to put everything in your local directory. But when the project gets bigger and you invite others to collaborate, you need a way to avoid merge conflicts while effectively sharing the code modifications.

You also need a way to recover a previous version—and the process of making a backup and copying-and-pasting gets old. You (and your teammates) want something better.

This is where SCM becomes almost a necessity. A SCM tool helps by storing your code in repositories, versioning your code, and coordinating among project members.

Although there are many SCM tools out there, Git is the standard and rightly so. I highly recommend using Git, but there are other open

source options if you prefer. Open source SCM tools shown in Table 23.2.

Table 23.2 – Open source SCM tools

| Name                                    | License           |
|-----------------------------------------|-------------------|
| <u>Git</u>                              | GPLv2 & LGPL v2.1 |
| <u>Subversion</u>                       | Apache 2.0        |
| <u>Concurrent Versions System (CVS)</u> | GNU               |
| <u>Vesta</u>                            | LGPL              |
| <u>Mercurial</u>                        | GNU GPL v2+       |

The CI/CD tool can automate the tasks of checking in and checking out source code and collaborating across members.

**Step 3: Build automation tool.** Now you can check out the code and commit your changes to the source control, and you can invite your friends to collaborate on the source control development. But you haven't yet built an application.

To make it a web application, it has to be compiled and put into a deployable package format or run as an executable. (Note that an interpreted programming language like JavaScript or PHP doesn't need to be compiled.)

Enter the build automation tool. No matter which build tool you decide to use, all build automation tools have a shared goal: to build the source code into some desired format and to automate the task of cleaning, compiling, testing, and deploying to a certain location.

The build tools will differ depending on your programming language, but here are some common open source options to consider. Open source build automation tools are shown in Table 23.3.

**Step 4: Web application server.** So far, you have a packaged file that might be executable or deployable. For any application to be truly useful, it has to provide some kind of a service or an interface, but you need a vessel to host your application.

Table 23.3 - Open source build automation tools

| Name           | License     | Programming Language |
|----------------|-------------|----------------------|
| <u>Maven</u>   | Apache 2.0  | Java                 |
| <u>Ant</u>     | Apache 2.0  | Java                 |
| <u>Gradle</u>  | Apache 2.0  | Java                 |
| <u>Bazel</u>   | Apache 2.0  | Java                 |
| <u>Make</u>    | GNU         | N/A                  |
| <u>Grunt</u>   | MIT         | JavaScript           |
| <u>Gulp</u>    | MIT         | JavaScript           |
| <u>Buildr</u>  | Apache      | Ruby                 |
| <u>Rake</u>    | MIT         | Ruby                 |
| <u>A-A-P</u>   | GNU         | Python               |
| <u>SCons</u>   | MIT         | Python               |
| <u>BitBake</u> | GPLv2       | Python               |
| <u>Cake</u>    | MIT         | C#                   |
| <u>ASDF</u>    | Expat (MIT) | LISP                 |
| <u>Cabal</u>   | BSD         | Haskell              |

For a web application, a server is that vessel. An application server offers an environment where the programming logic inside the

deployable package can be detected, render the interface, and offer the web services by opening sockets to the outside world. You need an HTTP server as well as some other environment (like a virtual machine) to install your application server. For now, let's assume you will learn about this along the way (although I will discuss containers below). There are a number of open source web application servers available, shown in Table 23.4.

Table 23.4 – Open source web application servers

| Name                | License                | Programming Language |
|---------------------|------------------------|----------------------|
| <u>Tomcat</u>       | Apache 2.0             | Java                 |
| <u>Jetty</u>        | Apache 2.0             | Java                 |
| <u>WildFly</u>      | GNU Lesser Public      | Java                 |
| <u>GlassFish</u>    | CDDL & GNU Less Public | Java                 |
| <u>Django</u>       | 3-Clause BSD           | Python               |
| <u>Tornado</u>      | Apache 2.0             | Python               |
| <u>Gunicorn</u>     | MIT                    | Python               |
| <u>Python Paste</u> | MIT                    | Python               |
| <u>Rails</u>        | MIT                    | Ruby                 |
| <u>Node.js</u>      | MIT                    | Javascript           |

Now the DevOps pipeline is almost usable. Although it's possible to stop here and integrate further on your own, code quality is an important thing for an application developer to be concerned about.

**Step 5: Code testing coverage.** Implementing code test pieces can be another cumbersome requirement, but developers need to catch any errors in an application early on and improve the code quality to ensure end users are satisfied. Luckily, there are many open source tools

available to test your code and suggest ways to improve its quality. Even better, most CI/CD tools can plug into these tools and automate the process. There are two parts to code testing: *code testing frameworks* that help write and run the tests (shown in Table 23.5), and *code quality suggestion tools* (shown in Table 23.6) that help improve code quality.

Table 23.5 - Code test frameworks

| Name              | License                | Programming Language |
|-------------------|------------------------|----------------------|
| <u>JUnit</u>      | Eclipse Public License | Java                 |
| <u>EasyMock</u>   | Apache                 | Java                 |
| <u>Mockito</u>    | MIT                    | Java                 |
| <u>PowerMock</u>  | Apache 2.0             | Java                 |
| <u>Pytest</u>     | MIT                    | Python               |
| <u>Hypothesis</u> | Mozilla                | Python               |
| <u>Tox</u>        | MIT                    | Python               |

Table 23.6 - Code quality suggestion tools

| Name               | License                | Programming Language |
|--------------------|------------------------|----------------------|
| <u>Cobertura</u>   | GNU                    | Java                 |
| <u>CodeCover</u>   | Eclipse Public (EPL)   | Java                 |
| <u>Coverage.py</u> | Apache 2.0             | Python               |
| <u>Emma</u>        | Common Public License  | Java                 |
| <u>JaCoCo</u>      | Eclipse Public License | Java                 |
| <u>Hypothesis</u>  | Mozilla                | Python               |

---

| Name           | License | Programming Language |
|----------------|---------|----------------------|
| <u>Tox</u>     | MIT     | Python               |
| <u>Jasmine</u> | MIT     | JavaScript           |
| <u>Karma</u>   | MIT     | JavaScript           |
| <u>Mocha</u>   | MIT     | JavaScript           |

Note that most of the tools and frameworks mentioned above are written for Java, Python, and JavaScript, since C++ and C# are proprietary programming languages (although GCC is open source).

Now that you've implemented code testing coverage tools, your DevOps pipeline should resemble the DevOps pipeline diagram shown at the beginning of this tutorial.

**Optional steps. Containers.** As mentioned above, you can host your application server on a virtual machine or a server, but containers are a popular solution. The short explanation is that a VM needs the huge footprint of an operating system, which overwhelms the application size, while a container just needs a few libraries and configurations to run the application.

There are clearly still important uses for a VM, but a container is a lightweight solution for hosting an application, including an application server. Although there are other options for containers, Docker and Kubernetes are the most popular.

**Optional steps. Middleware automation tools.** Our DevOps pipeline mostly focused on collaboratively building and deploying an application, but there are many other things you can do with DevOps tools. One of them is leveraging Infrastructure as Code (IaC) tools, which are also known as middleware automation tools. These tools help automate the installation, management, and other tasks for middleware software.

For example, an automation tool can pull applications, like a web application server, database, and monitoring tool, with the right configurations and deploy them to the application server. Several open source middleware automation tools are presented in Table 23.7.

Table 23.7 – Open source middleware automation tools

| Name             | License       |
|------------------|---------------|
| <u>Ansible</u>   | GNU Public    |
| <u>SaltStack</u> | Apache 2.0    |
| <u>Chef</u>      | Apache 2.0    |
| <u>Puppet</u>    | Apache or GPL |

This is just the tip of the iceberg for what a complete DevOps pipeline can look like. Start with a CI/CD tool and explore what else you can automate to make your team's job easier.

## 23.2 DevSecOpS

### 23.2.1 Features and purposes

The development of information technologies in the end of the past - the beginning of the present centuries has led to the emergence of a new direction in science - information security. This issue has been devoted to monographs, scientific articles by many scientists. The works have shown the dependence of information security on time and presented its mathematical model. Further development of information security was cybersecurity (CS), which replaced it in information technology. One of the main preventative methods of providing CS is to find and eliminate various vulnerabilities that arise in the process of software development. Vulnerabilities allow for various attacks that can be used to access websites of different companies and agencies, credit card information, personal data of citizens, etc.

An analysis of recent research and publications indicates that over the last 2 years, about 216 unique incidents have been identified among ordinary users related to cyberattacks (accounting for 26% of the total). The most stolen data is medical information and payment card details. Software applications are complex and can potentially have many different security issues. Problems range from bad code to improperly configured servers and all hardware in between. There are currently

many ways to test the security of a software product with its advantages and disadvantages.

Despite this, the problem of hacking and hacking is still relevant and, unfortunately, is picking up at the same speed with which information technology is evolving. One way to solve this problem is to create a DevSecOps software. More precisely, it is not a software product, but a new ideology in which every programmer is responsible for the safety of their product.

The purpose of DevSecOps is to bring developers of all areas to a high level of professionalism in the field of security in a short period of time. DevSecOps aims to ensure that every security flaw, upon detection, is timely identified and recorded, autotested, and added to a permanent build process to close the most urgent and important security gaps.

Figure 23.2 shows how DevSecOps integrates into software development. DevSecOps basic concepts will be presented the analysis of the technology of using the DevSecOps software for security testing in the form of approaches or ideas used to achieve this goal.

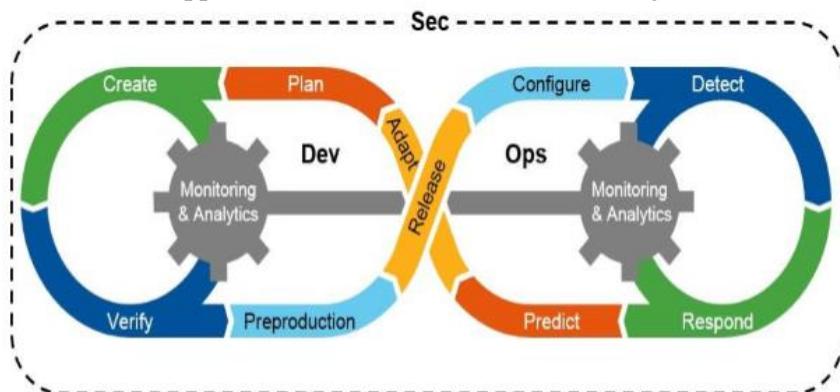


Figure 23.2 - DevSecOps integration into software development lifecycle

### 23.2.2 Approaches

**Approache 1.** “Automate {in origin “kill”} them all”. Speed is one of the main strengths of DevOps. In the context of continuous integration and continuous deployment (CI / CD), the ability to quickly

get a working software product. For the sake of security, to be part of this workflow, it must be automated. Security controls and tests must be implemented early on and throughout the development lifecycle, and they should be driven automatically as developers introduce new versions of the code 50 times a day for a single application. Automation has become a key feature of DevSecOps in integrating with the very mature DevOps practice. This is a great advantage over the Waterfall Development Model, where automatic safety tests are launched just before they are released. More and more tools have emerged with a wide range of security analysis and testing capabilities throughout the software development lifecycle, from source code analysis to post-deployment integration and monitoring.

These include Checkmarx, Splunk, Contrast Security, Sonatype, Tanium, InSpec, FireEye, and Metasploit. You need to automate thoughtfully. Trying to run automatic testing of the entire program source code every day can be time consuming and you may lose track of daily changes. Consideration should also be given to implementing automated Dynamic Application Security Testing (DAST) in the software development lifecycle. Unlike static analysis that focuses on search potential security issues in the code itself, DAST looks for real-time vulnerabilities while the application is running. DAST Automation scans and runs tests against recent or new code changes to identify security vulnerabilities listed in the Open Web Application Security Project (OWASP) list the most common flaws, such as SQL injection errors, that can be skipped during static analysis of a program code that can be skipped during static analysis of applications code.

**Approach 2.** Checking the vulnerability of the generated code. Despite growing concern about the risks of using third-party software components, companies use software with open source applications. A separate audit conducted by the company in more than 1,000 commercial applications showed that 96% of them include open source components. More than 6 in 10 applications contain known security vulnerabilities in these components, and some have been there for four years. Despite this, only 27% of respondents said they have processes for automatically identifying and tracking patches for known shortcomings in open source software.

Understanding the use of open source is the key to more extensive adoption of DevSecOps methods. Developers often do not have time to

view the code in their open source libraries or read documentation, so automatic processes for managing open source and third-party components are a major requirement for DevSecOps. During the work it is necessary to monitor the use of open source contextual and other vulnerabilities in the code and what impact these vulnerabilities may have on the dependent code. Code dependency checks are fundamental to DevSecOps, and utilities like OWASP Dependency-Check do not allow you to use code with known software vulnerabilities. OWASP works by testing code and dependent open source or component libraries to find out any key disadvantages of OWASP. It works with a constantly updated database of all known open source software vulnerabilities.

**Approach 3.** Trusting the Tool SAST tools allow developers to get instant feedback on defects that can cause security issues while writing code, test the code as they write it, to get instant feedback on defects that can cause security issues.

These tools help developers identify and eliminate potential security vulnerabilities during the normal workflow, and should therefore be an important component of DevSecOps practice. However, the key to implementing such tools is to think little. Often when a security team implements a static test tool in the CI / CD pipeline, the team tends to include checks on a whole host of security issues and this ends up with other problems that have the difficulty of supporting such processes. Instead, it is much better to include one or two security checks at a time and get developers to use the idea of security rules in the workflow. For example, with the implementation of the SAST tool in development, it is possible to start by including a set of tests to capture SQL injection errors.

As soon as developers find out how the tool helps them catch coding errors, they are more likely to work with it. "Before you incorporate more and more rules, you need to build trust in the tool."

**Approach 4.** Some tools may be more useful than others. Every day, there are new tools needed for security testing, so there are several key considerations when buying them:

- Security products should be able to integrate into the development pipeline and allow the development and security team to work together, not just throwing things at the fence to each other. The security testing product should make it easy for developers to quickly

initiate testing and get results without having to leave their existing toolkit;

- Other key requirements are speed and accuracy. Security must work quickly. But false positives can be an absolute killer in a DevOps environment;

- A tool that requires a developer or security engineer to take a timeout to verify test results is of little help. The results generated by the tools should be fast, accurate and immediate. Tools are needed to help developers identify and prioritize vulnerabilities as they write software. Code vulnerability identification should be based on an understanding of the software itself, as opposed to comparing it with signatures.

**Approach 5.** Threat modeling is difficult but still necessary. It is recommended to use threat modeling and risk assessment before moving to DevSecOps. A threat modeling exercise can help a security organization better understand asset threats, asset types and sensitivities, existing asset security controls, and any gaps in controls that need to be addressed. Such assessments can help identify deficiencies in application architecture and design would miss other security approaches.

Doing threat simulation in a DevOps environment can be tricky because it can slow down the speed of the CI / CD process. You cannot automate threat mapping processes in the same way as for any other DevOps boundary. Value for overall success DevOps efforts because it causes developers to think about their software from the perspective of an attacker.

**Approach 6.** Teaching Developers to Secure Coding. There may be some problems with using DevSecOps. One of the biggest is simply convincing developers of the feasibility of taking this approach. Getting the investment and time it takes to prepare a development team for secure coding is another big problem. Developers often don't know that they are coding in a dangerous way. Many DevSecOps methods and tools are still being developed, and many are still unknown in DevSecOps technology. But it is obvious that in a world of continuous integration and rapid release cycles, it is no longer possible to ignore application security. Advantages and disadvantages of DevSecOps. It's easy to spell out DevSecOps's benefits - automating processes from the very beginning of the programming process reduces

the likelihood of incorrect administration and errors that often lead to downtime or open up opportunities for attacks. Automation also eliminates the need for IB specialists to configure the console manually.

Thus, security features such as identity management, access (IAM), firewall operation, and vulnerability scanning are programmatically activated in the DevOps process. As a result of this approach, teams of ISPs can focus on policy setting. Experts estimate that 80% of development teams will use DevSecOps by 2021. Figure 23.2 shows how DevSecOps integrates into application development. One of the major benefits of DevSecOps is that every team involved in development is responsible for security. This approach leads to the creation of special tools aimed at enhancing security at different stages of the DevOps pipeline. A major drawback is the fact that the number of security professionals well-versed in DevSecOps still remains depressingly small. To train such professionals, it is necessary to study courses that are exotic to most programmers. CS specialists focus on endpoint security, so they are not particularly interested in DevSecOps.

### **23.3 SDN and DevOpS**

#### ***23.3.1 SDN and DevOpS interconnection***

The introduction of Software Defined Networking (SDN) and Network Functions Virtualization (NFV) has ushered in a new era of innovation that enables communication service providers (CSPs) to create highly automated networks and introduce new customized services. Leading CSPs recognize that innovation does not only come from within; they are constantly looking outside the organization for partners with whom they can jointly capitalize on new market opportunities. An innovative professional service partner can help CSPs take advantage of these immediate opportunities while facilitating a long-term transformation strategy to achieve a sustainable competitive advantage.

The evolution of virtualization technology has disrupted traditional service delivery. Alternative cloud service or Over-The-Top (OTT) providers such as Skype and Line 2 are leveraging these virtualization technologies to rapidly roll out a new platform and services. As enterprises and consumers shift their applications to a cloud-based environment, these nimble OTT players, supported by automated and

programmable platforms, can swiftly scale up new services to address unanticipated demands as well as “fail fast” by almost instantaneously scaling down unsuccessful services. Rapid innovation has slowly but surely rendered conventional network connectivity a commodity.

Telecommunication networks are migrating from traditional hardware and appliance-centric deployments to cloud-based deployments, with software as the critical component of all network functionality. At the heart of this revolution are two technologies: Network Function Virtualization (NFV), and Software Defined Networking, both of which aim at virtualizing network applications as well as the network connectivity. Both these technologies, and the interaction between them, have been undergoing trials over the past few years, and new standards as well as architecture options have begun to emerge.

While most of the initial focus has centered on defining the solution architecture, the stakeholders responsible for operating these networks, so, network and IT operations teams, need to still iron out operational aspects which are critical to seamless delivery of end user services. Migration from network element-centric to software centric operations will drive fundamental changes in the network operating model across multiple dimensions, from tighter integration across network, IT, and architecture teams to new processes, and tools to manage the network.

In this point of view, we present some of the leading practices for software-centric network operations, based on successful early stage implementations, that can help Communications Service Providers (CSPs) effectively manage their services and end user experience within the NFV and SDN domain. Key drivers for migration to NFV and SDN are presented on Figure 23.3.

### 23. SDN in Context of Devops Technology

| Challenge                                                                                                                                 | Driver                                                                                                                                                                                                                                | Impact                                                                                                                                                                                                                                                                                       |
|-------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  <p><b>Increased revenue to cost disparity</b></p>       | <ul style="list-style-type: none"> <li>• Large CAPEX required to support increasing traffic volumes</li> <li>• Rapidly declining revenues (on a per traffic unit basis)</li> </ul>                                                    |  <ul style="list-style-type: none"> <li>• Reduced service provider profitability</li> <li>• Continued growth in expense – both CAPEX and OPEX</li> </ul>                                                    |
|  <p><b>Increased proprietary hardware in network</b></p> | <ul style="list-style-type: none"> <li>• OEM-built differentiators into proprietary hardware and software</li> <li>• Historically, specialized hardware has provided better performance</li> </ul>                                    |  <ul style="list-style-type: none"> <li>• Vendor lock-in leading to high costs for expansion and upgrade</li> <li>• Limited ability to introduce new vendors due to compatibility issues</li> </ul>         |
|  <p><b>Heavy effort to launch new services</b></p>       | <ul style="list-style-type: none"> <li>• Custom development to ensure that proprietary hardware and software systems work together</li> <li>• Coordination required across multiple organizational silos to launch service</li> </ul> |  <ul style="list-style-type: none"> <li>• Limited ability to challenge competition such as OTT players with disruptive offers</li> <li>• Increased time to catch up with competition when needed</li> </ul> |
|  <p><b>Limited flexibility and agility</b></p>           | <ul style="list-style-type: none"> <li>• Network resources are mostly static and tied to a geographic location</li> <li>• Dynamic adjustments to alleviate traffic hotspots and congestion not supported</li> </ul>                   |  <ul style="list-style-type: none"> <li>• Networks need to be over-provisioned to handle worst case traffic scenarios</li> <li>• Sub-optimal utilization of network resources</li> </ul>                    |

Figure23.3 - Key drivers for migration to NFV and SDN

Changing business dynamics and operational challenges are driving the shift to SDN/ NFV.

From a CSP perspective, while user data traffic has been growing exponentially with the increase in OTT and other data-centric services driving high capital investments, revenues have not kept pace. NFV and SDN are complementary technologies which leverage cloud infrastructure and can help both increase revenues with the rapid introduction of new services, and reduce expenses by shifting from expensive proprietary hardware to lower cost commodity hardware.

With NFV, functionality such as firewalls, load balancers, deep packet inspection and IP Multimedia System (IMS) nodes which were traditionally implemented with hardware-based appliances, are

delivered as software-based Virtual Network Functions (VNFs) on a carrier-grade cloud infrastructure. SDN, on the other hand, simplifies the connectivity between physical and virtual network elements at layer 2/3 via network virtualization protocols such as OpenFlow.

NFV and SDN together offer an elegant solution for CSPs looking to address the challenges driven by business dynamics and operational considerations for today's telecom networks. Some key underlying industry and business drivers for migration towards NFV and SDN are shown in Figure 23.4.

NFV and SDN will introduce a radical shift in telecom network architecture. NFV and SDN principles can be applied to most telecom access and core level network elements. Transport elements such as routers and switches in both the Local Area Network (LAN) and Wide Area Network (WAN) network segments can be replaced by commodity switches supporting SDN approaches such as OpenFlow.

Similarly, most hardware-based elements such as IMS nodes, Evolved Packet Core (EPC) platforms and Content Delivery Network (CDN) platforms, can all be virtualized on the cloud infrastructure with NFV. CSPs are also exploring alternatives to migrate hardware-based Customer Peripheral Equipment (CPE), such as Set Top Boxes (STBs), to NFV based principles.

As illustrated in Figure 23.4., in a completely virtualized environment, most services would be based on a common cloud platform which can deliver Telco-grade capabilities.

However, the migration path towards the network shown in Figure 23.4. is likely to be a phased one, primarily governed by business decisions and investment lifecycles.

Operations teams will be faced with the reality of needing to manage hybrid deployments, including both physical and virtual network elements, for an extended period of time, and need to be equipped with the necessary tools, processes, and skills to do so.

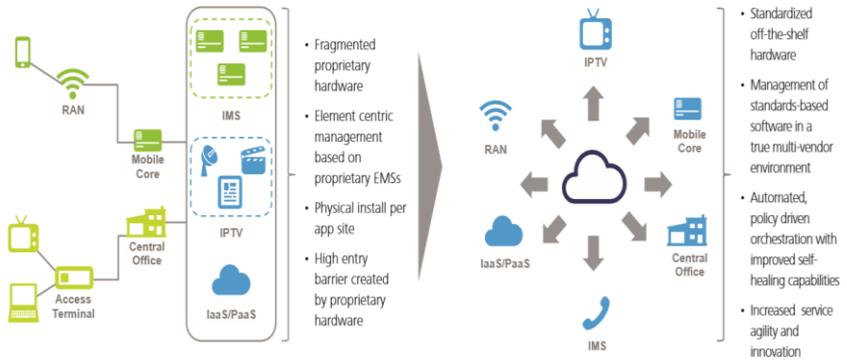


Figure 23.4 - Telecom Architecture shift towards a NFV/SDN Environment

Migration to the NFV/SDN architecture will impact most operations functions

With the migration of networks to a virtualized and software-centric model, current operations functions and processes need to undergo major changes to ensure delivery of carrier grade performance. Key considerations for effective operations in NFV/SDN networks include:

- Service strategy and design needs to maintain status quo in terms of operational performance for traditional services being migrated to NFV/SDN.
- Carrier grade performance needs to be ensured by leveraging features such as dynamic creation and migration of virtual network functions to meet availability requirements.
- Operations needs to migrate to “management by exception” wherein most common errors and performance degradations are addressed via automated self-healing and self-optimization rules.
- Critical functions such as fault, outage, and performance management need to be supported with smooth handoffs across different teams which maintain physical and virtual network resources.
- The skillset of operations teams needs to be expanded to include scripting capabilities (or their equivalent via GUI-based tools) to be able to effectively create “recipes” for managing software VNFs.

- A DevOps-based model which drives closer coordination between operations and development teams needs to be introduced to improve service agility and quality.

The role of operations spans across the entire service lifecycle, and each of these stages is impacted by the introduction of NFV and SDN based networks. The entire operations model including processes, tools and technology, as well as people and organization needs to be redesigned for each functional area within Service Design and Fulfillment, Service Operations and Readiness, as well as Service Assurance, as shown on Figure 23.5.

| Service lifecycle area           | Functional domains            |
|----------------------------------|-------------------------------|
| Service fulfillment              | Activation and provisioning   |
|                                  | Change management             |
| Operations support and readiness | Capacity management           |
|                                  | Inventory management          |
|                                  | DevOps                        |
| Service assurance                | Performance management        |
|                                  | Fault management              |
|                                  | SLA/OLA management            |
|                                  | End-to-end service management |

Figure 23.5 - Functional Domains by Service Lifecycle

### 23.3.2 Leading practices for SDN and DevOps

**Leading practices for service fulfillment.** *Activation and provisioning.* For an ideal service launch experience, it is necessary to ensure that setup and end-to-end orchestration via Management and Orchestration (MANO) happens without any errors. This can be ensured by leveraging the following practices:

- Activation and provisioning needs to be enabled via an intuitive portal which provides a simplified workflow, and pre-defined templates

for standard activities such as service definition and composition, service activation, as well as service modification.

- Provisioning should be based on industry standard protocols such as YANG, NETCONF, and TOSCA – which enable end-to-end chaining of components from multiple vendors in a seamless manner. Newer protocols also include support for rollback, to enable a revert to the original configuration in case any step of the end-to-end provisioning fails.

***Leading practices for operations support and readiness.***

*Change management.* One of the key benefits of implementing a SDN and NFV based network is increased agility. This is, in part, enabled by the fewer errors in change management because fewer manual steps are needed for sign-off and change implementation. The following practices can help in creating an automated change management process, which speeds up realization of the approved changes:

- A software-based workflow should be implemented to acquire approvals for changes, and automatically effect approved changes via the centralized orchestrator.

- Logically isolated test environments, built using SDN, can provide the ability to simulate multiple What-If scenarios and quantify impact of planned changes in a staging environment.

*Inventory Management.* In the NFV and SDN world, inventory management needs to be considered at the service, virtual network application, and resource level (Virtual Machines (VM) and physical server). With a highly dynamic virtual environment, one click access to the most up to date inventory becomes a necessity. To support these requirements, the following leading practices need to be adopted:

- Physical inventory data needs to be enhanced to include VNF and virtual network details in order to build an integrated view of utilization of logical and virtual resources across the infrastructure.

- A software repository will be needed to maintain details such as package versions and license usage.

- Auto discovery algorithms and version controlled archival systems need to be implemented which can help establish a real-time topology view and inventory reporting system. This reduces troubleshooting issues by providing the ability to identify the exact

topology at the time of an event.

*DevOps.* To achieve improved multi-service release stability and greater deployment agility, network changes will need to be managed in a methodical and consistent way, while eliminating need for device/hardware specific scripts, and reliance on specific team members. To meet this requirement in a virtualized and software defined network infrastructure engineers need to apply DevOps principles pioneered in the enterprise cloud environment. Some leading DevOps practices are as follows:

- The network's tolerance for frequent changes needs to be increased by automating testing and deployment of changes across multiple non-production and production environments.

- Creating an automated test suite allows changes to be verified and risks to be identified through event driven triggers across multiple environments, thus, avoiding last-minute surprises.

- Operations is deeply involved with solution design and testing of end-to-end capabilities prior to the software drops in the production environment. Feedback from operations on production networks is tracked, maintained, and rolled into subsequent product sprints.

### ***Leading practices for service assurance.***

*Performance management.* For effective management of services in the virtualized environment where performance is highly dependent on underlying cloud infrastructure, self-learning and predictive techniques must be developed to manage end-to-end service performance by intelligently correlating inputs at all levels and across locations. This can be achieved by adopting some of the leading practices as outlined below:

- New or revised KPIs/KQIs e.g., Infrastructure Response Time, VNF Contention Analysis, and sophisticated algorithms need to be defined that can correlate inputs at all levels and provide insightful performance views across VNFs and virtual infrastructure.

- Predictive analytics needs to be leveraged to proactively manage resources based on predicted faults, dynamically update policies and rules based on real-time traffic characteristics. This can help minimize the occurrence of issues across the virtualized infrastructure.

- Self-optimization capabilities need to be introduced in performance management modules which can optimize configuration

based on current network performance e.g., scale up VMs, add new VNF instances for load balancing, configure new routes between VMs, etc..

*Fault management.* Early fault detection and mitigation is key to deliver carrier grade availability and improve end user customer experience. With the ability to proactively correlate physical and virtual level faults at a service level and performing VNF/network topology reconfiguration, Mean Time To Repair (MTTR) can be greatly reduced. Leading practices for proactive fault management include:

- The service model should be leveraged to identify all components and links impacted by a particular fault. This can be done by using the YANG model to identify which components of a service are impacted, trigger policy based alarms, and suppress duplicate alarms.

- Policy driven self-healing strategies need to be implemented to route around faults identified via monitoring of various instances of a VNF across VMs and performing distributed failure checks.

*SLA/OLA management.* To be able to maximize benefits from the use of virtualization, stringent Service Level Agreements (SLAs) need to be enforced onto the groups providing operations support for the underlying cloud infrastructure. This is needed to ensure that the carrier-grade requirement for availability (e.g., 5 nines) and other regulatory (e.g., NEBS) compliance requirements are met.

Additionally, Organization Level Agreements (OLAs) also need to be updated to encompass all types of VNFs hosted in the network. Implementing the following practices will ensure effective SLA/OLA management:

- Carrier-grade SLA/OLAs need to be enforced on Commercial off the Shelf (COTS) hardware and software components to ensure that off-the-shelf solutions can support carrier-grade network requirements. These SLAs and OLAs also need to be enforced across organizations supporting the underlying platform on which network services are provided.

- A common SLA/OLA framework needs to be established with all vendors providing software-based VNFs or controllers. While the framework can be used to establish implementation guidelines, it must be flexible enough to support different requirements based on VNF type.

- SLAs and OLAs need to include key operational parameters such as service response time and scalability, packets lost, etc. and not be limited to the time in which an assigned ticket is acknowledged. End-to-end Service Management.

To manage and meet expectations on a per-customer basis for multiple services the focus needs to shift from merely monitoring network and node level KPIs, and turn towards analysis and correlation of performance at every layer of the network stack. The following practices will enable this correlation:

- End-to-end ) with integrated dashboards which provide the ability to drill down along the VNF chain all the way to the underlying virtual and physical resources and help localize issues.

- Cross domain correlation based on metrics for service accessibility, integrity, and retention which are built on new/revised KPIs/KQIs with inputs from VNFs, virtualized infrastructure, and network layers.

So, the evolution of virtualization technology has disrupted traditional service delivery. Alternative cloud service or OTT providers such as Skype and Line 2 are leveraging these virtualization technologies to rapidly roll out a new platform and services.

As enterprises and consumers shift their applications to a cloud-based environment, these nimble OTT players, supported by automated and programmatic platforms, can swiftly scale up new services to address unanticipated demands as well as “fail fast” by almost instantaneously scaling down unsuccessful services. Rapid innovation has slowly but surely rendered conventional network connectivity a commodity.

*Transforming to a Virtualized Environment.* To remain relevant in today’s market and avoid marginalization, CSPs must leverage the latest SDN and NFV innovations to provide virtualized end-to-end solutions that immediately address customers’ evolving requirements. CSPs also need to initiate a foundational transformation to build a sustained long-term competitive advantage. It can be defined as a foundational transformation in the business model, service development processes, skills, and culture.

*Business Model Evolution.* Traditional network infrastructures are designed and deployed in a rigid and complex fashion, with hardcoded workflows and limited flexibility. Service deployment can take from 12

to 18 months, require large upfront capital investments, and demand significant resources to integrate, test, and deploy. As a result, CSPs have traditionally taken a risk-averse approach to new service deployment, limiting their ability to respond to market changes and exploit new opportunities.

A virtualized network based on SDN and NFV technologies transforms this business model and disrupts traditional network economics. Virtualization technologies can significantly reduce upfront capital expenses (CapEx), while a highly scalable and flexible IP infrastructure layer can be optimized instantaneously for efficiency, lowering operational expenses (OpEx). An automated service orchestration layer improves time to market, enabling CSPs to quickly capitalize on new market opportunities with new services.

This increased agility enables CSPs to transform their business, allowing them to offer new services and data analytics as part of a platform-based, on-demand, and pay-as-you-grow model. Lower CapEx and OpEx also allow CSPs to effectively expand their service footprint and target new customer segments and geographies. The resulting expansion of the service portfolio increases customer relevance and drives profitability.

*DevOps Practice and Agile Development Methods.* Service development has traditionally relied on a waterfall process comprised of multiple stages, each with highly defined requirements that must be completed sequentially. Features are predefined and functionalities are delivered all at once.

Needless to say, traditional service development is a lengthy process, compounded by the need to perform time-consuming manual testing over a complex hardware-centric infrastructure.

As a result, by the time the service or application is finally delivered, the market has moved on and customer requirements have evolved.

The emergence of DevOps, a new collaborative practice, establishes a process that involves developers and operational organizations collaborating, facilitating an exchange of ideas, and expediting decision making processes that lead to real action.

Agile development, on the other hand, is a software development methodology involving cross-functional teams defined within the DevOps process. The agile development approach promotes service

flexibility, where software development focuses on evolutionary development, early delivery of incremental features, and continuous improvements.

Moving towards a combined DevOps practice and an agile development methodology enables CSPs to dramatically accelerate the development process, reducing service delivery from months or years to mere days - all while continuously delivering relevant innovation.

## **23.4 DevOpS and IoT**

### ***23.4.1 General***

It's increasingly apparent that the development of software for the Internet of Things (IoT) and the management of those systems once they are in operation cannot be separated making IoT software an area ripe for "DevOps." More than a buzzword, DevOps has the potential to help accelerate system development, ensure system quality, and optimize system reliability in the field.

DevOps is already being used in the IoT enterprise systems where the business logic resides. This paper sets forth six sound reasons why DevOps should, and likely will, become standard practice in the development and deployment of software for gateways and edge devices as well, and outlines a technology infrastructure that can help organizations implement IoT DevOps more quickly and easily.

The integration of development and operations, dubbed "DevOps," is a hot topic in IT circles—so hot, in fact, that a standard definition, let alone formalized DevOps structures and best practices, is yet to emerge . One source characterizes DevOps as a "culture, movement, or practice" emphasizing collaboration between developers and IT operations teams with the goal of creating an "environment where building, testing, and releasing software can happen rapidly, frequently, and more reliably."

Note that this definition refers to "culture" rather than organization. While it's true that DevOps may ultimately require an organizational change, it first requires a cultural change to break down silos that separate those who build software and systems from those who implement and operate them. (The evolution of DevOps may be likened to that of agile development, which began as a movement with

the “Agile Manifesto” and is still thought of more as a set of principles than a process).

For developers of embedded systems, the concept of DevOps may at first seem foreign. Historically, the development team built the software (or a device) and handed it off to another team for release and support. But in IoT that model is a recipe for something far short of success, if not failure.

The reliable performance of an IoT solution requires a constant feedback loop, regular monitoring, speedy issue resolution, and frequent upgrading. Internet connectivity creates the opportunity for constant infusion of innovation into the system, without waiting for the next “big bang” release. It’s a process of continuous learning that necessarily requires developers and operators to collaborate closely every day.

Agile development is the forerunner to DevOps. In agile development, designers, testers, developers, and integrators merge into cross-functional teams that have end-to-end responsibility for specific functions or subsystems (see Figure 23.6).

That responsibility includes delivery, which ideally can be automatic once the software passes internal testing and quality assurance. Automation of delivery makes possible the concept of continuous deployment, which increasingly goes hand in hand with agile methodologies.

If your organization practices agile development and continuous deployment, you are on the evolutionary path to DevOps. You may even be practicing DevOps in an ad hoc fashion without realizing it. The next step is to formalize a DevOps organizational process and structure, supported by technology that integrates the building, testing, deployment, and management of IoT applications on a single platform with a high level of automation. But before we discuss how you do it, let’s be clear about why you should.

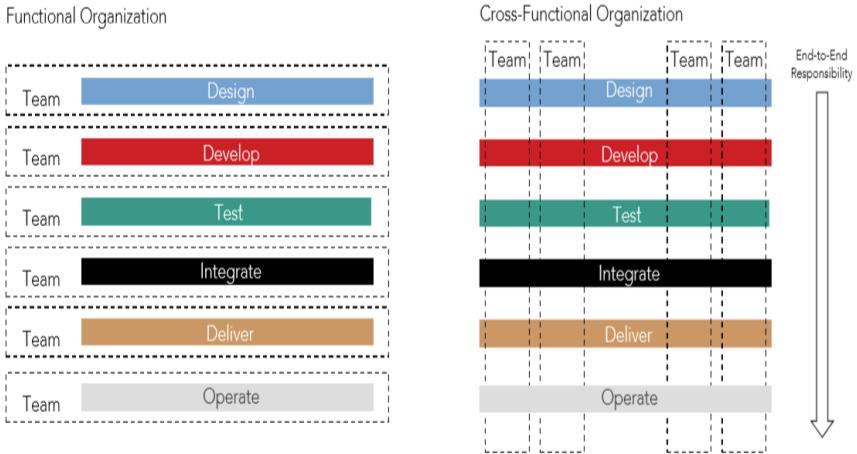


Figure 23.6 - Functional vs. cross-functional organization

### 23.4.2 Reasons DevOps matter in IoT

*Six reasons DevOps matters in IoT.* DevOps has already gained a foothold at the enterprise level. Witness the social media networks or video or music streaming services that are constantly pushing new content to users. Or consider the mobile device makers and application providers that make new releases of operating systems and other apps available for download on a regular basis. Now, the next frontier for DevOps is IoT, specifically the edge devices (the “things” in the Internet of Things) that perform the IoT system operations and feed data back to the enterprise. Here’s why:

- The evolution effect: DevOps is simply bound to happen, in some shape of form, as the next step in the evolution from agile development to continuous deployment. Organizations that embrace DevOps and formalize it through integrated, cross functional teams are likely to have a decided advantage over those that do not.

- The spreading effect: If software on enterprise servers is being updated regularly or continuously, the systems that are connected to those servers and dependent on that software will likely require frequent updating as well. At some point, teams that develop the enterprise software will expect faster release cycles in the other parts of connected systems.

- The infrastructure in place: With systems now connected via the Internet and the cloud, it's possible to automatically deploy and regularly upgrade software in multiple field devices remotely.

- Software-defined “anything”:

Increasingly, it is the software deployed on a device (regardless of the hardware) that differentiates it and defines its functionality. That means that when functionality needs to be updated, it is more often going to entail a software update rather than an electrical or mechanical modification.

- New business models and revenue streams: The big promise of IoT is that it makes possible new business models and sources of revenue that couldn't otherwise exist. The ability to constantly deliver new software updates makes it possible to sell services that generate continuous revenues, rather than simply the one-time sale of the underlying product.

- Greater productivity and cost-efficiency: A process that accelerates development cycles without compromising quality through more effective collaboration is simply a better, faster, smarter way to work—with the potential to drive down operating costs.

The case for evolving DevOps from enterprise systems to IoT edge devices is fairly compelling. IoT application and device developers are under enormous pressure to deliver quality solutions and meet tight time-to-market demands. Because IoT systems may be expected to perform for many years, development and operations teams must work together to plan for their entire lifecycle, from design through end-of-life. Companies that can meet these challenges stand to gain a significant competitive advantage—and a transition to a formalized DevOps organizational structure would seem to be the answer for adapting to this new environment.

So why isn't it happening more quickly? Let's look at some of the obstacles and challenges to DevOps implementation.

*Overcoming the obstacles.* Change is rarely easy, and instituting DevOps is no exception. Organizations are likely to encounter several obstacles in the transition.

- Cultural and organizational change is difficult: A transition to DevOps entails overcoming years of ingrained cultural perceptions and behavior. Changes in reporting relationships, responsibilities, and accountabilities are bound to be a bit rocky. An organization must recognize the need to change and then build the processes and systems

necessary to accomplish the DevOps vision.

- DevOps is inherently difficult at the device level: Unlike the enterprise software environment, where server hardware is fairly standardized, IoT systems can be very large, unique, and complex, with a wide variety of hardware platforms. Where enterprise systems have built-in redundancy, there is typically very little redundancy for software embedded in field devices. Reducing the risk of costly failure requires exhaustive production-level testing and quality assurance, which lengthens development cycles.

- Reliability is paramount: When software is continuously deployed, it has to work as promised. IoT solutions have very high demands for reliability, quality, security, and safety. Correcting problems in deployed software can be extremely cumbersome and inefficient, and the business risk is high if the manufacturer has a contractual obligation to ensure performance as expected. Quality assurance is an essential ingredient of any DevOps model.

There are few tools that actually support the DevOps paradigm, which calls for agile code sprints, automated testing, fast and automated feedback loops, and collaborative teams with a high degree of autonomy and communication. What's needed is a clear path between development platforms and field systems, so that DevOps teams can monitor system health, detect potential issues, and act on them before they become problems.

*How technology can enable DevOps.* DevOps in IoT is not inevitable. It requires commitment, collaboration, communication, and a willingness to change. It can, however, be made easier with technology that integrates system development, testing and debugging, deployment, monitoring, and management on a single platform. Unlike conventional development tools designed to support functional organizations working in horizontal layers, an integrated approach would enable true cross-functional collaboration in a vertical model (refer to Figure 23.6).

It would allow system developers and those responsible for operationalizing the software to work as a coordinated team in a centralized, cloud-based environment, thereby accelerating the delivery of applications with full quality assurance and enabling effective troubleshooting of systems in the field.

It's important to think of DevOps not simply as the merger of

development and operations, but as the intersection of development, quality assurance (QA), and operations (see Figure 23.7) - QA being the essential step that ensures a system will work properly before going into operation.

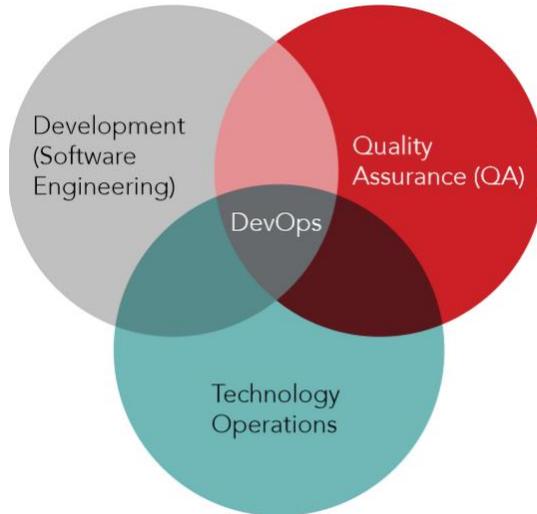


Figure 23.7 - DevOps as the intersection of development, quality assurance, and operations

Helix Cloud maps to this three-stage DevOps model through the integration of three core components:

- Wind River Helix App Cloud serves the development side of DevOps. It equips application developers with ready-to-use tools and software development kits (SDKs) for any hardware variant. App Cloud makes it possible to easily build applications independent of device operating system and hardware complexity.

By providing developers with the appropriate tools and target systems, App Cloud helps mitigate integration issues between application and platform software and cut down on team handovers. As a cloud platform, it also allows anytime, anywhere access to tools and enables large, geographically dispersed development teams to collaborate across borders and time zones.

- Wind River Helix Lab Cloud addresses the testing and quality assurance aspect of DevOps. It allows instant access to virtual hardware of whole systems at representative scale, enabling teams to use full-

system simulation for testing and QA of complex and large-scale IoT systems. A simple login provides any engineer with access to the cloud-based virtual lab. Using on-demand simulation software as a complement to hardware, teams can automate testing in entirely new ways and create any number of virtual target systems for parallel testing.

This significantly shortens the cycle between application development and system testing. With Lab Cloud, teams can stage and test systems at representative scale in a pre-production environment so they can move into production with confidence in system continuity.

- Wind River Helix Device Cloud is the platform for managing and operating devices from the cloud—the “bridge” between development and operations. It enables operators to safely and securely update, monitor, service, and manage devices in the field. Device Cloud automatically collects and integrates data from hundreds or thousands of disparate devices, machines, and systems, enabling operators to track device status and content, share data among engineers, and proactively determine when updates are needed.

Collectively, the Helix Cloud suite enables organizations to transition into a DevOps team structure. Specifically, it:

- Facilitates a new paradigm of “always connected” collaboration that isn’t restricted by geography
- Helps break down the silos that separate those who develop software and systems from those who deploy and operate them, enabling teams to work cross-functionally in a collaborative spirit and making the release of software fast, reliable, and automated
- Allows testing of software at scale before deployment as well as proactive management of field devices, which together enable continuous quality assurance

So, connected IoT edge devices have a lifecycle beyond “deploy, break and fix, and retire.” Connectivity creates the opportunity to continuously infuse incremental innovation across the system lifecycle. DevOps puts organizations in the best position to capitalize on that opportunity.

As the lines between software creation and operation begin to disappear, so too must the organizational lines that separate system developers from system operators. The DevOps concept has been proven at the enterprise level and is evolving toward gateways and edge

devices as a means to meet the unique challenges of developing and managing IoT systems. While there are obstacles to overcome, both cultural and practical, a technology platform that integrates system development, testing and QA, deployment, and management can provide the necessary infrastructure for implementing DevOps, empowering cross-functional teams to accelerate system development, ensure system quality, and optimize system reliability in the field.

### **23.5 Work related analysis**

The section is based on analysis of publications and materials of leading companies in DevOps methodology, SDN and IoT.

A few USA and EU universities including ALIOT project partners conduct research and implement education MSc and PhD modules related to DevOps and connection of this methodology with SDN and IoT. In particular, the following courses and programs have been considered:

- Washington University in St. Louis [12];
- Coimbra University, Portugal: IoT course for MSc [22]. The courses represents a new stage in the digital evolution and focuses on the Internet of Things for smart transport and cities, and the development of tools to transform city infrastructure;
- KTH University, Sweden: three MSc programs including IoT related topics in Information and Network Engineering [23] and Communication Systems [24];
- Newcastle University, United Kingdom: MSc Programme on Embedded Systems and Internet of Things (ES-IoT) MSc [25].

### ***Conclusion and questions***

The improvements assured by implementation of agile models of software and systems development are moving downstream toward IT operations with the evolution of DevOps methodology. In order to meet the demands of an agile business, IT operations need to deploy applications in a consistent, repeatable, and reliable manner. This can only be fully achieved with the adoption of automation.

Widespread platforms, like AWS, MS Azure, Google Cloud, etc. have been analysed. These solutions support numerous DevOps principles and practices that IT departments can capitalize on to improve business agility.

This section has been dedicated to analysis of DevOps principles and practices supported on the well-known platforms, like the following:

- AWS,
- MS Azure,
- Google Cloud, etc.

A brief introduction to the origins of DevOps sets the scene and explains how and why DevOps has evolved. Interconnection of DevOps, Software Defined Networks (SDN) and IoT has been analysed.

In order to better understand and assimilate the educational material that is presented in this section, we invite you to answer the following questions.

1. Describe basic concepts of DevOps. Which main features of this methodology are?
2. Describe principles of DevOps methodology. Which are main elements of this methodology?
3. Describe Agile evolution to DevOps. What does CI/CD mean?
4. Which are features of Blue–Green Deployment (BGD) as a practice of DevOps?
5. Describe a five-step approach to creating a DevOps pipeline. Which are features of the steps:
  - CI/CD framework?
  - Source control management?
  - Build automation tool?
  - Web application server?
  - Code testing coverage?
6. Which are features of optional steps such as containers, middleware automation tools?
7. Which are DevSecOps features and purposes?
8. Which are DevSecOps approaches: “Automate {in origin “kill”} them all”? Checking the vulnerability of the generated code and others?
9. How are SDN and DevOps connected?
10. Describe leading practices for SDN and DevOps for:

- service fulfillment,
  - operations support and readiness,
  - service assurance.
11. Which are reasons DevOps matter in IoT?
  12. How do DevOps application influence on characteristics of developed IoT systems

### ***References***

1. Introduction to DevOps on AWS, [https://d0.awsstatic.com/whitepapers/AWS\\_DevOps.pdf](https://d0.awsstatic.com/whitepapers/AWS_DevOps.pdf)
2. A beginner's guide to building DevOps pipelines with open source tools, <https://opensource.com/article/19/4/devops-pipeline>
3. DevOps - Technology and Tools overview, [https://www.gecko.rs/sites/default/files/pdf/Gecko\\_Solutions\\_DevOps\\_Technology\\_Overview.pdf](https://www.gecko.rs/sites/default/files/pdf/Gecko_Solutions_DevOps_Technology_Overview.pdf)
4. Devops in the internet of things. Six reasons it matters and how to get there, <https://events.windriver.com/wrcd01/wrcm/2016/08/WP-devops-in-the-internet-of-things.pdf>
5. Practicing Continuous Integration and Continuous Delivery on AWS, <https://d1.awsstatic.com/whitepapers/DevOps/practicing-continuous-integration-continuous-delivery-on-AWS.pdf>
6. DevOps for IoT Applications using Cellular Networks and Cloud Athanasios Karapantelakis, Hongxin Liang, Keven Wang, Konstantinos Vandikas, Rafia Inam, Elena Fersman, Ignacio Mulas-Viela, Nicolas Seyvet, Vasileios Giannokostas, <https://www.ericsson.com/assets/local/publications/conference-papers/devops.pdf>
7. Vlasov, Y., Illiashenko, O., Uzun, D., Haimanov, O. Prototyping tools for IoT systems based on virtualization techniques (Conference Paper). Proceedings of 2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies, DESSERT 2018, 9 July 2018, P. 87-92
8. M. H. Syed, E. B. Fernández. Cloud Ecosystems Support for Internet of Things and DevOps Using Patterns, Conference: 2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI), DOI: 10.1109/IoTDI.2015.31

9. AWS IoT Plant Watering Sample, <https://docs.aws.amazon.com/iot/latest/developerguide/iot-plant-watering.html>

10. The DevOps Handbook: An Introduction Summary, <https://caylent.com/devops-handbook-introduction-summary/>

11. The Definitive Guide to Scrum: The Rules of the Game, <https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf#zoom=100>

12. Cloud Tutorial: AWS IoT, <https://www.cse.wustl.edu/~lu/cse521s/Slides/aws-iot.pdf>

13. Allan K. (2018), “Automated Security Testing Best Practices” <https://phoenixnap.com/blog/devsecopsbest-practices-automated-security-testing>

14. Litz S. (2015) “What is DevSecOps” <http://www.devsecops.org/blog/2015/2/15/what-is-devsecops>

15. Savant S(2018) “What is the difference between DevOps and DevSecOps”, <https://www.quora.com/What-is-the-difference-between-DevOps-andDevSecOps>

16. GitLab, (2018) “Static Application Security Testing (SAST)”, [https://docs.gitlab.com/ee/user/project/merge\\_requests/sast.html](https://docs.gitlab.com/ee/user/project/merge_requests/sast.html)

17. S. Harris, “Physical and Environmental Security. In CISSP Exam Guide”, USA McGraw-Hill, 6th ed., pp.427-502 2013.

18. Network Transformation with NFV and SDN, <https://www.juniper.net/assets/us/en/local/pdf/whitepapers/2000628-en.pdf>

19. Operationalizing SDN and NFV Networks, <https://www2.deloitte.com/content/dam/Deloitte/us/Documents/technology-media-telecommunications/us-tmt-operations-sdn-and-nfv-networks.pdf>

20. Tempus SEREIN project official website <http://serein.eu.org/>

21. Erasmus+ ALIOT project official website <http://aliot.eu.org/>

22. Internet Of Things Course - Immersive Programme Master in City and Technology [<https://apps.uc.pt/search?q=Internet+of+Things>]

23. MSc Programme in Information and Network Engineering [<https://www.kth.se/en/studies/master/information-and-network-engineering/master-s-programme-in-information-and-network-engineering-1.673817>]

24. MSc Programme in Communication Systems [<https://www.kth.se/en/studies/master/communication-systems/description-1.25691>]

25. MSc Programmes to Embedded Systems and Internet of Things  
[<https://www.ncl.ac.uk/postgraduate/courses/degrees/embedded-systems-internet-of-things-msc/relateddegrees.html>]

**PART VII. DEPENDABILITY AND SECURITY OF IOT**  
**24. DEPENDABILITY AND SECURITY MODELS OF IOT**

DrS. Prof. V. V. Sklyar, DrS. Prof. V. S. Kharchenko (KhAI)

*Contents*

|                                                                                   |     |
|-----------------------------------------------------------------------------------|-----|
| Abbreviations .....                                                               | 284 |
| 24.1. Dependability and security concepts for IoT.....                            | 285 |
| 24.1.1. Taxonomy of safety and security requirements.....                         | 285 |
| 24.1.2. Dependability, safety and security attributes taxonomy .....              | 287 |
| 24.1.3 Risk analysis fundamentals .....                                           | 289 |
| 24.2 Dependability and safety models for IoT .....                                | 290 |
| 24.2.1 Reference architectures of IoT.....                                        | 290 |
| 24.2.2 Redundancy and self-diagnostics implementation in IoT systems .....        | 292 |
| 24.2.3 Dependability and safety indicators.....                                   | 296 |
| 24.2.4 Failure Mode, Effect and Criticality Analysis (FMECA) of IoT systems ..... | 300 |
| 24.3 Security models for IoT.....                                                 | 302 |
| 24.3.1 IoT systems treats .....                                                   | 302 |
| 24.3.2 Security measures .....                                                    | 307 |
| 24.3.3 Threat and attacks modeling for IoT systems.....                           | 310 |
| 24.4 Work related analysis .....                                                  | 312 |
| Conclusions and questions.....                                                    | 314 |
| References .....                                                                  | 315 |

### ***Abbreviations***

C&C – Command and Control

EUC – Equipment Under Control

FMECA – Failure Mode, Effect and Criticality Analysis

ICS – Industrial Control System

IEC – International Electrotechnical Commission

IEEE – Institute of Electrical and Electronics Engineers

IIoT – Industrial IoT

ISA – International Society of Automation

ISMS – Information Security Management System

RAMS – Reliability, Availability, Maintainability, Safety

RBD – Reliability Block Diagrams

SIL – Safety Integrity Level

## **24.1. Dependability and security concepts for IoT**

### ***24.1.1. Taxonomy of safety and security requirements***

Taxonomy of safety and security requirements is based on analysis of relevant standard in this area, such as IEC 61508 “Functional safety of electrical/electronic/programmable electronic safety-related systems” and ISA/IEC 62443 “Security for Industrial Automation and Control Systems”. These functional safety requirements can be divided in some following categories [1]:

- Requirements to functional safety management;
- Requirements to functional safety life cycle;
- Requirements to systematic (system and software design) failures avoidance;
- Requirements to random (hardware) failures avoidance.

A scope of the above requirements is highly dependent from as named Safety Integrity Level (SIL) [2] which establishes relation between IoT system risk level and a scope of the related safety assurance countermeasures. The discussed approach can be represented in a view of a diagram (see Fig. 24.1).

The above approach can be applied for IoT security concept. Firstly, Security Levels shall be implemented for IoT system taken into account risks levels (see Section 24.1.3). Secondly, Information Security Management System (ISMS) shall be implemented and coordinated with functional safety management issues. Thirdly, a common security and safety life cycle shall be established to cover all the process of IoT system development, verification and validation. Fourthly, common safety and security risks shall be avoided to implement coordinated countermeasures against random (hardware) and systematic (system and software design) failures. Examples of common safety and security random failures avoidance countermeasure include redundancy, self-diagnostic, electromagnetic disturbances protection and others (see Fig. 24.2) [3].

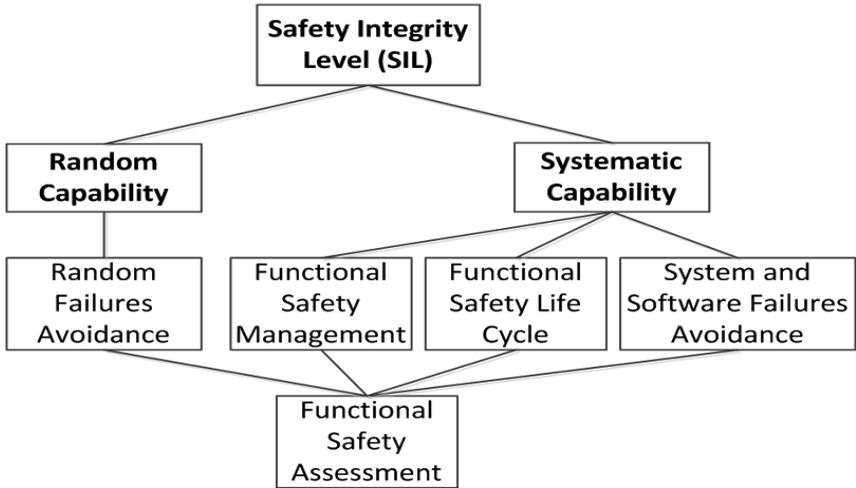


Fig. 24.1 – A concept of IoT safety requirements

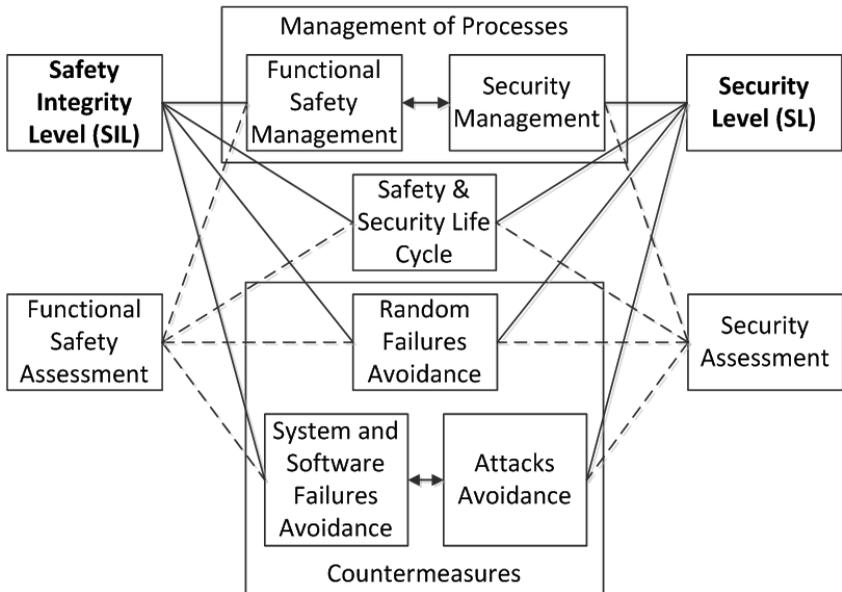


Fig. 24.2 – A concept of IoT harmonized security and safety requirements

Examples of common safety and security systematic failures avoidance (attacks avoidance for security) are access control and configuration control. Fifty, assessment shall be periodically performed for both, security and safety. The discussed approach is the base for security and safety coordination, as it is represented on Fig. 24.2.

### ***24.1.2. Dependability, safety and security attributes taxonomy***

Four of the attributes RAMS (Reliability, Availability, Maintainability, Safety) used to be considered as extensions for “classical” Reliability. The paper “Basic Concepts and Taxonomy of Dependable and Secure Computing” [4] launched in 2004 the new IEEE Transactions on Dependable and Secure Computing. It explains the complexity of dependability in relation with security of modern computer-based systems (see Fig. 24.3).

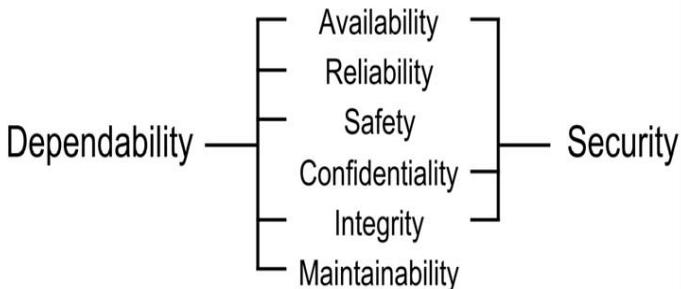


Fig. 24.3 – Dependability and security attributes

In the [2], dependability is considered as an integrating concept including the following attributes:

- Availability is a readiness for correct service;
- Reliability is a continuity of correct service;
- Safety is an absence of catastrophic consequences for the user and the environment;
- Integrity is an absence of improper system alterations;
- Maintainability is an ability to undergo modifications and repairs.

Security is a composite of the attributes availability, integrity, and confidentiality. When addressing security, availability is considered for authorized actions as well as integrity is considered for a proper authorization. Confidentiality is a supplementary, in comparison with dependability, security attribute, which means the absence of unauthorized disclosure of information.

Standardized definition of dependability is the following: “dependability is the property to keep within the established values of the parameters under all the stated conditions within a stated period of time.” The above definition supposes a taxonomy which contains the following four attributes of dependability:

- Reliability is continuity of the operation state during some time;
- Durability is continuity of operation with periodic maintenance and repairs until retirement time; it is highly related with long term operation;
- Maintainability is an ability to support operation state and to turn back to operation state after periodic maintenance and repairs;
- Storability is an ability to support all dependability attributes during storage.

To harmonize two dependability taxonomies (RAMS with the standardized taxonomy) and security attributes let’s consider the following statements [1]:

- Availability is a combination of Reliability and Maintainability what is from equation  $A = \text{MTTF} / (\text{MTTF} + \text{MTTR})$ , where MTTF – Mean Time to Failure, MTTR – Mean Time to Restoration;
- Accessibility is more appropriate term for safety domain the Availability. However Accessibility is a part of Availability, so such relation is established;
- Safety takes a care mostly about the failures of Safety Functions (dangerous failure), which are intended to achieve or maintain a safe state of a system. So there is a relation between Reliability and Safety, and this relation is established via Safety Functions;
- At the same, Safety includes both Safety Functions and Integrity, what is stated in the standards IEC 61508 as the confidence level (sometimes, probability) of a system satisfactorily performing the specified safety functions under all the stated conditions within a stated period of time.

The considered approach to analysis of dependability, safety and security attributes allows representing all attributes in a view of one diagram (see Fig. 24.4).

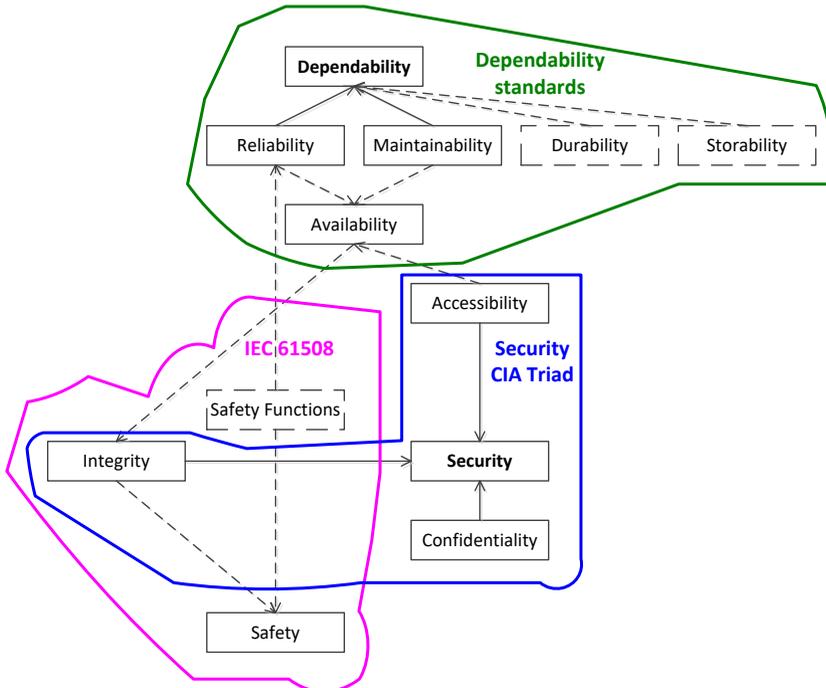


Fig. 24.4 – Integrated taxonomy of dependability, safety and security attributes

### 24.1.3 Risk analysis fundamentals

Risk is a basic concept and indicator of safety or security, which is a combination of the probability of an undesirable event and its consequences [2]:

$R(t) = P(t) \cdot C$ , where  $P(t)$  is the probability of an undesirable event,  $C$  is the potential damage.

Risk assessment can be quantitative and qualitative, where a qualitative one operating with such categories as “high”, “medium”, “low”, etc.

If an undesirable event and damage from it are stated, then the risk is numerically equal to the probability  $P(t)$  of the occurrence of fixed damage. For example, the risk of a nuclear power plant accident with the release of radioactive products into the atmosphere today is not more than  $10^{-7}$  1/year.

In information security a quantitative assessment can be made as a boundary value for single loss expectancy (SLE):  $SLE = AV \cdot EF$ , where AV is asset value, EF is exposure factor which expresses a percentage of damage to asset value because any of threat. To get annual loss of expectancy (ALE) it is needed to take into account annual risk concurrency (ARO):  $ALE = SLE \cdot ARO$ . Investment to security protective measures during one year cannot be more than ALE value.

The ALARA / ALARP principle (as low as reasonably applicable / practicable) is widely used for risk assessment and management. This approach implies risk reduction as much as possible to achieve due to actually available limited resources.

## **24.2 Dependability and safety models for IoT**

### ***24.2.1 Reference architectures of IoT***

Requirements for IoT components have been identified by different vendors, system integrators, consortia etc. IoT Reference Architecture is a subject of standardization, what is developing now by International Electrotechnical Commission (IEC) and Institute of Electrical and Electronics Engineers (IEEE). The IoT Reference Architecture should describe the system or systems for the IoT, the major components involved the relationships between them, and their externally visible properties (see Fig. 24.5).

To specify a possible structure of IoT system let's consider a medical application (Fig. 24.6) which includes Blood Pressure Devices (device layer), Local Network Router & Medical Service Gateway (network layer), Cloud Data Center (service layer), and User Application (application layer).

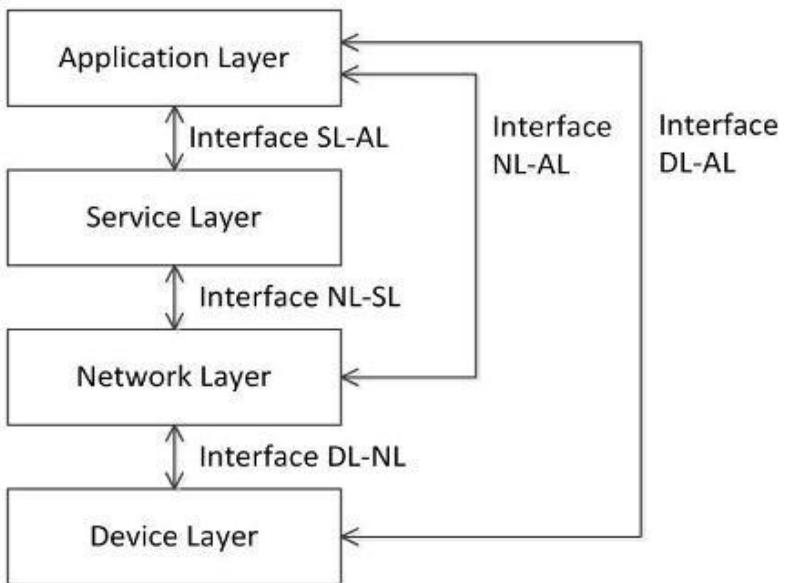


Fig. 24.5 – IoT Reference Architecture

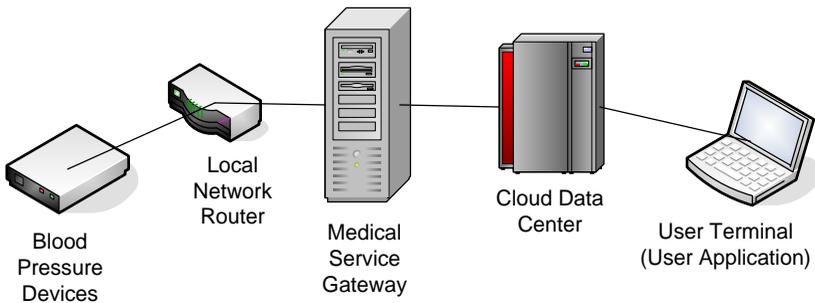


Fig. 24.6 – An example of medical IoT system

Device layer is represented by sensor networks which are connected with mini-computers or controllers. Device Layer has a typical structure of the Industrial Control System (ICS), what is

reflected in architecture of Industrial IoT (IIoT) or Internet ICS (IICS) as well as in Industry 4.0 concept (see Fig. 24.7). Such architecture is a result of hybridization of ICS with IoT.

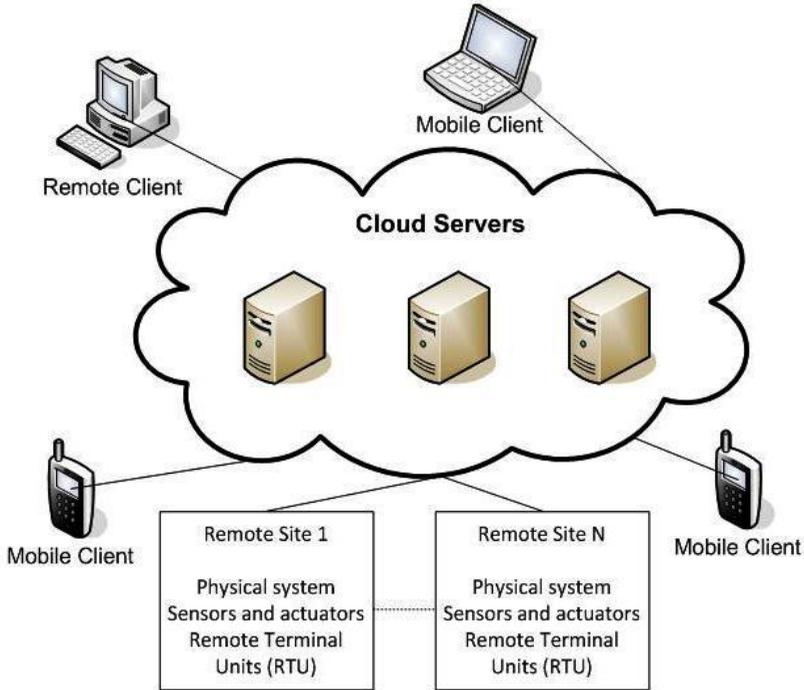


Fig. 24.7 – An example of architecture of Industrial IoT system

### ***24.2.2 Redundancy and self-diagnostics implementation in IoT systems***

To provide redundancy of IoT systems let's investigate a system represented on Fig. 24.6. For this system we added a redundant channel as well as redundant cross-channel communication which allow recovering system in a case of single failures. Cloud Data Centers need to have communication link to synchronize stored data in accordance with implemented time intervals (see Fig. 24.8). Redundancy can be implemented for some components as well for a system. Since redundancy for some components does not provide a big value for a

system (for example redundant Blood Pressure Devices for a single patient), a criteria “Reliability / Cost” has to be used for system efficiency analysis.

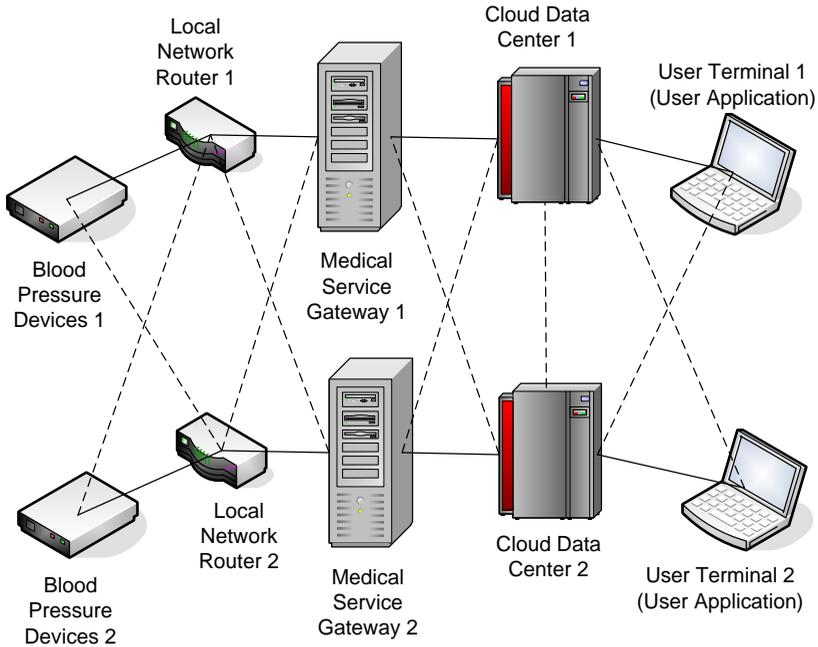


Fig. 24.8 – Redundant medical IoT system

Redundant IIoT systems have to implement redundancy for both ICS (RTU) and IoT parts. For this approach let's investigate a system represented on Fig. 24.7. Fig. 24.9 represents a single connected to IIoT site, and such sites can be multiple. Also redundancy can be implemented for components as well as for a system. Fig. 24.9 represents inter-channel communication links, but channel can be separated to implement independency.

The typical ICS includes [1,2]:

- power supply components;
- field equipment (sensors and actuators);

- programmable logic controllers, including input and output modules and control modules;
- network equipment, servers, and human-machine interface components.

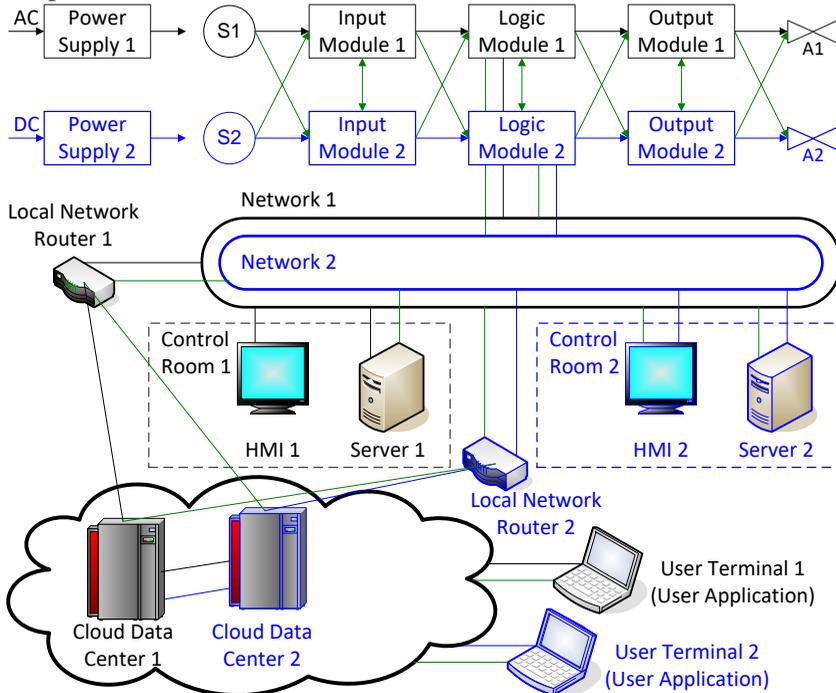


Fig. 24.9 – Redundant Industrial IoT system

Ideally, maximum independence is ensured by power supplying of independent channels of the system from independent inputs. The diagram shows that the first channel is powered by an alternating current, and the second channel is powered by direct current. Then, in case of problems with power supply in one of the power supply systems, only one of the channels will be de-energized. Ensuring continuity and quality of power supply even in extreme conditions is a vital aspect of ensuring the safety and security of control systems.

Redundant sensors, controllers and actuators may be used. Protocols of information exchange can be organized between channels, or maximum independence between channels can be realized, and then there will be no exchange.

In addition, a redundant network architecture and a duplicate man-machine interface with redundant computing components and data storages can be implemented.

Redundant architectures with “2-out-of-3” and “2-out-of-4” voting logic are also used In ICS important for safety and security.

Self-diagnostics of digital devices can be described as on Fig. 24.10. Along with the main algorithms of digital control, in parallel, the system implements the processing of diagnostic data and watchdog functions. All these three processes are performed independently of each other, and independent clock sources, different chips, etc. can be used [1,3].

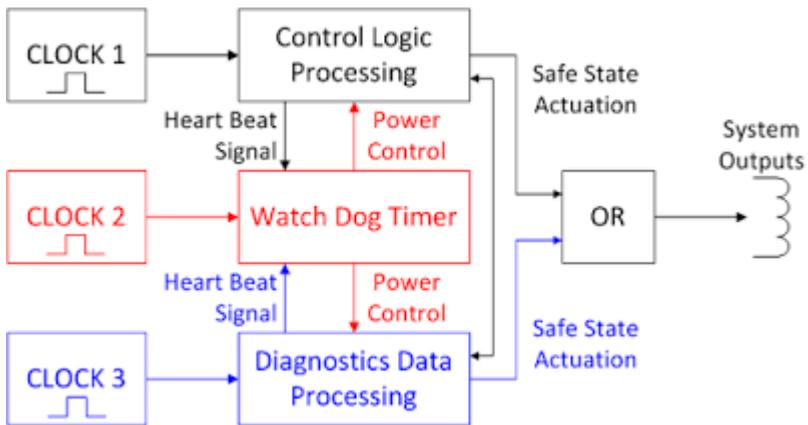


Fig. 24.10 – Self-diagnostics of IoT

Watchdog monitors the simplest response (heart bit) from the chips that perform data processing, and when a problem is detected (the response is stopped), it turns off the power and puts the system into a safe state. In addition, the watchdog timer can monitor the power level and produce a similar shutdown command if there is a dangerous power deviation from the specified level. Safe state for safety systems, as a rule, consists in removing power from the output analogue and discrete

outputs. If necessary, the safety system can supply power to the actuators, but then the output requires additional signal converters.

If self-diagnostics detects a critical failure (for example, hardware failure, hardware or software configuration violation, data transmission violation, etc.), a command is issued to put the system into a safe state, which is performed as if the command came from main control logic.

Now we generalize typical functions of digital devices self-diagnostics. The functions of the watch dogs and power control have been already considered. An important diagnostic function is to control the configuration of software and hardware. This property also affects information security. During operation, each hardware module periodically transmits information about its serial number and configuration of the loaded software (for example, check-sum). If a configuration failure occurs, then the system performs the specified protective actions, up to a transition to a safe state and power off.

Another option to perform hang-up monitoring is internal or external timers that check the execution time of control logic loops. Some procedures can be restarted several times, and in case of several unsuccessful restarts, a decision to transit to a safe state can be made.

An important function of control systems is to ensure the accuracy of measurement of input and output analogue signals. To diagnose measurement accuracy, redundant analogue-digital converters (ADCs) and digital-analogue converters (DACs) can be used, in which the processing results are compared and a diagnostic message is generated on the coincidence or discrepancy of the results.

Much attention in the control systems is paid to the transfer of data packages, both through communication channels and in the processing distributed between the software and hardware components. Here, methods such as transmission confirmation, timeout control, integrity monitoring and data packages transmission sequence, cyclic redundancy codes (CRC) are used for self-diagnostics. To protect information during data transmission encryption algorithms can be used.

### ***24.2.3 Dependability and safety indicators***

The basic concept of functional safety assessment is dividing a common failure rate  $\Lambda$  (let us begin with the exponential distribution with a constant failure rate) into dangerous and safe failures as well as

into detected and undetected failures. This is a main difference of functional safety from reliability. From this point of view we have four failures sets (see Fig. 24.11):

- Safe Detected failures with a failure rate  $\lambda_{Sd}$  – failures which put the equipment under control (EUC) to a safe state and are discovered by self-diagnostics;

- Safe Undetected failures with a failure rate  $\lambda_{Su}$  – failures which put the EUC to the a state and are not discovered by self-diagnostics;

- Dangerous Detected failures with a failure rate  $\lambda_{Dd}$  – failures which put the EUC to a potentially dangerous state and are discovered by self-diagnostics;

- Dangerous Undetected failures with a failure rate  $\lambda_{Du}$  – failures which put the EUC to a potentially dangerous state and are not discovered by self-diagnostics.

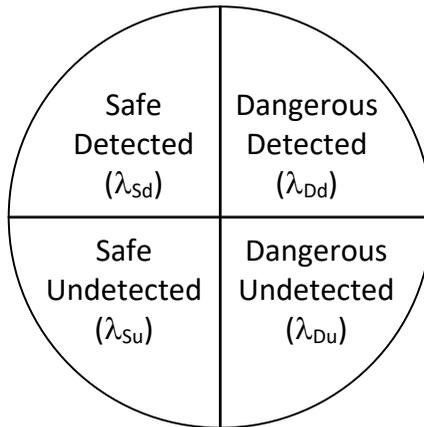


Fig. 24.11 – Failures theoretical-set model

So, there are some obvious dependencies following from Fig. 24.11:

- Common failure rate is  $\Lambda = \lambda_{Sd} + \lambda_{Su} + \lambda_{Dd} + \lambda_{Du}$ ;
- Dangerous failure rate is  $\lambda_D = \lambda_{Dd} + \lambda_{Du}$ ;
- Safe failure rate is  $\lambda_S = \lambda_{Sd} + \lambda_{Su}$ ;
- Detected failure rate is  $\lambda_d = \lambda_{Sd} + \lambda_{Dd}$ ;
- Undetected failure rate is  $\lambda_u = \lambda_{Su} + \lambda_{Du}$ .

Also a lot of relative metrics can be extracted from dependencies between sets cardinality and different failure rates values. The most important from these metrics are the following:

– Safe Failure Fraction (SFF) in accordance with IEC 61508 is  $SFF = (\lambda S + \lambda Dd) / \Lambda$ ;

– Dangerous Failure Fraction (DFF) in accordance with IEC 61508 is  $DFF = 1 - SFF = \lambda Du / \Lambda$ ;

– Diagnostic Coverage (DC) for dangerous failures in accordance with IEC 61508 is  $DC_D = \lambda Dd / \lambda D$ ;

– More widely used equation for Diagnostic Coverage is  $DC = \lambda D / \Lambda$ ;

– Proof Test Coverage (PTC) should be calculated from the total failure rates for the using the formula  $PTC = 1 - \lambda Du_{aPT} / \lambda Du$ , where  $\lambda Du_{aPT}$  is  $\lambda Du$  after Proof Test.

In addition, IEC 61508 requires the following indicators to be determined for the system components:

– Lifetime – the time during which the element performs its functions without breaking the properties;

– Periodic proof test interval – the time between conducting periodic tests, which cover such components that cannot be diagnosed during operation; thus, dangerous undetected failures are identified;

– Diagnostic test interval – the time between conducting tests in the process of operation;

– Mean Repair Time (MRT), which may be equivalent to Mean Time to Restoration after failure (MTTR).

To move ahead with safety indicators we need to introduce some definitions from the standards series IEC 61508.

– Safety Function is a function to be implemented by a safety-related system or other risk reduction measures, that is intended to achieve or maintain a safe state for the EUC, in respect of a specific hazardous event; all the above indicators are usually calculated for specified Safety Functions; sometimes for ICS a term Safety Instrumented Function (SIF) is used as equal;

– Safety Integrity is a probability of a safety-related system satisfactorily performing the specified safety functions under all the stated conditions within a stated period of time;

– Safety Integrity Level (SIL) is a discrete level (one out of a possible four), corresponding to a range of safety integrity values, where SIL 4 has the highest level of safety integrity and SIL 1 has the lowest;

– Mode of Operation is a way in which a safety function operates, which may be either

- Low Demand Mode: where the safety function is only performed on demand, in order to transfer the EUC into a specified safe state, and where the frequency of demands is no greater than one per year; or

- High Demand Mode: where the safety function is only performed on demand, in order to transfer the EUC into a specified safe state, and where the frequency of demands is greater than one per year; or

- Continuous Mode: where the safety function retains the EUC in a safe state as part of normal operation.

IEC 61508 states different Safety Indicators depending from the Mode of Operation.

For Low Demand Mode average probability of dangerous failure on demand (PFDavg) shall be calculated. PFDavg is mean unavailability of a safety-related system to perform the specified safety function when a demand occurs from the EUC.

The IEC 61508 states that only Dangerous Undetectable failures contribute to PFDavg, the last can be calculated as  $PFD_{avg}(Du) = 1 - A(Du) = U(Du) = \lambda Du / (\lambda Du + \mu Du)$ , where  $\mu Du$  is restoration rate of Dangerous Undetectable failures.

Also for Dangerous failures  $PFD_{avg}(D) = 1 - A(D) = U(D) = \lambda D / (\lambda D + \mu D)$ , where  $\mu D$  is restoration rate for all the Dangerous failures.

For High Demand Mode and Continuous Mode average frequency of a dangerous failure per hour (PFH) shall be calculated. PFH is the average frequency of a dangerous failure of a safety related system to perform the specified safety function over a given period of time.

Usually PFH is defined as failure rate, so on the base of Dangerous Undetectable failures  $PFH(Du) = \lambda Du$ , and on the base of all the Dangerous failures  $PFH(D) = \lambda D$ .

Also the IEC 61508 states that PFH can be calculate as unavailability or as unreliability depending from a safety-related system application conditions.

#### ***24.2.4 Failure Mode, Effect and Criticality Analysis (FMECA) of IoT systems***

FMECA differs from other methods of dependability and safety analysis in that it puts together all the tasks of calculating safety indicators. The standard IEC 60812:2006 “Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA)” has been developed is to describe this method.

At the initial stages of the FMECA application, it is recommended to apply a hierarchical decomposition of the system, for example, using Reliability Block Diagrams (RBD).

The safety and security analysis sequence using FMECA includes the following steps:

- analysis of the structure and functions of the system;
- division of the system into its parts and elements, based on the influence of element failures on system failures and the level of detail;
- drawing and analysis of RBDs for system decomposition;
- determination of types of failures and operating modes of the system;
- determination of the effects of failures and their criticality;
- determination of root causes of failures;
- determination of failures rate;
- determination of methods for detection and compensation of failures; for this self-diagnostic approach is analysed, both for hardware and software, as well as the diagnostic coverage;
- calculation and analysis of dependability and security indicators; bottom-up analysis has to be performed, i.e. elements are assembled in units, parts and the system as a whole; the obtained indicators are compared with the specified requirements.

For the above steps, different levels of details may be applied. Usually for safety systems, the analysis takes into account all electronic components, such as resistors, capacitors, diodes, etc.

FMECA is performed for the identified safety and security functions from the point of view of the software and hardware involved in the execution of the function. For these functions, the states of dangerous failures have to be defined and described. The results of the analysis are recorded in the form of FMECA tables (see Table 24.1).

Table 24.1 – A part of FMECA table for a hardware module

| Unit                  | Failure Mode          | Failure Cause      | Failure Effect                     | Failure Criticality | Failure Diagnostics           | Failure Recovery      | Failure Rate               |
|-----------------------|-----------------------|--------------------|------------------------------------|---------------------|-------------------------------|-----------------------|----------------------------|
| Power Supply Unit     | Loss of power         | Short break        | Loss of power of the module        | Dangerous           | Online check of voltage value | Safe state transition | $7 \cdot 10^{-8}$ 1/hour   |
| Clock Frequency Unit  | Loss of clocks pulses | Short break        | Power off of Micro-controller Unit | Dangerous           | Watch Dog                     | Safe state transition | $2 \cdot 10^{-8}$ 1/hour   |
| Micro-controller Unit | No contact            | Fault of soldering | Power off of Micro-controller Unit | Dangerous           | Watch Dog                     | Safe state transition | $1,5 \cdot 10^{-8}$ 1/hour |
| Micro-controller Unit | RAM error             | Fault of a chip    | Not trusted Micro-controller Unit  | Dangerous           | RAM test                      | Safe state transition | $2,5 \cdot 10^{-8}$ 1/hour |

## 24.3 Security models for IoT

### 24.3.1 IoT systems treats

An actual landscape of IoT threats is represented in the [5]. IoT threats are grouped in 8 categories and are briefly discussed in Table 24.2.

Table 24.2 – IoT systems treats

| Threat                              | Threat description                                                                                                                                                                                                                                                                                                                                     |
|-------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Nefarious activity / Abuse</b>   |                                                                                                                                                                                                                                                                                                                                                        |
| Denial of Service                   | This attack can be bi-directional. It can target an IoT system resulting in system unavailability and production disruption caused by a massive number of requests sent to the system. On the other hand, an attacker may take advantage of a large number of IoT devices and create an army of IoT botnets as a platform to attack some other system. |
| Malware                             | The penetration of malicious software in an IoT aimed at performing unwanted and unauthorised actions, which may cause damage to an OT system, operational processes and related data. Ransomware, viruses, Trojan horses and spyware are common examples of this threat.                                                                              |
| Manipulation of hardware & software | Threat of unauthorized manipulation of devices software or applications within an OT system by an attacker. In terms of IoT systems, an attacker's actions may include manipulation of an industrial robot, manipulation of remote controller devices suppressing state of a control device and modification of its configuration.                     |
| Manipulation of informa-            | The threat of unwanted and unauthorized data modification by an attacker. This may apply to compromising OT or production supporting systems,                                                                                                                                                                                                          |

| Threat                                          | Threat description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| tion                                            | and manipulation of process data. Possible consequences may include inappropriate decisions based on falsified data.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Targeted attacks                                | The threat of a cyberattack targeting a specific organisation (or a specific person in this organisation). Such attack aims at harming an organisation possibly to take control over the system using various technical means such as compromising key devices and falsifying telemetry deceiving unaware operators. Other impacts include damage of reputation or theft of company secrets. This attack is different from wider scale attacks whose objective is to infect any company that connects to a certain website prepared by an attacker or any company that uses a device or software with a certain vulnerability. |
| Abuse of personal data                          | The threat of compromising personal / sensitive information stored on devices or in the cloud. The attacker's goal is to gain unauthorised access to this kind of data and use it in an illicit manner.                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Brute force                                     | The threat of gaining unauthorised access to an organisation's resources (i.e. data, systems, devices, etc.) through a large number of attempts to guess the correct key or password. IoT systems that allow the utilisation of uncomplicated or default passwords for devices and systems may be especially vulnerable to such attacks.                                                                                                                                                                                                                                                                                       |
| <b>Eavesdropping / Interception / Hijacking</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Man-in-the-Middle attack / Session              | The threat of active eavesdropping, where messages exchanged between unaware affected parties are relayed by an attacker. The attacker may just listen to the exchanged messages or modify or delete transmitted information, leading to communication                                                                                                                                                                                                                                                                                                                                                                         |

| Threat                               | Threat description                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| hijacking                            | disruption.                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| IoT communication protocol hijacking | The threat of an attacker taking control of an existing communication session between two network components, which may lead to the disclosure of passwords and other confidential information.                                                                                                                                                                                                                                                                                     |
| Network reconnaissance               | The threat of revealing internal network information (e.g. connected devices, used protocols, open ports and used services, etc.) to an attacker who manages to scan a network passively. With this knowledge, the attacker can plan which actions to take next to compromise system operation.                                                                                                                                                                                     |
| <b>Physical attack</b>               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Vandalism and theft                  | The threat of causing physical damage to the device by a saboteur who gains physical access to the operational environment – either an outsider who has managed to bypass insufficient physical security measures or an insider, e.g. a disgruntled employee who, for some reasons, wants to harm the organization. This threat also includes theft.                                                                                                                                |
| Sabotage                             | The threat of tampering with a device by a saboteur who gains physical access to the operational environment – either an outsider who manages to bypass insufficient physical security measures or an insider, e.g. a disgruntled employee who, for some reasons, wants to harm the organization. The attacker may take advantage of improper configuration of ports and possibility exploit open ports. The attacker may also use access to execute unauthorized operator actions. |

| Threat                                        | Threat description                                                                                                                                                                                                                                                                                                                                               |
|-----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Unintentional damages (accidental)</b>     |                                                                                                                                                                                                                                                                                                                                                                  |
| Unintentional change of data or configuration | The threat of disrupting an operational process by unintentional data or configuration change in the IoT system performed by an insufficiently trained employee. Even with good intentions, an unskilled employee, unaware of the consequences, may introduce improper changes to the system, especially if he or she receives higher than necessary privileges. |
| Erroneous use or administration               | The threat of disrupting an operational process or causing physical damage to the device by unintentional misuse of a device by an insufficiently trained employee. Even with good intentions, an unskilled employee                                                                                                                                             |
| Damage caused by a third party                | The threat of damaging assets caused by a third party. The third parties may have access to the OT system, for example, for maintenance or software update purposes. If this access is not controlled in a sufficient way, security breaches of a third party organisation may affect the company that receives the service.                                     |
| <b>Failures / Malfunctions</b>                |                                                                                                                                                                                                                                                                                                                                                                  |
| Failure or malfunction of a sensor / actuator | The threat of failure or malfunction of IoT end devices. This can occasionally happen, especially if proper maintenance and compliance with the devices' manuals and instructions during the exploitation is not ensured.                                                                                                                                        |
| Failure or malfunction of a                   | The threat of failure or malfunction of control system. This can occasionally happen, especially if proper maintenance and compliance with the devices' manuals and instructions during the exploitation is not                                                                                                                                                  |

| Threat                                | Threat description                                                                                                                                                                                                                                                        |
|---------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| system                                | ensured.                                                                                                                                                                                                                                                                  |
| Software vulnerabilities exploitation | The threat that an attacker takes advantage of IIoT end device firmware or software vulnerabilities. Such devices are often vulnerable due to lack of updates, usage of weak or default passwords and improper configuration..                                            |
| Failure of service providers          | The threat of disruption of processes that rely on third party services in case of failure or malfunction of these services.                                                                                                                                              |
| <b>Outages</b>                        |                                                                                                                                                                                                                                                                           |
| Communication network outage          | The threat of unavailability of communication links related to problems with cable, wireless or mobile network.                                                                                                                                                           |
| Power supply outage                   | The threat of failure or malfunction of the power supply. If no emergency power supply exists for critical systems, any power supply disruption may result in serious consequences due to a sudden shutdown of production processes.                                      |
| Loss of support services              | The threat of failure or malfunctions of systems supporting                                                                                                                                                                                                               |
| <b>Legal</b>                          |                                                                                                                                                                                                                                                                           |
| Violation of rules and regulations    | The threat of legal issues and financial losses related to personal data processing, e.g. related to the usage of IIoT end devices without complying with local laws or regulations. In operations within the European Union, these requirements are imposed on companies |

| Threat                     | Threat description                                                                                                                                                       |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                            | by the GDPR.                                                                                                                                                             |
| Failure to meet a contract | The threat of violating contractual requirements by components manufacturers and software providers in case of failure to ensure the required security measures.         |
| <b>Disaster</b>            |                                                                                                                                                                          |
| Natural disasters          | The threat of natural disasters such as floods, lightning strikes, heavy winds, rain and snowfall, which may cause physical damage to the environment components.        |
| Environmental disasters    | The threat of incidents and unfavorable conditions such as fires, pollution, dust, corrosion, explosions, which may cause physical damage to the environment components. |

### ***24.3.2 Security measures***

The document [5] describes three groups of IoT measures, including policies, organizational practices, and technical practices.

This first group of security measures mostly refers to policies and procedures that should be established within organizations to help ensure a good level of cybersecurity, especially where IoT solutions are concerned. In addition, privacy issues have been covered in the context of manufacturers who should ensure that their solutions do not violate privacy regulations, and operators, who should be sensitized to privacy related risks and made aware of how to utilize IoT devices without exposing users' personal information. Policies include four the following categories which contains 24 practices [5]:

- Security by design includes security measures which should be applied from the very beginning of product development;

- Privacy by design includes security measures related to privacy and protection of personal data. These measures should be applied from the first stages of product development;

- Asset Management includes security measures regarding asset discovery, administration, monitoring and maintenance;
- Risk and Threat Management includes security measures regarding the recommended approach to the process of risk and threat management adapted to IoT environment.

Organization principles and governance are indispensable factors that are usually critical in terms of company security. The following security measures explain how industrial companies should operate, what organizational rules and responsibilities they should establish and follow and what approach they should adopt towards their employees and third party contractors to handle effectively cybersecurity incidents, manage vulnerabilities and ensure security of IoT solutions throughout their lifecycle. Organizational practices include six the following categories which contains 27 practices [5]:

- Endpoints lifecycle includes security measures related to security at different stages of product (including end devices and infrastructure) lifecycle, including the procurement process, supply chain, handover phase, exploitation and end-of-life;

- Security Architecture includes security measures regarding the architectural-based approach and establishment of security architecture.

- Incident handling includes security measures regarding the detection and response to incidents that may occur in IoT system environments;

- Vulnerabilities management includes security measures on the vulnerability management process, related activities and vulnerability disclosure;

- Training and Awareness includes security measures regarding the recommended approach related to security training and raising awareness of employees working with IoT devices and systems;

- Third Party Management includes security measures related to third party management and control of third party access.

Apart from implementing policies and organizational practices, security also needs to be addressed through the appropriate technical capabilities of IoT solutions and the environments where they are deployed. The technical security measures listed below constitute a last piece of the puzzle enabling IoT and Industry 4.0 companies to improve their level of security. Technical practices include ten the following categories which contains 59 practices [5]:

- Trust and Integrity Management includes security measures that can help ensure the integrity and trustfulness of data and devices;
- Cloud security includes security measures regarding various security aspects of cloud computing;
- Business continuity and recovery includes security measures regarding the development, testing and reviewing of company’s plan to ensure resilience and continuity of operations in the event of security incidents;
- Machine-to-Machine security includes security measures regarding key storage, encryption, input validation and protection in Machine-to-Machine communications security;
- Data Protection includes security measures regarding protection of confidential data on various levels of an organization and management of access to data;
- Software/Firmware updates include security measures regarding verification, testing and execution of patches.
- Access Control includes security measures regarding the control of remote access, authentication, privileges, accounts and physical access;
- Networks, protocols and encryption include security measures those can help ensure security of communications through proper protocols implementation, encryption and network segmentation;
- Monitoring and auditing includes security measures regarding the network traffic and availability monitoring, logs collection and reviews;
- Configuration Management includes security measures regarding security configuration, management of changes in configuration, devices hardening and backup verification.

### ***24.3.3 Threat and attacks modeling for IoT systems***

A kill chain is a systematic process to target and engage an adversary to create desired effects. U.S. military targeting doctrine defines the steps of this process as find, fix, track, target, engage, assess (F2T2EA): find adversary targets suitable for engagement; fix their location; track and observe; target with suitable weapon or asset to create desired effects; engage adversary; assess effects.

This is an integrated, end-to-end process described as a “chain” because any one decency will interrupt the entire process [6].

With respect to computer network attack or computer network espionage, the definitions for these kill chain phases are as follows:

1. Reconnaissance – Research, identification and selection of targets, often represented as crawling Internet websites such as conference proceedings and mailing lists for email addresses, social relationships, or information on specific technologies.

2. Weaponization – Coupling a remote access trojan with an exploit into a deliverable payload, typically by means of an automated tool (weaponizer). Increasingly, client application data files such as Adobe Portable Document Format (PDF) or Microsoft Office documents serve as the weaponized deliverable.

3. Delivery – Transmission of the weapon to the targeted environment. The three most prevalent delivery vectors for weaponized payloads by advanced persistent treats actors, as observed by the Lockheed Martin Computer Incident Response Team (LM-CIRT) for the years 2004-2010, are email attachments, websites, and USB removable media.

4. Exploitation – After the weapon is delivered to victim host, exploitation triggers intruders' code. Most often, exploitation targets an application or operating system vulnerability, but it could also more simply exploit the users themselves or leverage an operating system feature that auto-executes code.

5. Installation – Installation of a remote access trojan or backdoor on the victim system allows the adversary to maintain persistence inside the environment.

6. Command and Control (C&C) – Typically, compromised hosts must beacon outbound to an Internet controller server to establish a C&C channel. Advanced malware especially requires manual interaction rather than conduct activity automatically. Once the C&C channel establishes, intruders have “hands on the keyboard” access inside the target environment.

7. Actions on Objectives – Only now, after progressing through the first six phases, can intruders take actions to achieve their original objectives. Typically, this objective is data exfiltration which involves collecting, encrypting and extracting information from the victim environment; violations of data integrity or availability are potential

objectives as well. Alternatively, the intruders may only desire access to the initial victim box for use as a hop point to compromise additional systems and move laterally inside the network.

The threats and risks previously listed could be used by attackers to cause cascade effects and further damages at different levels in the infrastructures. The different attack scenarios and the level of importance of each attack have been gathered from the desktop research as well as the information provided by the experts who have contributed to the report. It is worth noting that the attacks may take place throughout the whole process, and the impact that attacks may have on each specific part of the process has also been analysed.

There are the following types of the most critical attacks of IoT systems [7]:

- Against the network link between controller(s) and actuators;
- Against sensors, modifying the values read by them or their threshold values and settings;
- Against actuators, modifying or sabotaging their normal settings;
- Against the administration systems of IoT;
- Exploit Protocol vulnerabilities;
- Against devices by injecting commands into the system console;
- Stepping stone attacks;
- DDoS using an IoT botnet;
- Power source manipulation and exploitation of vulnerabilities in data readings;
- Ransomware.

Let's consider a botnet attack, as an example for modeling. This attack entails the exploitation of some vulnerability inside a device to inject commands and obtain administrator privileges, with the purpose of creating a botnet made up of those vulnerable IoT devices. A botnet is a network of automatic devices that interact to accomplish some distributed task. Due to the characteristic interconnection of IoT devices and their poor configuration, carrying out such an attack is simple. This attack scenario is based on the Mirai botnet, which has conducted several of the most forceful attacks in recent history, and has proven capable of attacking varied kinds of targets. Therefore, with potential targets such as a hazardous energy infrastructure, the impact of a Mirai's attack can reach extremely critical levels. A kill chain for a botnet attack which includes the following actions:

- Scanning open ports in IoT devices that are accessible over the Internet, which are usually poorly protected by default usernames and passwords that, users never change;
- Gaining access to the device;
- Commands injection into the device’s console;
- Obtain administrator privileges;
- Connect the device to a C&C;
- Execution a malicious script;
- Deleting the script itself afterward and running in-memory;
- Spread attacking the same way other vulnerable devices, in order to gather an IoT device army, conscripting them into a botnet;
- Control the botnet from a C&C centre, in order to launch distributed attacks.

#### **24.4 Work related analysis**

Some lack of standardization efforts is identified in area of IoT safety and security [8]. An issue is a new security focus in IoT applications area. For example, the [8] identifies the following five main IoT areas: connected vehicles, consumer IoT, health and medical devices, smart buildings, and smart manufacturing.

Cybersecurity objectives for traditional information technology systems generally prioritize confidentiality, then integrity, and lastly availability. IoT systems cross multiple sectors as well as use cases within those sectors. Accordingly, cybersecurity objectives may be prioritized very differently by various parties, depending on the application.

The increased ubiquity of IoT components and systems heighten the risks they present. Standards-based cybersecurity risk management will continue to be a major factor in the trustworthiness of IoT applications. Analysis of the application areas makes it clear that cybersecurity for IoT is unique and requires tailoring existing standards and creating new standards to address challenges, for example:

- pop-up network connections,
- shared system components,
- the ability to change physical aspects of the environment, and
- related connections to safety.

However in the last 1-2 years this gap is partly fulfilled by technical reports describing good practices in IoT area [5,7]. Also there are researches targeted to development and adjustment of dependability and security models for IoT systems as the followings.

A team of University of Coimbra focuses on the paradigm of the fog orchestration as a basic of IoT Service Layer with open challenges, technological directions [9]. Another research direction of the team is features of resilience for cyber-physical systems [10].

A.-L. Kor and C. Pattinson from Leeds Beckett University pay attention to social aspects of IoT implementation in frame of user-oriented design methodology named SMART-ITEM [11].

We can consider the typical studies in area of IoT safety and security, which take into account the following research scenario [5]:

- To defines relevant terminology to promote common understanding of relevant dependability and cybersecurity issues;
- To categorizes in a comprehensive taxonomy of the assets across the information process and value chain;
- To introduce detailed hazards and threats taxonomy based on related risks and attack scenarios;
- To maps the identified hazards and threats to assets, thus facilitating the deployment of dependability and security measures based on the customized requirements of interested stakeholders;
- To list dependability and security measures related to the use of IoT system and map them against the aforementioned hazards and threats.

### ***Conclusions and questions***

Dependability and security models are represented in the section based on concept “safety risks – hazards; security risks – treats”, and implementation it in the appropriated approaches to assess the most important indicators.

Dependability and safety models are mostly quantitative based on probabilistic analysis of indicators values. Security models are mostly qualitative based on threats analysis and the associated attacks scenario.

A common basic of safety and security risks, hazards and threats allows to consider integration of these IoT systems features. A set of

requirements to IoT systems contains issues of process management, life cycle, random and systematic failures avoidance, as well as attacks avoidance.

In order to better understand and assimilate the educational material that is presented in this section, we invite you to answer the following questions.

1. What types of risks arise in IoT systems, and what is the nature of each of these risks?
2. How are dependability, safety and security interrelated?
3. What are the challenges in terms of safety and security those are created by the rapid increase in the number of devices connected to the Internet?
4. What are the groups of requirements for safety and security?
5. What attributes of functional safety does IEC 61508 define?
6. How do safety attributes constitute a common system with attributes of security and dependability?
7. What attributes does the abbreviation RAMS include?
8. What is the difference between reliability, dependability, availability and safety?
9. What is risk? Please let's explain features of concept of risk for different systems.
10. How can risk be assessed qualitatively and quantitatively?
11. What is the principle of ALARA?
12. What are the main indicators of dependability and formulas for their calculations?
13. What are the main indicators of safety and formulas for their calculations?
14. Why is it necessary to analyze simultaneously indicators of dependability, safety and security?
15. How is the method of Reliability Block Diagrams applied?
16. How is the method of Failure Mode, Effect and Criticality Analysis (FMECA) applied?
17. What are the main threats of IoT systems?
18. Which security measures are recommended to be implemented for IoT systems?
19. What are the most serious cyber-attacks of IoT systems?
20. How threats and attacks can be modeled for IoT systems?

21. What modifications of FMECA technique do you know that can be applied for security analysis (for example technique IMECA and others)?

### *References*

1. Скляр В.В. Обеспечение безопасности АСУТП в соответствии с современными стандартами. – Инфра–Инженерия, 2018.

2. Rausand M. Reliability of safety–critical systems : theory and application. – John Wiley & Sons, Inc., Hoboken, New Jersey, USA, 2014.

3. Федоров Ю.Н. Справочник инженера по АСУ ТП: Проектирование и разработка. – Инфра–Инженерия, 2008.

4. Avižienis, A., Laprie, J.-C., Randell, B. and Landwehr, C. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing (2004), 1(1): 11-33.

5. Good Practices for Security of Internet of Things in the context of Smart Manufacturing. – The European Union Agency for Network and Information Security, 2018.

6. Baseline Security Recommendations for IoT in the context of Critical Information Infrastructures. – The European Union Agency for Network and Information Security, 2017.

7. Hutchins E., Cloppert M., Amin R. Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains. – Lockheed Martin Corporation, 2017.

8. NISTIR 8200, Interagency Report on the Status of International Cybersecurity Standardization for the Internet of Things (IoT). – National Institute of Standards and Technologies, 2018.

9. Velasquez, K., Abreu, D.P., Assis, M. et al. Fog orchestration for the Internet of Everything: state-of-the-art and research challenge. Journal of Internet Services and Applications (2018) 9: 14.

10. Curado M. et al. Internet of Things. In: Kott A., Linkov I. (eds) Cyber Resilience of Systems and Networks. Risk, Systems and Decisions. Springer, 2019.

Kor A.L., Pattinson C., Yanovsky M., Kharchenko V. IoT-Enabled Smart Living. In: Dastbaz M., Arabnia H., Akhgar B. (eds) Technology for Smart Futures. Springer, 2018.

## 25. SAFETY AND SECURITY MANAGEMENT OF IOT

DrS. Prof. V. V. Sklyar (KhAI)

### *Contents*

|                                                                       |     |
|-----------------------------------------------------------------------|-----|
| Abbreviations .....                                                   | 318 |
| 25.1 Safety and security management requirements to IoT .....         | 319 |
| 25.1.1 Safety and security management plan .....                      | 319 |
| 25.1.2 Human resource management.....                                 | 321 |
| 25.1.3 Configuration management .....                                 | 322 |
| 25.1.4 Tools selection and evaluation .....                           | 324 |
| 25.1.5 Documentation management .....                                 | 326 |
| 25.1.6 Safety and security assessment.....                            | 327 |
| 25.2 Safety and security life cycle for IoT .....                     | 329 |
| 25.2.1 Overall life cycle .....                                       | 329 |
| 25.2.2 Safety and security life cycle: design top-down brunch .....   | 330 |
| 25.2.3 Safety and security life cycle: integration down-top brunch .. | 331 |
| 25.2.4 Requirements tracing.....                                      | 331 |
| 25.3 Review, analysis and testing techniques for IoT .....            | 334 |
| 25.3.1 Documents review .....                                         | 334 |
| 25.3.2 Static code analysis.....                                      | 335 |
| 25.3.3 Functional testing .....                                       | 335 |
| 25.3.4 Code structural testing.....                                   | 336 |
| 25.4 Work related analysis .....                                      | 337 |
| Conclusions and questions.....                                        | 338 |
| References .....                                                      | 339 |

### ***Abbreviations***

FMECA – Failure Mode, Effect and Criticality Analysis

IEC – International Electrotechnical Commission  
ISA – International Society of Automation

ISMS – Information Security Management System

ISO – International Standardization Organization

NIST – National Institute of Standards and Technologies

SAD – System Architecture Design

SCA – Static Code Analysis

SRS – Safety Requirements Specification

SSLC – Safety and Security Life Cycle

SSMP – Safety and Security Management Plan

TP&S – Test Plan and Specification

TR – Test Report

V&V – Verification and Validation

## **25.1 Safety and security management requirements to IoT**

### ***25.1.1 Safety and security management plan***

General structure of requirements to safety and security is considered in Section 24.

The umbrella part of requirement is related with safety and security management. Safety and security management plan (SSMP) is the document, which states the main safety and security issues for specific IoT system or systems development and operation project.

The SSMP covers a set of processes which can be developed in a view of separated document. There are the following safety and security processes which have to be reflected in the SSMP [1]:

- Human Resource Management (see 25.1.2);
- Configuration Management (see 25.1.3);
- Tools Selection and Evaluation (see 25.1.4);
- Verification and Validation (see 25.3);
- Requirements Tracing (see 25.2.4);
- Documentation Management (see 25.1.5);
- Safety and Security Assessment (see 25.1.6).

Also SSMP has to cover the following issues (see Fig. 25.1) [2,3]:

- Project Policy and Strategy is a declarative description of how and why the goals of the project will be achieved;
- Project Management is reasonable applicable to project performance since, for example, the IEC 61508-2 (Annex B) requires applying this method to protect the product against systematic failures;
- Quality Management System it important to implement quality for all products and processes; special attention is paid to interaction with suppliers of products and services that affect safety and security;
- Information Security Management System (ISMS) has to cover activities in accordance with requirements of ISO/IEC 27000 “Information technology – Security techniques – Information security management systems” or any other relevant ISMS framework [4];
- Safety & Security Life Cycle has to be described in SSMP stage by stage (see 25.2).

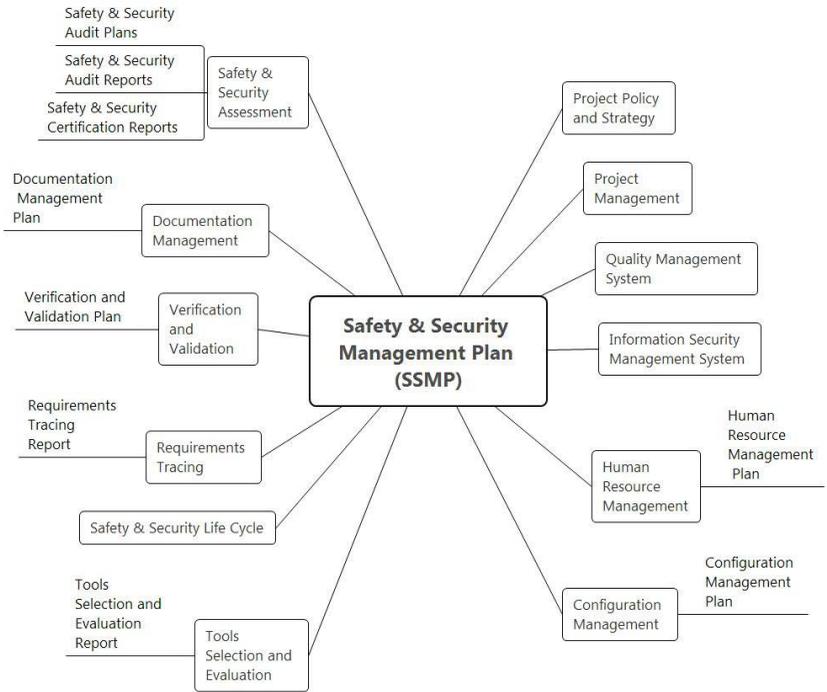


Fig. 25.1 – Structure of Safety and Security Management Plan (SSMP)

All the above activities cover both safety and security issues. Additionally ISMS has to cover activities like the following: asset management, identification and authentication, access control, system perimeter protection, work stations, servers, and other devices protection, network and communications protection, cloud infrastructure protection, database protection, cryptography, monitoring and recovery, incidents response and investigation. All appropriate measures and activities have to be implemented for the considered IoT system.

### 25.1.2 Human resource management

For detailed personnel management planning, an appropriate Human Resource Management Plan has to be developed. Note that this plan does not apply to the organization as a whole, but only to the participants in the project of IoT system development and certification against safety and security requirements. The personnel management plan should contain (see Fig. 25.2):

- Organizational chart of the project with a description of project roles;
- A list of project participants indicating project roles and responsibilities for planning and performing work at various stages of the life cycle;
- The competence matrix and the conclusions on the adequacy or lack of competencies of the appointed performers, i.e. what knowledge and skills are required for a particular project role and to what extent a particular employee corresponds to them;
- Personnel training activities aimed at achieving and maintaining the above mentioned competences that are critical for the implementation of the project; training plans and reports should be documented;
- Communication plan for the project participants;
- A list of the signatures of personnel, indicating the familiarization with this plan.

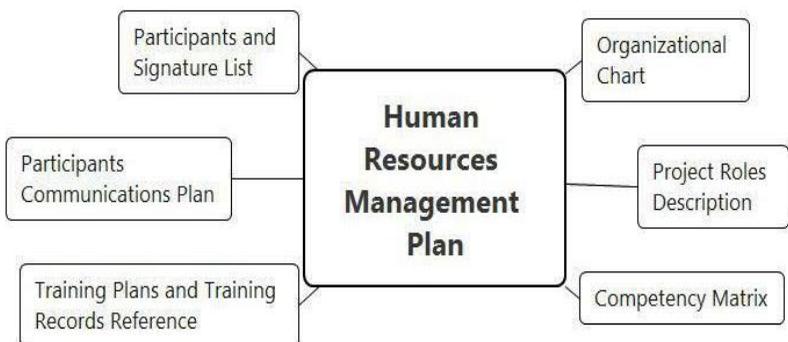


Fig. 25.2 – Structure of Human Resource Management Plan

### 25.1.3 Configuration management

When defining configuration items in the context of safety and security, it is important to understand that they include not only source codes and program builds, but also development and testing tools, a complete set of design, user, and any other relevant documentation, including design documentation, according to which all mechanical, electrical and electronic components are manufactured (see Fig. 25.3). Such a structure can serve as the basis for the project repository.

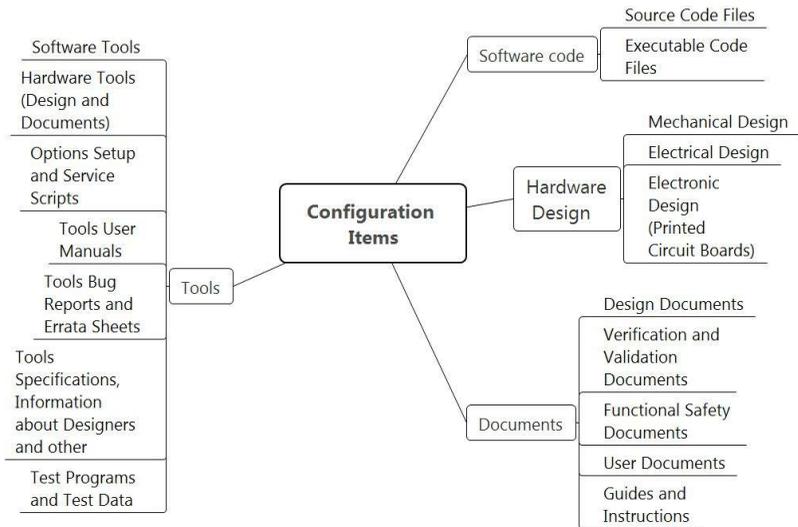


Fig. 25.3 – A set of configuration items of IoT system

Configuration management directly depends on the used electronic document management tools, however, some the following general points can be included in the Configuration Management Plan (see Fig. 25.4):

- The roles and responsibilities of project participants in the configuration management process; the Configuration Management & Change Control Board of the key project participants should be organized with all those, whose opinions are important to consider when making changes;

- An approach to planning and maintaining the configuration management process;
- Resources of the configuration management process, first of all, the applied tools of electronic document management (SVN, Git, etc.);
- The procedure for the identification of the configuration items and the formation of baselines (basic versions);
- The procedure for applying tools to control the versions of software and hardware components of the product and to account for their status;
- The procedure for accessing configuration components and backup storage;
- The procedure and periodicity for configuration audits;
- The procedure for analyzing and eliminating the detected defects and bugs including those found during operation;
- The procedure for change control, including impact analysis and validation of changes.

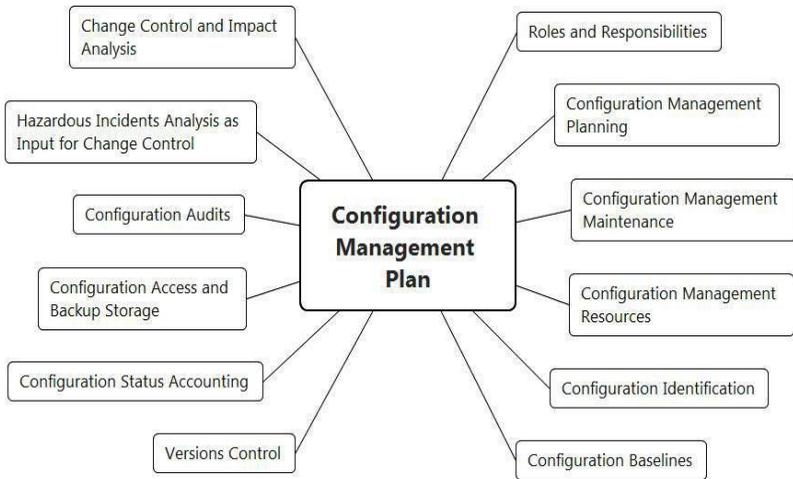


Fig. 25.4 – Structure of Configuration Management Plan

### 25.1.4 Tools selection and evaluation

The IEC 61508 “Functional safety of electrical/ electronic/ programmable electronic safety-related systems” states the following tools classification depending on the degree of influence on the final product, system, or software (see Fig. 25.5):

- Class T1 tools do not generate any outputs that directly affect the executable code; it includes text and image editors, configuration management tools (those do not directly generate code), action & bug trackers;

- Class T2 tools support testing and other types of verification and validation (for example, static code analysis or test coverage analysis); there is no direct impact on the executable code, however, a problem in the test tools may lead to errors in the software that may not be detected; this class should include not only software, but also software / hardware simulators of input / output signals; it should be noted that design tools for mechanical, electrical and electronic components (for example, printed circuit boards design tool) can also be assigned to class T2;

- Class T3 tools generate outputs that directly affect the executable code, such as translators and compilers that are components of Integrated Development Environments (IDE) & Software Development Kits (SDK), scripts to support builds and controller logic configuration.

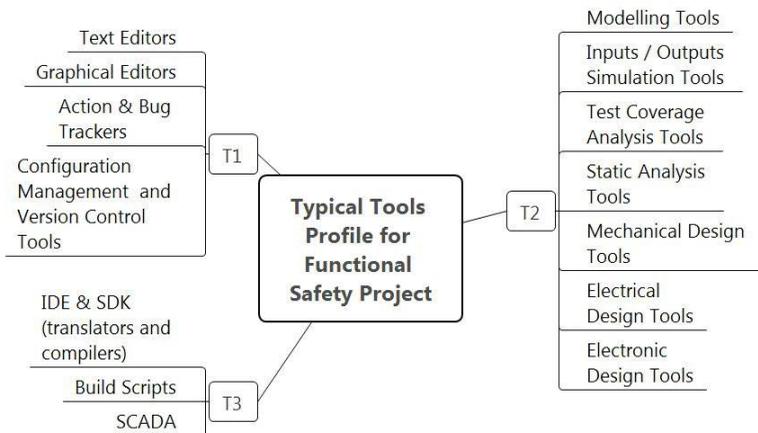


Fig. 25.5 – Tools classification

To ensure compliance with safety and security requirements, it is advisable to develop a special report on the selection and evaluation of tools that shall cover the following issues (see Fig. 25.6):

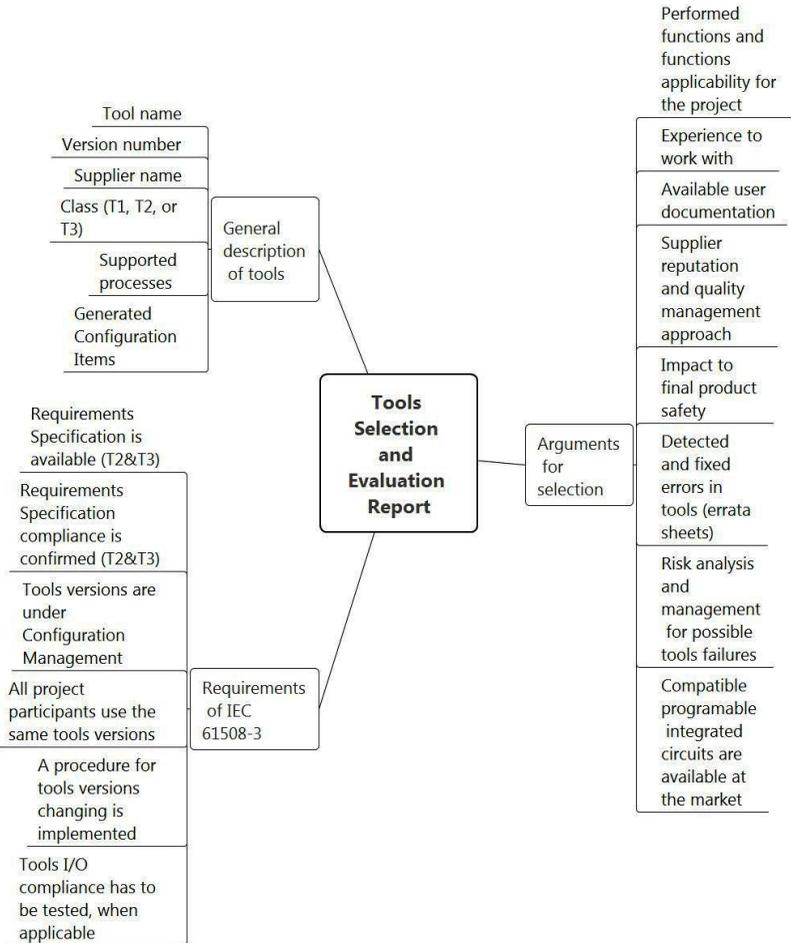


Fig. 25.6 – Structure of Tools Selection and Evaluation Report

– A description of the used stack of tools (both software and hardware, both commercially available and in-house) used for product

development, testing, and supporting processes (configuration management, documents processing, project management, etc. .) for each of the tools you should specify: type (to support which process is used), name, version number, supplier name, class (T1, T2 or T3), as well as generated outputs in terms of Configuration Items;

– Results of evaluation (analysis) of tools according to a set of predetermined criteria, such as, for example: the functions performed and their applicability in this project, experience of use, available documentation, information about the supplier (market reputation, quality management system, approach to configuration management and etc.), the impact on the safety of the product, the errors found and eliminated, the possible risks of use in terms of failures and the strategy for managing these risks, the availability of compatible products on the market programmable chips (for software development and electronic projects);

– The results of the analysis for compliance with the requirements for the tools specified in IEC 61508-3, such as:

- for tools of classes T2 and T3 requirements specifications or user documentation should be available that uniquely describe how the operation takes place;

- for tools of classes T2 and T3 their compliance with the requirements specification or user documentation has to be documented (for example, in the form of a certificate);

- the versions of the tools used should be monitored, since not all versions can meet the specified conditions; all project participants must use the same version; for transitions between versions the appropriate procedure should be applied;

- if the tools are used as a single technological complex (for example, code and tests are generated based on the specification), their compatibility with each other should be tested.

### ***25.1.5 Documentation management***

For detailed documentation management a related Documentation Plan has to be developed. That plan does not apply to the organization as a whole, but only to the participants of a considered project for developing a product important to safety and security. The Documentation Plan has to cover the following issues (see Fig. 25.7):

- Requirements to identification, development, execution, coordination and approval of documents;
- Review procedures and criteria for evaluating documents (for example, in the form of checklists);
- A list of project documents and allocation of responsibility for the development, review and approval;
- The procedure for access to documents and access rights of project participants;
- The procedure for making changes to documents, accounting policy and version changes;
- Requirements to use of electronic document management system;
- A structure of the project repository.

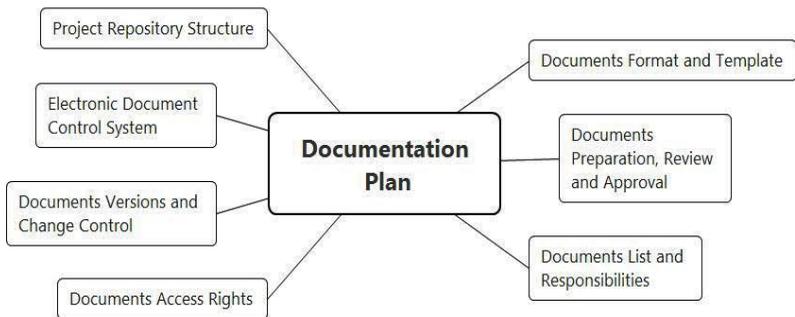


Fig. 25.7 – Structure of Documentation Plan

### ***25.1.6 Safety and security assessment***

To assess safety and security during the project, periodic safety and security audits are conducted. These audits can be either internal (conducted by the project team) or external (performed by the third party). Another kind of audits is the certification audit, which is conducted upon completion of the project work by the certification authority. According to the results of the certification audit, a certificate of compliance with the standards requirements is issued. In addition, the certification authority may also participate in periodic audits. Audits should be conducted according to pre-developed plans. In the audit

plan, the following issues have to be defined (see Fig. 25.8 as an example for Functional Safety Audit):



Fig. 25.8 – Structure of Functional Safety Audit Plan

- Periodicity of audits (for example, at the completion of each of the development stages);
- Areas of assessment in terms of the structure of products and processes;
- Involved participants, organizations and other required resources (temporary, financial, required tools, etc.);
- The level of independence of auditors; as noted above, audits can be internal and external; in general, the issue of independence in evaluating safety has its traditions in various industries and countries;
- Competencies of the employers performing the audit;
- Expected results;
- Corrective actions performance;
- An approach to document audit results and requirements for the content of audit report, which shall be issued based on the results of audits;
- Checklists, including a specific set of requirements (issues), compliance with which should be evaluated during the audit; the initial

data for compiling an audit checklist are the requirements of SSMP and other plans related to ensuring of safety and security.

## 25.2 Safety and security life cycle for IoT

### 25.2.1 Overall life cycle

Existing standards do not describe a life cycle for IoT systems. Thus, we propose interpretation of Safety & Security Life Cycle (SSLC) based on requirements to critical programmable systems (see Fig. 25.9). Used abbreviations are given below.

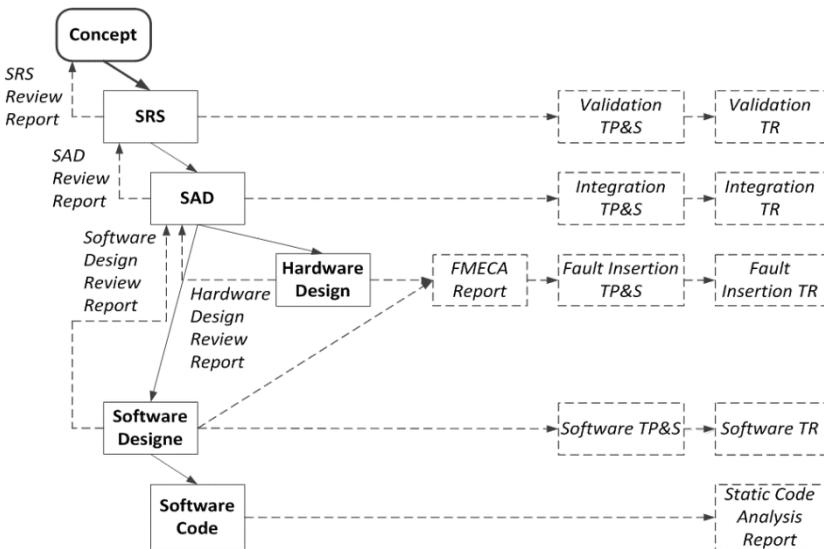


Fig. 25.9 – V-shape Safety & Security Life Cycle

The content of the SSLC stages and the relevant documents are detailed in subsections 25.2.2, 25.2.3. The life cycle model includes the sequentially performed steps (in the diagram, the steps are indicated by the names of the final documents). For top-down branch details of design is performed from system level to hardware and software parts. For down-top branch staged integration with appropriate testing is performed before for software and hardware parts, and after for programmed component and for system as whole.

### ***25.2.2 Safety and security life cycle: design top-down brunch***

Step by step description of the design details has to be done for IoT system and its hardware and software parts. This design development includes the following stages (see Fig. 25.9).

Development of the product concept means creating of a top-level concept document (for example, a contract) which defines the needs of enterprises or businesses and automation processes, including the identification of hazards and threats.

Development of the Safety Requirements Specification (SRS) covers describing the system in the form of a “black box”, that is, “what it is performed” and not “how it is performed”. The SRS has to contain functional and safety requirements, including modes, time characteristics, interfaces, signals, self-diagnostics, periodic testing, limiting external conditions and other.

Review of the SRS for compliance with the requirements of the Concept is a stage of Verification and Validation (V&V) process.

Development of the System Architecture Design (SAD) represents the system in a view of a “white box”, that is, “how it is performed”, and not “what it is performed”, including a detailed structure and behavior description.

Review of the SAD for compliance with the requirements of the SRS is a stage of V&V.

Development of the Hardware Design covers creating of design documentation for hardware which includes both projects of electronic boards and drawings of mechanical structures and electrical parts, including cables, power supply and interface components for field equipment (sensors and actuators).

Review of the Hardware Design for compliance with the requirements of the SAD is a stage of V&V.

Failure Mode, Effect and Criticality Analysis (FMECA), is the stage of V&V process (see 24.2.4). When performing FMECA, the hardware structure is primarily taken into account, however, the diagnostic and fault tolerance mechanisms implemented in the software are also taken into account.

Development of the Software Design covers creating of documentation for the software on the basis of which coding is carried out.

Review of the Software Design for compliance with the requirements of the SAD is a stage of V&V.

Software Coding covers creating of source code development.

Static Code Analysis (SCA) is a stage of V&V when code is verified for compliance with the Software Design including coding rules and others.

### ***25.2.3 Safety and security life cycle: integration down-top brunch***

Step by step integration of hardware and software parts has to be done for IoT system. This integration and associated testing include the following stages (see Fig. 25.9).

Software Testing is a stage of V&V when code is verified for compliance with the Software Design. It includes unit and integration testing, as well as both functional and structural testing. Before testing, the Software Test Plan and Specification (TP&S) has to be developed, and the results shall be documented in the Software Test Report (TR).

Fault Insertion Testing is a stage of V&V when code is verified for compliance with the results of FMECA. Testing is performed after seeding defects in hardware and software. Inputs for testing are produced by FMECA in the part of analysis of the implementation of self-diagnostics. After that malfunctioning hardware and software is tested to check implementation of self-diagnostic functions.

Integration Testing is a stage of V&V when integrated system parts are verified for compliance with the SAD.

Validation Testing is a stage of V&V when integrated system is verified for compliance with the SRS. Validation may include, in addition to functional testing, also testing for resistance to external environmental impacts.

### ***25.2.4 Requirements tracing***

Requirements tracing is one of the processes of a wider area of knowledge called Requirements Engineering [5]. Requirements tracing is a method for managing changing requirements and related artifacts. Requirements tracing solves three main tasks:

- To ensure the implementation at the lower level of all the requirements of the upper level,
- To prevent from undocumented functions appearing on the lower level,

- To support testing of all requirements.

Specialized software is used to manage requirements. One of the most famous of these tools is IBM Rational DOORS. To perform requirements tracing, documents must be prepared for this process by arranging requirements identifiers and tags that define the boundaries of the wording of requirements.

In the considered life cycle (see Fig. 25.9), requirements are traced between design documents as follows. First, direct tracing of requirements from SRS to SAD is performed. Then backtracking of the requirements from SAD to the SRS is performed in order to make sure that the SAD does not include extra functionality that is not documented in the SRS. After SAD, the design process is divided into two streams, which are Hardware Design & Software Design. Forward and backward tracing is performed for both documents. Hardware Design includes mainly drawings in which it is problematic to place tags, so the Hardware Design Review Report is laid out under the tracing (see Fig. 25.10).

During testing, requirements tracing is done by extracting requirements from project documents. For complicated projects, the development of test documents can take place in two stages. First, a test plan is developed, containing a list of test requirements, and then test cases are developed for each requirement in the test specification. During testing, direct tracing of requirements is carried out from the project document to the test plans and specifications, and then to the testing report. For testing by the method of seeding defects, the list of tests is extracted from the FMECA report by analyzing self-diagnosed failures.

For a reasonable set of failures, a set of tests is made, on which diagnostic functions are checked. Backward tracking is not critical here, because if additional tests are performed that are not due to project documents, this will not affect safety and security (see Fig. 25.11).

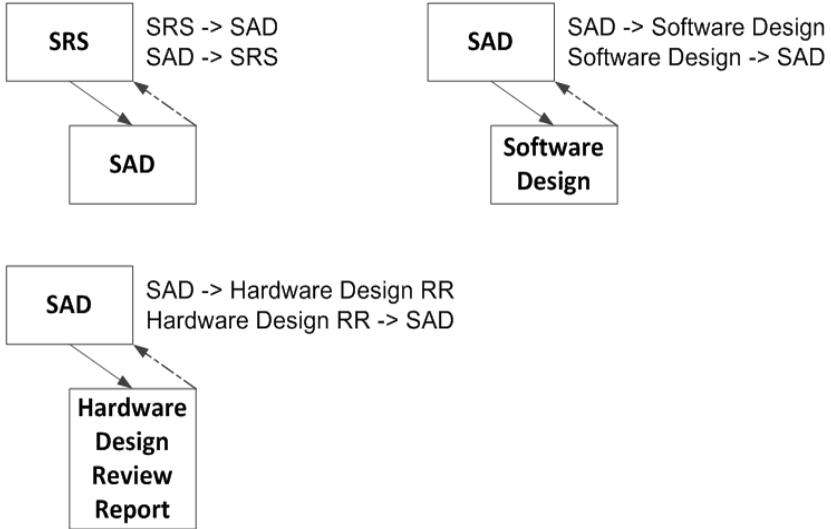


Fig. 25.10 – Requirements tracing for design stages

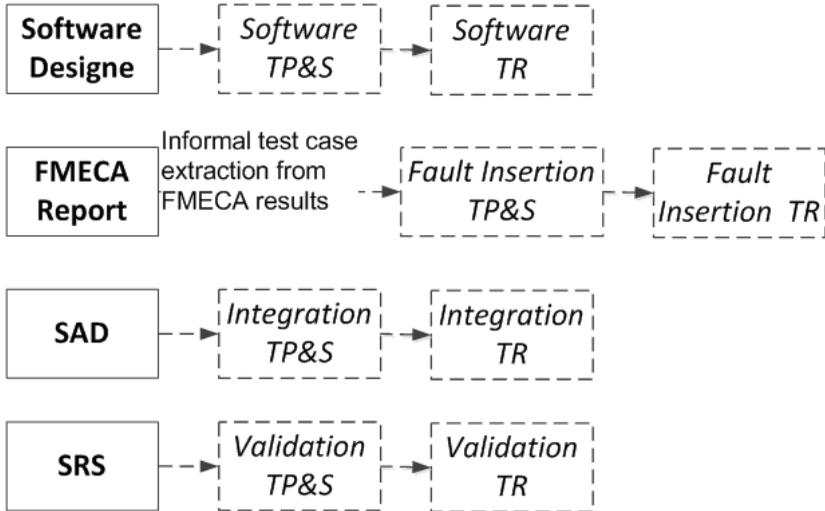


Fig. 25.11 – Requirements tracing for testing stages

## **25.3 Review, analysis and testing techniques for IoT**

### ***25.3.1 Documents review***

Document review performed for design of IoT system is the process whereby review team to a case sorts through and analyzes the documents and data. During review, a design document has to be verified against input requirements [2].

Assessment criteria of documents include compliance of documents with a requested set of functional and non-functional requirements. Requirements stated in the document have to be verifiable and testable, as well as feasible. Formal and semi-formal methods can be applied to describe design functionality. Quality criteria are assessed to ensure a document is clear, precise, unambiguous, maintainable and understandable. If requirements tracing is implemented, it has to be an additional activity of documents review related to check how a document is fit for tracing.

All criteria of a document assessment are stated in a document check-list, which is used as a tool of document review.

For software design document some specific assessment criteria have to be stated in relation with desirable software architecture. There are the following assessment criteria for software design documents review/

Software architecture has to be as simple as possible. It should not be many levels of hierarchy. It should not be much complexity from point of view of program brunches and loops. It should be using of as many standardized and proven in use components as possible.

Software architecture has to have relationship with the software requirements, and this relationship has to be clearly explained and motivated. Also it can be approved via requirements tracing. All requirements have to be covered. Functionality which is not described in requirements has not to be implemented in architecture. Flexibility of the architecture has to be demonstrated.

Components and interfaces of software architecture have to be precisely described. Routine kind, name, parameters and their types, return type, pre- and post-condition, usage protocol etc. have to be described. File name, format, permissions have to be described.

Software architecture can be described in a view of different UML diagrams which have to represent logical, process, and physical view.

All the following cross-cutting issues have to be resolved: exception handling, initialization and reset, memory management, built-in test facilities.

### ***25.3.2 Static code analysis***

SCA is the analysis of computer software that is performed without actually executing programs, in contrast with dynamic analysis, which is analysis performed on programs while they are executing [6]. In most cases the analysis is performed on some version of the source code, and in the other cases, some form of the object code. SCA is usually combined with code review. Code review is performed manually and it is similar with documents review. SCA supposes using automated tools.

There are different methods of SCA, and the most important of them are described below.

Coding rules verification is checking of software code against some requirements to coding, for example, using of forbidden code construction, coding style, valuables naming, etc. These rules are described in coding standards.

Control flow analysis checks program control graph against using knots and complicated loops constructions.

Software complexity analysis checks such parameters as quantity of loops and binary decisions, quantity of program interfaces, entrance and exit points, etc.

Formal methods are the term applied to the analysis of software and computer hardware whose results are obtained purely through the use of rigorous mathematical methods. The mathematical techniques used include semantics and abstract interpretation.

### ***25.3.3 Functional testing***

Functional testing is a type of software or system testing whereby the system is tested against the functional requirements [6]. Functions are tested by feeding them input and examining the output. Functional testing ensures that the requirements are properly satisfied by the application. This type of testing is not concerned with how processing occurs, but rather, with the results of processing. It simulates actual

system usage but does not make any system structure assumptions. For IoT functional tests are performed for software as well as for all levels of integrated system, including devices, networks, clouds and application terminals.

TP&S has to be developed before testing implementation. Test plan is developed by traceable extraction of requirements from design documents. 100% of requirements have to be tested. After that test cases have to be developed for each of the requirements. Some test cases are simple but some test cases can include many test scenarios. Acceptance criteria have to be developed for every test case.

Test tools have to be proven in use and evaluated before its selection for some specific IoT system project. Automated tests can be implemented when it is reasonable. Functional tests results have to be documented in the TR.

A feature of software code verification is the analysis of software criticality, during which various software modules are differentiated depending on their participation in safety and security functions. This allows concentrating while providing the focus on the most responsible software and reasonably reducing the scope of measures to ensure the safety and security for the secondary software.

To perform a software criticality analysis, a method called HAZOP (Hazard and Operability) analysis, that is, an analysis of hazards and functioning, can be applied. As a result of performing analysis for software, the performed scope of diagnostic functions can also be justified and initial information for performing FMECA can be obtained.

#### ***25.3.4 Code structural testing***

Structural testing also known white-box testing is a method of testing software that tests internal structures or software as opposed to functional testing [6]. In structural testing an internal perspective of the software are used to design test cases. This is complimentary to functional testing. The tester chooses inputs to exercise paths through the code and determine the expected outputs. Usually it is performed during unit testing.

At the integration level structural testing can be implemented for checking path between units. The following coverage criteria can be applied for structural testing: branch coverage, statement coverage,

decision coverage, modified condition / decision coverage (MC/DC), path testing, etc.

This is complimentary to the basic functional testing, and has to be documented in the TP&S as well as in the TR. Test coverage criteria have to be documented.

When functional tests are performed, structural test coverage can be defined for the functional tests. 100% code coverage can indicate no additional structural tests are needed. However, if code coverage is less than 100%, then additional structural tests cases have to be developed to cover the rest. If finally obtained code coverage is less than 100%, it has to be augmented.

#### ***25.4 Work related analysis***

Safety and security management area is well known and nowadays it is more practical than theoretical. It means there are not many researches in this area. Achieved technological level is committed in modern standard, for example, such as IEC 61508 “Functional safety of electrical/electronic/programmable electronic safety-related systems” or ISA/IEC 62443 “Security for Industrial Automation and Control Systems”. Concerning IoT it is difficult to identify issues which would be specific from point of processes management.

For functional safety management there are some published methodologies directed to certification of safety critical systems [1-3]. Agile development methodologies are widely used for development of non-critical software. However it seems reasonable to use agile methods for safety critical applications taken into account safety requirements. So there are researchers who pay attention to combine and implement agile and safety methodology [7].

ISMS frameworks are developed by some institutions, for example, by National Institute of Standards and Technologies (NIST) [4]. Also NIST analyzes the best practices and standards relevance for IoT [8,9]. In 2019 NIST is planning to issue a new report named “Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks”, which has been already drafted.

For different techniques of software testing and static analysis there are many different techniques and even a brief review of this area would take many pages. For example, researchers of University of

Coimbra pay attention to testing of heterogeneous platforms interoperation. To implement that, they developed special online service, which can be applicable for IoT systems and components [10]. In area of static code analysis the same team is working to investigate vulnerabilities in the source code of web applications [11]. The obtained results can be applied for safety and security critical IoT systems.

### *Conclusions and questions*

Safety and security management issues are a part of critical IoT systems requirements which are extracted from appropriated standards, good practices and frameworks. Safety and security management contain the following main issues:

- Human Resource Management;
- Configuration Management;
- Tools Selection and Evaluation;
- Verification and Validation;
- Requirements Tracing;
- Documentation Management;
- Safety and Security Assessment.

Safety and Security Management Plan has to be developed and implemented as an umbrella document considering the above parts.

Safety and Security Life Cycle implementation is a core of management processes. This life cycle is V-shaped, so it contains top-down brunch related with design and down-top brunch related with integration. Verification and validation activities have to be performed after each of design and integration stage to confirm compliance of the stage inputs with obtained output results. Verification and validation methods include documents review, static code analysis, as well as functional and structural testing.

In order to better understand and assimilate the educational material that is presented in this section, we invite you to answer the following questions.

1. What requirements should be taken into account for managing and assessment of safety and security?
2. What structure of SSMP has to be implemented?

3. What documents can be developed to supplement the functional safety management plan, and in what cases it is advisable to develop such documents?
4. What structure should have Human Resource Management Plan?
5. What part of the Human Resource Management Plan should be developed during the preparatory work for the certification project?
6. List the components of the IoT system configuration.
7. What structure should have Configuration Management Plan?
8. Describe the algorithm of change control for the configuration items.
9. Describe the computer tools classification.
10. Describe the typical set of tools used in IoT systems projects.
11. What structure should have Tools Selection and Evaluation Report?
12. What are the criteria for tools selection and evaluation?
13. What is the relationship between tools, coding rules and software verification?
14. What structure should have Documentation Plan?
15. What types of audits are conducted to assess safety and security?
16. What structure should have Safety and Security Audit Plan?
17. Describe a structure of V-shape life cycle.
18. What is a difference between software life cycle and IoT system life cycle?
19. What is a purpose of requirements tracing?
20. Which design and test documents have to be covered with requirements tracing?
21. Describe documents review method.
22. Describe static code analysis method.
23. Describe functional testing method.
24. Describe structural testing method.

### ***References***

11. Скляр В.В. Обеспечение безопасности АСУТП в соответствии с современными стандартами. Инфра–Инженерия, 2018.

12. Medoff M., Faller R. Functional Safety – An IEC 61508 SIL 3 Compatible Development Process. exida L.L.C., Sellersville, PA, USA, 2010.
13. Smith D., Simpson K. Functional Safety. A Straightforward Guide to applying IEC 61508 and Related Standards. Elsevier Butterworth–Heinemann, Oxford, UK, 2004.
14. NIST SP 800-53 Revision 4, Security and Privacy Controls for Federal Information Systems and Organizations. National Institute of Standards and Technologies, 2015.
15. Standard glossary of terms used in Requirements Engineering, Version 1.3. Requirements Engineering Qualification Board, 2014.
16. Standard glossary of terms used in Software Testing, Version 2.3. International Software Testing Qualifications Board, 2014.
17. Hanssen G., Stålhane T, Myklebust T. SafeScrum® – Agile Development of Safety-Critical Software. Springer, 2018.
18. NISTIR 8200, Interagency Report on the Status of International Cybersecurity Standardization for the Internet of Things (IoT). – National Institute of Standards and Technologies, 2018.
19. NIST SP 1500-201, Framework for Cyber-Physical Systems. National Institute of Standards and Technologies, 2017.
20. Martins B., Laranjeiro N., Vieira M. INTENSE: INteroperability TEstiNg as a Service // Proceedings of 2017 IEEE International Conference on Web Services (ICWS 2017).
21. Nunes P., Medeiros I., Fonseca J. at all. Benchmarking Static Analysis Tools for Web Security. IEEE Transactions on Reliability (2018), 67(3): 1159-1175.

## 26. ASSURANCE CASE FOR IOT

Prof., DrS V. V. Sklyar, Prof., DrS V. S. Kharchenko (KhAI)

### *Contents*

|                                                                                       |     |
|---------------------------------------------------------------------------------------|-----|
| Abbreviations .....                                                                   | 342 |
| 26.1. Assurance Case fundamentals .....                                               | 343 |
| 26.1.1. Assurance Case concept and history.....                                       | 343 |
| 26.1.2. Standards for Assurance Case .....                                            | 346 |
| 26.2. Safety and security techniques and measures for IoT.....                        | 347 |
| 26.2.1. Claims, Arguments and Evidence (CAE) notation.....                            | 347 |
| 26.2.2. Update and application of Claims, Arguments and Evidence (CAE) notation ..... | 350 |
| 26.2.3. Goal Structuring Notation (GSN).....                                          | 356 |
| 26.3. Security informed and energy efficiency informed Assurance Case for IoT .....   | 357 |
| 26.3.1 Tools for development of Assurance Case .....                                  | 357 |
| 26.3.2. Assurance Case structure for IoT systems.....                                 | 360 |
| 26.4 Work related analysis .....                                                      | 363 |
| Conclusions and questions.....                                                        | 364 |
| References .....                                                                      | 366 |

### ***Abbreviations***

CAE – Claim, Argument and Evidence

GSN – Goal Structuring Notation

IEC – International Electrotechnical Commission

ISO – International Standardization Organisation

PMM – Power Modes Management

## **26.1. Assurance Case fundamentals**

### ***26.1.1. Assurance Case concept and history***

Final safety and security assessment is running after completion of all development, verification and validation stages. In this section we discuss how can all project artifacts be represented for safety and security assessment, and what is the way to most effectively confirm compliance with the safety and security requirements? The answer to these questions is provided by the Assurance Case methodology, which is widely used in the practice of safety and security assessment.

The Assurance Case is a structured set of arguments and documentary evidence that justify the compliance of a system or service with specified requirements [1].

Licensing and certification authorities check the Assurance Case, as an integral document proving compliance with the entire set of requirements to safety and security. The Assurance Case can be either compiled by the project team or outsourced.

The historical and theoretical origins of the Assurance Case lie in the field of logical reasoning, such as operations with logical predicates, including the implication. In 1958, the British philosopher Stephen Toulmin published the book “The Uses of Argument” [2], in which he expanded the operation of logical inference with the degree of confidence and additional arguments and counter-arguments. In addition, Toulmin proposed to present the argument in graphical form, and this approach has since become widespread. Toulmin's notation operates on the following entities (Fig. 26.1): data (D) is the initial data for analysis, claim (C) is the goal of logical implication output (If D So C), warrant (W) is an additional argument, qualifier (Q) is the degree of confidence in the results of inference, rebuttable (R) is an additional counter-argument. This approach was initially used exclusively in the humanities.

At the same time, after the Second World War, the rapid development of complex industries, such as nuclear energy, space technology, oil and gas, chemical industries, and transport began. All this was accompanied by the introduction of new at that time automation technology. As a result, humanity was faced with man-made disasters of unprecedented scale.

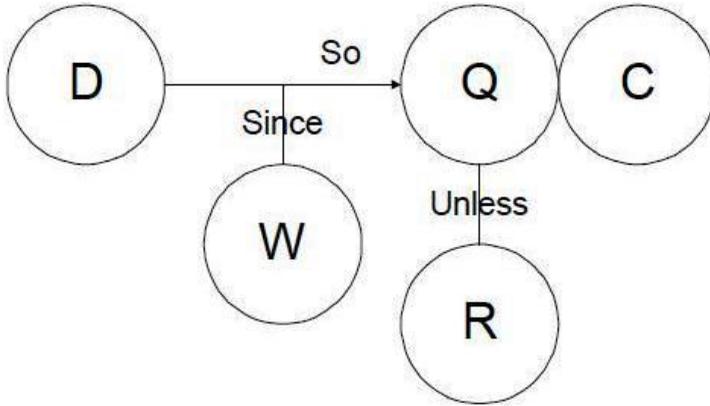


Fig. 26.1 – Argumentation proposed by Stephen Toulmin

Also, in the post-war world, human life was recognized as the highest value. The level of acceptable technical risk was set by law at a fairly hard-to-reach level of  $10^{-6}$  1 / year, i.e. one death per million people per year from technical risks.

Thus, the predecessor of the Assurance Case is historically the Safety Case. The concept of the Safety Case originated in the 1950s, although the term itself appeared later. The first regulatory document requiring the development of a Safety Case for hazardous industrial facilities is the European Union's "CIMAH (Control of Major Accidents Hazards) Regulations". The widespread introduction of the Safety Case into practice began to occur after an unprecedented accident on the Piper Alpha oil platform in the North Sea, which claimed the lives of 167 people in 1988 [3].

All of the above has led to new approaches in safety assessment and assurance. In the 1990s, Toulmin's argument was used as the basis for the development of semi-formal notations to justify safety [1]. The work was done in the UK, at the University of York, where Goal Structuring Notation (GSN) was developed. Adelard developed the Claim, Argument and Evidence (CAE) notation in parallel. These notations are used in the present, and then we consider them in more detail (see subsection 26.2).

Initially, the focus was on functional safety issues (Safety Case), then with the advent of the information security problem, a similar

approach was extended to the Security Case, and with it came the understanding that it was necessary to work simultaneously on providing both safety and security features. Currently, the term Assurance Case means the justification of both safety and security.

In justifying safety and security, we need to confirm the compliance of a certain system or software with the requirements set. At the same time compliance with a particular requirement is the goal of the Assurance Case. In addition, there is a set of documented evidence that requirements are met. To associate evidence with goals and requirements, an argumentation system is used, which is given special attention in the Assurance Case (Fig. 26.2). The lack of arguments or evidence indicates a failure to comply with safety and security requirements [1].

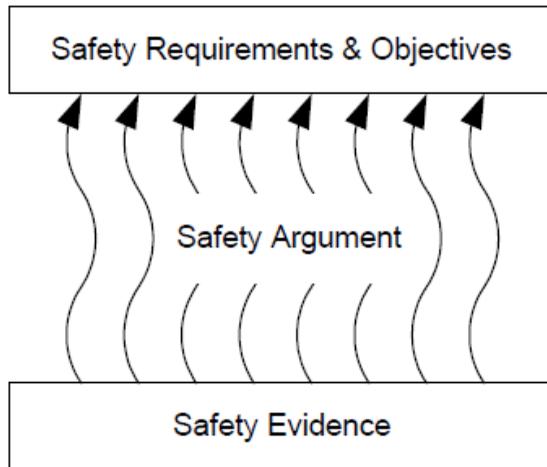


Fig. 26.2 – Objectives, arguments and evidence of safety

The Assurance Case should be developed in stages throughout the life cycle, starting from the first stage of the concept and contract (Fig. 26.3). Then, over the course of development, deviations from requirements can be quickly identified and corrected with less cost. At the same time, assessment of the implementation of both product requirements and requirements for safety and security management processes is supported.

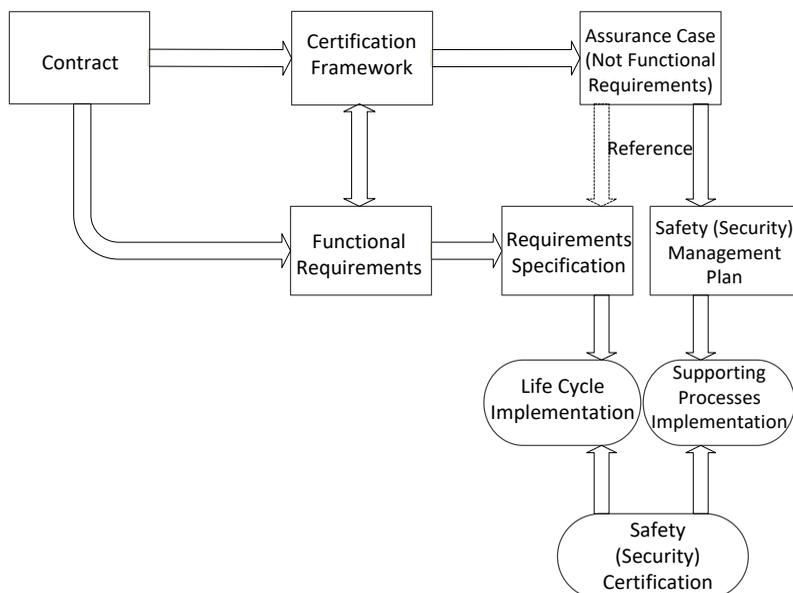


Fig. 26.3 – General approach to applying the Assurance Case in a certification project

### 26.1.2. Standards for Assurance Case

To date, regulatory documents have been developed that regulate the use of the Assurance Case in the nuclear power industry, aviation, the automotive industry, etc. The most general provisions for the application of the Assurance Case relating to system and software engineering are given in the standards of the ISO/IEC 15026 series “Systems and software engineering – Systems and software assurance” [4], which includes four parts:

- Part 1: Concepts and vocabulary;
- Part 2: Assurance case;
- Part 3: System integrity levels;
- Part 4: Assurance in the life cycle.

Object Management Group (OMG) developed Structured Assurance Case Metamodel (SACM) [5]. Goal Structured Notation Community Standard [6] is closely related with OMG SACM providing the GSN description.

ISO 26262:2011 “Road vehicles – Functional safety” standard in ten parts requires the Safety Case implementation for automotive systems. The document “Common position of international nuclear regulators and authorized technical support organizations – Licensing of safety critical software for nuclear reactors” [7] describes software Assurance Case applicability in nuclear industry recognized by such countries as Belgium, Canada, Germany, Finland, Spain, Sweden, and UK. The document “European Organization for Safety of Air Navigation (EUROCONTROL) – Safety Case Development Manual” [8] describes Safety Case applicability for European Air Traffic Management Systems.

In the U.S. two huge government organizations, such as National Aeronautics and Space Administration (NASA) and Department of Homeland Security (DHS) have already implemented Assurance Case approach for their products, services and regulatory documents. NASA established the Robust Software Engineering Group in the Intelligent Systems Division for support of Independent Verification and Validation [9] that is implemented by NASA for space programs as well as for Unmanned Aircraft Vehicles. The DHS Cyber Emergency Response Team (US-CERT) implements Assurance Case methodology to establish security assurance ecosystem. The last activity, including lecture courses providing, is widely supported in Software Engineering University, which is a part of Carnegie Mellon University [10].

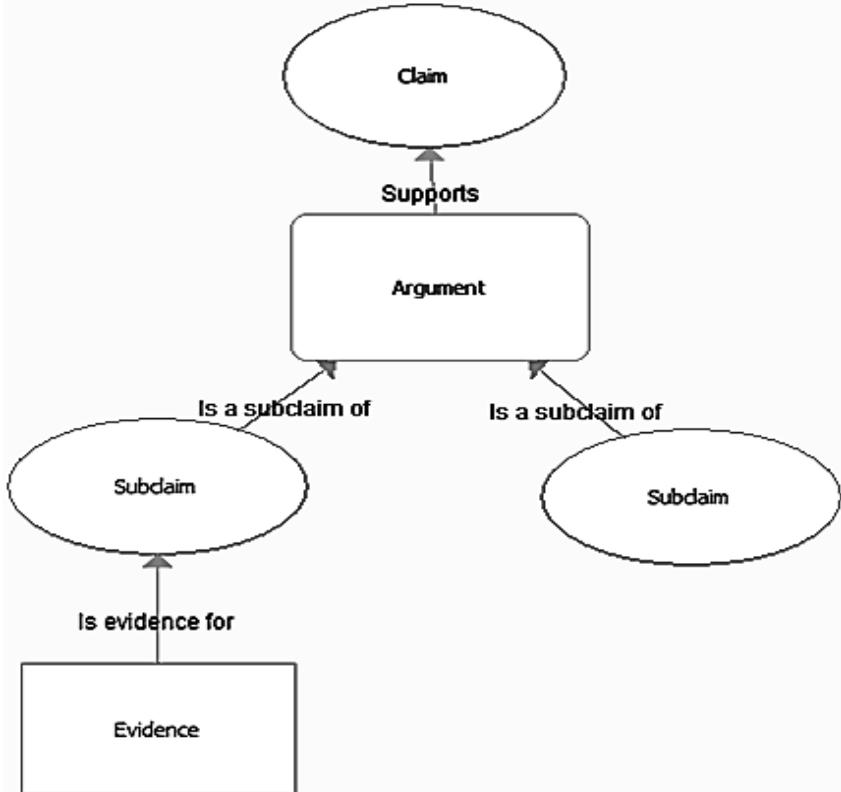
## **26.2. Safety and security techniques and measures for IoT**

### ***26.2.1. Claims, Arguments and Evidence (CAE) notation***

The CAE (Claim, Argument and Evidence) notation operates with three specified entities: claim indicates the achievement of the required system properties, evidence provides a documented basis for argumentation, demonstrating the achievement or non-achievement of goals, and arguments are built using inference rules and link evidence with objectives. Arguments such as deterministic (or logical),

probabilistic, and qualitative are commonly used. To designate claims, arguments and evidence, graphic primitives are introduced that have different shapes (Fig. 26.4).

Fig. 26.4 – Claim, Argument and Evidence (CAE) notation: main components



Building a hierarchy of goals and sub-goals is the first step in the development of the Assurance Case. As shown in the diagram (Fig. 26.4), the structure of goals, arguments and evidence is not necessarily three-level, for example, additional sub-goals can be used to support the argument.

As an example of using CAE notation, consider the general case of the formation of requirements for system functional safety [11]. The main goal is adequate, accurate and complete wording of the requirements. For this, the following subgoals must be achieved (Fig. 26.5):

- Requirements for the management of functional safety have to be defined;
- Regulatory requirements established in standards, laws and other regulatory documents have to be defined;
- Safety criteria have to be defined;
- Integration requirements have to be defined.

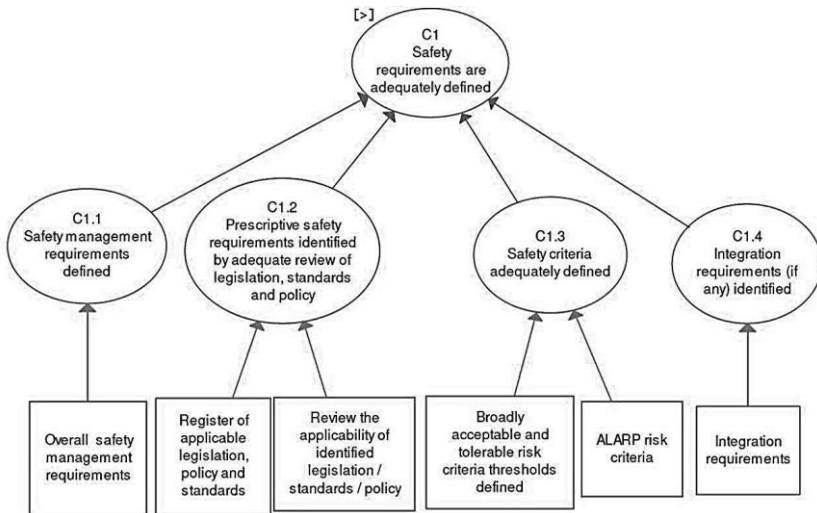


Fig. 26.5 – CAE notation: an example for functional safety

This diagram does not show the argumentation system, since this is a general case, and the argumentation strategy may be different. The requirements stated in regulatory documents, the results of risk analysis, etc. are used as evidence.

### ***26.2.2. Update and application of Claims, Arguments and Evidence (CAE) notation***

Usually CAE notation is applied in graphical view, but tabular view can also be used. Claim, Argument and Evidence should be located respectively in the fields of the table (Table 26.1). Let's consider an example from the standard IEC 61508 "Functional safety of electrical/ electronic/ programmable electronic safety-related systems" relating to personnel management (see subsection 25.1.2).

Table 26.1 – A table view of CAE notation

| <b>IEC 61508</b> | <b>Claim</b>                                                                  | <b>Argument</b>                                                                             | <b>Evidence</b> |
|------------------|-------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|-----------------|
| 1/6.2.1          | Responsibilities of the project participants                                  | <b>HR1:</b> Organizational Chart.<br><b>HR2:</b> Project Roles Description                  | –               |
| 1/6.2.3          | Understanding by the project participants of their roles and responsibilities | <b>HR6:</b> Participants and Signature List                                                 | –               |
| 1/6.2.4          | Communications of the project participants                                    | <b>HR5:</b> Participants Communications Plan                                                | –               |
| 1/6.2.13         | Evaluation and assurance of the project participants competencies             | <b>HR3:</b> Competency Matrix.<br><b>HR4:</b> Training Plans and Training Records Reference | –               |
| 1/6.2.14         | Issues affected to the project participants competencies                      | <b>HR3</b><br><b>HR4</b>                                                                    | –               |
| 1/6.2.15         | Documentation of the project participants competencies                        | <b>HR3</b><br><b>HR4</b>                                                                    | –               |

| <b>IEC<br/>61508</b> | <b>Claim</b>                              | <b>Argument</b>          | <b>Evidence</b> |
|----------------------|-------------------------------------------|--------------------------|-----------------|
| 1/6.2.16             | Monitoring of safety management processes | <b>HR1</b><br><b>HR2</b> | –               |

Table 26.1 describing CAE contains fields used according to the following purpose:

- IEC 61508 – a reference to part (before the slash "/") and clause of IEC 61508;

- Claim – a brief statement of the requirement; note that for convenience, the entire requirement can be placed in a table according to the text of the standard; in the table under consideration, only those requirements related to personnel management are selected;

- Argument – an approach to represent compliance with the requirement; several approaches can be applied to ensure compliance with the same requirement (one-to-many relationship), and the same approaches can be used for different requirements (many-to-one relationship or many-to-many relationship); if we consider the Human Resource Management Plan (Fig. 25.2), it becomes clear that its structure is determined by the arguments derived from the requirements of IEC 61508; a graphical representation of the structure of the Human Resource Management Plan confirms the effectiveness of using the graphic Mind Map notation for a simplified description of the Assurance Case; the arguments are assigned the numbered identifiers from HR1 to HR6, also according to the order of their entry into the structure of the Human Resource Management Plan (Fig. 25.2);

- Evidence – in this example, that field of the table is not populated, since the assessment of compliance with the requirements is determined for each specific project based on an audit of the developed documents and the implemented processes.

The table describing the Assurance Case may also include fields for independent evaluation by a third party and description of corrective actions. Consider, by the example of the Human Resource Management Plan, the application of the process approach to managing and evaluating the safety at all stages of the life cycle. To fulfill this task, we modify the CAE notation. A reasoning strategy may be supported

by compliance criterion and coverage criterion. Compliance criterion clarifies how compliance with requirement and claim can be achieved. Coverage criteria applies to multiple hierarchical requirements (for example, when all requirements must be verified during the testing process). Thus, CAE notation is transformed into CAEC notation (Claim, Argument, Evidence and Criteria) (Fig. 26.6).

The second component of the amended methodology is the notation describing the promotion of the Assurance Case through the stages of the life cycle. V-shaped life cycle is implemented for IoT system (see subsection 25.2), which includes phased development and phased verification and validation. Thus, the Assurance Case must be supplemented after each of the stages of development, verification and validation (Fig. 26.7).

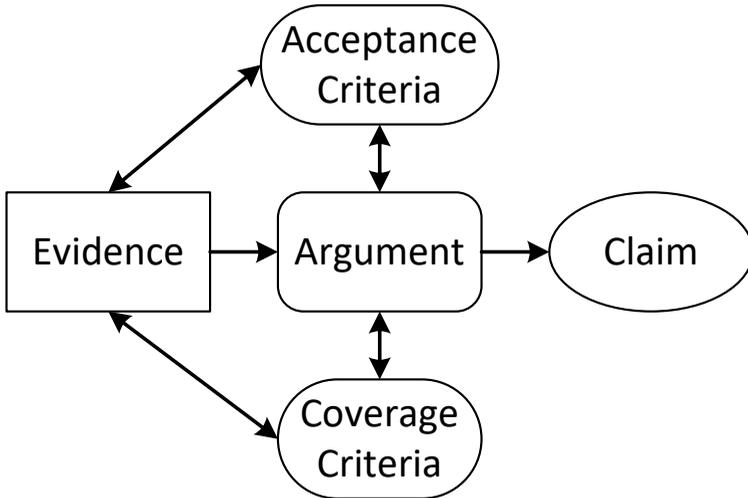


Fig. 26.6 – Claim, Argument, Evidence and Criteria (CAEC) notation: main components

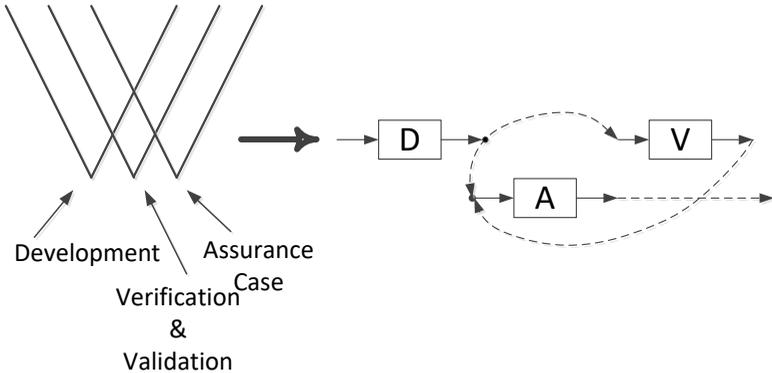


Fig. 26.7 – The relationship between the components of the life cycle (development, verification and validation, Assurance Case)

This approach is described in the form of DVA notation (Fig. 26.8), what means Development, Verification & Validation, and Assurance Case.

The DVA notation includes the following data sets transmitted between components:

- $D_I = \{d_{i1}, d_{i2}, \dots, d_{iK}\}$  – input development process data transmitted from the previous stage of the life cycle;
- $V_I(D) = \{v_{id1}, v_{id2}, \dots, v_{idL}\}$  – the input data of the verification and validation process transmitted from the development process;

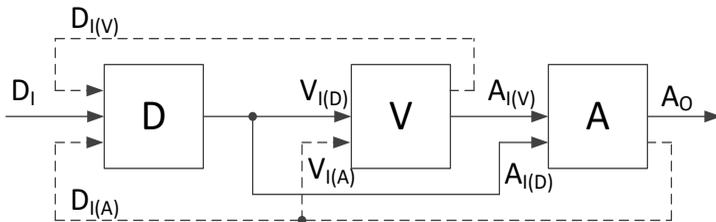


Fig. 26.8 – The relationship between the components of the life cycle (development, verification and validation, Assurance Case)

- $A_{I(D)} = \{a_{id1}, a_{id2}, \dots, a_{idM}\}$  – input data of the Assurance Case process, transmitted from the development process;
- $A_{I(V)} = \{a_{iv1}, a_{iv2}, \dots, a_{ivN}\}$  – input data of the Assurance Case process, transmitted from the verification and validation process;

- $D_{I(V)} = \{d_{iv1}, d_{iv2}, \dots, d_{ivP}\}$  – input development process data transmitted from the verification and validation process (feedback);
- $D_{I(A)} = \{d_{ia1}, d_{ia2}, \dots, d_{iaQ}\}$  – input development process data transmitted from the Assurance Case process (feedback);
- $V_{I(A)} = \{v_{ia1}, v_{ia2}, \dots, v_{iaR}\}$  – input data of the verification and validation process transmitted from the Assurance Case process (feedback);
- $A_O = \{a_{o1}, a_{o2}, \dots, a_{oS}\}$  – output data of the Assurance Case process (this is also output data of the life cycle stage), transmitted to the input of the next life cycle stage after resolution of all findings and anomalies.

The application of the considered CAEC and DVA notations constitutes an approach called Assurance Case Driven Design [12]. The goal of the approach is to reduce certification costs by consistently preparing and correcting the Assurance Case, starting from the very first stages of the life cycle. Thus, the Assurance Case supports and guides the development, verification and validation process.

From the point of view of life cycle organization, the application of the Assurance Case methodology should be coordinated during development, quality assurance, safety and safety assurance, as well as during assessment and certification, like DevOps (development and operation) methodology (Fig. 26.9).

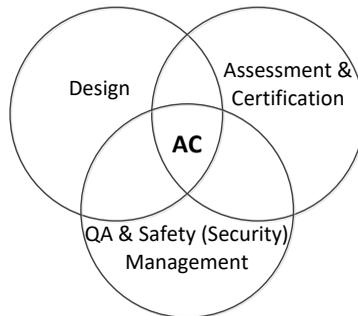


Fig. 26.9 – The diagram of components interaction for development of the Assurance Case

Let's consider applying the Assurance Case methodology throughout the life cycle stages. To do this, we use the example of assessing the compliance of the Human Resource Management Plan

with the requirements of IEC 61508 (Table 26.1). Below is a list of the stages of the Safety and Security Life Cycle, including development, verification and validation (Table 26.2).

At each stage, the compliance of the human resource management process with each of the requirements of the Human Resource Management Plan should be verified.

The use of the Assurance Case methodology allows determination of compliance with the requirements at the argument level {HR1, ..., HR6}. Records of compliance checking and the associated results are phased into the documented Assurance Case.

Table 26.2 – The Assurance Case Driven Design application through Safety and Security Life Cycle (Fig. 25.9)

| <b>SSLC stage</b> | <b>ID</b> | <b>HR1</b> | <b>HR2</b> | <b>...</b> | <b>HR6</b> |
|-------------------|-----------|------------|------------|------------|------------|
| Concept           | D1        | A(D1,HR1)  | A(D1,HR2)  | ...        | A(D1,HR6)  |
| SRS               | D2        | A(D2,HR1)  | A(D2,HR2)  | ...        | A(D2,HR6)  |
| SRS Review        | V2        | A(V2,HR1)  | A(V2,HR2)  | ...        | A(V2,HR6)  |
| SAD               | D3        | A(D3,HR1)  | A(D3,HR2)  | ...        | A(D3,HR6)  |
| SAD Review        | V3        | A(V3,HR1)  | A(V3,HR2)  | ...        | A(V3,HR6)  |
| HW Design         | D4        | A(D4,HR1)  | A(D4,HR2)  | ...        | A(D4,HR6)  |
| HW Design Review  | V4        | A(V4,HR1)  | A(V4,HR2)  | ...        | A(V4,HR6)  |
| FMECA             | V5        | A(V5,HR1)  | A(V5,HR2)  | ...        | A(V5,HR6)  |
| SW Design         | D5        | A(D5,HR1)  | A(D5,HR2)  | ...        | A(D5,HR6)  |
| SW Design Review  | V6        | A(V6,HR1)  | A(V6,HR2)  | ...        | A(V6,HR6)  |
| SW Coding         | D6        | A(D6,HR1)  | A(D6,HR2)  | ...        | A(D6,HR6)  |

| SSLC stage               | ID  | HR1        | HR2        | ... | HR6        |
|--------------------------|-----|------------|------------|-----|------------|
| Code Analysis and Review | V7  | A(V7,HR1)  | A(V7,HR2)  | ... | A(V7,HR6)  |
| SW Testing               | V8  | A(V8,HR1)  | A(V8,HR2)  | ... | A(V8,HR6)  |
| Fault Insertion Testing  | V9  | A(V9,HR1)  | A(V9,HR2)  | ... | A(V9,HR6)  |
| Integration Testing      | V10 | A(V10,HR1) | A(V10,HR2) | ... | A(V10,HR6) |
| Validation Testing       | V11 | A(V11,HR1) | (V11,HR2)  | ... | A(V11,HR6) |

### 26.2.3. Goal Structuring Notation (GSN)

GSN (Goal Structuring Notation), like CAE, operates with entities such as goal (indicated by a rectangle and is analogous to a claim), argumentation strategy (indicated by a parallelogram and is analogous to argument), and a solution (indicated by a circle and is analogous to evidence) (Fig. 26.10).

The context is used for informational support of goal setting. Assumptions and justifications can be used to support argumentation. The goal structure is also hierarchical. It should be noted that the GSN is described in the GSN Community Standard [6], and Structured Assurance Case Metamodel [5] is developed by Object Management Group.

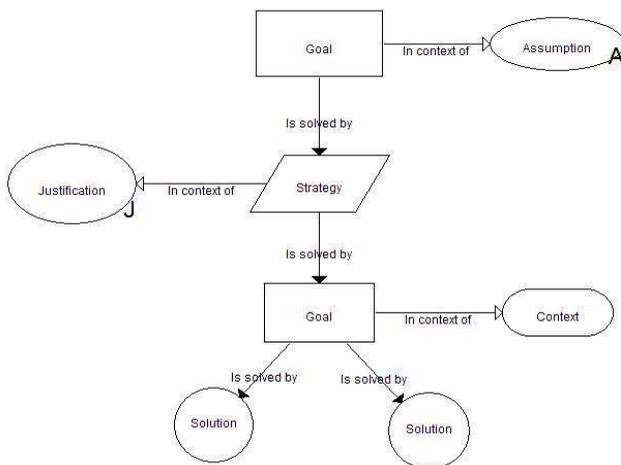


Fig. 26.10 – Goal Structuring Notation (GSN) notation:  
main components

### 26.3. Security informed and energy efficiency informed Assurance Case for IoT

#### 26.3.1 Tools for development of Assurance Case

Today, there are three of the most functional software tools that are used to create and maintain the Assurance Case. All of them have a paid license.

The first and the most widely used tool is the ASCE (Assurance and Safety Case Environment), which has been developed and maintained by the British company Adelard since the 1990s. In the UK, the development of the Assurance Case is required by laws and standards in many areas related to security, so ASCE has a fairly large market here (Fig. 26.11).

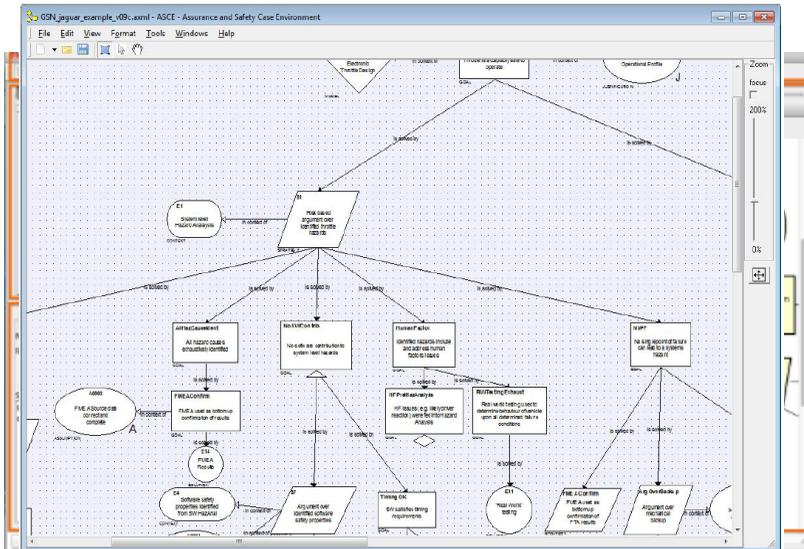


Fig. 26.11 – Adelard ASCE program interface

Adelard ASCE supports both CAE and GSN. The main part of the tool is a graphic editor, in which additional text or hyperlink information may be attached to graphic blocks. The program supports the export of charts in HTML and MS Word formats. It is impossible to download the ASCE software from the Adelard website on your own; you must fill out a request for either a 30-day trial version or an academic license, after which the request will be reviewed by the company.

The next software tool is Astah GSN (Fig. 26.12) developed by Change Vision company from Japan. The company was created in 2006. Astah GSN was developed as a part of the Astah Professional toolkit, which is a media for complex systems modeling.

As the name suggests, this program supports only GSN. In addition, it can create Mind Map diagrams. In the graphical editor, you can attach text and hyperlinks to graphic symbols. Charts are saved in the internal format of the program (\*.agml). It supports the export of diagrams in the form of figures, as well as in the XMI format (XML Metadata Interchange).

Fig. 26.12 – Astah GSN program interface

You can download a trial version of the software from the Astah GSN website. Supported operating systems are Windows, MacOS, and Linux. The trial version will work 50 days. User manuals and video demonstrations are also available on the site.

The software tool NOR-STA (Fig. 26.13) was developed by the Polish company Argevide, which was founded by the staff of the University of Gdansk. NOR-STA supports its own TRUST-IT notation (Fig. 26.14), which complies with the provisions of the standard ISO/IEC 15026. The difference is that, instead of a graphical representation, the NOR-STA uses a structured hierarchical list. Entities in hierarchical Assurance Case list are indicated by different icons. To confirm compliance with the claim, the argumentation strategy is used, and facts or observations, rationale, assumptions and sub-claims are used as analogue of the evidence.

Unlike the two previous desktop applications, NOR-STA is used online and supports distributed team work. For privacy purposes, you can install NOR-STA on a dedicated server, and then the data repository will be stored on it.

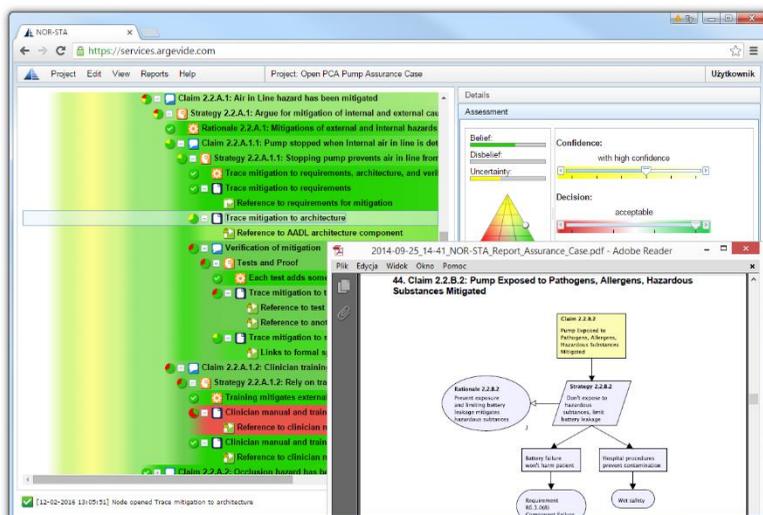


Fig. 26.13 – NOR-STA program interface

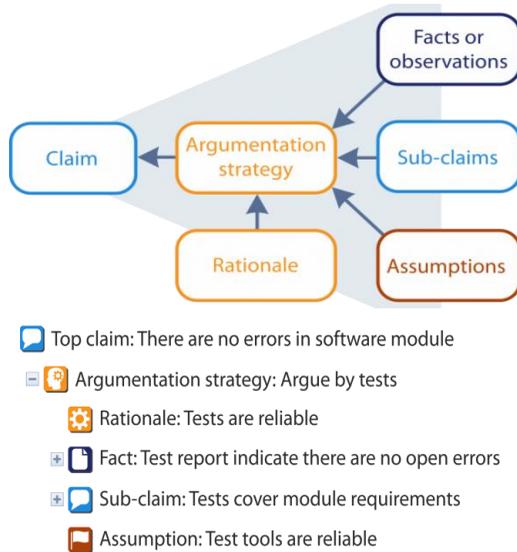


Fig. 26.14 – Trust-IT notation and an example of its application

In the considered example (Fig. 26.1), the main goal is to demonstrate the absence of errors in the software module. To this end, testing has been chosen as the argumentation strategy. The rationale for the strategy is the development and execution of reliable tests. The actual confirmation of compliance is that the test report does not contain unresolved errors. An additional sub-goal is to cover all the requirements for the software module with tests. An own argumentation strategy can be developed for this purpose. As an assumption we assume that the testing tools used are reliable.

Data can be presented as a GSN diagram, and you can also convert to Word, Excel, PDF, and XML formats. At the request of the user, a 30-day trial access can be provided on the NOR-STA website for using this software.

### 26.3.2. Assurance Case structure for IoT systems

Fig. 26.15 represents “a big picture” for IoT Green Assurance Case by joining all elements of assessment. Firstly, Green ITs are directed to support sustainable development. For that sustainability assurance part is included in the Assurance Case to check an influence to resources, ecology, society, and economy. Secondly, the main issues related with safety and security

requirements assurance and assessment shall be incorporated to the Assurance Case. After that it makes a sense to submit above six Green IT principles adopted for IoT.

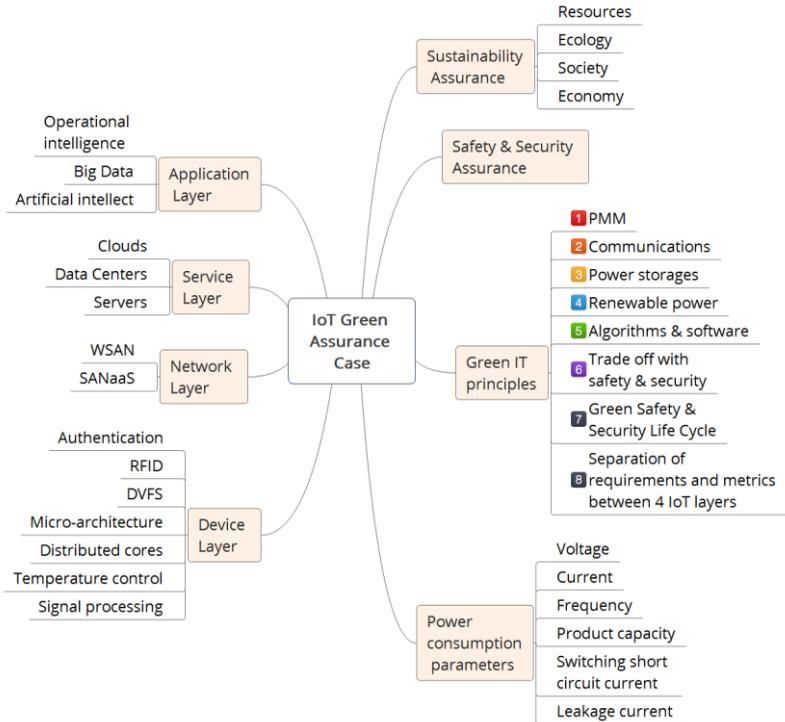


Fig. 26.15 – IoT Green Assurance Case

Power consumption parameters are also included as a part of the Assurance Case since they provide a well-defined part of requirements to IoT-based system. For example, processor power consumption can be calculated in accordance with the following equation:

$$P = A \cdot C \cdot V^2 \cdot f + A \cdot \tau \cdot V \cdot I_{\text{short}} \cdot f + V \cdot I_{\text{leakage}},$$

where the first item measures the dynamic power consumption caused by the charge and discharge of the capacitive load at the output of each key, which is equal to the product of the capacity – C, the square of supply

voltage –  $V$ , the processor's frequency –  $f$ , and the coefficient  $A$ , which characterizes the activity of the keys in the system; the second item is the power expended as a result of short-circuit current, which takes place at the time of switching the logical element; the third item are losses due to current leakage.

The main specific part of Assurance Case includes specific requirement to green features that should be implemented for each of the four of IoT layers.

Device layer includes mainly field sensors and actuators as well as programmable controllers in a form of on-board computers. Device layer is relatively simple and well defined, so for it we can describe common green features with higher degree of certainty. One of the key technologies applied at the device layer is techniques of identification and authentication. For example, Radio Frequency Identification (RFID) is widely used at the present in IoT systems. It is important to specify device identification functions to meet Green IT principles.

After that Power Modes Management (PMM) with power-aware scheduling based techniques shall be implemented at the device layer. For example, Dynamic Voltage and Frequency Scaling (DVFS) is widely used for embedded systems to alter the voltage and/or frequency of a programmable component based on performance and power requirements.

Micro-architecture solutions support saving energy in specific components with dynamical reconfiguration. For example, there are different techniques for buffering, memory compression, memory size adjusting, cache providing for simultaneous reading / writing access etc. Use of distributed cores for calculation allows to manage multitask environment and assign task to alternative programmable components, such as Digital Signal Processors (DSP), Field Programmable Gates Arrays (FPGA) and other. Tasks are assigned depending which component is more appropriate from the point of view of energy consumption. Temperature control techniques are used for devices since a temperature mode affects longevity of components operation.

Finally, signals processing techniques are widely used at the device layer, so power efficiency of signals processing algorithms and software can noticeably decrease power consumption.

Wireless Sensor Networks (WSN) with different protocols and topologies should be based on advanced communication technology

such as, for example, cognitive radio with autocorrecting of power efficiency parameters or Multiple Input Multiple Output (MIMO) with enforcement the capacity of communication channel. Since the same sensors and networks can be used for different application, service providers operate global and local WSN infrastructure which can be considered as “Sensor Network as a Service” (SNaaS).

There are a lot of features to be implemented for green clouds which serve as a platform for the service layer. Modern green data centers and green servers are based on techniques using PMM, advanced communications, and advanced power storages.

Successful operation of application layer depends on implementation of power efficient algorithms in software. From this prospective a business can be supported with operational intelligence based on advanced technique of big data and artificial intellect.

#### **26.4 Work related analysis**

Existing standards in the area of the Assurance Case as well as other important publications are considered in 26.1.2. In the present subsection in addition to the many stories of successful use of the Assurance Case, we discuss stories when the unsuccessful application without proper analysis did not allow identifying problems and led to accidents. As with any methodology, the mere fact of applying the Assurance Case or any other security measures is not sufficient to ensure that the hazards are eliminated and the risks are reduced. At the same time, although the basic goals and philosophy of the Assurance Case are fairly well defined, there is a limited understanding of how best to put this methodology into practice.

Similar issues were discussed earlier by different researchers [13-15]. The main problems, rather than those related to the Assurance Case methodology, but to the general issues of security assurance and evaluation, are as follows [14]:

– The “Apologetic Assurance Case”: the Assurance Cases which avoid uncomfortable truths about the safety and security of systems in production so that developers do not have to face the (often economically and politically unacceptable) option of re-design (“X doesn’t quite work as intended, but it’s OK because...”);

– The Document-Centric View: the Assurance Cases which have as their aim to produce a document. The goal of the Assurance Cases should not simply be the production of a document; it should be to produce a compelling argument;

– The Approximation to the Truth: the Assurance Cases which ignore some of the rough edges that exist. For example, the Assurance Cases which claims in a Goal Structured Notation diagram that “All identified hazards have been acceptably mitigated” and direct the reader to the Hazard Log when, in reality, the mitigation argument is not so straightforward;

– The prescriptive Assurance Cases: the Assurance Cases which have become run-of-the-mill or routine or simply comprise a parade of detail that may seem superficially compelling but fails to amount to a compelling argument;

– The Assurance Case Shelf-Ware: the Assurance Cases which are consigned to a shelf, never again to be touched. The Assurance Case has failed in its purpose if it is “so inaccessible or unapproachable that we are happy never to refer to it again”;

– Imbalance of skills: The skills are required of both someone to develop the Assurance Case and someone to challenge and critique the assumptions made. Too often, the latter skills are missing.

Based on the analyzed problems in evaluating the safety of complex systems, an approach to their solution was proposed [13]. This approach is formulated as a system of principles SHAPED, which stands for short (“Succinct”), carried out under the control of the operating organization (“Home-grown”), “Accessible” to all interested parties, “Proportionate” in terms of focusing on the main dangers and risks, “Easy-to-understand”, and “Document-lite”.

Thus, the Assurance Case is one of the integral tools for evaluating and ensuring safety and security, and the effectiveness of its application depends on the competencies of the specialists involved, the organization of the process and the correct application of the recommended principles.

### *Conclusions and questions*

The Assurance Case is a structured set of arguments and documentary evidence that justify the compliance of a system or service with specified requirements. Thus, the Assurance Case is an

integral methodology for evaluating safety and security. That allows building a clear structure of the products and processes artifacts throughout the entire life cycle. Licensing and certification authorities will check the Assurance Case, as an integral document proving compliance with the entire set of requirements to safety and security.

The predecessor to the Assurance Case has historically been the Safety Case. Regulatory documents requiring the use of the Safety Case for hazardous industrial facilities appeared in the 1980s. With the development of information technology and the emergence of cyber threats, the Security Case began to be developed. As of today, the Assurance Case usually means the justification of safety together with security.

The Assurance Case should be developed by stages throughout the life cycle, starting from the stage of concept and contract. Then, over the stages of development, deviations from safety and security requirements can be promptly identified and corrected at lower cost. At the same time, assessment of the implementation of both requirements to product processes is supported.

For graphical representation of the Assurance Case, semi-formal notations such as Claim, Argument and Evidence (CAE) and Goal Structuring Notation (GSN) are used. In addition, the Assurance Case may be represented in a table view.

In order to better understand and assimilate the educational material that is presented in this section, we invite you to answer the following questions.

1. Define the Assurance Case methodology.
2. What is the history of the development of the Assurance Case methodology?
3. What types of notations can be used to represent the Assurance Case?
4. What standards and other regulatory documents govern the application of the Assurance Case?
5. Give a description of the CAE notation.
6. What are additional criteria for CAE notation?
7. Define the Assurance Case Driven Design approach?
8. Give a description of the GSN notation.
9. What regulatory documents govern the GSN notation?

10. What software tools are used to support the Assurance Case methodology?
11. Which organizations are the most active in promoting the Assurance Case methodology?
12. What are the main disadvantages and advantages of applying the Assurance Case methodology?
13. What role does the human factor play in applying the Assurance Case methodology?
14. What does the SHAPED system of principles mean for the Assurance Case methodology?
15. What the basic structure of the Assurance Case for IoT systems?

### ***References***

1. Kelly T. *Arguing Safety: A Systematic Approach to Managing Safety Cases*. PhD thesis. Univ. of York, 1998.
2. Toulmin S. *The Uses of Argument*. Cambridge University Press, 1958.
3. Cullen W. *The Public Enquiry into the Piper Alpha Disaster*. Department of Energy, London, HM Stationery Office, 1990.
4. ISO/IEC 15026, *Systems and software engineering – Systems and software assurance (in 4 parts)*, 2011-2015.
5. *Structured Assurance Case Metamodel, v2.0*. Object Management Group, 2016.
6. *GSN Community Standard, Version 1*. Origin Consulting (York) Limited, 2011.
7. *Common position of international nuclear regulators and authorised technical support organisations – Licensing of safety critical software for nuclear reactors*, 2015.
8. *Safety Case Development Manual*. European Organization for Safety of Air Navigation (EUROCONTROL), 2006.
9. Denney E., Pai G. *Safety Case Patterns: Theory and Applications*. Research report NASA/TM–2015–218492. NASA, 2015.
10. Weinstock C., Goodenough J. *Towards an Assurance Case Practice for Medical Devices*, Technical Note CMU/SEI-2009-TN-018. SEI, 2009.
11. Ye F., Cleland G. *Weapons Operating Centre Approved Code of Practice for Electronic Safety Cases*. Adelard LLP, 2012.

12.Sklyar V., Kharchenko V. Green Assurance Case: Applications for Internet of Things. Green IT Engineering: Social, Business and Industrial Applications. Studies in Systems, Decision and Control, vol 171. Springer, Cham, 2019.

13.Haddon-Cave C. The Nimrod Review. An independent review into the broader issues surrounding the loss of the RAF Nimrod MR2 Aircraft XV230 in Afghanistan in 2006. Crown Copyright, 2009.

14.Kelly T. Are Safety Cases Working? Safety Critical Systems Club Newsletter, Vol. 17, n. 2, 2008.

15.Скляр В.В. Обеспечение безопасности АСУТП в соответствии с современными стандартами. – Инфра-Инженерия, 2018.

## **27. SECURITY OF IOT BASED BLOCKCHAIN TECHNOLOGY**

DrS. Prof. V. V. Yatskiv, Ass. Prof., Dr. N. G. Yatskiv (TNEU)

### ***Contents***

|                                                                    |     |
|--------------------------------------------------------------------|-----|
| Abbreviations .....                                                | 369 |
| 27.1. Bases of blockchain technology and examples of application . | 370 |
| 27.1.1 The principle of the blockchain technology .....            | 370 |
| 27.1.2 Block structure and Merkle tree .....                       | 372 |
| 27.1.3 Blockchain cryptography.....                                | 375 |
| 27.2 Consensus algorithms in blockchain technology.....            | 377 |
| 27.2.1 Proof of Work algorithm .....                               | 378 |
| 27.2.2 Proof of Stake algorithm.....                               | 381 |
| 27.3 Blockchain technology for the IoT security .....              | 384 |
| 27.3.1 Blockchain and the IoT.....                                 | 384 |
| 27.3.2 Benefits of Integrating Blockchain with IoT.....            | 388 |
| 27.3.3 Main challenges of blockchain in IoT .....                  | 390 |
| 27.4 Work related analysis .....                                   | 394 |
| Conclusions and questions.....                                     | 397 |
| References .....                                                   | 399 |

## **Abbreviations**

PoW – Proof of Work

PoS – Proof of Stake

DPoS – Delegated Proof of Stake

LPoS – Leased Proof of Stake

PoI – Proof of Importance

dBFT – Delegated Byzantine Fault Tolerance

PoC – Proof of Capacity

PoA – Proof of Activity

PoB – Proof of Burn

PoET – Proof of Elapsed Time

ADEPT – Autonomous Decentralized Peer-to-Peer Telemetry

GUID – Global Unique Identifier

ECDSA – Elliptic Curve Digital Signature Algorithm

IANA – Internet Assigned Numbers Authority

## **27.1. Bases of blockchain technology and examples of application**

### ***27.1.1 The principle of the blockchain technology***

In 2008, the author or group of authors known under the pseudonym Satoshi Nakamoto published the paper "Bitcoin: A Peer-to-Peer Electronic Cash System" with a description of the concept and principles of the payment system as a peer-to-peer network [22]. In 2009, the Bitcoin cryptocurrency protocol was submitted and the client application was published. The key feature of the proposed concept was that online payments between customers are carried out without a central financial institution that serves as a trusted entity, using cryptographic methods and a public financial transaction database (distributed ledger), which consists of a chain of blocks (Blockchain) [4].

Blockchain is a distributed data structure which consists of the blocks sequence, each block typically contains a hash pointer as a link to a previous block, thus forming a chain of blocks (Fig. 27.1).

Blockchain is a new information technology that has wide variety of uses in many industries. The first and most famous example of the use of the blockchain technology is the cryptocurrency Bitcoin [22]. Currently, cryptocurrency has become a recognized means of payment, a virtual currency that is accepted by large and small enterprises, corporations and services.

The first block in the chain (parent block, genesis block) is considered as a separate case, since it does not have a previous block (Fig. 27.1). Blockchain works as a distributed database that records all transactions on the network. Operations have a timestamp and are stored in blocks where each block is identified by its cryptographic hash.

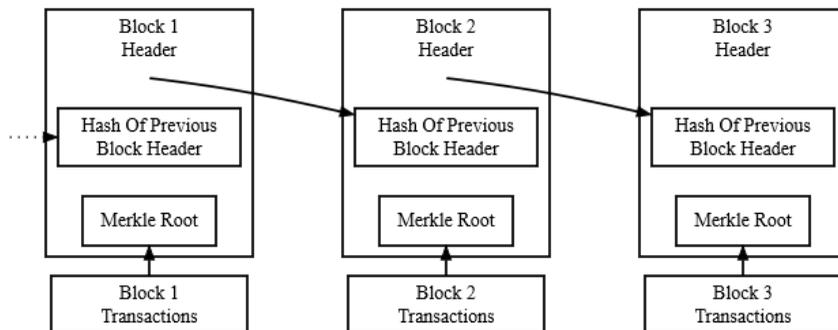


Fig.27.1 – Simplified sequence of blocks

Blockchain is completely stored in each network node. Blockchain does not require trust between the nodes of the network, since any node can independently check whether its database copy coincides with stored copies in other nodes. Let us consider blockchain technology principle on example of Bitcoin. Cryptocurrency Bitcoin uses the cryptographic hash function SHA-256 [4]. To verify the data integrity in the block Merkle tree is used, which represents a special data structure that contains information about performed transactions.

For this, a hash is calculated from each transaction, and then the new hash of pair is calculated from each pair of hashes. This procedure is repeated until only one hash remains. If the pair for the hash is absent, then it is transferred to a new level without changes (Fig. 27.2).

The blockchain technology, like the Internet, has built-in error tolerance. While storing information blocks that are identical throughout the network, blockchain:

- is not able to be controlled by one person;
- does not have a single point of failure.

There are two types of blockchain [12, 21]:

- public blockchain is an open, supplementary database. This type of block is used in the Bitcoin cryptocurrency. Each participant can record and read data;

- private blockchain has a record / read data limitation. Only specific, pre-chosen entities have the ability to create new transactions on the chain.

Among the features of blockchain it should be emphasized:

1) decentralization - there is no server in the chain. Each participant is a server. It supports the work of the blockchain;

2) reliability - a blockchain nodes consensus is required to record new data. This allows to filter operations and record only legitimate transactions. It is impossible to change the hash.

3) transparency - information about transactions, contracts, and so on is stored in open access. However, this data cannot be changed;

4) theoretical unlimited - theoretically, blockchain can be supplemented by records to infinity. Therefore, it is often compared to a supercomputer;

5) universality - blockchain technology can be applied not only in the financial area, but it can be integrated into multiple areas (personality authentication, jurisprudence, real estate, Internet of Things, etc.).

### ***27.1.2 Block structure and Merkle tree***

The transaction group is recorded after the check in a special block (Fig. 27.2). The block consists of a header and a list of transactions (TrA, TrB, ...). The block header includes the block hash, previous block hash (Previous Hash), transaction hash (Merkle Root), and additional service information (Nonce, Timestamp) [22, 27].

Timestamp indicates when the block was created and provides evidence that the data in the block existed at a specific time.

The following data is required to create a new block: hash of the previous block in the chain; Merkle hash for transactions that must be included in a block; time (Timestamp) and disposable code (Nonce), selected in a pseudo-random manner. It is necessary to calculate the hash of the header of the new block, which must begin with a given number of zeros to confirm the correctness of the block. This task is known as proof of work (proof of work), based on two principles: 1) to make transaction confirmation costly in the form of computer calculations to users of the network; 2) receive reward for transaction verification.

The new block is accepted by other network nodes if the header hash value is equal to or less than a given number, value of this number periodically varies. When the result is computed, the generated block is sent to other nodes that check it. If the check is successful, then the block is added to the chain and the next block must include its hash.

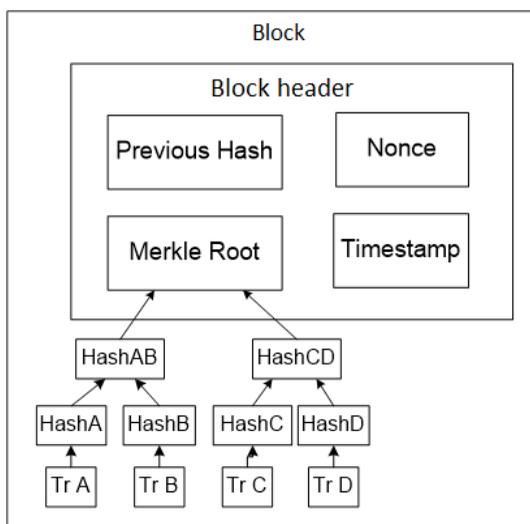


Fig. 27.2 – Structure of a block

The work, that nodes need to perform to create a new block requires a lot of time and computational resources. This reduces the probability that two blocks can be produced at the same time, but this situation is still possible. When this happens, a temporary fork in blockchain is created. In this case, nodes can build a chain on different branches. To prevent this situation, each node tracks all branches, but nodes will try to expand only the longest branch. In this case, the length is determined not by the number of blocks, but by the total volume of work spent on the creation of a branch, and is determined by the number of zeros at the beginning of the block hash.

The computational complexity of transaction verification allows to avoid dependence on the number of network nodes controlled by attacker. Thus, only the total computational power of the nodes affects the verification. Therefore, an attacker requires significant computational resources to modify information in a block or to create an incorrect block, which makes it virtually impractical.

Since blockchain copies are stored in distributed network nodes, blockchain technology is resistant to problems with temporary or

permanent disabling of nodes due to hardware or communications failure, and also the connection of new nodes. The more nodes are in the network, the more reliable the storage in blockchain. Blockchain does not have a single point of failure, unlike a centralized system with a single server, which ensures high reliability of data storage.

Merkle Tree is a data structure, also known as binary tree of hash lists. In the case of the Bitcoin, the Merkle tree is constructed as follows (Fig. 27.3) [4].

We start with three transactions, A, B, C, which form the leaves of the Merkle tree, as shown in Figure 27.3.

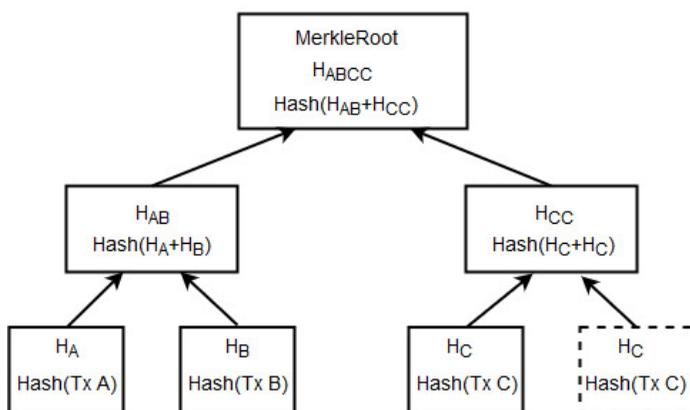


Fig.27.3 – Merkle tree

1. The transactions are not stored in the Merkle tree; rather, their data is hashed and the resulting hash is stored in each leaf node as  $H_A$ ,  $H_B$ ,  $H_C$ :  $H_{\sim A \sim} = \text{SHA256}(\text{SHA256}(\text{Transaction A}))$ .

2. Consecutive pairs of leaf nodes are then summarized in a parent node, by concatenating the two hashes and hashing them together.

That string is then double-hashed to produce the parent node's hash:  $H_{\sim AB \sim} = \text{SHA256}(\text{SHA256}(H_{\sim A \sim} + H_{\sim B \sim}))$ .

3. Because the Merkle tree is a binary tree, it needs an even number of leaf nodes. If there is an odd number of transactions to summarize, the last transaction hash will be duplicated to create an even number of leaf nodes. In our example transaction C is duplicated.

4. The process continues until there is only one node at the top, the node known as the Merkle root.

Using the Merkle tree in Blockchain allows to provide "genuineness" of transactions in a block. If you change at least one transaction, then Merkle root will also change. Therefore, the following situation is impossible. For example, miner produced a new block and started sending it over the network. At this time, the attacker intercepts the block and removes some transaction from the block, and then sends the already changed block.

It is enough to calculate the Merkle root and compare it with what is written in the header block to check block integrity.

### ***27.1.3 Blockchain cryptography***

Cryptography is the core of the blockchain, which provides reliable work of the system. The blockchain architecture suggests that the trust among the network participants based on the mathematics principles, so, it is formalized. Cryptography also guarantees security, based on transparency and the ability to verify all transactions, rather than limitation of the system visibility (perimeter security) [4].

Blockchain technology uses cryptography as a means of user protection, ensuring transaction security and protecting all information.

Various cryptographic technologies can guarantee the invariability of the blockchain transactions log file, resolve authentication tasks, and control access to the network and data in the blockchain.

Cryptography in blockchain is based on three components: hash functions, asymmetric encryption and digital signature.

Hashing is the process of converting the input data of any length into a source string of fixed length. For example, a hash function can get a string with any number of characters (one letter or a whole story), and provide a string with a strictly defined number of characters as an output.

A reliable hash function provides collision protection (it is impossible to get two identical hash values from different input data) and has the so-called avalanche effect, when the slight change in input data completely changes the output.

Public key cryptography. Public key cryptography or asymmetric cryptography allows to share information using a public key that you can share with anyone.

Instead of using one key, separate keys are used for encryption and decryption (public key and private key) [17].

A combination of user's public and private keys is used to encrypt the information, while the recipient's private key and the sender's public key decrypt it. It is impossible to determine which private key is based on the public key. Thus, user can send his public key to anyone, without worrying that someone will have access to his private key. The sender can encrypt files, and be confident that they will be decrypted only by a specified party.

In addition, a digital signature which ensures the data integrity is created while using public key cryptography. This is achieved by combining a user's private key with the data that he intends to sign, using a mathematical algorithm.

Since the actual data is part of the digital signature, the network does not recognize them as valid, if any part of it is fake. Editing even the small piece of data changes the shape of the entire signature, making it wrong or obsolete. Due to this, blockchain technology can guarantee that any recorded data is true, accurate and unchanged. Digital signatures provide immutability for the data recorded in blockchain.

**Digital signature.** The digital signature provides authentication and authentication in the same way as conventional signatures, only in digital form.

Digital signatures are one of the key factors in ensuring the security and integrity of data recorded in blockchain. They are the standard parts of the most blockchain protocols, mainly used for transactions and transaction blocks protection, transfer of confidential information, software distribution, contract management, and many other cases where it is important to detect and prevent any external interference. Digital signatures use asymmetric cryptography-this means that information can be shared with anyone using a public key.

Digital signatures have three key advantages while using it for information storage and transfer in blockchain. First of all, they guarantee the integrity. Theoretically encrypted data that is transmitted may be imperceptibly changed by a hacker. However, if this happens, the signature will be also changed and become incorrect. Therefore, data with a digital signature is not only protected from viewing, but also allows you to find out whether it was faked.

While using the blockchain technology, user has public and private keys, both of them are strings of random numbers and letters. The public key, also called public address, and it can be compared to an email address. The private key must be written and stored in a secure place. Ideally, it can be a piece of paper or a hardware wallet, because it's almost impossible to hack them.

There is no option "I forgot my private key" in Blockchain. If a private key is lost, then everything encrypted with that key will be lost.

Digital signatures are unique and they are created using the following three algorithms [4]:

- an algorithm for generating keys, which provides private and public keys;
- a signature algorithm that combines data and a private key to create a signature;
- an algorithm that verifies the signatures and determines whether a message is genuine or not, based on the message, public key and signature.

The key features of these algorithms are:

- it is absolutely impossible to work with a private key based on the public key or data that it has encrypted;
- it ensures the authenticity of the signature on the basis of the message and the private key which is verified using the public key.

## **27.2 Consensus algorithms in blockchain technology**

Consensus is the decision-making process of group in which all group members agree to support the decision in the interests of the whole. The voting is deciding according to the majority of the votes, without taking into account the interests of the minority, but on the other hand, it guarantees the achievement of an agreement that benefits the entire group.

The decision-making method is called the "mechanism of consensus". The consensus mechanism aims to achieve the following objectives [21]:

- 1) agreement seeking - should bring about as much agreement from the group as possible;
- 2) collaborative - all the participants should aim to work together to achieve a result that puts the best interest of the group first;

- 3) cooperative - all the participants shouldn't put their own interests first and work as a team more than individuals;
- 4) egalitarian - every vote has equal weightage;
- 5) inclusive - as many people as possible should be involved in the consensus process;
- 6) participatory - everyone should participate in the overall process.

The consensus algorithm for blockchain represents a set of mathematical rules and functions that allow to reach an agreement between all participants and provide the network's performance. It determines the order in which the transaction blocks will be included in the chain. For instance, blockchain requires a consensus in order to avoid double spending.

Among the existing consensus algorithms, let's consider two basic types: Proof of Work and Proof of Stake. Each of them has its own peculiarities, advantages and disadvantages.

### ***27.2.1 Proof of Work*** algorithm

Proof-of-Work is one of the first and most widespread algorithms for achieving consensus and distributing rewards for an generated block among the network users.

The most commonly used functions used in proof of work systems include:

Partial hash inversions. The most popular system in Hashcash [4] uses partial hash inversions to send an e-mail. About 252 hash calculations is needed for the header of one message, it must be counted for each new message. At the same time checking of the computed code correctness is fast because of the use of a one-time SHA-1 computation with a pre-prepared timestamp.

Functions based on Merkle trees [4]. The most famous example is the Bitcoin, where multi-level hashing is used as proof of work: each block containing a hash of the previous block

There is thus no way to change the block without changing hashes in all subsequent blocks. At the same time, the chain integrity verification is limited by a one-time computation of the current and previous block hash. Hash is considered true(verified) only if the hash value of that block is less than the specific target value that determines the complexity (difficulty) of Mining. It is necessary brute force search

with the override of arbitrary values of the nonce value to find such hash [17, 20, 26].

PoW has two characteristic features: to reach a consensus, user must solve a computationally difficult problem; verification of the result is done very quickly, unlike the solution.

Proof of Work peculiarities:

– consensus solves the main problem of anonymous networks – the "Sybil attack". This is a situation when the attacker tries to surround the victim's node, that is, to gain access to all the nodes next to it. Having captured the channels of input and output information, he will be able to pass the false information to the victim. In bitcoin built on the PoW algorithm, this ability is leveled off, because the victim's node chooses other nodes randomly, eliminating the complete victim's environment;

- the proof is not transferred to other blocks, that is, it excludes the possibility of its stealing (the proof is the computation result which consumes the energy);

– proof cannot be obtained in advance. Each new block has a link to the previous block, so you can compute new proof only with the appearance of a new block;

– PoW ensures the honest distribution of the reward for a block according to the power of the computer. If the power (hash rate) is 5% of the network, then the miner creates a 5% block and receives 5% of the reward;

– real computational resources are spent on proof obtaining therefore, the miners lose the incentive to affect the nodes and transmit false information because there is a risk of losing the invested money.

The main disadvantage of the PoW consensus algorithm is the high computational cost.

High power consumption. The miners are consuming massive amounts of electricity, but their calculations are used only for the network needs, that leads to waste energy. Today there is no idea how to use of the calculations results for any other purposes.

Environmental aspect. Energy consumptions forces to increase the generation of electricity by burning a large amount of fuels, including fossil and non-renewable fuels. It enhances environmental pollution.

Pronpensity to attack "51%". If the attacker controls 51% of the network he can control the whole blockchain and perform transactions

at wish. Attacker can interfere with the transactions, canceling them and doing other manipulations, because its power is higher, and therefore, it will be accepted "his" chain, and not legal. Today, such attacks are unlikely due to the extremely high hash rate.

Let us consider the PoW algorithm as an example of Bitcoin cryptocurrency. Each block in Bitcoin consists of two parts:

- block header of key parameters, including block creation time, reference to the previous block and the Merkle tree root of the block of transactions;
- block list of transactions.

To reference a specific block, its header is hashed twice with the SHA-256 [4] function; the resulting integer value belongs to the interval  $[0; 2^{256} - 1]$ . To account for different possible implementations, we will use a generic hashing function  $\text{hash}(\cdot)$  with a variable number of arguments and range  $[0; M]$ . For example, arguments of the function can be treated as binary strings and merged together to form a single argument that can be passed to the SHA-256 hashing function

The block reference is used in the proof of work protocol; in order for a block to be considered valid, its reference must not exceed a certain threshold [19]:

$$\text{hash}(B) \leq M/D, \quad (27.1)$$

where  $D \in [1, M]$ – is the target difficulty. There is no known way to find  $B$  satisfying (27.1), other than iterating through all possible variables in the block header repeatedly. The higher the value of  $D$ , the more iterations are needed to find a valid block; the expected number of operations is exactly  $D$ .

The time period  $T(r)$ , for a miner with hardware capable of performing  $r$  operations per second to find a valid block is distributed exponentially with the rate  $r/D$  [19]:

$$P\{T(r) \leq t\} = 1 - \exp(-rt/D).$$

Consider  $n$  Bitcoin miners with hash rates  $r_1, r_2, \dots, r_n$ . The period of time to find a block  $T$  is equal to the minimum value of random variables  $T(r_i)$  assuming that the miner publishes a found block and it

reaches other miners immediately. According to the properties of the exponential distribution,  $T$  is also distributed exponentially [19]:

$$P\{T \stackrel{\text{def}}{=} \min(T_1, \dots, T_n) \leq t\} = 1 - \exp(-\frac{t}{D} \sum_{i=1}^n r_i);$$

$$P\{T = T_i\} = \frac{r_i}{\sum_{j=1}^n r_j}.$$

The last equation shows that the mining is fair: a miner with a share of mining power  $p$  has the same probability  $p$  to solve a block before other miners.

### 27.2.2 Proof of Stake algorithm

In proof of stake algorithms, inequality (27.1) is modified to depend on the user's ownership of the particular PoS protocol cryptocurrency and not on block properties. Consider a user with address  $A$  and balance  $bal(A)$ . A commonly used proof of stake algorithm uses a condition as [19]:

$$hash(hash(B_{prev}), A, t) \leq bal(A)M/D, \quad (27.2)$$

where  $B_{prev}$  – denotes the block the user is building on,  $t$  – is the current UTC timestamp.

For various reasons, some cryptocurrencies use modified versions of (27.2) which we discuss in the corresponding sections.

Unlike (27.1), the only variable that the user can change is the timestamp  $t$  in the left part of equation (27.2). The address balance is locked by the protocol; e.g., the protocol may calculate the balance based on funds that did not move for a day. Alternatively, a PoS cryptocurrency may use unspent transaction outputs as Bitcoin does; in this case, the balance is naturally locked. A proof of stake protocol puts restrictions on possible values of  $t$ . For example, if  $t$  must not differ from the UTC time on network nodes by more than an hour, then a user can attempt no more than 7200 values of  $t$ . Thus, there are no expensive computations involved in proof of stake.

Together with an address  $A$  and a timestamp  $t$  satisfying (27.2), a user must provide a proof of ownership of the address. To achieve this, the user can sign the newly minted block with his signature; in order to produce a

valid signature, one must have a private key corresponding to the address  $A$ .

The time to find a block for address  $A$ , is exponentially distributed with rate  $bal(A)/D$ . Consequently, the (27.2) implementation of proof of stake is fair: the probability to generate a valid block is equal to the ratio of user's balance of funds to the total amount of currency in circulation. The time to find a block for the entire network is distributed exponentially with rate  $\sum_a bal(a)/D$ .

Thus, if the monetary supply of the currency  $\sum_a bal(a)$  is fixed or grows at a predictable rate, the difficulty  $D$  should be known in advance [19]:

$$D = \frac{1}{T_{ex}} \sum_a bal(a),$$

with  $T_{ex}$  denoting the expected time between blocks. In practice,  $D$  needs to be adjusted based on recent blocks because not all currency owners participate in block minting.

The most well-known consensus method is proof of work (discussed in 27.2.1) which is used by bitcoin. However, due to its high computational and bandwidth requirements, it does not seem to be practical for IoT networks.

Therefore, we present other existing consensus methods and discuss the possibility of applying them to a blockchain based IoT network [17, 21, 26].

1. Delegated Proof of Stake (DPoS). DPoS is a system in which a fixed number of elected entities (called *block producers or witnesses*) are selected to create blocks in a round-robin order. Block producers are voted into power by the users of the network, who each get a number of votes proportional to the number of tokens they own on the network (their *stake*). Alternatively, voters can choose to *delegate* their stake to another voter, who will vote in the block producer election on their behalf.

2. Leased Proof of Stake (LPoS). LPoS is an advanced version of the Proof of Stake (PoS) algorithm. Traditionally in the Proof of Stake algorithm, each node holds a certain amount of cryptocurrency and is eligible to add the next block in to the blockchain. However, with Leased Proof of Stake, users are able to lease their balance to full nodes. The higher the amount that is leased, the better the chances are that the full node will be selected to produce the next block. If the node is selected, the

user will receive a percentage of the transaction fees that are collected by the node.

3. Proof of Importance (PoI). PoI is a Blockchain consensus algorithm that considers the overall productivity of users in the network. It was first used by NEM (New Economy Movement) which is a Blockchain technology company aiming to process transactions more efficiently and introduces reputation to the cryptosystem.

4. Practical Byzantine Fault Tolerance (PBFT). PBFT is an algorithm that optimizes aspects of Byzantine Fault Tolerance (in other words, protection against Byzantine faults) and has been implemented in several modern distributed computer systems, including some blockchain platforms. These blockchains typically use a combination of pBFT and other consensus mechanisms.

5. Delegated Byzantine Fault Tolerance (dBFT). Delegated Byzantine Fault Tolerance is a sophisticated algorithm meant to facilitate consensus on a blockchain. Although it is not in common use as of yet, it represents an alternative to simpler proof of stake, proof of importance and proof of work methods.

6. Proof of Capacity (PoC). POC is a consensus mechanism algorithm used in blockchains that allows the mining devices in the network to use their available hard drive space to decide the mining rights, instead of using the mining device's computing power (as in the proof of work algorithm) or the miner's stake in the cryptocurrencies (as in the proof of stake algorithm).

7. Proof of Activity (PoA). PoA is one of the many blockchain consensus algorithms used to ensure that all the transactions occurring on the blockchain are genuine and all users arrive at a consensus on the precise status of the public ledger. Proof of activity is a mixed approach that marries the other two commonly used algorithms – namely, proof of work (POW) and proof of stake (POS).

8. Proof of Burn (PoB). Proof of burn is one of the several consensus mechanism algorithms implemented by a blockchain network to ensure that all participating nodes come to an agreement about the true and valid state of the blockchain network thereby avoiding any possibility of cryptocurrency double spending. Proof of burn follows the principle of “burning” or “destroying” the coins held by the miners that grant them mining rights.

9. Proof of Elapsed Time (PoET). POET is a blockchain network consensus mechanism algorithm that prevents high resource utilization and high energy consumption, and keeps the process more efficient by following a fair lottery system. In recent times, leading microchip manufacturer, Intel has been working on its proprietary consensus protocol. The new standard is an integral component of the Hyperledger Sawtooth blockchain framework and is used to provide enclave in Intel's Software Guard Extensions (SGX). Table 27.1 summarizes the comparison between various consensus algorithms [17].

Table 27.1 – Consensus Protocol Comparison.

|                             | PoW             | PoS           | PoET           | BFT and Variants | Federated BFT   |
|-----------------------------|-----------------|---------------|----------------|------------------|-----------------|
| Blockchain type             | Permissi-onless | Both          | Both           | Permissi-oned    | Permissi-onless |
| Transaction finality        | Probabilistic   | Probabilistic | Probabi-listic | Immediate        | Immediate       |
| Transaction rate            | Low             | High          | Medium         | High             | High            |
| Token needed?               | Yes             | Yes           | No             | No               | No              |
| Cost of participation       | Yes             | Yes           | No             | No               | No              |
| Scalability of peer network | High            | High          | High           | Low              | High            |
| Trust model                 | Untrusted       | Untrusted     | Untrusted      | Semi-trusted     | Semi-trusted    |

## 27.3 Blockchain technology for the IoT security

### 27.3.1 Blockchain and the IoT

Blockchain technology would give better solution to the problems faced by IoT systems. In the growing scenarios of IoT systems, there are more chances for having increased number of interacting things or devices in it. This would lead to many hurdles because, in IoT systems,

mostly the collected data is maintained in the central servers. If the devices want to access the data, they have to interact using the centralized network and the data flow will happen through the central server, this process flow is clearly depicted in Fig. 27.3. In such large-scale IoT systems, the centralized server will not be an effective approach [14, 20]. Most of the IoT systems, that are implemented as of now are relying on centralized server concept. For handling the huge data processed in large scale IoT systems, there is a need for increasing the internet infrastructure. One best way to solve this is to have decentralized or distributed networks where “Peer-to-Peer Networking (PPN), Distributed File Sharing (DFS), and Autonomous Device Coordination (ADC)” functions could be capable [14].

Blockchain can carry out these three functions allowing the IoT systems to track the huge number of connected and networked devices. Blockchain allows a peer to peer messaging in faster way with the help of distributed ledger as shown in Fig. 27.4. The data flow process in IoT with Blockchain technology is different from only IoT system. In IoT with BC, the data flow is from sensors-network-router-internet-distributed blockchain-analytics-user. Here, the distributed ledger is tamper proof which does not allow in misinterpretation, wrong authentications in data. Blockchain complexly eliminates the Single Thread Communication (STC) in IoT making the system more trust less. With the adoption of Blockchain in IoT, the data flow will become more reliable and secure [14].

A blockchain-based, decentralized IoT can become a truly revolutionary approach to transaction processing among devices (see Fig. 27.5).



Fig. 27.3 – Data flow in Internet of Things [14]

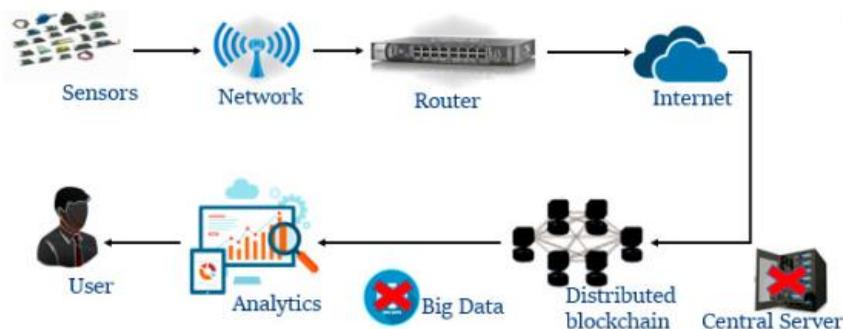


Fig. 27.4 – Data flow in Internet of Things with Blockchain Technology [14]

It is important to note that while Bitcoin contains an escalating difficulty in the blockchain mining process to restrict the issuance of currency, no such restriction is necessary in our vision of blockchains for the IoT. For the ADEPT implementation of a blockchain-based IoT, we chose the Ethereum protocol in its alpha version.<sup>6</sup> Ethereum's improvements to the traditional blockchain approach of Bitcoin, the Turing complete scripting languages it introduced and its ability to create binding contracts were extremely compelling for our PoC [18].

The project for Autonomous Decentralized Peer-to-Peer Telemetry (ADEPT) led by IBM and Samsung [18] aims to promote device autonomy, and to this end they use blockchain technology to ensure code execution on edge devices. ADEPT uses three protocols: Telehash, Bittorrent and Ethereum, for messaging, file sharing and blockchain, respectively. Blockchain technology provides authentication, engagement, contracts and checklists.

When integrating blockchain, it needs to be decided where these interactions will take place: inside the IoT, a hybrid design involving IoT and blockchain, or through blockchain [20].

Fog computing has also revolutionized the IoT with the inclusion of a new layer between cloud computing and IoT devices and could also facilitate this integration. Below, these alternatives

(shown in Fig. 27.6) are described together with their advantages and disadvantages [20]:

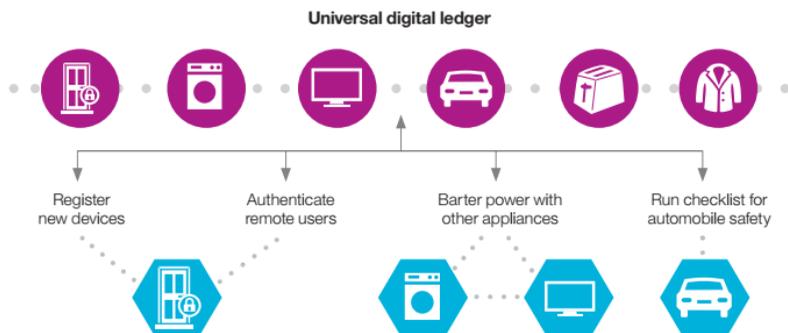


Fig. 27.5 – The blockchain functions as a distributed transaction ledger for various IoT transactions

**IoT–IoT:** this approach could be the fastest one in terms of latency, and security since it can work offline. IoT devices have to be able to communicate with each other, which usually involves discovery and routing mechanisms.

Only a part of IoT data is stored in blockchain whereas the IoT interactions take place without using the blockchain (Fig. 27.6 a).

- **IoT–Blockchain:** in this approach all the interactions go through blockchain, enabling an immutable record of interactions. This approach ensures that all the chosen interactions are traceable as their details can be queried in the blockchain, and moreover it increases the autonomy of IoT devices. Nevertheless, recording all the interactions in blockchain would involve an increase in bandwidth and data, which is one of the well-known challenges in blockchain (Fig. 27.6 b). On the other hand, all IoT data associated with these transactions should also be stored in blockchain.

- **Hybrid approach:** lastly, a hybrid design where only part of the interactions and data take place in the blockchain and the rest are directly shared between the IoT devices. One of the challenges in this approach is choosing which interactions should go through the blockchain and providing the way to decide this in run time (Fig. 27.6 c).

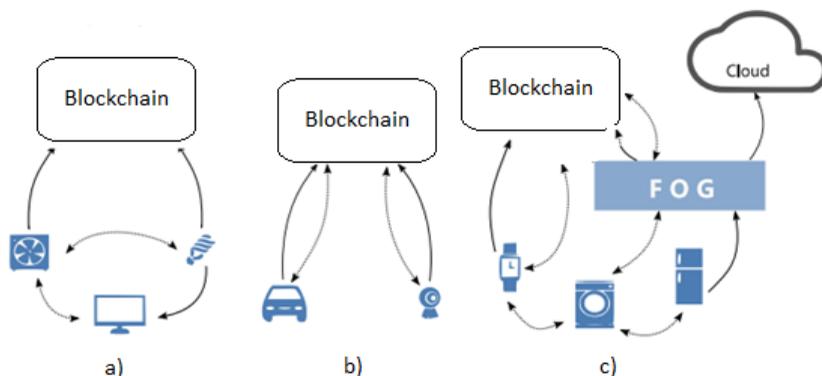


Fig. 27.6 – Blockchain IoT interactions

### 27.3.2 Benefits of Integrating Blockchain with IoT

There are many benefits of adopting blockchain with IoT, as shown in Fig.27.7. These benefits can be summarized as follows [11, 12, 13, 20, 26, 28]:

**1. Decentralization.** Because of the decentralized architecture of IoT, blockchain is most suitable as a security solution in IoT. The shift from a centralized architecture to a P2P distributed one will remove central points of failures and bottlenecks [24]. The majority of participants must verify the transactions in order to approve it and add it to the distributed ledger. There is no single authority that can approve the transactions or set specific rules to have transactions accepted.

Therefore, there is a massive amount of trust included since the majority of the participants in the network have to reach an agreement to validate transactions. Other benefits that come with the decentralization of the architecture are an improvement of the fault tolerance and system scalability. It would reduce the IoT silos, and additionally contribute to improving the IoT scalability and becomes more robust to DoS attacks.

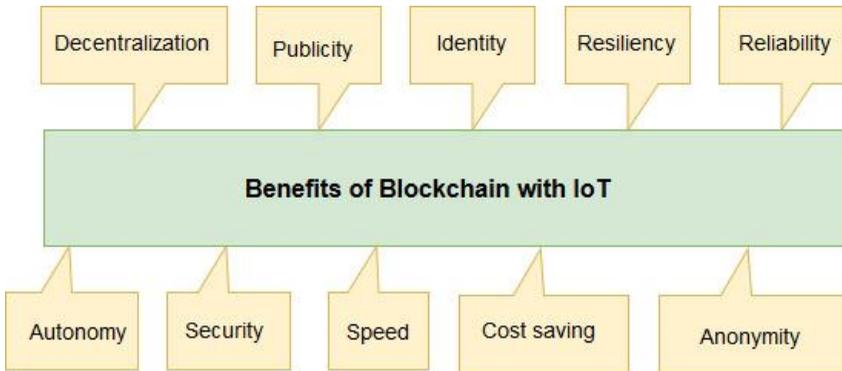


Fig. 27.7 – Benefits of integrating blockchain with IoT

**2. Publicity.** All participants have the ability to see the all the transactions and all blocks as each participant has its own ledger. The content of the transaction is protected by participant’s private key [3, 12], so even all participants can see them, they are protected. The IoT is a dynamic system in which all connected devices can share information together and at the same time protecting users’ privacy.

**3. Identity.** Using a common blockchain system participants are able to identify every single device. Data provided and fed into the system is immutable and uniquely identifies actual data that was provided by a device. Additionally, blockchain can provide trusted distributed authentication and authorization of devices for IoT applications.

**4. Resiliency.** Each node has its own copy of the ledger that contains all transactions that have ever made in the network. So, the blockchain is better able to withstand attack. Even if one node was compromised, the blockchain would be maintained by every other node. Having a copy of data at each node in the IoT will improve information sharing needs. However, it introduces new processing and storage issues.

**5. Reliability.** IoT information can remain immutable and distributed over time in blockchain. Participants of the system are capable of verifying the authenticity of the data and have the certainty that they have not been tampered with. Moreover, the technology

enables sensor data traceability and accountability. Reliability is the key aspect of the blockchain to bring in the IoT.

**6. Autonomy.** Blockchain technology empowers next-gen application features, making possible the development of smart autonomous assets and hardware as a service. With blockchain, devices are capable of interacting with each other without the involvement of any servers.

**7. Security.** Information and communications can be secured if they are stored as transactions of the blockchain. Each transaction, before being sent to blockchain network, is signed by the node and must be verified and validated by miners. After the validation, it's practically impossible to forge or modify transactions already saved in the blockchain. This provides a proof of traceable events in the system. Blockchain has the ability to provide a secure network over untrusted parties which is needed in IoT with numerous and heterogeneous devices [12]. In other words, all IoT network nodes must be malicious to perform an attack.

**8. Speed:** A blockchain transaction is distributed across the network in minutes and will be processed at any time throughout the day.

**9. Cost saving.** Existing IoT solutions are expensive because of the high infrastructure and maintenance cost associated with centralized architecture, large server farms, and networking equipment. The total amount of communications that will have to be handled when there are tens of billions of IoT devices will increase those costs substantially [11].

**10. Anonymity:** The nodes in blockchain are identified by their public keys (or the hash of public keys). These pseudonyms don't link any information about the identity of the participating nodes. This feature has been criticised as it increases the use of cryptocurrencies in the illegal online market. However, it could be seen as an advantage if used for other purposes, for example, electoral voting systems.

### ***27.3.3 Main challenges of blockchain in IoT***

Despite the blockchain's benefits mentioned above, it is still some challenges to be solved in order to adapt the blockchain technology in IoT. We enumerate the following challenges (Fig.27.8) [11, 12, 13, 20]:

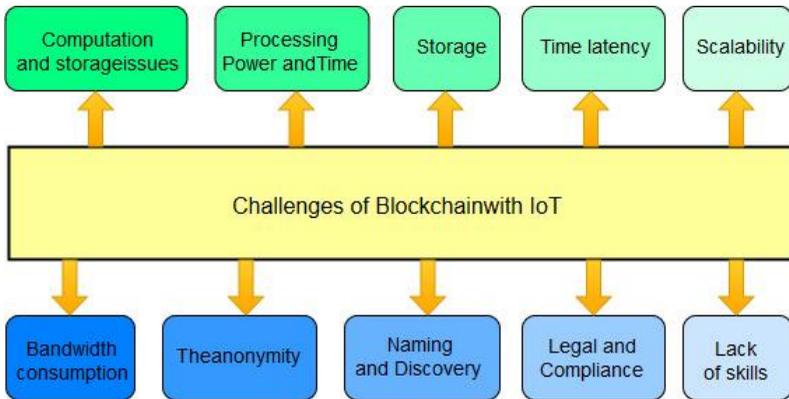


Fig. 27.8 – Main challenges of blockchain in IoT

**1. Computation and storage issues.** As most of IoT devices have limited capabilities in terms of computation and storage resources, the blockchain needs to be customized before its application as security solution in IoT. To address the problem of adaptability, one solution may consist to add a new application level that hides the details of blockchain implementation, namely the PoW. This solution allows the resource-constrained IoT devices to involve in the system without computing the PoW.

**2. Processing Power and Time.** The processing power and time needed to achieve encryption for all the objects included in a blockchain system. IoT systems have different types of devices which have very different computing capabilities, and not all of them will be able to run the same encryption algorithms at the required speed.

**3. Storage.** One of the main benefits of blockchain is that it eliminates the need for a central server to store transactions and device IDs, but the ledger has to be stored on the nodes themselves. The distributed ledger will increase in size as time passes and with increasing number of nodes in the network. As said earlier, IoT devices have low computational resources and very low storage capacity.

**4. Time latency.** In bitcoin blockchain, the validation of transactions takes about 10 minutes, which creates a problem for real time applications.

**5. Scalability.** Scalability issues in the blockchain might lead to centralization, which is casting a shadow over the future of the

cryptocurrency. The blockchain scales poorly as the number of nodes in the network increases. This issue is serious as IoT networks are expected to contain a large number of nodes.

**6. Bandwidth consumption.** As IoT devices generate a lot of transactions, this includes an important problem if it is necessary to validate each of those transactions that consume a lot of bandwidth.

**7. The anonymity.** Actually, blockchain doesn't ensure a fully anonymous transactions. Indeed, the peers are identified by pseudonyms that can be tracked but they are still unlikable (impossibility of extracting identity of the person from its pseudonym).

**8. Naming and Discovery.** The blockchain technology has not been designed for the IoT, meaning that nodes were not meant to find each other in the network. An example is the Bitcoin application in which the IP addresses of some "senders" are embedded within the Bitcoin client and used by nodes to build the network topology. This approach will not work for the IoT as IoT devices will keep moving all the time which will change the topology continuously.

**9. Legal and Compliance.** The blockchain is a new technology that will have the ability to connect different people from different countries without having any legal or compliance code to follow, which is a serious issue for both manufacturers and service providers. This challenge will be the major barrier for adopting blockchain in many businesses and applications.

**10. Lack of skills.** The blockchain technology is still new. Therefore, a few people have large knowledge and skills about the blockchain, especially in banking. In other applications, there is a widespread lack of understanding of how the blockchain works [2]. The IoT devices exist everywhere, so adopting the blockchain with IoT will be very difficult without public awareness about the blockchain.

### *27.3.4 Blockchain-based the IoT security solutions*

Let us consider and summarize some of the intrinsic features of blockchain that can be immensely useful for IoT in general, and IoT security in particular [12].

**1. Address Space.** Blockchain has a 160-bit address space, as opposed to IPv6 address space which has 128-bit address space [4]. A blockchain address is 20 bytes or a 160-bit hash of the public key generated by ECDSA (Elliptic Curve Digital Signature Algorithm).

With 160-bit address, blockchain can generate and allocate addresses offline for around  $1.46 * 10^{48}$  IoT devices.

The probability of address collision is approximately  $10^{48}$ , which is considered sufficiently secure to provide a GUID (Global Unique Identifier) which requires no registration or uniqueness verification when assigning and allocating an address to an IoT device.

With blockchain, a centralized authority and governance, as that of the Internet Assigned Numbers Authority (IANA), is eliminated. Currently, IANA oversees the allocation of global IPv4 and IPv6 addresses. Furthermore, blockchain provides 4.3 billion addresses more than IPv6, therefore making blockchain a more scalable solution for IoT than IPv6.

Lastly, it is worth noting that many IoT devices are constrained in memory and computation capacity, and therefore will be unfit to run an IPv6 stack.

**2. Identity of Things (IDoT) and Governance.** Identity and Access Management (IAM) for IoT must address a number of challenging issues in an efficiently, secure, and trustworthy manner. One primary challenge deals with ownership and identity relationships of IoT devices. Ownership of a device changes during the lifetime of the device from the manufacturer, supplier, retailer, and consumer.

The consumer ownership of an IoT device can be changed or revoked, if the device gets resold, decommissioned, or compromised. Managing of attributes and relationships of an IoT device is another challenge. Attributes of a device can include manufacturer, make, type, serial number, deployment GPS coordinates, location, etc. Apart from attributes, capabilities, and features, IoT devices have relationships. IoT relationships may include device-to-human, device-to-device, or device-to-service. An IoT device relationships can be deployed by, used by, shipped by, sold by, upgraded by, repaired by, sold by, etc.

The approaches like *TrustChain* are proposed to enable trusted transactions using blockchain while maintaining the integrity of the transactions in a distributed environment.

**3. Data Authentication and Integrity.** By design, data transmitted by IoT devices connected to the blockchain network will always be cryptographically proofed and signed by the true sender that holds a unique public key and GUID, and thereby ensuring authentication and integrity of transmitted data. In addition, all

transactions made to or by an IoT device are recorded on the blockchain distributed ledger and can be tracked securely.

**4. Authentication, Authorization, and Privacy.** Blockchain smart contracts have the ability to provide a de-centralized authentication rules and logic to be able to provide single and multiparty authentication to an IoT Device. The smart contracts can spell out also who has the right to update, upgrade, patch the IoT software or hardware, reset the IoT device, provision of new keypairs, initiate a service or repair request, change ownership, and provision or re-provision of the device.

**5. Secure Communications.** IoT application communication protocols as those of HTTP, MQTT, CoAP, or XMPP, or even protocols related to routing as those of RPL and 6LoWPAN, are not secure by design [12].

With blockchain, key management and distribution are totally eliminated, as each IoT device would have his own unique GUID and asymmetric key pair once installed and connected to the blockchain network. This will lead also to significant simplification of other security protocols as that of DTLS, with no need to handle and exchange PKI certificates at the handshake phase in case of DTLS or TLS (or IKE in case of IPsec) to negotiate the cipher suite parameters for encryption and hashing and to establish the master and session keys. Therefore, light-weight security protocols that would fit and stratify the requirements for the compute and memory resources of IoT devices become more feasible.

#### **27.4 Work related analysis**

The integration of blockchain with IoT have investigated in the next papers.

Bin Yu et al. 2018 in [25] demonstrate the applicability of blockchain to IoT devices and data management with an aim of providing end-to-end trust for trading. The authors first demonstrate that Blockchain, which is designed to remove the trusted third-party in a decentralized system, is an ideal solution to resolve the trust issue in IoT ecosystems. They then describe how with the help of blockchain, different parties can trust and verify the data and also the ownership of IoT devices and their related data can be traced.

This paper of Nazri Abdullah et al. 2017 [1] presents drawbacks of Kerberos implementations and identifies authentication requirements that can enhance the security of Big Data in distributed environments. The enhancement proposed by authors is based on the rising technology of blockchain that overcomes shortcomings of Kerberos such as numerous security issues, replay attacks, DDoS and single point of failure are some examples.

Angelo Caposelle et al. 2018 in [6] present a sustainable model for fostering the creation of s-health applications, identify and discuss the existing challenges, and explore the role of blockchain in overcoming some of them. The authors explain how mobile s-health applications can improve prediction, prevention, and prescriptive care, while generating feedback that make cities smarter when accounting for and adapting to individual needs and as a result, the constantly ongoing societal challenge of improving individual life will receive additional support.

The article of Blesson Varghese et al. 2018 [23] describes how distributed-ledger technologies (such as blockchains) provide a promising approach to support the operation of a marketplace and regulate its behavior (such as the GDPR in Europe) and operation. The authors describe two scenarios – smart cities and healthcare, that provide context for the discussion of how such a marketplace would function and be utilized in practice. Given this context, they described a marketplace for services that can exist at the network edge.

In [8] Ferrer E. C. considers how the combination of blockchain technology and swarm robotic systems can provide innovative solutions to emergent issues, by using the robots as nodes in a network. Proposed by author new security models and methods can be implemented in order to give data confidentiality and entity validation to robot swarms, therefore making them suitable for trust-sensitive applications. Distributed decision making and collaborative missions can be easily designed, implemented, and carried out by using special transactions in the ledger, which enable robotic agents to vote and reach agreements.

In [20] it is provided an extensive description of the main challenges that blockchain and IoT must address in order for them to successfully work together, in particular, key points where blockchain technology can help improve IoT applications. The authors present

possible ways of integration and platforms that are integrating IoT and blockchain in a general context.

Hany F. et al. 2018 in [11] provide an overview of the integration of the blockchain with the IoT with highlighting the integration benefits and challenges.

The authors conclude that the combination of blockchain and IoT can provide a powerful approach which can significantly pave the way for new business models and distributed applications.

Panarello A. et al. 2018 in [17] present a comprehensive survey on blockchain and IoT integration. In this paper analyzed the current research trends on the usage of blockchain-related approaches and technologies in an IoT context and point out the main open issues and future research directions.

Lee B., & Lee J. H. 2017 in [15] consider a secure firmware update issue, which is a fundamental security challenge for the embedded devices in an IoT environment and propose a new firmware update scheme that utilizes a blockchain technology to securely check a firmware version, validate the correctness of firmware, and download the latest firmware for the embedded devices.

In [5] is presented a decentralized, peer-to-peer platform called BPIIoT for Industrial Internet of Things based on the Block chain technology which enables peers in a decentralized, trustless, peer-to-peer network to interact with each other without the need for a trusted intermediary.

Liu B. et al. 2017 in [16] propose a blockchain-based framework for Data Integrity Service, the relevant protocols and a prototype system, conduct the performance evaluation of the implemented prototype system and discuss the test results.

Christidis K., & Devetsikiotis M. 2016 in [7] consider smart contracts-scripts that reside on the blockchain that allow for the automation of multi-step processes. It is shown that the blockchain-IoT combination is powerful and can cause significant transformations across several industries, paving the way for new business models and novel, distributed applications.

Gantait A. et al. 2017 in [10] discuss the use of blockchain in IoT solutions and explore how different industries are leveraging these two technologies to build end-to-end automated and secured solutions. In

[9] it is shown the use the IBM Watson IoT platform and IBM Blockchain service to build a sample use case.

There are some universities in the USA and EU (including ALIOT project partners) which conduct research and implement MSc and PhD educational modules related to Blockchain and connection of this methodology with IoT protection. In particular, the following courses and programs have been considered:

- IBM Blockchain Course - Blockchain for Developers [29];
- Linux FoundationX: Blockchain: Understanding Its Uses and Implications [30];
- University of California at Berkeley. Blockchain Technology [31];
- University of Oxford: Oxford Blockchain Strategy Programme [32];
- Princeton university. Bitcoin and Cryptocurrency Technologies [34];
- University System of Georgia. Cybersecurity and the Internet of Things [35].
- KTH University, Sweden: Master's programme in ICT Cloud and network infrastructures (CLNI). It includes topics related to blockchain in the cloud and network infrastructure [36].

Course is a new stage in digital evolution and focuses on the study and blockchain technology usage in various areas including the Internet of Things.

### ***Conclusions and questions***

The formation of a knowledge system of safety and security of Internet of Things systems is becoming an important part of the process of training specialists in the field of computer science.

The basics of blockchain technology and examples of implementation in the Internet of things are discussed and the consensus algorithms used in the blockchain technology are considered in this section.

The principles of ensuring the Internet of things safety and security using the blockchain technology are discussed.

The advantages and the existing problems of the blockchain technology integration in the Internet of things are highlighted.

IoT applications have to deal with security problems at different levels, but with an additional complexity due to the lack of performance and high heterogeneity of devices.

The increasing number of attacks on IoT networks, and their serious effects, make it even more necessary to create an IoT with more sophisticated security.

Blockchain can enrich the IoT by providing a trusted sharing service, where information is reliable and can be traceable. Data sources can be identified at any time and data remains immutable over time, increasing its security. In the cases where the IoT information should be securely shared between many participants this integration would represent a key revolution.

However, one of the main challenges in the integration of the IoT with blockchain is the reliability of the data generated by the IoT. Blockchain can ensure that data in the chain are immutable and can identify their transformations, nevertheless when data arrives already corrupted in the blockchain they stay corrupt. Corrupt IoT data can arise from many situations apart from malicious ones.

For effective usage of Blockchain technology in the IoT should be developed the Blockchain architecture that takes into account the above-mentioned IoT constraints and provides decentralized security and confidentiality of data. In this section, the materials for module PCM 3.4 of PC 3 course “Dependability and Security of IoT” are presented. They can be used for preparation to lectures and self-learning.

In order to better understand and assimilate the educational material that is presented in this section, we invite you to answer the following questions.

1. What is Blockchain?
2. Describe the block structure in Blockchain.
3. What is a genesis block?
4. What is a Merkle tree?
5. What is a hash function?
6. Explain the notion of collision for hash functions.
7. Explain the avalanche effect of hash function.
8. What is a consensus algorithm?
9. Explain the principle of Proof of Work algorithm.
10. Explain the principle of Poof of Stake algorithm.
11. Explain the principle of the 51% attack.
12. Explain the principle of the Sybil attack.

13. What is the process of mining?
14. What are the disadvantages of Proof of Work algorithm.
15. What are the alternative consensus algorithms.
16. How does the Blockchain technology ensure the reliability of IoT data?
17. How does the Blockchain Technology secure IoT?
18. How does the Blockchain Technology provides anonymity in IoT?
19. Describe the benefits of the integration of Blockchain with IoT.
20. Describe the existing challenges of *integrating Blockchain with IoT*.

### **References**

1. N. Abdullah, A. Hakansson, E. Moradian. "Blockchain based approach to enhance big data authentication in distributed environment," in *Proc. 9th International Conf. Ubiquitous and Future Networks (ICUFN), IEEE*, 2017, P. 887-892.
2. A. Banafa. Internet: <https://iot.ieee.org/newsletter/january-2017/iot-andblockchain-convergence-benefits-and-challenges.html>, Febr. 12, 2019..
3. T. Ahram, A.Sargolzaei, S.Sargolzaei, J.Daniels, B. Amaba. "Blockchain technology innovations," in *Proc. Technology & Engineering Management (TEMSCON), 2017 IEEE Conference on*, 2017, P. 137–141.
4. A.M. Antonopoulos. *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*. California, Sebastopol: O'Reilly Media, Inc., 2014.
5. A. Bahga, V. K. Madiseti. "Blockchain platform for industrial internet of things". *Journal of Software Engineering and Applications*, vol.9(10), P. 533-546, 2016.
6. A. Capossele, A.Gaglione, M.Nati, M.Conti, R.Lazzeretti, P. Missier. "Leveraging blockchain to enable smart-health applications," in *Proc.2018 IEEE 4th International Forum on Research and Technology for Society and Industry (RTSI)*, 2018, P. 1-6.
7. K. Christidis, M. Devetsikiotis. "Blockchains and smart contracts for the internet of things," *IEEE Access*, 4, 2016, P.2292-2303.

8. E.C.Ferrer. "The blockchain: a new framework for robotic swarm systems," in *Proc. of the Future Technologies Conference*. Springer, Cham, 2018, P. 1037-1058.

9. A. Gantait, J.Patra, A. Mukherjee. "Implementing blockchain for cognitive IoT applications," Part 2: Use vehicle sensor data to execute smart transactions in Blockchain. IBM DeveloperWorks, 9, 2017.

10. A. Gantait, J.Patra, A.Mukherjee. "Implementing blockchain for cognitive IoT applications," Part 1: Integrate device data with smart contracts in IBM Blockchain. IBM DeveloperWorks, 9, 2017.

11. H. F. Atlam, A.Alenezi, M. O.Alassafi, G. Wills. "Blockchain with Internet of Things: Benefits, Challenges, and Future Directions". *International Journal of Intelligent Systems and Applications (IJISA)*, vol.10 (6), P.40-48, 2018.

12. M. A. Khan, K. Salah. "IoT security: Review, blockchain solutions, and open challenges". *Future Generation Computer Systems*, no.82, P.395-411, 2018.

13. D. E. Kouicem, A.Bouabdallah, H. Lakhlef. "Internet of things security: A top-down survey". *Computer Networks*. Elsevier, In press, 141, P.199-221, 2018.

14. N. M. Kumara, P.K.Mallickb. "Blockchain technology for security issues and challenges in IoT". *Procedia Computer Science*, 132, P. 1815-1823, 2018.

15. B. Lee, J. H. Lee. "Blockchain-based secure firmware update for embedded devices in an Internet of Things environment". *The Journal of Supercomputing*, vol.73(3), P. 1152-1167, 2017.

16. B. Liu, X. L.Yu, S.Chen, X.Xu, L. Zhu. "Blockchain based data integrity service framework for IoT data," in *Proc.Web Services (ICWS), 2017 IEEE International Conference on*, 2017, P. 468-475.

17. A. Panarello, N.Tapas, G.Merlino, F.Longo, A.Puliafito "Blockchain and IoT integration: A systematic survey". *Sensors*, vol.18(8), 2575, P.1-37, 2018.

18. B. S. Panikkar, S.Nair, P.Brody, V. Pureswaran. "ADEPT: An IoT Practitioner Perspective," Internet: <http://pdf.yt/d/esMcC00dKmdo53->, 2015 Nov. 20, 2018..

19. "Proof of Stake versus Proof of Work". White Paper, Internet: <https://bitfury.com/content/downloads/pos-vs-pow-1.0.2.pdf>\_\_Jan. 22, 2019..

20. A. Reyna, C.Martín, J.Chen, E.Soler, M. Díaz. “On blockchain and its integration with IoT. Challenges and opportunities”. *Future Generation Computer Systems*, vol. 88, P. 173–190, 2018.

21. M. Salimitari, M. Chatterjee. “An Overview of Blockchain and Consensus Protocols for IoT Networks”. arXiv preprint arXiv:1809.05613, 2018.

22. Satoshi Nakamoto. “Bitcoin: A Peer-to-Peer Electronic Cash System”. Internet: <https://bitcoin.org/bitcoin.pdf> Dec. 12, 2018.

23. B. Varghese, M.Villari, O.Rana, P.James, T.Shah, M.Fazio, R.Ranjan. “Realizing Edge Marketplaces: Challenges and Opportunities”. *IEEE Cloud Computing*, vol.5(6), P. 9-20, 2018.

24. P. Veena, S.Panikkar, S.Nair, P. Brody. “Empowering the Edge - Practical Insights on a Decentralized Internet of Things”. *IBM Institute for Business Online. Value*, 17, Apr. 2015. Available: <http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?infotype=PM&subtype=XB&htmlfid=GBE03662USEN#loaded> Dec. 6, 2018..

25. B. Yu, J.Wright, S.Nepal, L.Zhu, J.Liu, R.Ranjan. “IoTChain: Establishing trust in the internet of things ecosystem using blockchain”. *IEEE Cloud Computing*, vol.5(4), P.12-23, 2018.

26. Z. Zheng, S.Xie, H. N.Dai, H.Wang. “Blockchain challenges and opportunities: A survey”. *Int. J. Web and Grid Services*, vol. 14 (4), P.352-375, 2018.

27. N.G.Yatskiv, S.V.Yatskiv “Perspectives of the Usage of Blockchain Technology in the Internet of Things”. *The Scientific Bulletin of UNFU*, vol. 26, n.8, P. 381-387, 2016. (In Ukrainian)

28. V.Yatskiv, N.Yatskiv, O. Bandrivskiyi. “Proof of Video Integrity Based on Blockchain”, in *Proc. Advanced Computer Information Technologies (ACIT), 2019 IEEE 9th International Conference on*, 2019, P. 431-434.

29. IBM Blockchain Course- Blockchain for Developers. <https://developer.ibm.com/courses/all-courses/blockchain-for-developers/> July 29, 2019..

30. Linux FoundationX: Blockchain: Understanding Its Uses and Implications. <https://www.awin1.com/cread.php?awinmid=6798&awinaffid=427859&clickref=ddblockchainlinux&p=https%3A%2F%2Fwww.edx.org%2F>

professional-certificate%2Flinuxfoundationx-blockchain-for-business July 29, 2019..

31. University of California at Berkeley. Blockchain Technology. <https://www.edx.org/course/blockchain-advancing-decentralized-technology> July 29, 2019..

32. University of Oxford: Oxford Blockchain Strategy Programme. <https://www.sbs.ox.ac.uk/programmes/oxford-blockchain-strategy-programme> July 29, 2019..

34. Princeton university. Bitcoin and Cryptocurrency Technologies. <https://www.coursera.org/learn/cryptocurrency> July 30, 2019..

35. University System of Georgia. Cybersecurity and the Internet of Things. <https://www.coursera.org/learn/iot-cyber-security> July 30, 2019..

36. KTH University, Sweden: Master's programme in ICT Cloud and network infrastructures (CLNI) <https://www.kth.se/en/studies/master/ict-innovation/introduction-1.609472> July 30, 2019..

## **PART VIII. DEVELOPMENT AND IMPLEMENTATION OF IOT-BASED SYSTEMS**

### **28. BASIC CONCEPTS AND APPROACHES TO DEVELOPMENT AND IMPLEMENTATION OF IOT SYSTEMS**

Prof., DrS. I. S. Skarga-Bandurova,  
Ph.D. Student A. Y. Velykzhanin (EUNU)

#### *Contents*

|                                                                    |     |
|--------------------------------------------------------------------|-----|
| Abbreviations .....                                                | 404 |
| 28.1 IoT-based system development process .....                    | 405 |
| 28.1.1. Phases and deliverables of an IoT technical strategy ..... | 405 |
| 28.1.2 The IoT development strategies .....                        | 408 |
| 28.2 Strategies to planning IoT architectures .....                | 413 |
| 28.3 The base components of the IoT systems .....                  | 419 |
| 28.3.1 Major types of technological offerings from IoT .....       | 419 |
| 28.3.2 IoT device classification.....                              | 421 |
| 28.3.3 IoT device design flow .....                                | 423 |
| 28.3.4 Relationship between Sensor Networks and IoT .....          | 424 |
| 28.4 The IoT development boards and platforms for prototyping..... | 426 |
| 28.5 The IoT platforms: types and selection criteria .....         | 429 |
| 28.6 Work related analysis .....                                   | 431 |
| Conclusions and questions.....                                     | 432 |
| References .....                                                   | 433 |

## **Abbreviations**

C2C – Cloud-to-Cloud

CPU – Central Process Unit

D2D – Device-to-Device

D2S – Device-to-Server

FPGA – Field-Programmable Gate Array

GPU – Graphics Processing Unit

I/O – Input/Output

IIoT – Industrial Internet of Things

IoT – Internet of Things

IT – Information Technology

M2C – Machine-to-Cloud

M2M – Machine-to-Machine

MEMS – Micro Electromechanical System

OT – Operational Technology

PCB – Printed Circuit Board

RFID – Radio Frequency Identifier

S2S – Device-to-Server

SN – Sensor Network

TCP – Transmission Control Protocol

ThingML – Internet of Things Modelling Language

UDP – User Datagram Protocol

## **28.1 IoT-based system development process**

Internet of Things (IoT) industry has following three key characteristics [1]:

- being a driver for customer-facing innovation it is developed by an associations and government more then by market;
- it has complicated and diverse industrial chain without a unique subject of responsibility;
- it requires the coordination of the development processes as well as interacting with humans to learn from his intelligence and provide more accurate analytics;
- it is in need of new standards, highly professional developers, highly exclusive products.

In these conditions, it is essential to perform careful thinking about how to promote IoT solution and how to hold it out. Moreover, further development of IoT is close related to the public welfare, ability and willingness of people to use it. This means that rapid challenge the traditional development process, ways to collect fees form customers IoT as well as wide application of new strategies for operation and integration platforms.

### ***28.1.1. Phases and deliverables of an IoT technical strategy***

In many areas, product developments processes take a lot of time to fully mature. The rapid growth of the IoT and Industrial Internet of Things (IIoT) raises another incarnation of the product development process. The IoT market is the highly integrated and flexible system with the open and scalable architectures for integration of relatively cheap sensors, processors, new connectivity protocols, and power supply sources. In this context, development processes should take into account a vide variety of custom solutions, different types of hardware and software platforms, possible adoption of hardware development environments, and provide the best solution to robust IoT design.

The process of development and implementation of IoT-based systems is a sequence of related activities, starting with the project definition phase where objectives and information needs are considered and ending with the dissemination of the information product for use by

communities, scientists and decision-makers. The structure of IoT technical program includes the following elements (Fig. 28.1):

- objective settings;
- preliminary surveys and resource estimation;
- IoT solution strategy;
- choosing IoT reference architecture;
- solution design;
- monitoring design;
- system implementation;
- quality-assurance procedures;
- data management and product development.

The component connection framework indicates of the phases and deliverables of an IoT technical strategy under each of the elements are described. These components and their connection links need to be adequately considered during the planning process of an IoT system. The planning process conveniently can be classified into the following three main subsequent phases.

The first phase “Project definition” includes defining the needs and establishing the objectives of IoT (such as in support of monitoring or research and policy) and what issues are to be addressed. With the objectives defined, it can then be decided what data are needed and how they will be used. Then IoT solution strategy, IoT reference architecture and technology platforms, are defined. IoT reference architecture provides a set of architectural patterns, standards, and best practices for use in developing IoT solutions [2]. Detailed technical reference architecture can be created only after obtaining a clear understanding of the IoT solution ecosystem.

The second phase “Project development” comprises the developing phase, which should consider and include:

- the solution design with data specification, defining device capabilities, prototyping, platform finalization and performing risk analysis;
- the planning of a monitoring network with the choice of location for the sampling operations, supported by preliminary investigations (inventories and surveys) needed before the program is started, so that issues, problems and risk factors can be clearly identified and evaluated;

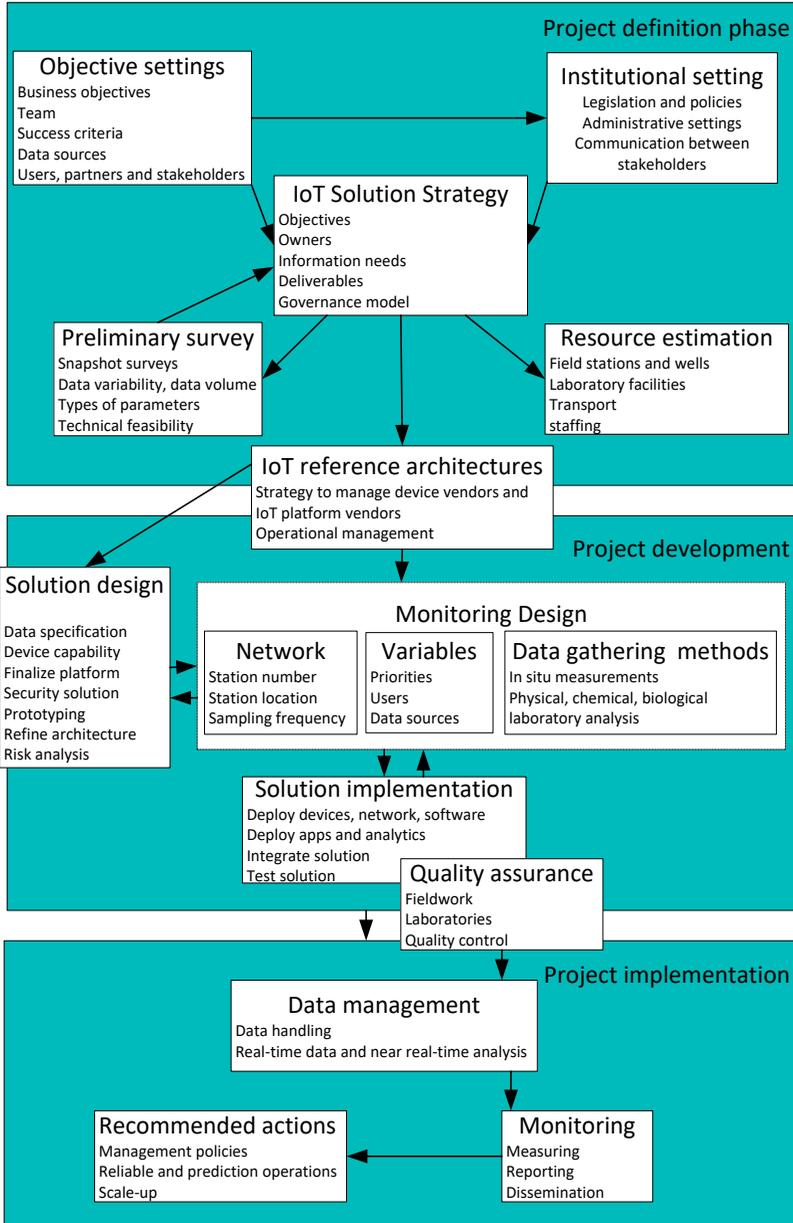


Fig. 28.1 – Phases and deliverables of an IoT technical strategy

- the selection of physical, chemical or biological variables, i.e. which variables to monitor for different uses and in relation to different data sources;
- the definition of sampling procedures and operations, such as in situ measurements with different devices, manual or automated measurements, for sampling appropriate media, sample pre-treatment and conservation, identification and shipment;
- the planning of field measurements (frequency) – in appropriate cases;
- the definition of the resources required for the realization of an IoT program, e.g. the available national facilities, the inventory of field stations, equipment and instruments, vehicles and other transportation means, office and field staff involved in complex IoT activities, human resources development and training required, internal and external communication needs and, finally, the estimated costs of the program.

The third phase “Project implementation” comprises the actual operations (implementation) of the program, with:

- the setting up of a quality-assurance system at the strategic/organizational, tactical and operational levels, essential for ensuring the reliability of information obtained by monitoring, covering field and laboratory analysis, data management, data handling, real-time and near real-time analysis, and the application of IoT standards and indices;
- the policies management, implementation reliable and prediction operations and development of products, leading to the reporting and dissemination of results and findings.

### ***28.1.2 The IoT development strategies***

It is assumed that implementation of IoT system intends the use of different languages, namely, Python, Java, SQL queries, etc., hardware and software components, execution on external devices as well as active interaction with third-party APIs.

There are two main strategies when developing IoT systems, namely, mashup-driven and model-driven [3]. Mashup-driven strategy assumes that IoT system is developed by combination or mashing up existing services and typically based on familiar web development tools and approaches (e.g. prototyping). Thus, this strategy is often

applicable for designing personalized, customized, short-lived and non business-critical applications [4].

Model-driven methodology assumes that IoT system is described on a higher level of abstraction permitting an expressive systems modeling possibly with code generation [3].

*Model-driven methodology for the development of IoT-based systems.*

The model-driven approach for the design and development of IoT-based systems was proposed in [3, 5], there are some e.g. Internet of Things Modelling Language (ThingML) [6] that provides a customizable code generation framework which can be customized to specific languages, middleware, operating systems, libraries and systems [7]. In [5], Mezghani et al. identified two main phases of model-based methodology: (1) Requirements Identification and (2) Requirements Formalization (Fig. 28.2).

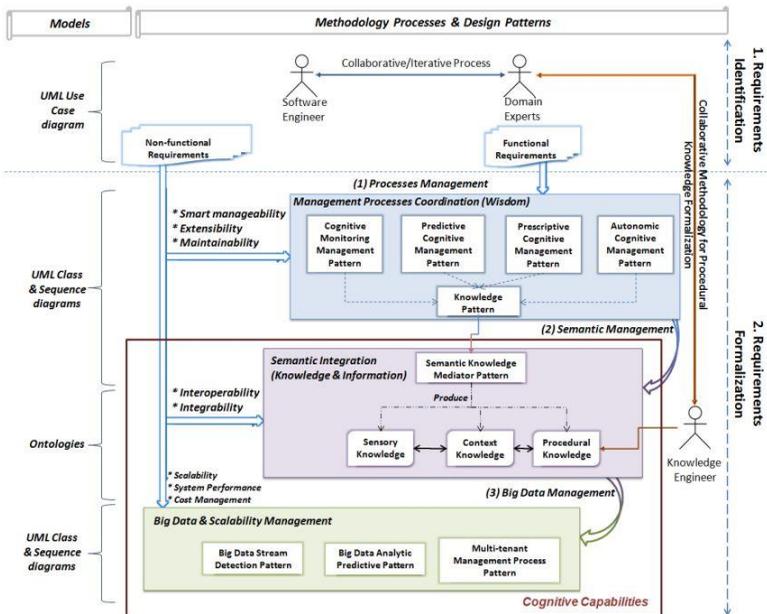


Fig. 28.2 – A model-driven methodology for the design of IoT-based system [5]

The first phase “Requirements identification” is grounded on consultations and discussions with the domain experts to gathering information about the system functionality and identification of the non-functional requirements. This process is an iterative, where the functional requirements can be represented using the UML diagrams that describe the functions of the IoT system.

The second phase “Requirement formalization” is targeted on structuring requirements and development of models for describing the relationship of system processes.

Within this phase, it is possible to detach the sub-levels with specific challenges related to the design of IoT systems such as the coordination, integration, big data management, etc.

A main advantage of model-driven approach is the platform independent modeling, which enables code generation for specific IoT platforms.

*Mashup-based methodology for the development of IoT systems.*

Mashup tools make it possible to perform very quick prototyping and leverage converting, transforming, and combining the data from one or multiple services to meet the project goals. They also enable connecting different services to create new processes. In addition, some mashup tools e.g. Clickscript [8], WotKit [9], Paraimpu [10] can provide simulation means and support interoperability between different platforms. They could be quite efficient in describing system architecture, message flow (like activity diagrams), and deployment.

The main phases of mashup-based methodology may include:

*Phase 1:* Exploring the development landscape to identify the most suitable tools currently available to satisfy IoT project goals.

When choosing the tool it is crucial to take in consideration following requirements. The good solution should:

- use the most widely used platforms;
- be open-source and relies on open standards;
- be deployed locally;
- support several programming languages.

*Phase 2:* Choosing platform for integration between services and remote platforms.

For example, this phase can be performed using Jupyter Notebook an open-source computational notebook that enables to combine code,

text and visualization in a single document whose underlying structure is JSON [11].

*Phase 3:* For the development of an emerging industry, it is very important to have standards. The development of IIoT also needs much more standardization.

*Phase 4:* Data processing and manipulation.

In IoT projects, data collection process consists of IoT data acquisition and conventional data acquisition directed on supporting legacy systems. In turn, IoT data acquisition may involve five alternative IoT session layer protocols namely AMQP, CoAP, DDS, MQTT, and XMPP. Depending on the application one or more protocols for IoT communication can be selected for the target system. More detail information about protocols and standards for data collection and transferring can be found in Chapter 29. The detailed feature diagram given in Fig. 28.3 can be further extended with respect to specific project requirements.

Data processing features mainly depend on the application type and include Image/ Video processing, data mining, data logging, and decision-support features. One or more features might be used at the same time. Depending on the application requirements these features can be extended to use different processing features.

Data visualization consists of monitoring and mapping functions. Monitoring consists of environment monitoring and yield monitoring functions. Mapping includes yield, soil type, and light mapping features. System management includes sensor control, actuator control, system control features such as device identification, node discovery, and directory and naming services.

Sensor control consists of several sub-features such as soil sensing, light sensing, weather sensing, and water sensing. Also, system control includes vehicle control and UAV/Drone control features. Finally, external services feature contains externally communicated systems such as weather forecast, finance services, and other external systems.

Mashup tools typically provide a graphic editor for interconnection of services in one application as well as modeling message flow between the components. Components can be sensors, IoT gateways, external Web-services, etc. In this context, mashup methodology can be

seen as specific forms of end-user programming that is limited to the specific model.

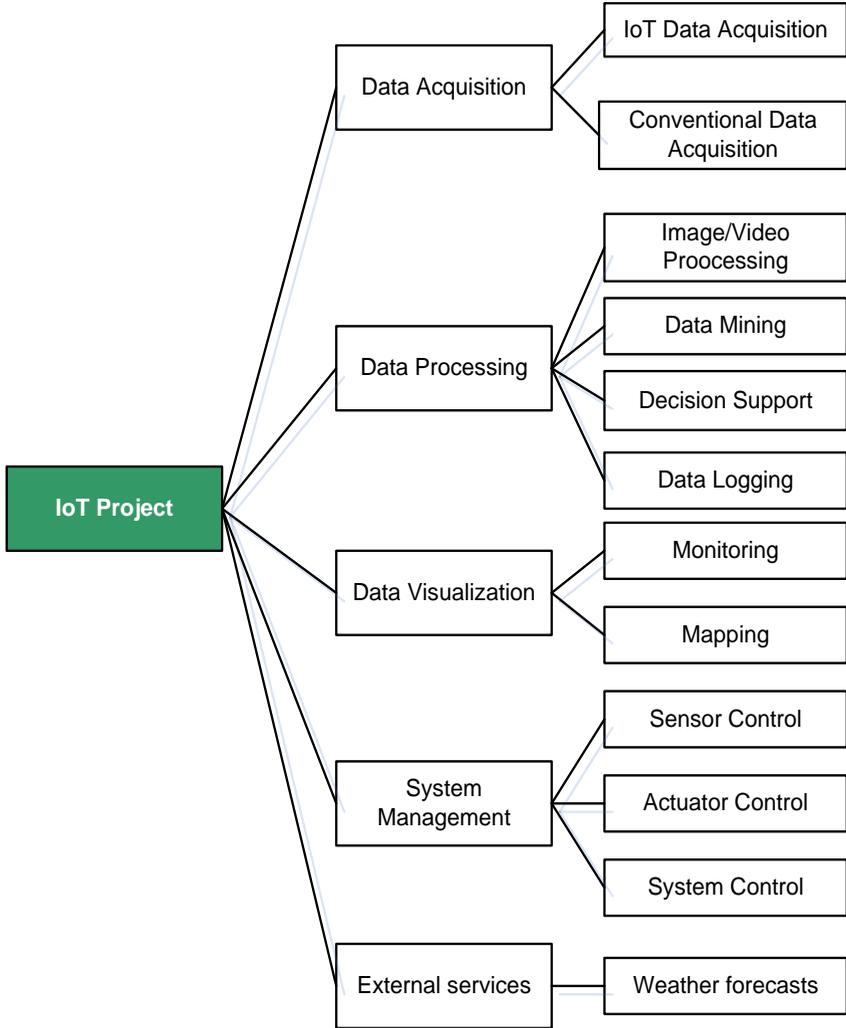


Fig. 28.3 – Data processing in IoT project

As it mentioned in [3], mashup tools can benefit from concepts in model-based approaches. Also, model-driven approaches can fit better to the IoT and provide easy to use tools.

### 28.2 Strategies to planning IoT architectures

There are many variations of the architectural layers, components and interactions among them. In this section we discuss IoT reference architectures and features of Industrial IoT System Architecture.

While planning IoT system architecture it is necessary to guarantee uninterrupted service of all components and fusion the physical and virtual assets. To reach this goal, the IoT systems have to be dependable, adaptable, highly-scalable, human-centric and secure. The overall IoT System Architecture is presented in Fig. 28.4.

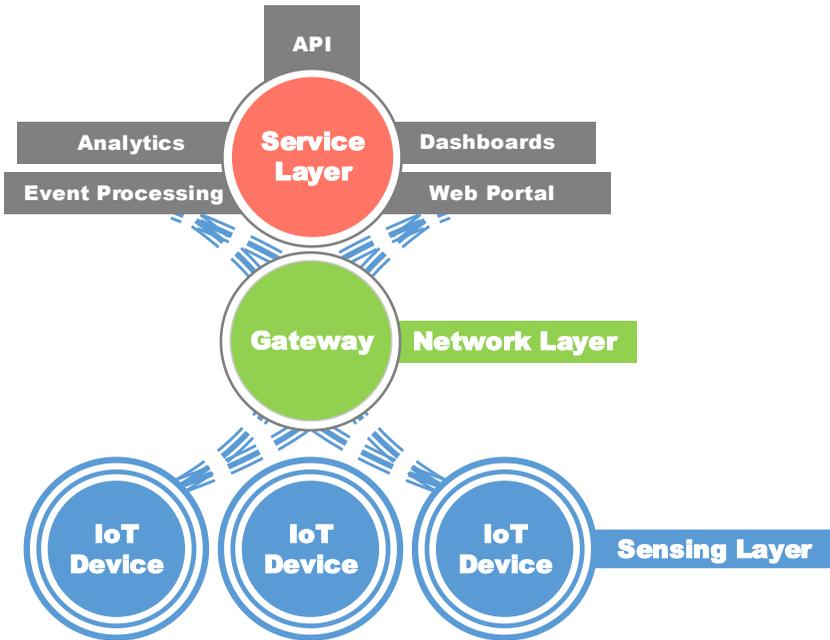


Fig. 28.4 – IoT layers

The Sensing Layer includes sensing devices such as sensors, actuators, embedded systems, and Radio Frequency Identifier (RFID).

The Network Layer includes the gateway and the routes for the data transmission from gateways to different application users.

The Service Layer provides information services according to the user and system requirements. The Service Layer may include powerful data centers and different data servers for the data mining, analysis, processing, storage, and applications.

Geber A. [12] compared different architectures for several IoT applications. An example of reference architecture shown in Fig. 28.5 (a) contains the drivers with embedded sensors and actuators, the gateway, the IoT integration middleware where the processing logic is executed and the application that uses the data from the previous layer.

Hence, the simplest reference architecture includes three layers: sensor, network and application. More sophisticated structure could contain from five to seven levels: device, network, processing, application, business, management and security.

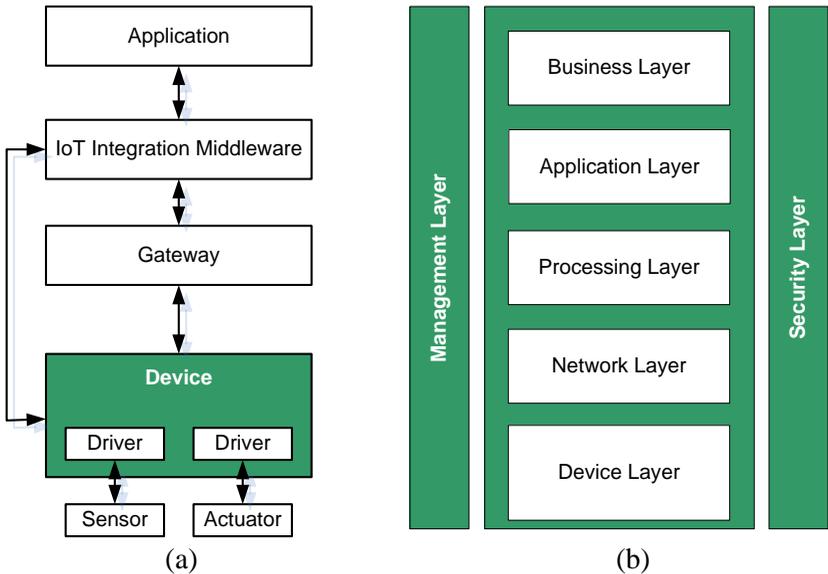


Fig. 28.5 – The two types of IoT architectures: the IoT reference architecture (a) [12] and the IoT 7-layer architecture (b) [13, 14]

The architecture (shown in Fig. 28.5 (b)) was proposed in [13]. It gives a similar overview of the IoT layers.

The device layer consists of sensors, actuators, embedded systems and physical devices. The main task of this layer is to identify and collect data and specific information obtained from environment by sensors and IoT-devices. These data is transferred to the network layer. The network layer is dedicated for networking connectivity and further transporting data. An alternative name of this layer is transport layer. The main requirements beside time and quality include providing secure transmitting data gathered from sensors to the procession layer. The transmission can be wired and/or wireless.

The procession layer is responsible for service management and consists of functionality for setting up and taking down of the association between the IoT connection points. Most of standards and protocols supporting operation of this layer use the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP) for transport. However, they have different architectures and characteristics for various purposes.

Feature diagram of procession layer communication protocols of IoT is given in Fig. 28.6. There are three types of source-target relations available in procession layer protocols: Device-to-Device (D2D), Device-to-Server (D2S), Device-to-Server (S2S).

In some studies, these features are respectively: Machine-to-Machine (M2M), Machine-to-Cloud (M2C), Cloud-to-Cloud (C2C).

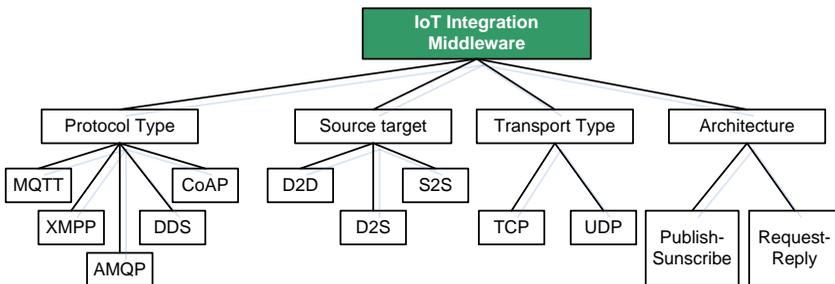


Fig. 28.6 – Feature diagram of procession layer communication protocols of IoT [14]

There are many criteria to select the right protocol for IoT processing layer. Further information on the selection of the proper IoT session layer protocol could be found in [16].

Procession layer protocols are similar to the ones in the network layer. For all communication protocols, the transport layer could be either the UDP or TCP. Some protocols like DDS support both UDP and TCP.

Another important process is the selection of network layer protocol since using TCP and/or UDP changes the characteristics of the communication from performance and security perspectives. As it mentioned above, the IoT devices have different functionality and transmission range. For low power devices and networks adoption of TCP in the network layer is less feasible. Instead, the UDP protocol has to be used. On the other hand, UDP does not provide common security protocols (SSL/TLS) that why TCP is required for supporting security in IoT system. From this perspective, the processing layer protocols can be either publish-subscribe or request-reply. In publish-subscribe architecture, sensors send data to a topic on which cloud software that are registered to this topic read data. Interesting feature that in this architecture publishers and subscribers do not need to know each other, and do not need to be operating at the same time.

This type of communication is well suited for data that must flow from one producer to many consumers. On the other hand, for the request-reply architecture, senders and receivers do need to know each other. The requester sends a request message and waits for the response. When the sensor receives the request, it responds with a reply message. The session layer protocols of IoT generally use publish-subscribe architecture except in the case of CoAP in which a request-reply pattern is adopted.

The application layer consists of different IoT services and manages the system using the data from the processing layer.

The business layer defines business logic and workflows it takes care of the ownership and is responsible for the application management. This layer is dedicated to management of all IoT systems, services and applications within the domain.

The security layer is a side-car layer relating to the other five layers and provides the security functionality.

Complexity is one of the biggest challenges during planning IoT solutions due to they involve many IoT devices that communicate with each other and with cloud services and applications producing a huge amount of data. IoT devices can be connected either directly to a network or through a gateway (Fig. 28.7), which enables the devices to communicate with each other and with cloud services and applications [15].

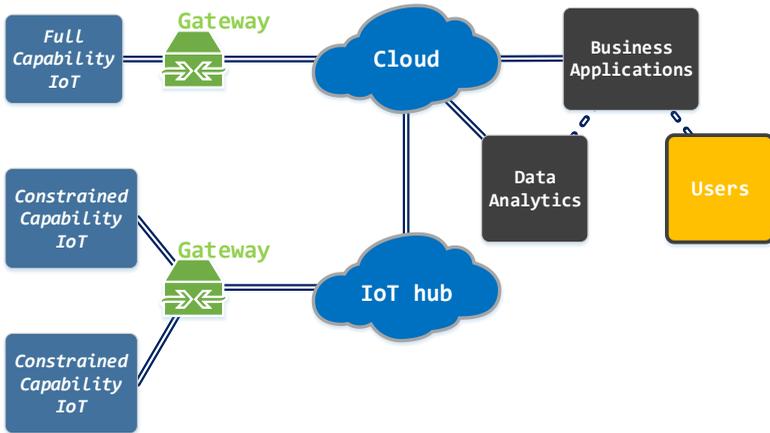


Fig. 28.7 – Simplified IoT System Architecture

Industrial IoT system architecture presented in Fig. 28.8 includes IT (Information Technology) and OT (Operational Technology).

OT conditionally divided on two levels and has the following characteristics: Triggered by event, Changing data, low latency real time; Use of MQTT, COAP (D2S, ~10ms, collect) or, DDS (D2D, ~0.1ms, distribute).

The first level of edge processing (low latency) involves:

- real time connectivity with data aggregation, device management, device security, communication gateway and processing;
- support low latency industrial real time process management and event generation;
- connects actuators, analyzers, drives, vision, video, controllers and robotics.

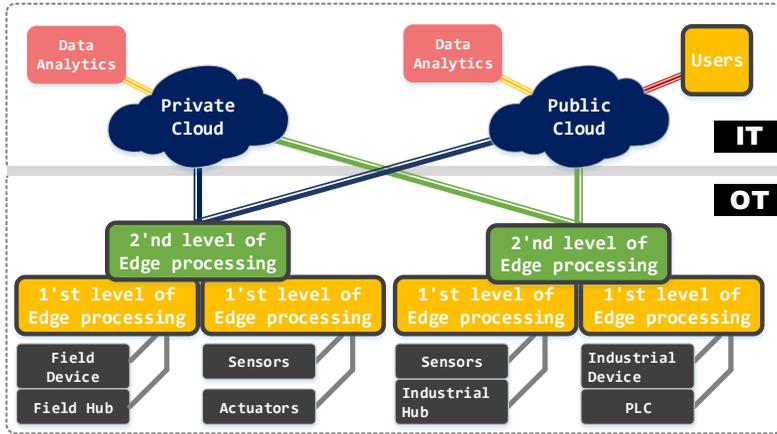


Fig. 28.8 – Industrial IoT System Architecture

The second level of edge processing (medium latency) supports medium latency industrial process management, alarm generation, storage. Existing plant field is connected with sensors, data management, advanced analytics, decision making, people, and automation. This level allows enterprise to control and communicate to field assets.

Essential differences between IIoT and general class of IoT are:

- a large number of nodes;
- controlled latency at various levels of hierarchical data processing;
- keep existing plant field running.

The main benefits of IIoT architectures are:

- asset optimization;
- process optimization;
- business Optimization.

Industrial IoT requirements:

- existing manufacturing field industrial I/O devices including sensors, actuators, analyzers, drives, vision, video, and robotics;
- accommodate large number of nodes;
- controlled latency at various levels of hierarchical processing;

- high reliability, high availability, safe & resilience to failure;
- provides smarter services (monitoring, alarm management);
- non-invasive IT integration with OT.

IT is top down designed from business point of view and is a well-defined level while OT is defined bottom up with different vendor proprietary equipment:

- communication network system that works in presence of internet, intermittent internet or, independent and connects to edge nodes for real time processing;
- high access security and provision;
- human interaction.

Talking about the overall architectures (Fig. 28.5), we refer to the reference architectures. It should be mentioned that reference architectures vary depending on the application domain; however, most IoT reference architectures describe the following common capabilities:

- managing devices and their data;
- connectivity and communication layers;
- analytics and applications.

Typically, IoT design & implementation teams are cross-functional in nature and have specialists from different layers (HW, firmware, networking, data scientists and so on). The IoT expert should have practical exposure in all the layers and should combine an end-to-end solution together.

## **28.3 The base components of the IoT systems**

### ***28.3.1 Major types of technological offerings from IoT***

From previous chapters we define five main components that support the Internet of Things. Depending on levels of complexity they include: things (sensors, actuators, smart devices, and embedded systems), IoT gateways or simple hubs, cloud or integrating hubs, and enhanced services. Moving up, the components become more complex and their connectivity increases [17]:

*1. Sensors, actuators, smart devices, and embedded systems* are the components of the sensing layer. They collect data from different physical, human, and natural environments in an intelligent and

collaborative way, and temporarily store these data. Their connectivity enables two key capabilities: gathering and analyzing data from the environment.

2. *IoT gateways (Simple hubs)* are the devices that connect endpoints to broader networks. When integrated into products such as vehicle engines; washing machines; or home heating, venting, and air conditioning (HVAC) systems, the computing intelligence and storage embedded in a simple hub allows these products to adapt over time to the user's behavior and to optimize for efficiency.

3. *Integrating hubs* connect simple hubs and provide a diverse array of services that fit more or less seamlessly together.

4. *Network and cloud services* provide the infrastructure for functioning IoT. They can either be public (accessible to the population at large) or private (protected behind an organization's firewall). These services deliver the seamless and transparent connection to the Internet that hubs require, along with the cloud computing power needed to collect, store, and analyze vast amounts of data from myriad endpoints. They can also provide the infrastructure needed to build or connect to social networks, so that users of the IoT can compare experiences and share data.

5. *Data centers, services and analytics* this category comprises the most technologically sophisticated components of the IoT. Enhanced services enable to collect and analyze data from different platforms and deliver broad interactive functions.

These five technological options, from endpoints to enhanced services, provide a menu of diverse opportunities for companies building IoT businesses. There are several technological options (levels of complexity) on the road of developing IoT ecosystems (Table 28.1).

Depending on complexity levels they may include all major types of technological offerings from IoT: IoT devices, simple hubs, integrating hubs, and enhanced services.

Table 28.1 – Technological options for developing IoT solutions [17]

| Tasks        | Levels of complexity                            |                                                                                                                                                       |                                                                                                                        |                                                                                   |
|--------------|-------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|
|              | Smart devices and Sensors                       | IoT gateways / Simple Hubs                                                                                                                            | Integrating Hubs                                                                                                       | Services Analytics                                                                |
| Optimization |                                                 |                                                                                                                                                       | Industrial platforms for interconnecting analytics engines and business operations<br>Large-scale digital city systems |                                                                                   |
| Adaptation   | Stand-alone GPS navigation devices              | Auto insurance telematics systems<br>Smartphone apps that use location tracking                                                                       | Protocol-based platforms allowing diverse devices in a building to interconnect to one another and internet            | Emerging systems for setting insurance rates based on health and driving behavior |
| Control      | Motion- or light responsive alarms and controls | Internet-connected systems for heating, cooling, and ventilation<br>Bluetooth-enabled object identification sensor systems (iBeacon, Estimote Beacon) | Systems for controlling lights and appliances through remote or mobile devices                                         | Potential connected car traffic management systems                                |
| Monitoring   | Simple thermostats and motion sensors           | Fitness activity sensors and hub systems                                                                                                              | Medical wearables that feed data to online diagnostic platforms                                                        |                                                                                   |

**28.3.2 IoT device classification**

Devices are the base components of IoT infrastructure; they can be classified in many ways based on the type of data they handle such as, environmental, medical, financial, etc. or the economical sector where they are used such as, manufacturing, transportation, retail, consumer and home.

Geber et al. [15] characterize IoT devices on their high-level capabilities:

- data acquisition and control;
- data processing and storage;
- connectivity;
- power management.

Another classification for IoT devices based on the potential risks and impact on living beings during their operating is proposed in [18]. According to this view, IoT devices classified on three types:

*Type A:* If the loss of one or all of the security objectives causes severe physical, economic or social harm to the living thing. For example, malfunction of wireless pacemaker, a vehicle brake system, or a farm irrigation system.

*Type B:* If the loss of one or all of the security objectives causes minor physical, economic or social harm to the living thing<sup>5</sup>. For example, malfunction of one of the components of a heating, ventilation and air conditioning (HVAC) control system may cause heat exhaustion to humans and animals.

*Type C:* If the loss of one or all of the security objectives causes very minor or no harm to the living thing. For example, a cash register cannot process financial transactions online.

Table 28.2 – Examples of IoT devices and their types as it proposed in [18]

| Type A                                             | Type B                                | Type C                                                |
|----------------------------------------------------|---------------------------------------|-------------------------------------------------------|
| Medical pumps, monitors, implants, connected cars. | HVAC control systems, traffic lights. | Alarms, cameras, dishwashers, lights, garage openers. |

The selection of any type of IoT is a risk-based decision that should take into account other factors unique to the IoT system goals. In this case, developers can expand each type with “subtypes” to offer further sub-classification and granularity or create an IoT risk index.

For example, Type A(1) = Life support system, Type A(2) = stand-alone wireless blood pressure monitor.

### 28.3.3 IoT device design flow

IoT device design consists of software development for Central Process Unit (CPU) implementation, interface drivers, and, when application demands dictate, hardware design for custom accelerators, CPU customization, and board-designs. Typical IoT device design flowchart is depicted in Fig. 28.9.

Many IoT applications begin with prototyping both hardware and software on existing platforms. To minimize design and development costs, it is possible to reuse existing platforms. In this case, initial prototypes are typically designed on existing CPU-based platforms. With the initial implementation, designers can evaluate performance and quality to determine whether a CPU-only, Graphics Processing Unit (GPU)-based, or Field-Programmable Gate Array (FPGA)-based platform is necessary to achieve goals [19].

To minimize time-to-market, software and hardware are often developed at the same time. A software team is concentrating on software features, embedded compilation, drivers and device integration with cloud computation services. In parallel, the hardware team performs system-level modeling, component selection, design implementation, integration and verification. Despite the generally parallel development processes, the software and hardware design flows influence each other; software algorithm demands may alter hardware performance objectives, and hardware implementation choices can influence how software is designed and implemented.

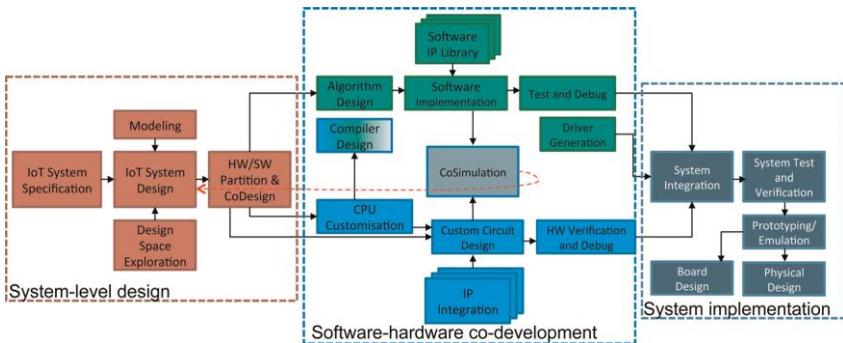


Fig. 28.9 – Typical IoT device design flowchart [19]

When designing IoT device, it is necessary to solve a number of issues related to the implementation of the following set of required components:

- devices and/or sensors for measuring the selected parameters;
- the method of placement of sensors that will be constantly in contact with the object of monitoring;
- power supply sources;
- data transmission facilities;
- housing for installation and protection of measuring instruments and accessories;
- tools to protect the station from possible interference and the environment.

Indeed, if a hardware implementation is created, there may be no need for design and optimization of the software version.

#### ***28.3.4 Relationship between Sensor Networks and IoT***

The sensor networks are the second essential component of the IoT-based system. The IoT comprises sensors and actuators. The data is collected using sensors. Then, it is processed and decisions are made. Finally, actuators perform the identifiable system operations (Fig. 28.10).

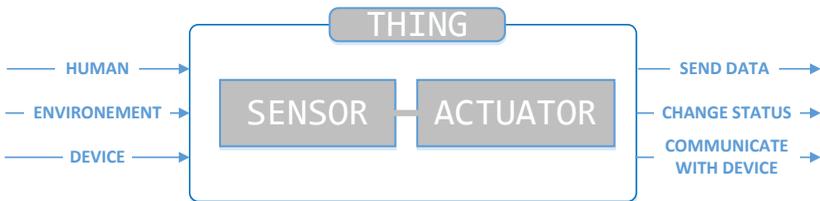


Fig. 28.10 – Interaction structure by means of sensor and actuator

The differences between Sensor Network (SN) and the IoT are largely unexplored and blurred. We can elaborate some of the characteristics of both SN and IoT to identify the differences (Fig. 28.11).

SN comprises of the sensor hardware (sensors and actuators), firmware and a thin layer of software. The IoT comprises everything that SN comprises and further it comprises a thick layer of software such as middleware systems, frameworks, APIs and many more software components. The software layer is installed across computational devices (both low and high-end) and the cloud.

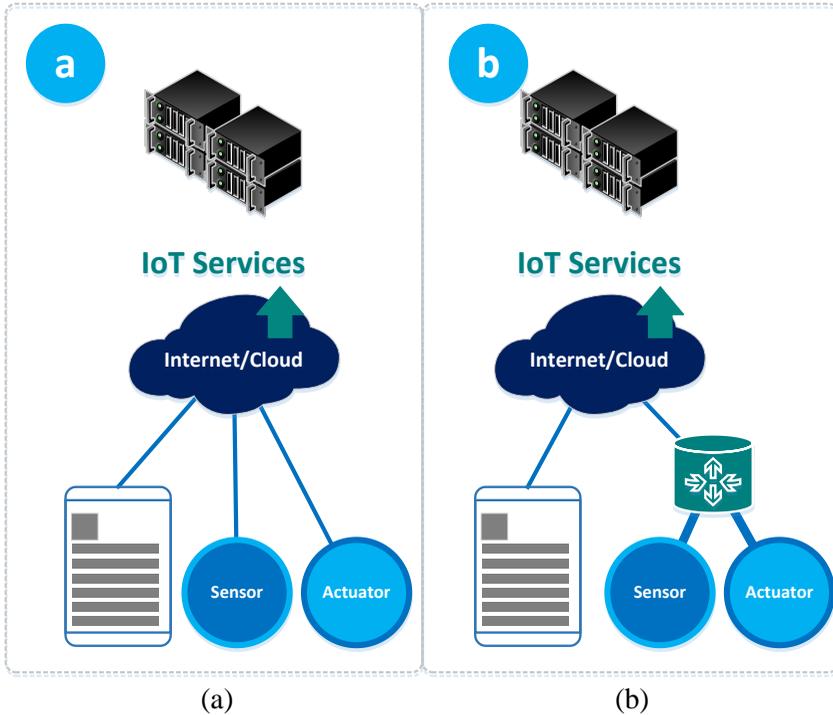


Fig. 28.11 – Two approaches for device connection: direct network connection (a) and connection through the gateway (b)

From their origin, SNs were designed, developed, and used for specific application purposes.

In the early days, sensor networks were largely used for monitoring purposes and not for actuation. In contrast, IoT is not focused on specific applications, instead IoT can be considered as a general purpose sensor network.

During the stage of deploying sensors, the IoT would not be targeted to collect specific types of sensor data, rather it would deploy sensors where they can be used for various application domains. For example, company may deploy sensors, such as pressure sensors, on a newly built bridge to track its structural health. However, these sensors may be reused and connect with many other sensors in order to track traffic at a later stage. Therefore, middleware solutions, frameworks, and APIs are designed to provide generic services and functionalities such as intelligence, semantic interoperability, context-awareness, etc. that are required to perform communication between sensors and actuators effectively.

Sensor networks can exist without the IoT. However, the IoT cannot exist without SN, because SN provides the majority of hardware (e.g. sensing and communicating) infrastructure support, through providing access to sensors and actuators. There are several other technologies that can provide access to sensor hardware, such as wireless ad-hoc networks. However, they are not scalable and cannot accommodate the needs of the IoT individually, though they can complement the IoT infrastructure.

### **28.4 The IoT development boards and platforms for prototyping**

In many cases, IoT development process includes prototyping using a single platform or their combinations that matches the desired target as closely as possible. IoT development boards contain standard communications, sensor interfaces, and general purpose I/O connections so that the user can easily integrate sensors, actuators, communications, and Micro Electromechanical System (MEMS) to prototype their system.

Although these prototypes will not be used for production releases, they play an important role in demonstrating a proof-of-concept and evaluating overall feasibility. Prototypes may have limited modeling fidelity to the final product characteristics, yet even rough estimates of size, power/energy, performance, and reliability can be represented as expected product feasibility [19].

For prototyping certain IoT project an existing CPU boards, GPU boards, or FPGA boards can be used, as shown in Fig. 28.12.

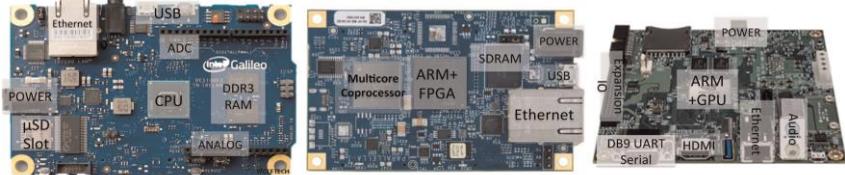


Fig. 28.12 – Examples IoT development boards [19]

The most popular IoT platforms for prototyping are Arduino, Raspberry Pi, ESP8266 and Spark Core. Table 28.3 summarizes information about these platforms to understand their capabilities and limitations.

The Raspberry Pi is ideal for study server-based IoT projects. It can connect to both LAN and Wi-Fi networks. However, it is not recommend using the Raspberry Pi for projects where there is a custom made Printed Circuit Board (PCB) or to integrating Raspberry Pi into a finished product.

Arduino is ideal for logging sensor data and controlling actuators via commands posted on the server by another client.

Table 28.3 – Comparison of IoT platforms for prototyping

| Prototyping platform | Connecting to the Internet                                                                                                    | Features                                                                                                              | Advantages                                                                                                                                                   | Disadvantages                                                                                                                                     |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| Arduino              | Arduino Yun has built-in capability to connect to Wi-Fi and LAN networks, others rely on the Wi-Fi shield and Ethernet shield | Device capabilities vary across the official Arduino models, and between the dozens of third-party compatible boards. | <ul style="list-style-type: none"> <li>- easy to use</li> <li>- has a huge community for technical support.</li> <li>- easy to create a prototype</li> </ul> | <ul style="list-style-type: none"> <li>- all Arduino boards, apart from Yun, need an external module so as to connect to the internet.</li> </ul> |
| Raspberry Pi         | Wireless or via an Ethernet cable.                                                                                            | High storage space, RAM, powerful processor.                                                                          | <ul style="list-style-type: none"> <li>- don't need extra shields or hardware to connect to</li> </ul>                                                       | <ul style="list-style-type: none"> <li>- not easy to set up and code apps</li> <li>- cannot be</li> </ul>                                         |

28. Basic concepts and approaches to development and implementation of IoT systems

|            |                                                                                                                                          |                                                                                                                                                                                                                                                                           |                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                                         |
|------------|------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|            |                                                                                                                                          | Supports many programming languages to create server-side apps                                                                                                                                                                                                            | the internet.<br>- automatic connecting to wi-fi or LAN, so long as the router that has DHCP configured.<br>- a huge Raspberry Pi community.                                                 | easily integrated into a product<br>- numerous OS<br>- not easy to decide on the best OS for the device you are creating.                                                                                                                                                                                               |
| ESP8266    | Self-contained Wi-Fi module that can provide any microcontroller with access to Wi-Fi networks                                           | Preprogramd AT commands that enable to hook it onto Arduino board and use Wi-Fi capabilities. Easy connecting to Arduino board or another microcontroller over UART. The module can be integrated with application specific devices and sensors easily through its GPIOs. | - plug and play Wi-Fi module.<br>- don't need to write huge chunks of code to get started.<br>- very cheap,<br>- easy to integrate the ESP8266 on a PCB<br>- perfect for building prototypes | - lack of SPI communication and support of SSL.<br>- doesn't have 5V to 3V logic level shifting<br>- lacks of voltage regulator on-board<br>- sometimes uses big spikes of current (the Arduino cannot supply).<br>- need to use an external 3.3v voltage regulator.<br>- harder to use in comparing to other platforms |
| Spark Core | Wi-Fi enabled IOT development platform. The back end of the Spark Core is a website platform cloud that allows to send and receive data. | TM32F103CB ARM 32-bit Cortex M3-based microcontroller and CC3000 Wi-Fi module                                                                                                                                                                                             | easy to integrate on a PCB as the Spark Photon chip, it is easy to use, has a good online community and comes connected to a cloud platform.                                                 | it is linked to the Spark platform, which you have no control over                                                                                                                                                                                                                                                      |

There are some instances where Arduino boards are used for server functions such as hosting a simple web page.

The ESP8266 is best used in client applications such as data logging and control of actuators from online server applications.

The Spark Core platform is ideal for both server and client functions. It can be used to log sensor data onto the Spark.io cloud or receive commands from the cloud. Spark cloud is available for free.

### 28.5 The IoT platforms: types and selection criteria

An IoT platform plays an important role in the IoT architecture. When building an IoT project or system, connected devices send data to cloud platforms. These platforms store data and use it to build charts.

Table 28.4 – Type of services

| Type of IoT Platform              | Overview                                                                                                                                | Example                                                                                                    |
|-----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|
| End-to-End                        | Provide the hardware, software, connectivity, security, and device management tools to handle millions of concurrent device connections | <a href="#"><u>Particle</u></a>                                                                            |
| Connectivity Management Platforms | Low power and low cost connectivity management solutions through Wi-Fi and cellular technologies                                        | <a href="#"><u>Mulesoft</u></a> ,<br><a href="#"><u>Hologram</u></a> ,<br><a href="#"><u>Sigfox</u></a>    |
| IoT Cloud                         | Monitor and track millions of simultaneous device connections.                                                                          | <a href="#"><u>Google Cloud IoT</u></a> ,<br><a href="#"><u>Salesforce Cloud IoT</u></a>                   |
| Data Platform                     | Combine different tools to route device data and manage / visualize data analytics                                                      | <a href="#"><u>Clearblade</u></a> ,<br><a href="#"><u>Azure</u></a> ,<br><a href="#"><u>ThingSpeak</u></a> |

An IoT platform is an integrated service that offers us the things we need to bring physical objects online and provides the following services:

- rapid application development and deployment;
- device management;
- integration;
- data storage;

- dashboard creation;
- security services.

The relative importance of these capabilities will vary from project to project and depend on the use case.

There are hundreds of IoT platforms available from a range of vendors. Before starting new project you should to assess your business needs and analyze how will your needs change over time.

There are following selection criteria for choosing the right Internet of Things platform for your project [20, 21]:

- connectivity: the type of connectivity (a Wi-Fi or cellular solution) for IoT system;
- type of service: some services are purely connectivity platforms, while others are end-to-end solutions that offer the hardware, software, and connectivity;
- solution lifetime: how long has the IoT platform been in business? The IoT platform offering services for 4+ years is a good solution;
- geographic coverage: does the IoT platform cover the regions your business needs?;
- data plan: does the vendor offer a fair data plan, for example if you decide to pause or suspend your data services at any time as well as the ability to control amount of data that is used. It is essential to achieve a vendor/technology agnostic solution which is easily transferable to someone else in case such needs arise;
- security / privacy: assess how IoT platform combats security issues. The Gateways of your cloud platform should offer SSL or DTLS encryption;
- managed integrations / API Access: how does the vendor integrate cellular modems, sim-cards, device diagnostics, firmware updates, cloud connections, security, application layer, RTOS, etc., into a simple package;
- redundancy and disaster recovery: does your cloud platform provider have a dedicated infrastructure to handle your data? How often is the data backup taken?;
- IoT ecosystem: the relationships between the services the IoT platform;

- data access: does the service match your needs in integrating the data acquired through the IoT platform with current cloud service?;
- hardware: does the vendor offer any off-the-shelf applications, developer kits, or starter packages for the specific use case you are targeting?;
- device management: how does the vendor allow you to monitor, segment, and manage IoT devices that are out in the field?;
- edge intelligence: IoT platform needs to be able to extend itself seamlessly from cloud to fog and even mist and support new topologies for decentralized computing;
- OTA firmware updates: how does the vendor allow you to send updates and fix bugs on your devices remotely? It is a simple or complex process?

### **28.6 Work related analysis**

Approaches to design and development of IoT-based systems has been proposed for several applications in software development life-cycle for the Internet-of-Things [7], designing autonomic systems and investigating their ability to support IoT challenges [22], identification of design challenges and developing a model-driven methodology for the IoT-Based Systems [5, 7], using Computational Notebooks for IoT Development [23].

Our partners from EU universities are actively involved in research and development for IoT-based systems, applying new knowledge in business and the educational process. For example, the University of Coimbra suggests several courses in this topic, namely, “Intelligent Sensors”, “Emergent Internet services”. The objectives of course on Emergent Internet services are the knowledge about telematics applications fundamentals, Internet applications and emerging Internet services.

School of Engineering at Newcastle University focuses on a broad range of communications, sensors and signal processing. One of their courses EEE8009 “Wired and Wireless Network Technologies” introduces a broad coverage of modern communication networks and network technologies, transmission and switching; to provide students with knowledge of the issues relating to modern telecommunications systems, protocols, flow and error control. Another one is “Embedded

Systems and Internet of Things” (ES-IoT) delivers an understanding of Embedded Systems and Internet of Things and their enabling technologies. It is industrially focused, tailored to the demands of companies that design and manufacture mobile electronic [24]. This course gathers five fields of knowledge which work well together: tools, techniques and design of Embedded Systems and Internet of Things (ES-IoT) and subsystems; scientific and engineering principles and practices of Computing Science and Electronic Engineering; embedded computer systems architecture; networking and communication systems; computer programming.

### *Conclusions and questions*

In this chapter, the materials for module PCM4.1 “Basic concepts and approaches to development and implementation of IoT systems” of PhD course “Development and implementation of IoT-based systems” are presented. They can be useful for preparation to lectures and self-learning for lecturers, PhD-students, IoT developers, etc.

In field of IoT solutions, research challenges are distributed in almost all aspects of their development and implementation, ranging from the enabling devices to the top level business models. So the research space for a complete IoT solution shows a cross-layer and multidisciplinary pattern

We intend to raise a series of research problems about the IoT architectures, device architectures and system integration to obtain an efficient IoT system. Also, we discussed an effective research approach to resolve an essential challenge in nowadays research on the IoT – the lack of basic technology, standards, and practical business requirements.

To develop a comprehensive solution for a particular application, developers must integrate multidisciplinary knowledge of ICT, management, business administration, and the target application. Moreover, to obtain good solution, the specific knowledge of the target area and application is essential. This chapter is directed on encourage readers on continue investigating the design methodologies and system models as well as development new applications.

In order to better understand and assimilate the educational material that is presented in this section, we invite you to answer the following questions.

22. What are the key characteristics of IoT industry?
23. Phases and deliverables of an IoT technical strategy?
24. What are the main strategies for developing IoT systems?
25. What are the differences between mashup-driven and model-driven approaches?
26. What methodology best fits to quick prototyping?
27. What are the main phases of mashup-based methodology?
28. How many layers can IoT architecture consist of?
29. What are the requirements for network layer?
30. What is the reference architecture?
31. What are the main benefits of IIoT architectures for enterprises?
32. What are the major types of technological offerings from IoT?
33. Which technological options provide diverse opportunities for companies building IoT businesses?
34. What types of IoT devices are known?
35. What are the differences between sensor network (SN) and the IoT?
36. What are selection criteria for choosing the right Internet of Things platform for your project?

### **References**

1. Y. Zhang and J. Yu, "A Study on the Fire IOT Development Strategy", *Procedia Engineering*, vol. 52, pp. 314-319, 2013. DOI: 10.1016/j.proeng.2013.02.146.
2. "Defining your IoT governance practices", IBM Developer, 2019. [Online]. Available: <https://developer.ibm.com/articles/iot-governance-01/>. [Accessed: 25- Feb- 2019].
3. C. Prehoferand and L. Chiarabini, "From Internet of things mashups to model-based development". *IEEE 39<sup>th</sup> Annual Computer Software and Applications Conference*, pp. 499-504, September 2015. <https://doi.org/10.1109/COMPSAC.2015.263>
4. M. Blackstock and R. Lea "IoT mashups with the WoTKit", *IEEE 3<sup>rd</sup> International Conference on the Internet of Things (IOT 2012)*, pp. 159–166, October 2012. <https://doi.org/10.1109/IOT.2012.6402318>

5. E. Mezghani, E. Expósito, K. Drira, "A Model-Driven Methodology for the Design of Autonomic and Cognitive IoT-Based Systems: Application to Healthcare", *IEEE Transactions on Emerging Topics in Computational Intelligence*, Vol. 1 (3), pp. 224-234, June 2017. DOI: 10.1109/TETCI.2017.2699218
6. F. Fleurey and B. Morin, "ThingML: A Generative Approach to Engineer Heterogeneous and Distributed Systems". *IEEE International Conference on Software Architecture Workshops (ICSAW)*, pp. 185–188, April 2017. <https://doi.org/10.1109/ICSAW.2017.63>
7. J. Dias and H. Ferreira, "State of the Software Development Life-Cycle for the Internet-of-Things", *Arxiv.org*, 2019. [Online]. Available: <https://arxiv.org/pdf/1811.04159>. [Accessed: 25- Feb- 2019].
8. D. Guinard, V. Trifa, E. Wilde, "A resource oriented architecture for the web of things", *Internet of Things (IOT)*, November-December 2010. DOI: 10.1109/IOT.2010.5678452
9. M. Blackstock and R. Lea, "IoT mashups with the WoTKit", *3rd International Conference on the Internet of Things (IOT)*, pp. 159–166, October 2012. DOI: 10.1109/IOT.2012.6402318
10. A. Pintus, D. Carboni, A. Piras, "Paraimpu: a platform for a social web of things", *21st International conference companion on World Wide Web. ACM*, pp. 401-404, April 2012. [Online]. Available: <https://www2012.universite-lyon.fr/proceedings/companion/p401.pdf> [Accessed: 01- Oct- 2018].
11. A. Rule, A. Tabard, J. Hollan, "Exploration and Explanation in Computational Notebooks". *CHI Conference on Human Factors in Computing Systems - CHI '18*, paper No. 32, April 2018. DOI:10.1145/3173574.3173606
12. J. Guth, U. Breitenbücher, M. Falkenthal, F. Leymann, L. Reinfurt, "Comparison of IoT Platform Architectures: A Field Study based on a Reference Architecture". *IEEE Conference on Cloudification of the Internet of Things (CIoT)*, pp. 1-6, November 2016.
13. M. Wu, T. Lu, F. Ling, J. Sun, H. Du, "Research on the Architecture of Internet of Things". *3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*. Vol. 5, pp. 484–487, August 2010.
14. Ö. Köksal and B. Tekinerdogan, "Architecture design approach for IoT-based farm management information systems", *Precision Agriculture*, 2018. Available: 10.1007/s11119-018-09624-8 [Accessed 25 January 2019].
15. A. Geber, "Simplify the development of your IoT solutions with IoT architectures", *IBM Developer*, 2019. [Online]. Available: <https://developer.ibm.com/articles/iot-lp201-iot-architectures/>. [Accessed: 25 January 2019].
16. Ö. Köksal and B. Tekinerdogan, "Feature-driven domain analysis of session layer protocols of Internet of Things". *IEEE International Congress on*

*Internet of Things, ICIOT*, pp. 105–112, June 2017. <https://doi.org/10.1109/IEEE.ICIoT.2017.19>.

17. F. Burkit, "A Strategist's Guide to the Internet of Things", [Online]. <https://www.strategy-business.com/article/00294?gko=a9303> [Accessed: 25- June-2019].

18. F. Uribe, "The Classification of Internet of Things (IoT) Devices Based on Their Impact on Living Things", *SSRN Electronic Journal*, 2018. DOI: 10.2139/ssrn.3350094.

19. D. Chen, J. Cong, S. Gurumani, W.-m. Hwu, K. Rupnow, Z. Zhang "Platform choices and design demands for IoT platforms: cost, power, and performance tradeoffs", *Journal IET Cyber-Physical Systems: Theory & Applications*, pp. 1-8, 2016.

20. J. Lee, "How to Choose the Right IoT Platform: The Ultimate Checklist". [Online]. Apr. 25, 2018. <https://hackernoon.com/how-to-choose-the-right-iot-platform-the-ultimate-checklist-47b5575d4e20> [Accessed: 25- June- 2019].

21. "Top 10 selection criteria to choose your IoT platform" [Online]. <https://iotify.io/top-10-selection-criteria-for-your-iot-cloud-platform/> [Accessed: 25- June- 2019].

22. C. Vidal, C. Fernández-Sánchez, J. Díaz, J. Pérez, "A model-driven engineering process for autonomic sensor-actuator networks," *International Journal of Distributed Sensor Networks*, vol. 2015, p. 18, 2015.

23. F. Corno, L. De Russis and J. Sáenz, "Towards Computational Notebooks for IoT Development", *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems - CHI EA '19*, 2019. DOI: 10.1145/3290607.3312963.

24. "Embedded Systems and Internet of Things MSc - Postgraduate - Newcastle University", [Ncl.ac.uk](https://www.ncl.ac.uk/postgraduate/courses/degrees/embedded-systems-internet-of-things-msc/#profile), 2019. [Online]. Available: <https://www.ncl.ac.uk/postgraduate/courses/degrees/embedded-systems-internet-of-things-msc/#profile>. [Accessed: 25- June- 2019].

## **29. MODELS FOR IOT-BASED DEVICES AND TECHNOLOGIES FOR DATA PROCESSING AND TRANSFER**

Prof., DrS. Yu. P. Kondratenko, Ass. Prof., Dr. G.V. Kondratenko,  
Ass. Prof., Dr. Ie.V. Sidenko, Ph.D. Student M.O. Taranov (PMBSNU)

### *Contents*

|                                                                                |     |
|--------------------------------------------------------------------------------|-----|
| Abbreviations .....                                                            | 437 |
| 29.1 IoT-based devices: models and network communication protocols .....       | 438 |
| 29.1.1 Types of models for IoT-based devices.....                              | 438 |
| 29.1.2 Tools and means for the development of information models.....          | 441 |
| 29.1.3 Network communication protocols for IoT-based devices.....              | 445 |
| 29.2 Technologies for data processing in IoT-based systems .....               | 447 |
| 29.2.1 Technologies for data collection and analysis from IoT devices .....    | 447 |
| 29.2.2 Technologies for data processing .....                                  | 451 |
| 29.2.3 Methods of management and forecasting .....                             | 453 |
| 29.3 Protocols and standards for data transfer between IoT-based devices ..... | 456 |
| 29.3.1 Protocols for data transfer.....                                        | 456 |
| 29.3.2 Standards for data transfer.....                                        | 458 |
| 29.3.3 Cybersecurity of IoT-based devices .....                                | 461 |
| 29.4 Work related analysis .....                                               | 464 |
| Conclusions and questions.....                                                 | 466 |
| References .....                                                               | 467 |

### ***Abbreviations***

CARP – Common Address Redundancy Protocol  
CoAP – Constrained Application Protocol  
DDS – Data Disturbing Service  
DODAG – Destination Oriented Directed Acyclic Graph  
DSL – Dictionary Specification Language  
IEEE – Institute of Electrical and Electronics Engineers  
IoT – Internet of Things  
LPWAN – Low-Power Wide Area Network  
MQTT – Message Queuing Telemetry Transport  
NFC – Near Field Communication  
OFDM – Orthogonal frequency-division multiplexing  
PBCC – Packet Binary Convolutional Coding  
QR – Quick Response Code  
RFID – Radio Frequency IDentification  
RPL – Routing over Low Power and Lossy Networks  
STOMP – Simple (or Streaming) Text Oriented Message Protocol  
UDP – User Datagram Protocol  
WLAN – Wireless Local Area Network  
XMPP – Extensible Messaging and Presence Protocol

## **29.1 IoT-based devices: models and network communication protocols**

### *29.1.1 Types of models for IoT-based devices*

A special role in the technology of the "Internet of Things" (IoT) is played by measuring instruments that provide the transformation of information about the external environment into data for further processing by IoT devices (IoT-based devices). At present, a wide range of measuring devices is used, from elementary sensors (eg, temperature, pressure, illumination), consumption accounting or metering devices (such as smart meters) to complex integrated measuring systems. Within the framework of the concept of the "Internet of Things", it is fundamental to combine measuring devices in the network (such as wireless sensor networks, measuring complexes), which enables the construction of interoperability systems [1, 2].

Nowadays, the IoT device is considered to be any device that can receive indications (data) from the environment and transfer them to a database, where they can be saved and, in some cases, analyzed. Therefore, the necessary condition is to connect devices to the computer network one way or another [1].

There is a sufficient number of classifications of IoT devices created by different manufacturers of both hardware and software. For example, Google has developed its classification of IoT devices for its own Google Smart Assistant platform [3].

To simulate the interaction between the IoT device and the IoT platform, you need to create an abstract description of the device, or so-called information model. On the basis of an information model, a code can be generated in the programming language required for the platform [1].

The information model of the IoT-device should be understood as the model of the object, which is presented in the form of information describing the essential parameters for this object and variables, the links between them, inputs and outputs, and allows you to simulate possible states of an object by submitting input data to the model [3]. Also, the term information model can be regarded as a set of information that characterizes the essential properties and states of the object, process, phenomenon, as well as the relationship with the

surrounding world. The existence of Internet technology is impossible without the existence of information models of IoT devices that need to be linked to the network [1].

One of the open source software tools that can be used to create information models is Eclipse Vorto (Fig. 29.1). This is an open source software tool that allows you to create and manage technologies that are compatible with other systems as well as information models. Information models, in the context of the software for their creation, are understood as a description of the attributes and capabilities of a real device. These models can be managed and shared in Vorto Information Model Repository which are repositories for information models. Also, Vorto allows you to integrate devices on different platforms [4].

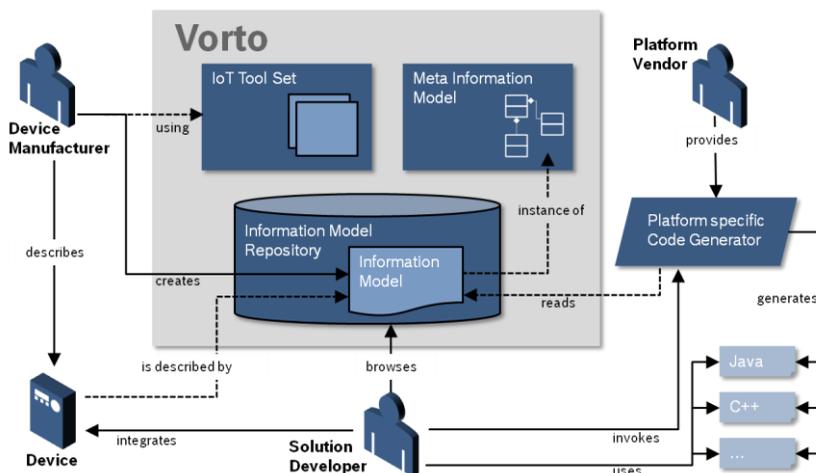


Fig. 29.1 – The structure of the Eclipse Vorto and its components [4]

The benefits of the Eclipse Vorto software are that with the help of it there can be solved the following tasks [5]:

1. *Development of information models of devices (description of devices and their purpose).*

Interoperability is one of the most important criteria in IoT. It is fulfilled with the help of IoT-platforms that are able to integrate devices and create an infrastructure for interoperability. It is important for device providers to enable platform vendors to integrate their devices without

much effort. A device that can be integrated on different platforms can be used in different scenarios. The Eclipse Vorto toolkit lets one create abstract, technological descriptions of devices. These descriptions are read and, thus, can be converted into the formats required to integrate into a particular platform. By providing such a description of the device, the device vendor makes it easy to integrate devices into platforms for which there are Vorto code generators [5].

In order to create an information model using Eclipse Vorto, you need to take the following steps [4]:

- a) download Eclipse Vorto Perspective;
- b) open Model Repository Browser;
- c) select model and generator (s), for example, Constrained Application Protocol (CoAP).

### *2. Creating platforms.*

There are plenty of smart devices in the market. The client should not be restricted by the devices of specific vendors, for example, the platform which its IoT environment is based on may not support other devices. At the same time platform vendors should integrate as many different types of IoT devices on their platform. Eclipse Vorto allows you to create a platform-specific code generator that transforms information models into the formats required to integrate into a specific platform. After implementing an appropriate code generator, it is easy to integrate the IoT devices that are available in the Eclipse Vorto repository [1, 4, 5].

### *3. Developing solutions.*

Decision makers that integrate IoT devices into specific platforms must write the code using information about the corresponding IoT device. The Eclipse Vorto software code generator infrastructure lets one do this automatically, which greatly reduces the programmer's or developer's time [5].

The process of constructing the information model of an IoT device includes the definition of the sources of personal data and their formats, the construction of a model and data structure and their further analysis (Fig. 29.2).

The corresponding technology of the independent abstraction of the IoT device creates a standard: the information model can be transformed into different formats, that is, the specific components of the model serve as a base for integration into different platforms.

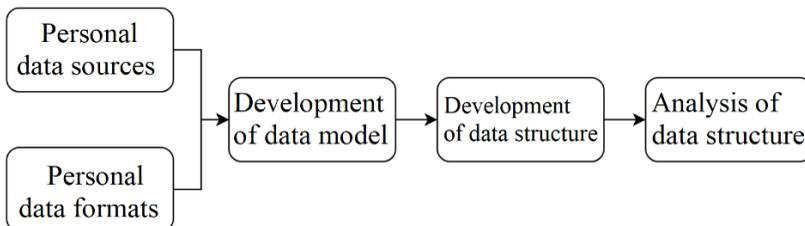


Fig. 29.2 – Scheme of the process of creating an information model of the IoT device

### 29.1.2 Tools and means for the development of information models

There are a large number of resources available on the market of software tools and means that allow the development of information models, track and process information from IoT devices. Most of them have some of the following features [2]:

- adding an informational model;
- displaying received data in real time in the form of diagrams;
- realization of data transmission using the Message Queuing Telemetry Transport (MQTT) protocol.

The corresponding resources which have this feature are ThingBoard (Fig. 29.3), Ubodots IoT dashboards (Fig. 29.4), Node-Red-UI, freeboard.io, and others [6].



Fig. 29.3 – The ThingBoard Web Application Interface [6]

These software tools enable adding information models to IoT devices and visualizing received data in real time, but they do not use the means of data mining (means of intellectual data analysis).

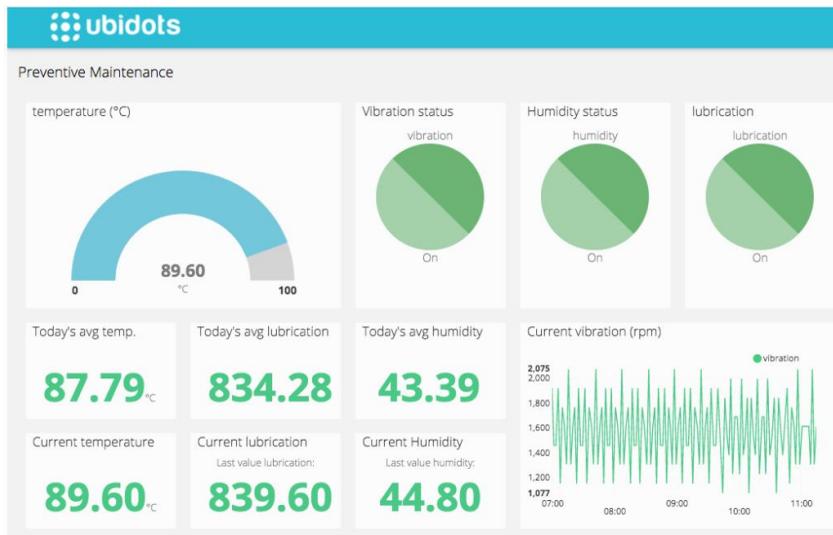


Fig. 29.4 – Ubidots IoT dashboards web application interface [6]

The Eclipse Vorto software tool is a tool for creating information models for a variety of IoT devices and generating the code of the relevant models for different types of IoT platforms [5].

Vorto solves the problem of describing IoT devices from different manufacturers in the form of information models. Such models are described at the abstraction level, thus they are not connected with any technological platform (Fig. 29.5).

For each functional block, the set of operations which it can perform and the set of events it processes is determined. The information model of functional blocks is created using the Dictionary Specification Language (DSL) [1, 4].

The convenience of using Vorto components is that users are not restricted to the formats of IoT devices or IoT platforms. Vorto offers mechanisms that let users use code generators that can convert the description of Vorto IoT-devices to various formats [5]. Available

formats include programming languages, such as Java and C ++, as well as formats for documentation purposes. It is also possible to transform models into formats defined by standardization organizations and industry consortia.

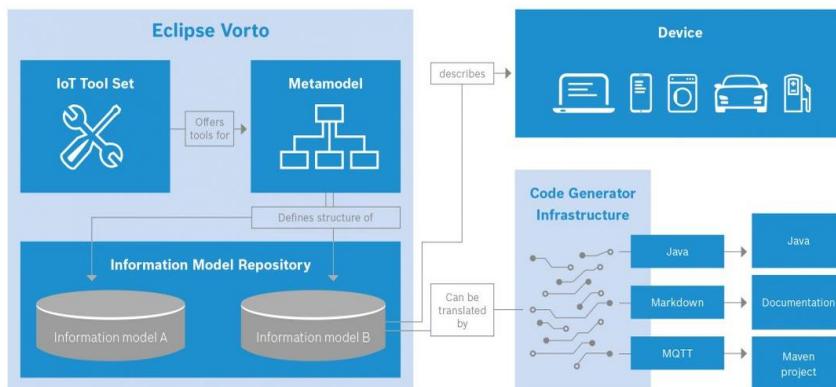


Fig. 29.5 – The scheme of the process of creating an information model by Eclipse Vorto

The information model of the Vorto IoT device contains various types of functional blocks, data types, and transfer units (Fig. 29.6).

A functional block provides an abstract view of the necessary functions of the IoT device for using with a specific application. Thus, it is a consistent set of functions. The corresponding set can be associated with a specific component of the device, for example, a battery, a global sensor, or a switch. Functional blocks have properties (attributes) and behavior patterns (operations). To ensure compatibility between different resources, the description of the functional block should be as abstract as possible. Functions which are specific to the device and cannot be modeled abstractly can be encapsulated, for example, in the functional unit of the device [4, 5].

Data types and conversion units are the smallest units of the model. They represent the states or properties of the model elements.

Information models are well structured and standardized but have some limitations. For instance, as the impossibility of implementing the logic of data transmission over protocols and receiving commands from

a server or monitoring system. Consequently, these models have only informational character.

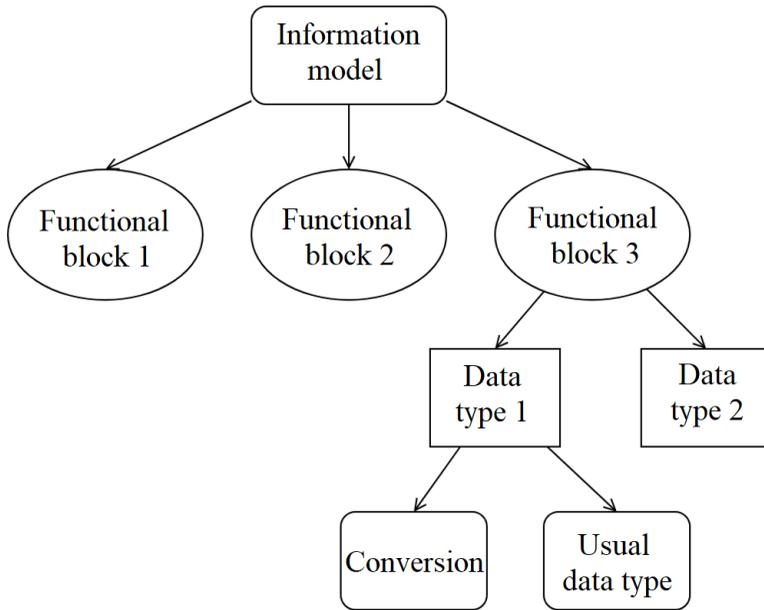


Fig. 29.6 – Structure scheme of the information model

Below is an example of the information model of the IoT device developed for controlling the indoor climate using the Eclipse Vorto.

*Functional temperature sensor unit:*

```

functionblock TemperatureSensor {
 status {
 mandatory currentTemperature as float with {
 measurementUnit : TemperatureUnit.Celsius
 }
 "Indicates the current temperature in Celsius."
 }
}

```

*Generated Data Types (Listing):*

```
description "Enum containing temperature measurement units."
enum TemperatureUnit {
 Celsius "Measurement unit: degree celsius.",
 Fahrenheit "Measurement unit: degree fahrenheit."
}
```

*The information model of the device for obtaining the data of the environment conditions of the room:*

```
infomodel EnvironmentState {
 functionblocks {
 humiditysensor as HumiditySensor
 temperaturesensor as TemperatureSensor
 }
}
```

### ***29.1.3 Network communication protocols for IoT-based devices***

After completing the stage of creating the information model of the IoT device and generating the code for the selected IoT platform, it is necessary to determine the protocols for connecting IoT devices to the network [1, 2].

Consider the following protocol options for connecting devices on the network [7, 8]:

#### ***1. MQTT.***

The message queuing telemetry transport protocol [2] was created about 15 years ago to monitor distant sensor nodes and was designed to save both energy and memory. The relevant protocol is based on the Publish-Subscribe communication model, in which the intermediary is responsible for transmitting messages to the clients of MQTT. It allows multiple clients to post messages and receive updates on various topics from the central server. It looks like subscribing to the YouTube channel, where you get a notification each time a new video is published.

Using the MQTT, the connected IoT device can subscribe to any number of topics that are hosted by the MQTT. Whenever another device publishes data on any of these topics, the server sends messages to all connected users of these topics, warning them of newly available data. The MQTT protocol works on embedded devices and mobile

platforms at the same time being connecting to scalable web servers over wired or wireless networks. It is useful for connections with remote embedded systems where network bandwidth is low or the connection is unpredictable. It is also ideal for mobile applications that work with small amounts of transmitted information. For example, the high performance and reliability of the MQTT protocol are demonstrated by Facebook Messenger, Amazon IoT (AWS-IoT), IBM Node-Red and others that use it to serve millions of people every day [3].

### 2. *CoAP*.

Constrained application protocol is the Internet Protocol for restricted devices (defined in RFC 7228). The CoAP is intended for application between devices in the same limited network, between devices and shared nodes on the Internet, and between devices on various constrained networks which the Internet connects to. This is an application layer protocol that is designed for network restricted IoT devices, such as nodes of network sensors. It can work on most devices that support User Datagram Protocol (UDP) or an analog of UDP. It implements the architectural style of Representational State Transfer (REST), which can be transparently mapped to HyperText Transfer Protocol (HTTP). However, CoAP also provides features that go beyond HTTP, such as national push notifications and group communication. Unlike MQTT, CoAP does not require the work of a broker server. On the implementation side, the Eclipse Californium project covers the implementation of the Java CoAP protocol, including Datagram Transport Layer Security (DTLS) security support. There is also a MicroCoAP project that provides the implementation of the CoAP for Arduino [2, 9].

### 3. *Bluetooth and Bluetooth Low Energy (BLE)*.

The Bluetooth protocol provides wireless communication through the radio frequency (2.4 GHz spectrum in the ISM band) using the standard that was originally used to exchange files between mobile phones. Bluetooth, as a rule, is divided into two categories [2].

Bluetooth Classic is designed to work on high-speed IoT devices, for example, streaming audio data wirelessly [3].

Bluetooth Smart or Low Energy/BLE is designed for low-battery IoT devices that carry small volumes of packet data.

Currently, Bluetooth should be understood as a complex network protocol developed specifically for the IoT. It provides a stable connection with a low power consumption. An obvious example is a connection between Bluetooth and BLE smartphone and fitness tracker. With a constant connection and a small amount of battery tracker, wireless data transmission is at a high level [10, 11].

So, today there are many different protocols and industry standards for connecting IoT devices, such as the above and Wi-Fi WebSockets, ZigBee, LoRA, Simple RF, Extensible Messaging and Presence Protocol (XMPP), Radio Frequency IDentification (RFID), Near Field Communication (NFC), etc. Nevertheless, the choice must be based on the requirements of the IoT system. For example, the MQTT protocol is extremely powerful and will be effective in developing corporate IoT systems. In the case of CoAP, a developer can create their own limited network environment and transfer information to the Internet through a proxy server. If the system does not provide Internet connection or large volumes of data transmitted, then Bluetooth Low Energy might be a better choice [12].

## **29.2 Technologies for data processing in IoT-based systems**

### ***29.2.1 Technologies for data collection and analysis from IoT devices***

Data transmission (data exchange, digital transmission, digital communication) is a physical process of data transfer (digital bit stream) in the form of signals from point to point or from point to multiple points. As a rule, this is done by means of telecommunication through the data transmission channel, for further collection and processing by means of computer facilities [2, 3].

Data-capturing Device is a physical device that has read/write functions and the ability to interact with physical things. The interaction may be carried out indirectly by means of data transfer devices or directly by data carriers connected to physical objects. A general purpose device has built-in processing and communication capabilities and can exchange data using wired or wireless technologies [1].

The unique aspect of IoT, compared to other network systems, is obviously the presence of a plurality of physical things and devices

other than computing devices and data processing devices. Fig. 29.7, adapted to the recommendations of Y.2060, depicts the types of devices in the ITU-T model. The model considers the IoT as a network of devices closely related to things. Sensory and actuating devices interact with physical things in the environment. Data acquisition devices read or write data on physical things by interacting with data transfer devices or data carriers [1].

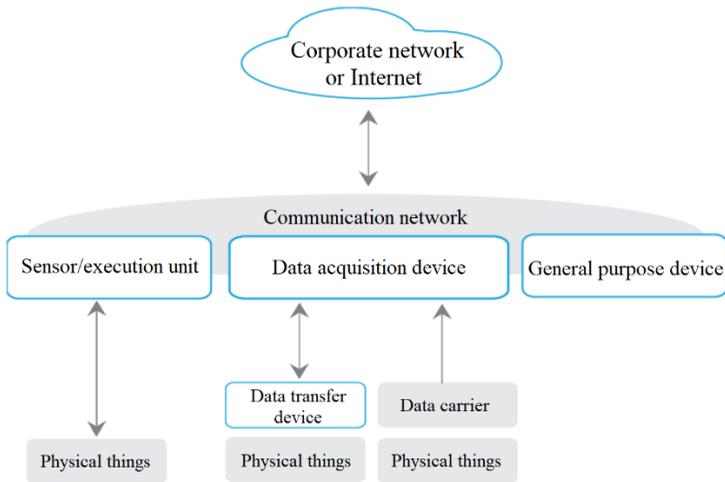


Fig. 29.7 – Model for collecting and processing data in IoT networks

Recommendation Y.2060 states that technologies used to interact between data acquisition devices and data transfer devices or data carriers include radio frequency, infrared, optical and galvanic innervation [2, 12]:

- RFID;
- infra-red labels used for military purposes, medical and other environments where you need to track the location and movement of personnel. It also reflects the infrared labels (stripes) on the military form, which work with the help of batteries and emit identification information. Remote controls used at home or in other environments for controlling electronic devices can also be easily integrated into the IoT;

– barcodes and Quick Response (QR) codes can serve as examples of optical data storage media;

– an example of galvanic innervation can be medical implants that use electrically conductive properties of the human body [9]. In the course of communication between the implant and the surface of the body, the galvanic pair transmits signals from the implant to the electrodes. This circuit requires very little energy, which reduces the size and complexity of the implanted device.

The last type of device in Fig. 29.7 is general purpose devices. They have the ability to process data. A good example is the "smart home" technology, which can integrate virtually any device in the IoT network for centralized or remote control [12, 13].

Within the computer or the communication device, the distances between the different units are too short. Thus, the normal practice is to transfer data between subdivisions using a separate wire. There are parallel and serial (consecutive) data transfer modes. The parallel operation mode results in minimal delays when transmitting each signal. The graphic representation of parallel transmission can be seen in Fig. 29.8. In the case of a parallel transmission, all data bits are transmitted simultaneously to separate lines of communication  $n$  lines are used to transmit  $n$  bits. Thus, every bit has its own line. All  $n$  bits of the same group are transmitted with each clock pulse from one device to another, that is, several bits are sent with each clock pulse. A parallel transmission is used for short-term communication. As shown in Fig. 29.8, for the transmission of 8-bit data from the sender to the receiver, there are used eight separate channels [2,3].

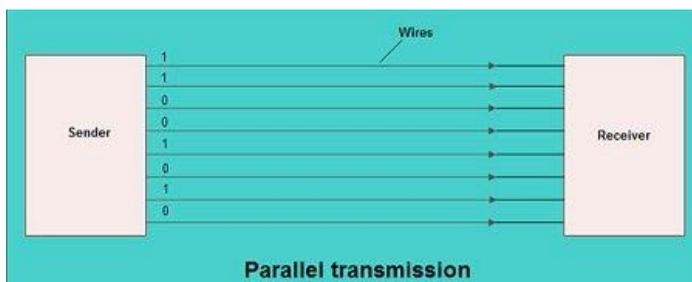


Fig. 29.8 – Parallel data transmission

The *advantage* of parallel transmission is the fast way of data transmission since several bits are transmitted simultaneously with one clock pulse [2].

A *disadvantage* is an expensive way of transmitting data since it requires  $n$  rows to transmit  $n$  bits simultaneously [2, 6].

During the consecutive data transfer between two separate devices, especially if the distance is more than a few kilometers, for cost reasons, it is more economical to use one pair of lines. Data is transmitted as one bit at a time, using a fixed time interval for each bit [6]. In a serial transmission, different bits of data are transmitted serially one after another. To transmit data from the sender to the receiver, only one communication line is required, not  $n$  lines. Thus all bits of data are transmitted through one line in series. Only one bit is sent in a single-pulse serial transmission. As shown in Figure 29.9, we assume that the 8-bit data 11001010 must be sent from source to receiver. Then the smallest significant bit (LSB) 0 will be passed rather to the first, then the other bits. The most significant bit (MSB), i.e. 1, will be transmitted at the end through one link line. The internal circuitry of the computer transmits data in parallel. Therefore, in order to convert these parallel data into consecutive data, there are used converter devices. These devices convert the parallel data into consecutive data on the sender's side so that they can be transmitted through one line. On the receiver side, the received consecutive data are again converted into a parallel form [1].

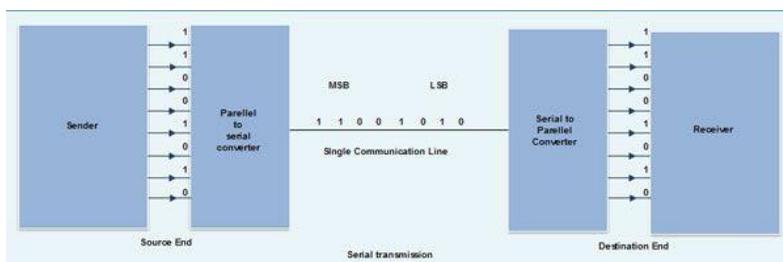


Fig. 29.9 – Consecutive data transmission

*The advantage* of a consecutive transmission is the use of a single line of communication. It reduces the cost of the transmission line compared to the parallel transmission.

Among the *shortcomings* of the successive transfer there are the following ones [6]:

- the use of conversion devices at the initial and final stages can lead to an increase in the total cost of transmission;
- this method works slower in comparison to parallel transmission since bits are serially transmitted one after another.

### ***29.2.2 Technologies for data processing***

Currently, there is a small amount of data processing technology from IoT devices. All of them are one way or another integrated into specific data processing devices or platforms.

Let's consider several well-known data processing technologies.

IBM's Watson Internet of Things is one of the most prominent technologies for cognitive processing of data from IoT devices. Developed by scientists, IBM technology of the Internet of things has unique capabilities in the field of machine learning and automatic processing of data coming from several sensory devices. It enables complex analysis and provides an appropriate automatic response in accordance with the aims of the object [14]. The Japanese corporation, Panasonic has announced their plans to use Watson's cloud-based IoT platform and is now examining options for integrating its sensors and smart devices with this platform. The connection of the video surveillance system, glass breakdown sensors, windows and doors opening, motion, etc. to IBM's cognitive computing system will optimize and make smart Panasonic homes even more intelligent. For example, the security system will not react if neighboring children get into the courtyard to fetch their own ball, while it will still be effective in case of an intruder penetration [15].

North Star BlueScope Steel, a manufacturer of rolling steel for the construction industry, will begin using the Watson Internet of Things cognitive technology and native devices to create innovative solutions to protect workers in extreme conditions. Employees working in difficult industrial conditions are exposed to various risks on a daily basis: thermal, chemical and toxic influences, open fire, mechanical

contact with industrial equipment. In 2017, there were registered almost 3 million industrial injuries. At the same time, there are no practical ways to check the mandatory safety and personal protective equipment used in potentially hazardous conditions. The use of native devices that collect information about different metrics, combined with Watson processing, allows you to transfer relevant information in real time to the company's management when potentially dangerous conditions arise. For example, a company may receive a combination of body temperature, accelerated pulse rate and real estate within minutes, which may indicate a heat shock with a possible lethal outcome. The same indicators may seem insignificant if taken separately [3].

In many IoT-systems, a distributed network of IoT devices can generate large volumes of data. For example, oil fields and refineries can generate up to terabytes of data daily. An airplane can generate several terabytes of data per hour. Instead of storing all of these data permanently (or at least for a long time) in a centralized repository accessible to IoT applications, it is often more appropriate to perform most of the data processing closer to IoT devices. Therefore, the task of the level of peripheral computing (edge computing level) is the transformation of network data streams into information, suitable for storage and higher-level processing. Processing elements at this level can deal with large volumes of data and carry out data conversion operations, which result in a much lower volume of storage. Published Cisco document on the IWF model [1, 3] contains the following examples of operations (processing) at the level of peripheral computing:

- analysis of data on criteria of belonging to processing at a higher level;
- reformatting data for the same high-level processing;
- processing of cryptographic data with an additional context;
- reduction and/or summarization of data for further high-level processing.

The processing elements at this level correspond to general purpose devices in the ITU-T model (Fig. 29.7). As a rule, they are physically deployed on the edge of the IoT network, that is, next to sensors and other data generation devices. Thus, part of the basic

processing of large volumes of data is removed from the application programs of IoT which are located centrally.

Processing at the peripheral level is sometimes called fuzzy calculations (Fog Computing). Fuzzy calculations and fuzzy services are expected to become an excellent feature of the IoT. Fuzzy calculations represent a trend in modern network technologies, the opposite of cloud computing. In cloud computing, a large amount of centralized storage and data storage resources is available to distributed users through cloud-based network structures for a relatively small number of users. In the fog computing, a large number of individual intellectual objects communicate with fuzzy network structures that perform computations and store resources, along with peripherals in the IoT. Fog Computing solves the problems that have appeared as a result of the operation of thousands or millions of "smart" devices, including security, privacy, network constraints, and delays. The term "Fog Computing" is chosen because the fog spreads over the earth, while the clouds are high in the sky [2, 3, 6].

### ***29.2.3 Methods of management and forecasting***

Forecasting is the process of making predictions of the future based on past and present data and most commonly by analysis of trends. A commonplace example might be estimation of some variable of interest at some specified future date. Prediction is a similar, but more general term. Both might refer to formal statistical methods employing time series, cross-sectional or longitudinal data, or alternatively to less formal judgmental methods. Usage can differ between areas of application: for example, in hydrology the terms "forecast" and "forecasting" are sometimes reserved for estimates of values at certain specific future times, while the term "prediction" is used for more general estimates, such as the number of times floods will occur over a long period [1].

Qualitative forecasting techniques are subjective, based on the opinion and judgment of consumers and experts; they are appropriate when past data are not available. They are usually applied to intermediate- or long-range decisions. Examples of qualitative forecasting methods are informed opinion and judgment, the Delphi method, market research, and historical life-cycle analogy [2].

Quantitative forecasting models are used to forecast future data as a function of past data. They are appropriate to use when past numerical data is available and when it is reasonable to assume that some of the patterns in the data are expected to continue into the future. These methods are usually applied to short- or intermediate-range decisions. Examples of quantitative forecasting methods are last period demand, simple and weighted N-Period moving averages, simple exponential smoothing, poisson process model based forecasting [2] and multiplicative seasonal indexes. Previous research shows that different methods may lead to different level of forecasting accuracy. For example, GMDH neural network was found to have better forecasting performance than the classical forecasting algorithms such as Single Exponential Smooth, Double Exponential Smooth, autoregressive integrated moving average (ARIMA) and back-propagation neural network [3].

Limitations pose barriers beyond which forecasting methods cannot reliably predict. Many events and values cannot be forecast reliably. Events such as the roll of a die or the results of the lottery cannot be forecast because they are random events and there is no significant relationship in the data. When the factors that lead to what is being forecast are not known or well understood such as in stock and foreign exchange markets forecasts are often inaccurate or wrong, as there is not enough data about everything that affects these markets for the forecasts to be reliable. In addition, the outcomes of the forecasts of these markets change the behavior of those involved in the market further reducing forecast accuracy [6].

Uprise of IoT have revolutionized major industries that includes industries, agriculture and healthcare and have expanded its scope not only to build smart cities but also to accurate forecasting of weather. Weather forecasting itself has its direct or indirect influence on various sectors of economy like Energy, transportation and other business and thereby, making this forecasting as a key element in an economy's growth. Remote sensing technology have opened the gates for real time analysis of weather data and have transformed the way that was used to collect and analyse weather data and build a strong database for reliable weather forecasts [3].

Let's discuss few means by which atmospheric data is collected. IoT enabled weather systems are designed to collect data from various vehicles on the road, vehicles moving on the road will wirelessly communicate the weather and road condition data that is inclusive of air temperature, barometric pressure, visibility or light, motion and other data needed. This data helps to build more accurate forecast and provide flexible real time monitoring at different time horizon. Sensors are installed on windshields, wipers and tyres of car. These sensors in integration with IoT help in collecting weather data which is further pooled in cloud for analysis [3].

Companies like IBM, Rainmachine and others are working towards expansion of IoT enabled weather forecasting [2].

As mentioned earlier, accuracy of weather forecasting directly or indirectly influences other sectors of economy to a great extent, it thus raises the need of a system that facilitates higher accuracy of real time monitoring and future weather prediction. Below you can have a look at key sectors that are benefited with IoT weather forecasting technology. *Agricultural process* i.e. preparation of soil, sowing, irrigation, harvesting and storage of crops is directly dependent on weather condition leaving farmers vulnerable to weather hazards. Development and expansion of IoT technology for weather forecasting will deliver vital weather prediction to farmers and accordingly farmers may use the intelligence to improve their crop fertility and cost along with taking essential steps to diversify weather hazards. Timely and accurate delivery of weather forecast will ensure higher productivity and lower the risk of weather hazard. We are well aware about uncertainty of unpleasant weather and risk factor attached to it in *transportation*. On successful installation of remote sensors on every vehicle moving on road. It would communicate every minor detail for analysis of weather change allowing the real time weather monitoring and forecasting report to cover even minute details like temperature, fog, road condition, light, flood, stormy and other condition that would add up to reliability and accuracy of the report [2, 6].

## 29.3 Protocols and standards for data transfer between IoT-based devices

### 29.3.1 Protocols for data transfer

Protocols for the exchange (transmission) of data between IoT devices are divided into groups depending on the area of the network on which they are used. There are the following areas: sensor node (Data Disturbing Service (DDS) protocol), sensor node-server (CoAP, MQTT, XMPP, Simple (or Streaming) Text Oriented Message Protocol (STOMP) protocols), server-server (Advanced Message Queuing Protocol (AMQP)). Consider some data transfer protocols between IoT devices over the Internet [1].

*The DDS protocol* implements a publication-subscription template for sending and receiving data, events, and commands among end nodes. Sender nodes create a "topic" (for example, information about temperature, location, pressure) and publish templates. DDS delivers created templates to nodes interested in relevant topics. UDP is used as a transport protocol. Also, DDS allows one to manage quality of service (QoS) parameters [2].

*The CoAP protocol* from the user's perspective is similar to the HTTP protocol but has a small header size that is suitable for networks with restricted capabilities. It uses client-server architecture and is suitable for conveying the state of the site to a server (GET, PUT, HEAD, POST, DELETE, CONNECT). As a transport protocol, UDP is used [3].

*The XMPP protocol* has long been used on the Internet for real-time messaging. The eXtensible Markup Language (XML) format is suitable for usage in IoT networks. It works on the publisher-subscriber and client-server architecture. It is also used to address devices in small networks (addressing the look "name@domain.com") [2].

*The MQTT protocol* collects data from a plurality of nodes and transmits it to the server. It is based on a publisher-subscriber model using an intermediate server-broker. Transmission Control Protocol (TCP) is used as a transport protocol. On the basis of MQTT, there was created a specialized protocol MQTT-SN for sensor networks [3].

*Routing protocols.* This section there are described some standard and non-standard protocols which are used to route data to IoT

applications. It should be noted that there is a conditional division of the network layer into two sublevels: a routing layer that processes packet transfers from source to destination, as well as an encapsulation layer that generates packets [6].

Routing over Low Power and Lossy Networks (RPL) is a protocol that can support various data transfer protocols. It creates a Destination Oriented Directed Acyclic Graph (DODAG) that has one route from each end node to the base node. First, each node sends a message about the information model of the IoT device, representing the base node. This message is distributed over the network, and the entire DODAG graph is being gradually built. During the broadcast, the node transmits to all information about its location on the network (DAO). This DAO message is actual for the base node that makes a decision on the place of departure, depending on the destination. When a new node wants to connect to a network, it sends a request and the base node answers with a confirmation message. The RPL nodes may be without states, which is the most common practice, or they can be with states [2].

The CORPL (cognitive RPL) protocol is positioned as an extension of the RPL developed for cognitive networks and uses the generation of the DODAG topology, but with two new modifications. CORPL uses conditional sending for the packet forwarding by selecting multiple conveyors (a set of conveyors) and coordinates between the nodes to select the next best step for forwarding the packet. DODAG is built in the same way as in RPL [14].

Common Address Redundancy Protocol (CARP) is a distributed routing protocol designed for underwater communications. It can be used for IoT because of its light packs. It takes into account the quality of communication, which is calculated on the basis of a successful transfer of data collected from neighboring sensors. There are two scenarios: network initialization and data transfer. In the network initialization, the HELLO packet is transmitted from the host to all other nodes in the network. During data redirection, the packet is transmitted from the sensor to the host in hop-by-hop-fashion mode. Each next step is determined by itself. The main problem of CARP is that it does not support the multiple uses of previously collected data. The improvement of CARP was done in E-CARP, allowing the host to

store previously received data. When new data is needed, E-CARP sends a ping packet that matches data from sensor nodes [15].

### ***29.3.2 Standards for data transfer***

Currently, there are several standards for data transmission in IoT networks. Let's examine some of them in detail.

The Wireless USB Standard is a wireless data standard developed by the Wireless USB Promoter Group. In September 2010, the Wireless USB 1.1 specification was completed. It involves increasing data rates, as well as the support of higher frequencies - up to 6 GHz and above. In the development, much attention was paid to improving energy efficiency. Devices made in accordance with specification 1.1, use less power in idle mode. Wireless USB 1.1 supports NFC technology, which simplifies the configuration and operation of Wireless USB devices.

Wireless USB Standard is intended to become a replacement for traditional USB drives. Typical devices include a keyboard, a mouse, a camera, a printer, external drives, etc. Wireless USB can also be used to easily shared usage of printers that do not have a standard network interface or are not connected to a print server [16-18].

The transmission parameters correspond to the standard USB version 2.0, but the bandwidth depends on the distance between the devices. At a distance of up to 3 meters, the data rate can theoretically reach 480 Mbps. At a distance of 10 meters - only up to 110Mbps (under optimum conditions). Wireless USB is designed for operation in the frequency range from 3.1 GHz to 10.6 GHz. Data transmission is encrypted with the help of AES-128 / CCM [6].

The Narrow Band Internet of Things Standard (NB-IoT) is a mobile communication standard for telemetry devices with low volumes of data exchange. It was designed by the 3GPP consortium in the framework of working on the standards of mobile networks of the new generation. The first working version of the specification was presented in June 2016. It was made for connecting a wide range of stand-alone devices to a digital communications network, for example, medical sensors, meters of resources consumption, devices of a smart home, etc. [12]. NB-IoT is one of the three IoT standards developed by 3GPP for mobile communications: eMTC (enhanced machine-type

communication), NB-IoT, and EC-GSM-IoT [2]. The eMTC standard has the highest bandwidth and is based on LTE equipment. The NB-IoT standard can be used both on LTE mobile devices and separately, including GSM. The EC-GSM-IoT standard provides the lowest bandwidth and extends beyond the GSM network. Among the benefits of NB-IoT there are the following ones [17]:

- flexible power management of devices (up to 10 years in a network of a battery with the capacity of 5 W\*h);
- huge capacity of the network (hundreds of thousands of connected IoT devices per base station);
- low cost of IoT devices;
- optimized for increasing the sensitivity of signal modulation.

A comparative analysis of some data transmission standards is given in Table 29.1.

Table 29.1 – Comparative analysis of LTE Cat 0, eMTC, NB-IoT, EC-GSM-IoT data transfer standards

|                                 | <b>LTE Cat 0</b>    | <b>eMTC</b>         | <b>NB-IoT</b>           | <b>EC-GSM-IoT</b>      |
|---------------------------------|---------------------|---------------------|-------------------------|------------------------|
| <b>Downlink Peak Rate</b>       | 1 Mbit/s            | 1 Mbit/s            | 250 kbit/s              | 474 kbit/s or 2 Mbit/s |
| <b>Uplink Peak Rate</b>         | 1 Mbit/s            | 1 Mbit/s            | 250 kbit/s or 20 kbit/s | 474 kbit/s or 2 Mbit/s |
| <b>Latency</b>                  | not deployed        | 10ms-15ms           | 1.6s-10s                | 700ms-2s               |
| <b>Number of Antennas</b>       | 1                   | 1                   | 1                       | 1-2                    |
| <b>Duplex Mode</b>              | Full or Half Duplex | Full or Half Duplex | Half Duplex             | Half Duplex            |
| <b>Device Receive Bandwidth</b> | 1.4 – 20 MHz        | 1.4 MHz             | 180 kHz                 | 200 kHz                |
| <b>Receiver Chains</b>          | 1                   | 1                   | 1                       | 1-2                    |
| <b>Device Transmit Power</b>    | 23 dBm              | 20/23 dBm           | 20/23 dBm               | 23/33 dBm              |

There is envisaged a great popularity of IoT devices with the ability to use mobile communication. In this case, the cost and maintenance costs become critical. One way to save money is not to

install a physical SIM card. The GSMA consortium in 2016 adopted the specification of Embedded SIM (eSIM) / Remote SIM Provisioning (RSP) for that purpose. The eSIM standard allows one to integrate the SIM card functional into the electronics of the modem, and the RSP describes the infrastructure and algorithms for interoperating trusted emission centers of SIM parameters, the mobile operator and the communication service user [18].

The Low-Power Wide Area Network (LPWAN) is a standard wireless low-bandwidth data transmission technology developed for distributed telemetry, inter-engineer interconnection networks, and IoT. LPWAN is one of the wireless technologies that provides a data collection environment for various equipment: detectors, meters, and sensors [1]. The LPWAN standard focuses on IoT systems that require guaranteed low data transmission, the ability of network IoT devices to last long using standalone power sources, and a large area coverage of the wireless network. The main areas of application of LPWAN are wireless sensor networks, automation of data collection on accounting devices, industrial monitoring and control systems [3].

For wireless data transmission, the following characteristics, such as efficiency, fault tolerance, adaptability, possibility of self-organization, play an especially important role in the IoT. The main interest then is Institute of Electrical and Electronics Engineers (IEEE) 802.15.4, an access control for the organization of energy efficient personal networks, and is the basis for such protocols as ZigBee, WiFi, Bluetooth, 6LoWPAN. IEEE 802.11 is a set of standards for communication in the Wireless Local Area Network (WLAN) of the frequency ranges 2.4, 3.6 and 5 GHz. They have been developed and supported by the LAN / MAN (IEEE 802) Standards Committee of the IEEE, which determine the interaction of wireless computer networks. The basic version of IEEE 802.11 (2007) has undergone the additions. These standards provide the basics of wireless network products that use the Wi-Fi brand. Let's consider some of them [2, 7]:

- IEEE 802.11a is a wireless LAN standard based on wireless data transmission in the 5 GHz range. The range is divided into three non-overlapping channels. The maximum data transfer rate is 54 Mbps, with speeds of 48, 36, 24, 18, 12, 9 and 6 Mbps also available;

– IEEE 802.11b+ is an upgraded version of the 802.11b standard that provides increased data rates. It differs from the original version of the Packet Binary Convolutional Coding (PBCC), doubling the maximum speed (up to 22 Mbit/s). Also announced solutions to productivity, increased to 44 Mbps;

– IEEE 802.11g is a WLAN standard based on 2.4 GHz wireless data transmission. In order to increase the data rate at a channel width similar to 802.11b, an Orthogonal frequency-division multiplexing (OFDM) method, or a PBCC method, is used;

– IEEE 802.11e (QoS) is an additional standard that ensures a guaranteed quality of data exchange by rearranging the priorities of different packages; it is required for such stream services as Voice over Internet Protocol (VoIP) or Internet Protocol Television (IPTV);

– IEEE 802.11n is a modern wireless LAN standard based on wireless 2.4 GHz data transmission. The 802.11n standard significantly exceeds the previous 802.11b and 802.11g standards by providing data rates at Fast Ethernet level. The main difference from the previous versions is the addition of the MIMO protocol (multiple-input-multiple-output) to the physical layer (PHY). The theoretical speed can be 150 Mbps;

– IEEE 802.11ac is a new standard for wireless local area Wi-Fi networks at frequencies of 5-6 GHz. If both IoT devices support this technology, data transfer speed may be greater than 1 Gbit/s (up to 6 Gbit/s 8x MU-MIMO). The standard requires up to 8 MU-MIMO antennas and 80 or 160 MHz channel extensions;

– IEEE 802.11ax is a follower of the 802.11ac standard. The operating ranges of the standard are 5 GHz and 2.4 GHz. The standard is still being developed and has the goal of providing a bandwidth of 10 Gbit/s.

### ***29.3.3 Cybersecurity of IoT-based devices***

The development of secure IoT-systems includes several levels that combine important IoT-security architectures at four different levels: device level, communication layer, cloud level, and life-cycle management level [19-21].

*The device level* refers to the hardware level of the IoT system, that is, the physical product. The ODMs and OEMs of IoT devices

increasingly integrate security features into their hardware and software of the device to enhance security directly at the IoT device level. Some manufacturers introduce trusted platform modules (TPMs) that act as a guarantor of trust, protecting confidential information and credentials (without releasing encryption keys on the chip). Even physical protection (for example, a full metal shield that covers all internal circuits) can be used to protect against interference.

*The level of communication* refers to the technologies of connection of IoT devices in the IoT-network, that is, the environment in which data is reliably transmitted/received. Confidential data is passed through physical level (e.g., WiFi, 802.15.4 or Ethernet), network level (eg IPv6, Modbus, or OPC-UA) or application level (eg MQTT, CoAP or web connectors). Unsafe communication channels can be susceptible to intruders, such as "men-in-the-middle". Data-oriented IoT-security solutions provide secure encryption of data during transmission. Even if they are intercepted, they will be useless for everyone, except for users (ie, people, devices, systems or applications) that have the correct encryption key. Firewalls and intrusion prevention systems are designed to analyze specific traffic flows (such as non-IoT protocols) which are embedded in IoT devices. They are increasingly used to identify unwanted intrusions and prevent harmful connection at the communications level [20].

*The cloud level* refers to software support for the IoT solution, which means that data coming from devices is analyzed and interpreted to generate statistics and perform actions. Security has always been one of the main topics for discussion when assessing the risk of using cloud and built-in solutions. Cloud providers are expected to provide secure and efficient cloud services by default, and protection from severe data breaches or solving problems with idle mode are becoming normal [21].

Important cyber security features in IoT [2]:

- the information stored in the cloud must be encrypted in order to avoid attack;
- checking the integrity of other cloud platforms or third-party programs that are connected with your cloud services;
- digital certificates for authentication;

- monitoring activity for tracking, registering and detecting suspicious activity;
- IoT devices and applications require regular security patches to protect against new threats.

Intruders can intercept or change the behavior of smart Home IoT devices in many ways. Some methods require physical access to the device, which makes the attack more difficult to carry. Other attacks can be done via the Internet from a remote location. The following is a different attack level scenario based on the access level.

An attacker who has access to a local home network may perform various attacks on the IoT device. There are, as a rule, two common access modes: through the cloud and direct connection. Depending on the function, the IoT device can use any of these methods to receive commands [19].

In the case of a cloud attack, a smart home device is constantly connected to the cloud. The device checks the cloud server to see if there are any commands to execute and then downloads its current state. In this case, the attacker will have to execute the Man in the middle (MITM) attack. In order to achieve this, one needs to try to redirect network traffic with network-level attacks, for example by changing Address Resolution Protocol (ARP) or Domain Name System (DNS) settings. A fake certificate can help attackers intercept HTTPS connections. Unfortunately, some IoT devices do not check if the certificate is trusted and belongs to the vendor, they only confirm the connection through HTTPS. Additionally, most devices do not perform mutual SSL authentication, and completely ignore certificate cancellation lists, allowing an attacker to use keys that were received due to data breaches [6].

Some IoT devices use direct connections to communicate with a hub or application on the same network. For example, a mobile application can scan a local area network for new devices and find them by sensing each IP address for a particular port. Another way is to use the Simple Service Discovery Protocol / Universal Plug and Play (SSDP / UPnP) to detect devices. This means that an attacker can do the same thing to find the right device. The most common mistake is the use of unencrypted network communications. Lack of encryption raises serious data privacy issues. Devices can transmit personal data,

registration data or tokens in plain text, allowing the intruder to intercept them [3].

The most common way for users to interact with IoT devices is to use a web browser or smartphone application. More powerful devices have a small web server and allow a user to apply a web interface to send commands. Other devices offer their own program interface (API) which the user can interact with. If a user wants to control devices remotely when they are not at home, they should be able to open the incoming port on the router. This can be done using the UPnP request or can be manually fulfilled by a user [21].

A smart home IoT device can include cloud-based services, depending on the device category. Other cloud systems allow remote control of IoT devices, such as bulbs or boilers. Some vendors even force a user to connect to their server cloud system and do not allow users to locally manage their devices. Companies either provide access to cloud services through an application for smartphones or a web portal where users can log in.

Most services do not block users in their accounts after several login attempts. At the same time, some servers of cloud services do not provide the possibility of two-factor authentication.

Malicious software installed on any IoT device which is connected to the home network may be able to interact with smart home devices and allow hackers to make larger-scale attacks.

For now, there have been no large-scale malware attacks on an IoT device. In addition, malicious programs and cyber-security violations of IoT devices attacking routers, and similar devices have been successfully detected several times.

#### **29.4 Work related analysis**

The issues discussed above can be supplemented by an analysis of the existing works of European partner universities of the ALIOT project on the topic of the section [22]. Models for IoT-based devices and technologies for data processing and transfer are considered in different university-partners, besides University of Coimbra, Leeds Beckett University, Consiglio Nazionale delle Ricerche - Istituto di Scienza e Technologie dell' Informazione "A.Faedo" (ISTI-CNR),

Royal Institute of Technology (KTH) and Newcastle University. So, let's consider the following projects.

An IoT application, such as real-time flood forecasting [6] and warning, requires the integration of machine and social sensors data to provide complementary and corroborative information. This aggregate data can be semantically tagged to generate and distribute events of interest (to particular subscribers).

Also important it is to consider the IEEE 802.11 medium access control protocol uses the distributed coordination function that supports asynchronous data transfer and an optional point coordination function that supports connection-oriented time-bounded data transfer [7]. Cooperative communication has been shown as an effective way to exploit spatial diversity to improve the quality of wireless links. The key feature of cooperative transmission is to encourage single-antenna devices to share their antennas cooperatively such that a virtual and distributed antenna array can be constructed, and, as a result, reception reliability can be improved and power consumption can be reduced significantly [9]. Due to broadcast transmission and unattended nature, and hostile environments a variety of denial of service (DoS) attacks are possible in both Wireless Sensor Networks (WSNs) and ad-hoc networks. Authors developed a formal framework which can automatically verify different wireless routing protocols against DoS attacks exhaustively [19]. Also this paper [12] presents some of the main application requirements for IoT, characterizing architecture, QoS features, security mechanisms, discovery service resources and web integration options and the protocols that can be used to provide them (e.g. CoAP, XMPP, DDS, MQTT-SN, AMQP). As examples of lower-level requirements and protocols, several wireless network characteristics (e.g. ZigBee, Z-Wave, BLE, LoRaWAN, SigFox, IEEE 802.11af, NB-IoT) are presented [8, 13]. WSNs leverage battery-powered embedded devices to sense from and act on the environment. Their characteristics are at odds with the lifetime requirements in monitoring of civil structures. In this paper [17] authors briefly describe the challenges at stake and how to address them, drawing from recent literature and our own real-world experience [18].

This paper [20] presents a quantitative architecture analysis method for the study of architectures of wide-area monitoring and

control systems focusing primarily on the interoperability and cybersecurity aspects. Also this paper [21] presents a distributed intrusion detection system (DIDS) for supervisory control and data acquisition (SCADA) industrial control systems, which was developed for the CockpitCI project.

In this paper [4] an approach to deal with these issues is presented. It makes use of the device description framework Eclipse Vorto. Using Vorto's capabilities to generally describe devices and interfaces and generating code, a proof of concept has been implemented where a smart home platform is connected to an electric vehicle in order to integrate the vehicle as part of a smart home platform. The paper discusses the challenges, introduces the proposed concept and gives some details on the implementation of the exemplary use case [5].

### *Conclusions and questions*

In this section, the materials for module PCM4.2 “Models for IoT-based devices and technologies for data processing and transfer” of PhD course “Development and implementation of IoT-based systems” are presented. They can be used for preparation to lectures and self-learning for lecturers, PhD-students, IoT developers, etc.

Recently, the direction associated with the development and implementation of IoT devices has become very popular and effective. This gave rise to such areas as neural network technologies, cloud and fog computing, control systems, computer vision, etc.

This chapter discusses the basic principles of constructing information models of IoT-based devices and tools for their creation, in particular Eclipse Vorto [5]. Also analyzed network communication protocols for IoT-based devices. In addition, an important component of the IoT network is the choice of technologies for data processing in IoT-based systems and methods of management and forecasting. Also considered the main protocols and standards for data transfer between IoT-based devices. Some attention is paid to cybersecurity in IoT [21].

The considered information models, data transfer protocols and standards, forecasting methods are widely used in all applications of the IoT, for example, home automation, climate control, environmental

monitoring, production automation, agriculture, medical applications, smart transportation, smart traffic, etc [1-3, 6].

In order to better understand and assimilate the educational material that is presented in this section, we invite you to answer the following questions.

1. What is an information model?
2. What is the purpose of the instrumental tool “Eclipse Vorto”?
3. What is the advantage of “Eclipse Vorto”?
4. What is not a component of the process of creating an information model in “Eclipse Vorto”?
5. What components are not included in the information model?
6. How the MQTT protocol is decrypted?
7. What is the frequency of data transfer by Bluetooth?
8. What does the RFID mean in the Y.2060 recommendation?
9. What is not included in the model for collecting and processing data in IoT networks (Y.2060 recommendation)?
10. Which data transmission has its own line for each bit?
11. Which method is not a qualitative forecasting method?
12. What is the Wireless USB Standard?
13. What is the basic version of IEEE 802?
14. What is the standard still being developed and has the goal of providing a bandwidth of 10 Gbit/s?
15. How many levels does IoT cybersecurity include?

### ***References***

1. D. Uckelmann, M. Harrison, and F. Michahelles, *Architecting the Internet of Things*. Berlin: Springer-Verlag, 2011.
2. M. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for Internet of Things: A Survey," in *IEEE Internet of Things Journal*, vol. 3, no. 1, 2016, pp. 70-95.
3. F. Hussain, *Internet of Things: Building Blocks and Business Models*. Cham: Springer, 2017.
4. J. Laverman, D. Grewe, O. Weinmann, M. Wagner, and S. Schildt, "Integrating Vehicular Data into Smart Home IoT Systems Using Eclipse Vorto," *IEEE 84th Vehicular Technology Conference (VTC-Fall)*, pp. 20-26, September 2016.

5. "Vorto Introduction," *Eclipse Vorto*, 2016, [online] Available: <https://www.eclipse.org/vorto/documentation/overview/introduction.html>.

6. R. Ranjan, O. Rana, S. Nepal, M. Yousif, P. James, Z. Wen, S. Barr, P. Watson, P. Jayaraman, D. Georgakopoulos, M. Villari, M. Fazio, S. Garg, R. Buyya, L. Wang, A. Zomaya, and S. Dustdar, "The Next Grand Challenges: Integrating the Internet of Things and Data Science," in *IEEE Cloud Computing*, vol. 5, no. 3, 2018, pp. 12-26.

7. N. Natchimuthu and A. Sajeev, "A communication protocol using a Markov type function for stations in a wireless local area network," *The 8th International Conference on Communication Systems*, pp. 829-833, November 2002.

8. A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies Protocols and Applications," in *IEEE Commun. Surv. Tutorials*, vol. 17, iss. 4, 2015, pp. 2347-2376.

9. I. Morns, O. Hinton, A. Adams, and B. Sharif, "Protocols for sub-sea communication networks," *Conference Proceedings on MTS/IEEE Oceans 2001*, pp. 2076-2082, November 2001.

10. C. Gomez and J. Paradells, "Wireless home automation networks: A survey of architectures and technologies," in *IEEE Communications Magazine*, vol. 48, no. 6, 2010, pp. 92-101.

11. Z. Sheng, K. Leung and Z. Ding, "Cooperative wireless networks: from radio to network protocol designs," in *IEEE Communications Magazine*, vol. 49, no. 5, 2011, pp. 64-69.

12. L. Novelli, L. Jorge, P. Melo, and A. Koscianski, "Application Protocols and Wireless Communication for IoT: A Simulation Case Study Proposal," *The 11th International Symposium on Communication Systems, Networks & Digital Signal Processing (CSNDSP)*, pp. 372-378, July 2018.

13. F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui, and T. Watteyne, "Understanding the Limits of LoRaWAN," in *IEEE Commun. Mag*, vol. 55, no. 9, 2017, pp. 34-40.

14. N. Huynh, V. Robu, D. Flynn, S. Rowland, and G. Coapes, "Design and demonstration of a wireless sensor network platform for substation asset management," in *Open Access Proceedings Journal*, vol. 1, 2017, pp. 105-108.

15. M. Gursu, M. Vilgelm, W. Kellerer, and E. Fazli, "A wireless technology assessment for reliable communication in aircraft," *IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, pp. 51-57, December 2015.

16. H. Ogai and B. Bhattacharya, "Experiments of Wireless Transfer Technology for Communication," in *Pipe Inspection Robots for Structural*

*Health and Condition Monitoring. Intelligent Systems, Control and Automation: Science and Engineering*, vol. 89, 2017, pp. 61-78.

17. L. Mottola, T. Voigt, I. Gonzalez Silva, and R. Karoumi, "From the desk to the field: Recent trends in deploying Wireless Sensor Networks for monitoring civil structures," *IEEE SENSORS Proceedings*, pp. 62-65, October 2011.

18. M. Massink, D. Latella, and J. Katoen, "Model checking dependability attributes of wireless group communication," *International Conference on Dependable Systems and Networks*, pp. 711-720, June 2004.

19. K. Saghar, W. Henderson, D. Kendall, and A. Bouridane, "Applying formal modelling to detect DoS attacks in wireless medium," *The 7th International Symposium on Communication Systems, Networks & Digital Signal Processing*, pp. 896-900, July 2010.

20. M. Chenine, J. Ullberg, L. Nordstrom, Y. Wu, and G. Ericsson, "A Framework for Wide-Area Monitoring and Control Systems Interoperability and Cybersecurity Analysis," in *IEEE Transactions on Power Delivery*, vol. 29, no. 2, 2014, pp. 633-641.

21. T. Cruz, L. Rosa, J. Proenca, L. Maglaras, M. Aubigny, L. Lev, J. Jiang, and P. Simoes, "A Cybersecurity Detection Framework for Supervisory Control and Data Acquisition Systems," in *IEEE Transactions on Industrial Informatics*, vol. 12, no. 6, 2016, pp. 2236-2246.

22. V. Kharchenko, D. Maevsky, E. Maevskaya, C. Phillips, and L. Vystorobska, "Employers' requirements-oriented assessment of IoT curriculum: The projects CABRIOLET and ALIOT," *9th International Conference on Dependable Systems, Services and Technologies (DESSERT 2018)*, pp. 677-681, May 2018.

## **30. INTELLIGENT METHODS AND APPROACHES FOR MANAGEMENT AND LEARNING OF IOT-BASED SYSTEMS**

Prof., DrS. Yu. P. Kondratenko, Ass. Prof., Dr. G. V. Kondratenko, Ass. Prof., Dr. Ie. V. Sidenko, Ph.D. Student M. O. Taranov (PMBSNU)

### ***Contents***

|                                                                               |     |
|-------------------------------------------------------------------------------|-----|
| Abbreviations .....                                                           | 471 |
| 30.1 Management systems and IoT platforms .....                               | 472 |
| 30.1.1 Types and capabilities of management systems and IoT platforms.....    | 472 |
| 30.1.2 Multi-criteria approach for choosing the IoT platform.....             | 475 |
| 30.1.3 Soft computing for the selection of specialized IoT platform           | 480 |
| 30.2 Multi-agent approach for development and management of IoT systems ..... | 482 |
| 30.2.1 Types and characteristics of agents .....                              | 483 |
| 30.2.2 Communication agents with the external environment .....               | 486 |
| 30.2.3 Data transfer techniques between agents in IoT systems .....           | 488 |
| 30.3 Methods and approaches for learning of IoT-based systems.....            | 490 |
| 30.3.1 General principles of M2M learning and self-learning systems .....     | 491 |
| 30.3.2 Technologies and applications of M2M learning .....                    | 493 |
| 30.3.3 Neural networks for learning of IoT-based systems .....                | 495 |
| 30.4 Work related analysis .....                                              | 497 |
| Conclusions and questions.....                                                | 498 |
| References .....                                                              | 500 |

### ***Abbreviations***

ACL – Agent Communication Language  
AI – Artificial Intelligence  
ANFIS – Adaptive Neuro-Fuzzy Inference System  
AWS – Amazon Web Services  
CUDA – Compute Unified Device Architecture  
DL – Deep Learning  
DM – Decision Maker  
FIPA – Foundation for Intelligent Physical Agents  
GPU – Graphical Processing Units  
GRASP – Greedy Randomized Adaptive Search Procedure  
GSA – Genetic Swarm Algorithm  
IoT – Internet of Things  
IT – Information Technology  
KIF – Knowledge Interchange Format  
KQML – Knowledge Query and Manipulation Language  
LT – Linguistic Term  
M2M – Machine-to-Machine  
MAC – Medium Access Control  
MAS – Multi-Agent System  
MCDM – Multi-Criteria Decision Making  
PaaS – Platform as a Service  
RB – Rules Base  
RL – Reinforcement Learning  
SA – Simulated Annealing

## **30.1 Management systems and IoT platforms**

### ***30.1.1 Types and capabilities of management systems and IoT platforms***

Internet of things (IoT) management system (software) helps manage strategies involving the connectivity of smart devices, as well as smart packaging, and their impact on business [1]. IoT refers to the wireless communication between devices and their ability to send, receive, and create data based on user activity and environmental factors. IoT management systems help businesses monitor and take action on the communication from and between registered devices, as well as control the devices from a remote interface on a desktop or mobile device when necessary. These systems log and store data from connected smart devices that provide real-time insights to help businesses uncover trends and become more efficient. IT teams within various organizations use IoT software to centralize activity and analytics related to a smart device network, receive alerts when performance is interrupted, and export relevant information to other IT infrastructure or analytics programs [1].

To qualify for inclusion in the IoT management category, a system must:

- sync with and monitor the activity of smart devices;
- provide tools for controlling, updating, and retrieving data from synced devices;
- allow actions to be taken based on data received from devices;
- provide dashboards and analytics for devices.

IoT management systems (software) are often understood as IoT platforms [2]. Let's consider in more detail the main types and capabilities of the IoT-platforms.

The IoT describes a network of interconnected smart devices, which are able to communicate with each other for a certain goal. But how easy is the process of realization which IoT platform is right for you? The market for IoT platforms is rapidly evolving. With an ever-increasing number of available platforms to choose from, the authors decided it would be helpful to lay out their features and capabilities for easy comparison using different methods of multi-criteria decision making [1, 3].

The development of the market of services and opportunities for information technologies leads to the emergence of the IoT concept. The IoT principle implies the interaction of familiar (for us in everyday life) things with the help of high-speed computer networks. In addition, IoT is a closer integration of physical devices and people among themselves to achieve specific goals. The desire of users to feel themselves in the role of IoT systems developers has pushed some companies to develop the special programmable platforms (IoT platforms). Such platforms allow coping with various tasks in the field of communications, information safety during data transmission, visualization of IoT systems performance, etc. [2, 3].

First of all, it is proposed to consider the possibilities (criteria) which have modern platforms.

*Device management* is one of the most important criteria expected from any IoT platform. The IoT platform should maintain a list of devices connected to it and track their operation status; it should be able to handle configuration, firmware (or any other software) updates and provide device-level error reporting and error handling. At the end of the day, users of the devices should be able to get individual device level statistics [1].

*Integration level* is another important criterion expected from an IoT platform [2]. The API should provide access to the important operations and data that needs to be exposed from the IoT platform.

*The level of safety and reliability* measures required to operate an IoT platform is much higher than general software applications and services. Millions of devices being connected to an IoT platform means that we need to anticipate a proportional number of vulnerabilities. Generally, the network connection between the IoT devices and the IoT platform would need to be encrypted with a strong encryption mechanism [1, 3, 4].

Another important criterion which needs attention is the *types of protocols for data collection* used for data communication between the components of an IoT platform. An IoT platform may need to be scaled to millions or even billions of devices (nodes). Lightweight communication protocols should be used to enable low energy use as well as low network bandwidth functionality [2, 4, 5].

*Variety of data analytics.* The data collected from the sensors connected to an IoT platform needs to be analyzed in an intelligent

manner in order to obtain meaningful insights. There are four main types of analytics which can be conducted on IoT data: real-time, batch, predictive, and interactive analytics [3, 5].

*Visualization* enables humans to see patterns and observe trends from visualization dashboards where data is vividly portrayed through line-, stacked-, or pie charts, 2D- or even 3D-models [1, 3, 5].

*Database functionality.* Scalable storage of device data brings the requirements for hybrid cloud-based databases to a new level in terms of data volume, variety, velocity, and veracity. Requirements for this criterion is an attempt to restore order in the processing and transfer of data from, for example, different platforms or even to other information systems [4, 5].

Consider some of the well-known platforms according to the main features and possibilities.

*Amazon Web Services (AWS) IoT Platform.* According to Amazon, their IoT platform will make it a lot easier for developers to connect sensors for multiple applications ranging from automobiles to turbines to smart home light bulbs [2]. The main features of AWS IoT platform are the registry for recognizing devices; software development kit for devices; device shadows; secure device gateway; rules engine for inbound message evaluation [4].

*Kaa IoT Platform.* Kaa is provided with open source code, which makes it easy to integrate into projects with "smart house" technology. This allows developers to create their own intelligent IoT systems much faster. In addition, it makes it possible to configure the corresponding IoT systems, or to combine them among themselves [3]. Main features of Kaa IoT platform perform real-time device monitoring; perform remote device provisioning and configuration; collect and analyze sensor data; analyze user behavior deliver targeted notifications; create cloud services for smart products [5].

*IBM Watson IoT Platform.* You can try out their sample apps to get a feel for how it all works. You can also store your data for a specified period, to get historical information from your connected devices. IBM Watson also offers some great security possibilities based on machine learning and data science [2]. Users of IBM Watson get device management; secure communications; real-time data exchange; data storage; recently added data sensor and weather data service [3].

*Microsoft Azure IoT Platform.* Representatives of Microsoft have cloud storage, machine learning, IoT services, and have even developed their own operating system for IoT devices [3]. Main features of Azure IoT platform are device shadowing; a rules engine; identity registry; information monitoring [4, 5].

*Bosch IoT Suite - MDM IoT Platform.* Bosch cloud offers its customers complete safety and reliability while storing the data on its secure server in Germany. The company hosts the cloud in Stuttgart, Germany. Using Platform as a Service (PaaS) the company can offer its service at fairly reasonable rates [2]. The main features of Bosch IoT platform are the PaaS; remote manager; analytics; cost-effective; ready-to-use [4].

### ***30.1.2 Multi-criteria approach for choosing the IoT platform***

Estimating and choosing a rational IoT platform is a rather complicated process for many reasons, including: multi-factor evaluation in the platform selection; complexity of preliminary consideration of all possible stages of decision making; lack of awareness of the peculiarities of modern information technology development and IoT services market; insufficient technical and material base and so on [6, 7].

In most cases, the choice of the IoT platform for the development of the IoT systems comes to the comparative analysis of their capabilities and taking the pricing policy for the services provided by the developers of their own IoT platforms into account. Besides, IoT developers often give preference to the well-known IoT platforms, without considering the criteria (factors) that in the future may affect the development, maintenance, updating, reliability, safety, and scaling of the developed IoT systems [4]. The study [2] noted that the following features and features of platforms should be taken into account when choosing an IoT platform: orientation toward the hybrid application environment; the ability to receive data and prepare it for the analysis; a statement of the owner of the cloud infrastructure; reliability and data safety; peripheral processing and data control.

One of the approaches to selecting an IoT platform is based on the defining a reference platform architecture that includes the benefits and capabilities of the existing modern IoT platforms [3]. Later on, a

comparative analysis of the selected platforms with the reference one is carried out and the best IoT platform is determined.

At the present time, there are several known methods of expert evaluation and selection of IoT platforms [11, 12], in particular, the analytic hierarchy process, the Delphi method and the decision making methods based on fuzzy sets and fuzzy logic [8-10]. The considered methods and approaches have some limitations and peculiarities of application. For example, the necessity of calculation of the expert judgment consistency; the limited number of levels of the hierarchy and the dimension of the paired-comparison matrix; the constant contact with experts for conducting the questionnaires; the need to update the structure of the model when changing the number of criteria and alternatives, etc. [4-10, 13].

Decision making process involves selecting one of the possible variants of decisions according to the criteria under deterministic and/or uncertain conditions. These possible variants of decisions are called alternatives. For the problem of selecting decisions, it is necessary to have at least two alternatives. When there are many alternatives, a decision maker (DM) cannot take enough time and attention to analyze each of them, so there is a need for means to support the choice of decisions. There is also a need of such facilities when the number of alternatives is small. In such problems, the number of alternatives, from the consideration of which the choice begins, is relatively small. However, they are not the only ones possible. Often, on their basis, new alternatives arise during the selection process. Primary basic alternatives do not always suit participants in the selection process. However, they help to understand what exactly is missing in the alternatives under consideration in this situation [6, 7]. This class of problems is called problems with constructed alternatives. In the modern theory of decision making, it is considered that the variants of decisions are characterized by different indicators of their attractiveness for DM. These indicators are called features, factors, attributes, or quality measures [7]. They all serve as the criteria for selecting a decision. In the vast majority of real problems, there are many criteria. The complexity of the decision making tasks is also affected by the number of criteria. With a small number of criteria (for example, for two), the task of comparing the two alternatives is fairly simple and transparent, the values of the criteria can be directly compared and a

preferred alternative can be developed. With a large number of criteria, the problem becomes immense for the DM. Fortunately, with a large number of criteria, they can usually be combined into groups with a specific semantic meaning. Such groups of criteria are, as a rule, independent. The identification of a structure on a set of criteria makes decision-making process meaningful and effective [6].

The traditional approach to operations research assumes the existence of a single criterion for assessing the quality of the decision [7]. However, the expansion of the field of application of operational research methods led to the fact that analysts began to face problems in which the existence of several criteria for assessing the quality of the solution is essential [6, 7].

The analysis of many real practical problems naturally led to the emergence of a class of multi-criteria problems. Most of the methods of multi-criteria decision making (MCDM) provide transformation of a multi-criteria problem into the one-criterion, which greatly simplifies the decision making process [6, 7, 10, 13].

The task of selecting the IoT platform is formulated as an MCDM problem and has the following form (decisions matrix):

$$Q(E_i) = \begin{pmatrix} Q_1(E_1) & Q_1(E_2) & \dots & Q_1(E_m) \\ Q_2(E_1) & Q_2(E_2) & \dots & Q_2(E_m) \\ \dots & \dots & \dots & \dots \\ Q_n(E_1) & Q_n(E_2) & \dots & Q_n(E_m) \end{pmatrix}; E_i \in E; (i = 1, 2, \dots, m; j = 1, 2, \dots, n), \quad (30.1)$$

where  $Q(E_i)$  is a vector criterion of quality for  $i$ -th alternative;  $Q_j(E_i)$  is the  $j$ -th component of the vector criterion of quality  $Q(E_i)$ .

The evaluation of the  $i$ -th alternative by the  $j$ -th criterion  $Q_j(E_i)$  have a certain scale of assessment and is presented by experts based on their experience, knowledge and experimental research in the field of specialized IoT platforms [10].

To solve the IoT platform selection problem, it is necessary to find the best alternative  $E^* \in E$  using data (30.1):

$$E^* = \text{Arg Max}_{i=1 \dots m} (Q(E_i)), E_i \in E, i = 1, 2, \dots, m. \quad (30.2)$$

The solution of the corresponding problems is found through the use of such methods as the selection of the main criterion, the linear, multiplicative and max-min convolutions, the ideal point method, the sequential concessions methodology, the lexicographic optimization [6, 7]. For some of them, it is necessary to determine the weight coefficients of the criteria, which sometimes is difficult with a large number of criteria [8-10].

Let's consider one of the existing multi-criteria decision-making methods, for example, ideal point method to solve the corresponding task of multi-criteria selection of the IoT platform [6].

The *ideal point method* implements the principle of an ideal decision. It postulates the existence of an "ideal point" for solving a problem in which the extremum of all criteria is achieved. Since the ideal point in most cases is not among the existing solutions, then there is a problem finding the "nearest" to the ideal permissible point. It would have been nice if there was a single objective notion of "distance", but it was not. If on a Cartesian two-dimensional subspace it is possible to apply the Euclidean metric, then, for example, the shortest path on the surface of a sphere is an arc, and not a straight line [6, 7, 10].

Thus, for solving the multi-criteria task using the ideal point method, it is necessary above all:

- determine the coordinates of the ideal point;
- select a metric which you can measure the distance to the ideal point.

To determine the coordinates of the ideal point you need to solve  $n$  one-criterion tasks for each of the optimization criteria:

$$Q_j(E_i) \Rightarrow \text{Max}; E_i \in E; (i = 1, 2, \dots, m; j = 1, 2, \dots, n). \quad (30.3)$$

Optimal values of the criteria for each of the one-criterion problems

$$Q_j^* \Rightarrow \text{Max}_{i \in \{1, 2, \dots, m\}} Q_j(E_i); E_i \in E; (i = 1, 2, \dots, m; j = 1, 2, \dots, n), \quad (30.4)$$

where  $Q_j^*$  is the optimal value of the  $j$ -th criterion;

will be the coordinates of the ideal point in the criteria space

$$Q^* = (Q_1^*, Q_2^*, \dots, Q_n^*), \tag{30.5}$$

where  $Q^*$  is the ideal point;  $Q_1^*, Q_2^*, \dots, Q_n^*$  are optimal values of  $n$  criteria (coordinates of the ideal point).

If the ideal point  $Q^*$  is permissible (but this happens very rarely), then the decision  $E^*$  is considered to be obtained. Otherwise, it is necessary to determine the distance  $d_\rho(E_i), (i = 1, 2, \dots, m)$  to the ideal point  $Q^*$ . To do this, it is necessary to choose a metric and finally to solve a one-criterion task of finding a point from the set of admissible decisions, which is closest to the ideal one [6].

Thus, the optimization problem takes the following form:

$$d_\rho(E_i) = \rho(Q(E_i) - Q^*) \Rightarrow \text{Min}; E_i \in E; (i = 1, 2, \dots, m), \tag{30.6}$$

where  $d_\rho(E_i)$  is a distance from ideal point  $Q^*$  to  $i$ -th alternative  $Q(E_i)$ ;  $\rho$  is a metric for measure the distance to the ideal point  $Q^*$ .

If the Euclidean metric [7] is chosen, then the criterion (30.6) takes the form:

$$d_\rho(E_i) = \sqrt{\sum_{j=1}^n (Q_j(E_i) - Q_j^*)^2} \Rightarrow \text{Min}; E_i \in E; (i = 1, 2, \dots, m; j = 1, 2, \dots, n), \tag{30.7}$$

where  $Q_j(E_i)$  are the coordinates of the  $i$ -th alternative in the criteria space;  $Q_j^*$  are the coordinates of the ideal point.

If the Hamming metric [7] is chosen, then the criterion (30.6) takes the form:

$$d_\rho(E_i) = \sum_{j=1}^n |Q_j(E_i) - Q_j^*| \Rightarrow \text{Min}; E_i \in E; (i = 1, 2, \dots, m). \tag{30.8}$$

The best alternative  $E^* \in E$  has the smallest distance  $d_\rho(E_i)$ .

### ***30.1.3 Soft computing for the selection of specialized IoT platform***

Appropriate MCDM methods have some limitations: the need to take into account the weight coefficients of the criteria; the provision of the Pareto-optimal set of alternative decisions; the lack of the ability to change the dimension of the vector of alternatives and criteria in real time; the significant impact of the weight coefficients that the expert determines, and the local criteria on the result [8-10].

Therefore, let us consider to use the soft computing approach, in particular, Mamdani-type fuzzy logic inference engine for selection of the specialized IoT platform [8, 9]. When selecting the specialized IoT platform, a large number of the criteria, that is sometimes not relevant and appropriate within the scope of a particular application, is used. According to the various studies and own experience, consider the use of the following important (main) criteria when selecting IoT platform [10]: level of safety and reliability ( $Q_1$ ); device management ( $Q_2$ ); integration level ( $Q_3$ ); level of processing and action management ( $Q_4$ ); database functionality ( $Q_5$ ); protocols for data collection ( $Q_6$ ); usefulness of visualization ( $Q_7$ ); variety of data analytics ( $Q_8$ ). In this case, criteria  $Q_1, Q_2, \dots, Q_8$  can act as input signals (coordinates) or factors  $X = \{x_1, x_2, \dots, x_8\}$  of the fuzzy logic inference system.

The structure  $y = f(x_1, x_2, \dots, x_8)$  of the fuzzy logic inference system for selection of the specialized IoT platform is presented in Fig. 30.1. It includes 3 fuzzy subsystems and has 8 input coordinates  $X = \{x_j\}, j = 1, \dots, 8$  and one output  $y$ , which are interconnected (by common properties) by the fuzzy dependencies  $y_1 = f_1(x_1, x_2, x_3, x_4)$ ,  $y_2 = f_2(x_5, x_6, x_7, x_8)$  and  $y = f_3(y_1, y_2)$  of the appropriate rules bases (RBs) of the 3 subsystems [8, 9].

To estimate the input coordinates  $X = \{x_j\}, j = 1, \dots, 8$ , three linguistic terms (LTs) with a triangular form of the membership function (MF), in particular "low - L", "medium - M" and "high - H", are chosen. Five LTs  $y_1, y_2 \in \{L, LM, M, HM, H\}$  are chosen for the evaluation of intermediate coordinates  $y_1$  - hardware level of the IoT

platform,  $y_2$  - software level of the IoT platform. The output coordinate  $y$ , which is intended for evaluation of the specialized IoT platform, has 7 LTs  $y \in \{VL, L, LM, M, HM, H, VH\}$ .

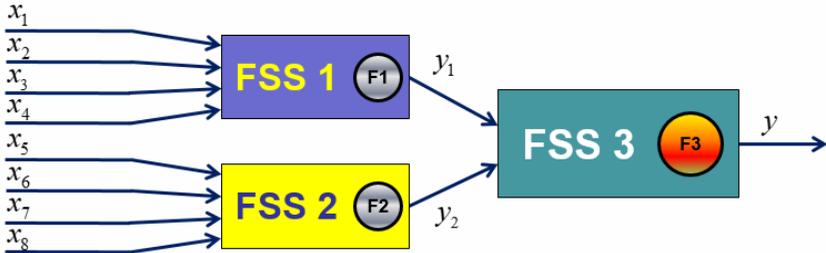


Fig. 30.1 – The structure of the fuzzy logic inference system for selection of the specialized IoT platform

The partial sets of rules of rules bases (RBs) for the first  $y_1 = f_1(x_1, x_2, x_3, x_4)$  and for the third  $y = f_3(y_1, y_2)$  fuzzy subsystems are given in Table 30.1.

Table 30.1 – The partial sets of rules of RBs for the first and for the third fuzzy subsystems

| № of rule | Subsystem $y_1 = f_1(x_1, x_2, x_3, x_4)$ |       |       |       |       | № of rule | Subsystem $y = f_3(y_1, y_2)$ |       |     |
|-----------|-------------------------------------------|-------|-------|-------|-------|-----------|-------------------------------|-------|-----|
|           | $x_1$                                     | $x_2$ | $x_3$ | $x_4$ | $y_1$ |           | $y_1$                         | $y_2$ | $y$ |
| 1         | L                                         | L     | L     | L     | L     | 1         | L                             | L     | VL  |
| 2         | L                                         | L     | L     | M     | L     | 2         | L                             | LM    | L   |
| ...       | ...                                       | ...   | ...   | ...   | ...   | 3         | L                             | M     | LM  |
| 37        | M                                         | M     | L     | L     | LM    | ...       | ...                           | ...   | ... |
| 38        | M                                         | M     | L     | M     | M     | 13        | M                             | M     | M   |
| ...       | ...                                       | ...   | ...   | ...   | ...   | 14        | M                             | HM    | HM  |
| 69        | H                                         | M     | M     | H     | HM    | 15        | M                             | H     | H   |
| 70        | H                                         | M     | H     | L     | M     | ...       | ...                           | ...   | ... |
| ...       | ...                                       | ...   | ...   | ...   | ...   | 24        | H                             | HM    | H   |
| 81        | H                                         | H     | H     | H     | H     | 25        | H                             | H     | VH  |

Using the Mamdani-type algorithm [14-16] to develop the fuzzy logic inference system for selection of IoT platform (Fig. 30.1), we eliminate the need to form weight coefficients for the criteria. In this case, this soft computing approach allows you to get rid of the limitations on the number of alternatives. Thus, alternatives can be evaluated in real time in unlimited quantities using the corresponding engine. In addition, a relevant fuzzy system can be trained with the help of, for example, the adaptive neuro-fuzzy inference system (ANFIS) neural network, which will give more accurate results of IoT platform evaluation [8,16].

### **30.2 Multi-agent approach for development and management of IoT systems**

In the modern world, the concept of IoT is impossible to imagine without the use of multi-agent technologies [17]. For each physical object, the program agent is brought into line with a certain degree of intellectualism, representing his interests in the network.

The application of distributed computing systems allows delegating complex tasks to software systems (agents), which, in turn, lets one represent and solve problems that are difficult to formalize. When the distributed access systems is designed, multiagent technology allows you to combine both protocol and any application software environment in a single system for processing and interacting with different types of data [18]. Such a system has the flexibility, scalability and efficiency of the distribution of load between servers.

According to the concept of multi-agent systems (MAS) and technologies, the agent has only part of the knowledge of the general problem, as a result, it is able to solve only part of the overall task. Therefore, to solve a complex problem, you need to have a plurality of agents that interact with each other, that is, a multi-agent system. Tasks in such systems are distributed among agents in accordance with their skills and capabilities. Any agent is an open system that has its own behavior. Thus, an agent is considered to be capable of perceiving information from a restricted environment, processing it on the basis of its own resources, interacting with other agents and acting in the environment for some time, pursuing its own goals [17].

### ***30.2.1 Types and characteristics of agents***

The foundations of multi-agent systems were formed as a result of the study of distributed computer systems, parallel computing and network technologies. The autonomy of individual system modules is the basis of multi-agency, and such modules are called agents. Each agent operates in a distributed system, where several processes, which may have been interconnected simultaneously, occur. An autonomous object or program that is capable of active motivated behavior and interaction with other objects in a dynamic environment is called an agent. Agents have the ability to receive messages by interpreting their content and generating new messages, which can be sent to other agents or to the core of the multi-agent message board system that will be available to all components of the system [17].

The multi-agent approach is used in various fields, among them there are distributed solutions of complex tasks, reengineering in the enterprise, interaction of robotic IoT systems [18].

There are two classes of tasks that are solved by multi-agent approach. The first class includes tasks of distributed control and planning. At the same time, the efforts of various agents are aimed at solving a common problem, in such tasks it is necessary to ensure effective interaction of agents. The second class includes tasks where each agent solves his problem independently, using shared resources [19-21].

The operation of the MAS is based on the principle of the division of responsibilities between individual subsystems, that is, in the common environment there are autonomous agents, whose work is aimed at satisfying the interests of different users. In this case, agents interact with each other while solving their tasks. These tasks include the management of information flows, network administration, and information search on the Internet, traffic management, collective decision of multi-criteria tasks, and many more [20].

The emergence of collaboration agents in distributed systems led to the formation of a modern representation of the agent. For a long time, the multi-agent paradigm has accumulated a significant theoretical base and experience [21]. Also, this research led to the emergence of different model agents, their types and characteristics, as

well as the tools and means necessary for their development. Different principles of agents' interaction were formed.

Increasing the complexity of tasks for IoT systems and the development of distributed computing has increased interest in the use of software agents. A software agent is an autonomous process that can respond to the environment and cause changes in conjunction with users or other agents [21]. It should be noted that the medium also affects the agent [1, 17]. Software agents are classified according to the following main features.

Based on mobility [20]:

- stationary agents;
- mobile agents.

The main difference between mobile and stationary agents in this classification is that mobile agents are able to move between nodes of the computing environment.

By type of interaction [1]:

- cooperative agents;
- competing agents.

A cooperative agent has the ability to integrate with other agents in the environment to solve a common task. In turn, competitors competing inherently competitive behavior for their own interests.

There are also many other features that can be used to classify agents [1]. First, it should be noted that agents can act as living beings. The signs that we shall consider further relate to the classifications of artificial agents (robots, automata or computer programs).

Agents can be generally divided into two large groups for functional purposes [17]:

- functional agents are those that exist and work in the real world and can be endowed with sensors to obtain information from the environment. An example of such agents may be robots;
- information agents exist only in the software environment, they mainly perform tasks related to computer calculations.

We distinguish the following types of agents by the ability to study [18]:

- agents capable of training, and the behavior of such agents is based on previously acquired experience;

- non-capable of learning agents, they act according to pre-written rules, which respond to changes in the environment.

By the ability of interaction [17]:

- stand-alone agents;
- know-how-to interact with other agents.

On the other hand, agents can be classified according to their ability to reason or “think”. It is the most effective approach in designing intelligent IoT systems. According to this classification, there are two types of agents: intellectual (cognitive) and reactive.

Intelligent agents have a well-developed mathematical model of the external world, which is constantly replenished. This is achieved through the presence of a knowledge base agent and mechanisms for analysis of actions. This type of agent is capable of conducting an analysis based on a model of the environment using a sample mapping method and, based on these data, of making decisions or performing certain work. When an agent has some resources, its knowledge base will contain information about the structure and status of resources, which will have a significant effect on subsequent behavior. Intelligent agent necessarily combines five main functions: cognitive; regulatory, ability to reason; communicative; resourceful [17-20].

In turn, reactive agents do not have data on the environment, data analysis mechanism, and resources. Therefore, these agents do not have a mechanism for predicting changes in the environment and their actions [17].

Also, the intelligent agent is characterized by higher autonomy than that of reaction agent, having its own goals, for the satisfaction of which they can use resources of other agents. In turn, reactive agents are highly dependent on the external environment and are capable of only corresponding reactions. Here is a comparison of these types of agents in Table 30.2 [17].

The typical tasks put of the agents include [21]:

- temporary calculations. The work of agents is carried out not only between fixed sub-networks of the network, but also between mobile platforms that are connected to the network. As an example, this can be the case: the mobile device is connected to the network and adds an agent which has to do some work, and then disconnects it. After

completing the agent's task, the device re-connects to another node of the network and downloads the results of its operation;

Table 30.2 – Characteristics of agents

| <b>Characteristics</b>                   | <b>Cognitive agents</b>                                                | <b>Reactive agents</b>                           |
|------------------------------------------|------------------------------------------------------------------------|--------------------------------------------------|
| The internal model of the external world | Developed                                                              | Primitive                                        |
| Speculation                              | Complex and reflexive                                                  | Simple one-step                                  |
| Motivation                               | Developed motivation system that includes beliefs, desires, intentions | The simplest incentives associated with survival |
| Memory                                   | Is                                                                     | None                                             |
| Reaction                                 | Slow                                                                   | Fast                                             |
| Adaptability                             | Low                                                                    | High                                             |
| Modular architecture                     | Is                                                                     | None                                             |
| Composition of the MAS                   | A small number of autonomous agents                                    | A large number of agents dependent on each other |

- search, processing and analysis of information. It is difficult for a person to work with large volumes of data, therefore the use of agents for the search and processing of information is effective enough;
- data monitoring. The agent in real time monitors the source of the data and notifies any changes.

**30.2.2 Communication agents with the external environment**

The main types of agents interaction with each other and with the environment include [17]:

- cooperation (it is the main form of interaction between agents and the environment, characterized by the unification of their actions, resources and means to achieve a common goal, with the division of functions between them);
- competition (confrontation, conflict);
- compromise (it is important to meet both your own requirements and the opponent's requirements);
- conformism (refusal of their claims in favor of the opponent);
- rejection of interaction.

The reactive agents interact with other agents in order to survive, their communication cannot be called intentional, it is based primarily on natural principles. Unlike intelligent agents, which co-operate "consciously" to meet the needs. After all, as an agent or system is able to be under the influence of the environment, it reflects its performance. Co-operation between agents and the environment can arise both on the principles of co-operation or forced, and on the basis of situational cooperation or voluntarily. Agreements and co-operation between agents is needed. One can distinguish the following main reasons for the cooperation of agents [19].

*A common goal.* As a rule, if the agents are bound by this cause, then they will interact with the type of cooperation. However, it is necessary to check that such cooperation does not lead to the destruction of the agent or its viability. There is another possible situation when agents do not coincide. Then there are conflicts between MAS objects. But in this situation conflicts can also have a positive impact on the system. They promote development and provide incentives for agents. There are systems with simultaneous interaction types of cooperation and confrontation. An example is the predator-victim model [20].

To achieve its goal, the agent needs some resources, that is, resources. If agents do not have shared resources - conflicts arise. To solve this problem, the rule is said to "win stronger". That is, a stronger agent will pick up resources at the weaker one. This can be called the most effective and easiest way to resolve such conflicts. But in some situations it is advisable to negotiate [21]. In this case, the agents are compromising, taking into account the interests of everyone.

Each agent utilizes a limited set of knowledge that he or she may need to achieve local and global issues. Therefore, he has to look for interactions with other agents [17].

Thus, the following circumstances are distinguished:

- 1) the agent is able to achieve the goal without the help of others, ie independently;
- 2) the agent is able to achieve the goal on its own, but through interaction the problem can be solved more effectively or faster;
- 3) the agent can achieve the goal only by using third-party help.

Agents can independently choose the type of interaction with each agent or environment, depending on the relevance of the connection.

In order to establish the order between agents in the process of interaction, there are obligations. With the help of commitments, you can predict the actions of other agents and plan their own. Below are the following types of obligations [1]:

- 1) the agent is obliged to other agents;
- 2) the agent is obliged to the group;
- 3) the group is obliged to the agent;
- 4) the agent is obliged to himself.

Formal representation of goals, commitments, desires and intentions, as well as all other data, forms the basis of the mental model of the intellectual agent that provides its motivated behavior in offline mode.

There are various forms of agent cooperation [17]:

- ordinary cooperation, which is achieved through the exchange of experience of each agent (sharing tasks, sharing of knowledge, etc.) without special measures to coordinate their actions;
- co-ordinated collaboration, if agents have to coordinate their steps (sometimes using the so-called coordinating agent) for the efficient use of resources and their experience;
- non-productive cooperation, if agents together use resources or solve common problems without sharing experience and interfering with one another.

### ***30.2.3 Data transfer techniques between agents in IoT systems***

In order for agents to transmit information to each other in distributed systems, they use agent interaction. To do this, the MAS uses [17]:

- universal programming languages, such as (Java);
- knowledge-oriented languages, i.e. knowledge representation languages (Knowledge Interchange Format (KIF)); language of agent interaction (Foundation for Intelligent Physical Agents (FIPA), Knowledge Query and Manipulation Language (KQML), AgentSpeak, April);
- language of agent specifications;
- specialized programming languages for agents (TeleScript);
- script description languages (Tc / Tk);
- languages of logical programming (Oz);

- lisp-like languages that are close to ordinary language.

Two different approaches can be used to develop a data exchange language between agents. The first approach is procedural, which means that communication is based on the implementation of instructions. Such a language can be designed and programmed on Java or on a development tool such as Tcl. The second approach is declarative, that is, communication is based on descriptions. This approach has become more widely used to create sharing languages between agents [1]. The most popular standards defining the language of agent communication are FIPA [11] and KQML [19].

*FIPA standard.* Developed by the FIPA Committee [17]. It includes the FIPA ACL (Agent Communication Language) language [7], through which agents can transmit messages of a certain format using various data services, and a LISP-like language describing the content of the FIPA SL (Semantic Language) message. The internal architecture of the FIPA standard consists of the following services, which are integrated into the general registry:

- message service;
- service of registration of agents in the environment (that is, in the MAS);
- service description of the language of communication agents;
- a register of all services.

This architecture can interact with external systems for managing and implementing current agent tasks.

*KQML standard.* Designed by the ARPA Committee (Advanced Research Projects Agency) [17]. It includes the KQML language that defines a set of performativity actions and a LISP-like language for describing the content of the message, KIF. The standard consists of three levels: the communicative level (describes parameters such as sender, receiver, and different message identifiers), message level (describes requests, control actions, and protocol for interpreting the message), and content level (contains information that accompanies message level requests).

The standard is characterized by the following main features:

- agents are connected by one-way communication channels, by which fixed communications are transmitted;

- communication channels may have a non-zero delay in the transmission of the message;
- when receiving a message, the agent determines who and what input message this message has arrived on;
- the agent can send the message only through a specific channel;
- messages for a particular addressee are received in the dispatch order;
- delivery of messages is absolutely reliable дійна.

The standard supports both synchronous and asynchronous transmission of messages. Agents can communicate directly with other agents (with a symbolic name), send out broadcast messages, or "ask" other agents-participants in the conversation. The following are examples of systems implemented with KQML [1, 17]:

- Next-Link software program [1] developed at Stanford University aims at exploring the principles of coordination, allowing established agents to carry out distributed design and design of systems;
- Logic Centered Design [18], developed at Lockheed AI Center's Center of Artificial Intelligence, which is positioned as an intelligent information system for designing systems;
- Concur [19] web server presentations developed at Stanford University by Gregory R. Olsen, which uses an agent approach to computing in distributed design systems;
- a software complex for distributed data acquisition on global temperatures and dampness [20], developed by Diane Weiss of MITRE.

### **30.3 Methods and approaches for learning of IoT-based systems**

Microsoft together with the technology developer for IoT Fathym helps to cope with dangerous glaciation of roads in Alaska caused by unusual edge fluctuations in temperature [1]. Fluctuations in temperature lead to glaciation of roads, but past experience no longer helps to cope with the situation. IoT sensors and machine learning came to the rescue. Fathym equipped snow-removal cars and light truck fleets of the county with a system of mobile sensors that track the temperature of the road, the amount of precipitation, and the condition of the road surface [22]. During the normal working day, sensors send data at 3-

second intervals to the cloud-based analytical platform WeatherCloud, which runs on the basis of cloud-based Microsoft services. The platform connects the findings with the forecasts of local meteorological stations and gives the result. For example, if WeatherCloud shows that in the north of the city there is more ice than in the south, it distributes chemical reagents, respectively. If the result warns of a decrease in temperature after 3 days, it does not send a command with reagents from which there will be even more ice. Such information allows not only to save money from the local transport department but also save lives. Annually in the USA 150 000 accidents occur on the roads through ice, injuries are received by 39 000 people, 550 people die. The Alaska Transportation Department is the first customer, but the company plans to transfer its experience to other states and abroad. The cloud's IoT platform, developed by RoadBotics, works with smartphone cameras and continuously monitors travel conditions when drivers drive on US roads. And the technology of deep learning helps to identify defects in the road surface [23].

### ***30.3.1 General principles of M2M learning and self-learning systems***

Machine-to-Machine (M2M) learning is a set of technologies that allow machines to exchange information with one another, or to transmit it unilaterally [22]. These can be wired and wireless sensors monitoring systems or any device parameters (temperature, stock levels, location, etc.). For example, ATMs or payment terminals can automatically transmit information over GSM networks, or if they have check paper or cash finished, or conversely because there is too much cash and the arrival of collectors is required. M2M is also actively used in security and safety systems, health systems, industrial telemetry systems, and positioning systems for moving objects based on GLONASS / GPS systems [1]. One of the subclasses of M2M is the use of mobile solutions, and it can also use the abbreviation M2M (Mobile-to-Mobile).

Self-learning systems these are intellectual information systems, which, based on examples of real practice, automatically generate the proper knowledge [23].

At the heart of self-learning systems, there are the methods of automatic classification of examples of real practice, which means

training on the samples. Examples of real situations accumulate over a period and constitute a training sample. As a result of learning the system automatically generates generalized rules or functions that determine the attachment of situations to the classes that the trained system uses in interpreting unfamiliar situations. From the general rules, the knowledge base is automatically formed, which is periodically adjusted as the information on the situations analyzed grows [1].

Distinguish the following types of self-learning systems.

*Inductive systems* [22]. A system with inductive output is a self-learning intelligent system, which works on the principle of induction by classifying examples by significant features. Inductive conclusion (from partial to general) generalizes the statement on the basis of the plural of partial statements. The generalization of examples on the basis of this principle is reduced to the choice of the classification mark from the set of given; detecting a plurality of examples by the value of the selected attribute; determining the belonging of these examples to one of the classes. The classification procedure can be represented as a decision tree, in which the intermediate nodes are the values of the signs of the sequential classification and in the end nodes the value of the attribute of belonging to a certain class.

*Neural Networks* are self-learning intelligent systems, which are built on the basis of learning real examples an associative network of concepts (neurons) for parallel solutions to it [23]. As a result of training, mathematical solving functions (transfer functions or activation functions) that form the dependencies between input and output characteristics (signals) are formed on the examples.

*Case-based reasoning systems* [22] are self-learning intelligent systems that, as units of knowledge, preserve the precedents of solutions (examples) and allow, upon request, to select and adapt the most similar precedents. In these systems, the knowledge base contains descriptions of non-generalized situations, and actually the situations themselves or precedents. Then the search for a solution to a problem is reduced to a search by analogy (abductive conclusion).

*Data warehouses* are self-learning intelligent systems that allow you to learn from databases and create specially-organized knowledge bases. Information repositories are a repository of meaningful information, are regularly exported from operational databases and are

intended for operational analysis of data (implementation of OLAP-technology). To extract meaningful information from databases, special methods (Data Mining or Knowledge Discovery) are based on the use of methods of mathematical statistics, inductive methods of constructing decision trees or neural networks [23-25].

### ***30.3.2 Technologies and applications of M2M learning***

The most successful algorithms of machine learning are those that automate the processes of decision-making by generalizing known examples. In these methods, known as teacher training or supervised learning, the user provides an object-response pair of algorithms, and the algorithm finds a way to get an answer to an object. In particular, the algorithm is able to find an answer to an object which it had never seen before, without any human help. Returning to the example of spam classification using machine learning, the user submits an algorithm to a large number of letters (objects) along with information about whether a message is a spam or not (answers). For a new email, the algorithm will determine the probability that this message can be attributed to spam [24].

The algorithms of machine learning, which are studied in the object-response pairs, are called learning algorithms with the “teacher”, as the "teacher" shows the algorithm of response in each observation, on which the learning takes place. Despite the fact that creating a set of objects and responses - this is often a labor-intensive process, which is carried out manually, learning algorithms with the teacher interpreted and the quality of their work easy to measure. If the task can be formulated as a task work with a teacher, and you can create a dataset, which includes answers, then it is likely that machine learning will solve this problem [25].

Let's consider examples of problems of machine learning with a teacher [22].

*Determination of postal code by handwritten digits on an envelope.* Here the object will be a scanned image of the handwriting, and the answer is the actual digits of the postal code. To create a dataset for building a model of machine learning, you need to collect a large number of envelopes [23]. Then you can independently read the postal codes and save the numbers in the form of responses.

*Definition of tumor benignity on the basis of medical images.* Here the object will be the image, and the answer is the diagnosis of whether the tumor is benign or not. To create a dataset for model building, you need a database of medical images. In addition, you need an expert opinion, so the doctor should look at all the images and decide which tumors are low-quality, and which are not. In addition to image analysis, you may need additional diagnostics to determine the high quality of the tumor [23].

*Detecting fraudulent activity in credit card transactions.* Here the object is a transaction with a credit card, and the answer is information about whether the transaction is fraudulent or not. For example, you are the institution issuing credit cards, dumpers have the purpose of saving all transactions and records of customer messages about fraudulent transactions [23].

With these examples, it's interesting to note that although objects and answers look quite simple, the process of data collection for these three tasks is significantly different. Despite the fact that reading envelopes are labor-intensive occupation, this process is simple and cheap. Getting medical images and performing diagnostics requires not only expensive equipment but also rare, highly paid expert knowledge, not to mention the ethical issues and issues confidentiality. In the example of detecting credit card fraud, data collection is much easier. Your customers will give you an answer by reporting fraud. All you have to do to get objects and responses related to fraudulent activity is to wait [22].

*Learning algorithms without a teacher or uncontrolled learning.* In the algorithms of learning without a “teacher”, only objects are known, and there are no answers. Although there are many successful areas for the application of these methods, they are usually more difficult to interpret and evaluate [24].

Let's look at examples of problems of machine learning without a teacher.

*Definition of topics in the set of posts.* If you have a large collection of text data, you can aggregate them and find the most common themes. You do not have any previous information about what topics are being considered and how much they are. Therefore, there are no known answers.

*Dividing clients into groups with similar preferences.* With a set of customer records, you can identify groups of clients with similar benefits. For a trading site, such groups may be "parents", "bookmates" or "gamers". Because you do not know in advance about the existence of these groups and their quantities, you have no answers [22].

*Detecting patterns of abnormal behavior on the website.* It is often useful to identify mistakes, patterns of behavior that are different from the norm. Patterns of abnormal behavior may be different, and, perhaps, you will get registered cases of abnormal behavior. Because in this example you only see traffic, and you do not know what is normal and abnormal behavior. It is a problem of learning without a teacher [23].

When solving the problem of machine learning with and without a teacher, it is important to present the input data in a format that is understandable for the computer. Often the data is presented as a table. Each data point you want to explore (each email, each client, each transaction) is a string, and each property that describes this data point (e.g., customer's age, amount, or transaction place) is a column. You can describe users by age, article, account creation date and shopping frequency in an online store. You can describe the image of a tumor using grayscale for each pixel or with the size, shape, and color of the tumor. In machine learning, each object or line is called a sample or a data point, and column properties. These examples are called characteristics or features [1, 22].

But no algorithm for machine learning will be able to predict data that does not contain any useful information. For example, if the only sign of a patient is his or her last name, the algorithm will not be able to predict his gender. This information is simply not available in the data. If we add another sign-the patient's name, the effectiveness of accurate prediction is higher, since often, knowing the person's name, one can judge his/her gender [22, 23].

### ***30.3.3 Neural networks for learning of IoT-based systems***

In the past few years, the Artificial Intelligence field has entered a high growth phase, driven largely by advancements in Machine Learning methodologies like Deep Learning (DL) and Reinforcement Learning (RL). Combinations of those techniques demonstrate unprecedented performance in solving a wide range of problems,

from playing Go at super-human level to diagnosing cancer like a specialist [22].

DL/RL innovations are happening at an astonishing pace (thousands of papers with new algorithms are presented in numerous AI related conferences every year). Though it is premature to predict the final winning solutions, hardware companies are racing to build processors, tools, and frameworks. They are trying to identify pain points and bottlenecks in DL workflows (Fig. 30.2), leveraging years of experience of researchers [23].

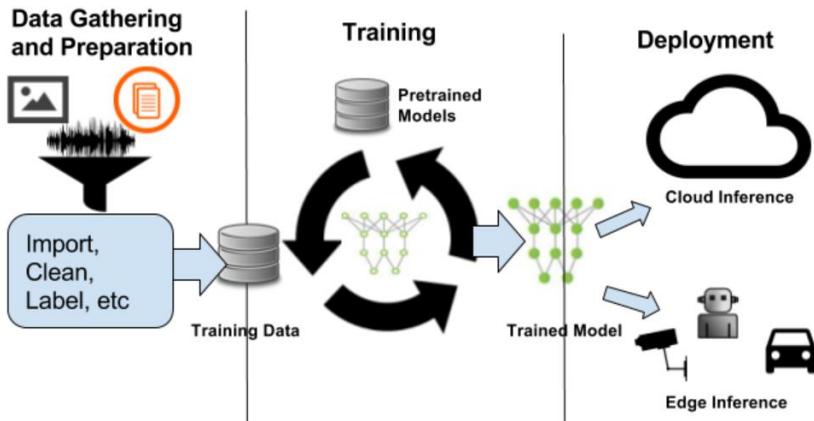


Fig. 30.2 – Basic Deep Learning Workflow [23]

Let’s consider some training platforms. Graphical Processing Units (GPU) based systems are usually the choice for training advanced DL models. Nvidia has long realized the advantages of using GPU for general purpose high performance computing [8, 9].

GPU has hundreds of compute cores that support a large number of hardware threads and high throughput floating point computations. Nvidia developed Compute Unified Device Architecture (CUDA) programming framework to make GPU friendly for scientists and machine learning experts to use [24].

CUDA toolchain has improved overtime, providing researchers a flexible and friendly way to realize highly complex algorithms. A few years ago, Nvidia aptly identified the DL opportunity and persistently developed CUDA support for most of DL operations. Standard

frameworks like Caffe, Torch, and Tensorflow all support CUDA [25]. In cloud services like AWS, developers have a choice between using CPU or GPU (more specifically Nvidia GPU). Platform choice depends on the complexity of the neural networks, budget, and time.

### **30.4 Work related analysis**

A lot of developments and approaches for management and learning of IoT-based systems belongs to Leeds Beckett University, Newcastle University, KTH Royal Institute of Technology and University of Coimbra. The paper [11] considers the architecture of a typical IoT Data Analytics Platform (IoTDAP), which starts with raw data collection from sensing devices and ends with complex data analytics and decision making activities. This research [12] aims at creating a resource-sharing platform to support such relationships, in the perspective that resource unconstrained devices can assist constrained ones, while the latter can extend the features of the former.

A hybrid multi-objective approach based on GRASP (Greedy Randomized Adaptive Search Procedure) and SA (Simulated Annealing) meta-heuristics is proposed [13] to provide decision support in a direct load control problem in electricity distribution networks. The incorporation of preferences is made operational using the principles of the ELECTRE TRI method, which is based on the exploitation of an outranking relation in the framework of the sorting problem. Diversity of hard logic and soft processors, interfaces and buses, self-diagnostics means are described in paper [14]. Addressed to the problem of translating the control knowledge of a human expert operator into fuzzy control rules, this paper [15] proposes an approach to automatically design a Mamdani fuzzy logic controller. The proposed approach is based on the use of a data set extracted from a process that has been manually controlled, and has the aim of learning a Mamdani logic controller with the capability to imitate the control action of an expert human operator. The results show that the proposed approach has the capability of designing the Mamdani fuzzy controller in order to successfully controlling the real experiment [15]. Knowledge gained through classification of microarray gene expression data is increasingly important as they are useful for phenotype classification of diseases. Different from black box methods, fuzzy expert system can produce interpretable classifier with knowledge expressed in terms of

if-then rules and membership function. This paper [16] proposes a novel Genetic Swarm Algorithm (GSA) for obtaining near optimal rule set and membership function tuning.

This paper [18] focus on the development of a hierarchical multi-agent framework for resilience enhancement over wireless sensor and actuator networks. Experimental results collected from a laboratory IPv6 based test-bed comprising distributed computational devices and heterogeneous communications, show unequivocally the relevance and inherent benefits of the proposed approach. In this paper [19] authors identify some of the existing MAS architectures for WSNs, and propose some novel architectures. Multi-agent platform and toolbox for fault tolerant networked control systems are considered in the paper [20]. This work aims to expand on previous investigations considering frequency control and examines distributed communication and control architectures through the medium of MAS focusing on voltage control in a radial microgrid. The investigation assesses control and communication performance across a range of agent architectures against four selected performance criteria, and an increasing agent population [21].

In M2M networks, an energy efficient scalable medium access control (MAC) is crucial for serving massive battery-driven machine-type devices. In this paper [24], authors investigate the energy efficient MAC design to minimize battery power consumption in cellular-based M2M communications [25]. Authors present an energy efficient MAC protocol that not only adapts contention and reservation-based protocols for M2M communications in cellular networks, but also benefits from partial clustering to handle the massive access problem.

### ***Conclusions and questions***

In this section, the materials for module PCM4.3 “Intelligent methods and approaches for management and learning of IoT-based systems” of PhD course “Development and implementation of IoT-based systems” are presented. They can be used for preparation to lectures and self-learnig for lecturers, PhD-students, IoT developers, etc. These module materials were developed by Prof. Yu. P. Kondratenko, Assoc. Prof. G. V. Kondratenko, Assoc. Prof. Ie. V. Sidenko, Ph.D. Student M. O. Taranov.

Recently, the direction associated with the analysis of the intelligent methods and approaches for management and learning of IoT-based systems has become very popular and effective. This gave rise to such areas as neural network technologies, cloud and fog computing, control systems, computer vision, etc [2, 3, 8-10].

This chapter discusses the types and capabilities of IoT platforms, multi-criteria approach and soft computing for choosing the IoT platform [6-10]. Also analyzed the concept of multi-agent approach in IoT, in particular, types and characteristics of agents, communication agents with the external environment and data transfer techniques between agents. In addition, an important component of the IoT network is the choice of methods and approaches for learning of IoT-based systems. Also considered general principles of M2M learning, self-learning systems and neural networks [22, 23].

The considered methods and approaches are widely used in all applications of the IoT, for example, medical and healthcare, transportation systems, building and home automation, manufacturing, agriculture, energy management, environmental monitoring, etc [1, 4].

In order to better understand and assimilate the educational material that is presented in this section, we invite you to answer the following questions.

1. What is the IoT platform?
2. What criterion is responsible for working with 2D- and 3D-models and graphs?
3. What platform is developed by Amazon?
4. What is the minimum number of criteria necessary when solving a problem using multi-criteria decision making?
5. What does this component  $Q_j(E_i)$  mean?
6. What is needed to solve the problem by MCDM methods?
7. What is the main principle of the ideal point method?
8. What is the Euclidean metric?
9. What is the Hamming metric?
10. What is the form of membership function in a linguistic term?
11. What is the ANFIS?
12. What is the main difference between mobile and stationary agents?
13. What is the functional agent?

14. What is the reaction of cognitive agents?
15. What are the most popular standards defining the language of agent communication?
16. What is the neural network?
17. What is the synonymous name of the learning algorithm without a teacher?

### ***References***

1. F. Hussain, *Internet of Things: Building Blocks and Business Models*. Cham: Springer, 2017.
2. G. Keramidas, N. Voros, and M. Hubner, *Components and Services for IoT Platforms*. Cham: Springer, 2017.
3. J. Guth, U. Breitenbucher, M. Falkenthal, F. Leymann, and L. Reinfurt, "Comparison of IoT platform architectures: A field study based on a reference architecture," *Cloudification of the Internet of Things (CIoT)*, P.72-77, November 2016.
4. Y. Kondratenko, G. Kondratenko and I. Sidenko, "Multi-criteria Decision Making and Soft Computing for the Selection of Specialized IoT Platform," in *Recent Developments in Data Science and Intelligent Analysis of Information. ICDSIAI 2018. Advances in Intelligent Systems and Computing*, vol. 836, O. Chertov, T. Mylovanov, Y. Kondratenko, J. Kacprzyk, V. Kreinovich, and V. Stefanuk, Eds., 2019, P. 71-80. DOI: 10.1007/978-3-319-97885-7\_8.
5. Y. Kondratenko, G. Kondratenko, and I. Sidenko, "Multi-criteria decision making for selecting a rational IoT platform," *IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, P. 147-152, May 2018.
6. A. V. Katrenko, V. V. Pasichnyk, and V. P. Pas'ko, *Decision making theory*. Kyiv: Publ. Group BHV, 2009 (in Ukrainian).
7. Y. P. Zaychenko, *Decision making theory*. Kyiv: NTUU "KPI", 2014 (in Ukrainian).
8. A. P. Rotshtein, *Intelligent Technologies of Identification: Fuzzy Logic, Genetic Algorithms, Neural Networks*. Vinnitsya: Universum Press, 1999 (in Russian).
9. A. Piegat, *Fuzzy Modeling and Control*. Heidelberg: Springer, 2001.
10. G. Kondratenko, Y. Kondratenko, and I. Sidenko, "Fuzzy Decision Making System for Model-Oriented Academia/Industry Cooperation: University Preferences," in *Complex Systems: Solutions and Challenges in Economics, Management and Engineering. Studies in Systems, Decision and*

*Control*, vol. 125, C. Berger-Vachon, A. Gil Lafuente, J. Kacprzyk, Y. Kondratenko, J. Merigó, C. Morabito, Eds., 2018, P. 109-124.

11. G. Kecskemeti, G. Casale, D. Jha, J. Lyon, and R. Ranjan, "Modelling and Simulation Challenges in Internet of Things," in *IEEE Cloud Computing*, vol. 4, no. 1, 2017, P. 62-69.

12. R. Silva, J. Sa Silva, and F. Boavida, "A symbiotic resources sharing IoT platform in the smart cities context," *IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, P. 192-197, April 2015.

13. E. Oliveira, C. Henggeler Antunes, and A. Gomes, "A hybrid multi-objective GRASP+SA algorithm with incorporation of preferences," *IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making (MCDM)*, P. 32-39, December 2014.

14. O. Illiashenko, V. Kharchenko, A. Kor, A. Panarin, and V. Sklyar, "Hardware diversity and modified NUREG/CR-7007 based assessment of NPP I&C safety," *9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, P. 907-911, September 2017.

15. J. Mendes, A. Craveiro, and R. Araujo, "Iterative Design of a Mamdani Fuzzy Controller," *13th APCA International Conference on Control and Soft Computing (CONTROLO)*, P. 85-90, June 2018.

16. P. Ganesh Kumar, T. Aruldoss Albert Victoire, P. Renukadevi, and D. Devaraj, "Design of fuzzy expert system for microarray data classification using a novel Genetic Swarm Algorithm," in *Expert Systems with Applications*, vol. 39, no. 2, 2012, P. 1811-1821.

17. R. Fagin, J. Halpern, Y. Moses, and M. Vardi, *Knowledge in Multi-Agent Systems*. Cambridge: MIT Press, 2003.

18. F. Janeiro, A. Cardoso, and P. Gil, "Multi-agent approach for resilience enhancement in wireless sensor and actuator networks," *Joint 17th World Congress of International Fuzzy Systems Association and 9th International Conference on Soft Computing and Intelligent Systems (IFSA-SCIS)*, P. 635-640, June 2017.

19. R. Tynan, G. O'Hare, D. Marsh, and D. O'Kane, "Multi-agent System Architectures for Wireless Sensor Networks," in *Lecture Notes in Computer Science*, 2005, P. 687-694.

20. M. J. G. C. Mendes, B. M. S. Santos and J. S. da Costa, "Multi-agent Platform and Toolbox for Fault Tolerant Networked Control Systems," in *Journal of Computers*, vol. 4, no. 4, 2009, P. 303-310.

21. C. Cameron, C. Patsios, and P. Taylor, "On the benefits of using self-organising Multi-Agent architectures in network management," *International*

*Symposium on Smart Electric Distribution Systems and Technologies (EDST)*, P. 335-340, September 2015.

22. R. Schneiderman, "Internet of Things/M2M-A (Standards) Work in Progress," in *Modern Standardization*, 2015, P. 203-234.

23. A. Laya, L. Alonso, J. Alonso-Zarate, and M. Dohler, "Green MTC, M2M, Internet of Things," in *Green Communications*, 2015, P. 217-236.

24. A. Azari and G. Miao, "Energy efficient MAC for cellular-based M2M communications," *IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, P. 128-132, December 2014.

25. G. Lawton, "Machine-to-machine technology gears up for growth," in *Computer*, vol. 37, no. 9, 2004, P. 12-15.

## **31. PROTOTYPING AND RAPID DEVELOPMENT OF IOT SYSTEMS**

Assoc. Prof., Dr A. P. Plakhteyev, MSc student  
H. A. Zemlianko (KhAI)

### *Contents*

|                                                                                                                |     |
|----------------------------------------------------------------------------------------------------------------|-----|
| Abbreviations .....                                                                                            | 504 |
| 31.1 IoT devices .....                                                                                         | 505 |
| 31.1.1 Interaction of end devices with IoT .....                                                               | 505 |
| 31.1.2 EDGE Computing for Association Sensor Networks.....                                                     | 507 |
| 31.1.3 Types and structure of IoT devices.....                                                                 | 509 |
| 31.2 Prototyping and rapid development principles .....                                                        | 511 |
| 31.2.1 Main concepts.....                                                                                      | 511 |
| 31.2.2 Techniques for prototyping of IoT systems.....                                                          | 513 |
| 31.2.3 Rapid Prototyping of Fragments of Internet of Things .....                                              | 515 |
| 31.2.4 Modeling of Access to WEB Resources.....                                                                | 517 |
| 31.3 Cases of IoT systems rapid development .....                                                              | 519 |
| 31.3.1 Case 1. Collection of data on an angular position of the mobile<br>platform of road laboratory.....     | 519 |
| 31.3.2 Case 2. Monitoring of temperature with use of a cloud service of<br>Thingspeak and access on WiFi ..... | 527 |
| 31.4 Work related analysis .....                                                                               | 530 |
| Conclusions and questions.....                                                                                 | 531 |

***Abbreviations***

BLE – Bluetooth Low Energy

BN – Boundary node

CPU – central processing unit

HRDMI – High Resolution Distance Measurement Instrument

IDK – IoT Development Kit

LED – Light-emitting diode

MCU – Microcontroller Unit

ROMDAS – ROad Measurement Data Acquisition System

SN – Sensor network

SoC – System on Chip

Sigfox – Global Communications Service Provider for the IoT

USART – Asynchronous serial interfaces –

The Internet of things develops very quickly. A variety application demands inclusion in development of specialists of different qualification. For entry into the market time of development is reduced with since normal 9-12 months and tends to reduction up to 3 - 6 months. Compliance to requirements for functionality, flexibility, opportunities of development, energy efficiency, and cost of products and to operating costs depends on quality of design. Eventually success in the market of projectable IoT of systems is defined. Unlike many other IoT applications the small-size devices capable to provide interaction with other devices by means of peer-to-peer or network connections are required.

### **31.1 IoT devices**

#### ***31.1.1 Interaction of end devices with IoT***

In the general scheme (Fig. 31.1) of the interconnection of nodes of sensor networks (SN), individual devices (Dev), sensors (S), actuators (A) are showed. At the Edge, Dew, Fog, Cloud (Cloudlet) levels, the state of S and A is represented by digital copies (Digital Twin).

Sensor networks can be wired and wireless. The most popular networks are Ethernet, RS485 and similar, CAN, LIN, etc. Wireless networks can be proprietary in the ISM bands 315, 433, 868 MHz, as well as 2.4 GHz networks based on the IEEE802.15.4 standard (ZigBee, 6LoWPAN, etc.) ), IEEE802.15.1 (Bluetooth v.2 .. 5), IEEE802.11 (WiFi). Low-speed sensor networks in industry and home automation are combined using high-speed interfaces - Ethernet, WiFi. In fact, a heterogeneous network is formed, where it is necessary to provide access to various nodes in each network and their interconnectivity. Boundary nodes are bridges (gateways) between segments of sensor networks [1].

Thus, the IoT fragment can be represented as a hierarchical structure from a variety of disparate sensor networks  $SN = \{SN1, SN2, SNi, \dots\}$ . Each network consists of a set of nodes:  $SNi = \{Ni, 1, \dots, Ni, j, \dots\}$  connected by the network communication interface  $Ci$  from the set  $C = \{C1, C2, \dots\}$ , by the protocol  $Pi$  from set  $P = \{P1, P2, \dots\}$ . The  $Ni, j$  node serves a set of sensors (S) and actuators (A). To implement

the functions of the node a microcontroller platform  $MCU_{i,j}$  is used that has a network interface  $C_i$ , as well as analog and digital interfaces, that form data streams from sensors  $S$  and for controlling actuators  $A$ . Networks are designed to collect data from sensors primary processing, accumulation, presentation in some form (indication, sound, video, etc.), control and management of executive devices. The network can be embedded in some object (robot, tool, machine, etc.), and the state of set  $S$  and  $A$  determines the state of the object.

Let the current state of the node of one network be determined by the state of the sensor  $S$ , and the node of the other network by the state of the actuator  $A$ . In so doing, the state  $S$  is displayed on the state  $A$ . The simplest example is that the state of the switch determines by the state of the lamp, which can be realized by their direct connection. Alternatively, the state  $S$  is determined by the MCU that is associated with the MCU that controls  $A$ .

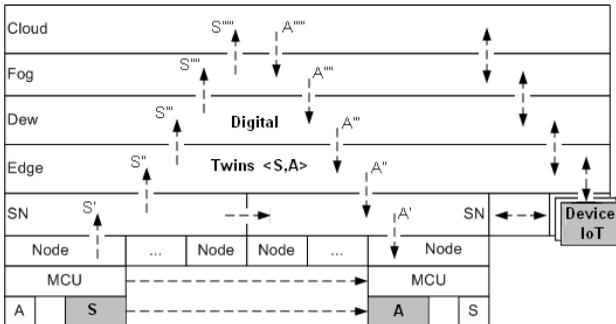


Fig. 31.1 – Methods of an interaction of network nodes

The logical link of the sensor to the executive device can be implemented: interconnectivity by sending a message  $S'$  that displays the state  $S$  to another network as a package  $A'$  representing the state  $A$ . In the case of different interfaces and protocols of associated sensor networks, a series of transformations  $S' \rightarrow A'$  is required. In the absence of direct network connection, Edge computing is used, operating with a digital representation of  $\langle S'', A'' \rangle$ , that is called Digital Twin [2]. Accordingly, Digital Twin at the levels Dew, Fog, Cloud, will be used in various forms, but displaying the current state of  $\langle S, A \rangle$ .

Synchronization  $\langle S', A' \rangle$ ,  $\langle S'', A'' \rangle$ , ...,  $\langle S''''', A'''''' \rangle$  requires certain computational costs and the expenditure of traffic, and hence - time costs. Generally, there are time intervals of the desynchronization in the meaning of various Digital Twin, which can affect the operation of systems sensitive to such uncertainties. Here, it should be entrusted Edge computing with critical to communication delays and direct interaction of sensor networks.

### ***31.1.2 EDGE Computing for Association Sensor Networks***

Cloud technologies implement a variety of IoT services for storage, data processing and remote access (Amazon Web Services IoT Platform, Microsoft Azure IoT Hub, Google Cloud IoT, IBM Watson IoT Platform, CISCO IoT Cloud Connect, ThingSpeak, etc.). This simplifies the development of applications for IoT. But these technologies have a lot of disadvantages [3]. There are restrictions on the intensity of the data flow for storage in the Cloud stores and a significant delay in access to these data. For a growing number of IoT sites the permanent access to the Internet is required. Also the increasing of bandwidth is needed. It prevents the use of Cloud technologies in real-time management systems and critical appointments. Excessive traffic arising in the process of access to remote resources causes increasing energy costs and the cost of access to information.

The solution is to approach resources to their consumers. Consequently, Fog computing [4-6], and then Dew computing [7] have appeared.

The IoT feature is the importance of the level boundary interaction of TCP/IP – oriented components and services with sensor networks (SN) and individual devices – stationary, mobile, moveable (EDGE computing). This level of interaction is difficult for formalizing because of variety of devices types, interfaces, network protocols, numerous vulnerabilities and strong requirements to power of IoT devices. Acquiring necessary skills for building the boundary level of IoT is the pressing challenge of training specialists in networking technologies.

Let's consider independent sensor networks  $SN_1$  (head node 1, internal nodes 11, 12, 13 and boundary nodes  $BN_{11}$ ,  $BN_{12}$ ,  $BN_{13}$ ),  $SN_2$  (head node 2, internal nodes 21, 22, 23 and boundary nodes  $BN_{21}$ ,  $BN_{22}$ ,  $BN_{23}$ ),  $SN_3$  (head node 3, internal nodes 31, 32, 33 and boundary nodes

$BN_3$ ,  $BN_{13}$ ,  $BN_{23}$ ). In Fig. 31.2, double lines show internetwork data flows. Let the data for node 21 come from the access point through the boundary node  $BN_1$ .

The chain is constructed:  $Data1 \rightarrow BN_1 \rightarrow BN_{12} \rightarrow 21$ . Data is delivered from node 31 via the chain:  $31 \rightarrow 3 \rightarrow BN_{23} \rightarrow 2 \rightarrow BN_{12} \rightarrow 1 \rightarrow BN_1 \rightarrow Data1$ . A shorter way, in the presence of  $BN_{13}$  is:  $31 \rightarrow 3 \rightarrow BN_{13} \rightarrow 1 \rightarrow BN_1 \rightarrow Data1$ .

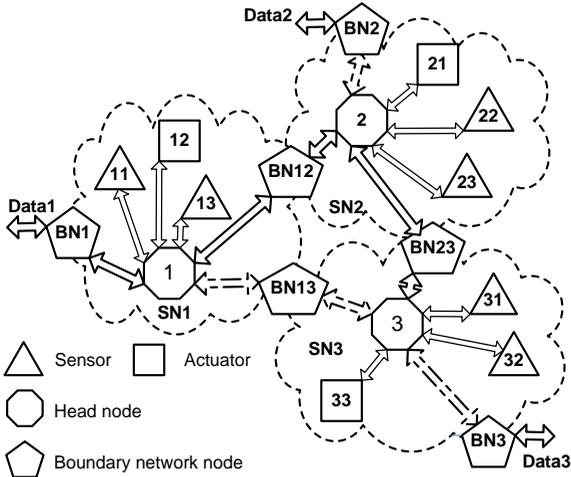


Fig. 31.2 – Combining of sensor networks with internetwork data transport

- Networks  $SN_1$ ,  $SN_2$ ,  $SN_3$  can be used for data exchange between the head and inner nodes and external networks via  $BN_1$ ,  $BN_2$ ,  $BN_3$ . The following conditions must be met: support for packet switching in selected networks; sufficient length of network messages for organizing data transfer over  $SN_1$ - $SN_3$  network protocols. Data transfer rate in networks ensures an acceptable delay in transmission of data packets; network traffic must have sufficient redundancy to accommodate additional traffic; head nodes 1, 2 and 3 allow the extension of the basic set of functions; energy costs for implementation of additional services should not extend beyond established limits.

- A number of sources [6,8-10] describe the results of the interaction of wired and wireless networks. The principal possibility of reliable transportation of CAN-packets through the IP network is shown. However, many sensor networks have too limited capabilities of interfaces and protocols to implement additional functions. This may require profound changes to the services of elements 1, 2 and 3, will affect a number of existing protocols and will lead to the emergence of new protocols that support prospective platforms of sensory networks.

Thus, the problem of constructing a common information field from independent heterogeneous networks is solved as follows. Network interfaces are assumed by the boundary nodes (BN). There are  $BN_1, BN_2, BN_3$  for external access to  $SN_1, SN_2, SN_3$  and there are  $BN_{12}, BN_{13}, BN_{23}$  for the interaction of networks. Boundary nodes that perform the function of gateways have the possibility of a simultaneous presence in at least two networks between which interaction is organized.

Each network controller through a network interface is related to the nodes of its network - sensors and drives and it performs a set of basic functions (services). Expansion of the set of functions gives access to the network controller from the side of the boundary node, which provides transport of data from one network to another.

Head nodes (network controllers) form requests (commands) to sensors and drives receive response messages in accordance with the internal logic of the network functioning. To provide communication with the global network of all nodes, without exception, that generates and receive data, it is advisable to use one entry point for a cluster of nodes within one or more sensor networks [8, 9].

### ***31.1.3 Types and structure of IoT devices***

Devices on which IoT conditionally is under construction share on:

1. Simple attached device.
2. Intelligent device.
3. Border gateway.

The simple attached device generates data, performs instant operations and carries out data transmission. As a rule, contains the microcontroller with limited resources, built in by software, does not

demand big costs of the equipment, provides basic functions of connection, basic tools of safety. These are the most mass devices to which specific, often contradictory requirements are imposed.

The intelligent device contains the microprocessor or SoC, the operating system and considerable hardware resources (Ready IoT). Provides data analysis on peripheral sections, support connectivity across multiple networks, makes decisions and carries out local calculations. Provides the maximum level of safety, controllability, interaction and compatibility, reliable work of solutions, support of cloud computing, the user interface and reduces data transmission cost. The border gateway is the intelligent device for computing Edge with the high level of safety, minimizes the problems connected with interaction of the physical and virtual world and scaling of the IoT systems.

IoT projects increasingly rely on existing out-of-the-box solutions. Benefits [5]:

- quicker Time To Market;
- access to crucial skills;
- secure by design;
- optimized to work with wider ecosystem;
- scale with ease;
- enable a more end-to-end offering.

The choice for independent development usually is accepted for simple and parts of intelligent devices. The structure of these devices is presented in a general view on Fig. 31.3.

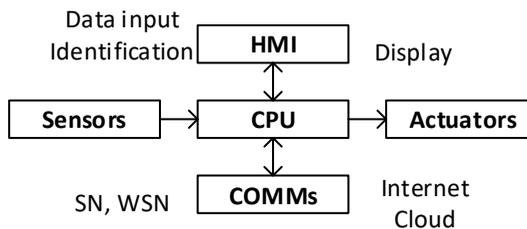


Fig. 31.3 – Block scheme of device IoT

Kernel of devices (CPU) can be ready (Fig. 31.4) or independently projectable modules on the basis of 8-32 bit microcontrollers [11-13].

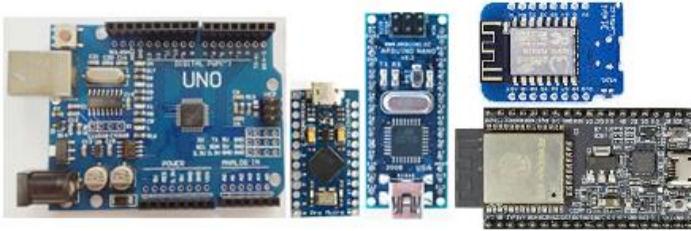


Fig. 31.4 – Kernel of IoT devices

Samples of sensors, indication and data entry, communication means in modular or submodular execution for prototyping or creation of end devices are widely presented at the market (Fig. 31.5).

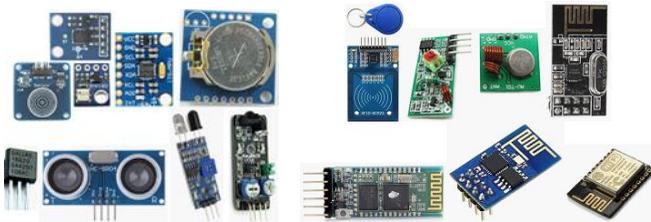


Fig. 31.5 – Sensors and communication modules of IoT devices

## 31.2 Prototyping and rapid development principles

### 31.2.1 Main concepts

There are following six phases in every IoT based system development life cycle model [14-16]:

- requirement gathering and analysis;
- design;
- implementation or coding;
- testing;
- deployment;
- maintenance.

Development of IoT of applications is the iterative process allowing eliminating errors and mismatching to requirements at different stages. Errors of initial stages of development are most difficult eliminated.

Design of the IoT components of systems includes:

1. Providing functional requirements:
  - modeling for decision-making (Matlab, Simulink);
  - distribution of functions between the IoT components of systems, use of support from mobile devices (smartphones, tablets and so forth);
  - rational distribution of functions between equipment rooms and software (minimization of hardware expenses);
  - use of the previous developments.
2. Depreciation of components:
  - rational choice of element base;
  - use of open platforms.
  - Reduction of weight and dimensional parameters:
  - rational configuration;
  - choice of cases of elements;
  - replacement of bulky elements (power supply, indication, management).
3. Decrease in terms of development of components:
  - use of the previous developments, resources of ecosystems;
  - rational choice of development tools, compilers, simulators.
4. Use of the previous developments.
5. Decrease in energy consumption (collecting energy for a power supply).
6. Reliability augmentation (resistance to failures, power failures and so forth).
7. Reduction of expenses on service.
8. Work in severe conditions of the environment.
9. Adaptation to new requirements.
10. Standardization of interfaces for Sensors, Actuators, network and between network interactions.
11. Interaction with services Edge, Dew, Fog, Cloud.
12. Complex use of different platforms.

At different development stages focus of fast prototyping is transferred to different components. Existence of lightweight IoT middleware for rapid application development is important [16-18].

### 31.2.2 Techniques for prototyping of IoT systems

Physical prototypes of simple devices can have virtual analogs, for example, in the environment of Proteus (Fig. 31.6). Virtual devices cannot reflect fully behavior of physical prototypes, but considerably accelerate intermediate prototyping.

Availability of components of an ecosystem of Arduino, Breadboards to fast assembly of prototypes cause their wide circulation, especially in education [11, 12].

All largest vendors of microprocessors, microcontrollers, SoC, communication means are guided by IoT and offer both end-to-end solutions, and means of fast prototyping. Elements of compatibility with shields of an ecosystem Arduino are often entered and in the same format own payments are offered.

ON Semiconductor provides configurable, end-to-end, rapid prototyping platforms for the Internet of Things [19]. These platforms enable development of energy efficient solutions for smart homes/buildings, smart cities, industrial IoT (Predictive Maintenance, Asset Monitoring, etc.) and personal IoT (Wearables, activity monitors, etc.).

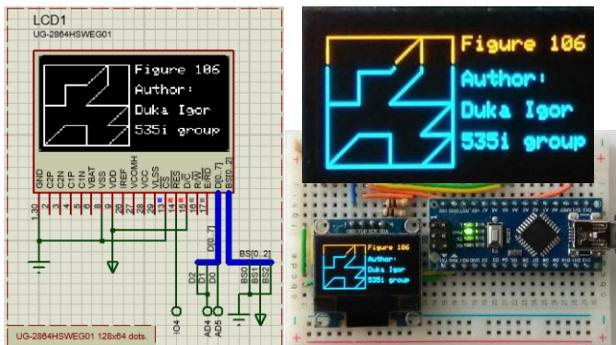


Fig. 31.6 – Means of prototyping of the simple IoT device

The IDK baseboard can be connected with different shields depending on the required IoT application. The IDK baseboard allows the user to create many types of IoT nodes and/or gateways depending on which shields are used with the baseboard. Programming/configuring the IDK requires the ON Semiconductor IDE software (Fig. 31.7).

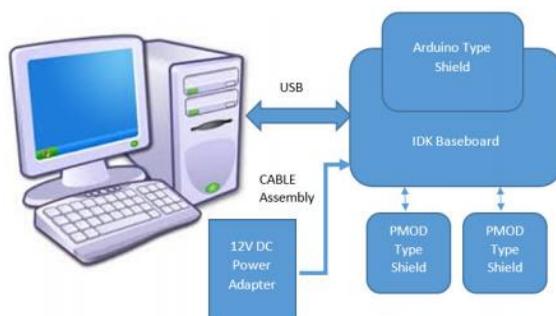


Fig. 31.7 – Hardware Setup

Based on the company’s highly sophisticated NCS36510 system-on-chip (SoC) with a 32-bit ARM® Cortex® M3 processor core, it has all the necessary hardware resources for constructing highly effective, differentiated IoT systems, along with a comprehensive software framework to attend to interfacing with the cloud (Fig. 31.8).



Fig. 31.8 – EVBUM2497/D IoT Prototyping Platforms

By attaching different shields to the IDK baseboard, a wealth of connectivity (WiFi, Sigfox, Ethernet, ZigBee and Thread protocols, etc.), sensor (motion, ambient light, proximity, heart rate, etc.) and actuator (with stepper and brushless motor driving, plus the ability to drive LED strings) options can be added to the system. This means that

compromises do not have to be made, and the most suitable technology can be chosen.

Offering a wide range of choices including configurable hardware, multiple cloud connectivity, easy-to-use development software, and application examples, these platforms reduce time-to-market and allow rapid deployment of IoT-enabled products.

Designed for expert makers, entrepreneurs, and industrial IoT companies, the Intel Edison module provides easier prototyping with a fully open source hardware and software development environment. It supports WiFi and BLE 4.0 connectivity (Fig. 31.9). This kit contains eleven, selected Grove sensors and actuators. It can be used to track indoor environment as well as to create smart-home applications [20].



Fig. 31.9 –Intel® Edison and Grove IoT Starter Kit

At production of single copies or the small IoT series of solutions these platforms are final option.

### ***31.2.3 Rapid Prototyping of Fragments of Internet of Things***

The option of rapid prototyping of a network fragment using wired and wireless access that realized with use a WiFi router shown in Fig. 31.10.

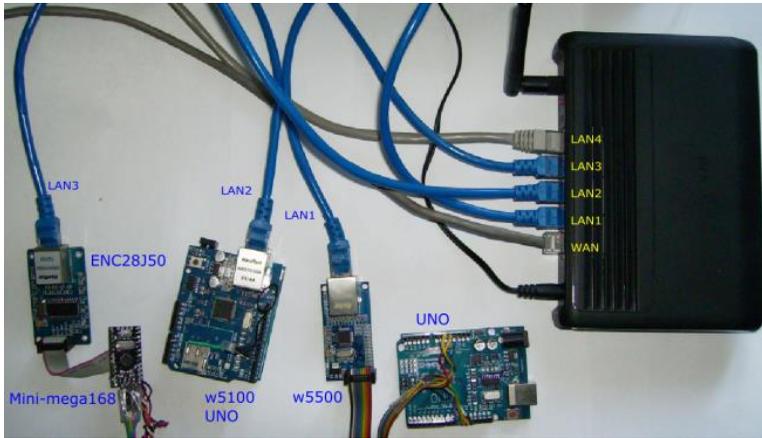


Fig. 31.10 – Rapid prototyping of the IoT network fragment

There are four Ethernet ports for connecting the end devices (Ethernet MCU) and the local server, the WiFi access point for connecting the IoT wireless devices: WiFi MCU (ESP 8266, Espressif ESP 32, etc.), SoM Raspberry Pi, laptops, smartphones, tablet computers. As the access point WiFi mobile devices GSM, DSL modems and WiFi MCU can be used. This creates a variety of tasks for building various Edge-level network configuration for building and analyzing IoT fragments, mastering promising IoT platforms.

Simple Ethernet MCUs are built using Ethernet - SPI converters Wiznet w5100, w5500, Microchip ENC28J60 and microcontroller platforms [21]. Converters implement TCP/IP protocol in hardware. Microcontrollers can be connected to sensors and actuators, perform the functions of the boundary nodes of sensor networks, and exchange information among themselves using built-in interfaces. Using multiple routers and connected devices allows the local server to simulate interaction at the boundary level of higher IoT levels (Fig. 31.11).

The prototype is the base for the development and research of industrial automation systems, a smart home that is based on technologies of the Internet of Things.



Fig. 31.11 – Example of use of shield w5100 for remote control via the Internet

#### ***31.2.4 Modeling of Access to WEB Resources***

Along with rapid prototyping, an effective tool for developing and debugging an application for the Internet of Things is the model approach. Thus, the Proteus modeling and development environment allows investigate the behavior of devices based on wired access ENC28J60 to a local network and also based on emulating access to web resources by intercepting TCP/IP packets.

Figure 5 shows the model view and the browser window with the result of querying the IP address of the device. This simple and affordable tool allows you to gain the skills of organizing an exchange using HTTP pages in a network with the TCP/IP protocol. Limited resources of microcontrollers and features of ENC28J60 allow using highly shortened HTTP pages that can be placed in one Ethernet frame and occupy the amount of available memory.

Proteus allows simulate the exchange between nodes of sensor networks, and also the interaction of microcontrollers with digital and analog sensors (temperature, humidity, pressure, approach, etc.), various actuators (lighting, electric drives, relays, etc.), indicators, alarms and various converters (Fig. 31.12).

### 31. Prototyping and rapid development of IoT systems

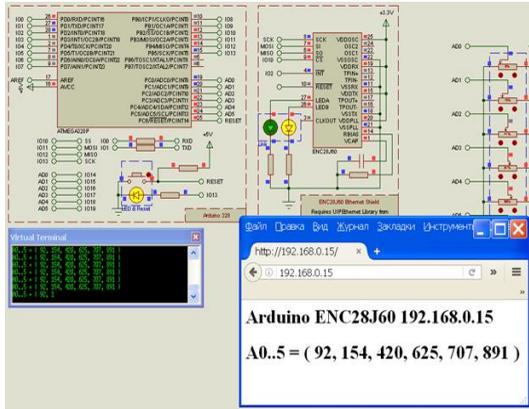


Fig. 31.12 – Modeling access to Web resources using the Proteus.

The program - a network analyzer for computer networks of Ethernet - Wireshark allows the user to browse all traffic passing on network connected with the modelled device [22]. The program is distributed for free. On Fig. 31.13 analysis of a frame of exchange with the browser and local control of data is given.

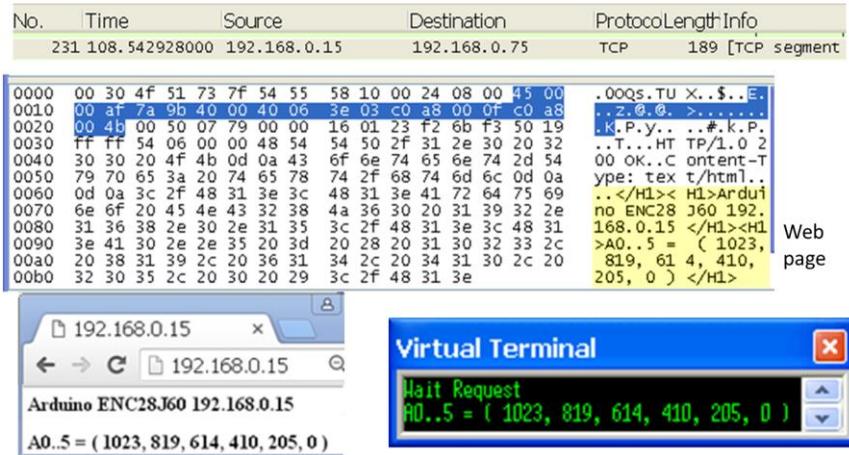


Fig. 31.13 – Analysis of the data field of Frame

For convenience of search/viewing of information on the necessary packets in the Wireshark program it is possible to filter the taken packets to the IP address or port number.

### **31.3 Cases of IoT systems rapid development**

A number of the practical tasks connected with development of systems of collection of information on mobile platforms complicate, and in some cases exclude debugging of hardware-software complexes in the real environment. It is necessary to resort to use of models of elements of interaction with networked environment, sensors and actuation mechanisms.

Case1 contains process description of independent development and prototyping of the attached IoT device for data collection of measurements with use of language of the low level. Can be similarly constructed some other IoT devices with tight restrictions on the used resources.

Case2 shows difference of modern approach to fast design of the device of monitoring of temperature with use of a cloud service on the basis of Arduino ecosystem by Ready IoT.

#### ***31.3.1 Case 1. Collection of data on an angular position of the mobile platform of road laboratory***

For condition monitoring of roads the ROMDAS® system (ROad Measurement Data Acquisition System) [23] is widely used and some other systems. There is similar road laboratory JIBC-3 [24] domestic development on the basis of the car. JIBC-3 contains a ruler from 18 laser sensors measuring a profile of a paving, HRDMI (High Resolution Distance Measurement Instrument) – the odometer for measurement of the passable way on the basis of an encoder, Digital inclinometer for measurement of slope angles, navigation instruments and the camera of video monitoring.

Each sensor creates a data flow with results of measurements through certain distances. These distances are counted by means of HRDMI which creates the pulse sequence with a frequency proportional to motion speed. Data from sensors are taken off through a certain number of div of impulses of HRDMI. Data in a special format also remain in memory of the on-board computer for further processing

(definition of places of damage of coverings, calculation of the IRI index of flatness of roads). For creation of a profile of the road, it is necessary to consider slope angles (pitch of X and a roll Y) platforms.

Data of an inclination on X, Y from Digital inclinometer are transferred with a frequency of 10 Hz and have a 22-character text format <D0... D21>:

<D0 ... D10> = "X=±xx.xxx", <CR>, <LF>

<D11 ... D21>= "Y=±xx.xxx", <CR>, <LF>

Example of data from an inclinometer:

"X=+12.345", \$0D, \$0A, "Y=-09.876", \$0D, \$0A

Received by X,Y will be transformed to 16-bit branching codes (shortint) of values with scaling ratio 1000 and about one tetrad is transferred in byte:

X=+12345= 0x3039, <0x#9, 0x#3, 0x#0, 0x#3> – для X=+12.345°

Y=-9876 = 0xD96C, <0x#C, 0x#6, 0x#9, 0x#D> – для Y=-09.876°

Here 8 tetrads of codes X and Y of each measurement are numbered by 4-bit codes (#) synchronization (0,1,2, ..., E,F,0.1.). New measurement gets the future issue from a cyclic row - 0,1,2, ..., E,F,0.1. Data from other sensors have a similar format.

Interaction of hardware-software functional modules of the channel of measurement of slope angles is given in Fig. 31.14.

Communication means and the software of the computer are provided by multichannel data reception of measurements, forming of the general data array with a binding to local maps for storage and further processing and use of results in local and cloud services.

MCU – a set of hardware-software modules of conversion of formats and temporary parameters of messages with results of measurements.

MCU modules interact as follows.

1. Receiver in real time analyzes an asynchronous data flow from Digital inclinometer.
2. Correct messages are used for conversion of values of corners X, Y from a text format in binary branching code of shortint.
3. The template of the output message forms.
4. The moments of the beginning of transfer of the next day off the message are defined.
5. Transmitter sends the 8th byte packets of the output message.

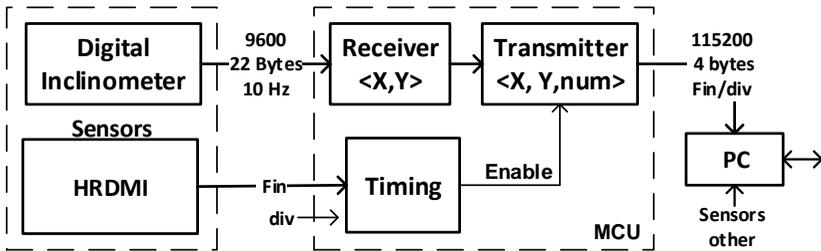


Fig. 31.14 – Functional modules of the channel of measurement of slope angles

Functions can be implemented program, equipment rooms and software and hardware tools of different microcontrollers. In the considered channel of measurements widespread AVR ATmega162 microcontrollers with two transceivers were used [25].

### Rapid software development

Considering intensity of exchange and implementation of conversions language of the low level - the graphic Algorithm Builder assembler is in real time selected. In comparison with the traditional assembler time of development of the program is several times reduced.

Typical for analysis and forming of message bars at exchange in IoT functional modules of the program:

- wait\_Rx1 – reception of characters
- read\_RxD1(value) – waiting of the character
- read\_dec (value1, value2) – waiting of digit
- read\_sign(signXY) – waiting of the sign of number
- OutMess – the output message
- Chr\_Bin – branching code from a line
- messXY – the output message
- Timer\_0\_Overflow – synchronization of transmission of messages
- USART0\_DR\_Empty – transfer of byte
- USART0\_Transmit\_Complete – the termination of a cycle of transfer

Fragment of programming module of reception of a corner of pitch of X (Fig. 31.15) it is similar to the module of reception of angle of heel of Y.

The macro of read\_RxD1 (value) carries out check of the accepted character and saves it in the buffer, and at an error reception of the message is interrupted. So there is a syntactic control and ignoring of incorrect messages.

How to check operation of the module of reception of the message without field tests?

1. To replace function of waiting of input of the character wait\_Rx1 with function of reading test characters from in-memory string

```

|mess1
▪ "X=-12.333", $0D, $0A
▪ "Y=+29.999", $0D, $0A

```

2. Input of the message from the virtual terminal Proteus.
3. To connect the Digital inclinometer emulator to a prototype.

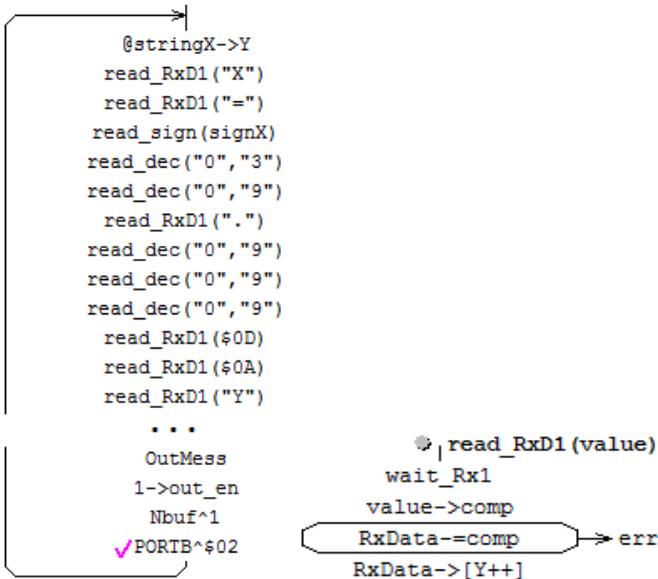


Fig. 31.15 – Reception of values of slope angles of X, Y

On the AVR microcontroller with asynchronous reception - the transmitter (ATtiny2313, ATmega8535, ATmega8/168/328, etc.) can implement programmatically the emulator of an inclinometer for transfer of value of fixed values of X and Y, or a series of values including incorrect.

For error trapping at consecutive exchange the additional bit of control of parity is used. AVR transceivers of microcontrollers support exchange of 9-bit codes, but the bit of Even Parity (Data bit 8) forms and processed programmatically. On the Fig. 31.16, the fragment of the program of data transmission with control of parity is shown. In the same way sending for transfer to the PC form.

In addition to functions of the Digital inclinometer emulator it is possible to assign function of emulation HRDMI to the microcontroller – generation of impulses with a program-controlled frequency of Fin. The internal timer counter having communication with an exit is for this purpose used.

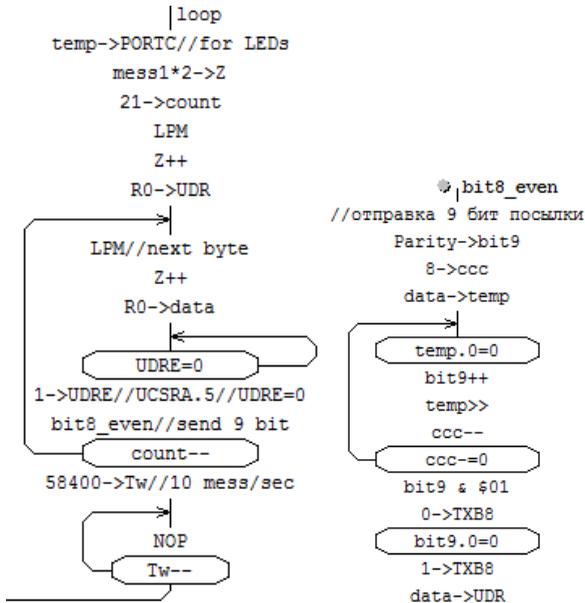


Fig. 31.16 – Transfer of the message with Even Parity bits

Generation of impulses happens hardware. It is possible to use management of frequency by means of external signals (Fig. 31.17). To each signal there corresponds the code of control of the timer.

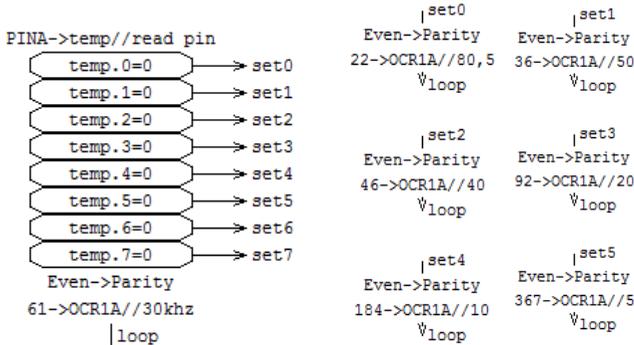


Fig. 31.17 – Fragment of the program of the choice of Fin (80.5 - 1.0 KHz)

For creation of the device the microcontroller with two asynchronous serial interfaces (USART0.1) is necessary for exchange on RS232, the hardware pulse counter HRDMI. The model - ATmega162 with clock rate 7.3728 MHz is selected. Setup in Algorithm Builder of speed and operation modes of Receiver USART1 for communication with an inclinometer and Transmitter USART0 for contact with the computer is shown on Fig. 31.18.



Fig. 31.18 – Setting mode USART1 and USART0

For prototyping the developed earlier printed circuit board for AT90S8515/ATmega8515/ATmega162 with sufficient number of external connectors was used (Fig. 31.19).



Fig. 31.19 – Prototype of microcontroller devices

After modification of the connection diagram COM port it is used for connection with the computer, the contact of Fin connects to HRDMI, Fin/div - contact of a pilot, a0.2 – inputs for the jumpers installation of the choice of div value. Indication of a power supply (Led), the button of reset (Reset) and the connector of onboard programming (ISP) is provided.

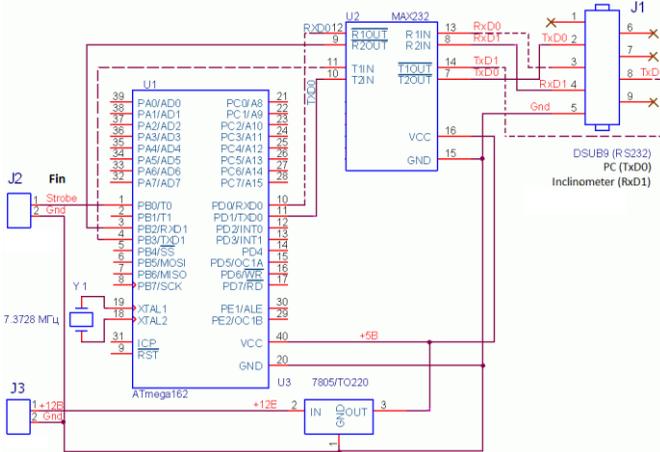


Fig. 31.20 – Function chart of the device

To the Com port connector (J1) according to (Fig. 31.20 with can be connected both an inclinometer (RxD1), and the computer (TxD0).

The prototype of the Digital inclinometer and HRDMI emulator is constructed on the basis of a board with the ATmega8535 microcontroller ("Sensors" on Fig. 31.21).

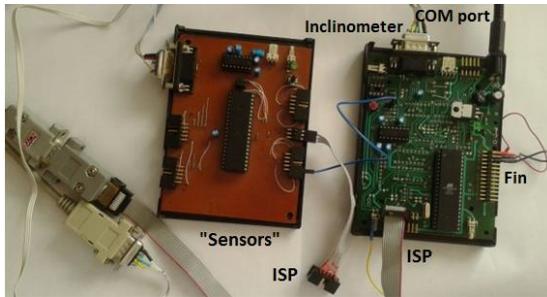


Fig. 31.21 – The stand for debugging of the microcontroller device

Verification of signal outputs of the emulator is executed with use of the logical analyzer. The analyzer has functions of record of time diagrams of signals, measurements of temporary parameters and decoding of sendings of standard interfaces (Fig. 31.22).

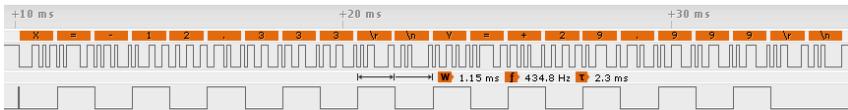


Fig. 31.22 – Signals emulator Digital inclinometer and HRDMI

Here from the emulator the line "X = 12.345\r\nY=-09.876\r\n" and impulses with a frequency of 997 Hz arrives.

For control of data transmission from the device on the COM1 computer the Terminal v1.9 program with the settings shown on is used Fig. 31.23. In a window of the terminal we receive the expected flow of enumerated output messages. The prototype set onboard mobile road laboratory passed rather long period of operation without additional debugging in field conditions.

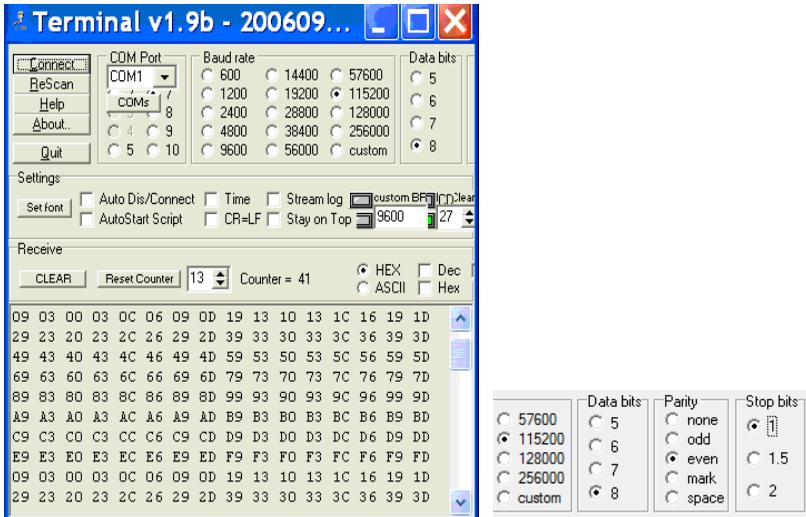


Fig. 31.23 – Control of data transmission from MKU to the computer

### 31.3.2 Case 2. Monitoring of temperature with use of a cloud service of Thingspeak and access on WiFi

On Thingspeak.com the channel for monitoring of temperature is registered. It is required values of temperature from the digital DS18B20 sensor periodically to send for storage and visualization to the canal and to exercise control of receipt of data. There is a ready decision for access to the Internet through WiFi – WeMos D1 R1 based on SoC ESP8266-12E from Espressif [27, 28]. Necessary elements and communications between them are shown on Fig. 31.24.

WiFi the microcontroller in WeMos D1 R1 works according to an algorithm on Fig. 31.25.

Libraries of high-level functions:

```

#include <OneWire.h>
#include <DallasTemperature.h>
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>

```

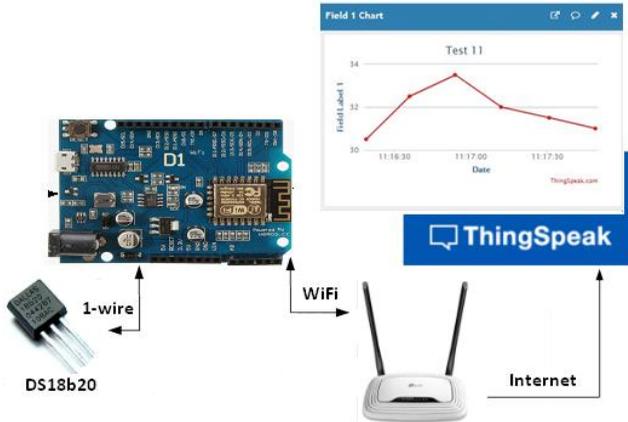


Fig. 31.24 – Monitoring of temperature with use of Thingspeak.com service

Parameters of WiFi network:

```
const char* ssid = "ssid ";
const char* password = "password ";
```

Attributes of access to thingspeak.com:

```
const char* host = "api.thingspeak.com";
const int httpsPort = 443;
```

Initialization of the sensor of temperature:

```
#define ONE_WIRE_BUS 2
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
```

Establishment of WiFi of connection:

```
WiFiClientSecure client;
WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
 delay(500);
 Serial.print(".");
}
}
```

In a basis cycle the value of temperature, sending to thingspeak.com channel and control reading is read out (Fig. 31.26):

```

void loop(void)
{
 ...
 sensors.requestTemperatures();
 ...
 sendData(sensors.getTempCByIndex(0));
 delay(1000);
}

```

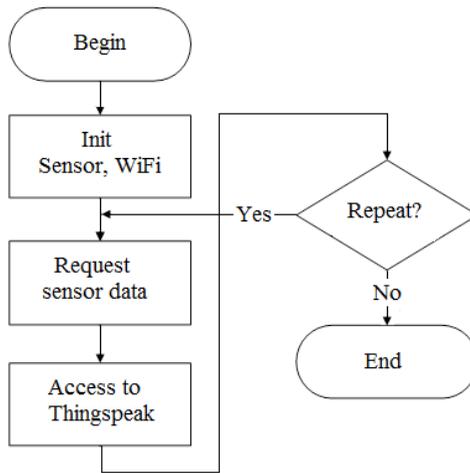


Fig. 31.25 – Scheme of an algorithm of functioning of the device

Function of formation of URL of inquiries thingspeak.com for sending data:

```

void sendData(float temp){
 if (!client.connect(host, httpsPort)) {
 Serial.println("connection failed"); return;
 }
 String url = /update?api_key=ZS2KDK8HRD&field1=
"+String(temp);
 client.print(String("GET ") + url + " HTTP/1.1\r\n" + "Host: "
+ host + "\r\n" + "User-Agent:
BuildFailureDetectorESP8266\r\n\r\n");
}

```

```
Serial.println("Request sent to " + String(host));
while (client.connected()) {
 String line = client.readStringUntil('\n');
}
String line = client.readStringUntil('\n');
}
```

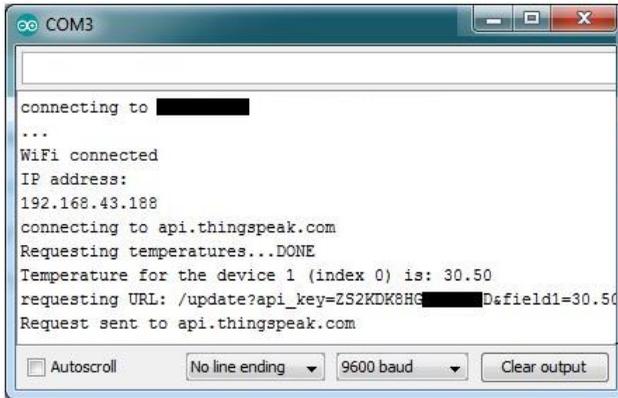


Fig. 31.26 – Control of establishment of connection, sending and data acquisition in the IDE Arduino terminal

In this example more difficult task, than in Case1 is solved much more simply and quicker thanks to ready hardware and libraries of high-level functions in the IDE Arduino coding environment.

### 31.4 Work related analysis

Technologies of development and prototyping on the basis of an ecosystem of Arduino and other platforms are used at many universities of Ukraine – National Aerospace University "Kharkiv Aviation Institute" [31], Zaporozhye National Technical University [32], Odessa National Polytechnic University [33] and others. Here Circuits, Tina, Fritzing, etc. is considered as physical prototyping and development of printed circuit boards of devices, and use of computer circuitry models in Proteus, Autodesk 123D.

In courses of many the US and EU countries universities, similar platforms and technologies are used. For example:

- course ME 2011 «Arduino Microcontroller» University of Minnesota [34];
- course “Physical Computing with the Arduino” Middlesex University London [35];
- course Stanford University Explore Courses - ARTSTUDI 130: “Interactive Art: Making it with Arduino”, EE 392B: “Industrial Internet of Things” [36];
- course Comp 366 / 450 “Microcontrollers - Building The Internet of Things (IOT)” Loyola University Chicago [37].

Course of ECE 4760 "Designing with Microcontrollers" Cornell University. School of Electrical and Computer Engineering is constructed on use of 8-bit platforms with architecture of AVR, and in the last years - 32-bit PIC32 platforms with architecture of MIPS. The numerous projects completed by development of prototypes are presented [38]. 10 best (according with opinion of authors) courses Arduino & IoT & Certification are described on [39].

### ***Conclusions and questions***

Objects of the physical world can be connected by one or several touch networks in about tens of sensors and actuation mechanisms and programmable computing modules (the robot, the car, the house, the machine, etc.). The state and behavior of object - "thing" is defined by data flows in networks, and on Edge, Dew, Fog and Cloud levels of global network are formed copies of his digital double (Digital twin).

Digital doubles have to reflect adequately a condition of physical objects, and impact on doubles – to cause the corresponding reaction physical objects and change of conditions of all copies of Digital twin for representation to users. Development and prototyping of such IoT components of systems and their deployment are very difficult. On the other hand, sensor networks and elements of network interconnection consist of rather simple devices with available development tools and prototyping. Rapid development assumes availability of functionally full range of elements cuts of fast assembly of devices rapid developments of programs of their debugging. Classical approach - development of the device with the program languages of the low level, but with visually way programming is considered. The example of modern approach on the basis of the open platform allows to implement

quickly devices of monitoring and remote control with access to Web services.

In order to better understand and assimilate the educational material that is presented in this section, we invite you to answer the following questions.

1. What is understood as the connected device?
2. What requirements are imposed to IoT devices?
3. In what difference between Edge, Dew, Fog and Cloud?
4. What function is performed by boundary knots of networks?
5. What stages are included by development of IoT of a system?
6. What enters a concept of the IoT platform?
7. What communication potential does the microcontroller have?
8. Call ready decisions for IoT.
9. What devices can perform the computing Edge functions?
10. How the device gets Internet access?
11. What potential does Proteus have?
12. What order of development of the microcontroller IoT device?
13. In what advantage of visual programming?
14. Why emulators are used?
15. Call means of fast prototyping of access to a cloud service?

### **References**

1. S. Kulkarni and S. Kulkarni, "Communication Models in Internet of Things: A Survey", *IJSTE - International Journal of Science Technology & Engineering*, vol. 3, no. 11, 2017.
2. R. Kienzler, "Digital twins and the Internet of Things", 2019. <https://developer.ibm.com/articles/digital-twins-and-the-internet-of-things/>. [Accessed: 25- Jun- 2019].
3. A. Botta, W. Donato, V. Persico, A. Pescap. "Integration of Cloud Computing and Internet of Things: a Survey". *Journal of Future Generation Computer Systems*, pp. 1-54, 2015.
4. S. Yi, Z. Hao, Z. Qin, and Q. Li, *Fog Computing: Platform and Applications*. Third IEEE Workshop on Hot Topics in Web Systems and Technologies, pp. 73-78, 2015
5. Fog computing: fog and cloud along the Cloud-to-Thing continuum. <https://www.i-scoop.eu/internet-of-things-guide/fog-computing-cloud-internet-things/> [Accessed 25 June. 2019].
6. N. Mohan, J. Kangasharju, "Edge-Fog Cloud: A Distributed Cloud for Internet of Things Computations". [https://www.cs.helsinki.fi/u/nmohan/documents/2016/EF\\_Nitinder\\_Jussi\\_UH\\_Final.pdf](https://www.cs.helsinki.fi/u/nmohan/documents/2016/EF_Nitinder_Jussi_UH_Final.pdf). [Accessed 25 June. 2019].

7. P. Ray, "An Introduction to Dew Computing: Definition, Concept and Implications".  
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8114187>. [Accessed 25 June. 2019].
8. S. Yuvraj, C. Jiannong, Z. Shigeng, *Edge Mesh: A New Paradigm to Enable Distributed Intelligence in Internet of Things*. IEEE ACCESS, 2017, Vol. 5, pp.: 16441-16458
9. T. Higuchi, H. Yamaguchi, and T. Higashino, *Mobile devices as an infrastructure: A survey of opportunistic sensing technology*. Journal of Information Processing, 23(2):94—104, 2015.
10. Brandon Keith Maharrey, Alvin S. Lim and Song Gao, "Interconnection between IP Networks and Wireless Sensor Networks". International Journal of Distributed Sensor Networks", December 4, 2012. <http://journals.sagepub.com/doi/full/10.1155/2012/567687>. [Accessed 25 June. 2019].
11. Marco Schwartz, *Internet of Things with Arduino Cookbook*. Packt Publishing, 2016.
12. P. Waher, *IoT: Building Arduino-Based Projects (+code)*. Apress. 2016.
13. P. Xiao, *Designing Embedded Systems and the Internet of Things (IoT) with the ARM Mbed..* Wiley. 2018.
14. F. Pramudianto, "Rapid Application Development in the Internet of Things: A Model-Based Approach", <https://publications.rwth-aachen.de/record/464316/files/464316.pdf> [Accessed 25 June. 2019].
15. Pádraig, S. and Lueth, K, "Guide to iot solution development", 2016. [<https://iot-analytics.com/wp/wp-content/uploads/2016/09/White-paper-Guide-to-IoT-Solution-Development-September-2016-vf.pdf>]. [Accessed 25 June. 2019].
16. G. Guan, W. Dong, Y. Gao, K. Fu and Z. Cheng, "TinyLink: A Holistic System for Rapid Development of IoT Applications". <https://ieeexplore.ieee.org/document/8116508>. [Accessed 25 June. 2019].
17. K. Karvinen, T. Karvinen, *IoT Rapid Prototyping Laboratory Setup*. International Journal of Engineering Education Vol. 34, No. 1, pp. 263–272, 2018
18. Configurable Rapid Prototyping Platform for The Internet of Things. [Online] Available at: <https://www.rs-online.com/designspark/iotidk-kit>. [Accessed 25 June. 2019].
19. EVBUM2497/D. IoT Development Kit (IDK). Quick Start Guide. [https://www.mouser.com/pdfdocs/ONsemi\\_IDK\\_QuickStart.pdf](https://www.mouser.com/pdfdocs/ONsemi_IDK_QuickStart.pdf). [Accessed 25 June. 2019].
20. Intel Edison and Grove IoT Starter Kit Powered by AWS. [http://wiki.seeedstudio.com/Grove\\_IoT\\_Starter\\_Kits\\_Powered\\_by\\_AWS/](http://wiki.seeedstudio.com/Grove_IoT_Starter_Kits_Powered_by_AWS/). [Accessed 25 June. 2019].

21. B. Hammell, *Connecting Arduino: Programming And Networking With The Ethernet Shield (+source code)*. CreateSpace Independent Publishing Platform, 2014.

22. S. Orgera, “How to Use Wireshark: A Complete Tutorial. Capture and view the data traveling on your network”. <https://www.lifewire.com/wireshark-tutorial-4143298>. Updated June 24, 2019. [Accessed 25 June. 2019].

23. ROMDAS System. <https://romdas.com/romdas-system.html> [Accessed 25 June. 2019].

24. І. Кіяшко, Р. Смолянюк, Д. Новаковський, О. Пархоменко and О. Мінаков, *Діагностика стану покриттів новітніми ходовими дорожніми лабораторіями: сучасний стан та перспективи розвитку*, Автомобільні дороги, no. 5(229), pp. 31-36, 2012.

25. Atmel AVR ATmega162 datasheet. [https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2513-8-bit-AVR-Microcontroller-ATmega162\\_Datasheet.pdf](https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2513-8-bit-AVR-Microcontroller-ATmega162_Datasheet.pdf). [Accessed 25 June. 2019].

26. G. Gromov, "Algorithm Builder for AVR", Atmel applications journal. [http://ww1.microchip.com/downloads/en/DeviceDoc/avr\\_builder.pdf](http://ww1.microchip.com/downloads/en/DeviceDoc/avr_builder.pdf). [Accessed 25 June. 2019].

27. I. Hendry, *Learn about the ESP8266 using Wemos shields*. Amazon Digital Services LLC . 2019.

28. M. R. Thakur, *NodeMCU ESP8266 Communication Methods and Protocols: Programming with Arduino IDE*. Amazon Digital Services LLC . 2018.

29. Global High-Density Interconnect (HDI) PCB Market – Industry Analysis and Forecast (2018-2026). <https://www.maximizemarketresearch.com/market-report/global-high-density-interconnect-hdi-pcb-market/30122/>. [Accessed 25 June. 2019].

30. R. Marvin, “The Best Low-Code Development Platforms for 2019”. <https://www.pcmag.com/roundup/353252/the-best-low-code-development-platforms>. August 10, 2018. [Accessed 25 June. 2019].

31. <https://khai.edu/en/>, <https://csn.khai.edu/>.

32. <http://www.zntu.edu.ua/zaporozhye-national-technical-university>.

33. <https://opu.ua/en>.

34. <http://www.me.umn.edu/courses/me2011/arduino/>.

35. <http://www.mdx.ac.uk/courses/summer-school/courses/physical-computing-with-the-arduino>.

36. <https://explorecourses.stanford.edu/>.

37. <http://cs.luc.edu/whonig/comp-366-488>.

38. <http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/>

39. <https://digitaldefynd.com/best-arduino-iot-tutorial-certification-course-training/>

УДК 62:004=111

173

Рецензенти: Dr. Mario Fusani, ISTI-CNR, Піза, Італія

Dr. Olga Kordas, KTH University, Стокгольм, Швеція

Viktor Kordas, KTH University, Стокгольм, Швеція

**173 Інтернет речей для індустріальних і гуманітарних застосунків. У трьох томах. Том 1. Основи і технології /** За ред. В. С. Харченка. - Міністерство освіти і науки України, Національний аерокосмічний університет ХАІ, 2019. -547 с.

ISBN 978-617-7361-82-3

Книга, що складається з трьох томів, містить теоретичні матеріали для лекцій та тренінгів, розроблених в рамках проекту Internet of Things: Emerging Curriculum for Industry and Human Applications / ALIOT, 573818-EPP-1-2016-1-UK-EPPKA2- CBHE-JP, 2016-2019, що фінансується програмою ЄС ERASMUS +. Том 2 описує моделі, методи моделювання та розробки для Інтернету речей (IoT). Книга складається з 4 частин для відповідних докторантських курсів: моделювання систем на основі IoT (розділи 16-19), програмно-визначувані мережі і IoT (розділи 20-23), надійність і безпека IoT (розділи 24-27), розроблення і впровадження систем на основі IoT (розділи 28-31).

Книга підготовлена українськими університетськими командами за підтримки колег з академічних закладів країн ЄС, що входять до консорціуму проекту ALIOT.

Книга призначена для магістрантів і аспірантів, які вивчають технології IoT, програмну і комп'ютерну інженерію, комп'ютерні науки. Може бути корисною для викладачів університетів і навчальних центрів, дослідників і розробників систем IoT.

Рис.: 158. Посилань: 430. Таблиць: 45.

Ця робота захищена авторським правом. Всі права зарезервовані авторами, незалежно від того, чи стосується це всього матеріалу або його частини, зокрема права на переклади на інші мови, перевидання, повторне використання ілюстрацій, декламацію, трансляцію, відтворення на мікрофільмах або будь-яким іншим фізичним способом, а також передачу, зберігання та електронну адаптацію за допомогою комп'ютерного програмного забезпечення в будь-якому вигляді, або ж аналогічним або іншим відомим способом, або ж таким, який буде розроблений в майбутньому.

## Анотації розділів

Розділ 16 присвячений опису загальних принципів функціонування плати Ардуіно і симуляції її роботи. Показані відмінності між фізичною та комп'ютерної симуляцією. Описуються методи симуляції, які можуть бути застосовані для плат Ардуіно. Наведено порівняльний аналіз різних програмних засобів, які можуть бути використані для симуляції. Детально описана робота з програмним комплексом протеус.

У розділі 17 розглядається тривірневе моделювання IoT/ІоЕ систем в їх структурі, поведінці і процесах синхронізації. Запропоновано візуальний моделінг, моделювання і перевірку архітектури, функціональності та часових особливостей IoT/ІоЕ систем і їх компонентів в статичному і динамічному режимах з використанням UML діаграм, мереж Петрі, часової логіки, відповідних методик та інструментів. Показано особливості моделювання на основі еволюційних генетичних і мультиагентних технологій.

При дослідженні надійності мікропроцесорних систем часто застосовується математичний апарат Марковських і напівмарковських моделей. Функціонування систем Інтернету речей при певних прийнятих припущеннях може бути описано за допомогою даних моделей. У розділі 18 наведено відомості про особливості створення Марковських і напівмарковських математичних моделей для опису процесу функціонування системи Інтернету речей. Наведено і описані допущення при розробці подібних моделей. Побудовано та досліджено Марковські моделі готовності систем Інтернету речей.

Розділ 19 присвячено моделюванню взаємодій в IoT системах. Авторами розглянуто архітектуру систем і загальні шаблони для моделювання взаємодій. Так як методи і моделі, що використовуються для проектування взаємодії залежать від складності проєктованих систем і класу вирішуваних завдань, в розділі розглядаються чотири приклади, що використовують різні підходи. У прикладі з віддаленої лабораторією GOLDi

демонструється застосування моделей FSM і Крипке, при проектуванні системи голосової навігації для Смарт-Кампусу використовувалися IFML моделі, а для моделювання кіберфізичних систем використовувалися «цифрові-двійники», що було показано на прикладі лабораторії ISRT.

У розділі 20 розглядаються основи технології програмно-конфігурованих мереж – базові принципи побудови та функціонування, основоположні технології, архітектурні особливості. Робиться також акцент на фундаментальних відмінних рисах технології, історичних передумовах, що сприяли виникненню останньої. Особливу увагу приділено огляду еволюції специфікації OpenFlow, яка формує базис забезпечення уніфікованого механізму взаємодії між контролером і комутаторами.

У розділі 21 розглядаються питання програмування і моделювання програмно-конфігурованих мереж. Аспекти програмування розглянуто на прикладі мови програмування Python. Наводяться та пояснюються базові команди конфігурування топології мережі, зокрема команди, присвячені вирішенню питань автоматизації названих дій. Приділяється увага середовищу моделювання Mininet і відповідній графічній оболонці MiniEdit.

У розділі 22 розглядаються питання пов'язані з низкою дослідницьких проблем що виникають при реалізації специфічних QoS моделей SDN шляхом розробки і впровадження алгоритмів і підходів, які забезпечують ефективну роботу SDN в IoT. Проаналізовано останні тенденції в використанні алгоритмів для технології SDN з точки зору їх придатності для створення і обслуговування великих магістральних мереж SDN / OpenFlow в інфраструктурі IoT. Обговорюються перспективи прогнозування продуктивності SDN з використанням методу об'єднання даних.

У розділі 23 аналізуються іноваційні, технологічні та бізнесові причини появи та розвитку методології Development and Operations (DevOps). Увага зосереджується на добре відомих платформах AWS, MS Azure, Google Cloud та інш. Стисло

описуються особливості методології DevOps, пояснюється як і завдяки чому вона розвивається. Обговорюються зв'язки та взаємодія DevOps, програмно-визначуваних мереж Software Defined Networks та Інтернету речей.

У розділі 24 розглядаються моделі функціональної та інформаційної безпеки. У рамках концепції функціональної та інформаційної безпеки запропоновано таксономію вимог, атрибути та основи аналізу ризиків. Моделі функціональної безпеки в основному кількісні, засновані на імовірнісному аналізі значень показників. Моделі інформаційної безпеки в основному якісні, засновані на аналізі загроз і сценаріїв атак.

В розділі 25 розглядаються вимоги до управління функціональною та інформаційною безпекою, включаючи управління персоналом, управління конфігурацією, вибір і оцінювання інструментальних засобів, управління документацією, а також оцінку безпеки. Докладно описується V-подібний життєвий цикл функціональної та інформаційної безпеки, включаючи трасування вимог. Розглянуто основні методи верифікації, такі як огляд документів, статичний аналіз коду, функціональне і структурне тестування.

Методологія Assurance Case розглядається в 26 розділі, як цілісний підхід до інтеграції вимог і артефактів безпеки. Для цього представлені основи Assurance Case, а також концепція і історія. Для графічного представлення Assurance Case використовуються напівформальні нотації, такі як «Мета, аргумент і підтвердження» (CAE) і «Нотація структурованих цілей» (GSN). Assurance Case для систем інтернету речей ґрунтується на врахуванні вимог до інформаційної безпеки та енергоефективності.

У розділі 27 розглянуто основи технології блокчейн та приклади її використання в середовищі Інтернет речей. Проведено аналіз алгоритмів консенсусу, які використовуються в технології блокчейн, і принципів забезпечення надійності та безпеки Інтернет речей з використанням технології блокчейн. Виділено переваги та існуючі проблеми інтеграції технологій блокчейн в Інтернет речей. Вирішення питання безпеки на різних рівнях застосування IoT є

більш складною проблемою через обмежену продуктивність та високу неоднорідність пристроїв.

У розділі 28 розглядається низка проблем, що пов'язані розробкою архітектур IoT, архітектур пристроїв та інтеграції базових компонент на основі IoT. Розглянуто ефективні підходи до розробки для подолання основних проблем, що виникають в процесі проектування та впровадження ефективного IoT рішення. Обговорюються базові компоненти систем IoT, фази та результати технічної стратегії IoT, а також критерії вибору для розгортання платформ IoT.

У розділі 29 розглянуті моделі IoT пристроїв і технології для обробки і передачі даних. У цьому розділі аналізуються основні принципи побудови інформаційних моделей IoT пристроїв і інструменти для їх створення, зокрема Eclipse Vorto. Також досліджені протоколи мережевих з'єднань для IoT пристроїв. Крім того, важливим компонентом IoT мережі є вибір технологій обробки даних в IoT системах і методів управління і прогнозування. Також розглянуті основні протоколи і стандарти для передачі даних між IoT пристроями. Деяка увага приділяється кібербезпеці в IoT.

У розділі 30 розглянуті інтелектуальні методи та підходи для управління і навчання IoT систем. У цьому розділі аналізуються типи та можливості IoT платформ, багатокритерійний підхід і м'які обчислення для вибору IoT платформи. Також проаналізовано концепцію мультиагентного підходу в IoT, зокрема, типи і характеристики агентів, зв'язок агентів з зовнішнім середовищем і технології передачі даних між агентами. Крім того, важливим компонентом IoT мережі є вибір методів і підходів для навчання IoT систем. Також розглядаються загальні принципи міжмашинного навчання, системи, що самостійно навчаються і нейронні мережі.

У розділі 31 розглянуті моделі інформаційної взаємодії елементів систем IoT. Наведено порядок розробки і швидкого прототипування пристроїв. Показані типові рішення для побудови систем IoT, використання віртуальних пристроїв для розробки

програмного забезпечення. Наведені приклади розробки та прототипування каналу вимірювань на основі малоресурсних мікроконтролерів. Показано прискорення розробки пристрою IoT з використанням сучасних відкритих платформ і бібліотек високорівневих функцій.

УДК 62:004=111

173

Рецензенты: Dr. Mario Fusani, ISTI-CNR, Пиза, Италия

Dr. Olga Kordas, KTH University, Стокгольм, Швеция

Viktor Kordas, KTH University, Стокгольм, Швеция

**173 Интернет вещей для промышленных и гуманитарных приложений. В трех томах. Том 1. Моделирование и разработка /** Под ред. В. С. Харченко. - Министерство Образования и науки Украины, Национальный аэрокосмический университет ХАИ, 2019. - 547с.

ISBN 978-617-7361-82-3

Книга, состоящая из трех томов, содержит теоретические материалы для лекций и тренингов, разработанных в рамках проекта Internet of Things: Emerging Curriculum for Industry and Human Applications /ALIOT, 573818-EPP-1-2016-1-UK-EPPKA2- CBHE-JP, 2016-2019, финансируемого программой ЕС ERASMUS +. Том 2 описывает модели, методы моделирования и разработки для Интернета вещей (IoT). Книга состоит из 4 частей для соответствующих докторантских курсов: моделирование систем на основе IoT (разделы 16-19), программно-определяемые сети и IoT (разделы 20-23), надежность и безопасность IoT (разделы 24-27), разработка и внедрение систем на основе IoT (разделы 28-31).

Книга подготовлена украинскими университетскими командами при поддержке коллег из академических организаций стран ЕС, входящих в консорциум проекта ALIOT.

Книга предназначена для магистрантов и аспирантов, изучающих технологии IoT, программную и компьютерную инженерию, компьютерные науки. Может быть полезна для преподавателей университетов и учебных центров, исследователей и разработчиков систем IoT.

Рис. : 158. Ссылок: 430. Таблиц: 45.

Эта работа защищена авторским правом. Все права зарезервированы авторами, независимо от того, касается ли это всего материала или его части, в частности права на переводы на другие языки, переиздания, повторное использование иллюстраций, декламацию, трансляцию, воспроизведения на микрофильмах или любым другим физическим способом, а также передачу, хранение и электронную адаптацию с помощью компьютерного программного обеспечения в любом виде, либо же аналогичным или иным известным способом, либо же таким, который будет разработан в будущем.

### **Аннотации разделов**

Раздел 16 посвящен описанию общих принципов функционирования платы АРДУИНО и симуляции ее работы. Показаны различия между физической и компьютерной симуляцией. Описываются методы симуляции, которые могут быть применены для плат АРДУИНО. Приведен сравнительный анализ различных программных средств, которые могут быть использованы для симуляции. Подробно описана работа с программным комплексом ПРОТЕУС.

В разделе 17 рассматривается трехуровневое моделирование IoT/IoE систем в их структуре, поведении и процессах синхронизации. Предложено визуальное моделирование, моделирование и проверка архитектуры, функциональности и временных особенностей IoT/IoE систем и их компонентов в статическом и динамическом режимах с использованием UML диаграмм, сетей Петри, временной логики, соответствующих им методик и инструментов. Показаны особенности моделирования на основе эволюционных генетических и мультиагентных технологий.

При исследовании надежности микропроцессорных систем часто применяется математический аппарат Марковских и полумарковских моделей. Функционирование систем интернета вещей при определенных принятых допущениях может быть описано с помощью данных моделей. В разделе 18 описаны особенности создания Марковских и полумарковских математических моделей для описания процесса функционирования систем Интернета вещей. Приведены и описаны допущения при разработке подобных моделей. Построены и исследованы Марковские модели готовности систем интернета вещей.

Раздел 19 посвящен моделированию взаимодействий в IoT системах. Рассмотрена архитектура систем и общие шаблоны для моделирования взаимодействий. Так как методы и модели варьируются от сложности проектируемых систем и класса решаемых задач, в главе рассматривается четыре примера,

использующие различные подходы. В примере с удаленной лабораторией GOLDi демонстрируется применение FSM и Крипке моделей, при проектировании системы голосовой навигации для Смарт-Кампуса использовались IFML модели, для моделирования кибер-физических систем использовались «цифровые-двойники», что было показано на примере лаборатории ISRT.

В разделе 20 рассматриваются основы технологии программно-конфигурируемых сетей – базовые принципы построения и функционирования, основополагающие технологии, архитектурные особенности. Акцент ставится также на фундаментальных отличительных особенностях технологии, исторических предпосылках, которые поспособствовали возникновению последней. Отдельное внимание уделено обзору эволюции спецификации OpenFlow, формирующей базис обеспечения унифицированного механизма взаимодействия между контроллером и коммутаторами.

В разделе 21 рассматриваются вопросы программирования и моделирования программно-конфигурируемых сетей. Аспекты программирования рассмотрены на примере языка программирования Python. Приводятся и поясняются базовые команды конфигурирования топологии сети, в частности команды, предназначенные для решения вопросов автоматизации названных действий. Внимание уделяется среде моделирования Mininet и соответствующей графической оболочке MiniEdit.

В разделе 22 рассматривается ряд исследовательских проблем, связанных с реализацией специфических моделей QoS через SDN путем разработки и внедрения алгоритмов и подходов, обеспечивающих эффективную работу SDN в IoT. Последние тенденции в использовании алгоритмов для технологии SDN были проанализированы с точки зрения их пригодности для создания и обслуживания крупных магистральных сетей SDN / OpenFlow в инфраструктуре IoT. Обсуждаются перспективы прогнозирования производительности SDN с использованием метода объединения данных.

В разделе 23 анализируются инновационные, технологические и бизнес-причины появления и развития

методологии Development and Operations (DevOps). Внимание фокусируется на хорошо известных платформах AWS, MS Azure, Google Cloud и других. Дается краткое введение в особенности методологии DevOps, объясняется, как и благодаря чему она развивается. Обсуждаются связи и взаимодействие DevOps, программно-определяемых сетей Software Defined Networks и Интернета вещей.

В разделе 24 рассмотрены модели функциональной и информационной безопасности для систем интернета вещей. В рамках концепции функциональной и информационной безопасности предложены таксономия требований, атрибуты и основы анализа рисков. Модели функциональной безопасности в основном количественные, основанные на вероятностном анализе значений показателей. Модели информационной безопасности в основном качественные, основанные на анализе угроз и сценариев связанных атак.

В разделе 25 рассматриваются требования к управлению функциональной и информационной безопасностью, включая управление персоналом, управление конфигурацией, выбор и оценивание инструментальных средств, управление документацией, а также оценку безопасности. Подробно описывается V-образный жизненный цикл функциональной и информационной безопасности, включая трассировку требований. Рассмотрены основные методы верификации, такие как обзор документов, статический анализ кода, функциональное и структурное тестирование.

Методология Assurance Case рассматривается в 26 разделе, как целостный подход к интеграции требований и артефактов безопасности. Для этого представлены основы Assurance Case, а также концепция и история. Для графического представления Assurance Case используются полуформальные нотации, такие как «Цель, аргумент и подтверждение» (CAE) и «Нотация структурированных целей» (GSN). Assurance Case для систем интернета вещей основывается на учете требований к информационной безопасности и энергоэффективности.

В разделе 27 рассмотрены основы технологии блокчейн и примеры ее использования в Интернет вещей. Проведен анализ алгоритмов консенсуса, используемых в технологии блокчейн, и принципов обеспечения надежности и безопасности Интернет вещей с использованием технологии блокчейн. Выделены преимущества и существующие проблемы интеграции технологий блокчейн в Интернет вещей. Решение вопроса безопасности на различных уровнях применения IoT является более сложной проблемой из-за ограниченной производительность и высокой неоднородности устройств.

В разделе 28 рассматривается ряд исследовательских проблем, связанных с разработкой IoT-архитектур, архитектур устройств и системной интеграции на основе IoT. Рассмотрены эффективные подходы к разработке для преодоления существенных проблем при разработке и внедрении эффективного решения IoT. Обсуждаются базовые компоненты систем IoT, этапы и результаты технической стратегии IoT, а также критерии выбора для развертывания платформ IoT.

В разделе 29 рассмотрены модели IoT устройств и технологии для обработки и передачи данных. В этом разделе анализируются основные принципы построения информационных моделей IoT устройств и инструменты для их создания, в частности Eclipse Vorto. Также исследованы протоколы сетевых соединений для IoT устройств. Кроме того, важным компонентом IoT сети является выбор технологий обработки данных в IoT системах и методов управления и прогнозирования. Также рассмотрены основные протоколы и стандарты для передачи данных между IoT устройствами. Некоторое внимание уделяется кибербезопасности в IoT.

В разделе 30 рассмотрены интеллектуальные методы и подходы для управления и обучения IoT систем. В этой главе анализируются типы и возможности IoT платформ, многокритериальный подход и мягкие вычисления для выбора IoT платформы. Также проанализирована концепция мультиагентного подхода в IoT, в частности, типы и характеристики агентов, связь агентов с внешней средой и технологии передачи данных между

агентами. Кроме того, важным компонентом IoT сети является выбор методов и подходов для обучения IoT систем. Также рассматриваются общие принципы межмашинного обучения, самообучающиеся системы и нейронные сети.

В разделе 31 рассмотрены модели информационного взаимодействия элементов систем IoT. Приведен порядок разработки и быстрого прототипирования устройств. Показаны типовые решения для построения систем IoT, использование виртуальных устройств для разработки программного обеспечения. Приведены примеры разработки и прототипирования канала измерений на основе малоресурсных микроконтроллеров. Показано ускорение разработки устройства IoT с использованием современных открытых платформ и библиотек высокоуровневых функций.

Олександр Валентинович Дрозд, Олег Олександрович Ілляшенко,  
Вячеслав Сергійович Харченко, Марина Олександрівна Колісник,  
Галина Володимирівна Кондратенко, Юрій Пантелійович Кондратенко,  
Олена Юрївна Масвська, Дмитро Андрійович Маєвський, Олександр  
Миколайович Мартинюк, Денис Сергійович Мазур, Максим  
Володимирович Нестеров, Анатолій Павлович Плахтєєв, Вадим  
Вікторович Шкарупило, Євген Вікторович Сіденко,  
Інна Сергіївна Скарга-Бандурова, Володимир Володимирович Скляр,  
Галина Володимирівна Табунщик, Микита Олександрович Таранов,  
Артем Юрійович Великжанін, Дмитро Дмитрович Узун,  
Юлія Олександрівна Узун, Наталія Георгіївна Яцків,  
Василь Васильович Яцків, Георгій Андрійович Землянко

**Інтернет речей для індустріальних і гуманітарних  
застосувань.**

**Том 2. Моделювання і розроблення**  
(англійською мовою)

Редактор *Харченко В.С.*  
Комп'ютерна верстка *Ілляшенко О.О.*

Зв. план, 2019  
Підписаний до друку 22.08.2019  
Формат 60x84 1/16. Папір офс. №2. Офс. друк.  
Умов. друк. арк. 33,95. Обл.-вид. л. 34,19. Наклад 150 прим.  
Замовлення 220819\_2

Національний аерокосмічний університет ім. М. Є. Жуковського  
"Харківський авіаційний інститут"  
61070, Харків-70, вул. Чкалова, 17  
<http://www.khai.edu>

Випускаючий редактор: ФОП Голембовська О.О.  
03049, Київ, Повітрофлотський пр-кт, б. 3, к. 32.  
Свідоцтво про внесення суб'єкта видавничої справи до державного реєстру видавців,  
виготовлювачів і розповсюджувачів видавничої продукції  
серія ДК No 5120 від 08.06.2016 р.

Видавець: ТОВ «Видавництво «Юстон»  
01034, м. Київ, вул. О. Гончара, 36-а, тел.: +38 044 360 22 66  
[www.yuston.com.ua](http://www.yuston.com.ua)

Свідоцтво про внесення суб'єкта видавничої справи до державного реєстру видавців,  
виготовлювачів і розповсюджувачів видавничої продукції  
серія ДК No 497 від 09.09.2015 р.