University of Pardubice

Faculty of Electrical Engineering and Informatics

DOCTORAL THESIS

MSc Filip Holík

Using SDN to Enhance IoT Security

Supervisor:	prof. Ing. Simeon Karamazov, Dr.
Co-supervisor:	Mgr. Josef Horálek, Ph.D.
Study Programme:	P2612 - Electrical Engineering and Informatics
Field of Study:	2612V070 Information, Communication and Control Technologies

Pardubice, 2018

Declaration of author

Pardubice, 28th August 2018 MSc Filip Holík

Acknowledgement

Most importantly, I would like to express my appreciation to related staff at the *Lancaster University*. Only because of their willingness and help during my several visits, I was able to conduct experiments, used in this work, on state-of-the-art equipment.

I would also like to thank to my colleagues for their support. Namely, to Paul Hooper, for his never ending patience and optimism during proofreading and while correcting my repetitive mistakes; to Sona Neradova for her continuous methodical recommendations; to Ilona Kucerova and other staff for perfect administrative support during my entire studies; and finally, to my supervisors for their consultations.

Last but not least, I would like to thank to my family and friends. Only their motivation and support allowed me to work on the thesis even during demanding periods.

Abstract

One of the most innovative technologies in computer networks - Software Defined Networking - has brought programmability into this, traditionally static and inflexible field. The programmability allows efficient implementation of new functionalities into modern networks, which can have dynamically evolving requirements. The typical example of such networks is an IoT network and its security requirements. An IoT network can be composed from smart buildings, smart homes, smart grids, smart mobility, and it can even interconnect all these fields into a large scale smart city network. Correct functionality of such a network is then critical as many areas of real life depend on it. Moreover, this network needs to dynamically evolve in future. Software Defined Networking is an ideal technology to realize these requirements and to efficiently enhance the security of these networks.

Use of Software Defined Networking in critical networks such as smart cities, depends on safe and functional control applications. This thesis proposes a blueprint for an effective development and implementation of such applications. The blueprint describes a 6-step development process, which gives a general guide of how to progress with the application development with security in mind. The blueprint then focuses mostly on the second phase - analysis - as it summarizes and presents a list of application requirements. Finally, the blueprint analyses the most common security threats and proposes approaches to mitigate these threats by Software Defined Networks. In the last part of the thesis, the blueprint is verified on a use case application of a smart city protection system, specifically developed for this purpose.

Keywords

Application blueprint, application development, attacks, Internet of Things, IoT security, OpenFlow, protection system, security design, security threats, smart cities, Software Defined Networks.

Název práce

Využití SDN pro zlepšení zabezpečení IoT

Anotace

Jedna z nejvíce inovativních technologií počítačových sítí - softwarově definované sítě - přinesla programovatelnost do této jinak tradičně statické a neflexibilní oblasti. Tato programovatelnost umožňuje efektivní implementaci nových funkcionalit do moderních počítačových sítí, které mohou mít dynamicky se měnící požadavky. Typickým příkladem jsou sítě Internetu věcí a jejich požadavky na bezpečnost. Sítě Internetu věcí mohou obsahovat chytré budovy, chytré domácnosti, sítě pro chytrou mobilitu a sítě typu smart grid. Všechny tyto oblasti mohou navíc být propojeny do jedné velké sítě chytrého města. Na té pak závisí mnoho oblastí reálného života, a proto je její správná funkcionalita kritická. S tím souvisí i možnost dynamického budoucího rozvoje této sítě. Softwarově definované sítě jsou ideální technologií pro realizaci těchto požadavků včetně efektivního zlepšení bezpečnosti.

Využití softwarově definovaných sítí v kritických sítích, jako jsou chytrá města, závisí na korektní funkcionalitě kontrolních aplikací. Tato práce předkládá plán postupu efektivního vývoje a implementace právě takových aplikací. Tento plán v šesti krocích popisuje doporučený postup vývoje aplikací s důrazem na bezpečnost. Plán se soustředí zejména na druhý krok - analýzu, ve které předkládá souhrn požadavků na aplikace. Analyzuje také typické bezpečnostní hrozby včetně způsobů jejich eliminace s použitím softwarově definovaných sítí. V poslední části práce je tento plán otestován na ukázkové aplikaci systému pro zabezpečení chytrých měst, která byla vyvinuta specificky pro tento účel.

Klíčová slova

Bezpečnost IoT, bezpečnostní hrozby, bezpečnostní systém, chytrá města, Internet věcí, návrh bezpečnosti, OpenFlow, plán vývoje aplikací, softwarově definované sítě, útoky, vývoj aplikací.

Contents

List of Figures

List of Tables

List of Abbreviations

6LoWPAN	IPv6 over Low-Power Wireless Personal Area Networks
ACL	Access Control List
AES	Advanced Encryption Standard
AP	Access Point
AMI	Advanced Metering Infrastructure
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
ARP	Address Resolution Protocol
ARPANET	Advanced Research Projects Agency Network
ASIC	Application Specific Integrated Circuit
BGP	Border Gateway Protocol
BSD	Berkeley Software Distribution
CCTV	Closed-Circuit Television
CLI	Command-Line Interface
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSD	Circuit Switched Data
CSMA/CA/D	Carrier Sense Multiple Access/Collision Avoidance/Detection
DER	Distributed Energy Resources
DDoS	Distributed Denial of Service
DDS	Data Distribution Service
DSL	Digital Subscriber Line
DSRC	Dedicated Short-Range Communications
DTLS	Datagram Transport Layer Security
EMC	Energy Management Center
ESS	Energy Storage Systems
EV	Electric Vehicle
FC	Fog Computing
FIB	Forwarding Information Base
FW	Firewall
ForCES	Forwarding and Control Element Separation
GBP	Group Based Policy
GOOSE	Generic Object Oriented Substation Event
GPRS	General Packet Radio Service
GPS	Global Positioning System
GPU	Graphics Processing Unit

GRE	Generic Routing Encapsulation
GSM	Global System for Mobile Communication
GSSE	Generic Substation Status Event
GW	Gateway
HAN	Home Area Network
HTTP(S)	Hyper Text Transfer Protocol (Secure)
HVAC	Heating, Ventilation, and Air Conditioning
HW	Hardware
I2RS	Interface to the Routing System
IAM	Identity and Access Management
IANA	Internet Assigned Numbers Authority
IBN	Intent-Based Networking
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IED	Intelligent Electronic Devices
IP	Internet Protocol
IPS	Intrusion Prevention System
IPsec	Internet Protocol Security
ISP	Internet Service Provider
ISR	Integrated Services Router
JSON	JavaScript Object Notation
L2/L3	Layer 2/3 of the ISO/OSI network model
LLDP	Link Layer Discovery Protocol
LoRa	Long Range
LPWAN	Low Power Wide Area Network
LTE-A	Long-Term Evolution Advanced
M2M	Machine to Machine
MANO	Management and Orchestration
MEC	Mobile Edge Computing
MitM	Man-in-the-Middle
MMS	Manufacturing Message Specification
MPLS	Multiprotocol Label Switching
MQTT	Message Queue Telemetry Transport
MTU	Maximum Transmission Unit
MU	Merging Unit
NAS	Network Attached Storage
NFC	Near Field Communication
NFV(I)	Network Functions Virtualization (Infrastructure)
NIST	U.S. National Institute for Standards and Technology

OF	OpenFlow
OBU	Onboard Unit
ONF	Open Networking Foundation
ONOS	Open Network Operating System
OpFlex	An Open Policy Protocol
OS	Operating System
OTA	Over The Air
OVS	Open vSwitch
OXM	OpenFlow eXtensible Match
OXS	OpenFlow eXtensible Statistics
PKI	Public Key Infrastructure
PLC	Power Line Communication
PMU	Phasor Measurement Unit
PPS	Packets Per Second
PTP	Precision Time Protocol
QAM	Quadrature Amplitude Modulation
QoS	Quality of Service
QR (Code)	Quick Response (Code)
RAM	Random Access Memory
RC4	Rivest Cipher 4
REST	Representational State Transfer
RF	Radio Frequency
RFID	Radio-Frequency Identification
RPC	Remote Procedure Call
RSTP	Rapid Spanning Tree Protocol
RSUC	Road Side Unit Controller
RTT	Round Trip Time
SB	Smart Building
SBC	Single Board Computer
SC	Smart City
SC-SDNA	Smart City - Software Defined Networking Application
SCADA	Supervisory Control And Data Acquisition
SCPS	Smart City Protection System
SDE	Software-Defined Environment
SDN	Software Defined Networking, Software Defined Networks
SHA	Secure Hash Algorithm
SIEM	Security Information and Event Management
SL	Smart Lighting
SMV	Sampled Measured Value

SNMP	Simple Network Management Protocol
SNTP	Simple Network Time Protocol
SSH	Secure Shell
SSL	Secure Socket Layer
STP	Spanning Tree Protocol
SG	Smart Grid
SH	Smart Home
SM	Smart Mobility
SV	Sampling Value
SW	Software
TCAM	Ternary Content-Addressable Memory
TCP	Transmission Control Protocol
TI	Tactile Internet
TLS	Transport Layer Security
TLV	Type-Length-Value
TN	Traditional Networking
TPM	Trusted Platform Module
TRILL	Transparent Interconnection of Lots of Links
TV	Television
UDP	User Datagram Protocol
UI	User Interface
UML	Unified Modeling Language
UMTS	Universal Mobile Communication System
UPnP	Universal Plug and Play
U.S.	United States
V2I/V	Vehicle to Infrastructure/Vehicle
VIM	Virtualized Infrastructure Manager
VLAN	Virtual Local Area Network
VM	Virtual Machine
VNF	Virtualized Network Functions
VoIP	Voice over Internet Protocol
VPN	Virtual Private Network
WAN	Wide Area Network
WAVE	Wireless Access in Vehicular Environments
WPA2	Wi-Fi Protected Access 2
WPAN	Wireless Personal Area Networks
WSN	Wireless Sensor Network
WiMAX	Worldwide Interoperability for Microwave Access
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

1. Introduction

The Internet has become one of the most important aspects of everyday life. In 2017, more than 50% of households worldwide had access to the Internet, while 71% of youth population (between 15 - 24 years) were using it [1]. Internet fulfils all the life roles, from work, to social, and even entertainment. The most recent innovations in this field are cloud streaming services, cryptocurrencies, and IoT (Internet of Things).

All these technologies push requirements on data networks further and further. Video streaming services transferring video in ultra high resolutions, or interconnection of billions of smart devices require significant data bandwidth. At the same time, they also require low latency, all-time reliability, and maximum security. These features can be difficult to achieve with traditional networking technologies and new approaches are therefore utilized. One of them is Software Defined Networking (SDN).

Software Defined Networking

SDN is a modern paradigm of computer networks. SDN slices the monolithic architecture of devices from the traditional IP networking into two layers: *forwarding* and *control*. The forwarding layer is left on a networking device, where it uses existing data structures for packet handling. The control layer, on the other hand, is moved to a separate and centralized device called an *SDN controller*. The controller manages forwarding logic within the whole network. The layer separation and control centralization allow dynamic programmability of any network functionality. This would be otherwise impossible in traditional networking devices, which have fixed features based on vendor support.

SDN is a technology experiencing a dynamic growth and it is currently at the centre of attention in the networking research community. SDN is explored from many points of view and SDN deployments in various fields are considered. SDN found its applicability particularly in data centre networks. Leading companies like Google, Amazon, or Microsoft have been successfully using SDN in their networks for many years. Use of SDN in this environment helps to reduce infrastructure costs, and allows full utilization of the existing topology and usage of advanced features (like intelligent load-balancing, or fast failover mechanisms). Nowadays, SDN is being applied in modern concepts including IoT networks like smart cities. It is also expected, that SDN will become the basic building block for future 5G networks [2].

Internet of Things

IoT is a concept, which merges the physical and digital worlds by interconnecting a vast amount of physical devices via the Internet. Typical IoT devices collect data and can be remotely controlled. Interconnection of these devices can then perform an intelligent behaviour. IoT nowadays covers various domains, from local (smart homes and smart buildings) via industrial (smart grids) to large scale (smart cities). All these domains can have various requirements, but the common theme is security.

Security of IoT Networks

Security of IoT networks presents a significant challenge, especially when rapid advancements in this field and different requirements of every IoT domain are considered. In local networks, protecting sensitive user data and ensuring privacy is the priority. In industrial networks, security is important to protect cyber physical systems, which control potentially dangerous equipment. These networks belong to critical infrastructure and they therefore also require maximum reliability. Smart cities then combine all the mentioned requirements together and add additional challenges - for example their long physical range. This significantly complicates the network protection as a large number of people can easily access the network, especially if a wireless communication is used.

The recent advancements in information technology field made attacks on data networks easier and more dangerous. They can now be launched remotely, on a massive scale, or can use infected zombie nodes. Most importantly, launch of an attack often does not require highly specialized knowledge, or any specific equipment and can therefore be done by practically anyone. Moreover, many of the innovative tools for new malware development are openly available and can be widely used. An example can be *Kali Linux*, which is a Linux OS with many pre-installed tools for penetration testing and (ethical) hacking. This package can be downloaded for free and can be used by anyone as detailed usage guides are widely available. These trends significantly complicate the security. Protection systems must consider all these mentioned directions to provide an appropriate protection.

Connecting Dots

SDN can significantly improve the world of IoT networks. Utilization of generic forwarding devices and a centralized controller ensure the flexibility of the network. The SDN controller also brings advantage in its centralization. Instead of having dedicated network boxes for every functionality (router, firewall, gateway, traffic inspection, VPN, etc.), all these features can be integrated within the controller in a form of software applications. Any new functionality (general, or security related) can be therefore implemented only by modifying the appropriate application, or by creating a new one, which can run in parallel with existing applications. There is no need to perform expensive and time consuming changes to the physical infrastructure, or to even buy and implement completely new networking devices. This concept can ensure that the SDN network will stay future-proof against evolving security threats and new network requirements. One thing is clear - the dynamically evolving nature of IoT networks is an ideal soil for unexpected growth.

2. Theoretical Background

This section describes basic theory of concepts used in this work. The two main technologies are presented: SDN and IoT. Their theoretical knowledge is required to understand more advanced contributions of this work.

2.1 Software Defined Networking

SDN is the main technology used in this work. This section explains SDN origin, current state, technical details, and related technologies.

2.1.1 Introduction

Motivation - Traditional IP Networking

Traditional IP networks widely used today are still based on a project ARPANET (Advanced Research Projects Agency Network) established in 1969. ARPANET started using packet-switching [3], instead of circuit switching, which was at that time commonly used mainly for communication in telephone networks. The first ARPANET installation connected only four nodes (in 1969), but in the following years quickly grew in size. Despite the growth, the ARPANET was still just a single network. The next evolution of computer networks came soon, specifically with the creation of the Ethernet in 1973. This was followed by a satellite network based on ARPANET - ALOHAnet (Additive Links On-line Hawaii Area) in mid 1970', and most importantly with the creation of the Internet [4].

The Internet started with interconnecting separate networks and therefore created a network of networks. The main work and the term *internetting* was described by V. Cerf and R. Kahn in 1974 [5]. This paper also introduced basic Internet principles, which are still used even today: autonomous, decentralized network, with the best effort service, and routers not maintaining states of the connection (stateless routers).

Since the 1970s, the Internet's complexity increased explosively. In order to support this growth, many protocols and features were implemented, but the basic TCP/IP model remained the same. This however means, that today's IP networks suffer from the following flaws:

- **Complexity of configuration** – any changes have to be made on all affected devices (note: except configuration via protocols like SNMP). If the changes are not made properly, parts of the network can become unavailable while configuration upgrade is in progress.
- **Complicated updates** – updates of the functionality and features depend on the vendors' decisions. To support sales of new products, older devices are often left without support of newer or more advanced protocols. Replacing these products with new hardware is costly and require longer time frame to implement.
- **Decentralized forwarding logic** – every device is making its own forwarding and routing decisions (even while the decisions are often based on communication with other networking devices).
- **Error prone** – configuration of large networks is often difficult and error prone. Troubleshooting (a process of finding and correcting errors) is often necessary

in complex networks. Unfortunately, it is a time consuming task with no time boundary or estimation of completion.

- **Static configuration** – there is no mechanism to automatically reconfigure a device setting on specific events.
- **Vendor dependence** – networking devices from different vendors are often not inter-operable due to various implementations of network protocols or supported features.

While these flaws might not have been an issue in the past, they became more apparent in modern networks utilizing advanced features like virtualization, fast failover, load-balancing, and others. SDN is a modern paradigm mitigating flaws of the traditional networking, while effectively supporting new features and protocols.

Concept of SDN

The most traditional definition of SDN is based on two premises: *separation of control plane and forwarding plane* and *logical centralization of the network control.* This concept is therefore slicing the traditional architecture of networking devices and defines a new networking abstraction via a vertical separation as depicted in Figure 2.1. Separation of control and data planes allows the control logic to be put on a dedicated device - *controller*, which can be easily programmed using common programming languages like C++, Java, or Python. This allows progressive change of the network behaviour and ensures future network evolutions [6].

SDN in its more recent concept is viewed as a "*framework with many solutions to a set of problems*" [7]. This definition indicates broad range of areas, where SDN can be deployed.

Regardless of the SDN definition, the most important features of SDN are:

- **Centralized control** – the whole network can be controlled from a single place, making network management simple and effective.
- **Innovation and evolution** - new features can be easily added into the network.
- **Management** - monitor and manage resources of networking devices and connectivity within the whole network.
- **Programmability** – configuration changes can be made quickly and dynamically.
- **Scalability** - ability to scale adequately with the growing network.
- **Vendor neutrality** – the network is based on open standards and devices from different vendors can be used together.
- **Virtualization** - allows using virtualized networking devices without need to specify their organization or location.

On the other hand, SDN is a relatively new technology and there are legitimate doubts, concerning its maturity. How well can a programmable approach cope with a well-known and proven concept of traditional networking, which has existed for almost 50 years? Can a generic software approach replace highly specialized hardware devices? How this approach affects performance? And what about security? Moreover, how network software should be written is, in itself, an extensive topic.

It is clear, that while offering many benefits, the SDN approach has also many pitfalls, including:

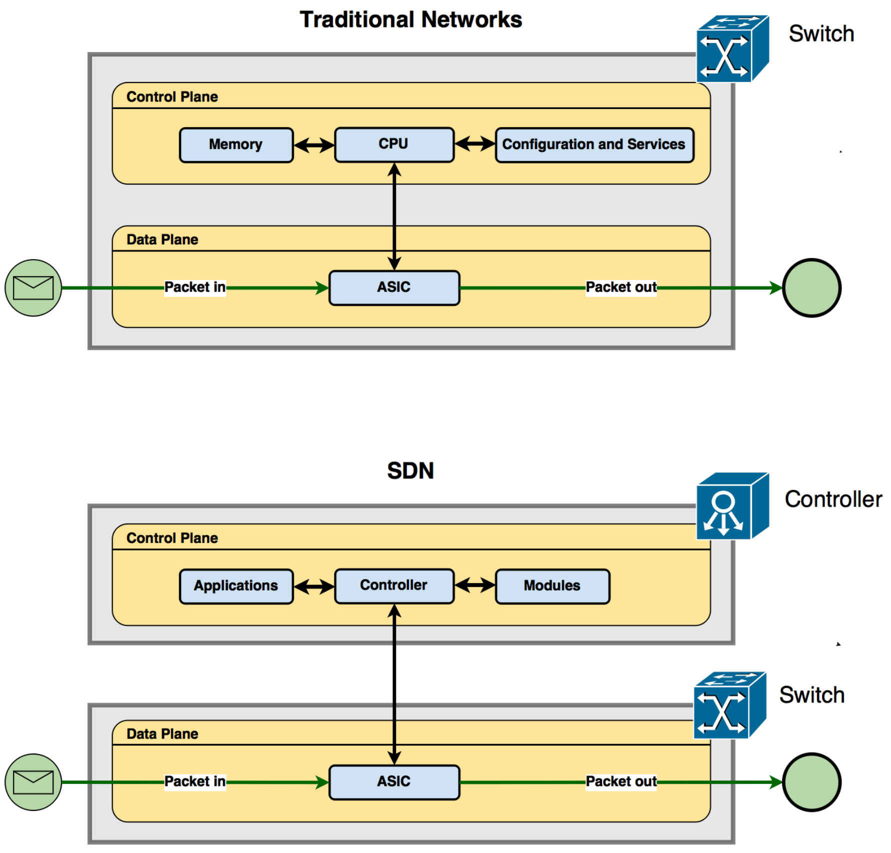


Figure 2.1: Traditional Networks and SDN

- **Centralized approach** - can manage the entire network effectively, but represents a single point of failure. If the controller is attacked, brought down, or just needs a software update, the entire network can become unavailable.
- **Generic boxes** - bring vendor independence and allow to combine hardware arbitrarily. Unfortunately, at the present moment, the cost of these boxes exceeds the cost for similarly equipped closed boxes of the traditional networking.
- **Programmability** - allows to write any code with almost any possible functionality. But a badly written application can bring the SDN controller down, or allow a possible attack.
- **Scalability** - can be more dynamic as the entire network configuration is present only in the controller. With the network expansion, the controller can however become a bottleneck, if its performance does not cope with the network requirements any longer.

2.1.2 SDN History

Path to SDN

SDN emerged from different technologies and concepts already used in networking during the 1990s. Most notable are two technologies: *active networking* and *control and data plane separation.*

Active networking emerged in the early 1990s and its purpose was to allow programmability within a network. The motivation was similar to SDN: faster deployment of new services, added network value, dynamic reactions to specific network conditions or applications, and a space for experiments. Active networking tried to achieve these goals with two different programming models. In the first one - *the capsule model* - the programmed code was carried inside data packets. In the second one - *the programmable model* - the code was pushed to the networking devices. Regardless of the promising goals, the active networking stayed mostly as a collection of test projects funded by the DARPA, and the deployment was never widespread. This was caused mainly by security issues, where especially in the capsule model, the code could be easily hacked during a transmission. Other reasons for its lack of success were poor performance and the deficiency of necessary applications. [8]

Control and data plane separation is an idea from early 2000s, when network traffic grew rapidly and moreover, a need for more effective network management functions arose. The control and data plane separation concept was aimed at ISPs and network administrators. This is the opposite of active networking, which was supposed to be used mostly by end users. Despite the promising network management goals (like dynamic path selection based on current traffic load, more flow control, minimized disruptions, and increased security) the technology was used only in the form of industry prototypes. Nevertheless, this concept introduced two new innovations later used in SDN: an open interface between control and data plane (via ForCES [9] or via usage of existing protocols like BGP[1]), and logically centralized control. [8]

SDN Emerges

Creation of SDN is connected with the invention of the OpenFlow protocol. This protocol was created in the academic environment of the Stanford University and initially became a synonym for SDN. OpenFlow managed to find a compromise between fully programmable networks and easy deployment into the existing networking architecture. By using existing switching hardware, OpenFlow could be adopted into legacy devices via a firmware update, instead of requiring a modification of the existing hardware. Unlike the previous work in this area, OpenFlow defined matching behaviour, based on different packet header fields (and not based only on the destination IP address). OpenFlow also introduced a vision of a network OS abstracted from networking devices and using three layer separation. These features ensured that OpenFlow quickly became deployed not only in campus environments, but also in data centres and other large scale networks. [8]

With the growing popularity of SDN and increasing support from industry, SDN became separated from OpenFlow. SDN grew in complexity and many different technologies, communication protocols, and controller platforms began to be integrated into it. OpenFlow now remains the most used SDN standard for communication between a controller and SDN devices, but there are other protocols in use as well. Currently, the development of OpenFlow is driven by a non-profit organization called ONF [10]. The ONF was established in 2011 with support of companies like Google, Facebook, Yahoo, Microsoft, Verizon, and many others. Its continuing goal is to promote development of SDN through open standards.

2.1.3 Existing SDN Deployments

Nowadays, there are several real world SDNs scenarios used worldwide. The most important ones are described.

[1]This solution allowed immediate deployment and backward compatibility, but limited functionality.

Azure - Microsoft

SDN implemented in Microsoft Azure is responsible for effective management of the world's most popular services like Office365, Skype, OneDrive, Xbox, and Bing search. Unlike in traditional SDN deployments, Azure is using multiple controllers for specific applications. These controllers work in clusters and are controlled by regional controllers in a hierarchical fashion. Communication between them is provided via a *northbound* API. On the forwarding plane, Azure is using *flow tables* similar to the ones defined in OpenFlow. In order to optimize latency and performance of switches dealing with large amount of traffic, an additional layer was put in place between forwarding devices and controllers. Microsoft added a virtual network agent on every hypervisor host. This agent is then contacting the controller in the case that no matching flow is found in the switch flow table. This solution moves the computational overhead from switches into more powerful host devices and allows more efficient use of OS-level API for communication.

One of the most important features of Azure's SDN is automation. If any customer defines its own network policies and addressing schemes via the web application, these changes are automatically pushed through controllers into the networking devices. This will dynamically adjust the topology. [11]

B4 WAN Network - Google

Google's B4 WAN is one of the largest networks of its type on the Internet; connecting 12 data centres scattered around the world. This network is used for data mirroring between data centres, index pushes, end user data replication, and internal applications[2].

B4 network has the following features:

- **Centralized traffic engineering** - allows to simplify the network management.
- **Maximum links utilization** - typical WAN links are designed to work utilized at around 30 - 40% in average, so in case of a link failure, traffic forwarding is not disrupted. By reliably utilizing links to almost 100 %, Google managed to reduce costs of these links by up to two thirds. Such a high utilization, while ensuring reliability, is made possible by using multipath forwarding based on available link capacity. This is achievable due to specific requirements of Google's WAN applications, which can sustain temporal downtimes or link capacity reductions. On the other hand, if high capacity is available, these applications can use it.
- **Separated hardware and software** - simplifies planned and unplanned network changes and allows independent updates of servers and switching hardware.

B4 is probably the world's largest SDN WAN in active use. Since its creation, it proved its usefulness and reliability with only one major outage, which was fully comparable to the outages found in similar WANs based on the traditional networking [12].

Bristol is Open

SDN deployment in a Smart City (described in section 2.4.1) architecture is being tested in Bristol's open programmable city project. The project includes a fibre core network,

[2]Google's end users are connected to a different network, which is built for services like Google search or Google email. B4 serves as a support internal network for providing connectivity between data centres.

meshed wireless network with nodes placed on lamp posts, and an experimental wireless connectivity for users. The network uses various wireless technologies (LTE, 802.11ac, 4G, and 60 GHz connections) and SDN-enabled switches [13]. Various data is gathered from users' smartphones.

The architecture combines SDN with NFV (Network Functions Virtualization - described in section 2.2.3) and allows running several applications simultaneously, while meeting tight delay requirements [14]. SDN is based on OpenFlow 1.3 and is controlled by a custom network operating system called *Smart City OS - NetOS*[3].

EC2 - Amazon

According to [15], EC2 (Elastic Compute Cloud 2 - part of Amazon Web Services[4]) uses SDN for automation, cloud formation, security zones, elastic load-balancing, and network isolation. Automation is crucial for cloud service as it allows dynamic and safe modification of the network topology, addition or removal of virtual servers, etc. Network isolation allows usage of overlapping IP addresses for different consumers. Although the Amazon is not publishing details about its SDN solution, it appears, that EC2 has used SDN since 2013 (but probably even longer).

Other SDN Deployments

Except for the mentioned networks, other large companies like Facebook are also using SDN deployments [16]. Probably the most interested companies in SDN implementation nowadays are large ISPs like AT&T and Verizon. This is due to the SDN potential of significant cost reductions, gained capacity, easier scalability, and reduced provisioning times [17].

2.1.4 SDN Architecture

SDN architecture, depicted in Figure 2.2, is composed from three layers and two interfaces between these layers. The layers are:

1. **Data plane** – contains network devices with the support of SDN. These include switches and routers which can be present in a form of physical hardware or implemented in software.

2. **Control plane** – is represented by SDN controller(s), which manages devices present on the data plane layer.

3. **Management plane** – a layer with applications, which contain network logic and are written by developers. Instructions are translated by the controller and forwarded into the appropriate networking devices.

In order to enable communication between these three layers, two interfaces are used:

1. **Southbound interface** – for communication between data and control planes. The most common representative is OpenFlow.

2. **Northbound interface** – for communication between control and management planes. Can be implemented using various protocols like REST (Representational State Transfer), RESTful API, RESTCONF, etc.

[3]More information: `http://www.zeetta.com/netos/`
[4]More information: `https://aws.amazon.com/what-is-aws/`

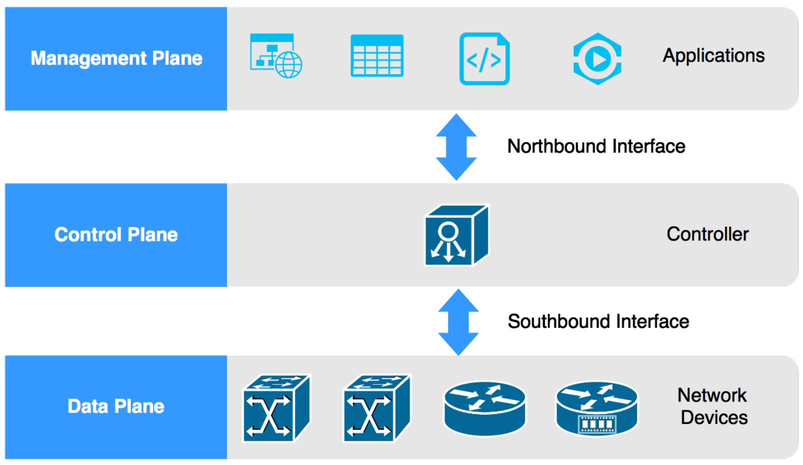


Figure 2.2: SDN Architecture

Data Plane

The data plane contains network devices, which can forward data according to commands from an SDN controller. These devices can be switches, routers, or any other, provided that they support at least one of the *southbound* interfaces and have the ability to store sets of rules received by this interface. These devices can be called SDN-enabled devices, forwarding devices, or data plane devices. For the purpose of this work, they will be from now on, called *forwarding devices*. A forwarding device can be hardware or software based, and it does not need to have any software for making its own autonomous forwarding decisions[5]. If a device supports OpenFlow, it must have at least one data structure called *flow table* (from OpenFlow version 1.1, multiple flow tables can be implemented). This data structure can be implemented either in hardware or in software. The flow table can store different set of rules learned from a controller. Each rule contains the following parts:

- **Match** – in a datagram, it specifies selected data fields, which need to be identical in order to be matched. The OpenFlow specification defines a large number of data fields, but the support of most of them is optional (they are described in section 2.1.5). Typically, support of an optional field is more limited in flow tables implemented in hardware (on HW-based SDN networking devices) due to their restrictions. Also not all of the fields can be typically matched in hardware flow tables at the same time.

- **Action** – defines what will happen to the matched datagram. Actions can be: forward to (specific port(s) or controller), drop it, or send it. In the last case, the datagram can be sent to: special table (group or metering tables used for special processing), other flow table (for additional processing), or to the *normal layer* (if the traditional IP networking is supported by the device).

- **Counters and statistics** – store information about matched datagrams. This includes duration of the rule, timeouts of the rule, number and byte counts of matched datagrams, and many other statistics.

[5]If the device contains a SW for making its own forwarding decisions, it can operate in two modes - in the traditional IP networking mode and in the SDN mode. Some devices also support combination of these modes (also referred to as *hybrid SDN*).

Table 2.1: Examples of SDN-enabled Networking Devices

Type	**Vendor**	**Devices**	**OF ver.**
HW, S	Cisco	4500	1.0, 1.3
HW, S	Cisco	Nexus: 5000, 5500, 6000	1.0, 1.3
HW, ISR	Cisco	4451-AX, 1000-AX	onePK*
HW, S	HP	2920, 3500, 3800(10), 5130, 5400, 5500	1.0, 1.3
HW, S	HP	5700, 5900, 5920, 8200, 10500, 12500	1.0, 1.3
HW, FS	HP	5700, 5930, 11900, 12900	1.0, 1.3
HW, S	Pica8	P-3297,P-3922, P-3930, P-5101, P-5401	1.4
SW, S	Ericsson	OpenFlow Software Switch	1.3
SW, S	Open	CPqD	1.3
SW, S	Open	Indigo Virtual Switch	1.0
SW, S	Open	Lagopus	1.3
SW, S	Open	Open vSwitch	1.0 - 1.5

Note: S = switch, FS = FlexFabric switch, ISR = Integrated Services Router
* onePK is OpenFlow-like proprietary protocol developed by Cisco
Data sources: official websites of mentioned vendors / GitHub for *open* vendors

When a packet arrives on a forwarding device, rules in the flow table are searched (higher priority rules are executed first) and the appropriate rule is either found, or a table miss is generated and the packet is discarded. Currently, there are many forwarding devices available, either HW-based or SW-based (which can be commercial or open-source). Some of these devices are stated in Table 2.1.

When choosing a forwarding device, the most important is the supported version of OpenFlow and the extent of implemented optional OpenFlow features. For example, the paper [18] performed an extensive testing of implemented OpenFlow features in two software switches with OpenFlow 1.3 support - OVS and Lagopus. The results showed, that while both switches implemented all 114 required features, OVS implemented only 556 optional ones (and omitted 321), while Lagopus implemented 862 optional features (and omitted only 15).

On top of that, HW-based devices have a limited size of hardware flow tables[6]. Hardware flow tables mostly use existing hardware tables of a switch, such as TCAMs (Ternary Content-Addressable Memory). Depending on the size of TCAMs, they can support from 8K flow rules (most common devices) to up to 1000K (state-of-the-art devices) flow rules. Typically in HW-based devices, flow tables are implemented as one hardware table (or maximally a few of them) and additional flow tables are implemented in software. This solution offers flexibility and larger space for flow rules, but the speed of rules matched in the software tables is significantly lower than in hardware tables.

Control Plane

The control plane is represented by an SDN controller, which is a software running on a hardware device (typically a server). The controller is connected to selected forwarding devices and it manages their flow tables. On top of that, the controller provides an abstraction layer with APIs to the management plane (for additional applications written by SDN developers). The controller itself can provide various features like topology information and implementation of network policies via different modules.

[6]The size of SW flow tables is also limited, but most often not so noticeably.

These modules can also provide basic networking functionalities known from traditional IP networks. This includes: hub, L2 switch, router, firewall, spanning-tree protocol (STP), IGMP (Internet Group Management Protocol) snooper, etc. Usage of these modules simplifies the deployment of SDN, so developers do not have to program all network functionalities by themselves.

Controllers' complexity greatly varies depending on their specialization and programming language. Controllers can be distributed as an open-source or commercial software. The list of the most common controllers and their basic features is shown in Table 2.2.

The standard controller deployment represents a single point of failure, which potentially compromises the network reliability. If the controller stops working, the whole network functionality can be lost[7]. For this reason, a network can contain multiple controllers. There are two controller deployment architectures:

1. **Centralized** – a single controller, representing a single point of failure.
2. **Distributed** – multiple controllers for redundancy, resiliency, and increased performance.

The **centralized architecture** is the most common. Controllers using this type are often programmed as highly concurrent systems using multiple threads. Thus, their performance is sufficient even for data centres and similarly large networks, if a multicore system is used. Some of these controllers (like Rosemary [19]) also offer a certain degree of application isolation to increase their resiliency [6].

The **distributed architecture** can scale to potentially serve network of any size. Multiple controllers can be deployed on a virtualized cluster of nodes, or on physically separated devices to ensure redundancy. While the distributed architecture offers significant benefits, there are also costs connected with this deployment type. Consistency of settings amongst all controllers is an issue. Changes are not made instantly and data values on controllers can vary. Only a few controllers support strong consistency, where data changes are simultaneously executed on all the controllers. The second issue is in the case of replacing a failed controller. If a controller fails completely, his neighbour can replace its role. A different situation is, if a controller starts to behave incorrectly. This can be impossible to detect by the neighbouring controllers, potentially making part of the network unusable.

The distributed architecture requires communication between controllers. This is achieved via additional interfaces - referred to as *westbound* and *eastbound*. These interfaces are sometimes used identically, but they can express different communication patterns. The westbound interface is used for communication between SDN controllers, while the eastbound interface is used for communication between a controller and the traditional network control plane [20].

Management Plane

The management plane provides a layer for advanced SDN applications, which use the northbound interface of an SDN controller. This approach has many benefits over applications implemented within a controller. Firstly, the application cannot negatively influence the controller, if it uses standard APIs. Secondly, the application is independent of the controller architecture. This means, that it can be written in any

[7]Controller connectivity loss / failover modes are described in *SDN Architectural Requirements* section 6.2.

Table 2.2: Most Common Open-source SDN Controllers

Name	Language	Architecture	OF ver.	Last version
Floodlight	Java	Centralized	1.0 - 1.4	1.2 (02/2016)
NOX	C++	Centralized	1.0	1.2 (2014)
OpenDaylight	Java	Distributed	1.0, 1.3.2	03/2018
ONOS	Java	Distributed	1.3	1.10 (05/2017)
POX	Python	Centralized	1.0	2013
RYU	Python	Centralized	1.0 - 1.5	4.23 (03/2018)

Note: valid on 31/05/2018
Data sources: official web sites and documentations of mentioned controllers

programming language and can use practically any technology. Finally, the application development can be greatly simplified as it does not require detailed knowledge of the controller architecture. The application can even be potentially migrated into a different SDN controller, if it supports the same API features.

Development of SDN applications on the management plane depends on the controller support of northbound APIs. If the controller does not support required functions, the control plane has to be modified (new features added).

2.1.5 OpenFlow

OpenFlow is the most common southbound interface protocol and it was initially a synonym for SDN. However later on, SDN became an extensive framework supporting many different protocols and technologies. From that point, OpenFlow has become just a southbound protocol, which can be used for SDN (but it is still the mostly used one). OpenFlow was firstly presented in the article [21], published in 2008, and the first specification (OpenFlow 1.0.0) was released one year later [22]. OpenFlow is a protocol allowing the control of data flows in a network via flow rules located in flow tables of forwarding devices. The communication between a controller and a forwarding device using OpenFlow can be secured by either SSL (Secure Socket Layer) or TLS (Transport Layer Security) [23].

The OpenFlow protocol exists in several versions. The major versions with their protocol number in hexadecimal format, release dates, and the most important features are summarized below [22, 24].

OpenFlow 1.0 (0x01, 12/2009)

The first official version of OpenFlow introduced the following basic OpenFlow features.

Flow table contains flow entries consisted from *header field* (for matching datagrams), *actions*, and *counters*. The flow table is managed by an SDN controller via an OpenFlow connection.

Flow match defines header fields, which can be used for datagram classification. The supported fields are: ingress port; Ethernet type and addresses; VLAN ID and priority; IP protocol, ToS and addresses; and transport layer ports.

Actions specify, what should happen with a matched diagram. Only two actions are required: forward (to: all, controller, local, table, or incoming port) and drop. Three optional actions include forward (to normal, or flood), enqueue, and modify-field.

Message types defined by the OpenFlow connection are divided into the following three categories:

- **Controller-to-switch** - these messages are sent from a controller to a forwarding device. They include: features request, configuration set, modify-state (flow tables and port properties), read-state, send-packet, and barrier.

- **Asynchronous** - these messages are sent from a forwarding device to a controller. They include: packet-in (typically for packets, which are not matched by specific flow rules), flow-removed (a flow can be removed manually, or automatically after a timeout expires), port-status (can be changed manually, as a reaction on STP, or during a link failure), and error (can provide notification about unsupported OpenFlow actions).

- **Symmetric** - they can be sent in both directions and they are typically used as a connection keepalive mechanism. They include: hello (connection initialization), echo (request/reply), and vendor messages (additional functionalities and future revisions).

OpenFlow 1.1 (0x02, 02/2011)

The most important features of the second OpenFlow version are: multiple flow tables, groups, new tags, virtual ports, and failover modes.

Multiple flow tables can better utilize specific hardware tables (L2, L3, TCAM), which can result in higher flexibility and better performance. This also allows the logical grouping of specific functionalities such as ACLs, QoS, or routing. Any incoming datagram is initially sent to the first flow table, which can contain the *forward to the next table* rule (via a *goto* instruction). A metadata (64 bits), which will be preserved during this forwarding, might be added to the datagram. All these tables are generic and support full set of matches and actions (but an actual support will depend on capabilities of the underlying HW table).

Groups merge ports into a single entity. Datagrams can be then forwarded from all bundled ports. This allows an efficient implementation of specific forwarding methods like multicast and flooding. Each group is composed from *buckets*, which contain actions. These actions are executed before the datagram is forwarded. There are four types of groups:

- **All (required)** - executes all buckets in the group. It can be used for multicast and broadcast forwarding, and also for flooding.

- **Indirect (required)** - executes only the defined bucket in the group. If there is only a single bucket, this group is the same as *all group*.

- **Select (optional)** - executes a single bucket based on a selection algorithm (the algorithm is out of the scope of OpenFlow specification and might be user-configured, or a simple *round robin*). This logic can be used for load-balancing and multipath.

- **Fast failover (optional)** - executes the first live bucket. The buckets are evaluated sequentially in the order they were defined. A live bucket must have all its associated ports alive (up). This can be used for redundancy and high availability - in a case of a port / link failure, a backup path can be used immediately. This reduces need to contact the SDN controller.

Added tags for MPLS (Multiprotocol Label Switching), VLANs, and QinQ bring support of these technologies. Corresponding header information can be added, modified, or removed.

Virtual ports can be used for forwarding over aggregated links, or tunnels, so the ports do not have to be only physical.

Failover modes define two actions, which can happen after a controller connection failure - *fail-secure* (continues to operate in OpenFlow mode) and *fail-standalone* (reverts to normal forwarding). This feature replaced the previously defined *emergency flow cache*, which was not widely used.

OpenFlow 1.2 (0x03, 12/2011)

The most important additions are IPv6 support, changes in the matching mechanism, and support of the multiple controllers architecture.

IPv6 support allows the matching of all common header fields of this protocol including: source and destination address, protocol number, traffic class, ICMPv6 type and code, IPv6 neighbor discovery header fields, and flow label.

Extensible match replaces the static fixed length structure of match fields used in previous OpenFlow versions. The new version uses OXM (OpenFlow Extensible Match), which is composed from type-length-value format (TLV). This format has a flexible size (5 - 259 bytes), which does not have to be padded. An arbitrary header field can then be defined via the following fields:

- **oxm_class (16 bits)** - defines a class used for matching packets. Typically, the *OFPXMC_OPENFLOW_BASIC* is used. Experimenter matches can be defined in the class *OFPXMC_EXPERIMENTER.*
- **oxm_field (7 bits)** - defines a match type value specific to class field. Together with *oxm_class*, these two fields are commonly called *oxm_type.*
- **oxm_hasmask (1 bit)** - specifies, if the OXM contains a bitmask or not. This bitmask has to be located in the payload.
- **oxm_length (8 bits)** - defines the length of the OXM payload in bytes. The total length, including header, is therefore this number + 4 bytes (for the header).

This mechanism allows flexible definition of new header fields. This ensures future scalability of the OpenFlow protocol, as new networking technologies can be added. It also allows definition of an experimenter field for various test scenarios. An additional metadata can be also added into a *packet-in* message.

The extensible match also introduces the pre-requisite system, which ensures consistency of match rules. For example, if a destination IP address should be matched in an IPv4 packet, the following fields have to matched as well:

$$\begin{aligned} type &= eth\_type \\ value &= \text{0x0800} \end{aligned} \tag{2.1}$$

These fields define, that the matching packet is really an IPv4 packet (0x0800) and that it therefore contains the destination IP address field.

Multiple controllers can be used for enhanced reliability during a failover. Forwarding devices can connect to multiple controllers in parallel. In this case, connections between a forwarding device and all controllers are kept open and messages from the device are duplicated to all connected controllers. Controllers can also initiate a hand-over of connected devices and therefore achieve load-balancing. There are three defined roles of controllers:

- **Equal** - this is the default state. The controller has full access to the forwarding device and all the controllers have the same role. All the messages from a forwarding device are sent to all the controllers.
- **Slave** - the controller has read-only access to the forwarding device and it does not receive any messages from the forwarding device (except *port-status* messages). The controller is also not allowed to send any messages, which would modify the state of the forwarding device, or which would send traffic from the device (only querying messages are allowed).
- **Master** - this role has the same features as the equal role, but only a single controller can be in the master role.

Other major changes include: extensible error messages and simplified flow-mod request.

OpenFlow 1.3 (0x04, 04/2012)

The most notable changes are IPv6 extension header matching, per flow meters, auxiliary connections, and duration for statistics.

IPv6 extension header is an optional header placed between the standard IPv6 header and header of a higher layer protocol. It includes several fields, which can be dynamically added and chained via the *next header* field. The added header fields, which can now be used for the matching process, are: hop-by-hop, router, fragmentation, destination option, authentication, encapsulating security payload, no next header, out of preferred order, and unexpected.

Per flow meters can measure and control the rate of packets and therefore allows implementation of QoS functionalities. These meters can be attached to any flow entry, including the entry *forward to controller*.

Auxiliary connections enable a forwarding device to create multiple TCP / UDP / DTLS connections to the controller. These connections utilize parallelized architecture of forwarding devices to achieve higher performance. They are mostly used for *packet-in* and *packet-out* messages.

Duration for statistics adds a required parameter to most of the statistics. It gives information about how long a specific statistic lasts. For a flow entry, it indicates, how long the entry is installed on the device. For ports, groups, and meters, it indicates, how long they are alive. The value also allows to more accurately calculate packet and byte data rates.

Other major changes include: flexible expression of a forwarding device capabilities, added table-miss flow entry, tunnel-ID metadata for logical ports (GRE tunnels, MPLS ports, and VxLAN ports), cookies in *packet-in* messages, filtering of messages from a forwarding device, and added matching of the MPLS *Bottom of Stack* field (which is set in cases, where MPLS label is reused).

OpenFlow 1.4 (0x05, 08/2013)

An important change in this version of OpenFlow is a new default TCP port used for communication between forwarding devices and a controller. It is now 6653 (allocated by IANA[8]), and the previous ports (6633, 976) should not be used any more. This needs to be considered, as some of the controllers, or forwarding devices still use old ports and therefore communication might not be established by default.

[8]More information: `https://tinyurl.com/yah5f6f7`

Other important changes include protocol extensibility, multi-controller monitoring, flow table eviction, vacancy events, and bundles.

Improved extensibility of the protocol is achieved by use of OXM (originally introduced for match header in OpenFlow version 1.2) for other parts of the protocol. It includes error properties and the following structures: port, table, queue, set-async, instruction, action, and experimenter.

Multi-controller monitoring allows a controller to see real time changes of a forwarding device's flow table, which were done by a different controller. The entire flow table, or its certain parts can be monitored. A similar feature allows monitoring of group and meter changes.

Flow table eviction adds an importance field to flow rules. When the flow table becomes full, the rules with lower importance can be automatically deleted to make space for new rules. This eliminates problematic situations, where a controller must decide, which rules to delete and which to keep. Such a decision process might be time consuming, which can result in a forwarding disruption.

Vacancy events allow notification to a controller about reached threshold of the flow table capacity. This gives the controller time to react, by for example, deleting lower priority flow rules, or by aggregating some of them. This mechanism can therefore prevent situations, where the flow table becomes full.

Bundles mechanism groups multiple OpenFlow messages into a single operation. When a bundle is committed, all its operations have to be either applied as a single operation, or rejected without being applied.

Other major changes include: more descriptive reasons for *packet-in*, support of optical port properties, synchronized flow tables (multiple local flow tables can allow faster operations), and new error codes.

OpenFlow 1.5 (0x06, 12/2014)

The version 1.5.1 (03/2015) is currently the most recent version, which is openly available [24]. The version 1.5 introduced mainly egress tables, support for non-Ethernet frames, improvements to statistics, TCP flags matching, and scheduled bundles.

Egress tables enable message processing on an output port (in addition to the existing processing on input ports). Multiple egress tables can be associated with a single port.

Packet type aware pipeline adds support for non-Ethernet frames (PPP). The mechanism uses the same OXM definition, which allows specification of an arbitrary frame type. Currently, there is a limitation of a single frame type defined per forwarding device.

Statistics improvements include use of a new extensible format - OpenFlow eXtensible Statistics (OXS), which is compatible with OXM and which uses the same TLV format. It expresses the existing fields (flow duration, flow count, packet count, and byte count) and also adds a new statistic: *flow idle time*. Another statistics improvement is a trigger, which enables the forwarding device to send a statistic reply only when a defined threshold is met. This reduces the high overhead otherwise caused by the statistics collection.

TCP flags matching can match all TCP flags including SYN, ACK, and FIN. These flags can be used to detect start, end, and state of a TCP connection. This functionality is essential for implementation of a stateful firewall (such a firewall can keep track of connection states and can perform the application layer inspection).

Scheduled bundles extends the traditional bundles introduced in OpenFlow version 1.4 with two features. The first one allows the setting of a specific execution time for the bundle. The second feature allows querying for bundle capabilities (if it supports atomic, ordered, or scheduled bundles).

Other major changes include: copy-field action (can copy a field from one header into another one), selective group bucket operations, improved set-field action (metadata and wildcard support), controller connection status (for monitoring status of other controllers), improved pipeline fields, port status messages sent even on OpenFlow configuration change, 64-bits long experimenter OXMs, and unified multipart requests (for group, port, and queue).

OpenFlow 1.6 (09/2016)

This version is not yet officially available and is accessible only for the members of *Open Networking Foundation* [10].

Concluding Remarks on OpenFlow Versions

There are also minor versions of OpenFlow (1.3.1, 1.3.2, etc.). These are not included in this brief summary as they mostly fix bugs and improve performance. They are included automatically in the closest higher version of OpenFlow. More information about each version can be found in the official specification [25].

Despite the fact that the newest OpenFlow version is almost two years old, nowadays, the most used versions are still 1.0 and 1.3. This is caused mainly by difficult implementation on hardware switches and the low motivation of vendors to implement new versions. The situation is better in software switches and controllers, where the more recent versions of OpenFlow are typically supported. Nevertheless, the OpenFlow connectivity between a controller and a forwarding device can be established only if both of these devices are using the same version of OpenFlow (the appropriate version can be negotiated automatically if more versions are supported).

2.2 SDN Related Concepts

The most traditional SDN definition was described in section 2.1.1, however because of the SDN complexity and scale, many different technologies and concepts are also considered to be SDN. This section gives a brief summary into some of these technologies. While some of them can be just the southbound API, some of them are complex systems supplementing, or extending the functionality of normal SDN. Also because of dynamic advancements in this field, some of the technologies are not yet finished or are not fully supported by the current SDN controllers.

2.2.1 Hybrid SDN

Hybrid SDN is a concept combining traditional networking (TN) with SDN. There are several different approaches using different technologies. Probably the most common concept is a mix of the three types of devices within a single network: TN devices (switches and routers not supporting SDN), SDN-enabled devices, and devices supporting both technologies. This concept was described in [26], where it was divided into four models as shown in Figure 2.3:

- **Topology-based model** - the network is separated into zones, which are controlled by either TN or by SDN. This model requires some form of communication between these two zones in order to achieve end-to-end connectivity across the network.

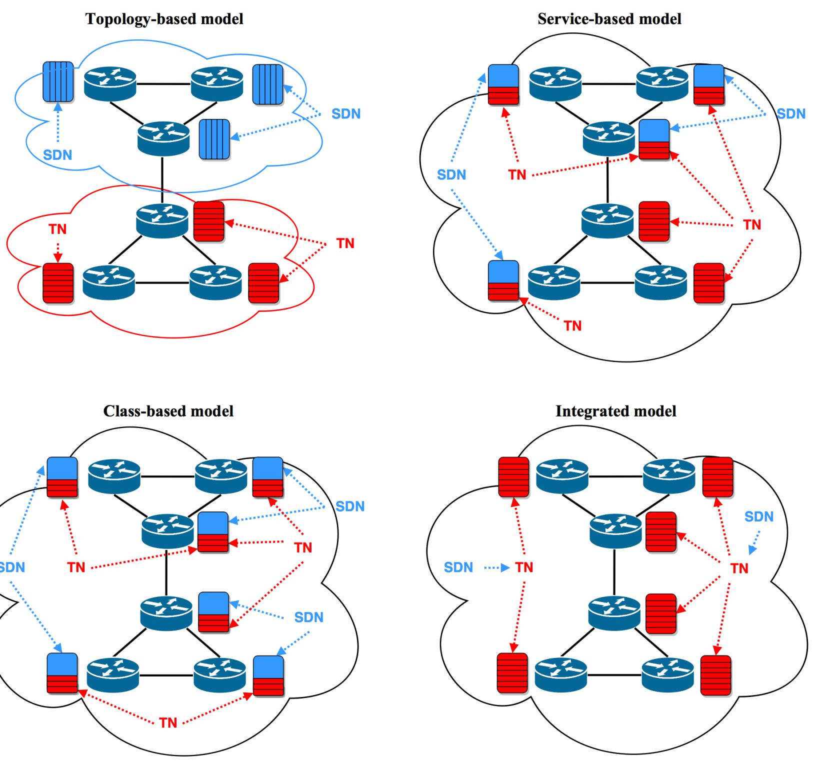


Figure 2.3: Hybrid SDN Models

- **Service-based model** - certain network services are managed by SDN, while others are managed by TN. Typically, a basic forwarding functionality is left to the TN stack, while SDN is responsible for more advanced features like load-balancing. For this reason, some of the nodes are controlled solely by TN, while some of them use a combination of TN and SDN.

- **Class-based model** - there are separate traffic classes present on every device, while these classes are controlled by a TN stack or SDN. Every device in the network is therefore controlled by both approaches.

- **Integrated model** - SDN is responsible for all controlling logic, but it uses TN protocols to achieve it. That means that the TN protocols are used as an SDN southbound interface. Typical protocols used for this functionality are BGP (Border Gateway Protocol) and MPLS.

Another concept of hybrid SDN is called *Hybrid OpenFlow*. In this case, the traffic is controlled by the flow table, which is managed by an SDN controller. The only difference is, that output action *normal* is used for certain flows. This action redirects the matching flows into the traditional stack present on the device. The traditional stack is then responsible for the forwarding.

2.2.2 SDN Related Protocols

These protocols can be used for implementation of SDN, or they mimic the SDN functionality.

GBP

GBP (Group Based Policy) is an implementation of an *intent system* and it is used for example in the OpenDaylight controller. The intent system allows the expression of configuration in a declarative way - to specify "*what to do*" instead of "*how to do it*" (imperative way). It can be also referred to as a declarative language. This approach can simplify configuration by permitting the use of easy to understand phrases like "*create a link from A to B with optimal link capacity for VoIP*". GBP is then responsible for translating such a command into the rules and for pushing these rules into appropriate devices. This approach is not dependent on underlying hardware technologies and the same intents can therefore be easily transferred amongst different network topologies, or used without modification when the network topology is changed. Another advantage is in increased code security via specification of semantic constraints, so the code is less prone to errors. In GBP, an intent is defined as a policy and it expresses law, rule, or regulation. More information about GBP can be found in the *GBP User Guide* of the OpenDaylight controller [27].

Intent-Based Networking (IBN) is now getting more attention, as it is view as a tool for management of complex future networks - including IoT networks [28]. It is expected, that the future IBN solutions will include AI, which should be able to automatically learn from the network events. In order to be really helpful solution, the IBN should be [29]:

- **Extensible** - it has to be able to evolve with new protocols, but also support the old ones.
- **Focused on business** - intents must support general non-technical terminology.
- **Holistic** - it has to cover the complete network as a single entity.
- **Validated** - it has to offer the option to show changes, which will be implemented, for validation.
- **Vendor independent** - it must be open, accessible via APIs, and not locked to a single vendor.

OpFlex

OpFlex (An Open Policy Protocol), being developed by Cisco and its partners, is another protocol using the declarative system. OpFlex is focusing on definition of policies, which are stored in a centralized entity. Each policy can, in an abstract way, define intended network behaviour like "*I want things to look like x*" [30]. In order to transfer specified policies into a networking device, XML (Extensible Markup Language) or JSON (JavaScript Object Notation) can be used, together with a standardized remote procedure call mechanisms (JSON-RPC) and secured via standard protocols (SSL, TLS). OpFlex is currently defined only as the IETF draft [31] and it is still a work in progress.

Other Protocols

- **Interface to the Routing System (I2RS)** - adds SDN functionality into the existing routing decisions. It interacts with the RIB (Routing Information Base) instead of the FIB (Forwarding) - unlike the traditional SDN. Interaction with the higher layer data structure allows addition of the following functionalities: inject and retrieve information, optimize traffic flows, choose a network exit point, rapid re-route on specific events, topology monitoring, and notifications. I2RS uses protocols of the traditional networking like CLI, SNMP (Simple Network Management Protocol) and NetConf. I2RS is still a work in progress, but it is partially implemented in OpenDaylight controller. More information about the current state of the I2RS can be found in the official charter document [32].

- **Abstraction and Control of Transport Networks (ACTN)** - a virtualization platform allows the management of non-homogeneous networks including the hybrid ones [33].
- **Forwarding and Control Element Separation (ForCES)** - a southbound protocol similar to OpenFlow, but unlike OpenFlow, it also supports wireless technologies including LTE (Long-Term Evolution) [34].
- **NETCONF** - a southbound protocol for configuration of devices. It has advanced features like transactions and event notifications. It can be used instead of traditional ways of configuration like CLI and SNMP. It is supported by OpenDaylight and ONOS (Open Network Operating System) controllers [35].
- **Open vSwitch Database Management Protocol (OVSDB)** - another southbound protocol for configuration of OVS. It is supported by OpenDaylight controller [36].
- **Path Computation Element Protocol (PCEP)** - a protocol allowing to combine devices supporting SDN with legacy devices for path computation. Messages are encapsulated in MPLS packets for compatibility with the traditional networking devices. It is supported by OpenDaylight controller [37].

2.2.3 Network Function Virtualization

NFV is a virtualization technology of various network equipment including switches, routers, servers, and storage devices. Its main goal is to simplify deployment of new network services and to reduce cost. NFV is based mainly on the following concepts from cloud computing: hardware virtualization (hypervisors), network virtualization (virtual switches), advanced management functions (automatic backups, snapshots, initialization, scalability), and open protocols (OpenFlow) and tools (OpenStack) for functions integration.

SDN is nowadays often connected with NFV. Although both technologies try to achieve similar goals, they are radically different. SDN is older (first appearance of OpenFlow was in 2008 [21], while NFV was created in 2012 [38]) and it was created in a university environment by researchers. NFV, on the other hand, was created by the ETSI - consortium of service providers [39].

The main goal of SDN is simply to separate control and data planes, and provide centralization. The main goal of NFV is to move network functions from specialized dedicated devices into virtualized generic machines. Both technologies therefore use network abstraction in order to run functionalities in software. In its basic form, the goal of NFV can be achieved by use of SDN. But most of the time, the NFV functionality is much broader and SDN is only one of the functions, which NFV uses. An important difference between these technologies is also the targeted ISO/OSI layer. While SDN focuses mainly lower layers (2 - 4), the NFV typically targets L7 (but can be deployed in L3 - L7).

NFV Benefits

NFV can reduce cost for infrastructure and power consumption, as the number of hardware devices is reduced and the infrastructure is therefore simplified. This is possible due to high volume sales of servers with standardized and interchangeable components, which are much cheaper and versatile than ASICs (Application Specic Integrated Circuits). Moreover, performance of virtualized devices running on these servers can be changed on demand and network capacity over-provisioning is thererfore not needed. Updates of these devices are also simplified.

NFV, due to its openness, also allows usage of multi-vendor resources in a single network. They can be shared amongst different applications, users, and operators. This also increases the network availability and efficiency, and enables use of advanced scenarios (for example testing of new features on a production network without influencing its performance). The network reliability and performance can be greatly improved by dynamic topology and configuration modifications based on real-time traffic. NFV also helps with management features like automated installation, scaling, re-use of prepared VMs, and advanced traffic analysis [40]. Finally, better and faster innovation can be achieved by rapid function deployment, which can be simply triggered on demand [38].

NFV Components

NFV is composed from the three main components:

1. **Infrastructure (NFVI)** - physical and virtual devices of the network.
2. **Virtualized Network Functions (VNF)** - any software functionality implemented into the network. A typical network contains multiple VNFs.
3. **Management and Orchestration (MANO)** - a layer responsible for effective management of the whole life cycle of the network.

1) NFV Infrastructure

NFVI provides resources for the VNF to run on. The communication between these two components is realized via the same datapath as network traffic. NFVI is managed by a Virtualized Infrastructure Manager (VIM) via a dedicated link. NFVI can be separated into three layers [41]:

1. **Physical infrastructure** - network devices (traditional L2/L3, or bare-metal switches), servers (computational HW), and storages (Storage Area Network - SAN, Network Attached Storage - NAS).
2. **Virtualization layer** - represented by a hypervisor, which manages the hardware resources and provides access to these resources to the applications. It is also responsible for VMs isolation and peripheral emulation.
3. **Virtual infrastructure** - it includes VMs, virtual storage and virtual networking (provides switching between VMs, security, and Internet connectivity).

2) Virtualized Network Functions

Current VNFs are similar to the SaaS model used in cloud environments. They are composed from small components (VNFCs) and together provide services, which would otherwise require a dedicated hardware.

Because of their novelty, the functions still have problems like slower performance (due to their SW nature), questionable security, and problematic compliance to standards. It is recommended to use them especially for services on L4 - L7, or to combine them with usage of SDN [42]. It is however expected, that in the near future, their applicability will move to lower layers as well.

3) NFV Management and Orchestration

MANO is responsible for management of infrastructure and VNFs. It has the ability to perform a complete network deployment testing. This can verify performance, stability, and can identify traffic patterns. Because the test can be fully automated, it can reduce cost and time, when compared to traditional network tests [43]. MANO has many open-source solutions like: OSM (Open Source MANO), Open-O, Tacker, OpenBaton; and also commercial ones [43].

MANO contains the following components[9] [40, 43]:

- **VIM** - controls and manages the NFVI. The most well-known example of a VIM is OpenStack [44].
- **VNF Manager** - performs complete life cycle management of multiple types of VNF instances (potentially from different vendors - it can use several interfaces and programming languages). It conducts actions like initializing, scaling, updating, and terminating.
- **NFV Orchestrator** - is responsible for management of two parts: network services (in coordination with a VNF Manager) and network resources (via VIMs). If multiple orchestrators are used, a centralized one (called Umbrella NFV Orchestrator) can be used for supervision and management of network services, which belong to its management domain.

NFV Challenges and Research Fields

The modern nature of NFV means, that there are some challenges, which still have to be addressed [38]. The two most notable ones are performance and security.

Performance of various virtualized network services has to be thoroughly tested and properly evaluated in all conditions. Because the software runs on generic (not a specialized) hardware, performance can be impacted. In this case, it can be desirable to improve it by code optimization or by using paralellization. The performance should ideally be consistent amongst different hardware platforms and on different hypervisors. **Security** has to offer protection from attacks, misconfigurations, and software flaws - on all virtual devices and all virtualized functions. Reliability should be further ensured by resilient hardware architectures. Subsequent security certifications might be required to prove, that the security level corresponds to security of similar traditional devices.

Hardware interoperability, portability and co-existence are few of the main benefits of NFV, but they require a unified and open interface to work correctly amongst different vendors. Migration tools also need to be developed to provide an option to operate in hybrid environments. **Effective management** is another challenging area, if full automation, flexibility, and continuous network stability are required. It needs a complex testing of various technologies and situations including resource re-allocation and infrastructure modification.

[9]The functionality of these components can vary as they do not have to be all included in an architecture. Also, there can be multiple components of the same type to provide specific features.

2.2.4 Programmable Data Plane

A programmable data plane is a newly emerging concept. In traditional networks and SDNs, the data plane on hardware switches is managed by highly specialized chips (ASICs). These chips perform fixed-functions and their usage comes from the past, when their performance was up to 100 times higher than performance of general programmable chips [45]. Today's situation is different, as there are now chips like FPGA (Field Programmable Gate Array) [46], or solutions like *reconfigurable match tables* [47], which can reach the same forwarding performance as fixed-function chips.

The most recent example of the programmable data plane approach is *P4* language [48]. P4 allows free definition of what a forwarding device should do and how to do it. Unlike in OpenFlow, there are no limitations on matched header fields, selected statistics, or restricted actions. This is why P4 is often referred to as OpenFlow 2.0, although that is not the case. But the fact is, that flexibility of P4 allows (amongst other things) implementation of the entire logic of OpenFlow.

The main advantages of P4 are: flexibility (forwarding specified by a program), expressiveness (HW-independent and sophisticated packet processing algorithms), abstract resource mapping, software engineering (type checking, code reuse, information hiding), component libraries, decoupled hardware and software, and debugging [49].

Since 2018, P4 is supported in ONOS [50] and there are initiatives to implement it into OVS as well[10]. The entire specification of the first version of P4 language can be found in [49].

2.3 Internet of Things

The origin of the term IoT is generally associated with the presentation of Kevin Ashton in 1999 [51], where he mentioned an idea of connecting RFIDs (Radio-Frequency Identification) of supply chain parts to the Internet. After several research-only attempts, the first large-scale real world application in this area (goods marking with RFID tags for inventory control) was accomplished by Wal-Mart and U.S. Department of Defence in 2005 [52].

Since the beginning, many (often contradictory) definitions of IoT have emerged. Probably the simplest definition is: "*IoT is simply the point in time when more "things or objects" were connected to the Internet than people*" [53]. This definition however does not state, what the IoT really is. A more describing definition is: "*The IoT is what we get when we connect Things, which are not operated by humans, to the Internet*" [54]. The most important concept of this definition is to omit the human element from the system. This is supported in the paper [55]: "*...smart sensors collaborate directly without human involvement...*". This comes from the premise, that in data processing, a human is slow, error-prone, and inefficient when compared to any electronic device used in IoT. The direct collaboration allows interconnection of ordinary devices (M2M communication), which can then perform intelligent decisions. These devices gain abilities to hear, see, and sense, and that makes them smart. Paper [56] mentions an important function of these devices: "*The IoT links the objects of the real world with the virtual world...* ". This feature becomes especially important in case of cyber attacks, which are no longer only virtual, but can influence the real world as well.

Two additional definitions contradict the previous ones by including humans: "*(IoT) refers to a world where physical objects and beings, as well as virtual data and environments, all interact with each other in the same space and time*" and "*(IoT) allows people*

[10]Available: `https://github.com/P4-vSwitch`

and things to be connected Anytime, Anyplace, with Anything and Anyone, ideally using Any path/network and Any service" [56]. While these definitions are conflicting with the previous ones, the second one adds a new key concept: heterogeneity in supported devices and communication protocols. The last important IoT concept was mentioned in this description [56]: "*(IoT) can be deployed also in inaccessible, or remote spaces (oil platforms, mines, forests, tunnels, pipes, etc.) or in cases of emergencies or hazardous situations (earthquakes, fire, floods, radiation areas, etc.)*". Such deployments can potentially save lives and resources.

The importance of IoT has been recognized relatively early even by politicians. In 2009, during a speech about IoT, Chinese prime minister Wen Jiabao presented the following equation: "*Internet + Internet of Things = Wisdom of the Earth*" [56].

All the mentioned definitions prove the extensive boom of IoT, which is now ubiquitous. Almost all new devices in IT have Internet connectivity options and they can be labelled as IoT devices. IoT is becoming integrated into more and more fields of cyber and physical life.

State of Internet of Things

According to the first definition, IoT originated in year 2011, when the number of connected devices overtook the number of living people on Earth [57]. Since then, the number of connected devices has been expanding rapidly. In 2013, it increased to 9 billion and by 2020 it is expected to be between 24 billion [57] to up to 50 [53] or even 100 billion[11] [56]. This significant increase will present a considerable challenge for the current Internet infrastructure, which needs to evolve correspondingly.

According to [58], the current IoT systems are isolated and do not address scalability and manageability of devices. Lack of use of a common model brings compatibility issues, which can lead to privacy and security problems. Moreover, every solution is built independently on each other, making portability and interoperability the most complicated and deployment of each solution costly.

An ITU report [59] identified the most important IoT challenges as: standardization and harmonisation, privacy implications, and socio-ethical issues. The ICTAG working group [60] identified the main challenges as:

- **Edge technologies** - smart sensors attached to real world objects and RFIDs.
- **Middleware systems** - to bring abstraction for cooperation of heterogeneous devices, networks, and sensors.
- **Networking technologies** - to support bidirectional communication via wired and wireless links with high availability.
- **Platform services** - effective management of enormous number of devices with scalability, high availability, safety, and security in mind.

2.3.1 IoT Architecture

There are currently several architectures in IoT, however there is not yet a single accepted reference model. The key elements of every IoT architecture are: scalability, modularity, extensibility, flexibility, resiliency, security, and interoperability between heterogeneous IoT devices [56, 58, 61]. Such an architecture must not be locked to a single OS, programming language, or communication protocol and should use open-source solutions. It should also support device search, discovery, and peer networking.

[11]These sources are using different estimation methodologies.

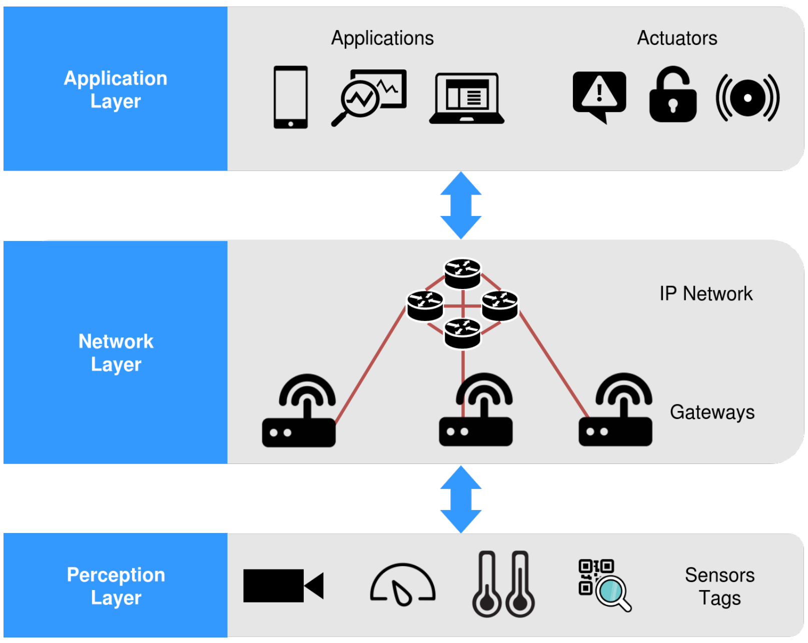


Figure 2.4: Three-layer IoT Architecture

Three-Layer Architecture

The most basic architecture, which is commonly accepted [62], is the three-layer architecture shown in Figure 2.4. This architecture is composed from the following layers [55, 61]: perception, network, and application.

1) Perception Layer

The perception layer contains "*things*" - real and physical entities, which exist in space and can be identified [56]. These things are able to "sense" - collect and monitor data about environment. They can be divided into three categories [56]:

- **Ultra low cost** - have no power source and very limited functionality. Things contain only necessary identification data. Other functionalities are performed in data centres with IoT applications. The typical example of ultra low cost devices are RFID tags, which can be read from up to 15 cm (new versions up to 7 meters) [59]. They are used in the following applications [59]: retail (automatic payments, automatic refiling), infotainment (provide additional information in galleries, cities, museums, etc.), lifestyle (chips for persons identification during sport events, chips used for special payments - in casinos, etc.), food safety (expired or infected food), etc.
- **Low cost** - they have sensing ability and their own memory for storing a limited amount of data. Some or most of the data, and application logic, is still implemented in data centres. These devices include smart materials (a carpet with integrated motion detectors), smart wearable (able to detect owner's physical status), etc. [59]
- **Smart** - are devices with a completely embedded system in the device itself. Gathered data is stored locally and devices can perform all advanced IoT functions - including sensing, monitoring, and data management. These devices typically include smart appliances (oven, fridge, camera), HVAC (heat, ventilation,

air conditioning) systems, smart vehicles, and robots [59]. In laboratory conditions, smart devices can be represented by SBCs (Single Board Computers) like *Raspberry Pi* or *Arduino*, equipped with appropriate sensors.

2) Network Layer

The network layer connects sensors with the data destination. The **first part** of this layer is the access layer network, which connects perception layer devices and gateways. This network has the following parameters [56]: topology (star, mesh, single/multi hop), size (number of supported nodes), coverage, connectivity (continuous, occasional, sporadic, manual), lifetime, mobility factor, energy constraints, cost, nodes heterogeneity, and communication type (radio frequency, acoustic, optical, etc.).

The **second part** of the network layer abstracts different communication technologies of sensors and processes the data into the common IP network. Typical devices used in this part of the layer are *gateways*, which can be realized by specialized switches, routers, or just SBCs.

The **third part** is the traditional IP network, responsible for reliable and fast delivery of data. This can be a challenge, as large amount of traffic can be generated by sensor devices. Use of QoS and virtual traffic separation can help.

The network layer can perform data processing as authors of the paper [61] suggested. In that case, an infrastructure, able to cope with the amount of data, must be implemented. An ideal technology for storage and processing similar quantities of data is cloud. Unfortunately, future research in advanced processing in this area is needed [61]. The network layer should ideally support the following features [56]:

- **Anonymous networking** - to ensure sensors' privacy.
- **Autonomous networking** - self-configuration, self-optimisation, and auto-recovery.
- **Caching and synchronization of messages** - as some IoT devices can lack permanent network connectivity (goods marked with RFID chips, interferences, limited battery).
- **Information sharing between IoT devices** - should be controlled by an authorization.
- **Mobility of IoT devices** - devices can dynamically change their position with various speeds.
- **Password and identity distribution mechanisms** - at the network level.
- **Scalable communication** - power awareness with an option to turn on/off links based on current traffic.

3) Application Layer

The application layer performs IoT functionalities based on processed data from the network layer. It can provide APIs to the variety of devices - the most common ones, used by regular users, will be smartphones, tablets, or laptops. Using this interface, users can gain insight into the monitored area and can control its properties. This is achieved by devices called actuators, which perform actions in the real world. They can be controlled either manually by users (devices like a climate control, heating, door locks, etc.), or automatically by IoT applications (devices like sprinkle system, alarm, horn, door locks, etc.). Smart behaviour of IoT is achieved by autonomous performing of these functionalities in reaction to real world events.

IoT-A

IoT-A (Internet of Things - Architecture) [58] is a document, which tries to achieve interoperability amongst various IoT platforms. In order to do so, the document presents a *reference model* and *reference architecture*.

The **reference model** describes basic concepts of IoT to provide a common understanding of the technology. The model consists of the following sub-models:

- **Communication model** - it provides abstraction for interaction with various heterogeneous devices and communication protocols.
- **Domain model** - includes IoT devices, services, virtual entities, and it defines relations between these instances.
- **Functional model** - it groups functionalities of the domain model and allows interaction with IoT instances.
- **Information model** - it provides information about the IoT system. It stores data in defined structures.
- **Trust, security, and privacy model** - it is responsible for security elements of the IoT system.

The **reference architecture** describes basic building blocks of IoT and design choices to fulfil the conflicting requirements of IoT systems: performance, ease of deployment, and security. It contains the following elements (ordered by layers):

- **Application** - the top layer contains logic of an IoT system.
- **Management** - this layer uses FCAPS (Fault, Configuration, Accounting, Performance, Security) framework, which is also used in smart grid (SG) applications [63]. This framework is extended to support IoT systems and not only pure networking solutions, which are the primary goal of the framework.
- **Service organization** - it connects various sub-groups and includes service composition, service orchestration, and service choreography. It also supports publish/subscribe communication model.
- **IoT process management** - contains process modelling and it connects executed applications to appropriate services.
- **Virtual entity** - it provides functionalities to subscribe and unsubscribe to data published by an IoT device. It also allows reading and writing of values into entity attributes.
- **IoT service** - it provides services for discovery, look-up, name resolution, and other functionalities.
- **Security** - it contains the following components: authentication, authorization, key exchange and management, trust and reputation, and identity management.
- **Communication** - it provides a common interface to many different IoT protocols. It contains three layers of communication: hop-to-hop, network, and end-to-end.
- **Device** - the bottom layer contains the physical IoT devices.

The architecture also describes four information exchange models: push-pattern, request/response, subscribe/notify, and publish/subscribe. There are also three deployments options for an application: on smart objects (for lightweight services), on gateways (for a more complicated code), or in the cloud (for advanced availability, if latency is not an issue). The application can also store data locally, remotely on a web, or locally with a web cache (hierarchical structure).

2.3.2 Other IoT Architectures

According to [61], several other IoT architectures exist, including tailored and clean slate architectures. **Tailored architectures** are integrated with other technologies, or use only a specific protocol. They include: IETF protocol suite (IEEE 802.15.4, 6LoWPAN, CoAP), SENSEI project (ended in 2011), CASAGRAS project (defines three layer architecture), Server-Based IoT Architecture (for IoT devices with extremely limited computational resources), and network virtualization (for enhanced security).

Clean slate architectures include: BRIDGE (Building Radio frequency IDentification solutions for the Global Environment - ended in 2009), IDRA (Information DRiven Architecture - for low performance WSNs), EPC (Electronic Product Code - communication between different types of networks), Cloud based approach (storage for data), and Social IoT (concepts from social relationships). Detailed description of these architectures can be found in the paper [61].

2.3.3 Future of IoT

It is expected, that IoT would evolve in various fields, and that it would acquire new functionalities like self-aware and self-organizing networks, and adaptive security [56]. These features could be achieved by SDN. The two most important technologies, which are expected to be soon connected with IoT are 5G networks and *Tactile Internet*.

5G

The major change in IoT communication technology is expected from 5G networks, which should arrive in 2020 [64]. According to [65], the 5G should become the backbone network for IoT as it should be able to carry about 1000 times more data traffic than cellular systems (4G) in 2015.

5G networks should provide low latency (around 2 ms [64]), authentication, high bandwidth, and reliability. If the technology becomes widespread, it could also solve the heterogeneity problem as it could be deployed in all communication scenarios. These features make the technology perfect for IoT.

The 5G white paper [66] summarizes the key use cases for 5G networks (stated data rates are using the following order: downlink / uplink speed):

- **Broadband access everywhere** - with expected minimum data rates of 50 / 25 Mbps and latency under 10 ms available anytime and anywhere.
- **Broadband access in dense areas** - supporting even bandwidth demanding applications like video, cloud services, and smart offices. Expected data rates are 1 / 0.5 Gbps and latency under 10 ms.
- **Broadcast-like services** - for high-demand events and news distribution, with data rates of up to 200 Mbps / 0.5 Mbps and latency under 100 ms.
- **Extreme real-time communication** - for *Tactile Internet* (explained below), with data rates 50 / 25 Mbps and extremely low latencies under 1 ms.

- **Higher user mobility** - providing stable connectivity even during high speed of users (trains). Expected performance is the same as in the broadband access.
- **Lifeline communication** - for reliable nature disaster warnings and information.
- **Massive IoT** - supporting large scale sensor networks, mobile video surveillance, and smart wearables. Expected performance in this category is low as the support of large number of devices and energy efficiency is more important. Data rates can be between 1 - 100 Kbps with the expected amount of traffic approximately one message per few hours.
- **Ultra-reliable communications** - for public safety and e-health services (including remote surgeries). This area can have data rates under 10 / 10 Mbps, but low latency - under 1 ms.

It is expected, that 5G networks will provide consistent user experience, with stable data rates and latencies (achieved in 95% areas and at 95% of time). Specific environments can have their own requirements. An example can be office buildings, where expected data rates can be up to 1 Gbps (while in normal outdoor areas, the data rate might be only 50 Mbps). Expected latency for the normal applications should be below 10 ms, while for high demanding applications, it must be under 1 ms. Moreover, these latencies must be ensured even during clients' movement. This will provide high mobility with seamless connection transitions.

Security is also an important part of 5G design, as it will be required by different applications used in these networks. Critical services like public safety and e-health will require guaranteed security. 5G will therefore use strong authentication, based on 4G solution, but with an option to implement *single sign-on* services. Comparing to 4G, 5G should provide better security with the same overhead. 5G will also include protections against typical security threats like eavesdropping, identity theft, and physical layer attacks. Privacy will be also covered as part of security features, both for users, machines, and all types of communications. The last area of security will be mobile networks, which have to be protected from the potentially harmful smartphones of users. These devices can be infected by malware, or can use dangerous applications to attack 5G networks and other devices. [66]

Other features of 5G networks include [66]: efficiency (cost, energy, and operation), ease of innovation and deployment, flexibility and scalability, and instantaneous network monitoring.

Tactile Internet

Tactile Internet (TI) is defined as *"Internet network combining low latency, a very short transfer, a high availability and high reliability with a high level of security"* [67]. After success of the mobile Internet and IoT, it is expected to be the next digital revolution. The key applications for TI are still to be researched, but nowadays, the most expected ones are [67–69]:

- **Augmented and virtual reality** - small mobile devices with limited performance (glasses) will use pre-processed content from the TI. These devices will be used for assistance, information systems, and other applications (using the concept *"see what others see"*).
- **Autonomous driving and traffic control** - continuous low latency communication between vehicles, and between vehicles and infrastructure will allow safe

and efficient autonomous driving. Traffic lights will not be necessary and cars can drive in close groups (*platooning*) to save fuel / electricity.

- **Education, culture, and therapy** - multi-modal human-machine interfaces with visual, auditory, and haptic interactions will allow remote learning and training in realistic simulations.
- **Healthcare** - remote diagnosis, rehabilitation, and even surgery should be possible with usage of tele-robots. These robots will provide audio and video information, and haptic feedback. The technology will allow remote participation of doctors on healthcare related tasks, as though they are physically present.
- **Other applications** - include the following expected use cases: industry automation (high frame rate sensors), smart grids (as they have very similar latency and reliability requirements), robotics, telepresence, gaming, and real-world simulations.

The most important technology for the TI will be 5G networks with their low latency connections. In IoT comunication, mobile networks present up to 90% of total latency [68]. TI will require latency under 1 ms, which cannot be delivered by today's mobile networks, which have the typical latency about 50 ms [68].

Other important technologies will be NFV, SDN, and fog computing (FC). Especially important will be SDN and its ability to manage the network centrally. This will be used for self-organizing networks, simplified management, organization of network flows, increased network capabilities, and network virtualization with guaranteed QoS. FC (further described in section 3.4) will be used to bring data analysis closer to the network edge in order to decrease latency. All cloud data centres are limited with transmission latency. A signal propagates near the speed of light and 1 ms latency is therefore achieved after 299.792 km. In practice, that means that when a two-way path and processing delays on transition devices are considered, the recommended distance between any device and the cloud centre should be under 100 km. [68, 69]

2.4 IoT Application Domains

IoT spans a broad range of application domains, including the following [55, 56, 70, 71]:

- **Aerospace and aviation** - it focus mainly parts classification and monitoring. Use of genuine components can improve safety and security. Typically, RFID tags are used for this purpose. Tags can be also utilized in areas like passenger and crew identification systems, and luggage and cargo systems.

  Critical airplane parts can be monitored online - by observing their properties (temperature, pressure, vibration, etc.), service inspections can be more efficiently managed. This can lower maintenance cost and improve safety.
- **Agriculture** - traceability of animals can detect an escaped animal, theft, and a non-typical health condition. This can reduce disease detection time and therefore can save animal lives as well as reduce unnecessary expanses.
- **Automotive systems** - allow complete car monitoring and control via a large amount of sensors and advanced intelligence. A car can assist the driver, or even perform an autonomous driving. The car systems are also able to communicate with each others (V2V) and with infrastructure (V2I).

- **Business environments** - includes smart and interactive offices with integrated face-to-face interfaces, linked documents, and real-time information about production data.
- **Education** - interactive school classes with remote conferences can increase learning quality. Additionally, pupils location can be monitored, which makes schools safer.
- **Environment monitoring** - safety and security of population can be improved by continuous and automated monitoring of nature events like tsunami, earthquakes, fires, floods, storms, and man-caused events like pollution, gas leaks, radiation leaks, etc.
- **Health and aged care** - patients' location within a building and their health status can be monitored by small wearable sensors (often implanted in a body). A smart building can react accordingly to ease patients movement (automatic door opening, lighting control, etc.). Early warning can be issued, if a patient's health status changes. Another application in health care can track, check, and label drugs use.
- **Industry** - any industrial process can benefit from IoT implementation. Monitoring of critical activities (oil or gas drilling, nuclear power generation, etc.) can help with preventing accidents and improving productivity and efficiency.
- **Information providing** - electronic tags (NFC, Bluetooth, and QR codes) can be placed on objects, or places of interests. Camera devices (drones) can be used to gather and provide real-time video footages.
- **Insurance** - objects like cars, mechanical devices, or even houses can be insured more effectively, if a monitoring device is attached to them. This will help insurance companies to verify an insurance event. It can also benefit users with cheaper insurance cost and faster compensation after an event. Example can be a car system like *LittleBox*[12], which uses GPS and motion sensors to detect various driving parameters. Based on the calculated driver score, insurance price can be determined.
- **Product life cycle management** - a product can be labelled and monitored during its entire life cycle. This helps to check its functionality, manage maintenance, and perform its disposal (recycling) at the best time.
- **Smart retail, logistics, and supply chain** - allows real-time monitoring of goods and stocks with RFID tags. Every item can be monitored - from industrial parts to food. Logistic processes and production can be therefore simplified and optimized. Recent initiatives by Intel [72, 73] target retail stores and vending machines, in order to improve inventory accuracy, enable low inventory alerts, optimize product placement, offer dynamic pricing (based on expiration dates, customers, loyalty programs, coupons, and advertisements), provide integration with social media, and introduce new user interfaces (gesture recognition, touch screens, etc.).
- **Smart buildings** - improved energy consumption and user comfort can be achieved by energy metering, monitoring, smart automation, and optional user control. Sensors inside buildings can also monitor user locations, and detect and react on any anomalies (health issues, burglary, fire, etc.).

[12]More information: `http://www.admiral.com/black-box-insurance/`

- **Transportation** - more efficient transportation of people and goods can be achieved by advanced screening systems, RFID tags, and intelligent transport systems.

- **Waste management** - status of all waste containers (fullness, air quality) can be monitored to achieve their efficient emptying. Disposal of electronic and dangerous waste, as well as collection of recyclable materials, can be monitored with RFIDs.

2.4.1 Smart Cities

A smart city (SC) can be defined as a city, which can "*effectively process networked information to improve outcomes on any aspect of city operations*" [74]. The main tools to accomplish this are: Internet, IoT, and web technologies. Another definition [75] states, that a SC is "*connecting the physical infrastructure, the IT infrastructure, the social infrastructure, and the business infrastructure to leverage the collective intelligence of the city*". Probably the most descriptive definition is however this one [76]: "*smart city is a seamless integration of personal, residential, commercial, municipal, private and public devices, equipment, buildings and systems for various safety, efficiency and convenience services for its citizens*".

Significance of the SC concept is becoming more and more important, as people tend to move into cities. According to [56], it is expected, that by 2050, 70% of Earth's population will live in cities. This will result in creation of mega-cities with population of dozens of millions of inhabitants. Such an enormous growth will make a high demand on city infrastructure, resources utilization, traffic congestion, and many other areas, which can be solved by the SC concept. Even cities in 2010 were responsible for 75% of worldwide demand for energy and they produced over 80% of all greenhouse gas emissions [77]. As paper [78] states, a SC is "*a strategy to mitigate the problems generated by the urban population growth and rapid urbanization*".

The main technologies of SCs are [74, 77]:

- **Buildings** - are responsible for 40% of all used energy worldwide and they include airports, hospitals, office buildings, public buildings, and homes. The key to reduce their power consumption and to increase their utility value, is smart automation. This includes safety systems (fire detectors and sprinklers, cameras, locks), climate control (temperature, humidity), lighting, and building-specific equipment (for example intelligent escalators, elevators, baggage systems, information systems, etc.).

- **Government** - online tools like virtual city hall, reservation systems, financial systems, and many others - often called *eGovernment*, are expected in modern cities. These virtual solutions provide an effective and convenient way of improving city productivity (faster response time and reduced commuting). The typical cost savings, depending on the solution, can be in a range of around 25% [77].

- **Public safety** - with an increased number of citizens, this area is more important than ever. The risk of terrorism, nature disaster, or other incident is increasing rapidly. Each such incident can, in turn have an even greater impact. Surveillance systems interconnected with tools for quick warning (city speakers, city lights, mobile phones, social media) can help reduce the threat impacts.

- **Resources** - to support increased demand, modern cities will have to be more efficient in dealing with various resources like electrical energy, water, gas, and even waste management (from the city perspective, waste management units can be considered as a resource). Real-time power consumption data can help with efficient generation and distribution, while smart sensors can detect any leakage. In combination with actuators, leakages can be greatly reduced.

- **Transport** - includes public transport, private transport, and also access for specific vehicles (emergency, supply, maintenance, repair, etc.). An efficient transportation system is the critical component for a city's productivity and energy consumption. Preferred use of public transport can be encouraged by intelligent systems, which can provide: current transport location, useful information (traffic, weather, news), and dynamic and automated payment.

- **Venues** - concerts, congresses, conferences, and sport events attract enormous numbers of people and SCs must dynamically adapt to these conditions. This may include infrastructure extensions (mobile and Wi-Fi networks capacity, additional power sources, implementation of QoS, etc.) and implementation of information boards, adaptive traffic control, and safety and security systems.

Smart City Applications

Applications of a SC are responsible only for very specific tasks. Each application should manage its own devices. According to [76,79], a SC should support the following applications: smart lighting, smart energy, smart security, and smart health.

Smart Lighting

Smart lighting (SL) systems are composed from the following components [76]: smart lamps (for example OSRAM[13], or Telensa[14]), networking infrastructure (wireless / wired), and a control system.

A SL system can use various types of smart lamps to achieve better energy consumption, lower light emissions, precise energy consumption monitoring, failure reporting, and additional services. The lower energy consumption is especially important as the used energy of traditional lights is enormous - in 2005 it reached 19% of global electricity used worldwide and was responsible for 6% of global emissions of greenhouse gases [80].

The lower energy consumption is achieved by modern LED (Light-Emitting Diode) systems, which by itself are about 70% more efficient, while having 3 - 4 times increased lifespan [81]. Moreover, the illuminating colour and intensity of these lights can be controlled, which can save energy even more. There are existing solutions, which propose to dim the lights during late night hours, if no nearby movement is detected[15]. This would also lower light pollution, which is becoming a problem in modern mega-cities. Finally, by controlling the colour and intensity, a SL system can adapt to the current weather conditions and make the traffic infrastructure safer.

Another advantage of SL is the ability to report real-time data about each light. This includes accurate information about energy consumption and also notifications, if a light fails. The cost savings for repair teams, which can be dispatched only when needed, can be significant.

[13]More information: `https://www.osram.com/ls/light-for/smart-city/index.jsp`

[14]More information: `http://www.telensa.com/smart-lighting/`

[15]Example of such a solution is CitySense. More information: `https://www.tvilight.com/citysense/`

Smart lamps can be also equipped with devices providing additional services. This can include solar cells (which can reduce the energy consumption even more), Wi-Fi access points (providing the Internet connectivity), and various sensors (cameras, motion detectors, climate sensors, parking sensors).

Smart Energy

Smart energy can utilize Virtual Power Plants (VPP), which is a program allowing participating of customers in energy generation, exchanging of price information, and sharing of energy demand data. The main goal of distributed energy resources is load reduction during peak hours. The system is composed from three components [76]: distributed generation (preferably a renewable energy source), energy storage systems (ESS), and information and a bi-directional communication system. ESS store excess energy and its state-of-the-art examples can be: Tesla's *Power Wall* (13,5 kWh / unit[16]), *Powervault* (2 - 6 kWh[17]), and many others.

There are also studies [82–84], which propose to use electric vehicles (EVs) as ESS. If such a vehicle is not in use (and its use in the near future is not planned), its battery can be used as an ESS. During high demand hours, the energy from the battery can be given to the grid (for some revenue for the owner). The battery can then be later charged back (during low demand hours). EVs have batteries with relatively large capacities (up to 100 kWh[18]) and part of this capacity can be therefore used without sacrificing mobility.

Smart Security

Smart security combines smart emergency plans and crime prevention systems. Combination of various types of CCTV (stereo cameras and vehicle license plate recognition), heat sensing, tracking and other smart sensors, can improve responses of emergency units during various security threats. A security system, composed from the mentioned sensors and managed by an integrated system, can help in search for missing persons and can prevent: murders, robberies, sex crimes, arsons, thefts, etc. [85] The key part in these scenarios is effective data processing. An integrated management system must be able to handle a massive amount of input data and process it in a minimal time.

On the other hand, massive deployments of these sensors affect citizens' privacy. According to study [86], such surveillance technologies are generally accepted only in public places (stations, public transport, malls, parks), but not in private places (schools, pubs, cars, homes).

Smart Health

Smart health is a platform for improving quality of healthcare services. Smart fitness devices can collect data about activity, walked steps, sleep patterns, and heartbeat. Other sensors located in clothes can provide additional information, mainly about activities, behaviours, and environment. The gathered data can then be used for faster emergency response and more accurate treatment. An ideal use case for deployment of these sensors is for elderly care. Smart sensors can detect anomalies and automatically call emergency.

[16]Source: `https://www.tesla.com/en_GB/powerwall`
[17]Source: `https://www.powervault.co.uk/choose-your-powervault/`
[18]Source: `https://www.tesla.com/models/design`

Another part of smart health is data integration between different health care centres and hospitals. All approved staff (doctors and nurses) should have access to patients data in real-time.

2.4.2 Smart City Domains

SC domains present complex systems, which are part of general IoT. They can also be part of a SC, where they can be interconnected with the management system. These domains will be described in more details in the following subsections.

Smart Buildings

Smart buildings (SBs or also *intelligent buildings*) are integrated into a SC with goals to reduce consumption of resources, integrate security mechanisms, and offer intelligent environment monitoring and (remote) control. The most important technology of SBs is a HVAC system for controlling the SB environment, while reducing energy demands. The system relies on smart metering devices, which give accurate information about consumed resources. A modern trend in this area is a cloud analysis tool, which can predict energy demands based on statistics, usage patterns, habits of employees, and expected weather. HVAC systems are used mostly in smart offices and smart factories. According to [87], other SB scenarios include smart schools and smart healthcare.

Smart schools use mostly environment and resource monitoring sensors to detect air quality, carbon dioxide levels, and consumption of electricity, water, and gas. This can help schools to ensure a safe environment. Another application is security via real-time students tracking and on-demand access for authorized personnel. Students can have special tracking objects (smart cards, chips, etc.), which can provide access to certain locations, manage student accounts, and interact with: other IoT devices, dining facilities, and library equipment.

Smart healthcare includes hospitals, retirement community housing, and similar health care centres. SB solutions can be used to lower operation costs, which can be important especially in these areas. Implementation of innovative IoT technologies can also monitor the health status of patients and automatically alert someone in emergency situations.

Smart Grids

Smart grid (SG) is a concept of integration of the traditional *power grid* with an information network [88]. SG can be defined as: "*... an electric system that uses information, two-way, cyber-secure communication technologies, and computational intelligence in an integrated fashion across electricity generation, transmission, substations, distribution and consumption to achieve a system that is clean, safe, secure, reliable, resilient, efficient, and sustainable*" [89]. SG is composed from many heterogeneous technologies, and it can therefore be referred to as the "*system of systems*" [90]. According to [91], the main goals of the SG are:

- Energy usage information for customers.
- Facilitation of plug-in EVs and other energy storage options.
- Improved resiliency to disruptions with self-healing capability.
- Incorporation of distributed energy sources.
- Increased power reliability.

- Optimized resources (using more optimal peak capacity).
- Reduction of greenhouse emissions.

The key concept of SG is a two-way information flow between energy suppliers and consumers. Using smart end devices like smart meters, smart thermostats, and other appliances; consumers can actively participate in controlling their power usage and therefore potentially reduce their energy fees and increase the grid stability.

According to [92], SG is one of the most important areas of IoT. Electricity consumption of modern society is increasing, and there are limits of how much energy can be generated at a given point in time. Moreover, even while a majority of electrical energy comes from the most ecological sources (nuclear being the first, and renewable resources second), during peak periods, traditional coal power plants are still being used. SG can motivate users to flatten out this peak consumption by dynamic usage of their devices.

Another SG advantage is improved reliability of the power grid. Unlike the traditional power grids, the modern ones are more widely distributed, have much higher peak demands, and use new energy generation sources with unpredictable performance. All these revolutionary steps require timely and precise information to support the system automation. Slow manual processing by a human element is no longer an option. Slow response times of the traditional grid during unexpected events were proven during various blackouts. The most serious ones were:

- **India, 2012** - The blackout started with a line tripping of 400 kV circuit breakers, which caused shutdown of a major transmission section and consequently affected several power stations. The following day, a relay problem caused additional power stations to go offline [93], affecting over 620 million people. According to the *United States Agency for International Development* [94], the blackout could have been prevented by implementation of SG with its automated fault detection and self-healing abilities.
- **Canada, 2003** - a fallen tree and a failed alarm, caused a cascade of failures, resulting in the most serious blackout in North American history. The blackout was responsible for 11 deaths, estimated damage cost was $6 billion, and it caused 50 million people to be without electricity for two days [95].
- **Italy, 2003** - a storm and insufficient trimming of trees near power lines caused a failure of the main power line between Italy and Switzerland. Afterwards, an increased load on the rest of the grid network caused two power lines between Italy and France to fail. This cascading effect cut Italy from two main energy exporting countries. The traditional grid was unable to compensate the sudden drop in frequency and the following blackout affected over 50 million people. [96]

Smart Homes

Smart home (SH) can be defined as a HAN connecting smart devices, which can be remotely controlled, most often from portable devices like smartphones or tablets. G. Bryant [97] adds to the definition: "*a home must seamlessly interconnect diverse devices, services and things so the home becomes perceptive, responsive and autonomous*".

Popularity, use, and sales of SH devices are growing rapidly. In 2016, these devices were responsible for almost 2/3 of all connected IoT devices and it is expected, that by 2020, there will be 13.5 billion of them [98]. These devices include entertainment devices (smart TVs, wireless speakers, game consoles, voice assistants), smart thermostats, smart appliances (fridges, ovens, kettles), smart lighting (including control of curtains and shutters), smart electrical outlets (with remote control, scheduling, and monitoring functionality), smart security devices (locks, surveillance systems, motion detectors), energy monitoring devices, cleaning robots, fitness devices (smart scales), electric car systems, etc. According to [99], the most popular type of devices is entertainment, followed by smart thermostats, security devices, and lighting systems.

The key technology of SHs is an integration system, which allows controlling of all devices from different vendors within a single platform. This is a difficult challenge - for example Intel in [100] managed to integrate only three distinct lighting solutions (*Phillips Hue, Cree*, and *Osram*), while stating, that with a wrong firmware, the solution can "go awry". More complex integration solutions are Apple's *HomeKit*[19] and Samsung's *SmartThings*[20]. These can be controlled from mobile applications, or voice assistants. Voice-activated speakers like *Google Home*[21], *HomePod*[22], or *Amazon Echo Dot*[23] can be controlled exclusively by voice. In April 2017, Google released SDK for its assistant technology [101], so more companies are now introducing their own versions of voice-activated speakers. The same assistant can be even experimentally implemented on SBCs like *Raspberry Pi.* These assistants integrate artificial intelligence and connected SH devices can be controlled by voice commands like "*set the temperature in living room to 20 degrees*".

2.4.3 Smart Mobility

Smart mobility (SM) or C-ITS (Cooperative Intelligent Transport Systems) is an area targeting an intelligent traffic management, most often in complex SCs. This includes public and private transport and all related areas (road conditions, parking lots, etc.). SM covers all types of V2X communications, where X can be vehicle, infrastructure, road, or any other entity. In addition to the requirements of standard IoT, this domain has also specific demands on node mobility.

SM can increase quality of life and productivity. Shorter transportation times, option to use efficient public transport (where commuters can work on their mobile devices), and less chance of delays are just some of the examples. Also, by minimizing traffic jams and optimizing traffic flows (no need to circle around looking for a parking place), produced emission, noise and consumed petrol / electricity are reduced. Congestions cause increased pollution, lower productivity, and also pose a security hazard (slower access for emergency vehicles). The optimisation can be achieved by dynamic traffic control systems - traffic lights, roads with adaptive traffic lanes, digital traffic signs (variable speed limits), etc. Another advantage is accurate prediction of users arrival time, independent of used type of transportation. SM can also significantly improve road safety. As [102] states, 1.2 million lives were lost globally in 2015 due to road traffic incidents. These incidents are the major cause of deaths amongst people in age group between 15 - 29 years[24].

SM covers mainly the following applications:

[19]More information: `https://www.apple.com/uk/ios/home/`

[20]More information: `https://www.smartthings.com/uk`

[21]More information: `https://store.google.com/product/google_home`

[22]More information: `https://www.apple.com/uk/homepod/`

[23]More information: `https://www.amazon.co.uk/Amazon-Echo-Dot-Generation-Black/dp/B01DFKBL68`

[24]Unfortunately, most of these deaths happened in less-developed countries, where SM is not expected to be implemented soon. Moreover, the most affected groups are pedestrians, cyclists and bikers, whose protection is the most complicated.

- **Mobility-as-a-Service** - is a concept of transporting people from one point to another in the most efficient way [103]. The concept is based on public transportation and (in the future) self-driven shared vehicles. Integration of digital platforms with smart sensors can provide effective transit services across various transportation forms. Users can pay with their smartphones and other payment devices (smart watches, special cards, NFC chips, etc.).

- **Smart parking places** - equipped with sensors detecting the place status. Sensors can be based on electromagnetic or proximity detectors and cameras. Data can then be used for parking place reservation, navigation, dynamic pricing, and lower traffic congestion - according to [104], up to 30% of traffic congestion in urban areas is caused by circling drivers searching for a free parking place.

- **Smart traffic management** - can lower congestions, which drastically influence the quality of city inhabitant lives. Traffic conditions can be detected by many sensors (in-vehicle, street, licence plate readers, CCTV cameras, mobile phones, etc.), and managed by a centralized system, which can control traffic lights, digital dashboard signs, and can provide notification in online applications. The traffic can be dynamically routed based on weather, road works, and accidents. Transit of priority vehicles can be sped up by dynamic adjustments of road signs. Collected traffic statistics can also be used for the future road re-constructions and for city expansion planning.

3. Related Work

This section describes related work in the SDN area. Firstly, general SDN research is described, followed up by research in more specific areas of IoT. Security, performance, and other important issues are considered as well.

3.1 General SDN Research

SDN is nowadays a highly active area of research. There are a lot of conferences targeting the SDN research like: *IEEE Conference on Network Function Virtualization and Software Defined Networks*[1], *The Symposium on SDN Research*[2], *Open Networking Summit*[3], *International Conference on Network and Service Management*[4] and many others. These conferences cover different aspects of SDN. According to [6], the current SDN research efforts can be generally divided into the following categories (more detailed explanation of each category with additional links can be found in the same source):

- **Controller platforms** - focus on hierarchy levels between distributed controllers. This covers eastbound / westbound APIs, controller modularity, load-balancing, consistency, and high availability.
- **Migration and hybrid deployments** - in order to deploy SDN in a traditional network, migration techniques have to be used. Hybrid SDN is a concept covering these migration techniques by various ways: deployment of SDN only on some devices and ports, or using OpenFlow features to combine SDN with traditional networking. This allows SDN interaction with existing legacy devices and protocols.
- **Performance evaluation** - is focused on CPU load bottlenecks, latency of reactive flow rule insertion, benchmarking tools, and controller placements. Most of the work is unfortunately conducted using only simulation tools, or analytical modelling, and not a real hardware.
- **Resilience** - focus mostly on utilization of the control plane and on comparison between SDN and traditional IP networks. Many papers describe protection schemes to ensure low latency recovery times during various failure scenarios.
- **Scalability** - address mainly overhead of the control layer (when the flows are processed in SW) and latency between a flow creation and insertion into the flow table. Less targeted issues are scalability in distributed architectures and mechanisms for consistency of these architectures.
- **SDN for ISPs and cloud providers** - SDN advantages makes the technology ideal for dynamic and complex networks of ISPs and cloud providers. These advantages, explored in many papers, include: fast recovery, efficient networking, adaptive traffic engineering, simplified fault-tolerance, performance isolation, easy resource migration, improved capacity usage, higher reliability, and cost reduction.

[1]More information: `http://nfvsdn2016.ieee-nfvsdn.org`
[2]More information: `http://conferences.sigcomm.org/sosr/2016`
[3]More information: `http://goo.gl/gZiJ2S`
[4]More information: `http://www.cnsm-conf.org/2016`

- **SDN in Software-Defined Environments** - fully programmable IT infrastructures - Software-Defined Environments (SDEs) - are based on dynamic and effective reconfiguration. SDEs include four Software-Defined blocks: Network, Storage, Compute, and Management. SDN comes as the latest block to fill the missing gap for a fully programmable network.
- **Security and dependability** - security of the SDN is an issue which still needs to be further explored and researched. Except for the same attacks as in traditional networks, SDN is also susceptible to attacks targeting controllers, controller applications, and SDN APIs. With SDN starting to be used in commercial applications, these topics are one of the most important areas for current and future research.
- **Switch design** - covers mainly performance issues like limited flow table capacity, speed of flow table modification, issues with specific switching hardware, and possible usage of more advanced hardware technologies (GPUs, modern CPUs) in existing switch architectures.

3.2 SDN in Internet of Things

According to [55], IoT is currently facing the following challenges, which will have to be addressed in the near future: availability, reliability, mobility, performance, management, scalability, interoperability, security and privacy.

Most of these challenges can be solved (at least partially) by SDN. **Availability** requires redundancy of critical services within an intelligent system, which can make sophisticated decisions. SDN controller and its applications can be a solution. **Reliability** is the availability over time and therefore it is an extremely important feature especially in critical IoT applications. SDN using a distributed controller architecture, can achieve this reliability. **Mobility** is expected in most of the deployment areas of IoT - on a user side as well as on a sensor side. Moving sensors need to seamlessly switch between different gateways, and caching has to be used to mitigate temporal data unavailability. Managing this logic from a centralized viewpoint (SDN controller) can simplify the functionality. **Performance** is still an area, which is not explored thoroughly. SDN can be used to implement QoS, load-balancing, proactive flow insertion, etc. On the other hand, specific features of SDN, especially related to software processing, have to be considered, to ensure maximum possible performance. **Management** of thousands of connected devices via various communication technologies can be accomplished more easily from a controller and appropriate applications. Also, a controller can act as an abstraction layer, unifying different technologies under a single API. **Scalability** by adding new features without disrupting the current services is the key feature for dynamically evolving IoT. SDN can support scalability with its programmability. **Interoperability** of many hardware and software technologies used in IoT is one of the biggest challenges. Some of the communication functionalities can be merged and therefore simplified by SDN. **Security and privacy** is very active research topic, but probably also the most complicated one. With enormous heterogeneous networks and lack of a common architecture, ensuring security and privacy is a real challenge. SDN can help with introducing dynamic and up-to-date applications for future enhancements of security and privacy on the communication layer.

The survey [105] adds few additional requirements. **A dynamic change of functionalities** of networking devices can be achieved by NFV (described in section 2.2.3) in combination with SDN. This change can be performed in real-time, depending on the application needs. The technology also allows simultaneous execution of multiple different functionalities. Node devices should be also **accessible** from anywhere and anytime. This can be used for a seamless remote control. The lastly mentioned feature is **network efficiency** in terms of utilized resources and consumed electrical power. This is especially important when a large scope of these networks is considered. SDN with flow-rule-based forwarding can increase this efficiency. As the servers can be dynamically turned on or off, depending on the utilization, the flow rules can automatically adjust.

On the other hand, even if SDN is utilized, IoT still presents a significant challenge. According to [106], the biggest issue is managing IoT heterogeneity. This results in various requirements on the network from specific IoT devices. These requirements (reliability, latency, jitter, bandwidth, and throughput) have to be optimized, often simultaneously. This presents a much more difficult deployment than in other areas - for example data centres (where typically only the maximum bandwidth, or the maximum link utilization is required).

SDN-based IoT Networks

The survey paper [105] divided SDN-based IoT networks into the following four aspects (more detailed description with links can be found in the paper):

- **Edge network** - includes sensors and actuators, which collect data. This requires an unified system for collection, effective aggregation, and processing. Most research in this area targets WSNs (Wireless Sensor Networks). Another big domain is network monitoring. SDN can effectively replace the traditional monitoring approaches (for example *sFlow*[5]).
- **Access network** - incorporates gateways and access points (APs), and its main functions are integration of various access technologies, dynamic resource allocation (to provide QoS), and support of multiple vendors (via an integrated architecture). The current research in the area focuses mostly on optical networks. Several integration solutions for different access technologies (wireless and optical) were introduced, together with architectures for publisher-subscriber type of communication, or a more effective QoS (for example via a *rule caching*).
- **Core (backbone) network** - is responsible for routing of traffic between the entire IoT network and therefore it requires mostly a reliable flow classification (for QoS). The current research in the SDN area addresses this issue only very briefly and only a few QoS applications are presented (via load-balancing and traffic monitoring). The security aspect of core networks is practically non-existing, or it combines SDN with traditional IDSs (Intrusion Detection Systems) [105].
- **Data centres** - they process and store data collected from connected IoT devices and the IoT network. The biggest concerns are efficient flow handling (long and short-lived flows) and issues connected with VMs (traffic awareness, mobility, energy-efficiency, over/under-subscriptions). Some research papers introduced SDN-based cloud architectures, routing approaches for clouds, and solutions for VM migrations.

[5]Available: `https://sflow.org/`

3.3 SDN in Specific IoT Areas

SDN research can be further divided by areas where SDN is deployed. These areas include: IoT (smart cities, smart grid, smart homes, smart mobility), data centres, and clouds. Each of these areas and the current state of the research will be further described. As mentioned in [106], the work in specific IoT areas is isolated and without a common layered SDN methodology. Moreover, most of the work ignores security issues.

3.3.1 SDN in Smart Cities

Implementation of IoT within a SC domain requires connecting up to millions of sensors, using predominantly wireless communication technologies. A general SDN architecture for implementation within a SC was proposed in [107]. Unfortunately, this architecture only briefly describes four basic layers of general SDN: data, network, control, and application layer. A more specific SDN architecture for achieving high bandwidth and lossless connection for sensor devices using IPv6 and 6LoWPAN [108], was proposed in [109]. This paper however proposes only a conceptual and simplified view on the architecture, with no real implementation conducted.

So far, only a very brief attempt to propose a conceptual utilization of SDN for securing SCs was done in [110]. Authors, in the two-page-long paper, proposed a secure IoT architecture for SCs. Unfortunately, this paper is only theoretical and does not describe the proposed solution in any detail. Their architecture suggests to use four following blocks:

- **Black networks** - are access networks for IoT nodes, where entire message encryption should be used. This would ensure confidentiality, integrity, and privacy, but due to high overhead, it is often impossible to implement.
- **Key management system** - is a centralized entity, which manages distribution of symmetric keys for all layers of communication protocols.
- **Trusted SDN controller** - a controller is viewed as a single entity, which routes messages between nodes and supports two routing modes for nodes with a limited transmission time: *synchronized wake times* (keeps track of nodes wake-up times and creates according dynamic routes), or *random* (it routes randomly over currently awake nodes).
- **Unified registry** - is a system, which unifies various heterogeneous technologies used in IoT - including communication protocols, physical protocols, and addressing schemes. Such a solution is however very unlikely to be built in the near future as the complexity of such a task is enormous.

More detailed security issues were addressed in the following two papers. The first one [111], focused on **reliability** (which can be considered to be a part of security) via reduction of network failure risks. Authors proposed a method to reduce the network overhead by implementation of a distributed network management. They utilized a fog-based SDN controller in an emulated network of a SC.

The second one [112], introduced a solution focused on mitigating **application level DDoS** (Distributed Denial of Service) attacks. The authors proposed the *ProDefense* framework for lightweight, scalable, and easy to manage detection of these attacks. Unfortunately, the framework was described mostly from a theoretical point of view, and the paper does not contain any specific implementation details. Moreover, the framework was not validated, so there are no practical results from testing. On the other hand, authors described classification of DDoS attacks and categorized the following five attack detection techniques: entropy, machine learning, traffic patern analysis, connection rate, and SNORT[6] combined with OpenFlow. Authors also provided a list of possible mitigation actions: drop packet, block port, redirect (the legitimate traffic to a new IP), control bandwidth (QoS), reconfigure the network, deep packet inspection, change of MAC/IP of the attacked device, quarantine, and isolation.

Smart City Networking Requirements

The survey paper [76], did a research of SCs literature and identified the main requirements of SC networks and usefulness of SDN solutions to accomplish these requirements. **Configurability and management** can be effectively achieved by SDN's centralized architecture and complete overview of the network. **Heterogeneity and interoperability** can be covered by SDN only on networking appliance level (not on end-user devices, nor communication protocols). **Privacy and security** of current SDN solutions still suffer from several issues, which are mentioned in section 3.3.2. **QoS** requirements can vary by specific domains within a SC, but most of the time, minimal latency and end-to-end delay are preferred. Certain areas like automated vehicles will require ultra-low latency links (under 1 ms), or so called "*zero-latency connectivity*", which are expected to be delivered with 5G networks [113]. There is a justified concern about SDN introducing an additional latency into the network [76]. This is further described in the following section 3.3.2. While it is generally true, this latency can be minimized with the proactive flow rule insertion and use of distributed controllers, configured for load-balancing. **Reliability** in the SDN area still needs more work [76]. **Scalability** can also be an issue in SDN as the amount of traffic of a typical SC can be enormous. The high number of flows can easily fill up flow tables of modern SDN devices. Approaches such as more general flow rules, tunnel encapsulation, or traffic separation can mitigate this problem.

3.3.2 SDN in Smart Grid Networks

Typical SG networks use **MPLS** technology, which was proven to be reliable and most importantly conforms to NERC's Critical Infrastructure Protection Standard (CIP-002) [114]. MPLS also has additional useful features like: traffic engineering, VPNs, QoS, and fail-over mechanisms. On the other hand, in order to support MPLS and all its features, relatively expensive networking devices have to be used.

Sydney et. al [115] presented, that by replacing MPLS by OpenFlow, a similar level of features can be achieved with potentially lower cost. In the experiment, they successfully implemented and tested an automatic fail-over mechanism, load-balancing, and basic QoS guarantees using OpenFlow. Moreover, this work was implemented on the GENI (Global Environment for Network Innovations) platform [116] within a limited timeframe of two weeks. This has proven the agility of SDN technology deployment.

Similar findings are shown in paper [117], where authors compared MPLS and OSPF with SDN solution. SDN handled the same features as these two protocols. Moreover, SDN achieved a more effective behaviour in case of an additional network load, when recovery time had to be minimized. SDN, unlike the traditional protocols, also gives an option for adding future functionalities. This was confirmed by the results presented in [118]. Authors of this paper demonstrated, that if OpenFlow rules are inserted pre-emptively, network performance can be higher than if MPLS is used.

[6]Available: `https://www.snort.org/`

Another area where SDN can be successfully used is **multicast**, which is often used in SG networks (for more effective communication, especially between sensor devices). The paper [119] proposed a suite of algorithms for fast packet loss detection and fast re-routing, using pre-computed backup multicast trees. Goodney et al. [120] proposed a multi-rate multicast solution for more effective data delivery - it reduces the amount of unnecessary traffic.

SG networks in substations are often built on L2 and are therefore using technologies like **VLANs** (802.1Q). VLANs are a popular technology, but they have their limits in the maximum number of supported virtual networks (4096), which can be a bottleneck in SG scenarios. This issue was also addressed by SDN in [121].

Security Concerns of SDN in SG Networks

SG is a technology, which requires reliability, resiliency, and security. Unfortunately, impact on security, when SDN is deployed in a SG network, has not yet been fully explored. The research in this field is either theoretical only [122, 123], or focused on very specific experimental areas. Theoretical research [122] classifies possible threats into three categories: compromised network switches, compromised grid devices, and compromised SDN controller or its applications. The second theoretical paper [123] summarises the main advantage that SDN can bring to security: provide perfect isolation and separation of different traffic. This isolation was already mentioned in [121], where authors implemented a solution mitigating the maximum number of VLANs limitation. Another experimental concept used to increase security is through use of load-balancing, which in the case of Man-in-the-Middle (MitM) attacks, limits the amount of captured data [124].

Further security concerns are practically unexplored. Many authors [117, 118, 125] claim that the following areas need further research: improving reliability of the SDN controller, understanding the effect of network latency on SG reliability, vulnerability of the control network via switches and intelligent end devices, integration of SDN's security mechanisms, and compliance to QoS and security standards. These authors are also very well aware of the complexity of research in this area, which requires knowledge across a broad range of fields.

Latency Issues of SDN in SG Networks

A specific concern for SDN in SG scenarios is SDN's impact on latency. High priority messages such as GOOSE (Generic Object Oriented Substation Event) and SV (Sampling Value) require latency to be lower than 3 ms [126]. Such a strict requirement is difficult to accomplish with existing SDN technologies, such as OpenFlow [127]. In the OpenFlow's default state, flows are inserted reactively, in response to an incoming packet. This packet has to be initially forwarded to a controller, which will then determine, what to do with the packet and install appropriate flow rules into the switch. Evidently, this process introduces additional latency, potentially exceeding the latency requirements of the aforementioned protocols.

Multicasting at L2 represents another challenge. Current switching devices cannot handle L2 multicast, resulting in incoming traffic being flooded across all interfaces. This significantly increases the amount of traffic within the network, resulting in congestion and additional latency. A number of works [119, 120, 128] have advocated the implementation of L2 multicast behaviour in a way, which reduces latency and network utilization. However, these experiments [119, 128] are conducted using emulators and therefore do not necessarily provide an accurate picture at these levels of granularity.

Further reductions in latency can be achieved, if the traditional STP is replaced by an SDN application [129, 130]. However, these concepts still require further investigation. The authors of [129] compare their solution to traditional STP and the more advanced TRILL (Transparent Interconnection of Lots of Links). The authors show that in the case of high network utilization (2,500 simultaneous flows), their solution can still deliver almost 100% of the frames, whilst TRILL delivers around 92% and STP only 31%. On the other hand, these results are based only on simulations in OMNeT++ (Objective Modular Network Testbed in C++)[7]. The authors of [130] claim that their solution can, in specific conditions, lower latency by up to 60% when compared to the standard STP implementation in the RYU controller. This is due to the slow convergence of a network when using default STP timers. Further work [131] suggests that even the speed of more advanced protocols like rapid STP (RSTP), or TRILL can be limiting in the context of SG.

3.3.3 SDN in Smart Homes

IoT within a SH generally shares the Internet connectivity with user devices like laptops, smartphones, tablets, and TVs. This creates an additional challenge in prioritizing the traffic, in order to ensure reliable and efficient functionality. This issue was addressed in [132], by using an SDN router to measure link capacity and to shape the bandwidth for transmitted data. Such utilization of a home router is, at least, questionable. While the benefits of such a solution are limited (similar functions can be achieved without SDN), there are also significant disadvantages. The home router would have to support SDN technology, which currently makes the router substantially more expensive. The router would have to receive commands from an SDN controller located (most probably) in an ISP network. This would mean, that these instructions would be sent via the same medium as the standard traffic, which is not recommended from security and performance perspectives. Finally, by making the router remotely controllable, there are additional privacy and security issues.

3.3.4 SDN in Smart Mobility

SDN in this area is not widely used, and if it is, it is not considering security in any detail. The paper [133] theoretically describes the benefits of using SDN in this domain: improved management, higher performance, and faster network innovations.

A review of the current work in SM was presented in [134]. Authors describe several initiatives of SDN deployments: Software Defined Wireless Networks (virtualization), Odin (light virtual AP abstraction), OpenRF (self-configurability), SoftRAN (mitigating suboptimal LTE), OpenRAN (virtualization), AnyFi (a controller manages all home APs), CellSDN (real time routing decisions), SoftCell (improved overall performance), Heterogeneous Cloud Radio Access Network (suppress of co-channel interferences), Distributed SDN Control Plane - DISCO (resilient inter-domain SDN topology), and a few other unnamed techniques for providing IP mobility (mainly for real-time applications). Authors summarize the advantages of SDN as: "*SDN-based solution can dramatically enhance performance of inter-connected moving terminals while providing minimum delay and latency to satisfy IoT mobility*" [134]. Another mentioned advantage is reduction of specific technology headers (VLAN tags, MPLS labels, etc.) - they might be omitted in the SDN solution.

[7]Available: `https://omnetpp.org/`

3.4 SDN in Data Centres and Clouds

A data centre is a place housing computer systems (servers) in order to provide computational performance, storage capacity, and associated networking connectivity. Current data centres are typically using virtualization to allow dynamic resource allocation, lower cost, and simplified management.

Cloud is a data centre, providing on demand shared computer resources typically over the Internet. In order to do so, cloud uses high availability Internet connections and virtualization technology. Cloud is closely linked to IoT. If an enormous amount of data is generated in IoT, cloud is a perfect solution for storing this volume of data. Cloud is also suitable for effective analysis of so called *Big Data*[8].

From the SDN deployment perspective a private cloud is viewed as a data centre with several specific requirements, depending on the cloud type and usage. For this reason, the thesis does not differentiate between SDN deployed in a data centre or cloud. A public cloud typically cannot be used for SDN deployment, as it often provides only a limited access to its infrastructure.

SDN Deployments in Data Centre Environments

Traditional data centres are built using the three layer hierarchical design based on L2. This design has several limitations which are becoming more apparent with growth of data centres complexity, number of supported users, and links capacity. The most important limitations caused by L2 are:

- **Large amount of broadcast traffic** - L2 topology contains single broadcast domain, where all broadcasted traffic is picked up by every device. This traffic includes ARP (Address Resolution Protocol) and DHCP (Dynamic Host Conguration Protocol) requests and it presents performance and security issues.
- **Large number of MAC addresses** - switches have a limited amount of memory for storing learned MAC addresses. This memory can get full relatively easily in data centre environments. In this case, the switch has to flood all the incoming traffic, resulting in poor performance of the network and possible security issues.
- **Limited number of maximum VLANs** - only 4096 VLANs are supported. This can be a limiting factor especially if the VLANs are used to separate different tenants of a data centre. In this case, no more than 4096 isolated tenants could use the cloud.
- **Redundant paths creating loops** - these loops are managed by L2 protocols[9], which are generally much less advanced than L3 routing protocols. This includes slower failure recovery, and lack of specific features (load-balancing, specific routing).

Implementing SDN into the three layer architecture can solve the mentioned issues, if all the switches are SDN-enabled and therefore centrally controlled. This approach was described in [135]. The traditional L2 protocols can be replaced by SDN solutions, which can achieve much faster recovery and higher efficiency [117]. The maximum number of learned MAC addresses is not important for SDN - only the size of the FIB table matters. Fortunately, a single switch can have multiple FIB tables and a software implementation of the FIB table can have practically unlimited size. If the

[8]Big Data is a term for large amounts of data generated from various devices.

[9]This includes protocols like STP, RSTP, Multiple Spanning Tree Protocol, and TRILL.

FIB table size is still an issue, multiple FIB entries can be merged into more general rules, which save the space. Another approach is to use tunnelling technologies, which use addressing only on their virtual end points [136]. Work [137] enabled to utilize up to 1 million virtual machines with full support of 64 000 tenants in a single data centre. Those tenants could be completely isolated and could therefore use independent IP addressing and utilize all VLAN assignments. By isolating the users, the amount of broadcast traffic is automatically reduced as well.

The paper [138] mentions other advantages of using SDN in a data centre: better monitoring using advanced tools like *sFlow* and a more effective troubleshooting using the push event notification model. The book [136] adds two more advantages: a more effective rapid deployment (via flexible adding, relocating, and removing of components), and predictable recovery times after a failure (unlike in traditional data centres, where this time is non-deterministic).

The majority of research in data centres targets performance. Temporal data congestions, resulting in increased latency and reduced QoS, can be limited, or completely mitigated using *Scalable Congestion Control Protocol* deployed in [139]. This protocol uses OpenFlow with a slight extension to control bursty flows (by adding three new action types).

Load-balancing, based on congestion, was further described in [137], where authors used a number of flows on a link. In their work, they were also able to achieve average failure recovery time of 100 ms (with injected monitoring packets). A different load-balancing method, based on large flows, was also described in [138]. Paper [140] confirmed, that by SDN deployment in a typical data centre using L2 topologies, performance can be increased by up to 45%. This increase is made possible by using intelligent decisions to fully utilize redundant links, common in these networks.

On the other hand, approaches replacing the traditional three layer design by more advanced topologies also exist. The paper [141] compares the traditional three layer design with the fabric topology. The fabric topology achieved a slightly higher total bandwidth, but it supported up to 13 times more hosts.

SDN Deployments in Cloud Environments

One of the most recent areas of research concerning SDN deployments in clouds, especially with connection to the IoT, is edge / fog computing (FC). One of the FC definitions is "*... a highly virtualized platform that provides compute, storage, and networking services between end devices and traditional Cloud Computing Data Centres*" [142]. The concept of moving processing resources closer to the end devices (IoT sensors) brings the following advantages: lower latency, support for mobility, real-time interaction, and on-line analytic. All these features can be achieved much more easily than in the traditional cloud, where the distance between end devices and the cloud is much longer and the network performance is therefore more dependent on the underlying infrastructure and its current state.

The paper [133] mentions SDN related architecture to support FC. In this architecture, a new logical element called *road side unit controller* (RSUC) is added. The RSUC connects several road side units (RSUs - devices deployed alongside roads and responsible for communication with vehicles) and is managed by the centralized controller. RSUCs store local road system information and perform selected services, so they are considered to be fog devices.

An even more recent subtopic of FC is mobile edge computing (MEC). According to [143], MEC provides: "*an IT service environment and cloud-computing capabilities at the edge of the mobile network, within the Radio Access Network (RAN) and in close proximity to mobile subscribers*". According to the same source, MEC is one of the key technologies for 5G networks - together with NFV and SDN. The paper [144] tries to fuse these three key technologies in order to enable the IoT wide deployment. The paper uses modified architecture proposed in [145], where the network layer contains *Software-Defined Fog-Enabled Gateways*. These gateways are responsible for merging different communication technologies of end devices and for providing fog computing functionalities. The final addition is the usage of distributed controllers for achieving high scalability.

4. Thesis Objectives

This section describes main objectives and contribution of the thesis. Firstly, the main aim of the thesis is defined via an analytical approach.

4.1 Finding SDN Research Gap - Security

SDN itself is, despite its relatively young age, a well researched topic. There are many published papers regarding SDN (already described in chapter 3) and also master's [23,146,147] and dissertation theses [148,149]. However, as described above and also in [6,117,118,125], some areas of SDN - mostly security - need more attention and have a higher potential for future research than others.

The security aspect of general modern networks is unfortunately an extremely wide topic. One of the most promising areas on which to focus the SDN research are IoT networks. But even this specialization does not narrow the scope much. As was already described in section 2.4, IoT can cover many application domains, which contain many different devices. For these reasons, it is necessary to narrow the scope even further.

4.2 Narrowing Research Scope

So far, the main identified thesis topic connects three areas: SDN, security, and IoT. The next step analyses IoT application domains for their applicability to utilise SDN.

4.2.1 SDN Applicability in IoT Application Domains

IoT application domains were described in detail in section 2.4. The following part lists these domains and discuss the possible SDN utilization in every domain. The summary of the SDN applicability is shown in Table 4.1.

- **Aerospace and aviation** - networks for aircraft parts tracking are scattered around the globe and therefore present a very difficult scenario in which to implement an SDN solution. On the other hand, networks for people and cargo identification are part of a LAN (typically an airport), but they have special requirements, mostly on security. Typically, in these environments, all deployed technologies must be verified by a security institute and certified for compliance with the given requirements (aviation). For this reason, deployment of every new technology is a slow and difficult process. In the SDN deployment, it would require development of a specific solution, which could not be used anywhere else. These factors make the deployment difficult and expensive.

- **Agriculture** - cattle monitoring is typically bound to a LAN, which uses a special radio transmission (non-IP). In this scenario, traditional SDN utilizing OpenFlow cannot be used. In a case of multiple geographically separated places, data transmission will most probably be conducted via an ISP network, where SDN cannot be used.

- **Automotive systems** - data networks inside cars must be optimized for special conditions, which a moving vehicle produces. This presents unique requirements on latency, reliability, and resiliency. A special hardware has to be used as well. It must be as small and light as possible, it should consume only necessary power, and at the same time, be maximally reliable and cheap. Deployment of SDN in such conditions is not feasible.

- **Business environments** - from the SDN perspective, business environments are part of SBs, which can be integrated into a SC infrastructure.
- **Education** - have the same applicability as business environments.
- **Environment monitoring** - sensors for environment monitoring have to be connected via a MAN, most often integrated within a SC network. In this case, it is an ideal domain for SDN deployment.
- **Health and aged care** - these domains typically cover only closed LANs, or utilize an encrypted connection into a cloud (via a VPN). SDN cannot be therefore effectively deployed in such scenarios.
- **Industry** - most often, it uses a private LAN with restricted network access, making the SDN deployment not suitable.
- **Information providing** - QR readers and surveillance cameras (stationary or mobile) have to be connected to a network via gateways (GWs), to provide information into a centralized service. This connection can be integrated with a MAN of a SC.
- **Insurance** - dynamic data from sensors placed on insured objects (cars, houses), or data from smartphone applications is typically transferred via different ISP networks. Therefore, SDN deployment is not feasible.
- **Product life cycle management** - product tracking uses various technologies with different network types scattered around the world. These networks typically use several different ISPs. An unified implementation of SDN is therefore not possible.
- **Smart retail, logistics, and supply chain** - logistics (product life cycle management) is typically scattered around the world and therefore not feasible for SDN. Smart retail and supply chains can be integrated as a part of a SC, if they cover data from local shops and use nearby warehouses. In this case, SDN can be used as a part of a SC.
- **Smart buildings** - they can be a part of a SC, where SDN can be applied in a MAN.
- **Transportation** - infrastructure for smart transportation is a part of a SC and therefore applicable for SDN deployment.
- **Waste management** - IoT sensors for waste management are implemented within a SC's MAN and therefore applicable for SDN deployment.

There are domains, where SDN cannot be utilized at all. This includes domains, which use non-IP networks, where OpenFlow is not supported. SDN also cannot be used in large scale networks, which use infrastructure of an ISP(s). From the domain owner perspective, this part of the network is not under his / her management and SDN cannot be therefore used in the entire network.

The second category are domains, where SDN can be deployed, but with a very limited scope. This includes the *aerospace, aviation, and industry* domains. Although, SDN can be used, the application will be strictly focused only to applicability in these

Table 4.1: SDN Applicability in IoT Domains

Domain	**Applicability**	**Note**
Aerospace and aviation	Limited	Specific solutions only
Agriculture	No	Non-IP / ISP's networks
Automotive systems	No	Specific requirements
Business environments	Yes	SC (smart buildings)
Education	Yes	SC (smart buildings)
Environment monitoring	Yes	SC
Health and aged care	No	LAN only / VPN
Industry	Limited	Specific solutions only
Information providing	Yes	SC
Insurance	No	ISP's networks
Product life cycle manag.	No	Scattered networks
Smart retail / supply chain	Yes (local)	SC
Smart buildings	Yes	SC
Smart transportation	Yes	SC (smart mobility)
Waste management	Yes	SC

Data source: own analysis

domains. This is caused by specific requirements of this area. An SDN application deployment would therefore be very costly, not re-usable and potential advantages might not be worth the effort.

The last category are domains, where an SDN solution can be deployed, and where its deployment is effective. All of these domains utilize networks with a larger geographical scope: MANs and WANs. From the SDN perspective, deployment in these networks makes more sense, as the solution can be more robust and can cover a larger amount of traffic and various demands. Moreover, all of these domains can be a part of a SC.

4.2.2 Smart City as Area of IoT

All the identified domains, convenient for an SDN application deployment, were identified as potential parts of a SC. A SC network is typically a MAN type, managed by a single administration. This is an ideal platform for an SDN deployment. If multiple SCs use the same management system, with the same architecture, they can be operated within the same administration domain. In this case, their network type would be WAN, and SDN could be used to cover all of these SCs.

4.2.3 Aim of Thesis

The SDN applicability analysis revealed, that the most ideal area of IoT for SDN deployment, with security in mind, are SCs. The thesis therefore focuses primarily this area and covers the following domains of a SC: smart buildings, smart grid, smart homes, and smart mobility.

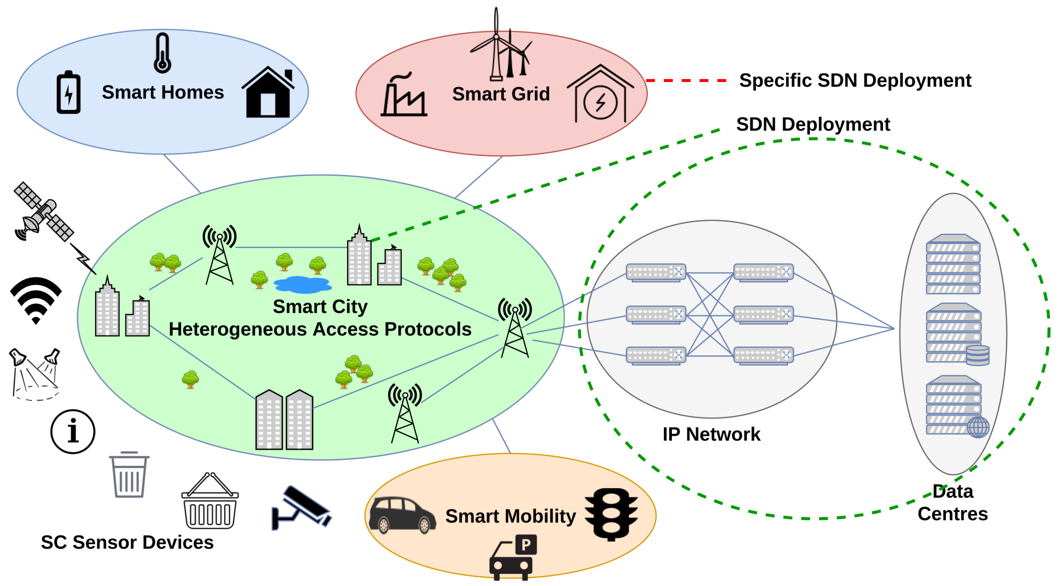


Figure 4.1: Deployment Scope of SDN in SCs

SDN Deployment Scope in SCs

It is important to mention, that SDN within a SC can be deployed only in the IP part of the network. The possible deployment scope is shown in Figure 4.1. The main part of a SC consists of different sensor devices, which use heterogeneous access protocols (these protocols are described in section 5.2). Data from sensor devices are received by gateways and encapsulated into a common IP frame, which can then be matched by OpenFlow. SDN can therefore manage aggregated data from all SC domains - SG, SHs, SM, and from various general sensors and nodes. SDN can be also interconnected with NFV functions used in data centres of the SC. This could further extend the functionality of these networks.

SDN can be further deployed within the SG part of a SC. In this case, SDN would probably be used inside transmission and distribution substations. These networks have the most strict requirements on latency, reliability, security, and privacy. It is therefore very unlikely, that the SDN used in this domain would be integrated with the SDN of the SC.

4.3 Thesis Motivation

SCs are nowadays becoming very popular and their use is expected to grow. Improving their security is therefore important. Moreover, new security legislatives and laws (such as *General Data Protection Regulation* - GDPR [150]), targeted on privacy protection of citizens data, are being implemented. To comply with these regulations, security of new technologies will have to be on an adequate level. The main advantage of SDN - programmability - allows SC networks to gradually evolve, in order to cover these changing demands. Once SDN is implemented in the network, introduction of new legislatives does not pose an existential threat to the network, as it can be updated just by a software modification. This is contrary to traditional networking, where such a change could require to buy, configure, and deploy a completely new networking infrastructure.

4.3.1 Traditional Networks Security Approach in SCs

SC networks based on traditional networking technologies use *legacy security protection methods* combined with *modern techniques of protection* like cloud and virtualization.

Legacy protection methods use combinations of multiple dedicated hardware boxes (IDSs, IPSs, and FWs) and software tools (ACLs on any ISR) to achieve a complex protection. These independent systems offer highly specialized functions. They can typically perform only a very limited scope of functionalities, but on the other hand, they can perform them very efficiently. The efficiency is supported by HW-accelerated circuits (HW boxes) and by highly specialized programming (SW tools).

Modern security techniques utilize mainly **cloud principles**. The main advantages of this principle are seamless updates and instantaneous hardware scalability. These are especially important criteria for security applications. These applications often need to keep a threat database as up-to-date as possible. Other advantages are connected with reliability. Cloud uses a redundant hardware, which can be seamlessly swapped even during operation. It also allows utilization of advanced backup solutions. Cloud can be also more effective against availability attacks like DoS. During these attacks, utilized hardware can be dynamically scaled according to the load. All these functionalities can be effectively managed from a centralized system. Additionally, most of these functions can be executed automatically, which significantly decreases any potential application downtime.

The second modern concept is **network virtualization** (NFV). This can be tightly connected with SDN, but it can be also utilized as a standalone solution. Virtualized network functions allow dynamical implementation of new security features. This can include access control, antiviruses, encryption solutions, IDSs / IPSs (Intrusion Detection / Prevention Systems), etc. Most importantly, NFV allows a logical network segmentation, which can significantly increase the network security.

Protection techniques of traditional networks can be classified into three main layers: *end devices* (PCs, laptops, tablets, smartphones, printers, all-in-one devices, etc.), *network access* layer (typically the first networking devices, providing connectivity to end devices), and *network distribution and core* layers (the rest of the network).

End Devices Security

- **Antivirus and antimalware solutions** - they are software programs deployed on end devices. The main purpose of these tools is to analyse files in use and to detect any potential threats. This includes files originating from the network, as well as local files, and files on removable devices. It is the main protection against internal threats, which are already within the network boundary.

- **End-point data loss prevention** - these software applications are deployed on workstations, where they monitor all outgoing data. If the data is sensitive and should not leave the workstation boundary, the communication is blocked. This solution therefore prevents against sensitive data leaks during the entire life cycle of the data. These applications can also control the workstation access to other services of the company - printers, servers, etc.

Network Access Layer Security

- **Access control** - this hardware, or software system allows definition of security policies to limit users' access to certain services, or parts of the network. It also provides a single point awareness of users' behaviour. Most often, the system controls the access layer of the network (wired and wireless links), but it can be also used on VPN connection endpoints, or on any other tunnelling technologies. The main purpose of the system is to contain any threats within the affected area.

- **Mobile devices security** - this software application allows management of mobile devices like smartphones, tablets, and laptops. It can support various OSs like Android, iOS, Chrome OS, OS X, Linux, Windows (including Windows Phone and Windows S), etc. It is used for monitoring and managing of mobile devices and it has remote functions for localization, locking, and erasing of a device. It can also enforce set security policies (for example used wireless network encryption) and limit access to a certain content (applications in stores, web services, games, etc.). These systems are becoming more common as the number of mobile devices increases and popularity of concepts like "*bring your own device*" rises [151].
- **Wireless security** - traditional wireless security protection methods (encryption, MAC filter, hidden SSID, etc.) can be accompanied by an additional system implemented in an AP, or as a dedicated module, which can be connected to an AP. The system can typically provide intrusion prevention, connected devices' localization, and monitoring of radio spectrum for rogue APs.

Network Distribution and Core Layer Security

- **Behavioural analytics** - is a tool, which analyses traffic patterns and network data and learns typical behaviour of the network. If there is a significant deviation from these parameters, the tool can quickly identify a potential threat and gives notification about it.
- **Firewalls (FWs)** - are systems, which monitor incoming and outgoing traffic and they act as the first line of defence. They can be SW-based or HW-based. Typically, SW-based FWs are used mostly on end devices. These FWs can be a part of an OS. Unfortunately, in a SC area, most of the end devices have limited hardware performance and minimal OSs, which make the implementation of these FWs impossible. The HW-based FWs are deployed at network boundaries, where they can monitor most of the traffic. They can be also integrated as a software within an ISR (Integrated Services Router).

  There are many types of FWs, from the most simple ones, which can monitor only L4 and lower (stateful inspection) to the next-generation FWs, which can scan the application layer, and can integrate the IPS functionality.
- **IDS, IPS** - both of these types of devices scan network traffic for threats, but they have different areas of deployment. IDS is typically deployed as an end node and is connected to the network via a SPAN (Switched Port Analyzer) port, which mirrors all the traffic. If it detects a threat, it gives a notification about it, but it cannot take any pre-emptive action. IPS, on the other hand, is typically deployed as a gateway device on the network boundary. It analyses traffic online and if it detects a threat, it can stop it immediately.
- **Network data loss prevention** - similarly to end-point data loss prevention, the goal of this system is to prevent sensitive data leaks, but at the network boundary. These systems can be implemented as a software service running in a GW, or running in a dedicated box.

- **SIEM (Security Information and Event Management)** - these tools collect information about network and therefore allow identification of a threat more quickly and respond to it more effectively. SIEM as a software can be implemented in a dedicated hardware box, or on a server. Collected data about nodes in a SC includes: device ID, IP addresses, location, authentication status, group, device information (MAC address, connection method, manufacturer, model) and software information (firmware version, antivirus information, running services).
- **VPN** - provides communication encryption, most often on public links. It can provide secure connectivity via virtual tunnels, between two remote networks, or between an end device and a network. In SC scenarios, connectivity between two networks is more common, as end devices often do not have a sufficient hardware performance to support resource-demanding VPNs.
- **Web security** - is a software tool for protecting publicly available websites. It can run on servers, networking devices, or on virtual machines. In a SC, these tools can protect SC websites, mobile applications, and APIs.

4.3.2 SDN Security Approach in SCs

The main advantage of an SDN-based application for securing IoT networks is, that such a solution can be integrated within a single system. The complete solution can therefore run on a single hardware device - a server with SDN controller (although to achieve high reliability, this device is typically redundant or virtualized). When compared to traditional network approaches, where multiple software and hardware solutions have to be integrated into the network, the single device greatly reduces the network complexity and cost, and makes the management easier. If SDN is already deployed in the network, integration of new security features depends only on the application. No changes to hardware, or networking infrastructure are required. This, together with other mentioned benefits of SDN (programmability, innovation, vendor independence) makes the deployment highly effective.

SDN can provide security at *access, distribution*, and *core* network layers. The only condition is, that OpenFlow-enabled networking devices are located at these layers. SDN can then perform most of the security functions like traditional networking. This includes: access control, behavioural analytics, firewall, IDS, IPS, network data loss prevention, and VPN. Other security features of traditional networking need specific applications (mobile devices security, SIEM), or a specialized hardware (wireless security). Nevertheless, even these solutions can be integrated with the SDN network.

The only layer, which is not covered by SDN is the *end devices* layer. Standard protections at this layer use software tools (antivirus, antimalware, and end-point data loss prevention systems) installed on end devices. However, these tools can be used as standalone systems even in SDN.

4.4 Thesis Goals and Contribution

Based on the performed analysis, this section defines the thesis goals and contribution.

4.4.1 Thesis Goals

The main goal of the thesis is to propose a blueprint for implementation of security improving SDN applications, into SC networks. This blueprint is based on an extensive analysis and some of its key concepts are verified in a use case application. Together, these areas form the three parts of the thesis.

1. Analysis of Smart Cities

The first goal of the thesis is to perform an analysis of SCs. This section firstly covers the general concept of SCs - including their architecture and commonly used communication protocols. Following the introduction, security analysis is conducted. Detailed steps of the analysis include:

- SCs general architecture and architectures of SC domains.
- Commonly used SC communication protocols.
- Security analysis of SCs and relevant issues.

2. The Blueprint

The main goal of the thesis is to propose a blueprint for implementation of SDN applications into SC networks. The blueprint is based on the performed analysis from the first goal, so it reflects up-to-date findings from the area.

The blueprint proposes a development process for SDN-based SC applications with focus mostly on security. It gives a step-by-step guide, with the most important sections described in detail. Some of the key assumptions of the blueprint are then verified on hardware devices. The blueprint section includes:

- The application development process with security life cycle.
- List of application requirements and specific security threats protection.
- Practical verification of several assumptions of the blueprint.

3. Use Case Application

The last goal of the thesis is to verify the proposed blueprint on a use case application. The application covers selected parts of a SC and integrates several mechanisms from the blueprint. In the last part, the functionality of key features of the application is verified. The use case application section includes:

- Definition of a use case topology of a fictious smart city.
- Description of the application.
- Model of the application, including architecture and UML diagrams.
- Implementation of the application, including description of functionalities.
- Verification of key parts of the application.

4.4.2 Thesis Contribution

To my very best knowledge, nowadays, there is no exhaustive blueprint for deployment of SDN applications into a SC domain. As the SC concept is more commonly used, its applications are becoming a part of a citizen's life. These applications should therefore be secure, effective, and reliable. A properly designed blueprint can help to achieve exactly that, if it is implemented correctly.

The main contribution of the thesis is therefore in proposing such a blueprint. The blueprint defines step-by-step process for the application development with emphasis on security life cycle, application requirements, and recommended protection procedures against specific attacks. The blueprint can be used mainly for applications focused on security, but it can cover applications targeting other areas of functionalities as well.

5. Analysis of Smart Cities

This section has several parts which cover the analyses of Smart Cities. It starts with a description of SC architecture, including all related domains. Following, the most commonly used communication protocols are analysed. The third part analyses security issues related to SCs and their domains. The last section explores requirements for SC applications, which does not have to utilize SDN. Any related issues, which might negatively impact deployment of an SDN-based application are described as well.

5.1 Smart Cities Architecture

Current SCs are built independently without a clearly defined architecture. For this reason, no fixed architecture is presented in this work. The only common principle is two-method approach to the SC backbone network. This network can contain *dedicated* links for every type of the SC domain (SG, SH, SM). Alternatively, traffic can be *integrated* within a single (logical) link. The similar concept was introduced in SG networks [152]. These two methods have the following properties:

- **Dedicated links** - each application domain has its own links. This will guarantee a dedicated link capacity for every domain and maximum security. On the other hand, the deployment is more expensive, requires multiple ports, has a higher power consumption, and complicates the wiring (multiple cables).
- **Integrated link** - a single physical link is shared amongst traffic from all SC domains. This method saves money and electrical energy, but requires an additional configuration. When properly configured, this method is more efficient in link utilization, but there is a risk of interferences between applications from different domains.

Regardless of the selected method, the final architecture is always composed from interconnected individual architectures of each domain. These individual architectures are described in the following sections.

5.1.1 Smart Buildings Architecture

SBs can integrate all types of electrical equipment located in the building. Most of these devices would require an interface to connect into an IP network. Devices can then communicate with a management system - they can collect data, be controlled from the system, or do both. Collected data can then be analysed and the building can perform an intelligent behaviour. These procedures can optimize resource utilizations of typical buildings up to 25% [153]. A typical SBs architecture includes four layers with the specified systems [153]:

1. **Edge devices** - it includes nodes, sensors, and other connected devices such as: an intelligent lighting (typically controlled from management applications, or by movement detection sensors), elevator and escalator systems, energy meters (data can be used for detailed analysis and more efficient resources usage), fire and alarm detectors (they can be integrated with the management system for faster response and better efficiency), and security systems (they control access to specific parts of the building and monitor and log security events).

2. **IP network** - provides connectivity between edge devices and a middleware platform.

3. **A middleware platform** - uses an *information management platform*, which provides an abstraction to all edge devices and provides APIs to overlaying applications.

4. **Overlaying applications** - they can control specific parts, or the entire SB. The examples of these applications are: infrastructure monitoring, management applications, mobile management applications (with a limited scope of functionalities - they can allow users to control climate, environment, and access specified areas of the building), and smart metering applications (precise information about energy consumption from smart energy meters).

Smart Building Network Requirements

SB LANs use the Ethernet standard, built on top of fiber optics, or twisted-pair copper cables. For the user convenience, the network also often provides a Wi-Fi connection. SB networks typically integrate data traffic of the building's users with traffic of all edge devices.

The standard *ISA95*, described in [154], defines a five layer architecture for integration of control systems in enterprise networks. These layers separate traffic from office LANs, control processes, monitoring applications, sensors, and other edge devices. This provides better security and performance, and it can be achieved by implementation of QoS.

A SB network is either connected to an ISP, or straight to the SC network. According to [155], such a connection can be realized via LTE. A more common approach is to use a wired technology (Metro Ethernet, cable, xDSL), which offers better performance and reliability. Utilization of these networks can vary significantly during a day. It mostly depends on working and non-working hours, but also on other specific conditions and dates. For this reason, it is necessary to design these networks with such a behaviour in mind. The network should be over-dimensioned to handle these traffic peaks.

5.1.2 Smart Grids Architecture

The conceptual architecture of SG is shown in Figure 5.1. This architecture defines the following five main building domains [156]:

1. **Generation** - represents large-scale power generation utilities. This typically includes power plants (nuclear, coal, gas, hydro), large photovoltaic systems, and wind farms. These utilities are connected to a transmission system and controlled by a generation management system.

2. **Transmission** - is an infrastructure for transporting generated electricity over long distances.

3. **Distribution** - is a grid network connected to the transmission grid via substations. It transports electricity to the customers.

4. **Distributed Electrical Resources (DERs)** - are small-scale power generation units (solar panels, wind turbines). The generated power of these units is typically between 3 - 10 000 kW [156]. These units are connected directly to the public distribution grid.

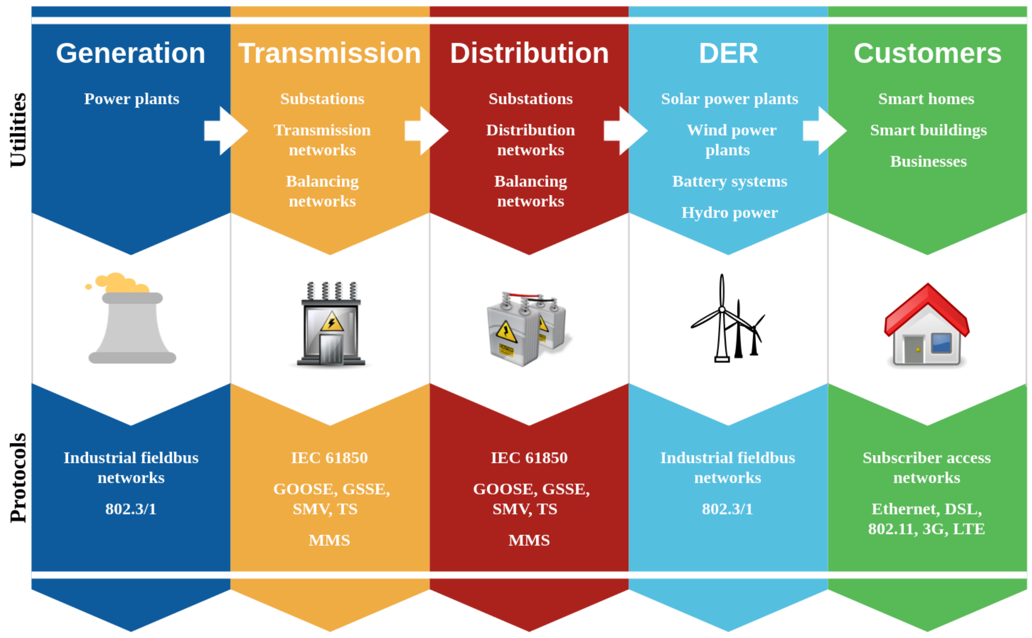


Figure 5.1: Smart Grid Conceptual Architecture

5. **Customer premises** - this includes end users of electricity - homes, offices, industrial premises, and commercial establishments. Users typically consume the energy, but they can also generate their own electricity and give it back to the grid. Customer premises might contain small scale generation units including solar cells, wind turbines, and EVs.

SG also defines a so called *microgrid* architecture, which includes only *distribution, DER*, and *customer premises*. The main goal of a microgrid is to make power generation and consumption as local as possible. This eliminates the need for a transmission system and therefore noticeably improves reliability and reduces the distribution losses.

Smart Grid Data Centres

A SG data centre is the main point for management of the entire grid network. It is connected to substations, field devices, electric systems, DERs, market systems, and back offices. For redundancy and more efficient management, multiple interconnected data centres are typically used. The most important system of a data centre is SCADA (Supervisory Control And Data Acquisition). This system monitors, collects, process, and logs real-time data from the entire grid, including data from substations. It allows the control of specific devices. SG data centres also use traditional management applications like ERP (Enterprise Resource Planning) systems, management systems (for meter data, energy, and distribution), and information systems (customer and geographic). A more detailed description of these systems and other ones used in IEC standards can be found in [63, 88, 157].

Smart Grid Substations

Modern SG substations are built according to IEC 61850 standard [158], as all-digital systems, and are therefore the most important component of SG for the potential SC integration. If this domain has the same administration, it can be integrated with the SC system, provided, that sufficient security requirements will be fulfilled. A SG substation is typically composed from the following components:

- **Ethernet switches** - they are in most cases represented by traditional industrial L2 switches with support of QoS, advanced redundant link protection protocols (RSTP, TRILL), and VLANs. These devices are now becoming replaced by L3 switches. Newer devices can utilize advanced routing protocols, which can achieve faster convergence times during the network change, or an equipment failure. If SDN is expected to be implemented into the SG substation network, these devices must support OpenFlow protocol.
- **Intelligent Electronic Devices (IEDs)** - these devices are responsible for monitoring and control of a grid equipment. They typically sample phase voltage signals and current signals. Depending on an event severity, IEDs transmit data to centralized applications, or they can process them locally, if an immediate action is required. A critical component of every IED is a GPS receiver. This receiver provides a precise time synchronization [159].
- **Merging Units (MUs)** - they collect and process analogue data from the process layer. Data is then digitized and sent via a *process bus* (which uses an Ethernet connection) to a control application [160, 161].
- **Non-conventional instrument transformers** - these devices have analogue sensors and they generate low voltage signals, which are sent to MUs [161].

Smart Grid Network Requirements

Networks in SG substations use a switched Ethernet. As [160] states, the main advantages of implementation of Ethernet networks into substations are lower costs. These costs include: installation (by utilizing a single, converged network), equipment (lower number of devices), maintenance (easier configuration), equipment migration (different brand of devices are compatible), extension (easy extendibility), and integration (an unified network with already deployed networking devices). Another advantage is flexibility of Ethernet, which can support implementation of new networking features. Additional benefits can be achieved by implementation of SDN, which is built on top of an existing Ethernet network. This can include additional expandability, programmability, improved security, easier manageability, and faster configuration. [162]

A typically used network topology is *ring*, which achieves redundancy via two independent paths. Nowadays, these links are most often realized via 1 / 10 Gbps fibre optic links (as they are immune to electrical interferences present in substations) [160].

SG substation networks have strict requirements mostly on low latency (which can be achieved by QoS features) and on high availability. High availability can be delivered by use of the traditional RSTP. This protocol can provide network convergence time in hundreds of milliseconds during a network change. The modern approach to high availability is to use fast L3 routing protocols (or L3-like protocols as TRILL) in combination with redundancy protocols like Hot Standby Router Protocol (HSRP), Virtual Router Redundancy Protocol (VRRP), and Gateway Load Balancing Protocol (GLBP).

Switches provide connectivity to IEDs, MUs, substation servers, and to external networks. A substation network must also provide a *time synchronization* for all devices, which require it. Critical components of a substation might be redundant (as is typical for some IEDs). Specific details of the substation network architecture can be found in the IEC 61850 standard [158]. This standard defines the following basic requirements on substation communications:

- Auto-configurable and flexible configuration support.

Table 5.1: Smart Grid Networks Latency Requirements

Communication Type*	**Maximum Latency***	
IED action required, command msgs.	3/10 ms or 20/100 ms	
Medium transmission speed required	100 ms	
Slow speed control functions	500 ms	
Continuous IED communication	3/10 ms	
Large files transfer	1000 ms	
Communication Type**	**Inside / Outside Substation**	
Information protection	4 ms	8-12 ms
Monitoring / control	16 ms	1 s
Operations / maintenance	1 s	10 s
Text transfer	2 s	10 s
Audio / video transfer	1 s	1 s
Other files transfer	>10 s	>30 s

Data sources: *[152], **[163]

- Guaranteed latency times for certain traffic types - listed in Table 5.1.
- High availability supported by redundant links.
- High speed communication (especially between IEDs).
- Multi-vendor interoperability.
- Network support through the utility enterprise.
- Security and based on standards.
- Support for sampled data (voltage, current) and transfer of large files.

5.1.3 Smart Homes Architecture

There is no standardized architecture, but generally, the main part of a SH is a Smart Home Energy Management System (SHEMS / HEMS) [84]. This system is integrated with a SG via an Energy Services Interface (ESI) [164]. SHEMS is composed from the following parts (multiple names are sometimes stated, as these systems are often called differently in various sources):

- **Energy Management Center (EMC)** [165], or **Home energy gateway** [166] - is responsible for SH monitoring, logging, control, management, and alarm functions.
- **Energy Management Panel (EMP)** - is an interactive interface, most often in a form of a real-time display. It allows users to control all of the EMC functionalities over a single device. Most typically, this device utilizes a touch-screen with standard control options.
- **Smart meters** - they measure energy consumption data and use a two-way communication with EMC and SG. This communication also contains information about the current energy tariff.

- **SH devices** - they can be represented by almost any device, which is able to connect to the SH network. They can be classified into the following categories [84]: manageable (their operation can be interrupted and usage time planned), unmanageable (they should not be interrupted, but their operation time can be planned), non-interruptible (they cannot be interrupted and must run in defined time periods), and others (air conditioning, ESSs, and EVs).

Smart Home Network Requirements

A typical network of a SH uses a single access device, which combines functionality of an AP, switch, router, firewall, and modem. Connectivity within a SH is realized via wired (Ethernet) and wireless technologies (802.11 standard family). According to [167], it was predicted that the most common Wi-Fi standard in 2018 would be 802.11ac. At the end of the year 2017, this standard was already supported by 83% of sold APs.

The 802.11ac standard has sufficient throughput and latency for all current applications used within a SH network. 802.11ac works in 5 GHz band and it utilizes MIMO (Multiple-Input Multiple-Output) and wider channel bandwidth to achieve data rates over 1 Gbps. Depending on the configuration, an AP's theoretical maximal data rate can reach up to 6.9 Gbps (with 8 spatial streams, 160 MHz channel bandwidth, and 256-QAM). [168] It can transfer demanding applications such as video in high resolution (4K), video-conference, and online gaming. At the same time, it can still provide a reliable connectivity for smart meters and other SH's sensors. These devices can be connected via older standards (802.11n or 802.11g) and use the more common 2.4 GHz band. This might be required, because these devices often do not support the most recent standards (as they can be older and optimized for low cost). The 2.4 GHz band offers slower data rates, but due to a longer wavelength, it has better coverage and range. Most of the current APs support simultaneous operation of both 2.4 GHz and 5 GHz standards via dual band operation.

The biggest drawback of the wireless technology are interferences. This can be especially evident in the 2.4 GHz band, which is also used by many non Wi-Fi devices (wireless devices like keyboards, mice, headphones, or baby sitters). Unfortunately, even appliances like microwaves can generate a noise signal in the same band. This noise can completely jam the legitimate Wi-Fi signal for a short time period. Such a temporal unavailability might not be an important issue for normal applications, but can cause problems to devices, which require high availability. Other sources of interferences are neighbouring wireless networks, which use the same band. Although 802.11g and 802.11n (in 2.4 GHz band) define up to 13 different channels, these channels do overlap. As a result, the maximum number of non-overlapping channels is 3 (if the basic 20 MHz channel width is used). In typical cities, where residential areas are composed from multi-storey buildings with lots of flats and offices, the number of overlapping wireless networks can easily reach dozens or more. Three non-overlapping channels are clearly not enough (especially if some APs use a 40 MHz channel width). Resulting interferences then cause transmission errors and significantly lower the network performance. In these scenarios, it might be necessary to use a different transmission technology for critical traffic types.

5.1.4 Smart Mobility Architecture

The two most important devices used in SM are RSUs (Road Side Units) and OBUs (Onboard Units). RSU is a stationary device, which communicates with OBUs and acts as a gateway to the traditional network. OBU is a unit located inside a vehicle.

It communicates with other OBUs and with RSUs [169]. Additionally, SM uses the following supporting technologies:

- **Big data analysis** - sensors from connected infrastructure gather a large amount of data, which must be analysed using advanced processing tools. Only these tools can process the data with sufficient speed and efficiency.
- **Connected infrastructure** - covers communication of vehicles, infrastructure, sensors, and transport system (collectively referred to as V2X). For V2V communication, a dedicated short-range technologies (in 5.8 - 5.9 GHz bands), or radio networks are used [103]. V2I communication uses e-Call, Wi-Fi, or mobile networks. Infrastructure sensors can be installed in street lamps, traffic signs, traffic lights, and gates.
- **Digital payments** - this can include payments for road tolls, parking, and transport fares. These payments can be done by a specialized hardware such as e-tags, RFID readers, and licence place readers (in combination with pre-paid, or post-paid accounts). Another option, suitable mainly for public transport, is payment on the move with digital accounts. Such a payment can be executed by smart mobile devices.
- **Mobile IoT networks** - are important for certain sensor communication. Their new generations (5G) will support a large number of parallel connections, high bandwidth, extremely low latency, and very low power consumption.

Architecture of SM communication is composed from the two main components:

- **Wireless Access in Vehicular Environments (WAVE)** - is defined by [169] and it operates on L3 - L7. It uses IPv6 on L3 as the only protocol.
- **Dedicated Short-Range Communications (DSRC)** - it operates on L1 - L2 and its most well-known protocol is IEEE 802.11p.

Smart Mobility Network Requirements

Networks for SM must comply with specific requirements of this type of communication. They include reliability, interferences resiliency, low latency, security, and minimal guaranteed transmission speed. According to [170], IEEE specified maximum latency limit in vehicular networks to be within 50 - 100 ms (depending on the specific usage area). SM uses two distinct communication networks, which have different requirements:

- **V2V** - is a safety-related communication, which requires low latency and high reliability. The maximum latency must be under 100 ms (50 ms for pre-crash sensing), with the minimum frequency of transmitted messages being 10 Hz. The communication includes: vehicle status warning (emergency electronic brake lights, abnormal condition), vehicle type warning (emergency vehicle, slow vehicle, motorcycle, vulnerable road user), traffic hazard warning (wrong way driving, stationary vehicle, traffic condition, signal violation, roadwork), and dynamic vehicle warning (overtaking vehicle, lane change assistance, pre-crash sensing, co-operative glare reduction).
- **V2I/I2V** - is mostly a non safety-related communication with not so strict requirements. This type has the maximum latency specified to 500 ms (except for optimal speed advisory for traffic lights and intersection management - which

require latency below 100 ms) and minimum frequency of messages being 2 Hz. The communication includes: traffic management (speed limits, optimal speed advisory for traffic lights, intersection management, co-operative flexible lane change, electronic toll collection) and infotainment (point of interest notification, local electronic commerce, media download, map download and update).

Details about these communications can be found in [171]. Furthermore, [172] specifies coverage and typical data rate requirements for the following three application types:

- **Active road safety applications** - this area is critical for road safety and has the most strict requirements. Latency must be under 100 ms and data rates between 1 - 10 Kbps with the coverage from 300 m to 20 km.
- **Cooperative traffic efficiency** - data from this area only increases productivity and efficiency, so the delivery is not critical. Latency should stay under 200 ms and expected data rates are 1 - 10s of Kbps with the coverage range from 300 m to 5 km.
- **Infotainment** - this area only increases user comfort by providing Internet access and additional information. Latency can be up to 500 ms, but expected data rates can be in a range of 10s - 100s of Kbps with unlimited coverage (over cellular networks).

While most of these requirements do not appear to be too strict, it is necessary to consider variable environment, weather conditions, and radio frequency congestion. A large number of devices and traffic sensors, in combination with periodic messages, can present a significant amount of data traffic. The paper [171] gives an example of 30 cars in a limited area: in peak conditions, these cars produce up to 750 Kbps of traffic (with 256 bytes long messages).

5.2 Analysis of SC Communication Protocols

This section analyses communication protocols used in SCs on data link layer (L2), application layer (L7), and security layer.

5.2.1 Smart City L2 Protocols

SC networks use mostly IP as a L3 protocol for communication [166]. IP acts as an abstraction layer for the underlying protocols. This allows use of various L2 protocols in two main categories - wired and wireless. This section describes the following most commonly used protocols in both of these categories.

Wired protocols [166]: Ethernet, G.hn, HomePlug, Insteon, and X10.

Wireless protocols [166, 173]: 6LoWPAN, Bluetooth, Cellular networks (CSD, GPRS, UMTS, LTE), DASH7, EnOcean, LoRa, ONE-NET, SigFox, Weightless, Wi-Fi, WiMAX, Zigbee, and Z-Wave.

Ethernet

The most common L2 protocol is Ethernet, defined by the IEEE 802.3 standard [174]. The protocol is widely used in today's LANs. The standard supports a large number of different variants for copper and fibre links with various speeds (from 10 Mbps up to 100 Gbps). Nowadays, the most common variant has a speed of 1 Gbps and range of up to 100 meters per segment.

A widespread deployment of Ethernet makes this technology cheap, well supported, and easy to deploy. For these reasons it is an ideal protocol for SHs, SGs, and SC backbone networks.

G.hn (ITU)

G.hn is the *home networking* technology, which utilizes various types of wires - telephone lines, coaxial cables, and power lines. It can achieve data rates up to 1 Gbps and can be easily integrated with IPv4 and IPv6 networks. The technology uses Orthogonal Frequency-Division Multiplexing (OFDM) modulation with a parity check code and forward error correction. Additionally, it uses a 128-bit Advanced Encryption Standard (AES). This encryption is used for confidentiality and message integrity. G.hn can also provide authentication and key exchange via X.1035 [175].

The most typical use of G.hn are SH and SG deployments. A G.hn network in a SH can complement a Wi-Fi network and is suitable for static devices (IPTV), or devices which require an additional security (NAS). G.hn is an ideal communication type for AMIs (Advanced Metering Infrastructure) used in SG networks as it utilizes the existing infrastructure.

HomePlug

HomePlug is a protocol, which uses power lines for communication. Depending on the quality of lines and HomePlug specification, it can achieve data rates from 14 Mbps to 500 Mbps with a range of about 300 meters. The protocol is also relatively well adopted and supported.

This technology is especially convenient for SG networks, where power lines are already in place and additional networking can therefore be avoided. On the other hand, it has two main disadvantages: interferences and range only within the same power circuit (so it cannot be deployed within a SG substation). Moreover, HomePlug does not support any encryption, so it is not suitable for critical networks. If security is required, a secure higher layer protocol (TLS) should be used.

Insteon

Insteon is a protocol developed specifically for IoT networks and it can utilize both wired (power lines) and wireless (RF communication) networks. It supports low-performance devices (which can have only 80 bytes of RAM - light switches, smart lamps, etc.) and it has an error correction to improve reliability. The wired version can reach up to 3 km, but it has low data rates - maximally 13 Kbps. The wireless version uses 902 - 924 MHz frequency bands and can reach up to 45 meters, if there is a direct line-of-sight.

Insteon adoption rate is moderate and recently, it has started to support modern devices like Amazon Echo and Apple's HomeKit (including Apple Watch). From these devices a SH can be controlled via the Insteon Hub.

X10

X10 is a protocol specifically designed for home automation and most of the time it uses power lines (but it can also use RF). The protocol achieves relatively low data rates - maximally 60 Kbps with range of up to 300 meters. This makes it ideal only for non-demanding applications like lights control.

Similarly to other power line technologies, X10 suffers from interferences, and information losses, which highly depend on the line quality and amount of message collisions (it does not use CSMA/CD or CSMA/CA). Other limitations of X10 are: only basic functionality (standard defines only a limited number of required control messages, while additional messages are optional), slow speeds, and lack of encryption.

6LoWPAN

6LoWPAN is a wireless protocol, which allows transmission of IPv6 packets over low-power and low bandwidth networks, such as 802.15.4. This is possible by IPv6 header compression, which reduces the header size from 60 bytes to 7 bytes [166]. The protocol can achieve speeds of up to 40 Kbps (in the 915 MHz band), or up to 250 Kbps (in the 2.4 GHz) within a 75 meters range [166]. The protocol is relatively well supported by the industry.

The technology is ideal for SHs (lighting, thermostats, and smart meters) and devices running on a limited battery power. The protocol can run on devices with 8-bit CPUs and with only 32 KB of flash memory. The protocol also supports an optional security mode, which uses the 128-bit AES encryption [176].

Bluetooth

Bluetooth is a transmission technology for WPANs (Wireless Personal Area Networks) and it is defined by the IEEE 802.15.1 standard [177]. Bluetooth has currently five main versions, which are not fully compatible (only backward-compatible) and have many differences. Bluetooth specification also defines two main forms: *basic rate* and *low energy*. The basic rate can achieve speeds up to 2.1 Mbps (54 Mbps if 802.11 mode is enabled), while low energy typically achieves only 125 Kbps. Both of these forms work in the unlicensed 2.4 GHz band and unlike most of the wireless technologies in this band, they use frequency-hopping spread spectrum (FHHS) modulation. This modulation is more resilient against narrow band interferences (because it hops between 79 different frequencies). [178]

Bluetooth has three topology options [179]:

- **Point-to-point** - it supports both forms of Bluetooth and it is ideal for device-to-device communication (speakers, in-car systems, fitness devices, or PC peripherals).
- **Broadcast** - is a one-to-many type of communication and is available only in low energy form. This topology is ideal for beacons in SC deployments (points-of-interest, item-finding, and way-finding).
- **Mesh** - is many-to-many communication and it is also available only in the low energy form. Mesh is ideal for large-scale networks like SCs, SG, SBs, WSNs, and asset tracking.

Bluetooth is an ideal technology for connecting SG nodes within a SH [180].

Cellular networks

CSD (Circuit Switched Data) is the original transmission technology of GSM (Global System for Mobile Communication). The main difference between CSD and more recent technologies is, that CSD creates a virtual circuit for every connection. The circuit guarantees a default transmission capacity of 33.8 Kbps. The transmission uses encryption and reliability mechanisms, so the maximum usable data transfer speed is only between 1.44 - 9.6 Kbps and it further depends on signal quality [173]. This technology was superseded by GPRS and is not being used any longer.

GPRS (General Packet Radio Service) is a packet oriented transmission technology of GSM. Unlike CSD, GPRS uses the *best effort delivery* (which is the typical technology used in most modern data network protocols). The transmission is more effective than CSD, because QoS is not guaranteed (but latency and throughput can therefore vary). GPRS can theoretically achieve speeds up to 114 Kbps, but in a single slot it typically reaches only 21.4 Kbps [173]. Latency of GPRS communication is about 150 ms [181]. GPRS was superseded by EDGE (Enhanced Data Rates for GSM Evolution) and UMTS.

UMTS (Universal Mobile Communication System) is the third generation GSM standard for mobile communication and it is maintained by [182]. UMTS is not compatible with GPRS as it uses different frequencies and requires new base stations (and therefore also explicit support from end devices). UMTS includes several transmission protocols with various data transfer rates. The most recent HSPA+ (Evolved High Speed Packet Access) can reach up to 42 Mbps with latency of 50 ms, while HSDPA (High Speed Downlink Packet Access) can achieve 7.2 Mbps with 100 ms latency [181].

LTE (Long-Term Evolution) is the UMTS successor. It uses several frequency bands in a range 0.7 - 2.7 GHz with bandwidth between 1.4 - 20 MHz. The theoretical maximum speed of LTE, when a single transport channel is used, is 75 Mbps [183]. Latency of LTE was also greatly reduced, when compared to previous GSM generations, and is typically around 20 ms [181].

LTE is well suited for non-safety related communication and for communication, which does not require minimal latency - typically V2I communication. In some extreme scenarios, where a large number of end nodes must communicate, special technologies may be necessary. The example is eMBMS (evolved Multimedia Broadcast/Multicast Service) coming in *LTE-A Cat. 9.* This technology is optimized for point-to-multipoint communication, but unfortunately it is designed only for static communications (concerts, theatres, sport stadiums). It does not specify details about handovers between different mobile cells, or between service providers. Additionally, the LTE modems are not certified for usage in safety and critical scenarios, such as use in Advanced Driving Assistance System (ADAS) - this system can, for example, provide information about current allowed maximum speed [171].

DASH7

DASH7 is a complete networking stack defined on L2 - L6. It fully supports bi-directional communication, mesh architectures, and multi-hop links. A single gateway can connect to thousands of devices, while it achieves very low latencies, when compared to technologies like LoRa (described below). It also utilizes LAN features like indoor localization with 1 meter precision and supports OTA (Over The Air) updates, roaming, and strong security via the AES 128-bit encryption. [184]

DASH7 supports two classes of communication with maximum data rate of 200 Kbps [185] and uses the 433 MHz band [186]. It has a range of 5 km and devices can be typically powered for year from a single battery.

EnOcean

EnOcean is a transmission technology, which utilizes an electro-dynamic energy converter to harvest energy from motion, light, and difference in temperature. EnOcean uses ultra-low power radio (EnOcean standard ISO /IEC 14543-3-10 [187]), but it is also prepared for support of IEEE 802.15.4. The EnOcean standard uses sub 1 GHz bands (the specific band depends on the country of use) and achieves data rates of up to 125 Kbps, while it typically uses only 50 µW for a single message transfer. The communication distance can be up to 30 meters inside a building and up to 300 meters in a direct line- of-sight. The technology also includes authentication via a unique 32-bit ID and uses the 128-bit AES encryption. [188]

The technology is suitable for ultra-low power devices and devices with a problematic connection to traditional power sources. These problems can include sensors placed over large areas (forests, gardens, water areas), sensors monitoring a structure integrity (buildings, dams, bridges, tunnels), and various SC devices. A possible usage within a SH is control of a lighting system via buttons powered by kinetic energy [188]. Nevertheless, EnOcean is not a widespread technology [166].

LoRa

LoRa is the physical layer technology for low power WAN (LPWAN) long-range communications. LoRa operates in 867 - 869 MHz frequency in 10 different bands. LoRa has a maximum range of about 5 km in urban areas and up to 15 km in open space [189]. It supports 8 different data rates in a range of 250 bps - 50 Kbps. The used data rate can be configured on a GW and it is a trade-off between transmission speed, range, and power consumption. [190]

The maximum supported packet size is 250 bytes and security is supported by the 128-bit AES encryption with key exchange. The LoRa architecture uses several keys: *application key, network session key*, and *application session key*. For end nodes identification, IEEE EUI64 ID is used.[191]

LoRaWAN is networking stack, which is build on top of the LoRa physical layer specification and it is defined mostly on L2 with some elements from L3. Except the LoRa, LoRaWAN can also use different physical layer specifications like frequency shift keying (FSK). LoRaWAN specifies used network topology as the *star-of-stars*, where nodes are connected to a GW. The GW is typically connected to a wired network, where a centralized server is located. Nodes communicate with the GW via a single wireless hop [189].

LoRaWAN also defines three main operation classes [190]:

- **A (receiver initiated)** - a sensor can transmit data at any time. The transmission is detected by all GWs in the range. A GW can respond only in two fixed slots (typically after 1 and 2 seconds). This mode is ideal for nodes running on a limited battery power. A typical life-expectancy of the battery in this mode is 10 years.
- **B (coordinated listening)** - it adds pre-programmed slots for communication from GWs to nodes. This mode is ideal, when communication to nodes is much more frequent.
- **C (continuous listening)** - nodes and GWs can transmit at any time. This mode is ideal only for devices with unlimited power supply as their radios are active all the time.

LoRa has a wide variety of possible usage in SCs, but only for certain applications. Due to the regulation limitations, *time on air* of every node is very limited - in Europe it is maximally 1%. This makes LoRa ideal only for applications with low frequency of messages. Moreover, relatively low transmission speeds predetermine LoRa for use with non-real time applications. Ideal use cases are therefore smart lighting, smart parking, smart waste, and weather sensors. [189]

Although LoRa is becoming widely used, there are concerns about its applicability in modern SCs [184]. The most notable ones are: incompletely defined networking stack, mainly one-way communication model, slow performance (high latency, low data rates), insufficient security, and no support of OTA updates. These flaws can significantly limit LoRa's use in modern SC architectures, where especially strong security and ability of OTA updates are a necessity. Despite these flaws, LoRa is currently deployed in more than 250 cities worldwide and has over 40 operators [192].

ONE-NET

ONE-NET is an open-source wireless transmission protocol for home automation, available under the BSD licence. Its implementation, written in C language, can be downloaded from the official website[1]. ONE-NET can achieve speeds in a range of 38.4 - 230 Kbps and reach up to 70 metres indoors and 500 meters outdoors [166]. Because the protocol is open-source, it can be deployed on various off-the-shelf microcontrollers and radios, but despite this, it is still not a widely deployed technology.

Sigfox

Sigfox [193] is a unique technology, because it is the first global network for connecting IoT devices. Sigfox's coverage is managed by regional network operators. Sigfox is currently used in 26 countries and it is planning to expand to 60 countries by the end of 2018 [192].

Sigfox uses *ultra narrow band* frequency of 200 kHz for long-range and low power transmission. A typical node's battery lifetime can be up to 20 years and transmission can reach up to 50 km (in rural environments), or 10 km (in urban environments). On the other hand, data transfer rates are very slow - between 100 and 600 bps. A message payload size is also very limited - maximally 12 bytes for uplink and 8 bytes for downlink. [193]

Sigfox uses the *star topology*, where a node message is broadcasted and captured by every base station in range. To improve the message resiliency against noise, the message can be transmitted multiple times in different frequencies. There are 400 channels in total, while the default number of transmissions is 3. Nowadays, Sigfox supports two-way communication, but downlink is very limited as the legislation allows sending only of 4 messages per day per device. [194]

Very low data rates and small message sizes limit Sigfox to be used only in very specific applications. Within SCs, it can be environmental sensors, which are scattered across large areas and their data payload sizes are small. These sensors can be used for detection of earthquakes, forest fires, floods, etc. However, the lack of encryption in Sigfox messages must be considered.

Weightless

Weightless is a set of three standards for LPWAN IoT communication. It is managed by the *Weightless Special Interest Group* and it is optimized for low throughput and not latency sensitive applications. Currently, it is defined for specific unlicensed and licensed bands in a range of 138 MHz - 923 MHz [195]. Weightless standards include:

[1]Available: `https://sourceforge.net/projects/one-net/`

- **Weightless-N** - is the most energy efficient version. It supports data rates of up to 100 Kbps with a range of up to 3 km in urban areas. Payload of the messages can have 20 bytes. Unlike other Weightless standards, this version supports only one-way communication [194].
- **Weightless-W** - uses TV white spaces in the frequency range between 470 - 790 MHz. It supports data rates of up to 10 Mbps and can reach up to 5 km in urban areas. The minimal defined data length is 10 bytes. The problem of Weightless-W is in used frequency bands, which are available only in specific regions [194].
- **Weightless-P** - uses two-way communication and two non-proprietary physical layer standards. It supports data rates of up to 100 Kbps with a range of up to 2 km in urban areas [194]. It also supports OTA updates, message acknowledgements and three types of message delivery: unicast, multicast, and broadcast [196].

All the Weightless standards support 128-bit AES and use the *star topology*. Weightless-P also optionally supports AES with 256-bit encryption. [194]

Wi-Fi

Wi-Fi is currently the most popular and widespread wireless technology for LANs. It is based on the IEEE standard 802.11 [197], which contains over 10 different physical layer protocols. Wi-Fi can operate in various frequency bands, most often 2.4 and 5 GHz, although the most recent versions use even higher frequencies (up to 60 GHz) to achieve faster speeds. In the most widespread version (802.11n), Wi-Fi can achieve up to 300 Mbps and reach up to 100 meters (in the 2.4 GHz frequency band) [166].

Wi-Fi is nowadays used in almost all devices with a connectivity ability. This includes laptops, smartphones, tablets, smart watches, smart TVs, game consoles, fitness devices, and even most of the SH equipment. Wi-Fi can use WPA2 encryption (Wi-Fi Protected Access 2), which uses AES, and which is even nowadays considered relatively secured. Older encryption technologies like WPA, or WEP (Wired Equivalent Privacy) contain found vulnerabilities and are therefore not recommended to be used. Unlike the previous technologies, Wi-Fi is not focused on low-performance devices and it is therefore the most versatile technology for connecting all-purpose devices. The disadvantage of Wi-Fi is paradoxically in its widespread use. Because it is such a common technology, used frequency bands are often over-utilized and the transmission suffers from interferences. This is especially true in city environments and 2.4 GHz frequency bands.

WiMAX

WiMAX (Worldwide Interoperability for Microwave Access) is a wireless communication protocol based on IEEE 802.16 standard [198]. WiMAX acts as the *last mile* technology and it is therefore a wireless alternative to digital subscriber line (DSL), or cable modem connections. WiMAX is classified as a WMAN (Wireless Metropolitan Area Network) with range of communication up to 50 km. WiMAX uses different frequency bands in range of 2 - 11 GHz and can reach theoretical speeds of up to 70 Mbps [180]. Such speeds are, however, not achievable in a real world usage. This was tested in an experimental measurement [199], where authors achieved average speeds of 4.8 Mbps with varying distance between 0.5 - 20 km.

WiMAX can be used for AMIs, where it can utilize its advantages like long range, fast transmission speeds, interferences resiliency, and security. On the other hand, the most problematic issue of WiMAX is propagation of high frequency signals (over 10 GHz). Such a signal is not able to penetrate most obstacles and can be therefore used only in direct line-of-sight applications. This might be a problem for AMIs. Another issue is the required dedicated hardware equipment, as WiMAX uses different frequencies than most of the common wireless protocols. [180]

Zigbee

Zigbee is the complete IoT solution for meshed networking. It provides the application layer (libraries) for interoperability between various specific devices, and network protocol stack (Zigbee PRO 2015) for connectivity. Its radio layer uses IEEE 802.15.4-2011 [200] in the 2.4 GHz frequency band. It can reach speeds of up to 250 Kbps with a range of 100 meters indoors, and 300 meters in the line-of-sight [201]. It uses the 128-bit AES encryption on the network layer with an optional encryption on the application layer [202].

Zigbee is a unique technology for its mesh architecture. It supports meshed networks containing up to 65 000 inter-connected devices and the communication is optimized for low-power consumption [201]. The Zigbee devices can operate up to 10 years using only batteries [166]. Zigbee is a widespread technology and it supports a wide range of SH devices from various domains (lighting, HVAC, alarm and security systems, green power devices, and others). Zigbee is defined as one of the standards for SG residential networks (within a SH) by NIST and was also identified as the most suitable communication technology for this deployment [203].

Z-Wave

Z-Wave is another meshed network technology, but unlike Zigbee, it uses the sub-1 GHz frequency band. The lower frequency implies slower speeds, but it makes the Z-Wave more power efficient and less prone to interferences. It can reach speeds between 40 - 100 Kbps with range of up to 35 meters. Every Z-Wave network can contain up to 232 devices. [204]

Z-Wave is widely used, especially in low-performance devices such as light switches, HVAC appliances, security sensors, and remote controls [166].

5.2.2 SC Domain Specific L2 Protocols

Additional L2 protocols are deployed in specific domains of SCs. These protocols will be only briefly described in the following sections.

Smart Grid Substation L2 Protocols

The typical SG substation contains the following protocols, which are defined in the IEC 61850 standard [157, 158, 160]:

- **Abstract Common Services Interface (ACSI)** - is a service, which maps several protocols into a general abstracted format. These protocols are typically MMS (Manufacturing Message Specification) and FMS (Field bus Message Specification). The IEC 61850 standard defines abstract services, which must be supported [205].

Table 5.2: SG Substation Communication Traffic Patterns

Traffic type	**Frequency**	**Msg. size**	**Data rate**
SV	4800 Hz	126 B	4.6 Mbps
Control signals	10 Hz	200 B	15.6 Kbps
File transfer	1 Hz	300 KB	2.3 Mbps
Status signals	20 Hz	200 B	31.3 Kbps
GOOSE messages	Variable	200 B	Variable

Data source: [162]

- **Generic Object Oriented Substation Event (GOOSE)** - is a connectionless protocol for messages originating from IEDs. The messages are mapped directly into Ethernet frames and therefore do not use MMS [160].
- **Generic Substation Status Event (GSSE)** - is a similar protocol to GOOSE, but it has a simpler message format for faster delivery. It is also mapped directly to the Ethernet frame to make the process more efficient. GSSE uses IEEE 802.22 [206] and it is typically utilized for transfer of real-time data between substations and control centres [207].
- **MMS** - is a standard [208], which interconnects SG devices (IEDs, PLCs - Programmable Logic Controllers, etc.) with their control systems. It does so by mapping objects of the IEC 61850 standard. Each MMS object has its own unique reference composed from a string of values. These values typically include: logical device name, logical node name, functional constraints, data, and attributes [160].
- **SMV (Sampled Measured Value)** - are messages used on the process layer of a SG network. Like GOOSE, SMV is also designed for critical messages with low delay requirements (they are therefore mapped directly into Ethernet frames). SMV messages are typically sent via the same network as GOOSE messages - in this case, at least a Gigabit Ethernet connectivity is recommended due to a possible high traffic load [209].
- **Time synchronization** - is critical for certain parts of SG. Most often, Simple Network Time Protocol (SNTP) [210] is used. This protocol uses UDP as the L4 protocol and therefore does not utilize MMS mapping. Version 4 supports both IPv4 and IPv6. SNTP can achieve time accuracies between 1 - 50 ms [210]. This is typically sufficient, if the protocol is combined with GPS. If a higher precision is required, Precision Time Protocol (PTP) [211] can be deployed. This protocol is designed for LANs and can achieve nanoseconds precision. This might be needed for Phasor Measurement Units (PMUs), which have the most strict requirements on time synchronization [152].

The paper [162] summarizes typical traffic patterns within SG substation networks. These values are shown in Table 5.2. Stated data rates are calculated as *frequency * message size* and apply for a single device using the specified traffic type.

Smart Mobility L2 Protocols

Two main SM communication networks (V2V and V2I) were described in section 5.1.4. V2I uses LTE communication, which was already described in section 5.2.1. V2V is then classified as DSRC and has strict safety-related requirements. Core protocols for DSRC are: IEEE 802.11p (L1 - L2[2]), IEEE 1609.3 (L3 - L4), and IEEE 1609.4 (L2) [212]. The most used protocol is IEEE 802.11p.

IEEE 802.11p is the communication protocol for V2X, but it is mostly used in V2V applications. It operates in the 5.9 GHz frequency and uses the 75 MHz bandwidth. Use of 5.9 GHz frequency band offers higher transmission speeds, but as the study [213] verified, the maximum transmission range is significantly lower than in sub-1 GHz bands. Effective range with 90% reliability in a SC environment is 150 meters for 802.11p, while sub-1 GHz band can reach up to 250 meters. Also, the 802.11p is more susceptible to line-of-sight obstructions. This makes it less convenient for complicated SC scenarios. The protocol was approved in 2009 and it is therefore well tested and widely-deployed. 802.11p is the most suitable protocol for V2X applications, unlike cellular networks. According to [171], even LTE is not suitable for low latency and high mobility use cases of V2V. Compared to cellular networks, typical latencies of 802.11p, in real-world scenarios with speeds of vehicles between 30 and 170 km/h, are around 1.5 ms [214]. On the other hand, there are experimental studies [172], which verified, that even 802.11p latency can be insufficient in SC situations with high traffic density. This was confirmed in [215], where stated latency of this protocol was 60 ms.

IEEE 802.11p supports the following data rates: 6, 12, and 24 Mbps (optionally also 9, 18, 36, 48, and 54 Mbps) [216]. Such transmission speeds are however unrealistic in real usage. A simulation under a real-world conditions [217], tested various parameters of 802.11p features and come to the following results (classified by area of deployment):

- **Cities** - maximum throughput 220 Kbps, delivery ratio 80%.
- **Highways** - maximum throughput 230 Kbps, delivery ratio 78%.
- **Rural areas** - maximum throughput 125 Kbps, delivery ratio 95%.

It is expected, that the 802.11p networks will stay the dominant technology for V2V communication, while future 5G networks can become the major technology for non-safety related V2I communication [171].

5.2.3 Smart City L7 Protocols

The application layer protocols (L7) have to be considered only if a deep packet inspection is required (data cannot be matched by OpenFlow). In that case, an SDN controller can inspect the payload section of packets to verify, if the content is legitimate. However, if the payload is encrypted, the controller will be unable to perform this inspection. This section briefly describes the most commonly used L7 protocols within SCs. A summary of these protocols is displayed in Table 5.3.

- **AMQP (Advanced Message Queuing Protocol)** - is a binary protocol, which sends data in a stream of bytes. It supports publish-subscribe and point-to-point communication schemes. It uses TCP and therefore guarantees reliability and message ordering.
- **CoAP (Constrained Application Protocol)** - is a specialized lightweight protocol for resource-constrained devices and networks. The protocol functionality is similar to HTTP, but it is greatly simplified (it uses binary format and only necessary functions). The protocol is designed to be used on devices with small

[2]It uses the lower L2 sublayer - MAC.

sizes of RAM, ROM, and can be even used on low-power 8-bit microcontrollers. Its ideal deployment is on low throughput (Kbps) networks with a limited size of datagrams (6LoWPAN). It minimizes the need for fragmentation and it is therefore often used in M2M (Machine to Machine) applications. The protocol uses UDP, but optionally supports reliability, security, and multicast communication [218].

- **DDS (Data Distribution Service)** - is a machine-to-machine protocol designed to support real-time applications. It supports the publisher-subscriber model and focuses on minimal overhead and predictability. These features are achieved with QoS, which can be user-configured. DDS does not use an L4 protocol and therefore operates straight on the IP layer.

- **HTTP (Hyper Text Transfer Protocol)** - is a stateless request-reply protocol with the client-server architecture and it uses TCP (default port 80). It utilizes URL for resource identification and it is widely used on the Internet and also in IoT, M2M communication, and automation.

- **HTTPS (Secure)** - it adds encryption implemented over TLS (TCP port 443) to the standard HTTP. HTTPS is based on PKI (Public Key Infrastructure), which uses two types of keys - *public* (everyone can use it to encrypt a message) and *private* (only the valid receiver can decrypt the message). HTTPS over SSL should not be used, as it contains several security flaws [219, 220].

- **MQTT (Message Queue Telemetry Transport)** - uses the publish-subscribe architecture, is a lightweight and open-source, and is therefore ideal for IoT and M2M communications. The protocol's architecture includes publishers, subscribers, and a centralized server called the *message broker*. The main advantage of this architecture is the fact, that it can be deployed in networks, which contain a firewall. Sensor devices (publishers) establish communication only with the broker and not with (potentially hundreds of) applications - subscribers. Such a traffic pattern can be easily allowed in any firewall [54]. Messages can be delivered by three approaches: at most once, at least once, and exactly once. The protocol also supports an extensive security, conforming to the NIST Cyber Security Framework [221]. This includes lightweight cryptography (AES), authentication, encryption and privacy (TLS, VPNs), and detection of compromised clients and servers. [222]

- **UPnP (Universal Plug and Play)** - is an architecture composed from multiple protocols. These protocols provide: devices discovery (SSDP - Simple Service Discovery Protocol), service detection (GENA - General Event Notification Architecture), action execution (SOAP), and event reports. UPnP is based on an extended HTTP and it can therefore use both TCP and UDP. It also supports both unicast and multicast communications. [54]

  Security can be ensured by the SOAP (Simple Object Access Protocol) protocol - the UPnP is then referred to as UPnP Security. This protocol can then provide identification, integrity, authentication, and authorization [223].

- **XMPP (Extensible Messaging and Presence Protocol)** - it uses an intermediary server(s) for message delivery (as MQTT) and it can therefore span firewalls. It supports the publish-subscribe model, request-response, asynchronous messaging, event subscription, and delayed delivery [224]. This is achieved by its distributed client-server architecture. The protocol is ideal for environments,

Table 5.3: Summary of SC L7 Communication Protocols

Protocol	L4 Protocol	Communication	Encryption
AMQP	TCP	P/S, P2P	Optional
CoAP	UDP	R/R	Optional
DDS	- (IP only)	P/S	No
HTTP	TCP	R/R	No
HTTPS	TCP	R/R	Yes
MQTT	TCP	R/R, P/S	Optional
UPnP	TCP/UDP	P2P	Optional
XMPP	TCP	R/R, P/S	Optional

Note: R/R = Request/Response,
P/S = Publish/Subscribe, P2P = Peer to Peer
Data sources: [54, 218–220, 222–224, 226–228]

where a real-time delivery (which can benefit from optional QoS) and/or bidirectional communication is required [54]. This includes areas like V2I and M2M.

The protocol is very flexible. It uses XML for data encoding, but it can be replaced by EXI (Efficient XML Interchange), if the minimal message size is required. On the other hand, XML is easy to encode, decode, parse, reuse, and debug. Several extensions of the protocols also exist - they are called *XMPP Extension Protocols* and they are managed by *XMPP Standards Foundation*. Currently, there are nearly 200 of these extensions[3]. The transmission uses clients' authentication via passwords and usernames and it is conducted over SASL (Simple Authentication and Security Layer) or TLS. On top of the authentication, the protocol also supports confidentiality via TLS. [225]

5.2.4 Smart City Security Protocols

SC networks can also use security protocols, which can spread across several networking layers. The purpose of these protocols is to secure communications, provide authentication, and ensure data integrity. The most common security protocols used in SC scenarios are:

- **DTLS (Datagram Transport Layer Security)** - is a protocol designed to be similar to TLS in both the security and code reuse. The main difference between DTLS and TLS is, that DTLS can be deployed over UDP networks. The current version of DTLS is 1.2 and most of its specifications come from TLS 1.2. These two protocols have the same cipher mechanisms, except for the new *authenticated encryption with additional data cipher suites* and replay protection. DTLS also has a built-in protection against DoS attacks - specifically against multiple handshake initiations and server reply message amplification [229]. DTLS is an appropriate protocol for delay sensitive applications like voice, video, and SG substation data. It can be therefore effectively used in SC networks.

- **IPsec (Internet Protocol Security)** - is a network suite, which contains many security protocols and it provides authentication and encryption. Unlike TLS,

[3]The complete list of XMPP extensions can be found in: `https://xmpp.org/extensions/`

IPsec works on the L3. IPsec is most often used to provide a VPN connection between two networks, or between a network and a client (but a host to host connection can be used as well). The current version of the framework is IPsec-v3 and it is composed from two main protocols: ESP (Encapsulating Security Payload) and AH (Authentication Header). The core specification of IPsec and the list of additional RFCs (Request for Comments) describing the framework can be found in [230].

- **SRTP (Secure Real Time Protocol)** - adds encryption and message authentication to the RTP [231] and RTCP (control) protocols. These protocols can be deployed within a SC network, if VoIP or video conference are demanded. These protocols utilize UDP for the minimum latency delivery. It also supports both unicast and multicast communications, and it uses AES for encryption and SHA-1 (Secure Hash Algorithm) for message integrity. [232]
- **TLS** - is the most widely used protocol for securing network traffic. It creates a connection-oriented secured channel, but it must operate on top of TCP. The main disadvantage of TLS is, that it cannot secure unreliable traffic types (UDP). TLS uses symmetric cryptography via AES or RC4 encryptions, and checks data integrity via SHA-1. [233]

5.2.5 Summary of Analysis

The analysis confirmed, that a SC can use a large number of heterogeneous protocols. From the SDN-deployment perspective, low-level details of these protocols and their implementations are not important, if they are used only on the access layer. In this case, data from each protocol is re-encapsulated into the standard Ethernet datagram on the first receiving device (a GW). This datagram can then be normally processed by the OpenFlow protocol. On the other hand, it is important to consider various specifics of these protocols. Especially their power requirements, maximum transmission speeds, use of encryption, and expected times of node connection establishment (initial, and on a node wake-up). These specifications can influence the target (SDN) application and will therefore be described in more detail. Specific data from the described wireless protocols is summarized in Table 5.4.

Power requirements specify, if a device has a limited power source, or not. If it has, the communication must be kept to minimum and no redundant, or unnecessary messages should be transmitted. In this case, the SDN application might need to utilize a message caching - a message to a node can be transmitted only in specified time intervals.

Maximum transmission speed of a protocol indicates required bandwidth of a connection. The stated value is valid only for a single communicating device and total required bandwidth is therefore calculated as $n * bw$, where n is a number of end devices and bw is maximum transmission speed of a single device. This value can be used as a guideline for designing sufficient capacity for *distribution* and *core* layer links. It can also indicate requirements on networking devices and an SDN controller. In most cases, the calculated maximum bandwidth does not need to be achieved and a much lower value will be sufficient. In typical networks, not all the devices transmit at the maximum speed at the same time. Additionally, the controller can be configured for QoS, based on the calculated value and number of devices.

Encryption is important mainly for two reasons. Firstly, it increases the packet size. This might cause problems on links with a low MTU (Maximum Transmission Unit). In this case, packets need to be fragmented, which introduces an additional

Table 5.4: SC Wireless Communication Protocols

Protocol	P	Max. speed	Encryption	Node E/W
6LoWPAN	M	250 Kbps	AES (O)	> 5 ms*[1]
802.11p	L	54 Mbps	(A)[2]	> 1.5 ms*
Bluetooth	L	1 Mbps	AES, block cipher	3/3 s
EnOcean	L	125 Kbps	AES	in ms*
LoRa	L	50 Kbps	AES	Class dep.
Sigfox	L	0.1 - 0.6 Kbps	No	20 s[3]
Weightless	L	0.2 Kbps - 10 Mbps	AES	4-8 s[4]
Wi-Fi	M	300 Mbps	WPA2	2/1 s
WiMAX	H	50 Mbps	AES, 3DES, EAP	100/100 ms
Zigbee	L	250 Kbps	RC4	30/15 ms
Z-Wave	L	100 Kbps	AES (O)	1-2 s

P = power rating (Low, Medium, High), E/W = establishment/wake-up times
O = optional, A = via an addition, * minimal latency of the technology
1 = in the unsecured mode
2 = with IEEE 1609.2TM-2013 for WAVE networks, source: `https://www.standards.its.dot.gov/Standard/405`
3 = window size for downlink messages, source: `https://ask.sigfox.com/questions/2966/message-latency.html`
4 = source: `https://www.ubiik.com/lpwan-comparisons`
Other data sources: [166, 176, 178–180, 188, 190, 191, 193–195, 202, 204, 216]

overhead and complicates the application's functionality. Secondly, it makes the message payload unreadable to the controller. This becomes a problem, if the application layer inspection is required. The application can then use only data from unencrypted headers of the message.

Node connection establishment times are a good indicator for maximum network latency of any given communication. An SDN application should respect these values, and if required, use the proactive flow rule insertion method. Complying with these limits will ensure, that each node will be correctly registered by the network. Loss of just a few of the connection initiation packets could result in a total loss of connectivity between the node and the application.

5.3 Security Analysis of Smart Cities

Attacks on SCs can use the same types as general IoT attacks. Nowadays, most of the SC's solutions are implemented without too much emphasis on security. SC is a new concept and therefore it does not have sufficient and proven threat model, testing procedures, security recommendations, and established response teams [234]. Interestingly, so far, there have been only a few attacks on SCs, but that can change literally every moment as SCs are becoming more and more widespread.

This can correspond to a similar rise in attacks focused on **critical infrastructure**. In the U.S. alone, the number of these attacks rose from less than 200 in 2012 to almost 300 in 2015 [235]. One of the most recent attacks on SCs happened in Dallas on April 8, 2017. The city's siren system was hacked and emergency sirens, used mainly for severe weather warnings, went off during the night. This caused panic amongst many residents and overwhelmed the city's emergency 911 telephone lines. As a response to the attack, the system was shut-down for 48 hours [236]. While no serious incident or damage was reported, in case of an emergency, the warning system would be unavailable.

General Types of Security Threats

Security threats can be generally classified into the two main categories [88]:

- **Unintentional threats** - they are not caused purposefully by an attacker. Most often, they represent a naturally occurring event like equipment failure, safety system failure (improperly implemented redundant power supply), supporting infrastructure failure (power outage, connectivity loss), and climate control failure. Equipment failures can be also caused by natural disasters like floods, fires, earthquakes, etc. System threats can also be caused by an operator's carelessness (configuration mistakes, missing configuration of redundant and high availability technologies, insertion of an infected removable device, or use of default or too simple user credentials).
- **Intentional threats** - they are performed by an attacker with the purpose to cause damage, or steal data. These threats can include all types of attacks performed by unsatisfied / former employees, hackers, vandals, terrorists, and even malicious entities like viruses and worms.

It is important to mention, that most security solutions are targeting the *intentional threats*. For this reason, protection against these threats is typically at a high level and system operators are well aware of these threats. On the other hand, *unintentional threats* are often left without consideration and without a proper response plan. They can then have much more serious consequences. For this reason, the application must consider these threats right from the initial development phase. This will allow creation of a robust architecture, which can cope with analysed threats.

The following sections are focused mostly on intentional threats.

5.3.1 Attacks on SC Applications

Attacks can be classified into three categories: by types of attackers, by targeted elements of SCs, and by attacked data.

Types of Attackers

In general, a SC can be attacked from three different groups of attackers [237]:

- **Service providers** - this category includes ISPs, cloud providers, companies responsible for smart metering, providers of location-based services, and government responsible for the SC. These entities have access to various SC data collected from different systems. The most typical attack is to re-use this data for other purposes (theft or trade).
- **Involved parties** - it includes various system manufacturers and software developers, responsible for certain sub-systems of the SC. The examples of these sub-systems can be smart lighting solutions, SM devices, cameras, smart cards, etc. Manufacturers of these devices and software developers are authorized to access these devices. This access might be necessary to ensure future software updates, or battery replacements. On the other hand, potentially sensitive data of these devices can be stolen, or these devices might be compromised. There is also a risk of hidden backdoor access being implemented into the component.

- **External attackers** - this category includes all other persons, which might not have any connection with the SC infrastructure, and which do not have any initial access to the system. They can however misuse any potential weakness of the SC to gain unauthorized access, steal data, target availability, or cause any other damage.

Attackable Elements

IoT-A [58] defines elements of an IoT network, which need to be protected. The following list summarizes elements, which are applicable in a SC environment and adds SC-specific systems. They are sorted by their operational domain in SCs (from a data centre to users).

- **Backend services** - include all SC applications located in server farms. They include services for data collection and analysis, management systems, and systems providing APIs to operators and users. These systems must be protected against application layer DoS attacks and unauthorized access. If an unauthorized access is gained, the complete SC can be vulnerable as this centralized system can control all the components of a SC.

  These services can be also deployed in public cloud data centres. This means, that the devices are accessible to practically anyone. The service can be therefore much more susceptible to DoS and brute force attacks, and security would mostly depend on the cloud company's policies. Moreover, if the cloud is not a private one (owned by the same company as the entire SC), an integration with an SDN solution is often not possible.

- **Infrastructure services** - include all functions, which provide basic connectivity and advanced networking functions. This covers: device discovery, device lookup, DNS (Domain Name System), routing protocols, loop-prevention mechanisms, etc. These functionalities are critical for a SC communication and must be accordingly protected with tools like authorization, authentication, encryption, identity management, key management, and trust and reputation systems.

- **Communications channels** - this includes all physical and virtual links of a SC network. They should be protected by encryption to ensure data integrity and prevent eavesdropping, tampering, and replay attacks. Correct security of communication channels also prevents routing types of attacks (black hole, worm hole, depletion, etc.). A SC network cannot be considered to be a closed-network as it spans a wide area and utilizes various services, nodes, and connection technologies. Protection is therefore required and its level depends on used communication channels.

- **Intermediary devices** - this includes all networking devices like switches, routers, GWs, APs, security devices, etc. They are vulnerable especially to DoS and manipulation attacks. The protection starts with physical security and device hardening. Remote access to a device should use secured protocols like SSH (Secure Shell), with appropriate security policies (warning messages, secure passwords, limited number of login attempts, automatic logout on inactivity, logging, etc.).

- **End devices** - this includes sensors, actuators, tags, and all other end devices used in a SC network. These devices might have different requirements on protection, according to their capabilities and performed actions. In general, their integrity (SW and HW) should be ensured, as well as physical protection. Implementation of effective, but demanding protections might be problematic on low performance and low power devices.

End devices also include potentially dangerous physical systems. Examples are SG systems, which utilize extremely powerful, yet sensitive equipment, located in power plants, distribution grid networks (with very high voltage), and electric substations. An attack on these devices can therefore result in physical damage and also possible injuries, or even deaths.

- **SC users** - people using SC applications can be vulnerable to various threats and they therefore need protection as well. The level of this protection depends on the user role in the application. While a typical SC user is mostly a data consumer (gets information about traffic, weather, etc.), or uses non-critical systems (smart parking, public Internet connectivity), some of the users might require critical availability. This can include e-health applications, which are responsible for users' health monitoring and automatic responses to dangerous situations. Any issue could have fatal consequences. Another example is SM, which utilizes high-speed vehicles. If the system is compromised, it can pose a serious threat. Additionally, this category includes users' privacy, as a lot of used information might be sensitive.

  The most common communication platforms include mobile applications and social media. Mobile devices can be misused to gain unauthorized access to the SC network, act as DoS initiators, or become a spying device. Attackers can also use social media for spreading false information (for example to cause mass panic), or to gather sensitive data.

The SC domains can be attacked as well:

- **Smart grid networks** - the Ethernet protocol brings advantages in wide support, low cost, and worldwide standardization; but on the other hand, also negatively affects the security. Moving from proprietary and closed industrial protocols to a widely used Ethernet makes the network vulnerable to a much wider scale of attacks. In these modern networks, security must be taken more seriously than in the legacy approach, when *"security by obscurity"* applied [88].

  Complexity of a SG system is very similar to SCs and IoT systems - it also contains a lot of heterogeneous technologies, devices, and protocols. This makes implementation of a single security solution, which would solve the security for the entire system, impossible. This was proven during the famous attack in Ukraine in 2015, where *BlackEnergy* malware was used to gain access to control centres of the SG, resulting in a massive power outage, influencing 225 000 customers [238].

  The standard IEC 62351 [88] identified the most crucial threats of SGs as: critical availability, nodes with limited performance, specific links, PLC (Power Line Communication), and dangerous cyber-physical systems.

- **Smart homes** - are a specific area of SCs as they are typically managed by owners of the house. The most typical deployment of SHs uses a single networking device for creating a small LAN (AP combining wireless gateway, modem, switch, and router). This device can be owned by the house residents, or only leased from an ISP. If this device is compromised, the entire SH network, including all connected devices, is vulnerable. The easiest way to perform an attack is via an unsecured wireless network, or an unsecured remote connection with default user credentials.

- **Smart mobility** - contains mainly a large number of end nodes like sensors and traffic control devices. Protection of these devices mostly from accesss attacks is very difficult and must be done separately.

Attackable Data Sources

According to [237], an attack in a SC can be performed based on four different data sources:

- **Leaked data** - originally data, which should stay private, but was leaked and accessed by an attacker. The leak can happen via a software bug, security vulnerability, misuse, unauthorized access, theft, or social engineering. It is impossible to prevent all types of these attacks, but correctly implemented security policies can lower the chance of such attacks.

- **Observable data** - this includes mainly data, which was transferred over openly accessible links (wireless, or PLC). In these networks, an attacker needs only to be in the physical coverage area to freely access the data. If no encryption is used, complete messages can be read (including payload). An attacker can also access common wired networks (Ethernet), but this is much more difficult and it requires physical access to the cable. A special tool must then be attached to the cable - most often a passive *wiretap*. This tool then transmits duplicated cable signals to the attached device.

- **Published data** - the attack misuses publicly available data (traffic information, video footage, weather information, information about users, etc.). By data analysis and correlation techniques, an attacker can examine aggregated data to get information about individual users.

- **Repurposed data** - original data, which was gathered by legitimate means, is then used for other than intended purposes. This might be a usage, which is in contradiction with agreed terms. An example can be telephone numbers of users, misused for advertisement calls.

5.3.2 Privacy Issues

SCs can collect a large amount of sensitive information about citizens. Data from public cameras, mobile phone locations, and credit card usage can precisely track users and map their habits. Sensitive medical information can be also part of a SC, especially if fitness devices are used. This data can cover not only physical appearance information (smart scales can measure detailed physical values), but also healthy information (sleeping patterns, walking habits, and heart beat statistics). SHs also collect a significant amount of data about the living patterns of families - when they are at home and when the house is empty, at what time the family usually sleeps, and what devices they use. This data can improve quality of life, but it also presents a considerable privacy issue. A significant percentage of this data can be freely accessible and therefore easily misused by an attacker.

Proper protection techniques are therefore critical. One way is to anonymize data, but this is not always possible. It is clear, that this data must be collected and stored securely, especially when it is sent to a cloud. Moreover, it is important to consider the cloud location. If the data spans a country boundary, different laws and legislatives for data protection can apply.

Security vs. Privacy Problem

IoT-A [58] mentioned an interesting problem with security and privacy in IoT systems. One of the most important security features is *trust.* Trust allows an entity to prove, that it is really the one, it claims to be. On the other hand, forcing an entity to prove its identity brings a *privacy problem.* By providing its information, the entity has to reveal certain sensitive data. With this data, the entity might be traced. In this case, all users of the system can see, what actions the entity performed.

IoT-A [58] favours prioritizing *trust over privacy.* Authors in [58] recommended to use a signed trust-certificate for each entity, but require the verification only for interactions with sensitive data, resources, or services.

Trust Entity

Certain IoT applications can require a higher level of privacy, while achieving the same trust. In this case, the solution is to use a *trust entity.* This trustworthy system binds various pseudonyms of entities to root IDs. Each time an interaction has to be verified, a new unique trust value is generated. Unfortunately, this system has two main disadvantages and it is therefore not recommended by [58]:

- **System bottleneck** - a large number of end devices in most of the applications, and frequent trust validations would have too high demands on the verification system.
- **A single point of failure** - a single verification system could be targeted by DoS attacks, or even worst, compromised by an attacker. If this would happen, the attacker could reveal identity of all the entities, or freely manipulate with their identities (for example replacing fake ones with legitimate ones).

Note about SDN: the two mentioned disadvantages are valid ones, but they pose the same type of problem as is already present in every SDN deployment. An SDN controller is already a single point of failure and can potentially become a system bottleneck as well. Therefore by solving these problems in an SDN application, it can be solved for the trust verification system as well. The easiest approach is to use a distributed architecture of SDN controllers. Multiple controllers would provide an additional performance via load-balancing and the architecture would be more resilient against attacks. On the other hand, especially synchronization between controllers complicates the system design.

Data Communication and Privacy

Study [237] identified several potential weaknesses of data communications, which could cause privacy violations:

- **Wi-Fi hotspots** - they often use insufficient encryption, or no encryption at all (open systems), so the communication can be easily captured. The captured traffic can reveal visited sites, traffic type, and even login credentials (especially shared token keys). Tools for capturing this information are publicly available even for smartphones, so an attacker does not need any sophisticated knowledge. The protection is to use strong encryption (WPA2) and secured web protocols (SSL/TLS).

- **Metadata** - header information in network messages are typically not encrypted, but they can reveal sensitive information. They contain information about who communicates with whom, when, and for how long. There are techniques for anonymous communication, but they require usage of a specialized software.
- **Fingerprinting** - much information used in communication remains static (MAC address, used browser, OS's parameters) and can therefore identify a user. Protection methods include frequent randomization of these parameters, but this technique cannot be used in all cases.
- **Mobile devices** - mobile operating systems can be compromised by malicious hardware and software. Every mobile device is composed from various hardware modules, which can run their own software (OS / firmware). If this code contains a vulnerability, a data leak may occur. An operating system of mobile devices can be also vulnerable, especially if a malicious application is installed. The main issue is the sensitive nature of this data, as current mobile devices contain a lot of sensors (microphones, cameras, localization chips, motion sensors, magnetic field sensors, wireless chips, etc.), which can all be misused.

Implementation of Privacy Protections

The paper [237] describes a process of developing and operating applications for SCs, with privacy in mind. The process contains the following seven steps:

1. **Privacy by design** - it includes seven principles, which are unfortunately often too general and vague, to be effectively implemented. They are: proactive privacy protection (instead of reactive actions only after a violation), privacy as a default setting, privacy embedded into the design, full functionality with privacy protection, privacy protection during the entire data life cycle, visibility and transparency, and respect for users privacy.
2. **Privacy requirements engineering** - it is a process of applying privacy to the application in a systematic way. There are several methods of this process - the most famous ones being *PriS* [239] and *LINDDUN*[4]. The process can be achieved with the following eight principles:
   (a) **Data minimization** - SC systems are typically collecting redundant and unnecessary information. For example, most of the CCTVs footage is not useful. Mechanisms for limiting collection of this data should be implemented to lower a chance of unnecessary privacy leaks.
   (b) **Data anonymization** - data stored in databases can often identify the owner. Unless necessary, this identifying data should not be stored, or anonymization techniques should be used.
   (c) **Differential privacy** - is similar to data anonymization, but it adds a random data into the database. This way, every query returns similar results regardless if the queried data is in the database, or not.
   (d) **Encryption** - it ensures data confidentiality via *a shared-key*, or *public-private keys*. Alternatively, an *identity-based encryption* can be used. This type allows encryption of a message by a publicly known string (name, address, email) and therefore does not require a pre-generated public key.

[4]Available: `https://linddun.org/`

(e) **Homomorphic encryption** - is a modern technique, which allows processing of an encrypted data without a need to decrypt it first. This could be applied in SC applications, where certain entities can process sensitive data, but can still achieve privacy. The disadvantage of this type of encryption is high computational requirements.

(f) **Zero-knowledge proofs** - this method is typically used for authentication, where it allows user verification without a need to reveal any information (password, or login). This technique can be applied in smart metering, SM systems (tolls, parking), and automatic payments in public transports.

(g) **Secret sharing** - this technique allows share of information among several users. Each user has a *secret share* and a certain number of *shares* is needed to reconstruct the original message (some shares can be redundant to prevent a potential data loss). This technique can be used in a SC for data aggregation, or distributed data stores.

(h) **Anonymous / pseudonymous digital credentials** - these techniques allow users to prove facts about themselves, without revealing their identities. This principle is also called unlinkability, but legitimate authorities can de-anonymize users to ensure accountability. This can be used for communication with SC cloud providers. Another use case is blind signature, which allows an entity to sign a message without being able to read the content. This technique has a potential use in SM, where it can be confirmed, that the message comes from the legitimate vehicle.

(i) **Secure multi-party computation** - this cryptographic technique utilizes two or more entities, which simultaneously calculate a single public function with different data. The technique provides confidentiality and unlinkability, but does not reveal private data and does not need a trusted third party entity. On the other hand, computation is still very demanding and not feasible for most of the SC applications. It can, however, be used in healthcare and similar applications, where privacy is really important and the number of verifications is not too large.

(j) **Private information retrieval** - this technique allows querying of a database without revealing the query. It therefore provides confidentiality, unlinkability, and undetectability. The simplest implementation of this method is to reply with the entire database. But this method has an obvious disadvantage in high demand on communication infrastructure and end device processing.

3. **Testing and verification** - a correct implementation of privacy protection must be tested and verified like every other software. Testing should aim at information leaks and program inputs and outputs. Privacy properties of used protocols can also be formally verified.

4. **Transparency** - users should be informed, what data is collected, where is it stored, and for what purpose. These practices, together with an option to opt-out from collection, increase the acceptance level of the technology. An example of a bad transparency was the first version of *Windows 10*. The system telemetry collected sensitive data and sent it to Microsoft's servers without user consent. After two years and many complaints, Microsoft had finally started to inform users about collected data. The update (*Creators Update*) also gave the option to better control, what data is collected and reported [240].

5. **Consent and control** - gaining users' consent in SC applications is substantially different than in a traditional software. A SC does not include only software parts, but it is interconnected with physical systems as well. It is therefore impossible to transparently allow users to view, update, or delete data obtained from SC systems like waste monitors, CCTVs, or motion detectors. Some systems, on the other hand, can be set to be controlled from users' devices. In this case, users can decide, what data is collected, stored, and analysed, for example via a smartphone application.

6. **Auditing and accountability** - any SC system needs external auditing to prove, that the data privacy is functional and in place. Accounting is important from both users and the SC application perspective. It allows tracking of used SC services (public transport, tolls, energy), and also detecting potential data breaches.

7. **Privacy architectures** - they allow merging of different protection methods in an effective and functional way. They guarantee, that the used cryptographic solutions and other technologies are fully functional.

5.3.3 Security Threats

SCs are vulnerable from the threats in the following list. These threats were collected mainly from threats of classical IoT networks. They are ordered by the level to which an SDN can mitigate them - this level is stated in brackets. The threats also include brief descriptions of suggested implementation of non-SDN and SDN protections (where applicable).

- **Availability attacks - DoS (SDN fully)** - DoS, as the most typical availability attacks, is at the same time one of the most common attacks in today's networks. An attacker targets availability of certain applications, servers, devices, or the communication network. The goal of the attack is to make a service unavailable for legitimate users. The three main target areas within SCs are applications, end devices, and the communication network. **Applications** running on a server can be attacked by a massive amount of requests, which exhaust the server resources and make the service unavailable. This process includes opening TCP sessions, or uploading / downloading large files. **End devices** are often nodes with a very limited hardware performance (they are optimized for low-power consumption and long-life operation). To overload these devices is therefore very easy, even with common tools like *ping*. Both of these types of attacks can be accelerated by use of *zombie* devices, which are (originally legitimate) devices under control of the attacker. **The communication network** can be attacked by generating an extreme amount of data traffic. This traffic can overwhelm data links, or networking devices. Use of zombie devices can also scale the attack. The physical layer of communication networks can also be targeted, especially if the network uses a wireless technology, which is accessible by anyone located within the network's range. This type of attack can be performed by a *jamming device*, or by sending *deauthentication* packets.

  Availability of certain components of SCs might be critical. Examples are SG networks. Businesses, hospitals, offices, electric transportation vehicles, and people rely on stable and non-stop electricity distribution. Data networks, which control the SG, must therefore ensure the grid's continuous availability and optimal functionality. This is achieved by compliance with latency requirements, reserved bandwidth use, and utilization of redundant equipment and links. Performance parameters can be achieved by implementation of QoS, but correct behaviour during a network failure requires an intelligent control system composed from a routing protocol, loop protection mechanism, and ideally a link aggregation technique.

Protection against this type of attack is extremely difficult, especially if the physical layer is targeted. If a higher layer is targeted, dynamic ACLs (which can be implemented in SDN) can be used to mitigate the attack. SDN have advantage in a centralized view on the entire network. Redundant links can therefore be used efficiently and a switch-over process during the attack can happen quickly.

- **Broadcast tampering (SDN fully)** - an attacker can misuse broadcast messages to transmit false data. Unlike in other attacks, the attacker does not have to know a target's IP address, as he can use any broadcast address. Such an attack can result in an easy DoS, active network scanning, or distribution of malware.

  This attack can be typically prevented with authentication and use of digital certificates with active revocation of expired ones. A protection SDN mechanism can disable broadcast messages globally, or it can implement a specific forwarding of these messages.

- **Brute force (SDN fully)** - an attacker tries to illegally access a certain service, application, or device, by "randomly" trying login combinations. This typically includes a combination of *username* and *password*. In the simplest form, this attack tries all possible combinations, but this is a highly ineffective method. To guess a password combined from letters, numbers, and special characters can be infeasible within a reasonable time frame (providing that the password length is sufficient). To guess even the most common insecure passwords (owner's name, surname, or typical words like "admin", "user", "root", etc.) takes a significant number of tries. Detection of such an attack is therefore relatively easy, as the high amount of data traffic will be evident. Several techniques exist to make the attack more effective. The most common one is the *dictionary attack*, which uses a database of common passwords, words, numbers, and their combinations. These sequences are then used instead of trying random characters. To guess a typical password is then much more effective and attack might be harder to detect and stop.

  The attack is analogous to DoS - a large amount of requests is generated. The protection methods are therefore similar.

- **Cross-platform attacks (SDN fully)** - a heterogeneous architecture of SCs integrates many different IT platforms. Typically, there can be mobile applications, social networks, information portals, storage services, databases, video portals, etc. Vulnerability of a single system can lead to attacks on other parts of the SC.

  The traditional protection method is to use VLANs. SDN can lower impacts of attacks by providing a virtual separation of traffic between different applications. In this case, communication of a single platform cannot influence other traffic flows.

- **Experimental features (SDN fully)** - SC is still a relatively new technology and it is therefore perceived as an opportunity for innovative research. New technologies can be therefore implemented into a SC network for research purposes. This is a tempting concept, but experimental integration of untested (or not properly tested) technologies into a real-world network can compromise the security.

SDN can provide virtual separation of the SC main network from the experimental ones. This technique creates a safe environment for testing of new features.

- **Malware (SDN fully)** - an attacker can insert a malicious code into a message. This code can damage the target device, steal information, or install a backdoor access.

  Detection of this attack is difficult, because an inspection on the application layer is required. For this reason, the most common protection mechanisms are located on end devices, where they acts as *last resort* protections. This includes antivirus programs, updated applications and operating system, and also installation of software only from trusted sources and with a signed code. A malicious code can be also detected during the message transmission, but this detection requires a specialized hardware or software. The most common tools are HW-based IPS and IDS, but these devices are costly, they increase the network's complexity, and can limit the performance (IPS only). An SDN controller can also perform a packet inspection, but only in software and if the data payload is not encrypted. The disadvantage of this solution is low performance, because the SDN controller has no hardware acceleration as dedicated, specialized boxes. Moreover, the controller's code is not optimized for such a task.

- **Routing attacks (SDN fully)** - routing protocols are critical parts of a network, as they are responsible for correct traffic delivery. These protocols can be targeted by attackers, most commonly by *falsification attacks*. These attacks use a malicious router, which advertises faked information. This can include false route modifications and corrupted, out-of-order, or delayed messages [241]. These attack types can alter the forwarding of messages according to the attacker's will. A consecutive and potentially more destructive attacks can then be launched. Especially vulnerable types of networks are WSNs, where *sinkhole attacks*, or *selective forwarding attacks* are often used. The second type of routing attack is *overload attack*, which combines routing and DoS attacks. In this case, a router is overloaded by exchange of an extremely large amount of routing information, which can cause router's memory overflow, making the device inoperative.

  Traditional routing protocols are well-known and their code is most often freely available. Any potential vulnerability can therefore be easily found and exploited. SDN, on the other hand, can be used to mitigate these attacks by implementing its own routing mechanisms. An SDN controller has overview of the complete network and can therefore control the routing process. This implementation can be custom made and therefore very difficult to attack. An attacker would have to analyse the routing mechanism first, and then modify the attack specifically for this particular solution. This does not apply for routing mechanisms implemented in open-source controllers, as their code is also freely accessible.

- **Eavesdropping (SDN partially)** - an attacker captures messages, which are sent unencrypted. The attacker can therefore read all their content. This type of attack is passive - captured messages are not modified, but only read. Therefore, detection of this attack is impossible. It is also not possible to prevent capturing of transmitted messages across the whole network.

The only mitigation technique is encryption. But it does not prevent against capture of the message, it only makes the content unreadable for the attacker. A partial protection method can be implemented in an SDN controller. The chance of the attack can be decreased (it will not eliminate the attack completely), if traffic of the flow will be forwarded straight via the controller, instead of going via the normal network. This will however significantly increase the controller's load, so it is not always possible.

- **GPS spoofing (SDN partially)** - manipulation with GPS data can fake location of certain entities as well as tamper with timestamps, which are both critical data for SC operations. Spoofing location of vehicles can cause not only financial losses and property damage, but most importantly people injuries, or even life losses. Similarly, manipulation with timestamps can have various consequences.

  One of the prevention mechanisms is an implementation of PTP. This protocol ensures, that the time on all network devices is current and precise. SDN can be utilized to lower a chance of the attack by caching messages and comparing their values. An obvious values difference between subsequent messages can indicate the attack.

- **MitM attacks (SDN partially)** - an attacker intercepts communication between two devices anywhere in the path. He / she can then read, modify, and resend (or any combination) all captured messages. The main danger of this attack is the fact, that it can stay undetected for a long time. If the attacker only resends messages, nothing suspicious would be detected on the communicating devices.

  The best protection against this attack is traffic encryption via a strong cryptography. In this case, even if an attacker capture the message, he is unable to read its content. The only readable information will be the message's headers. These fields are not encrypted, because networking devices need to use them for message forwarding, loop protection, QoS, etc. By modification of these fields, the attacker can still cause some damage. He can for example fake the message's source, change the destination, or cause the message discard via an expired TTL (Time To Live) value. Similarly to eavesdropping, an SDN controller can forward packets directly from the first forwarding device, to the last. This would again decrease a chance of MitM attack, but would not eliminate the possibility completely.

- **Node attacks (SDN partially)** - the attacks targeting nodes can have many forms. One of them is to compromise a node, while staying undistinguishable from legitimate nodes. In this way, an attacker can simply collect data, gain access into more secure parts of the network, or perform malicious actions. The compromised device might be very hard to detect, if an attacker does not use an aggressive action. Another problem is, that most of the SC sensors of the same type, come from a single vendor. Therefore, if a vulnerability is found, it is very easy to exploit this vulnerability on all the affected devices, potentially damaging large areas of the SC. Nodes are physically distributed across vast physical areas and are therefore extremely vulnerable to physical attacks. Events like floods, fires, or earthquakes, can be relatively easily faked by physical manipulation with a sensor. Impacts of such an attack can be very serious, if an automated action (for example evacuation) is triggered.

  Most of the nodes are unable to run a sophisticated encryption due to their limited performance - both in CPU capabilities and size of RAM. They also run a limited operating system with support of only the necessary protocols. Normal encryption solutions must be therefore modified in order to be supported. This is also the reason, why these protection mechanisms are often not used at all. Even more powerful nodes like SH devices often have critical security flaws, as verified by [242]. This includes NAS, where a configuration file with administrator password can be accessed remotely. After accessing the NAS via an administrator account, any malware, or a trojan horse can be uploaded into the system. Other vulnerable devices are home routers, which can have a "secret" function to allow the ISP remote access. Some of the devices also use primitive root passwords like "1". Even modern smart TVs have serious vulnerabilities in ability to execute a JavaScript code. Once a SH device is compromised, the entire HAN is vulnerable.

Lastly, most of the nodes are integrated into the network and then never updated. These devices are therefore vulnerable to all types of found attacks. Unfortunately, even large systems are often left without updates. An example is state-of-the-art smart building *Burj Khalifa*, which is still controlled by *Windows XP* [234]. Even when updates are issued, they are often not easy to implement. Examples are smart devices in SHs, where users must manually download an update file and load it into the device via a web interface. This process is, for most typical users, very tedious and therefore not widely used. A fast and efficient way of updates must be implemented in SCs to enable up-to-date protection against modern threats.

To protect the network, trust management techniques can be used. These techniques include use of certificates and digital signatures. To protect end devices against local attacks, software protections like antiviruses and updates should be used. If an attack to a node comes from the network (remote attack), it can be stopped by an ACL, or firewall, which can be implemented in an SDN controller. The best approach is not to allow any remote traffic to access the end devices. Only when an end device accesses a remote service, returning traffic is allowed. In this approach, an attacker would be unable to access the device. If this is not possible, each new flow to a device should go via an inspection on the SDN controller. Protection against a direct physical attack cannot be ensured by SDN.

- **Personnel issues (SDN partially)** - there are many different risks, that can come from the personnel department - purposeful attacks (theft, unauthorized action), laziness (use of a default password, turned off protection tools, leaving a device unlocked), or simply an accident (an infected removable device, mistake).

  All of these events cannot be mitigated, but a strong security policy and end nodes protection applications can help to lower the chance of a security incident. The personnel issues happen on the access layer and SDN can therefore not be used for this protection. On the other hand, SDN can lower damage impacts from these issues. It can help to stop DoS attacks and spread of an infection by detecting and blocking suspicious traffic. It can also verify, if a secure connection (VPN, Radius), or an approved login procedure was used. If not, it will block such a communication.

- **Replay attacks (SDN partially)** - valid data is captured by an attacker using the same method as in MitM, or eavesdropping attacks. Captured data is then sent to the server, in order to gain access to the system.

  Prevention can be achieved by implementation of session identifiers for every message [243], or by keeping a cache with previously received messages. Additionally, PTP can be used and messages can include encrypted timestamps, which would indicate a validity period. If a strong encryption, or a reliable CRC is used, the timestamp cannot be forged and verification on the receiving side would reveal a potential replay attack. The protection via a message caching can be also implemented in an SDN controller, but that would require software processing of each packet within a single flow. This could result in high controller load and a higher network latency. Also, in a case of high network / controller utilization, this protection technique may be unusable, as it would detect a large number of *false positive* events.

- **Specific communication links (SDN partially)** - some connection methods of SCs can be done via various technologies with specific limitations. Examples are low bandwidth, high latency links, and links with a limited MTU. Some connections are also composed from multiple links, where each link can have different specifications. On these links, some specific security technologies cannot be used. This may include VPNs, which need to add additional data into frame headers. Omitting these technologies can compromise the network security.

  SDN can partially solve the problem by using its own security solutions (like virtual separation, virtual tunnelling, etc.), while utilizing only fields in standard L2 - L4 headers.

- **Access layer network attacks** - most of the SC nodes are connected to the network via wireless, or PLC networks. These networks are convenient, but can be easily accessed or jammed. The level of security of these networks is therefore significantly lower, than in classical networks. The problem is also a lack of clear physical boundaries of these networks as the signal can propagate for large distances.

  A complete protection is impossible, but network monitoring can limit an attack scope. SDN cannot be used for the monitoring of these specific non-IP networks.

- **Collisions** - collisions during data transmissions occur mostly in wireless networks (Wi-Fi, RFID). Although, they can be caused by an attacker (then it is the DoS jamming attack), most often they just happen randomly. Wireless networks have built-in mechanisms (CSMA/CA in IEEE 802.11) to cope with collisions [197].

  The default collision protection mechanisms are designed to deal with a typical amount of collisions, which can happen during a normal network operation. In case of a purposeful attack, the amount of collisions would be much higher. In this case, additional anti-collision techniques can be used, but SDN cannot prevent this attack.

- **Data modification** - if an unencrypted message is captured by an attacker, the message's content can be modified. This integrity violation can result in unexpected consequences.

  A typical protection method is a message encryption via PKI. SDN cannot deliver an efficient protection against this type of threat.

- **Masquerade and sybil attacks** - an attacker pretends to be someone else by faking an identification certificate, or by using an expired one.

  PKI and a proper revocation mechanism, together with cryptography can be used as prevention. Implementation of these techniques within an SDN application is not feasible.

- **Timing attacks** - are similar to GPS spoofing attacks, but in this case, time-critical, or safety applications are the target. These applications are delay sensitive and to work correctly, they require a precise time information. Examples are grid protection messages, which often have to react on an event within a few milliseconds. Any delay, or an inappropriate time setting could result in a large scale blackout and physical damage to the grid equipment.

  Protection against this attack is to use timestamps in all the messages. Moreover, these timestamps must be protected (by a CRC, or signed with an encryption algorithm). Use of SDN for timestamp validation is ineffective, as it would require checking of all these messages in software. Moreover, this validation would probably be performed multiple times, as the messages would travel across the network.

5.4 Smart City Control Applications

This section describes issues and important considerations of software applications for control of smart cities.

5.4.1 Requirements for General SC Applications

The following requirements apply for all SC applications (not only security-related ones, or SDN-based ones). These requirements were collected from various sources and the requirements in specific categories might therefore overlap with requirements from other categories.

Functional Requirements

Functional requirements describe what the system should do, without explicitly stating, how it should be done. According to [244], the functional requirements for a general application in a SC are (ordered from the most important to the least important):

- **Interoperability** - different applications often use the same data, which can be distributed amongst multiple service providers. Data sharing with reasonable performance must be achieved. This requires use of a generic application architecture.

- **Usability** - high diversity of services used in SCs requires a unified UI for convenient control. This includes applications for citizens, and control and management applications for SC operators. All of these applications should be simple to use.

- **Authentication and authorization** - these basic security mechanisms must be implemented in all SC applications (for operators and users).

- **Availability** - all provided services should be available at all times. This can be a challenging task, especially when multiple isolated services are used.

- **Recoverability** - to ensure an overall reliability, the system must be able to efficiently recover from any failures. A specified time frame for this recovery should be defined.

- **Maintainability** - the amount of various services and nodes in SCs is very high and new functions are being dynamically added. An option to integrate new services and devices is therefore important. The architecture must be able to integrate even services, which were not existing during the original design of the application.
- **Open-source software** - it is important to reduce the cost for the solution as much as possible. Most of the SC applications (up to 83%) are not oriented to generate an income [244], so the cost should be kept low.
- **Confidentiality** - according to [244], only 22% of analysed applications in SCs use encryption, which provides confidentiality. This encryption might therefore be only an optional feature and it should not be required.
- **Performance** is according to [244] not a predominant concern in current SC scenarios. The main argument is, that nowadays, only 22% of current applications require high performance and reliability (and these are mainly safety and health related applications).

Non-functional Requirements

Non-functional requirements define features which the system should have, and specifications of how to achieve them. IoT-A [58] defines the following non-functional requirements for every IoT application, which are also applicable to SC scenarios:

- **Availability and resilience** - these requirements are the most critical in SC environments. The system must be able to stay fully, or partly operational even during diverse failures. A fault-tolerant and redundant hardware must be therefore used with high availability and load-balancing techniques. The software must be designed for failures and must support logging, transactional consistency, backup, and recovery.
- **Evolution and interoperability** - IoT and SCs are dynamic fields with rapidly evolving parameters. It is therefore important to implement software, which can be integrated with future extensions and can interact with new functions. The software should use extensible and standardized APIs, apply design techniques supporting changes (design patterns), and preserve usage of the same development environments.
- **Performance and scalability** - performance requirements of current SCs are nowadays often unclear due to heterogeneity of devices [58]. It is however expected, that these requirements will be more demanding in the future, with the increasing amount of devices and traffic. The system must therefore support future expansions of the infrastructure. Performance requirements can be achieved by QoS, distributed processing, partitioning, asynchronous processing, design choices, and by minimizing use of shared resources.
- **Privacy** - it includes collection of personal identifying information. To minimize the risk of misuse, the collection should be kept to minimum and if possible, data should be stored only locally. To protect the collected data, pseudonymization can be used and identifying data should not be transmitted over unsecured links (especially wireless and power lines). Data should not be accessible without an authorization. Privacy requirements are described in more details in the section 5.4.1.

- **Security** - includes confidentiality, integrity, security policies, and ability to detect and recover from failures. It is also the basic building block for achieving trust and privacy. Security is one of the most complicated requirements, as it spans all areas of the system and requires implementation of various technologies. Some of these technologies include: definition of threat models and access policies, infrastructure hardening, devices authentication, secure communication, and implementation of authorization. Moreover, some less-secured technologies such as wireless communication, PLC, or OTA updates should be used only if necessary. Specific security requirements and issues connected with them, will be described in a separate section.

- **Trust** - is especially important in M2M communications, where every participating subject must be verified. It is an especially problematic requirement for low performance end devices, where a trade-off between performance and security must be considered. Trust can be achieved by implementation of physical security, accountability, tampering detection, and verification of the system integrity.

Performance Requirements

As already discussed above, some researchers claim, that the performance requirements might not be a concern. Nevertheless, in real world scenarios, they should be considered.

In general, the most important performance parameters are latency, bandwidth, and link quality. These requirements rise with an increasing number of devices and supported services. Modern networks offer better performance and lower latencies (as described in section 2.3.3, this is expected to become even better in 5G networks), but there is always a certain latency to process a message in a centralized data centre. This is caused by physical constraints - the data centre can be located many (hundreds) km from an originating node. Messages, which need a faster response must be processed closer to the originating node - concept of FC described in section 3.4. An example use can be in SM messages (communication between a car and RSUs). If a car has a collision, all the surrounding cars must be warned and nearby traffic lights and street signs can be adjusted as well. These messages must be processed as fast as possible. They typically do not require a complicated processing, so a *fog server* does not need to have a very high computational performance (when compared to a full cloud server).

Expected bandwidth requirements in future SCs were given by [66]. The paper expects, that in 2020 a majority of the traffic will be consumed by broadband access for SC citizens (200 - 2500 devices, 875 Gbps / km^2), offices in SBs (75 000 devices, 17 Gbps / km^2), and by special events like music concerts, sport events, etc. (30 000 devices, 2.25 Gbps / event). This amount of traffic includes all SC applications, except low-power sensors. The amount of traffic from low-power sensors will be negligible when compared to the broadband access and the only concern will be therefore a very large number of these sensors (up to 200 000 / km^2).

Unfortunately, implementation of techniques like QoS, for ensuring minimal latency and guaranteed delivery for critical traffic, might be problematic. This is caused by the fact, that major parts of SCs use wireless networks, which might become congested, or can suffer from interferences. In such cases, low latency and sufficient bandwidth cannot be guaranteed. The probability of occurrence of these conditions increases if there are multiple wireless hops on the communication route.

Privacy Requirements

Privacy can be defined as *contextual integrity* - only relevant data is collected and it stays within the related domain [245]. Privacy is closely connected with security - without proper security, privacy cannot be achieved. This rule is particularly apparent in networking protocols, as [237] confirms: "*the security protocols are building blocks for privacy protection*".

Privacy is an especially important issue in SC scenarios as a large amount of sensitive data is collected. Moreover, users are often not able to reject automatic collection of sensitive data, if they are part of the city. Technologies like CCTVs, digital payments, or mobile cellular localization are practically impossible to avoid. To further complicate the issue, some studies ([244]), do not consider privacy (nor security) to be a major issue. This is however in contradiction with most of the current work [237], which emphasizes the importance of privacy.

The basic privacy protection principles are: *minimize, hide, separate, and aggregate.* One of the proposed approaches to achieve these goals was described in [246], where authors discussed the following eight principles. These principles correspond to the recommendations of *ISO/IEC 29100:2011 Privacy Framework* [247]:

- **Minimize** - the first principle states, that the amount of collected privacy data should stay as low as possible. There can be two approaches for selecting data, which should be collected - simple collect / do not collect; or to specify the maximum amount of collected information about a single entity. The implementation of this principle uses selection of collected data, anonymization, and use of pseudonyms.

- **Hide** - any personal information should not be plainly visible. The decision about who can see the data depends on the application. Implementation of this principle uses data encryption and encryption of communication links, to achieve unlinkability and unobservability.

- **Separate** - privacy can be improved by separating data amongst several physical locations. The complete information about a single entity cannot then be determined without distributed processing. Implementation of this principle separates the data into several databases. These databases should not be connected (unless it is really necessary) and whenever possible, local processing should be preferred over cloud processing.

- **Aggregate** - data should be processed with the least possible detail, but while it is still useful. Processing only aggregated data (for example over a group of users, time, or locations) protects sensitive information about a single user. This principle can be implemented by *data fusion methods* described in [248].

- **Inform** - users of a system must be informed, what information is stored and how is it processed. This includes information about data protection, and potential access of a third party.

- **Control** - a user should have an option to view, update, and also delete collected personal information. This is often covered by various legislations.

- **Enforce** - a privacy policy must be compatible with legal requirements and must be enforced during the system operation. Implementation includes definition of a privacy policy document and use of systems for user control (access control systems and privacy rights management systems).

- **Demonstrate** - this principle follows up the enforce principle, but it requires an option to verify, if the privacy policy corresponds to legal requirements. It also includes an ability to immediately detect a privacy breach and to determine the seriousness of this breach. Implementation of this principle includes privacy management systems, logging, and auditing.

Quality Requirements

The standard *ISO/IEC 25010*[249], recommended and analysed by [244], defines quality characteristics for a software product. These requirements improve quality of any software and they are particularly appropriate for any SC application:

- **Compatibility** - the product can share resources and common environments with other products without negatively impacting them. It also specifies, how well the product can exchange information.
- **Functional suitability** - the product covers all specified tasks and user objectives with correct results and required degree of precision.
- **Maintainability** - contains architectural recommendations for building an extensible software. It includes modularity (modification of a single component does not influence other components), reusability (components can be shared with other systems and applications), modifiability (easy change of the product's components, without lowering quality - for example by SW flaws and bugs), testability (effectiveness of providing tests to verify the product's criteria), and analyzability (ability to diagnose the product for flaws, bugs, failures, etc). This also allows identification of components, which need to be changed.
- **Performance efficiency** - specifies a degree, to which various performance parameters meet defined requirements. These parameters include processing times, throughput rates, capacity, and utilized resources.
- **Portability** - defines, how easy it is to modify the product to support new hardware or software, or to deploy it in a different usage scenario. It also covers deployment difficulty (installation and uninstallation).
- **Reliability** - how reliable the product is under normal conditions, and how it behaves, when there is a fault. It also includes availability and recoverability (if the product can restore saved data, and return to the desired state).
- **Security** - includes confidentiality (data available only to authorized users), integrity (no unauthorized access or data modification), non-repudiation (performed actions cannot be latter revoked), accountability (every action of an entity can be traced), and authenticity (identity of each entity can be proven).
- **Usability** - specifies, how easy it is to use the product, including protection against users' errors, usability of the UI, and access features for people with special needs. It also includes help functions of the product (which should provide guidance, how to operate the program), and description, from which users can determine, if the product meets their goals.

Apart from the product quality features, the standard [249] also defines quality characteristics for product usage. These include:

- **Effectiveness** - how easily can users accomplish the specified goals by using the product.

- **Efficiency** - how much resources are needed to accomplish the specified goals.
- **Satisfaction** - includes usefulness (satisfaction of users with achieving their goals by using the product), trust (confidence, that the product will continue to behave as intended), pleasure (ease of using the product), and comfort (degree of satisfaction with the product).
- **Freedom from risks** - how the product mitigates potential risks in the following areas: economic, safety, and environmental.
- **Context coverage** - specifies, how well the system performs in all the previous areas in the defined context (initial requirements are met) and beyond the context (initial requirements are extended with additional ones, which were not expected during the product's design phase).

Security Requirements

Various security standards and architectures have different requirements on security. The following list of requirements merges proposed requirements from the IEC 62351 standard [88] and the IoT-A architecture [58]. Only requirements applicable for a SC deployment are listed.

- **Authentication** - is responsible for verification of users and node devices. Authentication is successful, if a user (or a node device) provides valid credentials. The simplest form of authentication uses the combination of *username* and *password*. This is the most common type for users authentication and can be applied to control applications of a SC (management systems, portals with public information, and building climate control systems). For node authentication, more advanced forms of verification are typically used (PKI with pre-shared keys, digital certificates).
- **Authorization** - determines, if an entity requesting access to specific functions of a SC is allowed (authorized) to perform these functions. This can include access to the SC network and use of certain services and APIs. Authorization is controlled by a system, which allows definition of security policies and provides access control, based on these policies. Security policies can be added, updated, or deleted by system operators.
- **Availability** - it ensures continuous access to information for legitimate users. This includes prevention against DoS attacks. Protection mechanisms include: incident and vulnerability reporting, during-attack copying, post-attack recovery, rate limiting, access management, backup and recovery, network and management systems, ACLs, antivirus tools, IDSs, logging, and virtual network segmenting.
- **Confidentiality** - it prevents unauthorized access. Protection mechanisms include: encryption, certificate and key management, authentication for RBAC (Role-Based Access Control), passwords, certificates, firewalls, ACLs, antivirus tools, IDSs, and logging. A SC network can be organized into separated segments for each type of a smart system. This separation helps to prevent unauthorized access between different components of the SC.
- **Integrity** - it prevents unauthorized data modification and theft. Protection mechanisms for integrity are the same as for confidentiality - data must be protected during the entire life cycle. This can be a complex process in SCs with

large networks and multiple data centres. CRC and digital signatures are typically used for protection.

- **Key exchange and management** - this component can be used for secure communication between two devices, especially if a password agreement was not conducted a priori. The method can distribute keys in a secure way, and it registers security capabilities of each device. A secure communication can then be established. Similarly to the *trust entity*, this solution significantly increases the complexity of the SC infrastructure.
- **Non-repudiation / accountability** - it prevents a denial of performed actions. The system keeps logs of all performed activities for all nodes. Implementation of this protection might be required by current laws. It protects SCs from attackers, malicious operators, hacked sensors, etc. It requires credential establishment and identity establishment systems.
- **Trust and reputation** - it collects and calculates users' trust levels and reputations. These score metrics give information about a node behaviour. Using the metrics, a malicious node device can be detected. To support this feature, the system must manage nodes IDs and timestamps. A *trust entity* (with an identity management system) is typically required to implement these features.

5.4.2 Other Issues

This section describes non-security related issues of SCs. These issues can negatively influence an SDN deployment, or implementation of any security improving application. The issues have to be therefore considered when a new application is integrated into a SC infrastructure.

- **Data acquisition** - there are several different models for data acquisition in SCs. Most often, nodes in a SC run only on battery power and they are programmed to transmit data only in certain time intervals. Some of the networks (LoRa) even have legal requirements, which specify, how often a node can send a message. It is important to consider these requirements and also the fact, that the communication in this case is almost exclusively one-way only. Other models of data acquisition use more traditional publish-subscribe, or push methods.

  Another consideration in data acquisition is noisy data. A large part of collected data within a SC is not useful. This can include duplicated data, which is often present in IoT networks like LoRa. In these networks, data transmitted from a node is often received on multiple gateways and then forwarded via a wired network to the application server, which discards the duplicates. Some data also includes a location information. Based on an application use case, only data from a bounded physical area might be useful. Similarly, some data includes timestamps. It might be necessary to process only the message with the most recent timestamp, as the older ones might be outdated. Another example is data from social networks, which needs to be thoroughly categorized to provide only useful information. The ability to effectively and quickly find useful data can be critical in security scenarios. This data can be marked with a QoS for a priority delivery.
- **Demanding data processing** - large quantities of produced data are caused by the large scope of SCs. There is no unified data format, which would have to be used, so every entity can use a different one. This makes the data processing

demanding, because data conversion is required. Data in a common format (one of the most typical is XML) then has to be cleaned, de-duplicated, and aggregated, in order to provide useful information. All the processing requires significant power resources and incurs a latency. [250]

- **Dynamic scalability** - a SC is an always changing network of nodes, devices, users, and data. Events like sport shows, concerts, traffic jams, emergency situations, or even weather can result in large amounts of data generated in specific locations. The SC network and servers must be able to adjust to these changes and process the data as effectively as during normal operations. The SC's key traffic must not be influenced by these events. Advanced technologies (opportunistic routing, virtual separation, and QoS) might be used to support these requirements.
- **Heterogeneity** - SCs contain a vast amount of different nodes and sensors, using various access protocols. To merge the various data and to integrate it into a unified infrastructure is demanding. This integration includes data from several opposite domains (cars, sensors, users' data) and from different technologies (wired, wireless, RF, Wi-Fi). Moreover, some of the data must be effectively shared between different applications. For example, if a suspicious car within a SC has to be monitored, data from traffic cameras, parking lot sensors, EV charging stations, RSUs, and automatic payment gates can be combined to track the car. An effective coordination from a control centre is therefore required.
- **Mobility of nodes** - some of the nodes in SCs can be highly mobile and they can require a continuous connectivity. This includes seamless transitions between different connectivity options, which might require dynamic routing protocols.
- **Reliability** - some applications of SCs can be used for critical operations. An example is patient monitoring in hospitals, healthcare centres, or elderly homes. Lives of the patients can depend on the application's reliability. This often requires redundant equipment - including end devices (patient body monitoring sensors) and networking infrastructure. The application itself must then be able to cope with any device or link failures.
- **Separate networks** - some of the specific networks like SG substations can be located in remote sites without Internet connectivity. Centralized policy management, remote access, OTA updates, or even deployment of SDN in these areas is therefore not possible.
- **Smart buildings advanced automation** - this can include critical security systems like door locks, motion detectors, security alarms, emergency systems, HVAC, etc. Functionality of these systems must be ensured at all times. For example in an emergency situation (fire), an alarm must sound, doors must be automatically released, and HVAC system must take appropriate actions (limit spreading of the fire). A failure of the system could have fatal consequences. This automation also includes non-critical systems (lights, elevators), whose functionality is not critical and a failure does not typically pose a serious threat.

6. The Blueprint

This section proposes the blueprint for deploying SDN applications into Smart Cities, with the goal of improving security of these networks. Such an application will be referred to as **SSC-SDNA** (Secure Smart City - Software Defined Networking Application) in the following text. This application can focus on any area of security within SCs and can use any SDN features, as long as it uses the OpenFlow standard.

6.1 Application Development Process

There are many approaches to application development. The most known are: *Waterfall Model*, *Prototyping*, *Agile SW Development*, *Rapid Application Development*, *Spiral Model*, *Extreme Programming*, and *Rational Unified Process*. More information about these approaches can be found in[1].

General SC applications can utilize any of the mentioned development process, but in general, the process can be described in the following 6 phases:

1. Planning Phase

The goal of the first phase is to identify potentials of the project - its risks, costs and goals (with the possibility of future extensions in mind). In an SSC-SDNA, another important part is to identify all participating entities. This might include users, residents, employees, operators, but also various sensor devices, vehicles, etc. All these findings should be properly summarized in the project plan. The plan should estimate the project's chance of success, failure, time schedule, and required resources.

2. Analysis Phase

The main goal of the second phase is to gather requirements for the application. They specify, what will the application do, what are its inputs and outputs, and how will it operate. This phase normally includes iterative interviews with customers. This is more complicated in the SC area, with a large number of automated entities. In this case, the requirements must be gathered by other means as well. This includes surveys, analyses, and use cases describing the system operations. A comprehensive list of possible requirements for SSC-SDNAs is described in section 6.2.

3. Design Phase

The third phase specifies methods of achieving goals from the analysis phase. The main tool for the design specification is the system's model composed from various charts. Typically, these charts include: class diagrams, use case diagrams, and flowcharts. Some of the charts can have an integrated pseudo code and they can be used to generate bare structures of the final code. A part of the design phase also includes specification of the UI layout.

In an SSC-SDNA, this phase additionally includes:

1. **Choice of a controller** - use of standardized APIs ensures flexibility in terms of used controller, but nevertheless the choice of a controller is one of the most important decisions for all SDN applications. The selection process must consider the application requirements, goals, and features. Some of the controllers

[1]Available: `https://goo.gl/VFQNNK`

are specifically designed to cover particular areas like security, reliability, performance, high availability or, on the contrary, simplicity and fast deployment.

2. **Choice of the controller's architecture** - after the controller is selected, a deployment architecture must be chosen. There are two options: centralized and distributed (they were introduced in *Control Plane* section 2.1.4). The first one uses only a single controller, while the second one uses multiple controllers.

   The *centralized* architecture is simpler and faster to deploy, does not require additional hardware, and the application functionality can be much more simple as it does not have to deal with complex issues.

   The *distributed* architecture utilizes multiple controllers and therefore offers higher availability, reliability, and performance (via load-balancing). The cost for these features is a more complicated deployment, setting, and operation. The biggest problems are however with synchronization between controllers and with efficient load-balancing. It is also important to note, that not all controllers support this architecture. Currently, the only two open-source controllers, which support it are ONOS and OpenDaylight.

3. **Choice of APIs** - specific southbound and northbound protocols must be chosen for the application development.

   Typically, OpenFlow will be used as the *southbound* protocol, as it is the most widely used protocol and has the biggest potential for SDN applications. The important consideration is the OpenFlow version. Needed features should be compared to the OpenFlow switch specification [24] and the lowest version supporting all the required features can be selected. This version must be supported on the controller and on all forwarding devices. Different versions between the controller and forwarding devices are typically not compatible. On the other hand, on most of the devices and controllers, different versions can be configured. It is also necessary to verify, that the version used in the controller and devices supports all the features, as the optional features might sometimes not be implemented.

   The *northbound* protocol is nowadays mostly represented by the REST. This interface is open, is suitable for distributed and dynamic resources, is multiplatform, is compatible with legacy services, and has adequate performance [251]. The choice of a northbound protocol is especially important, if additional applications like OpenStack are expected to be integrated into the system.

4. **Choice of the application mode** - the software containing the main functionality can be either deployed as an application (which uses a northbound API for communication with the controller), or as an internal module of a controller. In the first case, the *application* can potentially be migrated to a different controller. This flexibility, on the other hand, is redeemed with a little bit slower performance.

   A *module* is developed "within" the controller's architecture and uses the same language. The module uses the controller's internal communication methods for exchange of information with shared controller's functionalities. The main advantage is better performance. On the other hand, the code cannot be simply migrated into a different type of controller and there is a higher chance of software flaws influencing the controller's stability. This can occur if the application is not written properly.

5. **Security threats protections** - the application should be protected from threats described in section 5.3.3. Based on the use case scenario, not all of the protections have to be implemented. Detailed design steps to protect the application are described in section 6.4.

4. Implementation Phase

This is typically the most time demanding phase as it includes programming of the application. The code should be written according to the guidelines from the design phase. The code development should progress in iterations to dynamically react to continuous consultations with system users.

Integral parts of this phase are two types of documentations - internal and external. The *internal* documentation should describe functions of the code and system interfaces. The main purpose of this type is to allow efficient modification of the code (if mistakes are found) and future expansions of the application. The *external* documentation is written for system users and describes the system's functions, behaviour, and tips on effective use.

The testing code of an SSC-SDNA can be deployed in an emulated environment to speed up the implementation. This typically includes a network in the Mininet emulator.

5. Testing and Integration Phase

In this phase, the complete product is deployed to the real infrastructure and its functionality is tested by both the system's users and system's developers. Any found flaws and bugs have to be corrected before the full system operation. The system's operators should also receive training on how to use the product correctly. This should be supported by created documentations. Other users of the system can utilize created guides (for example in a form of a webpage).

In an SSC-SDNA, most of the testing can be performed in the previous step in an emulated network. The real deployment into an existing network may negatively affect the functionality (unless the system uses a completely separate network). For this reason, the deployment should be performed only when the application is properly tested. Another approach might be to use the hybrid SDN, which allows preserving functionalities of the existing infrastructure and testing the new system at the same time.

The deployment phase includes the controller(s) integration. This is composed from the following steps:

1. **OS preparation** - a physical / virtual server has to be prepared to run the controller. Every controller has specific requirements, which have to be fulfilled. The chosen OS should be updated and optimized for running the controller (updated SW, closed unused ports, secured, implemented authentication and authorization mechanisms, etc.).

2. **Installation of the controller** - the selected controller should be installed according to manufacturer's recommendations. Any required tools and packages have to be installed as well.

3. **Settings of the controller** - the controller should be launched with optimized settings for the application. This might include running of only necessary modules, use of specific ports, and activation of appropriate protocols. Access to

the controller should be restricted to only authorized devices and via a strong security policy.

4. **Integration of the application** - the application should be installed and integrated with the controller according to guidelines from the design phase. The integration might require setting up non-typical communication ports or networking protocols.

The final step is to integrate networking devices with the application. This includes:

1. **Configuration of OpenFlow** - a proper version, port number, destination IP address, and other parameters must be configured on networking devices in order to successfully establish a connection with the controller. An encryption of the OpenFlow channel should be set up as well.

2. **Configuration of SSH** - SSH should be used as the only secure way for remote access to the networking device. Secured remote connection can significantly speed up any future configuration, troubleshooting, or modification, as a local physical visit will not be necessary.

3. **Configuration of a failover mode (optional)** - a networking device can be configured for legacy forwarding, if connectivity with the controller will be lost at any point in the future. The *fail-standalone* mode can provide basic connectivity even during such a failure. Alternatively, similar functionality can be achieved with the *fail-secure* mode. This mode however requires specification of appropriate OpenFlow rules in advance.

6. Maintenance Phase

The maintenance phase begins after the deployment of the finalized application. In this final stage, the product is monitored for any potential flaws, which could be corrected. The system performance is evaluated and parts of the system can be continuously updated as the system ages. In an SSC-SDNA, this phase might include updates of the controller's code, update to a new version of an API (OpenFlow), or implementation of new functionalities.

6.1.1 Security Life Cycle

Aside from the development process steps, there is also a security life cycle, which ensures security best practices. The security life cycle is a five-phase iterative process - it represents a never ending circle. Once the last phase is finished, the security process has to start all over again. The process itself should be executed in parallel with the previously defined development process. However, the initial phases must be conducted during the project's planning and analysis phases. Only if the security is considered from the beginning, it can be effectively and systematically implemented in all layers of the system's architecture. Any later security modifications of the system are costly and time consuming. Moreover, they can introduce potential bugs into the software and can provide unauthorized access to the system.

The security phases are [88]:

1. **Assessment** - every asset in a company needs to be classified based on its security requirements, attack probability, potential risks, and cost connected with a successful attack. Security assessment needs to be conducted periodically and it is therefore the first step in each iteration of the security process.

2. **Policy** - based on recommendations from the assessment phase, a plan with security policies is defined. It includes policies for management, implementation, and deployment of security within the domain. If the security assessment is changed, the security policies must be accordingly modified.

3. **Deployment** - this phase includes purchase, implementation, installation, and configuration of products and services chosen in the policy phase. This phase also includes implementation of management procedures for auditing, logging, and intrusion detection.

4. **Training** - it includes continuous education of employees about security threats, security technologies, corporate security policies, and legal requirements. Users need to be aware about modern threats and protections, while system administrators can learn about new security concepts. These updates can then help to improve the security process in the next iteration.

5. **Monitoring** - constant monitoring can detect attacks, security breaches, and even evaluate performance of the security infrastructure. Only if detected in time, these issues can be mitigated, or resolved as quickly as possible. Monitoring also includes auditing, which is typically done after an incident. It verifies, if new protection methods are sufficient. If a flaw is detected, the security process must be started over and improved in the next iteration.

6.2 SSC-SDNA Requirements

This section describes the applicable requirements, which are important for SSC-SDNAs. Data is based on requirements described in section 5.4.1 and from analysis performed in chapter 5. This data was further analysed for applicability in SDN applications. If necessary, original requirements were merged or separated, and classified into the most appropriate sections.

The following requirements represent an extensive list of the most common and interesting features for general SSC-SDNAs. However, their utilization have to be considered with regards to target application needs. Every application has different goals and not all of the requirements from the following list have to be therefore implemented. Also, a level of importance for specific requirements will vary based on the application goals.

Non-functional Requirements

The following non-functional requirements specify, how the system should implement the targeted behaviour.

- **Interoperability** - the application should use open and commonly used southbound and northbound SDN APIs (and potentially westbound and eastbound, if they are used as well). The most common southbound protocol is OpenFlow. For modern applications, version 1.3 or newer should be used. The typical northbound protocol is REST with JSON format for data exchange.

- **Usability** - UI of the application should be consistent, easy to use, and responsive on various types of devices (PCs, laptops, smartphones, tablets).

- **Maintainability** - the application should be programmed modularly. This will allow addition of new features via extra modules. These modules can use defined APIs, or the controller's internal interfaces.

- **Recoverability** - the application should allow creation of backup files and usage of them for potential recovery. The process can be either manual or automatic. The backup should contain physical data and the system configuration.

Performance Requirements

The performance requirements should be specified as explicitly as possible. Although used requirements will significantly vary based on the application goal, some general specifics can be summarized. The typical performance requirements should cover the following features:

- **Backup communication paths** - some SC applications can be packet-loss sensitive. In case of a link failure, data will typically get lost. To prevent this from happening, an instant switchover, during a link failure, can be utilized. OpenFlow 1.1 introduced a concept of *group tables*, which allow multiple ports to be specified for various data forwarding. It can be used for flooding, multicasting, and other specific forwarding. The *fast failover* group is ideal for implementation of backup communication paths, as data is forwarded to the first active port (detailed functionality was described in section 2.1.5).

- **Connectivity-loss detection delay** - forwarding devices and controllers are continuously exchanging *Hello* messages. If these messages are not received within a specified timeout interval (typically 3x - 4x longer than the interval between *Hello* messages), the controller is considered to be unavailable. The forwarding device can then transit into one of the backup modes: *fail-standalone* or *fail-secure*. The interval settings of *Hello* messages and timeout for connectivity-loss detection, should be specified according to the application needs and infrastructure capabilities. A typical default setting is 10 seconds for Hello messages and 30 seconds for connectivity-loss detection. More responsive applications might require more frequent settings (1 and 3 seconds respectively).

- **Load-balancing** - if there are multiple links in the network, the application can dynamically split traffic into them, based on predetermined criteria. *Group type select* of the OpenFlow protocol can be used to perform a simple load-balancing. This feature sends traffic to one of the links selected by an algorithm of the switch. The algorithms are vendor specific, but typically at least *hash* and *round-robin* are supported.

- **Maximum first packet latency** - this type measures latency of the first (or first few) messages in a flow. This latency is normally much higher, than the typical latency. This is caused mainly by ARP, which needs to resolve target MAC addresses based on IP addresses. Another delay is caused by an SDN controller, if reactive flow insertion is used. In this case, the controller must firstly determine, how to forward the flow, and insert corresponding flow rules into all forwarding devices in the path. The typical first packet latency can be[2] between 30 - 100 ms.

  If necessary, the mentioned additional delay can be mitigated by proactive rule insertion. It inserts flow rules into hardware flow tables in advance, so sub-ms first packet latencies can be achieved. The typical scenarios for this type of forwarding are SG substation messages, or certain types of V2X messages (accident reporting, emergency braking).

[2]All mentioned latency estimations depend on the topology size, used devices, utilized services, and the current network load.

- **Maximum typical end-to-end latency** - this type of latency presents typical traffic between two end devices and can be measured one-way, or two-ways as RTT (Round Trip Time). In both cases, it is measured after flow rules are created (and inserted into all forwarding devices), end devices have learned MAC addresses of other devices, and forwarding devices have built *port-to-mac* mapping. Each traffic type can have different latency requirements. The typical RTT latency in L2 networks can be around 1 - 5 ms.

- **Minimum bandwidth** - it defines the amount of data transferred between two points over a certain period of time and it is therefore stated in *Gbps / Mbps*. Each flow can have different bandwidth requirements. If a single connection is shared between several flows, QoS might be used to comply with minimal bandwidth requirements.

- **Minimum throughput** - it defines the amount of *usable* data transferred between two points over a certain period of time. Throughput is therefore always lower than bandwidth, which includes all data. Overhead of unusable data depends on used protocols and technologies. Each ISO/OSI layer adds its own header (and sometimes trailer as well), and therefore additional (not usable) data. Some protocols have features, which also affect throughput. An example is TCP and its *window size* (with a higher value, higher throughput can be achieved). Encryption techniques (VPN) add even more overhead. A proper specification of the targeted throughput therefore requires knowledge about all used protocols and technologies.

- **OpenFlow rules performance** - includes characteristics such as maximum number of supported flows, maximum latency of a flow rule modification, and forwarding performance based on the flow table fullness.

- **Statistics refresh rate** - OpenFlow protocol uses the request-reply communication for collection of statistics. The request messages are sent from a controller in defined timed intervals and forwarding devices reply with their statistics. The interval length depends on the application needs, but typically it is in range of 1 - 60 seconds. A shorter interval can provide a more precise insight into the network operation, while a longer interval introduces a lower overhead.

- **QoS** - SC traffic includes a large number of various protocols and data flows. Each of these flows can have different requirements on performance and delivery priority. An effective solution to support these requirements is to use QoS within OpenFlow protocol. QoS can classify flows based on the traffic type, source, destination, or any other parameters supported by OpenFlow. Based on these parameters, various actions can be performed with the flow. The flow can be forwarded, discarded, modified, or moved into a special *queue*. OpenFlow supports the following types of queues [25]:

  - **Guaranteed minimum rates** - only minimum data rate can be set (since OpenFlow 1.0).
  - **Guaranteed minimum and maximum rates** - minimum and maximum data rates can be set (since OpenFlow 1.2).
  - **Metered queues** - rate-monitoring of traffic can be used prior to the output processing (since OpenFlow 1.3).

An SDN controller can monitor utilization of forwarding devices and specific links, and control the network based on the results. If, for example, a congestion is detected, a lower-priority traffic can be forwarded via a slower path or discarded immediately.

- **Support of critical end nodes** - to ensure, that all critical end nodes will be able to transmit and receive data, the performance requirements should be analysed in the application design phase and deployed in the implementation phase. This will ensure, that even during a specific event, all important traffic will be delivered. Optionally, QoS can help to achieve this functionality.

Privacy Requirements

Privacy requirements described in section 5.4.1 include a lot of concepts, but only some of them can be applied in an SSC-SDNA. These include:

- **Aggregation** - when possible, the processed data should be aggregated, so the identity of a specific user is not revealed. This also includes inserted OpenFlow rules. They can reveal specific communication flows via information like source, destination, or type of traffic. By using aggregated flow rules, the connectivity can be preserved, but with a lower danger of privacy leaks.

- **Anonymization and minimization** - logging, metadata, fingerprinting, and other techniques should be used to minimize the amount of collected data as much as possible. If applicable, the data should be anonymized, so the users' privacy would not be violated in case of a security breach.

- **Enforcement and demonstration** - a privacy policy document with clearly specified rules and sanctions should be created and system operators should comply to it. Supporting systems for privacy management, logging, and audit should be used for detecting possible privacy breaches effectively.

- **Privacy protection** - depending on the application use case, privacy protection can target various areas. The application might deal only with encrypted data (user credentials, payload of data traffic), controller might be selected based on privacy requirements, and all the application's communication might be encrypted (including *vertical* and *horizontal* communications). Correct implementation of privacy protection will ensure unlinkability and unobservability, which correspond to the hide principle.

- **Transparency** - users accessing the system should be informed, what type of data is collected and how is it processed. There should also be an option to control the amount of collected data. Examples of such a notification are *cookie banners*, which have been required in the European Union since 2012 [252].

- **Virtualization** - certain data traffic flows can have higher potential for malicious content. An example is traffic from mobile devices. These flows should therefore be virtually separated from the other traffic types in order to increase the privacy protection. Virtual separation of traffic can also significantly reduce infrastructure cost and safely increase link utilization. This concept is similar to VLANs, but utilization of SDN can achieve better results (for example, there is no limit of 4096 VLANs). This concept can also be called *network overlays* or *isolated networks*.

Quality Requirements

Quality requirements describe mainly usage specifics and they are highly dependent on the application's goals. Some of them (performance, usability, security), were already described, or will be described in specific sections. Other important quality requirements for an SSC-SDNA are:

- **Compatibility** - the application must not negatively influence any part of the controller's architecture (including libraries and internal modules). This can be ensured by using only proven northbound APIs, or internal APIs of the controller. The application can also provide its APIs to allow external access to its data.
- **Effectivity, efficiency, and satisfaction** - the application should utilize only necessary resources (HW and SW - libraries and other shared code). The UI of the application should use typical control layouts and principles. This allows users to control the application intuitively and to make their work efficient.
- **Functional suitability** - all the features defined in the design phase of the product must be implemented and fully functional.
- **Maintainability and portability** - these requirements are typically automatically achieved in SDN by use of standardized open protocols (REST, OpenFlow) and controllers with modular architectures. The application should be built on the same principles as the targeted platform, to support code reusability, modifiability, modularity, testability, and analyzability. Use of OpenFlow protocol guarantees, that the application can be deployed on any forwarding device supporting the same version of the protocol. Use of a common northbound protocol then ensures, that the application can potentially use any controller. This also allows updating, or changing of the controller in the future.
- **Reliability** - can be classified into several levels. On the most basic level, reliability of the application under typical conditions is defined. The more advanced levels include the application behaviour during a failure. These levels can significantly increase the application complexity and might not be feasible for all scenarios. One of the solutions is to deploy the application on multiple controllers - on a distributed architecture. The application should also support backup and recovery of its settings and important data. In this case, the recovery process after a failure can be much more effective.
- **Safety** - the application must be safe for people to use, and it should also minimize the chance to cause a physical damage (economic, environmental, etc.). This might be challenging especially in cyber-physical systems like SG. Security and privacy requirements must therefore prevent, or at least minimize these risks.

SDN Architectural Requirements

These requirements are specific only for SDN and they were not part of the original requirements for general applications described in section 5.4.1.

- **Controller server's requirements** - an SDN controller can run on single, or multiple servers (distributed architecture). Alternatively, a virtualization might be used to utilize a single hardware for multiple servers. A controller must be chosen based on the application use case, number of devices, volume of traffic, security requirements, reliability requirements, and scalability. Each controller has specific hardware requirements. Typically, the more advanced controllers like OpenDaylight and ONOS have higher requirements. According to [253], just to run OpenDaylight or ONOS controllers, without any demanding applications, a server with at least 4 GB of RAM is required. For the SC environment with a large amount of traffic and demanding applications, a much more powerful device is needed.

For a controller, the key performance criteria are: flow modification per second, maximum number of managed switches, maximum number of processed OpenFlow messages per second, and other application specific parameters (a database size, maximum number of connected applications, etc.).

- **Distributed controllers architecture** - in most of the cases, SC applications require high availability and reliability. To accomplish these parameters, redundancy must be introduced to all critical parts of the network, including links, forwarding devices, SDN controllers, and servers running the application. To achieve continuous connectivity even during a fault, the distributed architecture of controllers must be used. This functionality has been supported in OpenFlow protocol since version 1.2 [25], and it is common in advanced SDN controllers (ONOS, OpenDaylight) as well.

- **Failover mode** - even when the distributed architecture is used, there might be a situation, when connectivity between controllers and forwarding devices will be interrupted. In this case, it is important to determine a behaviour, which the affected forwarding device should take. Typically, there are two possible modes, which should be considered based on the application goals:

  - *Fail-standalone* - when a connectivity loss is detected, all the flow rules are removed from flow tables and the forwarding device will transit into the legacy mode. In this case, it is important to configure the device for the legacy forwarding, so that it will behave desirably when the fault happens.
  - *Fail-secure* - in this mode, flow rules are kept in flow tables, unless their timers expire. The only flow rule, which will be immediately removed is the default one (*send it to the controller*). This approach allows preservation of the same SDN functionality as before the failure. On the other hand, until the connectivity is restored, no new flow rules can be learned. This mode does not need any configuration of the legacy forwarding.

  The decision, which mode to use, can be a trade-off between availability and security. The *fail-standalone* mode can be easily configured to preserve basic connectivity, although no traffic filtering, QoS, or any other advanced features, implemented in SDN, will be present in the network. The *fail-secure* mode can preserve these features, but there is a risk, that some less active flows can lose connectivity (as their flow rules expire). Also, any new flows will be automatically dropped as there will be no associated flow rule.

- **Forwarding device requirements** - the most critical feature for forwarding devices is support of OpenFlow and its version. Nowadays, the minimum recommended version is 1.3. As described in section 2.1.5, the older versions of the OpenFlow lack some important features. It is also important to check specific vendor implementations, as most of the OpenFlow features are optional and therefore do not have to be implemented in a specific device.

- **Hybrid OpenFlow** - functionality of the legacy forwarding can be combined with SDN, if the forwarding devices support this feature. In this case, some networking functions are left to the traditional mode of the forwarding device, and therefore do not have to be implemented in SDN. This can significantly simplify and speed up the application deployment process. Every incoming packet to a networking device is firstly sent to a flow table. The table can contain rules, which forward the traffic to legacy forwarding (*normal layer*).
- **Management connectivity requirements** - a deployment of SDN requires connectivity between forwarding devices and controller(s). This connection is used for OpenFlow traffic. It can be realized as *in-band*, where the link is shared with normal traffic. This type is easy to implement and has no additional cost. On the other hand, shared traffic can bring security issues and lower performance. If this type of connection must be used, a virtual separation of traffic (via VLANs) should be implemented.

  The second option is *out-of-band* connection. In this case, the link between a forwarding device and the controller is dedicated only to the OpenFlow traffic. This is the preferred connection type, as it offers the highest security, performance, and reliability.

Security Requirements

The following requirements are applicable for most of the SC deployments. They should be required even while their implementation can complicate the system development.

- **Authentication and authorization** - the application must support authentication and authorization to all areas, which contain private data. This includes mobile applications, backend services, and management systems. Typically, only one-way services (nodes providing public information) might not require any user credentials. On the other hand, for logging purposes and legislative requirements, most of the services (including publicly available APs) should require some sort of user verification.
- **Availability** - it might be required to implement a redundant architecture (compatible with the distributed architecture of OpenFlow protocol). Such an architecture can provide system availability even during a device failure, or an attack. In this case, it might be necessary to implement data synchronization between individual controllers. This might not be possible on controllers, which do not support such an architecture. Moreover, a protection mechanism against availability attacks (DoS) should be implemented as well. This can include traffic limiting, ACLs, and FWs.
- **Confidentiality and integrity** - to prevent data loss and unauthorized modification of data, use of encryption might be required. This encryption can be implemented within the application, on the controller, and on all related network communication links and devices. The application and controller might utilize encryption only for sensitive information. Encryption of the entire solution would significantly complicate the application development and future updates.

  Communication links should be protected especially between a controller and forwarding devices, where OpenFlow messages are exchanged. These messages are crucial for correct functionality of the entire network. If an attacker can modify these messages, the entire network will become compromised. Encryption on this layer might not be required only when the connection is realized via a closed *out-of-band* link.

- **Non-repudiation / accountability** - the application should support logging of performed actions. All the important actions, which can potentially influence the system behaviour, should be logged. This includes actions performed by authorized users. An example of such a log entry can be:

  Date, Time, User: Action, Details

  Where *details* can optionally log information about the user's device (used OS, MAC address, web browser) and access methods (location, IP address / console port).

  The application can also log events performed by the controller, or log messages received from forwarding devices. This can include flow modification messages, error messages, statistics, controller's load, etc. Example of this log entry can be:

  Date, Time, Controller: Event

  Where *event* parameters will depend on its type. Typically, if the event is an OpenFlow message, ID of the originating device should be included together with information about message type, message code, etc.

- **Secured nodes** - an additional network protection from bogus end nodes can be implemented by verifying a node's identity. A custom SDN solution can require a predetermined value from each end node. Only if this value is valid, the node will be allowed in the network. This value must be inserted into one of the supported header fields to be matched by OpenFlow.

  The most appropriate field for this functionality is the *flow label* in IPv6 header. This field has 20 bits and therefore supports up to 1 048 576 values. The purpose of this field is to speed up the routing process by quickly identifying a specific flow. With this approach, a router does not have to compare destination IPv6 address, but can quickly forward the packet based only on information in the flow label field. However, in today's networks, this field is rarely used and can therefore be safely utilized for SDN applications (at least in private LANs and MANs).

  A mechanism of the value selection can vary. It can be a value calculated by an encryption protocol, or a pre-selected value paired with the device's MAC address and stored in the controller's database.

- **Traffic filtering** - it is feasible to integrate functionality of traffic filtering into the application. Based on the application requirements, it can be a simple ACL, or a more advanced firewall. Both methods can *allow* and *deny* selected flows either manually or automatically. Depending on the targeted granularity, several OpenFlow fields can be matched. The most general approach is to block the entire traffic between two devices (via their MAC addresses). A more detailed approach can block only a selected flow (for example based on TCP ports).

- **VPN** - provides data encryption by establishing virtual tunnels over public networks. If a data encryption is required, VPN can be used for selected traffic. This traffic will have payload encrypted and SDN (or any other tool) will therefore not be able to perform the application layer inspection.

Other Required Technologies

The following technologies can improve the application security, resiliency, and availability. These technologies allow efficient implementation of features, which are not available in SDN, or where their implementation would be too complicated. A combination of SDN and traditional networking (via hybrid OpenFlow) can be used to integrate these features. This is the most efficient approach to offer the best possible functionality from both technologies. The particular supporting technologies are:

- **Identity and Access Management (IAM)** - is a system for controlling information about users. It supports authentication, authorization, audit, roles, and delegation. The system is capable of *single sign-on*, as the user information is stored in a centralized entity. The most well-known protocols are SAML (Security Assertion Markup Language), Open Authorization, and OpenID Connect. [254]

  These systems are relatively complex and it is therefore more efficient to integrate them into the network, than to implement their functionality within an SSC-SDNA. One of the main risks of custom implementation is the potential to introduce a security bug. This could compromise the entire system's security. The IAM system can be deployed on the same physical device as the SDN controller.

- **Key management systems** - these systems bind keys to entities and provide key exchange and management mechanisms (generation, handling, distribution, replacement, and storage). PKI is an example of a key management system. It is responsible mainly for providing authentication (via digital certificates) and encryption (via public-keys), and it is often used in SSL and TLS.

  There are many open-source and proprietary key management systems available. All these solutions are complex and custom implementation within an SSC-SDNA is not reasonable.

- **Reputation and trust systems** - the traditional approach to security is to protect resources from malicious users. Trust and reputation systems, on the other hand, protect users from resource providers. A centralized or distributed reputation system collects ratings from users about a particular service. Every provider then has a score, which can be used as a deciding factor, if to perform a transaction or not. The main difference between reputation and trust systems is the mechanism of reputation calculation. In reputation systems, the score is calculated based on ranking of all participating devices. In trust systems, the score is determined only between two entities. [255]

  These systems might become important in SCs, where everyone can provide certain services (public Internet access, information about specific places, etc). Reputation score can be also used for detecting malicious devices.

- **Trusted Platform Module (TPM)** - is a dedicated hardware chip for cryptographic operations. It allows data encryption and decryption, and provides *secure boot.* In this case, it stores a hashed value of the original OS. During each boot, this value is compared to a calculated value of the current state of the OS. If a change is detected, the system will not boot. TPM can protect smart nodes from physical attacks, but the main disadvantage is, that it requires a specialized hardware. [76]

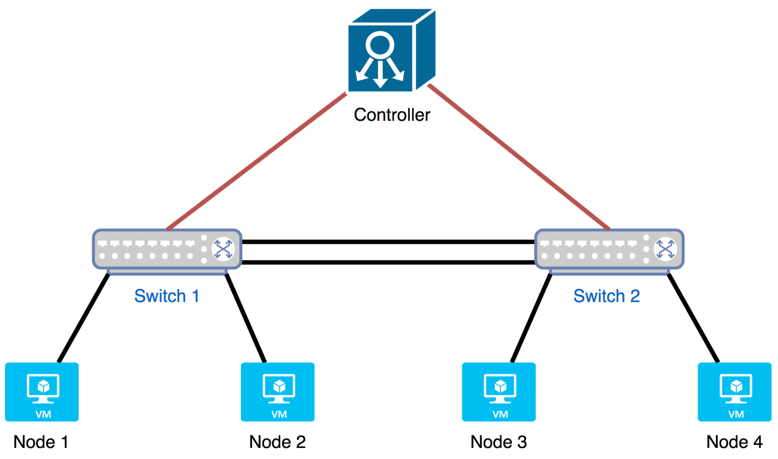


Figure 6.1: Topology of Real Deployment Performance Verification

6.3 Real Deployment Performance Verification

This section verifies performance requirements of SCs described in analyses from sections 5.1.3, 5.2.2, 5.2.5, and 5.4.1. The goal is to verify, if a typical SSC-SDNA can accomplish these requirements. Such an application would use OpenFlow, RYU controller, STP, and a failover-mode. Based on the results, a performance baseline for typical SDN-enabled forwarding devices is defined.

The following experiments were tested with the following equipment:

- **HP switches** - 3800, OS version: KA.15.17.0008
- **Cisco switches** - 2960, OS version: 15.0(2)SE4
- **RYU controller** - version 3.9

The topology of all experiments is presented in Figure 6.1. Primarily, HP switches were used (Cisco switches were used only for the STP verification). The redundant link between switches was used only for the STP verification - in other experiments, it was disabled.

Flow Matching Process Testing

Flow tables on hardware devices can be implemented in several data structures, most often in TCAMs. These tables should provide the consistent lookup times, no matter how full the table is (even if the table is almost full, lookup times should be the same as when the table is almost empty). The purpose of this experiment was to verify this assumption.

Firstly, only several flow rules were inserted into the flow table of *switch 1* and throughput between node 1 and 2 was measured. Then a *script* was used to generate *general* OpenFlow rules until the table was completely full. The performance was tested again. In the last scenario, the script was modified to generate more *specific* rules. These rules contained source and destination MAC addresses, VLAN ID, IP protocol, and source and destination TCP ports. The following code was used to create the match:

```
match = parser.OFPMatch(eth_src=rand_mac_src, eth_dst=rand_mac_dst, eth_type=0x0800,
   vlan_vid=10, ip_proto=6, tcp_src=rand_tcp1, tcp_dst=rand_tcp2)
```

Table 6.1: Results from Flow Matching Process Testing

Experiment	Throughput	Avg. RTT	Max. RTT
1) Empty table (few rules)	927 Mbps	0.604 ms	1.083 ms
2) Full table, general rules	928 Mbps	0.688 ms	1.283 ms
3) Full table, specific rules	927 Mbps	0.526 ms	0.780 ms

Note: Transmitted amount of data in all the experiments was 1.08 GB.

Detailed results are shown in Table 6.1. The variance in measured throughputs and RTTs between the three scenarios was negligible. The maximum RTT showed some differences, but this can be contributed to normal operation of the network (latency caused by other networking protocols, device utilizations, etc.).

The experiment also revealed one interesting fact: *HP 3800* switches were able to store only 500 flow rules in hardware tables. Up to 65 000 entries could be stored in software tables, but this would significantly lower the performance.

RYU Controller Performance Testing

This scenario tested, how well can the RYU controller cope with strict latency requirements. The goal was to test, how long it takes the controller to react on a network event. Data traffic was sent between node 1 and 2 and the time between *packet-in* events and *flow-modification* messages was measured. In total, ten experiments were conducted and the average measured latency was: **4.491 ms**.

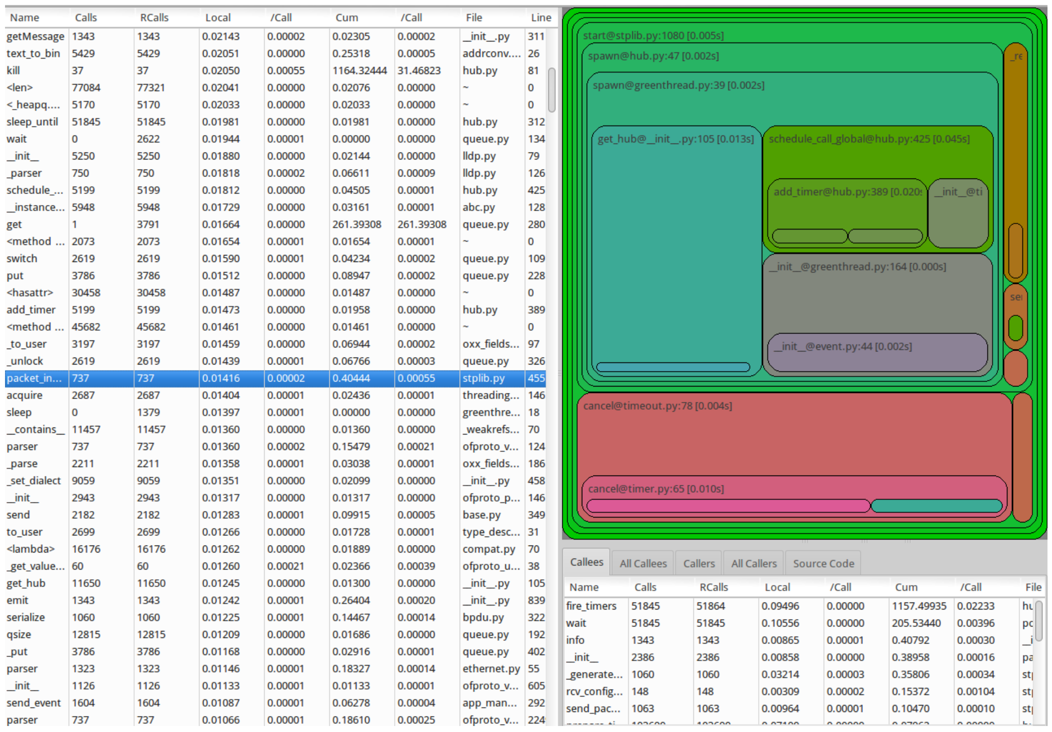
An additional verification was performed by the *code profiling* technique. In this case, a special code was inserted into several parts of the RYU controller. The goal was to measure, how much time the controller spent executing specific parts of the code. The controller was running in standard mode with normal amount of traffic present in the network. The experiment run for 228 seconds and number of function calls and time spent in certain functions were (ordered by the longest spent time amount):

- **getMessage**: 1343 calls, 21.43 ms
- **sleep_until**: 51845 calls, 19.81 ms
- **init (lldp)**: 5250 calls, 18.80 ms
- **packet_in**: 737 calls, 14.16 ms
- **init (ofproto)**: 2943 calls, 13.17 ms
- **serialize**: 1060 calls, 12.25 ms

The complete results from the experiment are shown in Figure 6.2. The graphical diagram in the right part of the figure displays the code structure of the STP mechanism implemented in the RYU controller.

Reactive Rule Insertion Delay Testing

The previous experiment measured response time between a *packet-in* and a *flow-modification* message. This scenario tested the impact of this response time. *Ping* was used to generate ICMP (Internet Control Message Protocol) messages between the same two devices (node 1 and 2) and the response times were captured. As expected, the first packet of every flow had a significantly higher RTT than the following ones. Average RTT values from 10 measurements for the first three packets were:


start@stplib.py:1080 [0.005s]
spawn@hub.py:47 [0.002s]
spawn@greenthread.py:39 [0.002s]
get_hub@__init__.py:105 [0.013s]
schedule_call_global@hub.py:425 [0.045s]
add_timer@hub.py:389 [0.020
__init__@ti
__init__@greenthread.py:164 [0.000s]
__init__@event.py:44 [0.002s]
_re
sei
cancel@timeout.py:78 [0.004s]
cancel@timer.py:65 [0.010s]
Callees
All Callees
Callers
All Callers
Source Code

Name	Calls	RCalls	Local	/Call	Cum	/Call	File	Line
getMessage	1343	1343	0.02143	0.00002	0.02305	0.00002	__init__.py	311
text_to_bin	5429	5429	0.02051	0.00000	0.25318	0.00005	addrconv....	26
kill	37	37	0.02050	0.00055	1164.32444	31.46823	hub.py	81
<len>	77084	77321	0.02041	0.00000	0.02076	0.00000	~	0
<_heapq....	5170	5170	0.02033	0.00000	0.02033	0.00000	~	0
sleep_until	51845	51845	0.01981	0.00000	0.01981	0.00000	hub.py	312
wait	0	2622	0.01944	0.00001	0.00000	0.00000	queue.py	134
__init__	5250	5250	0.01880	0.00000	0.02144	0.00000	lldp.py	79
_parser	750	750	0.01818	0.00002	0.06611	0.00009	lldp.py	126
schedule_...	5199	5199	0.01812	0.00000	0.04505	0.00001	hub.py	425
__instance...	5948	5948	0.01729	0.00000	0.03161	0.00001	abc.py	128
get	1	3791	0.01664	0.00000	261.39308	261.39308	queue.py	280
<method ...	2073	2073	0.01654	0.00001	0.01654	0.00001	~	0
switch	2619	2619	0.01590	0.00001	0.04234	0.00002	queue.py	109
put	3786	3786	0.01512	0.00000	0.08947	0.00002	queue.py	228
<hasattr>	30458	30458	0.01487	0.00000	0.01487	0.00000	~	0
add_timer	5199	5199	0.01473	0.00000	0.01958	0.00000	hub.py	389
<method ...	45682	45682	0.01461	0.00000	0.01461	0.00000	~	0
_to_user	3197	3197	0.01459	0.00000	0.06944	0.00002	oxx_fields...	97
_unlock	2619	2619	0.01439	0.00001	0.06766	0.00003	queue.py	326
packet_in...	737	737	0.01416	0.00002	0.40444	0.00055	stplib.py	455
acquire	2687	2687	0.01404	0.00001	0.02436	0.00001	threading...	146
sleep	0	1379	0.01397	0.00001	0.00000	0.00000	greenthre...	18
__contains__	11457	11457	0.01360	0.00000	0.01360	0.00000	_weakrefs...	70
parser	737	737	0.01360	0.00002	0.15479	0.00021	ofproto_v...	124
_parse	2211	2211	0.01358	0.00001	0.03038	0.00001	oxx_fields...	186
_set_dialect	9059	9059	0.01351	0.00000	0.02099	0.00000	__init__.py	458
__init__	2943	2943	0.01317	0.00000	0.01317	0.00000	ofproto_p...	146
send	2182	2182	0.01283	0.00001	0.09915	0.00005	base.py	349
to_user	2699	2699	0.01266	0.00000	0.01728	0.00001	type_desc...	31
<lambda>	16176	16176	0.01262	0.00000	0.01889	0.00000	compat.py	70
_get_value...	60	60	0.01260	0.00021	0.02366	0.00039	ofproto_u...	38
get_hub	11650	11650	0.01245	0.00000	0.01300	0.00000	__init__.py	105
emit	1343	1343	0.01242	0.00001	0.26404	0.00020	__init__.py	839
serialize	1060	1060	0.01225	0.00001	0.14467	0.00014	bpdu.py	322
qsize	12815	12815	0.01209	0.00000	0.01686	0.00000	queue.py	192
_put	3786	3786	0.01168	0.00000	0.02916	0.00001	queue.py	402
parser	1323	1323	0.01146	0.00001	0.18327	0.00014	ethernet.py	55
__init__	1126	1126	0.01133	0.00001	0.01133	0.00001	ofproto_v...	605
send_event	1604	1604	0.01087	0.00001	0.06278	0.00004	app_man...	292
parser	737	737	0.01066	0.00001	0.18610	0.00025	ofproto_v...	224

Name	Calls	RCalls	Local	/Call	Cum	/Call	File
fire_timers	51845	51864	0.09496	0.00000	1157.49935	0.02233	hu
wait	51845	51845	0.10556	0.00000	205.53440	0.00396	pc
info	1343	1343	0.00865	0.00001	0.40792	0.00030	_i
__init__	2386	2386	0.00858	0.00000	0.38958	0.00016	pa
_generate...	1060	1060	0.03214	0.00003	0.35806	0.00034	st
rcv_config...	148	148	0.00309	0.00002	0.15372	0.00104	st
send_pac...	1063	1063	0.00964	0.00001	0.10470	0.00010	st

Figure 6.2: Results of Code Profiling in RYU Controller

1. 17.2 ms
2. 0.582 ms
3. 0.668 ms

The increased latency of the first packet is caused by processing on the controller. The controller must decide in software, where to forward the packet and consequently insert a flow rule into the appropriate device. The following messages are therefore handled by this rule (in HW) immediately on the switch.

STP Stability Testing

The goal of this experiment was to verify the stability of STP. The redundant link was sequentially brought up and down, and traffic downtime was measured (traffic was transferred between node 1 and 3). Behaviour of the STP implemented in the RYU controller was compared to STPs implemented in HP and Cisco switches. The results are displayed in Figure 6.3. The figure shows, that there is no significant difference between the software implementation of the STP in the RYU controller and its counterparts in HP's and Cisco's switches (defined downtime and uptime intervals of standard STP are 30 seconds).

Fail-standalone Mode Testing

If connectivity between an OpenFlow device and a controller is lost, the device can transit into one of the following modes:

- **Fail-standalone** - all OpenFlow rules present in the device are removed and the legacy forwarding takes place.

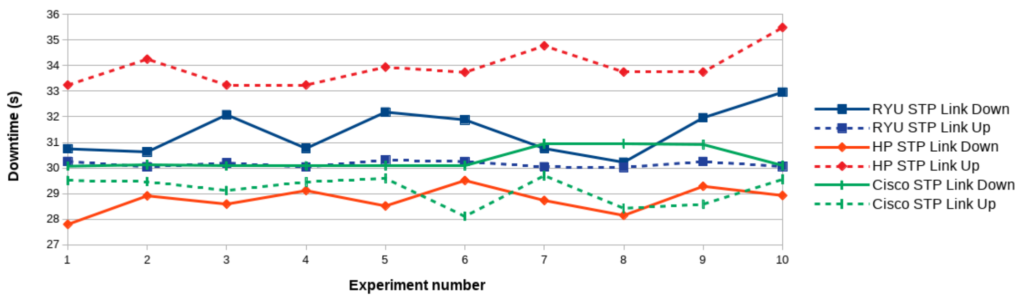


Figure 6.3: Downtime Variances in Different STP Implementations

- **Fail-secure** - only the default OpenFlow rule (*send it to the controller*) is removed. Other OpenFlow rules continue to expire (or not) according to their set timeouts.

The experiment tested, if the HP switch will transit into the configured *fail-standalone* mode, after the connectivity with the controller is lost. The link between the device and the controller was purposefully brought down and the time of transition was measured. All the switch timers were left to default values, so it took almost 30 seconds before the switch made the transition. Afterwards, it started to act as a traditional L2 switch with pre-configured switching behaviour.

Transition times from 10 experiments are shown in Figure 6.4. Displayed values show the time between the shutdown of the link (detected by the device), and the device transition into the *fail-standalone* mode. All flow rules were always removed exactly one second after the transition. Data shows relatively consistent performance across conducted experiments. The average time of transition was **28.012 seconds** and the spread between minimal and maximal value was **3.62 seconds**.

6.3.1 Real Deployment Performance Baseline

The performed testing verified, that an SSC-SDNA can accomplish SC performance requirements and that it can compete with similar solutions based on traditional networking.

Flow table lookup times were consistent and did not depend on the flow table occupancy. The RYU controller showed very reasonable reaction time, with an average latency around 4.4 ms (between *packet-in* and *flow-modification* messages). The STP implemented in the controller then had very similar results to the STP used in commercial devices of traditional networking. Measured transition times of the *fail-standalone* mode were also relatively consistent (almost 13% deviation should typically not represent any major problems, but certain applications might need to be aware of this behaviour).

Latency Performance Issues

One of the most strict SC requirements, as analysed, is latency in certain situations. Tests of the most common type of the flow rule insertion method - reactive - measured the average first packet latency to be 17 ms. This clearly exceeds previously analysed SC requirements: 3 ms for SG networks, 1.5 - 5 ms for certain wireless protocols (connection establishments), and 1.5 ms for V2V networks. The 17 ms latency, caused by software processing, is considerable larger than an average hardware processing latency

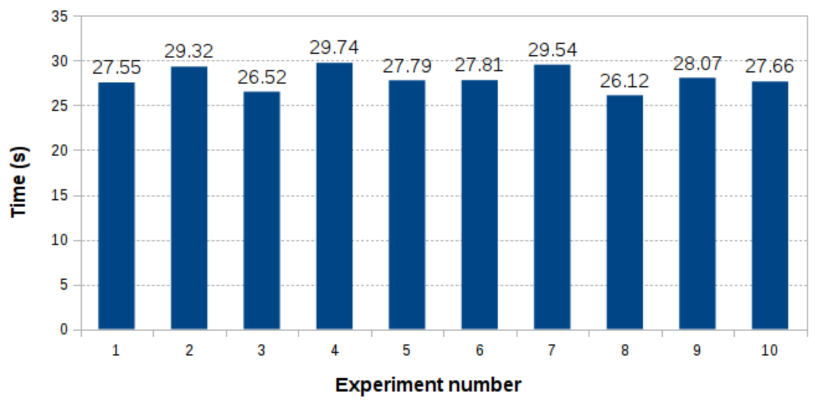


Figure 6.4: Fail-standalone Mode Transition Times

of subsequent packets - 0.6 ms. Although latency of the first message in traditional networking is also higher due to various protocols (ARP), it does not reach this high value. Moreover, the reactive flow rule insertion latency can significantly increase in more complicated network topologies. This increase will depend on the controller approach to handle reactive flow insertion. In general, there are two approaches:

- **Sequential** - The easiest approach implemented in simpler controllers (RYU) responds to all *packet-in* messages received sequentially from all switches along the path. This approach therefore linearly increases the first packet latency.
- **Instant** - A more advanced approach (implemented for example in Floodlight) sends *flow-modification* messages to all corresponding devices along the entire path at once - in reaction to the first *packet-in* message. The first packet latency should therefore not vary significantly between network topologies with different sizes.

Applications requiring lower latencies have to consider proactive flow rule insertion, where flow rules are preloaded into the devices in advance. This approach can eliminate the increased first packet latency at the cost of potentially higher utilization of the flow table, and more complicated application logic. If the minimal latency is required, static IP to MAC mapping should be also configured on end nodes to eliminate the additional latency caused by ARP.

Performance Baseline

The tests verified, that *HP 3800* switches can accomplish the requirements, but with a single severe limitation. The maximum number of supported OpenFlow rules, which could be saved in hardware flow tables, is only 500. This would be a significant limitation in SC scenarios, where a large number of flows must be supported. For this reason, the next generation of these switches - *HP Aruba 3810M* - was selected as the baseline. This line of L3 OpenFlow capable switches supports up to 64 000 OpenFlow rules and also has improved performance. Detailed specifications of these switches are shown in Table 6.2. This table should be used as the baseline for defining minimum performance requirements for SDN-enabled forwarding devices used in SCs. Any device, which can achieve these requirements should be adequate for an SC application (providing that it will support other required features as well).

Table 6.2: Performance Requirements on Forwarding Devices

Recommended specifications	**Reasons and impacts**
CPU: 2x 1.2 GHz	Overall device performance, SW features
RAM: 2/4 GB	Overall responsiveness, SW features
OF rules capacity: 64 000 e.	SDN forwarding in HW
MAC table capacity: 64 000 e.	Forwarding speed, attacks protection
Flow mod/s: >1 000*	Performance in managing OF rules
Forwarding capacity: 320 Gbps	For effective throughput of 190 Mbps
Forwarding latency: under 3 μs	For 1 Gbps links

* As tested in [256] for *Pica8 P-3290* switch.
Data source: [257, 258]

6.4 Specific Security Threat Protections

General Protection Steps

The IEC 62351 standard [88] defines protection steps for general applications. These steps can be used in SDN security-enhancing applications to mitigate threats analysed in section 5.3.3. The steps include:

1. **Deterrence and delay** - tries to avoid, or at least delay an attack.
2. **Detection of attacks** - if the attack was not avoided, it has to be detected, so subsequent security measures can react.
3. **Assessment of attacks** - evaluates severity of the attack. Based on this evaluation, the next steps can apply various measures.
4. **Communication and notification** - relevant users and operators have to be notified about the attack.
5. **Response to attacks** - the system can perform automatic actions, or only give the system operators option to take a manual action. A proper and timely response can stop the attack and prevent further damage and future attacks.

The following section describes applicability of these steps to mitigate the described threats. Only those attacks, where SDN can be used completely, or at least partially, will be considered. Also, there are two following considerations, which apply for the SDN:

The first part of the first step - *deterrence* - is common for all types of attacks. It consists of strong and legally appropriate legislations available in all communication channels of the company. When an attacker tries to attack a company's assets, he should be notified about legal consequences. This includes *banner of the day messages* on all access ports (console ports, virtual links, and remote logins) and warning messages on websites.

The fourth step - *communication and notification* - is also the same for all analysed attacks. The application must notify operators via a defined communication interface. An operator should see the attack severity, attack source, and applied action (if it was applied automatically).

These two steps are therefore skipped in the following discussion. Additionally, the theoretical background of all attacks will be no further discussed as it was already described in section 5.3.3.

Protection Against Availability Attacks

Delay - the application slows down the attack by forwarding all the first packets of a new flow to the SDN controller. This behaviour corresponds to the reactive flow rule insertion and will be used, until the controller decides, how to forward the future packets from the flow. The controller must be dimensioned to handle the initial burst of traffic. This can be achieved by implementation of distributed architecture, in which multiple controllers can load-balance the traffic.

If the amount of traffic is too high, it can be blocked immediately by the receiving forwarding device. Alternatively, this flow can be moved to a special output queue. This queue can have QoS policies set to slow down the traffic by limiting the number of packets transmitted over a certain time period.

Detection - is typically based on collected traffic statistics. If these parameters exceed specified values, DoS attack is detected. The most common statistics are: number of packets and amount of bytes received from a single source per defined time interval.

Assessment - severity of DoS is assessed based on the collected statistics of the flow. Different levels of DoS attacks can be defined for various amounts of traffic. For each level, there can be a different predefined action, which will be applied.

Response - the application should automatically take appropriate actions to stop the DoS attack. This can be accomplished by inserting *deny* rules (set to drop all the traffic from the specified source). Based on the attack severity, these rules can be automatically inserted across all the networking devices. This will save the controller load and speed up the decision process in the case, that the attacker would perform the same attack via a different forwarding device.

It is also important to consider an appropriate composition of the blocking rule. For example, in some cases, blocking all communication from the attacker might not be the best solution. If the attacker uses a legitimate device, which transmits critical data, and this device is "only" partially under control of the attacker, transmission of this data should be left unmodified. Only the DoS attack packets should be dropped. But especially in this case, operators should be notified about the attack and about the possibility of a critical data hijack.

Operators should have an option to override the automatically taken action. But in the case of a severe attack, they should be informed and warned if they try to re-allow legitimately blocked traffic. Such an action could result in collapse of the system.

Protection Against Broadcast Tampering

Delay - all the broadcast messages can be set to be forwarded to the controller, for a more detailed analysis. This can be achieved with the default flow rule.

Detection - based on the message content and its source, it can be determined, if the message is legitimate, or if it is an attack. A broadcast message can be calculated as:

$$BroadcastAddress = DestinationAddress \vee SubnetMask \tag{6.1}$$

Note: symbol $\vee$ represents *bitwise OR*. The broadcast address is then calculated by setting 1's on all bits in host's IP address, which are located on the 0's positions of the subnet mask. This can be calculated with the following formula:

$$BroadcastAddress = NetworkAddress + [255 - SubnetMask] \tag{6.2}$$

The calculated broadcast address can then be compared with the destination address of the message to determine, if it is broadcast or not.

Assessment - the attack's severity depends on the number of messages, the message content, and used broadcast address. Not all the broadcast messages have the same scope. For example, broadcast address *192.168.10.31/30* will be destined to a maximum of 2 devices, while broadcast address *192.168.10.255/24* will be destined to up to 254 devices.

Response - the controller can decide, how to process the broadcast message. The controller can forward the message to all devices, forward it only to selected ones, or discard it completely. Additionally, a flow rule can be inserted into the affected switch, to process the similar subsequent messages directly in hardware.

Protection Against Brute Force Attacks

Delay phase is the same as in availability attacks protection.

Detection - unlike in availability attacks, which are configured to monitor all traffic, the brute force protection typically monitors only certain devices and applications. The detection therefore targets mostly destination IP addresses, and TCP ports.

Assessment - number of packets, or amount of traffic, which would be classified as a brute force attack might be generally much lower than for a DoS attack. For example, more than 3 login attempts within 10 seconds might already trigger the protection.

Response - taken action can generally be less restrictive than in DoS attacks. For example, a *dynamic deny* rule can be inserted with a defined *idle timeout.* If the attack stops for the defined period, the blocking rule will be automatically deleted and traffic re-allowed. As in DoS attacks, these actions should be taken automatically.

Protection Against Cross-platform Attacks

Delay - the typical implementation of this protection establishes virtual layers a priori using proactive flow rule insertion, so used rules do not allow any interaction between flows from different domains. Therefore, there is no need to delay potentially dangerous traffic. Domain separation might be determined based on MAC addresses, IP addresses, traffic types, or any other value in a header field. If the reactive flow rule insertion is used, traffic from a new flow must be firstly forwarded to the controller.

Detection - cross-platform traffic can be detected only if it is sent to a controller. The controller must compare parameters of the flow to determine, if the traffic is legitimate, or if it is trying to reach a different domain. If the flow rules are already established, such traffic would be blocked automatically.

Assessment - each layer can have defined priority and level of protection. An attempt to reach a specific layer can then be classified.

Response - if the controller detects an attempt to perform a cross-platform attack, it can insert a deny rule for this traffic flow. Most often, no action is necessary, as the attack would be stopped automatically on the first forwarding device.

Protection Against Experimental Features

This protection uses the same concept as the cross-platform attack protection. The individual protection steps are therefore not described. The only difference is, that flow rules in this protection must be strictly established before the experimental traffic is allowed into the network.

Protection Against Malware Threats

Delay - a suspicious traffic is sent to the SDN controller for data analysis.

Detection - the SDN controller must examine the packet payload. Content of the packet is compared to a database with known malware threats (or their signatures). A successful detection of a malware is possible only if the payload encryption is not used.

Assessment - a malware can be classified by its severity level, which is found in the malware database. According to this level, a specified action can be performed.

Response - the controller can automatically block all the traffic from the infected flow, but that could also disable legitimate traffic. A more appropriate action would be to mirror the traffic to an experimental network for further inspection. All the following traffic from the flow should go via the controller for detailed analysis. This would be accomplished by inserting a specific flow. This action would however increase the controller's load and increase latency of the affected flow.

Protection Against Routing Attacks

Protection against these attacks can vary, as there are more types of these attacks. The following steps describe the protection against *falsification attacks* performed with a malicious router.

Delay - a malicious router would probably firstly try to establish a communication with the controller, which would delay the attack by default. The router could however also start to immediately send malicious messages to neighbouring devices. The prevention against this behaviour is to shutdown all unused ports by default. Links from legitimate devices should be protected by *switchport security features* (allow only specified MAC addresses and limit the number of connected devices).

Detection - the malicious router can be detected by the controller, if the device is not expected to appear in the network. This can be easily confirmed, if the router does not have the corresponding ID. Neighbouring forwarding devices can also detect a new device via LLDP (Link Layer Discovery Protocol). They can then notify the controller.

Assessment - attack severity can depend on location of the malicious router and the amount of traffic, which it tries to send into the network.

Response - the OpenFlow connection between the malicious device and the controller would not be established. Additionally, the controller can send a command to neighbouring forwarding devices to shutdown the ports leading to the malicious device. These actions would effectively isolate the malicious device, and therefore prevent any damage to the network.

Partial Protection Against Eavesdropping

The attack is completely passive, so delay, detection, and assessment steps cannot be used.

Response - forwarding all traffic via the controller can decrease the chance of eavesdropping, if a message is being sent straight from the first forwarding device to the last (via the controller). This will make the forwarding route shorter and limit the number of places, where the attack can be performed. It requires use of the default flow rule and routing implemented on the controller. The controller would then have to use *packet-out* message to forward the original packet to the destination forwarding device. The problem is, that this forwarding would have to be used continuously, which would negatively affect the network performance. Moreover, the attack could still happen between end devices and the first / last forwarding devices, and also between forwarding devices and the controller, if an *in-band* connection is used.

Partial Protection Against GPS Spoofing

Delay - each message from a selected flow is forwarded via an SDN controller, which compares the location information in the message with the previously cached messages.

Detection - if the message's location differs from the previous ones by more than a specified value, the GPS spoofing attack is detected. Unfortunately, there is always chance of a *false negative* detection, for example in the case, when the attacker sets a nearby location, which is not so suspicious.

Assessment - severity of the attack can be evaluated by the location offset between the packet's location and previously cached positions.

Response - the controller can decide to drop the affected traffic immediately, or to forward it to an experimental network for further analysis.

Partial Protection Against MitM

This attack is very similar to eavesdropping, but because it is not passive, it can be detected.

Delay - in defined time periods, latency statistics of each flow are compared on the controller with previously cached data. This comparison needs to be conducted on the controller and would therefore increase latency of messages chosen for inspection.

Detection - is very hard and can be successful only in ideal conditions. A significant deviation between the current latency and previously stored ones can indicate the attack. On the other hand, there is high risk of a *false positive detection*, as the deviation can be caused simply by an increased network load.

Assessment - severity of the attack can be classified based on the message payload, or potentially changed variables within the message headers.

Response - an operator, or the application itself, can decide to enable direct forwarding of messages from the selected flow via the controller. This would result in higher controller load, so cautious monitoring should be performed. In the case, that a defined threshold is exceeded, traffic forwarding should return to normal.

Partial Protection Against Node Attacks

The following protection mechanism can mitigate only *remote* node attacks.

Delay - all new incoming flows to a device should go via an inspection on the controller, where outgoing traffic from the device is logged. The controller can then compare, if the traffic originates from the node device or not.

Detection - if the traffic comes from an unexpected source, or to a destination, which it should not, it can be detected as suspicious.

Assessment - based on the traffic flow (source and destination), traffic protocol, and amount of traffic, the attack can be classified.

Response - based on set rules, the attack can be blocked, and a new flow rule inserted into the affected forwarding device to block any future similar flows in hardware.

Partial Protection Against Personnel Issues

Delay - traffic monitoring has to be utilized by data forwarding via a controller. This also slows down a potential attack.

Detection - the controller can detect a malicious behaviour based on comparison with the typical baseline traffic. For example, if the amount of traffic is significantly higher, if the traffic does not use encryption, or if a source is different than expected, an attack might be detected.

Assessment - severity of the attack might be classified based on its damage potential. For example, use of an older version of a protocol might not be a critical problem, but traffic containing a malware surely is.

Response - the controller can decide, how to process the flow. The most restrictive action would be to drop all traffic from the source. The less restrictive would be to drop only the traffic flow associated with the attack. Traffic can be also slowed down (if DoS attack is detected), or forwarded to a specific network (a concept of *honeypot*[3]).

Partial Protection Against Replay Attacks

Delay - each message from a selected flow is forwarded via an SDN controller, which compares timestamp in the message with the current time.

Detection - if the message's timestamp differs from the local time by more than a defined interval, the replay attack is detected. Unfortunately, there is always chance of a *false positive* detection, for example if network is more loaded than during typical conditions.

Assessment - network statistics can be checked for the current load. This can limit the *false positive* detection. There are no specific assessment levels of these attacks.

Response - is the same as in GPS spoofing.

Partial Protection Against Specific Communication Links

Implementation of a new security protocol, developed solely in SDN, might be a difficult process, but it can help with improving security on otherwise unsecured connections. This protection is therefore not aimed at stopping an attack, but to make security possible even on specific links. The protection steps are therefore not discussed.

An example of this approach might be utilization of an otherwise unused header field to insert an encrypted value (a concept of *secured nodes* described in *Security Requirements* section 6.2). Only if the value is correct, the traffic is allowed. This assumes, that an attacker does not know the correct value. Such a solution does not increase the packet size and might be therefore supported on all the links along the path.

[3]Honeypot is a security mechanism, which presents fake data to the attacker. Data appears valid, so the attacker continues with the attack. This provides enough time for detailed attack analysis and at the same time, protects the legitimate part of the network.

7. SC Protection System

This section describes a use case application for securing Smart Cities using SDN. The application is an example of an SSC-SDNA described in chapter 6 and it has the defined features. The application is called *The Smart City Protection System* and in the following text, it will be referred to as the SCPS, or simply *the application* or *the system.*

7.1 Use Case Smart City

The SCPS had to be tested in a use case SC scenario. This section describes the process of creating such a scenario with a fictitious topology.

7.1.1 Smart City Introduction

Choosing a City

The *City of Lancaster* in the United Kingdom was chosen as an inspiration. Lancaster is a city and non-metropolitan district in the north-west of England with a population of about 140 000. The city is composed from several districts: City Centre (*Lancaster*), *Morecambe*, *Heysham*, and *Scotforth.* The other smaller villages will not be further covered in this work. Lancaster currently does not use any SC applications, but its features make the SC implementation attractive. These features include [259]:

- **Ecological friendliness** - Lancaster is one of the few British cities, which heavily emphasizes bicycle transportation. The city has many cycle paths, which connect major districts. Electricity is provided mainly from the Heysham nuclear power station and several wind farms. Lancaster also provides several options to charge EVs. The SC implementation could therefore cover electric vehicle stations and smart lighting.
- **Flood risks** - Lancaster is located around the river *Lune* and many smaller streams. Due to its location and frequent rains, the city is prone to floods. The SC could provide CCTV and smart environmental sensors for water level monitoring.
- **Geographical diversity** - Lancaster district is located in the river *Lune* delta and mostly on hilly ground. These properties present natural obstacles and make some districts difficult to access. This can complicate, for example, waste collection. The SC could provide smart waste sensors, which provide notification about containers status. This could reduce the number of rides necessary to collect the waste.
- **Heavy traffic** - Lancaster is located next to the *M6 motorway* and it is intersected by the *A6* - another busy artery. Both of these routes and Lancaster city centre, suffer from regular traffic jams, especially during rush hours. The SC could improve these conditions by implementation of smart traffic lights and smart parking.
- **Tourist place** - Lancaster is a popular tourist destination as it is easily accessible and located in near proximity of one of the England's most famous national parks - *Lake District.* The SC could provide tourist information by smart information providers and smart access points.

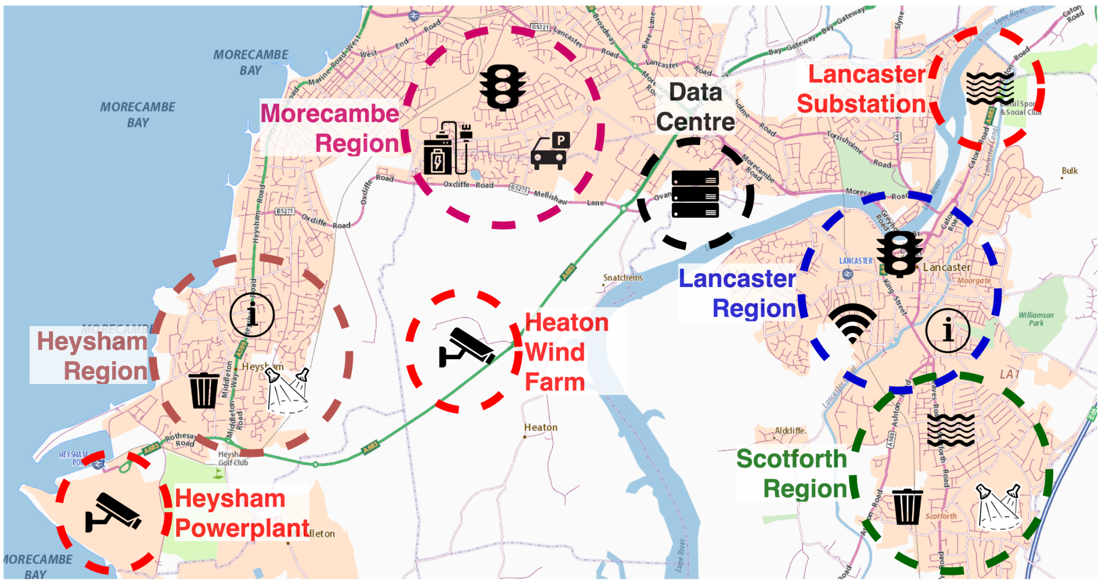


Figure 7.1: Use Case Smart City - Regions and Applications

Choosing SC Applications

The listed SC applications, which were found based on Lancaster features, will be assigned to specific districts of the SC. Additionally, a SG network will be included as a part of the SC. The SG area will include: Lancaster substation, Heaton wind farm, and Heysham powerplant.

The SC will therefore cover the following regions and applications:

- **Lancaster** - smart access points, smart information providers, and smart traffic lights.
- **Morecambe** - electric vehicle stations, smart parking, and smart traffic lights.
- **Heysham** - smart information providers, smart lighting, and smart waste.
- **Scotforth** - smart environmental sensors, smart lighting, and smart waste.
- **Lancaster substation (SG)** - smart environmental sensors.
- **Heaton wind farm (SG)** - CCTV.
- **Heysham powerplant (SG)** - CCTV.
- **SC data centre** - servers.

The physical layout of these regions including the SC applications is shown in Figure 7.1.

7.1.2 Smart City Topology

The network topology corresponds to the physical regions. The SC has a centralized data centre (DTC), which contains two servers. The data centre connects all regions and areas of the SC and therefore uses the *star network topology.*

The main regions of the SC (Lancaster, Morecambe, Heysham, and Scotforth) each contain three switches connected in the *ring topology.* This topology has an advantage in redundancy. However, because the SCPS does not implement STP (or any other loop-prevention technology), redundant links are disabled.

SG regions are connected via a single switch. In a real deployment, this device would have to be virtualized, or duplicated to provide redundancy. Interconnecting links could be also redundant and bundled into a single logical link.

Each regional switch acts as the access layer device and it provides connectivity to SC nodes. The detailed topology of the SC is shown in Figure 7.2. The following shortcuts are used for the SC applications (numbers in the brackets represent the system ID, which will be used in the following sections):

- **SER** - server (0)
- **SAP** - smart access point (1)
- **SIP** - smart information provider (2)
- **STL** - smart traffic lights (3)
- **SPS** - smart parking sensor (4)
- **EVS** - electric vehicle station (5)
- **SLS** - smart lighting system (6)
- **STS** - smart trash sensor (smart waste) (7)
- **SES** - smart environmental sensor (8)
- **CTV** - CCTV (9)
- **SCD** - SCADA (10)

The regions use the following shortcuts and IDs: data centre (DTC, 1), Lancaster (LAN, 2), Morecambe (MOR, 3), Heysham (HEY, 4), Scotforth (SCO, 5), Lancaster substation (LSS, 6), Heysham wind farm (HWF, 7), Heysham power plant (HPP, 8).

The topology also contains a single virtual tunnel between Lancaster substation (LSS) and the SCADA server (SCD001) located in the Heysham power plant.

The *Mininet* script for building the entire topology is shown in Attachment A - SC Topology Script.

IPv4 Addressing

IPv4 addresses use subnet *10.0.0.0/8* and are composed according to this scheme:

10.REGION_ID.SYSTEM_ID.NODE_ID

For example, device EVS002 in Morecambe has IP address: *10.3.5.2*, where:

- **10** = Network (**10**.0.0.0/8)
- **3** = Region ID (Morecambe)
- **5** = System ID (electric vehicle station)
- **2** = Node ID (002)

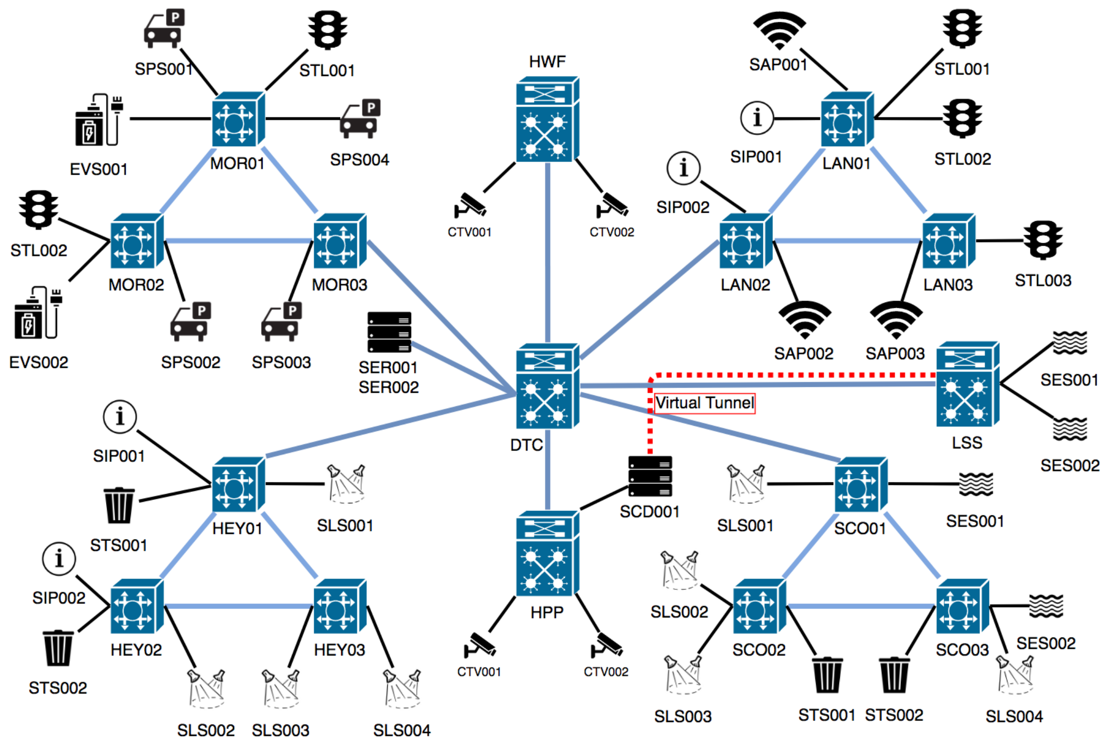


Figure 7.2: Use Case Smart City - Network Topology

MAC Addressing

MAC addresses use a simplified version of all 0's, where only the last octet is used for addressing. This method is ideal for debugging and troubleshooting. There are always 10 MAC addresses reserved for each region. The data centre has addresses within 01 - 10 range, Lancaster 11 - 20, Morecambe 21 - 30, Heysham 31 - 40, Scotforth 41 - 50, Heysham wind farm 51 - 60, Lancaster substation 61 - 70, and Heysham power plant 71 - 80. MAC addresses within each region are assigned sequentially.

IPv6 Addressing

IPv6 is used only for CCTV nodes and the main server (SER001). These devices use both IPv4 and IPv6. The assigned addresses are:

- **SER001_DTC** - FC00::99/64
- **CTV001_HWF** - FC00::1/64
- **CTV002_HWF** - FC00::2/64
- **CTV001_HPP** - FC00::11/64
- **CTV002_HPP** - FC00::12/64

All nodes and their detailed addressing information are shown in Table 7.1. Switch numbers represent a forwarding device within a region, to which a device is connected.

Table 7.1: Use Case Smart City - Topology Addressing

Region	Device ID	Switch	IPv4	MAC Address
DTC	SER001	1	10.1.0.1	00:00:00:00:00:01
	SER002	1	10.1.0.2	00:00:00:00:00:02
LAN	SAP001	1	10.2.1.1	00:00:00:00:00:11
	SIP001	1	10.2.2.1	00:00:00:00:00:12
	STL001	1	10.2.3.1	00:00:00:00:00:13
	STL002	1	10.2.3.2	00:00:00:00:00:14
	SIP002	2	10.2.2.2	00:00:00:00:00:15
	SAP002	2	10.2.1.2	00:00:00:00:00:16
	SAP003	3	10.2.1.3	00:00:00:00:00:17
	STL003	3	10.2.3.3	00:00:00:00:00:18
MOR	STL001	1	10.3.3.1	00:00:00:00:00:21
	SPS001	1	10.3.4.1	00:00:00:00:00:22
	SPS004	1	10.3.4.4	00:00:00:00:00:23
	EVS001	1	10.3.5.1	00:00:00:00:00:24
	STL002	2	10.3.3.2	00:00:00:00:00:25
	SPS002	2	10.3.4.2	00:00:00:00:00:26
	EVS002	2	10.3.5.2	00:00:00:00:00:27
	SPS003	3	10.3.4.3	00:00:00:00:00:28
HEY	SIP001	1	10.4.2.1	00:00:00:00:00:31
	STS001	1	10.4.7.1	00:00:00:00:00:32
	SLS001	1	10.4.6.1	00:00:00:00:00:33
	SIP002	2	10.4.2.2	00:00:00:00:00:34
	STS002	2	10.4.7.2	00:00:00:00:00:35
	SLS002	2	10.4.6.2	00:00:00:00:00:36
	SLS003	3	10.4.6.3	00:00:00:00:00:37
	SLS004	3	10.4.6.4	00:00:00:00:00:38
SCO	SLS001	1	10.5.6.1	00:00:00:00:00:41
	SES001	1	10.5.8.1	00:00:00:00:00:42
	SLS002	2	10.5.6.2	00:00:00:00:00:43
	SLS003	2	10.5.6.3	00:00:00:00:00:44
	STS001	2	10.5.7.1	00:00:00:00:00:45
	STS002	3	10.5.7.2	00:00:00:00:00:46
	SLS004	3	10.5.6.4	00:00:00:00:00:47
	SES002	3	10.5.8.2	00:00:00:00:00:48
HWF	CTV001	1	10.6.9.1	00:00:00:00:00:51
	CTV002	1	10.6.9.2	00:00:00:00:00:52
LSS	SES001	1	10.7.8.1	00:00:00:00:00:61
	SES002	1	10.7.8.2	00:00:00:00:00:62
HPP	CTV001	1	10.8.9.1	00:00:00:00:00:71
	CTV002	1	10.8.9.2	00:00:00:00:00:72
	SCD001	1	10.8.10.1	00:00:00:00:00:73

7.2 SCPS Development Process

The development process described in section 6.1 is designed for real and extensive applications. Therefore some of its phases were significantly simplified in the SCPS, as it is only a use case verification application.

7.2.1 Planning Phase

The practical goal of the application was to cover several areas of protections, defined in section 6.4. Based on the assessment conducted within a security life cycle process (described in section 7.2.6), the following areas were selected:

- DoS protection
- Brute force protection
- Cross-platform protection
- Traffic filtering (ACL)
- Secured nodes

7.2.2 Analysis Phase

The analysis phase mainly collects system requirements. In the SCPS, there were no real users of the system, so the requirements were selected from the lists presented in section 6.2. Because the system is just a use case example, only specific requirements were implemented. Implementation of all requirements would significantly complicate the development process, and in some cases, even be impossible within the emulated environment.

Implemented Requirements

These requirements were selected and implemented in the SCPS:

- **Interoperability** - the system uses open-source protocols and common standards. Southbound API uses OpenFlow; and northbound API uses REST with JSON for data exchange.
- **Usability** - UI of the system is realized as a web application. It uses a responsive template and can be access from any device. It also uses common and consistent control elements.
- **Maintainability** - the system is composed from dedicated modules, which can be easily maintained and expanded. This includes main components integrated within the SDN controller and web application as well.
- **Maximum first packet latency** - the system supports creation of virtual layers and links, which insert OpenFlow rules reactively to minimize latency.
- **Statistics refresh rate** - for maximum responsiveness, time intervals of 1 second were chosen for statistics collection. These statistics provide information about present OpenFlow rules.
- **Virtualization** - the system supports virtual separation of traffic via the cross-layer protection module.

- **Compatibility** - the system uses standard APIs of the controller for data exchange, and provides common interfaces to access its data.
- **Effectivity, efficiency, satisfaction** - the system uses only necessary libraries, data structures, and other resources. The application can be used effectively due to the responsive design of the UI.
- **Functional suitability** - all the features defined in the design phase of the system were implemented.
- **Maintainability and portability** - is automatically achieved by use of an SDN controller.
- **Backup communication paths** - the system can support redundant paths via the cross-platform protection module, which can create virtual tunnels.
- **Proactive rule insertion** - by default, this behaviour is not used in the system, but can be enabled in the cross-platform protection module. This requires a definition of virtual links or layers.
- **Virtual network layers** - are implemented in the cross-layer protection module.
- **Traffic filtering** - the system uses a simple ACL, which can block or allow selected traffic.
- **Secured nodes** - the functionality allows creation of a database of nodes (MAC addresses) and their flow labels. The tool then automatically drops all other IPv6 traffic.

Not Implemented Requirements

The following requirements were not implemented in the SCPS:

- **Recoverability** - backup and recovery of the system's files were not required in the tested scenario.
- **Other performance requirements** - backup communication paths, connectivity-loss detection delay, load-balancing, maximum typical end-to-end latency, minimum bandwidth, minimum throughput, OpenFlow rules performance, QoS, support of critical end nodes, controller server's requirements, and forwarding device requirements were not considered, due to use of an emulated environment.
- **Other privacy requirements** - aggregation, anonymization and minimization, enforcement and demonstration, privacy protection, and transparency were not implemented, because of lack of real SC data. The simulated topology contains only virtual nodes, which use mostly *ping* messages for traffic simulation. This data therefore cannot be protected from the privacy point of view.
- **Reliability** - the SCPS in the current version uses RYU controller, which does not support the distributed architecture. Therefore even additional measures for advanced reliability were not considered.
- **Safety** - due to the lack of implemented security and privacy requirements, safety in cyber-physical networks deployment is not considered.
- **Distributed controllers architecture** - the architecture is not supported by the selected controller (RYU).

- **Failover mode and hybrid OpenFlow** - used software switches in the emulation do not support these features.
- **Management connectivity requirements** - connections between forwarding devices and the controller, in the emulated environment, are automatically realized via virtual *out-of-band* links.
- **Security requirements** - all the safety requirements, except secured nodes, traffic filtering, and virtualization; were not implemented due to the testing nature of the system.
- **Supporting technologies** - IAM, key management system, reputation and trust systems, TPM, and VPN were not used in the system.

7.2.3 Design Phase

The main goal of the design phase is to select architectural components used in the system. The following ones were selected for the SCPS.

Choice of Controller

The following open-source controllers were considered for the system development: RYU, Floodlight, OpenDaylight, and ONOS. All these controllers support OpenFlow in recent versions, so they could be considered (details are explained in *Choice of APIs* section below). The **RYU controller** was eventually chosen for several reasons. The main reason was its simplicity, which allowed a relatively rapid development of the system. RYU also has low hardware requirements and therefore testing of the system could be performed even on not very powerful devices. Lastly, advanced features (distributed architecture, redundancy, and advanced security) of other controllers were not needed, as these functions were omitted in the use case scenario.

Choice of Controller's Architecture

The selected RYU controller supports only the centralized architecture. This was not a problem, as the system is only for demonstration and validation purposes and therefore does not have to provide extra redundancy and reliability.

Choice of APIs

OpenFlow was selected as the most common and complete southbound protocol. The OpenFlow version was selected, based on the application requirements:

- **Support of IPv6** - OpenFlow 1.2+
- **Duration field for statistics** - OpenFlow 1.3+

OpenFlow 1.3 was therefore selected as the minimal version, which supports all the required features.

Choice of Application Mode

The system had to be implemented both as an *application* and as the controller's *module*. The custom module was needed, because the RYU controller does not provide all needed features. The application part then provides a relatively complex presentation layer, which would not be possible to implement (efficiently) as one of the controller's modules.

The benefit of this architecture is in its scalability, as any future modification can be easily incorporated. On the other hand, the system is tightly linked to the RYU controller and its portability to other controllers is limited. It would require complete rewriting of the module part.

Application Architecture

For clarity purposes, the detailed application architecture with all the connected diagrams is described in section 7.2.7.

7.2.4 Implementation Phase

Implementation details are described in a separate section 7.3. Due to the system's nature, dedicated documentations were omitted. This chapter can be considered to be a reduced-size documentation.

7.2.5 Testing, Integration, and Maintenance Phases

These phases were also significantly simplified. No real deployment was conducted, so the entire system was tested only in the emulated environment of the Mininet. The verification of the system was performed mostly from the performance perspective and it is described in section 7.5.

7.2.6 Security Life Cycle

The security life cycle is aimed mainly at large scale business projects and therefore its applicability in the use case application was limited. For this reason, only the reduced assessment phase was conducted. This phase was completed during the planning phase, when the system scope was being determined. This timing allowed more precise specification of the scope. The details gathered in the assessment phase are summarized in Table 7.2. Assets are ordered by the overall risk level. Costs connected with successful attacks are omitted as their estimation would depend on the real application specifics.

Based on the analysed assets, the attack probability and potential risks in smart cities scenarios, the most important protection methods were identified as: **brute force, DoS, firewall / ACL**, and **cross-layer** for virtual separation.

The following steps of the security life cycle were not implemented, or implemented only partially. The policy phase aims at employees, which were not present in the use case. The deployment phase was partially conducted within the implementation phase of the development process. The training phase also needs employees, so it was completely omitted. Monitoring of the system was partially implemented within the system's modules (traffic monitoring, DoS and brute force monitoring), however no auditing was used.

7.2.7 SCPS Architecture

The SCPS is composed from two main parts: the *SDN controller modules* and the *web application*. This architecture is shown in Figure 7.3.

Table 7.2: SCPS Assets (Security Life Cycle)

Assets	**Attack Probab.**	**Potential Risks**	**Security Requirements**
		Very High Risk	
SCPS app.	high	high, app. only	AAA, BF, DoS, HTTPS
IoT nodes	high	medium, local	FW, H, V
		High Risk	
Data centre ser.	medium	high, global	AAA, BF, DoS, FW
Access l. dev.	medium	medium, local	H, switchport security
SDN server	low	high, global	AAA, BF, DoS, H
		Medium Risk	
Core l. dev.	low	high, global	H, DoS, V
SCADA server	low	high, SG	AAA, BF, FW

H = hardening (configuration and physical if available), V = virtual separation

1. SCPS Controller Modules

The main part of the system is the controller subsystem, which consists of several modules.

The main module is based on the *simple switch module* included in the RYU controller. The functionality of the module was considerably extended, so it also provides communication between other modules. As a southbound protocol, OpenFlow 1.3 is used for the communication with the networking devices.

The polling module contains a timer, which periodically sends OpenFlow requests to forwarding devices. The main request is for the content of their OpenFlow tables. Replies are then handled by the main module. The main module runs an instance of the polling class in a new thread, so the timer does not block the entire system. The module uses the same version of OpenFlow (1.3) as the main module.

The library module provides specific features to the main module. It can create flow rules, load files, save files, etc.

The REST API module contains a class, which provides an interface to external applications. The module handles HTTP requests (either GET or POST) and sends replies back.

The REST API module provides the following functions, which can be accessed via HTTP:

- **sc/flows/*dpid*** - returns flow table of the switch specified by the DPID parameter.
- **sc/rule_delete** - takes a match rule in the POST argument and deletes the corresponding flow rules (if they exist).
- **sc/secured_nodes** - returns the database (in form of a list) of secured nodes. The list contains devices MAC address and IPv6 flow label.
- **sc/dpids** - returns a list of DPIDs of all connected switches.
- **sc/settings** - takes a JSON file in the POST argument, containing all settings of the application. The method returns applied settings in a new JSON file.

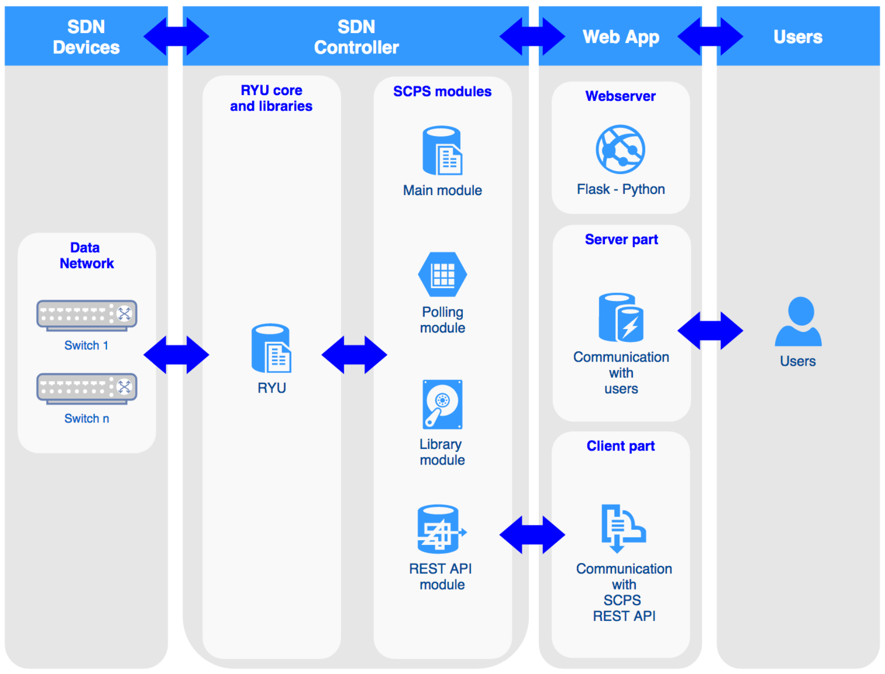


Figure 7.3: SCPS Architecture

- **sc/status** - returns status information of the SCPS. This includes notification messages, number of connected switches, users, etc.

2. SCPS Web Application

The second most important part of the system is the web application, which communicates with SCPS modules via the REST API module. The web application provides a GUI, which can be accessed by clients using any web browser and any device. The application acts as a server and a client at the same time.

The server part provides access to users. They can connect to the application as to any other website. In the testing environment, the application runs on the localhost address and TCP port 5000.

The client part connects to the REST API of the controller's part. In this mode, the web application sends HTML requests and receives replies as a normal client. Replies are then processed, forwarded to the server part, and sent back to clients.

The web application is built on top of the *Flask framework*[1], which provides a lightweight webserver functionality. The graphical template is based on *ElaAdmin Template*[2]. For the data exchange with the controller's modules, JSON format is used.

Class Diagram

The class diagram of controllers modules is shown in Figure 7.4. The diagram shows only the most important attributes. All the constants are omitted due to space restrictions.

The web application part of the system does not contain any classes and therefore is not included in the class diagram. The main file of the web application part contains only functions, that are registered to handlers, which are in turn managed by the *Flask framework*.

[1]Available: `http://flask.pocoo.org/`
[2]Available: `https://github.com/puikinsh/ElaAdmin`

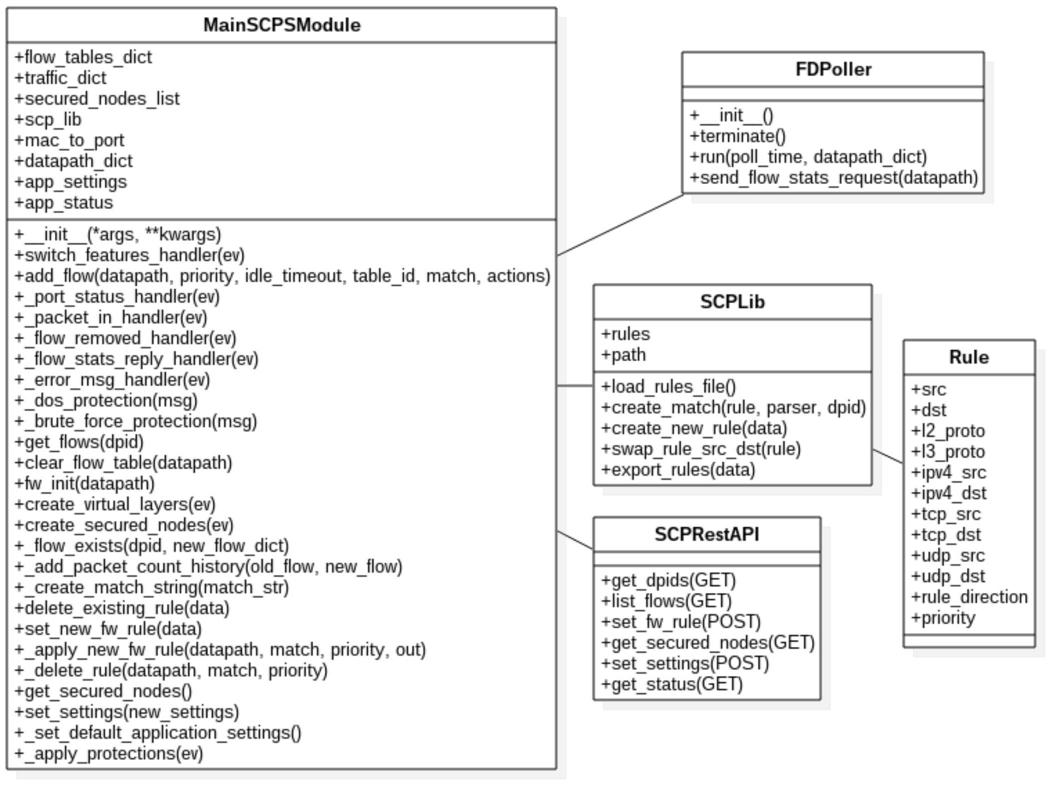

MainSCPSModule
+flow_tables_dict
+traffic_dict
+secured_nodes_list
+scp_lib
+mac_to_port
+datapath_dict
+app_settings
+app_status
+__init__(*args, **kwargs)
+switch_features_handler(ev)
+add_flow(datapath, priority, idle_timeout, table_id, match, actions)
+_port_status_handler(ev)
+_packet_in_handler(ev)
+_flow_removed_handler(ev)
+_flow_stats_reply_handler(ev)
+_error_msg_handler(ev)
+_dos_protection(msg)
+_brute_force_protection(msg)
+get_flows(dpid)
+clear_flow_table(datapath)
+fw_init(datapath)
+create_virtual_layers(ev)
+create_secured_nodes(ev)
+_flow_exists(dpid, new_flow_dict)
+_add_packet_count_history(old_flow, new_flow)
+_create_match_string(match_str)
+delete_existing_rule(data)
+set_new_fw_rule(data)
+_apply_new_fw_rule(datapath, match, priority, out)
+_delete_rule(datapath, match, priority)
+get_secured_nodes()
+set_settings(new_settings)
+_set_default_application_settings()
+_apply_protections(ev)
FDPoller
+__init__()
+terminate()
+run(poll_time, datapath_dict)
+send_flow_stats_request(datapath)
SCPLib
+rules
+path
+load_rules_file()
+create_match(rule, parser, dpid)
+create_new_rule(data)
+swap_rule_src_dst(rule)
+export_rules(data)
Rule
+src
+dst
+l2_proto
+l3_proto
+ipv4_src
+ipv4_dst
+tcp_src
+tcp_dst
+udp_src
+udp_dst
+rule_direction
+priority
SCPRestAPI
+get_dpids(GET)
+list_flows(GET)
+set_fw_rule(POST)
+get_secured_nodes(GET)
+set_settings(POST)
+get_status(GET)

Figure 7.4: SCPS Controller Modules Class Diagram

Use Case Diagram

The use case diagram, shown in Figure 7.5, represents activities, which a system operator can perform. The operator interacts mostly with the web application part of the system. All these interactions are displayed in the diagram. The web application processes the user action and forwards a new request to the REST API of the controller's modules. More advanced parameters of the system might be modified by changing variable constants directly in the *scps_main_module.py* file.

Activity Diagram

The activity diagram, shown in Figure 7.6, illustrates the process of displaying the main web page of the system. A user accesses the web page by entering the system IP address into a web browser (by default *127.0.0.1:5000*). The HTTP request is received by the server part of the web application, translated into a new HTTP request and sent to the REST API of the controller's module. This API accesses parameters within the request and calls the appropriate function (*get_flows*) of the main module (*scps_main_module.py*). The function returns flow tables as a *dictionary variable*. The REST API function then validates received data and returns a corresponding HTTP reply. If data length is zero, a 400 error page is generated, otherwise, data is inserted into the JSON format and included in the reply body. The web application receives the reply and checks its response code. If there is a 400 error code, an error page is displayed. If the code is a success (200), data is loaded from the JSON format and parsed. The main webpage is then displayed by calling the *render_template* function.

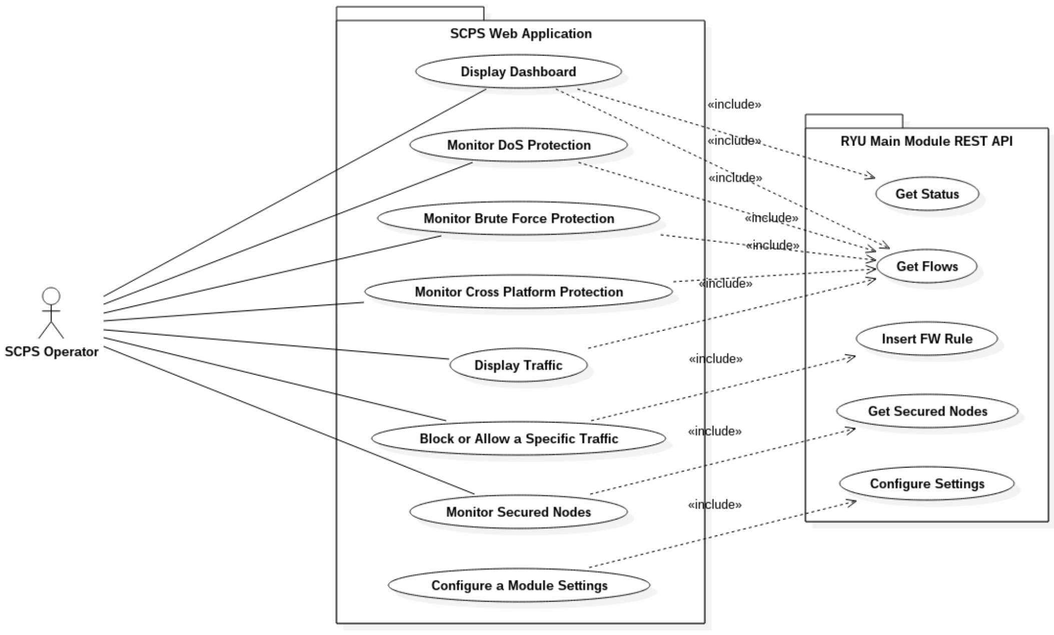


Figure 7.5: SCPS Use Case Diagram

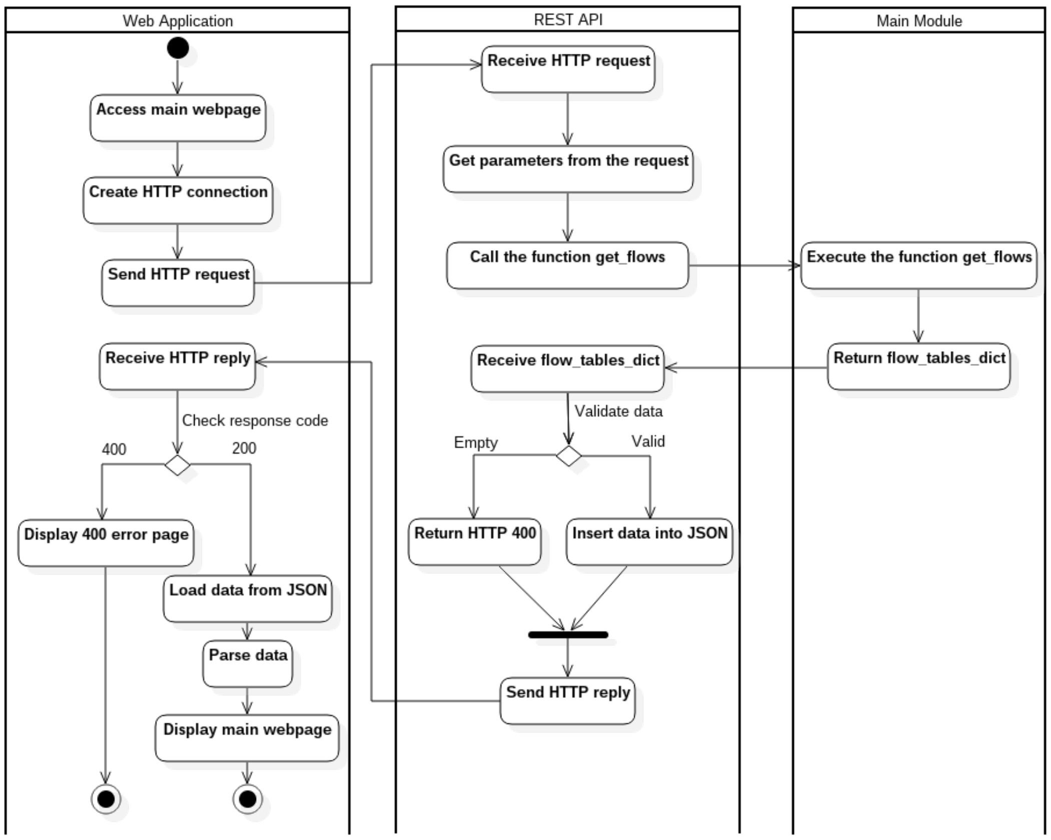


Figure 7.6: SCPS Activity Diagram

7.3 Application Implementation

This section describes the application implementation in detail. This include the environment used for the application development, tools used in connection with the application, structure of source files, and the process of launching the application.

Development Environment

The SCPS was developed in an emulated environment of a *virtual machine*. This allowed continuous development using several devices. The following software was used:

- **Various OSs** - depending on used device (Kubuntu 17.10, Linux Mint 18.3, Windows 10)
- **VirtualBox** - several versions, depending on the host OS
- **Virtual machine:**

  Ubuntu Mate 16.04.3 (Guest OS)

  Mininet 2.2.1

  RYU SDN Controller

  Flask Framework

7.3.1 Used Tools

The following tools were used in the SCPS.

Ubuntu Mate

Ubuntu Mate[3] was chosen, because of its good support of all other used tools. Mate desktop was chosen for its low performance requirements. Mate is a lightweight desktop, so it is ideal for emulated environments, where performance is important. The most recent LTS (Long-Term Support) version was chosen for maximum stability.

Mininet

Mininet[4] is a network emulator, which can run virtual hosts (using a real kernel), switches, and SDN controllers. Mininet is released under the BSD open-source licence and it is ideal for teaching, learning, and testing purposes. Because it allows running of real software switches (including Open vSwitch), an SDN application can be fully tested, before it is deployed in a real network.

Mininet is used for creating the SC topology. All sensor devices and servers are represented by virtualized hosts running Linux. They therefore support common networking tools like ping, iPerf, and even installation of more advanced applications.

[3]Available: `https://ubuntu-mate.org`

[4]Available: `http://mininet.org/`

RYU SDN Controller

The SDN controller was chosen based on requirements described in section 7.2.3. The RYU controller[5] offers the best features for the SCPS example scenario. Its main advantage is simplicity. Use of a more advanced SDN controller, with a more complicated architecture would significantly complicate the development. In that case, the development of an SCPS-scale application would not be feasible by a single person within a reasonable time frame. The SCPS uses RYU's libraries, handlers, and key functionalities from the *simple_switch_13.py* built-in module.

Flask Framework

Flask[6] is considered to be a web development *microframework* - it keeps its core simple and more features can be added by extensions (for example database, validation, or various libraries). Flask uses two main technologies - *Jinja* template engine and *Werkzeug* WSGI toolkit.

Flask is used for the web application part, where it performs server and client functionalities. The Jinja template engine is used mainly for inheritance purposes, which allow easy scalability and extensibility of the web application.

7.3.2 File Structure

File structure of the SCPS is as follows (the *static* folder contains a large number of various html resources and its complete content is therefore not shown):

```
scps
 ├── scps_main_module.py
 ├── sc_lib.py
 ├── fd_poller.py
 └── webapp
      ├── scps_web_app.py
      ├── static
      │    └── css, icons, images, and js resources
      └── templates
           ├── action-fw.html
           ├── action-settings.html
           ├── bf_protection.html
           ├── cross_protection.html
           ├── dos_protection.html
           ├── firewall.html
           ├── main.html
           ├── page-error-404.html
           ├── page-error-500.html
           ├── secured_nodes.html
           ├── settings.html
           └── template.html
```

[5]Available: `https://osrg.github.io/ryu/`
[6]Available: `http://flask.pocoo.org`

7.3.3 Running SCPS

The SCPS system runs completely in the virtual machine. After the machine is started, three terminal windows have to be opened. These windows are used for running the RYU controller, the SC topology, and the web application.

Launching RYU Controller

The RYU controller can be launched by the *ryu-manager* command with specification of an appropriate module. In the SCPS case, it is the *scps_main_module.py*. If forwarding devices do not use the default port for OpenFlow connection, the port must be specified in the *ofp-tcp-listen-port* parameter.

The following command launches the main module of the SCPS within the RYU controller.

```
filip@DT-VM:~/scps$ ryu-manager --verbose scps_main_module.py --ofp-tcp-listen-port 6633
```

Launching SC Topology

Launch of a Mininet topology typically requires a lot of parameters to be set. For this reason, the SCPS uses a script, which includes the complete settings of the topology with all its parameters (including OpenFlow port 6633). Launch is therefore maximally simplified and can be executed by the following command (Mininet requires an administrator privileges).

```
filip@DT-VM:~/scps$ sudo ./lan_sc_topology.py
```

Launching Web Application

The web application uses Flask framework, which can be started by the *flask run* command. To run the correct file (*scps_web_app.py*), it must be firstly exported into Flask. The following two commands will start the web application.

```
filip@DT-VM:~/scps/webapp$ export FLASK_APP=scps_web_app.py
filip@DT-VM:~/scps/webapp$ flask run
```

Note: once the file is loaded into Flask, the web server can be restarted just by stopping the process and executing the *flask run* command again.

7.4 SCPS Functionality

This section describes the functionality of the system. Some of the GUI elements, which are not described, serve only for illustration purposes (and might be used in future versions of the system).

SCPS Workflow

The SCPS works in the following steps:

1. **Application initialization** - the first step is the application initialization, which creates all used variables, creates and starts a new thread for switch polling, and registers the REST API.

2. **Periodical polling** - the thread for statistics polling, calls the function for statistics queries in periodic intervals (1 second by default). This function sends the *FlowStatsRequest* message to all connected switches.

3. **Event handling** - the main module contains handlers for important OpenFlow events, including: switch features, port status, *packet-in*, flow removed, flow statistics reply, and error messages. If any of these events happen, the corresponding function is executed.

4. **REST API handling** - the API listens on defined URLs for any HTML requests. If a request is received, the corresponding function is executed and a reply is sent back to the web application.

5. **Web application communication** - the application runs as a web server and listens for HTML requests from clients. Legitimate requests are processed, and if they require data from the controller, a new HTML request is generated and sent to the REST API of the main module. A valid reply from the main module is then processed (parsed from a JSON file) and passed to the web page for a final rendering.

6. **Web application rendering** - the rendering engine uses a combination of HTML, CSS, Python, Jinja2, and several tools (*jquery, sweetalert, toastr*, etc.) for a dynamic presentation of data.

This application workflow for selected examples is further described in Attachment B - SCPS Workflow. The attachment includes demonstrations of the most important parts of the application code from various source files.

7.4.1 Web Application Model

The model of the web application uses Jinja2 template engine[7]. This template allows better code modularity and scalability. The main navigational parts of the graphical layout, which are the same for all web pages, are defined in a single file *template.html*. Variable content of different web pages then extends this main file.

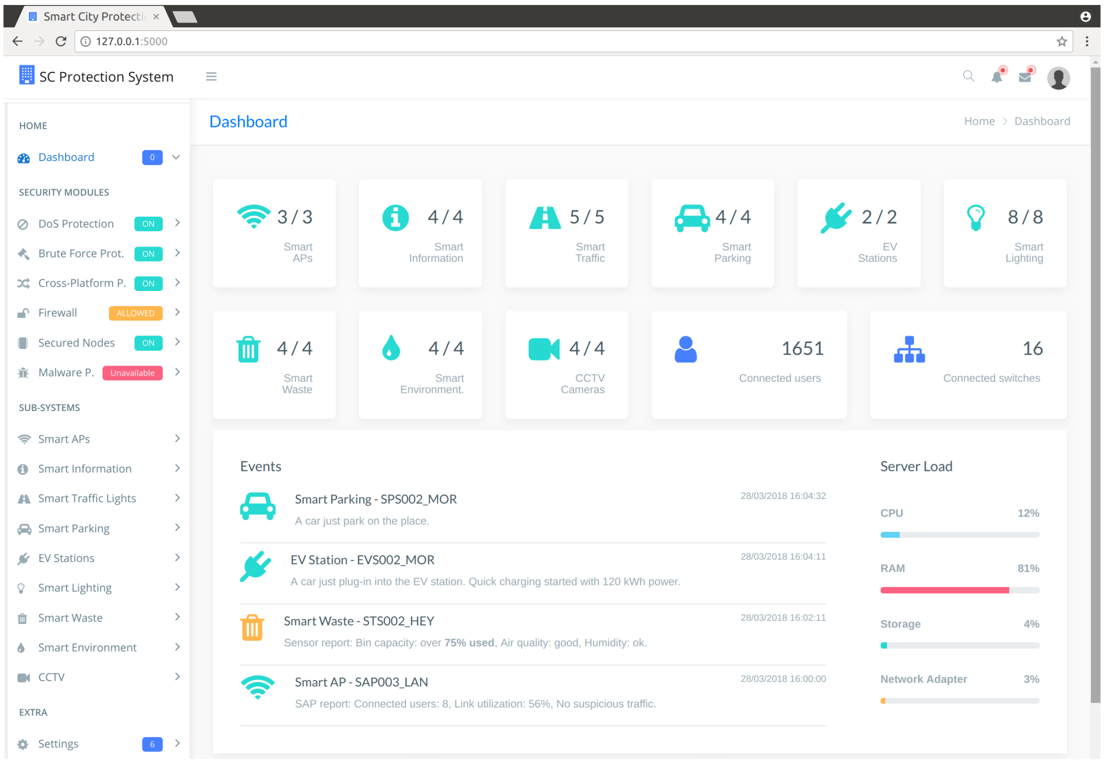
Dashboard

Dashboard represents the main page of the application and it is defined in the *main.html* file. It shows all the subsystems of the SC and number of connected and expected devices. If the numbers do not match, a problem is highlighted by a yellow icon and it also increases the number of events in the main left navigation column. A more serious problem within a subsystem (a node under an attack) will be shown as a red icon with an exclamation mark.

Information about number of connected nodes is based on the flow table of the DTC switch. This table is requested by the *get_data_from_connection_GET* function, which accesses the */sc/flows/dpid* REST API. An expected number of devices is included in the *topology file*, loaded in the web application.

The ideal state of the system is shown in Figure 7.7.

[7]More information: `http://flask.pocoo.org/docs/0.12/templating/`



3 / 3	Smart APs
4 / 4	Smart Information
5 / 5	Smart Traffic
4 / 4	Smart Parking
2 / 2	EV Stations
8 / 8	Smart Lighting
4 / 4	Smart Waste
4 / 4	Smart Environment.
4 / 4	CCTV Cameras
1651	Connected users
16	Connected switches

Events

Smart Parking - SPS002_MOR 28/03/2018 16:04:32
A car just park on the place.

EV Station - EVS002_MOR 28/03/2018 16:04:11
A car just plug-in into the EV station. Quick charging started with 120 kWh power.

Smart Waste - STS002_HEY 28/03/2018 16:02:11
Sensor report: Bin capacity: over **75% used**, Air quality: good, Humidity: ok.

Smart AP - SAP003_LAN 28/03/2018 16:00:00
SAP report: Connected users: 8, Link utilization: 56%, No suspicious traffic.

Server Load

CPU	12%
RAM	81%
Storage	4%
Network Adapter	3%

Figure 7.7: SCPS - Main Page (Dashboard)

Support Pages

The system contains two error pages - 404 (*Page not found*) and 500 (*Server error*). The first one is generated, if an user tries to access a non-existing page. The second one is typically displayed, if the connection to the RYU controller (via the REST API) is lost.

There are also special pages, which confirm execution of a specific action (for example that a FW rule or settings were applied).

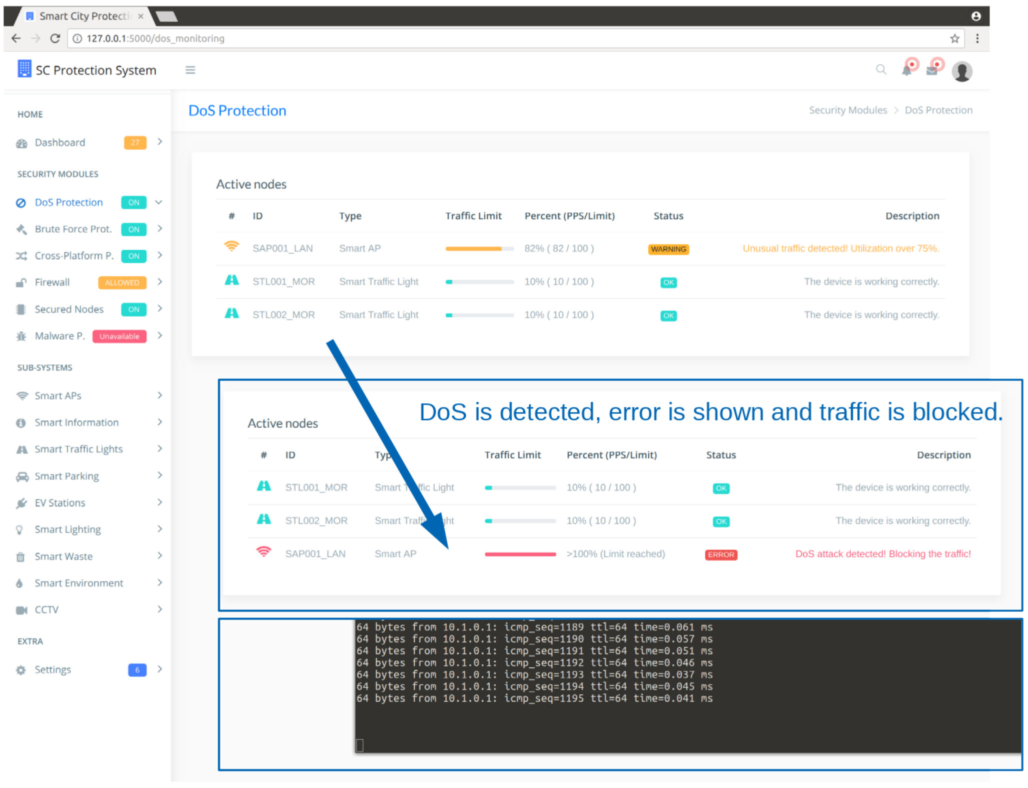
7.4.2 Security Modules

The web application supports different security modules. They are created as web pages extending the main template page *template.html*. Every module therefore has to define only the new functionality.

DoS Protection

The first security module is defined in the *dos_protection.html* file and it provides traffic monitoring of connected nodes. It displays connected nodes and their current traffic volume. Based on a set limit of PPS, a progress bar is shown. If traffic of a device reaches 75% of the allowed amount, a warning is shown (yellow) and if it reaches 100%, the traffic is blocked automatically and an error is generated (red).

The DoS functionality is based on the collected traffic statistics. The number of messages of each flow is collected and presented to the application. If it is higher than the set limit, a *deny rule* for the specified traffic is automatically inserted. In the current version of the application, the rule has idle timeout set to 30 seconds. This will ensure that if the attack stops, the rule will be deleted after 30 seconds, which will re-allow the traffic. The functionality of the module is shown in Figure 7.8.



Active nodes

#	ID	Type	Traffic Limit	Percent (PPS/Limit)	Status	Description
	SAP001_LAN	Smart AP		82% (82 / 100)	WARNING	Unusual traffic detected! Utilization over 75%.
	STL001_MOR	Smart Traffic Light		10% (10 / 100)	OK	The device is working correctly.
	STL002_MOR	Smart Traffic Light		10% (10 / 100)	OK	The device is working correctly.

Active nodes

#	ID	Type	Traffic Limit	Percent (PPS/Limit)	Status	Description
	STL001_MOR	Smart Traffic Light		10% (10 / 100)	OK	The device is working correctly.
	STL002_MOR	Smart Traffic Light		10% (10 / 100)	OK	The device is working correctly.
	SAP001_LAN	Smart AP		>100% (Limit reached)	ERROR	DoS attack detected! Blocking the traffic!

```
64 bytes from 10.1.0.1: icmp_seq=1189 ttl=64 time=0.061 ms
64 bytes from 10.1.0.1: icmp_seq=1190 ttl=64 time=0.057 ms
64 bytes from 10.1.0.1: icmp_seq=1191 ttl=64 time=0.051 ms
64 bytes from 10.1.0.1: icmp_seq=1192 ttl=64 time=0.046 ms
64 bytes from 10.1.0.1: icmp_seq=1193 ttl=64 time=0.037 ms
64 bytes from 10.1.0.1: icmp_seq=1194 ttl=64 time=0.045 ms
64 bytes from 10.1.0.1: icmp_seq=1195 ttl=64 time=0.041 ms
```

Figure 7.8: SCPS - DoS Protection

Brute Force Protection

The brute force protection module allows protection of selected devices (primarily servers) from brute forcing their login procedures. The module is defined in the *bf_protection.html*. It uses the same GUI as the DoS protection, but the traffic progress bar shows data based on 10 second intervals.

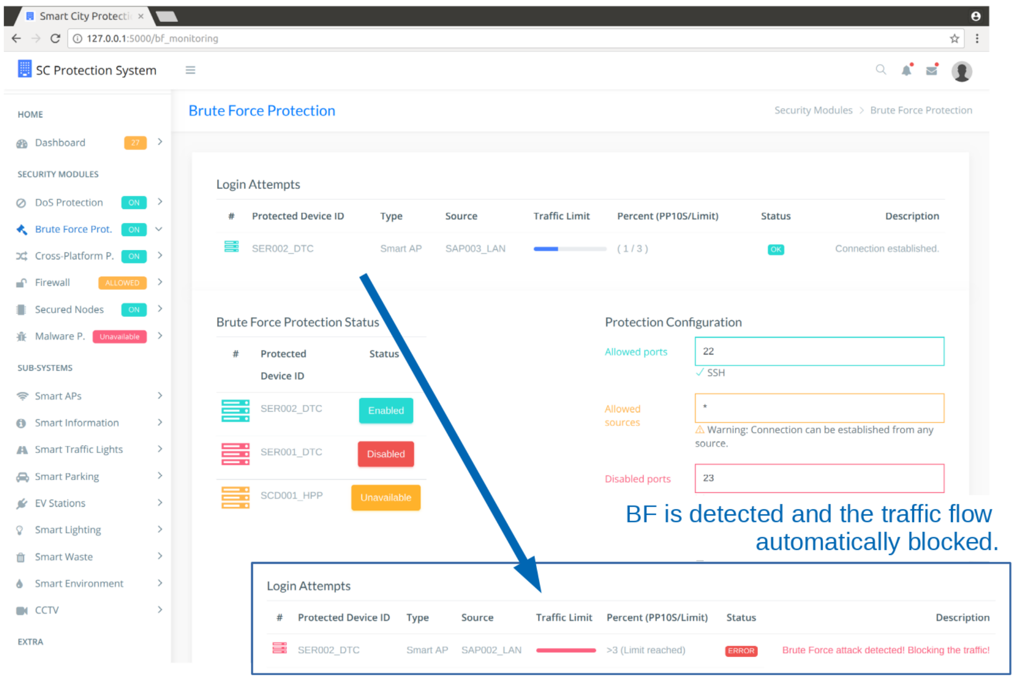
As in DoS, the functionality uses collected statistics, but the system calculates cumulative values in 10 second intervals. The system allows protection of various devices. In the current application state, SER002_DTC is protected from attacks from all sources and on all ports (for all traffic types). If an attack is detected, the same *deny rule* as in DoS is inserted automatically. The functionality of the BF protection module is shown in Figure 7.9.

Cross-platform Protection

The cross-platform protection is defined in the *cross_protection.html* file and it allows definition of virtual layers / tunnels for specific traffic. All the created layers are then monitored and the current traffic is shown. If there is no active traffic, a warning is displayed (but the layer / tunnel stays active).

In the current version of the application, a single tunnel is created automatically during the application start. It is defined in the *create_virtual_layers* function, which is executed in the application initial phase. The function creates virtual tunnel by inserting the following flow rule into the LSS switch:

```
match = parser.OFPMatch(eth_type = ether_types.ETH_TYPE_IP, ipv4_dst="10.8.10.1")
actions = [parser.OFPActionOutput(4)]
self.add_flow(datapath, self.VIRTUAL_LAYER_RULES_PRIORITY, 0, 0, match, actions)
```



Login Attempts

#	Protected Device ID	Type	Source	Traffic Limit	Percent (PP10S/Limit)	Status	Description
	SER002_DTC	Smart AP	SAP003_LAN		(1 / 3)	OK	Connection established.

Brute Force Protection Status

#	Protected Device ID	Status
	SER002_DTC	Enabled
	SER001_DTC	Disabled
	SCD001_HPP	Unavailable

Protection Configuration

Allowed ports	22 ✓ SSH
Allowed sources	* ⚠ Warning: Connection can be established from any source.
Disabled ports	23

BF is detected and the traffic flow automatically blocked.

Login Attempts

#	Protected Device ID	Type	Source	Traffic Limit	Percent (PP10S/Limit)	Status	Description
	SER002_DTC	Smart AP	SAP002_LAN		>3 (Limit reached)	ERROR	Brute Force attack detected! Blocking the traffic!

Figure 7.9: SCPS - Brute Force Protection

The virtual tunnel is simulated as an additional link connected between LSS switch and the SCD001 server. The functionality of the cross-platform protection module is shown in Figure 7.10.

Firewall

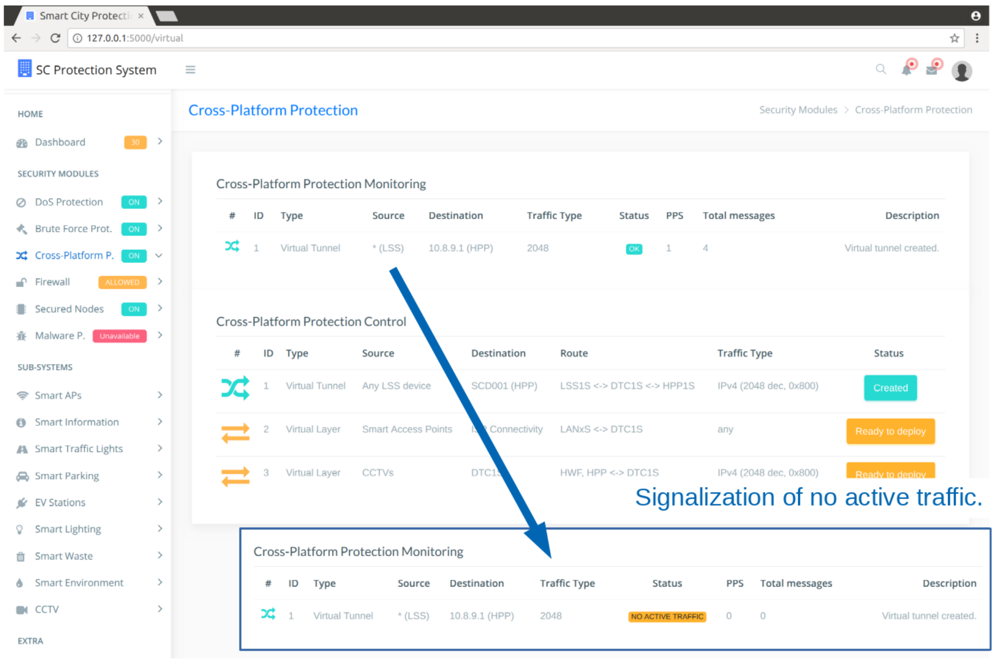
The firewall module (defined in the *firewall.html*) supports monitoring of traffic of various subsystems of the SC. Each flow shows the amount of traffic (current and total) and gives an option to *allow* or *block* the traffic (based on the current status).

After clicking on the *allow* or *block button*, a new flow rule is generated and inserted into the forwarding device. The previously used flow rule is deleted by the *OFPFC_DELETE* command. In the current version of the module, flow rules are created based on source and destination MAC addresses. This allows or blocks the entire communication between the devices. This stateless behaviour corresponds to the ACL mechanism, but due to a commonly used terminology, the module is named "Firewall". This also simplifies expansions of the system. If a more sophisticated (statefull) approach will be implemented, the module does not have to be renamed. The functionality of the firewall module is shown in Figure 7.11.

Secured Nodes

The last implemented module (defined in the *secured_nodes.html*) shows the status of secured nodes in the IPv6 network. Every device in the IPv6 network, which should access server SER001_DTC, must have assigned a *flow label*. Only if the flow label and MAC address of a device match with specified values, the traffic is allowed.

The database is manually created during the application launch. Currently, on-



Cross-Platform Protection

Cross-Platform Protection Monitoring

#	ID	Type	Source	Destination	Traffic Type	Status	PPS	Total messages	Description
	1	Virtual Tunnel	* (LSS)	10.8.9.1 (HPP)	2048	OK	1	4	Virtual tunnel created.

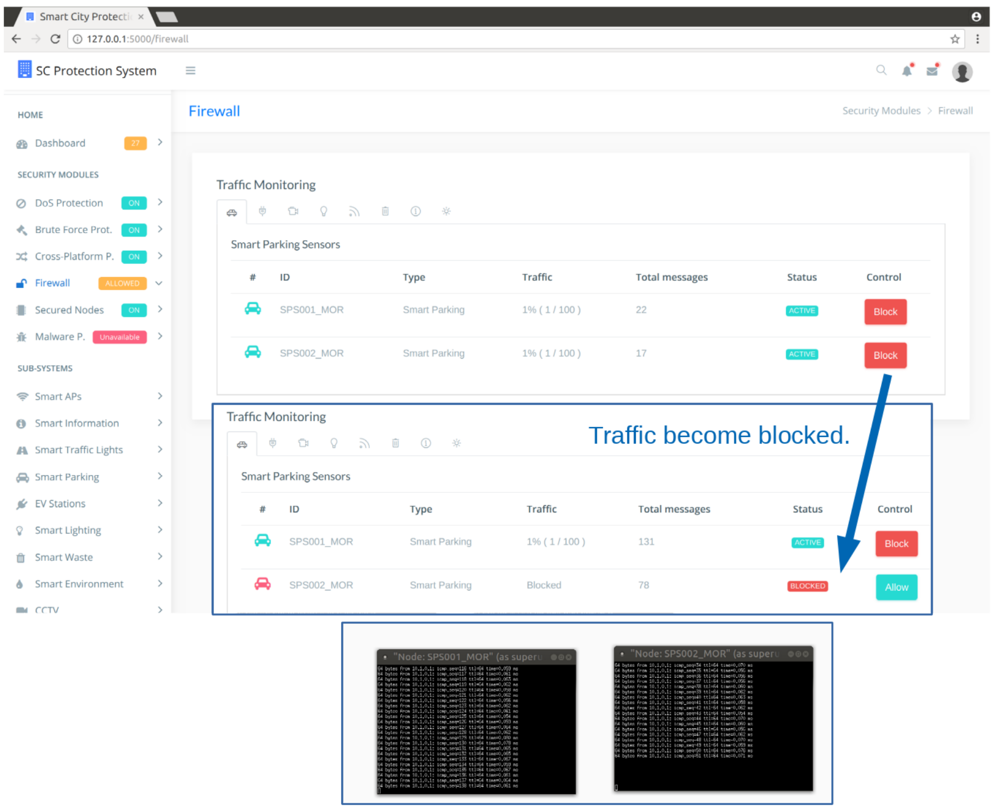
Cross-Platform Protection Control

#	ID	Type	Source	Destination	Route	Traffic Type	Status
	1	Virtual Tunnel	Any LSS device	SCD001 (HPP)	LSS1S <-> DTC1S <-> HPP1S	IPv4 (2048 dec, 0x800)	Created
	2	Virtual Layer	Smart Access Points	I[illegible] Connectivity	LANxS <-> DTC1S	any	Ready to deploy
	3	Virtual Layer	CCTVs	DTC1S	HWF, HPP <-> DTC1S	IPv4 (2048 dec, 0x800)	Ready to deploy

Cross-Platform Protection Monitoring

#	ID	Type	Source	Destination	Traffic Type	Status	PPS	Total messages	Description
	1	Virtual Tunnel	* (LSS)	10.8.9.1 (HPP)	2048	NO ACTIVE TRAFFIC	0	0	Virtual tunnel created.

Figure 7.10: SCPS - Cross-platform Protection



Firewall

Traffic Monitoring

Smart Parking Sensors

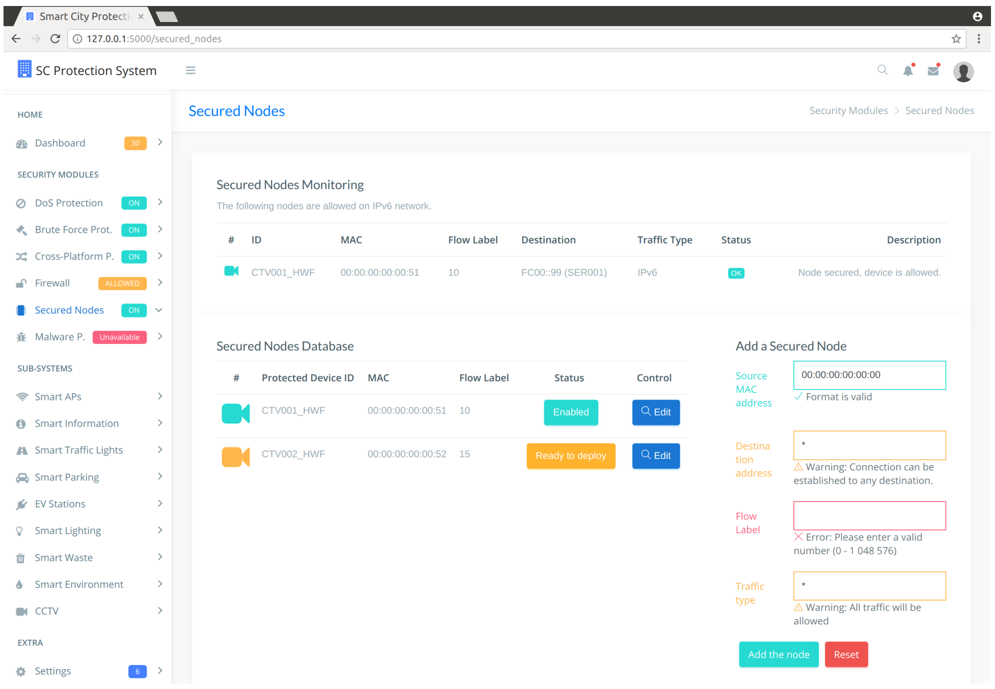
#	ID	Type	Traffic	Total messages	Status	Control
	SPS001_MOR	Smart Parking	1% (1 / 100)	22	ACTIVE	Block
	SPS002_MOR	Smart Parking	1% (1 / 100)	17	ACTIVE	Block

Traffic Monitoring

Smart Parking Sensors

#	ID	Type	Traffic	Total messages	Status	Control
	SPS001_MOR	Smart Parking	1% (1 / 100)	131	ACTIVE	Block
	SPS002_MOR	Smart Parking	Blocked	78	BLOCKED	Allow

Figure 7.11: SCPS - Firewall



Secured Nodes

Secured Nodes Monitoring

The following nodes are allowed on IPv6 network.

#	ID	MAC	Flow Label	Destination	Traffic Type	Status	Description
	CTV001_HWF	00:00:00:00:00:51	10	FC00::99 (SER001)	IPv6	OK	Node secured, device is allowed.

Secured Nodes Database

#	Protected Device ID	MAC	Flow Label	Status	Control
	CTV001_HWF	00:00:00:00:00:51	10	Enabled	Edit
	CTV002_HWF	00:00:00:00:00:52	15	Ready to deploy	Edit

Add a Secured Node

Source MAC address	00:00:00:00:00:00 ✓ Format is valid
Destination address	* ⚠ Warning: Connection can be established to any destination.
Flow Label	✗ Error: Please enter a valid number (0 - 1 048 576)
Traffic type	* ⚠ Warning: All traffic will be allowed

Add the node Reset

Figure 7.12: SCPS - Secured Nodes

ly CTV001_HWF is allowed to communicate, using the flow label 10. The inserted OpenFlow rules are the following:

```
#Default blocking rule
match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IPV6, ipv6_dst="fc00::99")
actions = []
self.add_flow(datapath, self.DENY_RULES_PRIORITY, 0, 0, match, actions)

#Allow rule for the device: ...:51, flow label: 10.
match = parser.OFPMatch(eth_type = ether_types.ETH_TYPE_IPV6,
        eth_src="00:00:00:00:00:51", ipv6_dst="fc00::99", ipv6_flabel=10)
actions = [parser.OFPActionOutput(8)]
self.add_flow(datapath, self.SECURED_NODES_RULES_PRIORITY, 0, 0, match, actions)
```

The functionality of the module is shown in Figure 7.12.

Settings

Protection modules can be enabled or disabled in the *Settings* page. This page also contains a general configuration of DoS and brute force protections. A limiting number of packets per defined time interval can be configured. The complete settings are then inserted into a JSON file and sent via the POST method to the REST API. The loaded configuration is then passed to the main module and applied. If the process is successful, a new JSON file with the current settings of the application is returned in a response message.

Unification of the complete settings into a single JSON files saves the processing performance and bandwidth. From the web application perspective, it is also a convenient approach to obtain all the information in a single HTTP request.

Adding New Modules

Additional modules can be easily added into the application. Firstly, a new web page must extend the *template.html* file, using the following code:

```
{% extends "template.html" %}
{% block title %}Name{% endblock %}
{% block head %}
  {{ super() }}
   <!-- Style definitions -->
{% endblock %}
{% block content %}
     <!-- Start Page Content -->
{% endblock %}
```

The page then has to be connected to the main *scps_web_app.py* file, where a new function has to be defined. This function then renders the page using *render_template* function.

A new page can use existing REST API. If a new functionality is required, the API must be extended. Based on the functionality complexity, the SCPS modules might also need to be expanded.

An example of a new page is the *Malware Protection module* which is shown in the left column of the application (with the status *Unavailable*).

7.5 SCPS Verification

This section verifies several key parameters of the SCPS - mainly from the performance perspective. Because the SCPS was tested in an emulated environment, the first scenario tests this environment to set a *performance baseline*, to which the SCPS can be compared to.

All the tests (unless stated otherwise) were performed on the following machine (referred to as the primary machine):

- **CPU** - AMD Ryzen 7 1800X 8-Core, 3.60 GHz
- **RAM** - 32 GB, 2400 MHz
- **Storage** - Samsung SSD 850 Pro, 256 GB
- **GPU** - NVIDIA GeForce GTX 1060 3 GB
- **OS** - Windows 10

The VM could use all 8 CPU cores up to 100% utilization, and had dedicated 8 GB of physical RAM.

7.5.1 Throughput Performance

This section verifies throughput of the pure Mininet and compares it to throughput measured in the SCPS.

Mininet Throughput

The first scenario verified performance of the Mininet environment using a minimal topology and RYU controller. The topology contained a single switch and two end devices connected to the switch. The RYU controller ran the default application *simple_switch_13.py*, which utilizes logic of a typical L2 switch. Throughput was tested using the *iPerf tool*[8]. This tool measures throughput between two end devices, where

[8]Available: `https://iperf.fr/`

Table 7.3: Throughput Performance of Mininet and SCPS

Traffic Source - Destination	**Experiment num. (Gbps)** 1.	2.	3.	4.	5.	**AVG (Gbps)**
Minimal topology, default simple switch module						
H1 - H2	30.3	30.1	30.5	30.7	30.3	**30.38**
SC topology, default simple switch module						
SAP001_LAN - SER001_DTC	27.3	26.2	26.6	27.8	27.7	**27.12**
SAP001_LAN - STS002_HEY	24.1	23.6	22.5	24.7	23.0	**23.58**
SC topology, SCPS						
SAP001_LAN - SER001_DTC	27.2	26.9	27.3	26.5	24.7	**26.52**
SAP001_LAN - STS002_HEY	25.2	25.0	24.9	25.2	25.1	**25.08**

it is deployed in *server* and *client* modes. In the test, measuring was done using TCP with default *window size* (85.3 KByte). The results are shown in the first part of Table 7.3. Measured throughput from 5 repeated experiments was consistent with an average value of **30.38 Gbps**.

The second scenario verified performance in the SC topology presented in section 7.1.2. The RYU controller was again launched with the *simple_switch_13.py* default module. Two types of data transfer were measured. In the first type, throughput between SAP001_LAN and SER001_DTC was measured. This represented a relatively short communication path via 3 hops (one-way). The second type measured throughput between SAP001_LAN and STS002_HEY. This represented a longer communication path via 5 hops (one-way) and the measured throughput was therefore slightly lower. These two measured values were marked as the *performance baseline*.

SCPS Throughput

This scenario tested throughput performance of the SCPS application. The SC topology was used as in the previous scenario, but the RYU was launched with the main module of the SCPS (*scps_main_module.py*). Throughput was measured between the same devices and the results are shown in the third part of Table 7.3. In order to measure throughput, DoS and brute force protections had to be disabled. Otherwise, the application was running under typical conditions and with default settings (1 second polling timer, web application connected and running).

The results show consistent behaviour and very similar performance to the performance baseline (26.5 vs. 27.1 and 25.1 vs. 23.6 Gbps). Even when compared to the minimal topology, the performance is just slightly lower. It can therefore be concluded, that the SCPS does not negatively influence the network performance.

Remark on Mininet Performance

Although Mininet is an emulator, its performance highly depends on the performance of the machine it is running on. For the comparison, the same experiments as in the previous sections (results in Table 7.3) were conducted on the following laptop device:

- **CPU** - Intel Core i7-3632QM, 4-Core, 2.20 GHz
- **RAM** - 8 GB, 1600 MHz
- **Storage** - Samsung SSD 850 EVO, 500 GB

- **GPU** - Intel HD Graphics 4000
- **OS** - Kubuntu 17.10

The VM could use all 4 CPU cores up to 100% utilization, and had dedicated 3 GB of physical RAM. The average results from 5 experiments are the following (values in brackets represent results from the primary machine):

1. **H1 - H2 (minimal topology, default RYU)**: 15.08 Gbps (30.38 Gbps)
2. **SAP - SER (SC topology, default RYU)**: 13.44 Gbps (27.12 Gbps)
3. **SAP - STS (SC topology, default RYU)**: 11.94 Gbps (23.58 Gbps)
4. **SAP - SER (SC topology, SCPS)**: 12.5 Gbps (26.52 Gbps)
5. **SAP - STS (SC topology, SCPS)**: 12.36 Gbps (25.08 Gbps)

The results were also consistent, but as can be seen, they were about 50% lower, than in the case of the primary machine. For consistency reasons, all the subsequent tests were performed only on the primary machine.

7.5.2 SCPS Performance - Latency

The second test verified Mininet latency performance. Latency is defined as a time interval between sending a message from a node A and receiving the message on a node B. It is therefore measured one-way only. This experiment required use of a traffic capturing tool to compare timestamps of messages on appropriate interfaces. The *Wireshark* tool[9] was chosen for this purpose. Because the end devices share the same kernel and OS libraries, their time was synchronized. The precise time difference (between sending and receiving a message) could therefore be detected.

Two experiments were conducted between the same devices as in the throughput test. In the first scenario, the first interface was on the LAN1 switch, leading to SAP001. The second interface was on the DTC switch, leading to SER001. In the second scenario, the first interface was the same and the second interface was on the HEY02 switch, leading to STS002.

Additionally, a RTT was also measured in these experiments, using output from the *ping* tool. The results from the three first ICMP messages in both experiments are shown in Table 7.4. The table shows separated times for request and reply messages, and RTT measured by *ping*. In all cases, the OpenFlow rules were already present in flow tables. As expected, the measured RTT is higher than the sum of request and reply messages, because it includes a processing time on the queried device. A significantly higher latency of the first message is caused by ARP, which translates the IP address to MAC address.

7.5.3 SCPS Protection Modules Performance

This section verifies performance and behaviour of the two protection modules - DoS and cross-layer. Testing of other modules was not necessary. Brute force protection uses the same concept as DoS (only different time intervals and number of packets), and firewall and secured nodes protections use manual flow rule creation.

[9]Available: `https://www.wireshark.org/`

Table 7.4: SCPS Performance - Latency

Source - Destination, packet n.	Request L.	Reply L.	RTT
SAP001_LAN - SER001_DTC, 1.	0.451 ms	0.166 ms	0.654 ms
SAP001_LAN - SER001_DTC, 2.	0.024 ms	0.006 ms	0.077 ms
SAP001_LAN - SER001_DTC, 3.	0.022 ms	0.006 ms	0.069 ms
SAP001_LAN - STS002_HEY, 1.	0.666 ms	0.268 ms	0.979 ms
SAP001_LAN - STS002_HEY, 2.	0.026 ms	0.009 ms	0.075 ms
SAP001_LAN - STS002_HEY, 3.	0.027 ms	0.010 ms	0.076 ms

DoS Protection Performance

This scenario verified the behaviour of the DoS protection module. The polling interval was set to 1 second, so the DoS protection was checked every second. The scenario measured transmitted data traffic between two end devices: SAP001_LAN and SER001_DTC. The maximum amount of traffic was generated between them by the *iPerf* tool.

The average amount of forwarded traffic between devices, before the protection was triggered, was **4.218 GB** (partial experiments measured: 4.13 GB, 9.77 GB, 2.72 GB, 1.43 GB, and 3.04 GB). This roughly corresponds to the maximum transfer rate of 30 Gbps, which can transfer about 3.75 GB (30 / 8) every second.

This traffic presents a significant amount and it might be too much to handle in real SC scenarios. In that case, a more responsive solution has to be implemented. One approach is to utilize a specialized hardware device. Another one would be a more frequent polling, which would however increase the overhead.

Cross-layer Protection Performance

This scenario verified performance of proactive flow rule insertion (used in cross-layer protection) and reactive flow rule insertion (used in normal forwarding). Cross-layer protection, used in SCPS, created a virtual tunnel between all devices connected to the LSS switch and SCADA server (SCD001_HPP). This connection was simulated by connecting the SCADA server to a specific port of the LSS switch. The traffic therefore had to cross only this single device.

In the first experiment, the virtual tunnel was created and IP-MAC mapping for the specified traffic was manually inserted into both end devices by the following modification of the *lan_sc_topology.py* file:

```
scd001_hpp.cmdPrint('arp -s 10.7.8.1 00:00:00:00:00:61')
ses001_lss.cmdPrint('arp -s 10.8.10.1 00:00:00:00:00:73')
```

In the second experiment, the virtual tunnel was not created, but the manual IP-MAC mapping was still set. The results of these experiments are shown in Table 7.5. It is clearly visible, that the latency and RTT are significantly lower in the case of proactive flow rule insertion. This difference would be even more noticeable with increased number of network hops. As RTT suggests, the biggest delay in reactive flow rule insertion is caused by the controller processing. It is important to mention, that this behaviour applies only to the first message of a flow. Performance of the following messages is similar regardless if the tunnel is used or not.

The results also show, that the first packet latency and RTT is about 50% lower than in latency testing in Table 7.4. This is due to the use of static IP-MAC mapping.

Table 7.5: Cross-layer Protection Performance

Source - Destination, packet n.	Request L.	Reply L.	RTT
Proactive flow rule insertion - virtual tunnel active			
SES001_LSS - SCD001_HPP, 1.	0.200 ms	0.1 ms	0.34 ms
SES001_LSS - SCD001_HPP, 2.	0.014 ms	0.003 ms	0.057 ms
SES001_LSS - SCD001_HPP, 3.	0.014 ms	0.002 ms	0.056 ms
Reactive flow rule insertion - virtual tunnel not-active			
SES001_LSS - SCD001_HPP, 1.	0.493 ms	0.0139 ms	9.52 ms
SES001_LSS - SCD001_HPP, 2.	0.001 ms	0.0004 ms	0.069 ms
SES001_LSS - SCD001_HPP, 3.	0.002 ms	0.0003 ms	0.068 ms

7.5.4 Verification Summary

The tests verified SCPS performance from several points of view. The system was compared to the clean RYU controller running the most basic module (containing logic of a simple L2 switch). The SCPS did not limit the performance by more than 2.2%, which can be considered to be a measurement error.

The only problematic part is the DoS protection, which under specific conditions can take more than one second to react. In that case, all the devices would be vulnerable for this period. This might not be a problem for typical servers and PCs, but can easily damage low performance SC nodes and sensors.

Solutions to this problem are not easy and they require either a combination of SDN with traditional protection devices (IPS), tuning of the polling interval, or implementation of extremely restrictive forwarding. This could include transmitting of all the protected flows via the controller. It would protect the end nodes, but on the other hand, it would make the controller more vulnerable instead.

8. Conclusion

The thesis introduced the concept of SDN, including its related technologies like NFV and hybrid SDN, and the relevant area of SDN deployment - IoT and its sub-domains. Analysis of the related work in this area revealed several potential gaps in the current research. Security of IoT was identified as the area with the biggest potential for the future research and was chosen as the main aim of this work. Because of the wide scope of this area, it was narrowed to focus mostly on smart cities. They have demanding security requirements, which are currently often ignored. The most important part and the main contribution of the thesis is the proposed blueprint.

Proposed Blueprint

The thesis proposed the blueprint for effective deployment of SDN applications into areas of IoT, specifically the area of smart cities. To my very best knowledge, the blueprint of this scope has not yet been presented in any literature. In order to propose this blueprint, several analyses have to be firstly conducted. These analyses covered the theoretical background of SDN, IoT, and related work, as well as smart city architectures, protocols, and security issues.

The main contribution of the proposed blueprint is the step-by-step development process of an SDN application into smart cities with focus on the security life cycle. The process described, in detail, requirements for SDN applications, which were gathered from the smart city analysis. All the requirements present an extensive list of features, which a smart city application should aim to acquire. The blueprint then described general protection steps against smart city security threats. Examples of detailed implementation of these protections in SDN were included.

All real-world applications, which use the blueprint, should comply to the development process. This process defines a step-by-step approach to effective deployment of an application, with security in mind. Strong security is supported by the included security life cycle methodology, which should be used in parallel with the development process.

The requirement part of the development process can be modified, based on the application needs. Typically, not all requirements have to be implemented in all scenarios and on the contrary, some additional requirements might be needed. Also, the level of importance of selected requirements will vary based on the specific application needs. The same applies for the protection against specific threats. They should be implemented based on the target scenario.

Blueprint Verification

The blueprint was verified in a use case application of the Smart City Protection System (SCPS). The application introduced an emulated topology of a fictitious smart city, composed from 8 domains, 16 forwarding devices, and 41 IoT nodes.

The application itself was developed according to the blueprint's development process including specific parts from the security life cycle. The assessment phase allowed identification of five of the most important protection mechanisms against common threats. These protections included: DoS, brute force, cross-platform, traffic filtering, and secured nodes. They defined the application scope identified in the first (planning) phase.

The analysis phase identified requirements for the application - 15 of the proposed requirements were implemented. These requirements were sufficient for the application demonstration. Most importantly, the implementation of most of the other requirements would not be possible, or would not have any benefits in the emulated environment. This included mostly performance related criteria, reliability, and recovery. Most of these requirements could be accomplished by use of a more advanced SDN controller, which would however significantly complicate the application development.

The design phase then described the decision process about used SDN controller, its architecture, used APIs, and choice of the application mode. Detailed structure of the system was designed and presented in several UML diagrams, which were used in the final implementation phase.

The key functionalities of the SCPS were tested and compared to the clean SDN solution of RYU controller. The tests verified, that the SCPS with its modules does not limit the network performance. The measured difference in performance was below 2.2%, which is negligible (especially in the given context). On the other hand, reaction times of the DoS module can be insufficient and additional means of protection might be necessary. The main advantage of the SDN solution remains - the application can provide at least a certain degree of protection at no additional cost. If a more advanced protection is needed, a dedicated specialized device can be integrated into the network and it can combine its functionality with the SDN application.

SCPS Remarks

The main purpose of the SCPS was to verify the proposed blueprint. For this reason, some parts were considerably simplified. This included both the network topology and application functionality. The network was designed as a flat network, so it did not use subnetting. Therefore, the system could omit routing. Otherwise, a more advanced controller module with this functionality, would have to be used. Moreover, redundant links were blocked, so a loop protection technique was not needed. Finally, the system performed only a limited scope of specifically given functionalities. This mainly applied to the web application interface, which contained many demonstration-only UI elements. Some of the modules also had limited functionalities, especially in terms of configuration abilities. Only general settings could be configured from the application and all granular changes had to be done in the main module code (by changing constants). Probably the most notable limitation was functionality of the firewall module, which behaved as a simple ACL and supported only filtering based on source and destination MAC addresses.

The simplification of the application was done mostly due to time constraints. Even while the RYU controller is one of the simpler ones, development of an application of similar scope can become tedious (especially if it is done by a single person). The development was significantly sped up by the proposed blueprint, but it still presented a major challenge.

Real Experiences from Using Blueprint

The blueprint was complexly verified during the entire development process of the SCPS. Firstly, the provided step-by-step development process significantly simplified the application development. This was true even while not all the process steps were needed, as the SCPS served only for demonstration and verification purposes.

Secondly, the list of application requirements proposed by the blueprint, allowed instant use of these requirements in the application development. This greatly shortened the analysis phase. Without this list, similar requirements would have to be gathered by several time consuming methods. Moreover, in the use case application, some of these methods (interviews with users) would not be applicable.

The list of protection methods (and their implementation in SDN), also greatly reduced the development time as the implementation of these features did not require performing of additional activities from the analysis and design phases. From 14 described protections, 5 were implemented into the SCPS. From the verification and testing procedures, it is clear, that these protections can enhance the security of smart city networks.

Although the SCPS could not verify all the aspects of the blueprint (as probably no single application will), the verified parts greatly helped in the application development. The blueprint provided clear scopes for the application, and reduced the development time.

Future Work

The main contribution of the thesis - the proposed blueprint was completed and currently does not require significant extensions. The validity of the blueprint should last, unless the smart city, or SDN concepts significantly change their architectures. This could potentially happen in the SDN area with the introduction of a new approach to complete network programmability. This could be achieved by P4, or by a new generation of OpenFlow. Assuming, that such a version will come (OpenFlow 2.0), it is expected, that it will no longer contain any fixed fields, but that it will be completely flexible. This would allow the full programmability of forwarding devices. These technologies could make the current SDN obsolete and incompatible with new approaches.

The SCPS is also in a usable state for its current purpose. If the application should be deployed in a real smart city, it would require a significant extension. This could however, be achieved effectively, as the application conformed to the best practices proposed in the blueprint. For example, security modules could be easily added or modified, without influencing other parts of the application. Probably the biggest concern in the real world deployment would be the selected controller architecture. RYU does not support distributed architecture, which would significantly limit the system reliability. Portability of the application to a more advanced controller would therefore be necessary. While the web part of the application could easily handle this transition, the controller part would have to be rewritten.

In summary, the blueprint fulfilled its role and the SCPS provided its verification. This was the application's main goal and, as such, was accomplished. The application was not designed to be deployed in a real smart city scenario.

Nevertheless, it is important to mention, that no SDN solution (nor any other technology) can ensure 100% protection to the entire smart city network. The biggest advantage of an SDN solution is, that it can provide *some* level of protection at *almost* no additional cost (assuming that SDN is already implemented in the network) - the implementation requires only the appropriate software modification.

Bibliography

[1] ITU, “ICT facts and figures 2017,” tech. rep., International Telecommunication Union, 2017. [Online]. Available: https://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2017.pdf. [Accessed: May 22, 2018].

[2] R. Guerzoni, R. Trivisonno, and D. Soldani, “SDN-based architecture and procedures for 5G networks,” in *1st International Conference on 5G for Ubiquitous Connectivity*, pp. 209–214, 2014.

[3] T. Marill and L. G. Roberts, “Toward a cooperative network of time-shared computers,” in *Proceedings of the November 7-10, 1966, Fall Joint Computer Conference*, AFIPS ’66 (Fall), (New York, NY, USA), pp. 425–431, ACM, 1966.

[4] J. F. Kurose and K. W. Ross, *Computer networking: a top-down approach featuring the Internet.* Pearson, second ed., 2002.

[5] V. G. Cerf and R. E. Icahn, “A protocol for packet network intercommunication,” *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 2, pp. 71–82, 2005.

[6] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-Defined Networking: a comprehensive survey,” *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[7] R. Jain, “OpenFlow, Software Defined Networking (SDN) and Network Function Virtualization (NFV),” *Tutorial at 2014 IEEE 15th International Conference on High Performance Switching and Routing*, 2014. [Online]. Available: http://goo.gl/kuBdgJ. [Accessed: July 19, 2016].

[8] N. Feamster, J. Rexford, and E. Zegura, “The road to SDN: an intellectual history of programmable networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, 2014.

[9] L. Yang, R. Dantu, T. Anderson, and R. Gopal, “Forwarding and Control Element Separation (ForCES) framework,” 2014. [Online]. Available: https://tools.ietf.org/html/rfc3746. [Accessed: July 26, 2016].

[10] ONF, “ONF overview,” 2016. [Online]. Available: https://www.opennetworking.org/mission/. [Accessed: July 26, 2016].

[11] S. McGillicuddy, “Microsoft’s Windows Azure network is a massive, virtual SDN,” 2014. [Online]. Available: http://goo.gl/sN8Iay. [Accessed: Aug. 11, 2016].

[12] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, “B4: experience with a globally-deployed software defined WAN,” *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, 2013.

[13] D. Simeonidou, “Bristol is open,” 2015. [Online]. Available: https://goo.gl/Ja7zy7. [Accessed: Aug. 31, 2017].

[14] Bristol Is Open Ltd, “SDN/NFV,” 2016. [Online] Available: https://tinyurl.com/y7ks5uqn. [Accessed: Aug. 31, 2017].

[15] Cloud Technology Partners, "SDN: uncovering Amazon's secret sauce," 2013. [Online]. Available: http://goo.gl/VZkDI5. [Accessed: Aug. 11, 2016].

[16] Y. Bachar, "Introducing "6-pack": the first open hardware modular switch," 2015. [Online]. Available: https://goo.gl/xQKgMQ. [Accessed: Aug. 11, 2016].

[17] T. B. Research, "Enterprise operators will turn to consolidation and NFV/SDN to improve profitability in 2016," 2016. [Online]. Available: http://goo.gl/JoMdKQ. [Accessed: Aug. 11, 2016].

[18] C. Ching-Hao and Y.-D. Lin, "OpenFlow version roadmap," 2015. [Online]. Available: http://speed.cis.nctu.edu.tw/~ydlin/miscpub/indep_frank.pdf. [Accessed: May. 30, 2018].

[19] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B. B. Kang, "Rosemary: A robust, secure, and high-performance network operating system," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, (New York, NY, USA), pp. 78–89, ACM, 2014.

[20] M. Jarschel, T. Zinner, T. Hossfeld, P. Tran-Gia, and W. Kellerer, "Interfaces, attributes, and use cases: A compass for SDN," *IEEE Communications Magazine*, vol. 52, no. 6, pp. 210–217, 2014.

[21] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[22] ONF, "OpenFlow switch specification, version 1.0.0," 2009. [Online]. Available: https://goo.gl/8FalPF. [Accessed: July 19, 2016].

[23] V. Khatri, "Analysis of OpenFlow protocol in local area networks," Master's thesis, Tampere University of Technology, 2013. [Online]. Available: https://goo.gl/LqBd9B. [Accessed: July 19, 2016].

[24] ONF, "OpenFlow switch specification, version 1.5.1," 2015. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf. [Accessed: May 13, 2018].

[25] ONF, "OpenFlow switch specification, version 1.5.0," 2014. [Online]. Available: https://goo.gl/JwyWgT. [Accessed: July 19, 2016].

[26] S. Vissicchio, L. Vanbever, and O. Bonaventure, "Opportunities and research challenges of hybrid software defined networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 70–75, 2014.

[27] OpenDaylight Wiki, *10. Group Based Policy user guide.* The OpenDaylight foundation, 2015. [Online]. Available: https://goo.gl/uMjgrp. [Accessed: July 26, 2016].

[28] A. Gonsalves, "Intent-based networking needed to run more complex networks," 2017. [Online]. Available: https://goo.gl/1LjYrQ. [Accessed: Sept. 6, 2017].

[29] J. Fruehe, "Sorting the hype from the truth with intent-based networking systems," 2017. [Online]. Available: https://tinyurl.com/yajm95cu. [Accessed: Sept. 6, 2017].

[30] Cisco, "Opflex: An open policy protocol white paper," tech. rep., 2015. [Online]. Available: http://goo.gl/D0oeI9. [Accessed: July 26, 2016].

[31] IETF, "Opflex control protocol, draft-smith-opflex-03," 2016. [Online]. Available: https://tools.ietf.org/html/draft-smith-opflex-03. [Accessed: July 26, 2016].

[32] IETF, "Interface to the routing system (i2rs)," 2016. [Online]. Available: https://datatracker.ietf.org/wg/i2rs/charter/. [Accessed: July 19, 2016].

[33] D. Dhody, X. Zhang, O. Gonzalez, D. Ceccarelli, and B. Yoon, "Abstraction and control of transport networks (actn)," 2016. [Online]. Available: https://datatracker.ietf.org/wg/actn. [Accessed: Aug. 12, 2016].

[34] J. H. Salim, "Forwarding and control element separation (ForCES) protocol extensions," 2014. [Online]. Available: https://tools.ietf.org/html/rfc7391. [Accessed: Aug. 12, 2016].

[35] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network configuration protocol (NETCONF)," 2011. [Online]. Available: https://tools.ietf.org/html/rfc6241. [Accessed: Aug. 12, 2016].

[36] E. B. Pfaff, B. Davie, "The Open vSwitch database management protocol," 2013. [Online]. Available: https://tools.ietf.org/html/rfc7047. [Accessed: Aug. 12, 2016].

[37] J. Vasseur and J. L. Roux, "Path computation element (PCE) communication protocol (PCEP)," 2013. [Online]. Available: https://tools.ietf.org/html/rfc5440. [Accessed: Aug. 12, 2016].

[38] ETSI, "Network functions virtualisation," White Paper 1, European Telecommunications Standards Institute, 2012. [Online]. Available: https://portal.etsi.org/NFV/NFV_White_Paper.pdf. [Accessed: July 20, 2017].

[39] ETSI, "About ETSI," 2017. [Online]. Available: http://www.etsi.org/about. [Accessed: July 26, 2017].

[40] ETSI, "Network functions virtualisation (NFV); management and orchestration; report on architectural options," 2016. Ref.: DGS/NFV-IFA009.

[41] SDNCentral LLC, "2017 NFV report series part i foundations of NFV: NFV infrastructure and VIM," market report, 2017. [Online]. Available: https://www.sdxcentral.com/reports/nfv-infrastructure-vim-download-2017/. [Accessed: July 20, 2017].

[42] SDNCentral LLC, "2017 NFV report series part 3: Powering NFV - virtual network functions (VNFs)," industry report, 2017. [Online]. Available: https://www.sdxcentral.com/reports/nfv-vnf-download-2017/. [Accessed: July 20, 2017].

[43] SDNCentral LLC, "2017 NFV report series part 2: Orchestrating NFV - MANO and service assurance," industry report, 2017. [Online]. Available: https://www.sdxcentral.com/reports/nfv-mano-and-service-assurance-download-2017/. [Accessed: July 20, 2017].

[44] OpenStack Foundation, "OpenStack open source cloud computing software," 2017. [Online]. Available: `https://www.openstack.org/`. [Accessed: July 21, 2017].

[45] N. McKeown and J. Rexford, "Clarifying the differences between P4 and OpenFlow," 2016. [Online]. Available: `http://p4.org/p4/clarifying-the-differences-between-p4-and-openflow/`. [Accessed: July 26, 2017].

[46] N. Instruments, "Introduction to FPGA technology: Top 5 benefits," tech. rep., 2012. [Online]. Available: `http://www.ni.com/white-paper/6984/en/`. [Accessed: July 20, 2017].

[47] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 99–110, 2013.

[48] P4 Language Consortium, "P4," 2017. [Online]. Available: `http://p4.org/`. [Accessed: July 20, 2017].

[49] P4 Language Consortium, "P4 language specification," may 2017. [Online]. Available: `https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.pdf`. [Accessed: Mar. 7, 2018].

[50] C. Cascone, "P4 support in ONOS," 2018. [Online]. Available: `https://onosproject.org/wp-content/uploads/2018/01/6-P4-support-in-ONOS.pdf`. [Accessed: Mar. 7, 2018].

[51] K. Ashton, "That 'internet of things' thing," *RFID Journal*, 2009. [Online]. Available: `http://www.rfidjournal.com/articles/view?4986`. [Accessed: Aug. 8, 2017].

[52] R. Macmanus, "Consumer electronics 2.0: MIT's Henry Holtzman on the Internet of Things," 2010. [Online]. Available: `https://readwrite.com/2010/01/02/redux_consumer_electronics_20_mits_henry_holtzman/`. [Accessed: Aug. 18, 2017].

[53] D. Evans, "The internet of things, how the next evolution of the internet is changing everything," tech. rep., Cisco Internet Business Solutions Group (IBSG) White Paper, 2011. [Online]. Available: `http://goo.gl/RdDkkO`. [Accessed: July 26, 2016].

[54] P. Waher, *Learning Internet of Things.* Packt Publishing, 2015.

[55] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys Tutorials*, vol. 17, pp. 2347–2376, Fourthquarter 2015.

[56] O. Vermesan, M. Harrison, H. Vogt, K. Kalaboukas, M. Tomasella, K. Wouters, S. Gusmeroli, and S. Haller, "Vision and challenges for realising the internet of things," tech. rep., European Commission Information Society and Media, 2010. [Online]. Available: `http://goo.gl/f37ZEJ`. [Accessed: July 26, 2016].

[57] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (IoT): A vision, architectural elements, and future directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.

[58] M. Bauer et al., "Internet of Things – Architecture IoT-A Deliverable D1.5 – Final architectural reference model for the IoT v3.0," technical report, 2013.

[59] ITU, "The Internet of Things," ITU Internet Reports, International Telecommunication Union, Tunis, Tunisia, 2005.

[60] FP7 ICT Advisory Group, "Working group on future internet infrastructure," tech. rep., FP7, 2008. [Online]. Available: `http://cordis.europa.eu/pub/ist/docs/future-internet-istag_en.pdf`. [Accessed: Aug. 21, 2017].

[61] I. Romdhani, R. Abdmeziem, and D. Tandjaoui, *Architecting the Internet of Things: State of the art.* Springer International Publishing, 2015.

[62] M. Wu, T.-J. Lu, F.-Y. Ling, J. Sun, and H.-Y. Du, "Research on the architecture of internet of things," in *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, vol. 5, pp. V5–484–V5–487, 2010.

[63] N. Enose and R. Analyst, "A unified management system for Smart Grid," in *ISGT2011-India*, pp. 328–333, 2011.

[64] A. Phan and S. T. Qureshi, "5G impact on Smart Cities," 2017. [Online]. Available: `https://www.researchgate.net/publication/315804922_5G_impact_On_Smart_Cities`. [Accessed: Feb. 18, 2018].

[65] A. Hellemans, "Why IoT needs 5G," 2015. [Online]. Available: `https://spectrum.ieee.org/tech-talk/computing/networks/5g-taking-stock`. [Accessed: Feb. 18, 2018].

[66] NGMN Alliance, "NGMN 5G initiative white paper," final deliverable (approved), 5G Initiative Team, 2015. [Online]. Available: `https://www.ngmn.org/fileadmin/ngmn/content/downloads/Technical/2015/NGMN_5G_White_Paper_V1_0.pdf`. [Accessed: Feb. 18, 2018].

[67] ITU-T, "The Tactile Internet," ITU-T Technology Watch Report August 2014, ITU-T, 2014. [Online]. Available: `https://www.itu.int/dms_pub/itu-t/opb/gen/T-GEN-TWATCH-2014-1-PDF-E.pdf`. [Accessed: Feb. 19, 2018].

[68] Z. S. Bojkovic, B. M. Bakmaz, and M. R. Bakmaz, "Vision and enabling technologies of tactile internet realization," in *2017 13th International Conference on Advanced Technologies, Systems and Services in Telecommunications (TELSIKS)*, pp. 113–118, 2017.

[69] H. S. Varsha and K. P. Shashikala, "The tactile internet," in *2017 International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, pp. 419–422, 2017.

[70] C. Perera, C. H. Liu, S. Jayawardena, and M. Chen, "A survey on internet of things from industrial market perspective," *IEEE Access*, vol. 2, pp. 1660–1679, 2014.

[71] A. Bassi, "Application domains for Internet of Things," 2013. [Online]. Available: `http://netsoc.future-internet.eu/fileadmin/netsoc_events/2013-03-4th_usage_WS/UA-1/IoT.pdf`. [Accessed: Aug. 25, 2017].

[72] Intel, "Intel® responsive retail platform," tech. rep., 2017. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/iot-rrp-solution-brief-final.pdf. [Accessed: Dec. 6, 2017].

[73] Intel®, "Intel® reference design for intelligent vending," 2017. [Online]. Available: https://www.intel.com/content/www/us/en/intelligent-systems/retail/reference-design-for-intelligent-vending.html. [Accessed: Dec. 6, 2017].

[74] I. Celino and S. Kotoulas, "Smart Cities [guest editors' introduction]," *IEEE Internet Computing*, vol. 17, no. 6, pp. 8–11, 2013.

[75] C. Harrison, B. Eckman, R. Hamilton, P. Hartswick, J. Kalagnanam, J. Paraszczak, and P. Williams, "Foundations for Smarter Cities," *IBM Journal of Research and Development*, vol. 54, no. 4, pp. 1–16, 2010.

[76] A. Gharaibeh, M. A. Salahuddin, S. J. Hussini, A. Khreishah, I. Khalil, M. Guizani, and A. Al-Fuqaha, "Smart Cities: A survey on data management, security and enabling technologies," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2456–2501, 2017.

[77] Siemens, "Sustainable cities," Report L7-Z719-V2-7600, Siemens, 2010. [Online]. Available: https://www.siemens.com/about/sustainability/pool/nachhaltige_entwicklung/sustainablecities_2010-08-11.pdf. [Accessed: Sept. 4, 2017].

[78] H. Chourabi, T. Nam, S. Walker, J. R. Gil-Garcia, S. Mellouli, K. Nahon, T. A. Pardo, and H. J. Scholl, "Understanding smart cities: An integrative framework," in *2012 45th Hawaii International Conference on System Sciences*, pp. 2289–2297, 2012.

[79] H. Arasteh, V. Hosseinnezhad, V. Loia, A. Tommasetti, O. Troisi, M. Shafie-khah, and P. Siano, "IoT-based smart cities: A survey," in *2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC)*, pp. 1–6, 2016.

[80] International Energy Agency, *Light's labour's lost*. IEA Publications, 2006. ISBN: 926410951X.

[81] Silver Spring Networks, "The business case for smart street lights," Whitepaper REV. 9/25/2013, Silver Spring Networks, 2013. [Online]. Available: https://www.silverspringnet.com/wp-content/uploads/SilverSpring-Whitepaper-Smart-Street-Light-Bizcase.pdf. [Accessed: Sept. 8, 2017].

[82] M. Tushar, C. Assi, and M. Maier, "Distributed real-time electricity allocation mechanism for large residential microgrid," *IEEE Transactions on Smart Grid*, vol. 6, no. 3, pp. 1353–1363, 2015.

[83] M. H. K. Tushar, A. W. Zeineddine, and C. Assi, "Demand-side management by regulating charging and discharging of the ev, ess, and utilizing renewable energy," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 1, pp. 117–126, 2018.

[84] J. Choudhary, S. Pal, V. S. Pareek, and R. Kumar, “Assessment of EV and ESS performance based load scheduling in smart home,” in *2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES)*, pp. 1–7, 2016.

[85] C. WooChul and N. JoonYeop, “Relative importance for crime prevention technologies as part of smart city based on spatial information,” in *2017 Smart City Symposium Prague (SCSP)*, pp. 1–5, 2017.

[86] J. van Heek, K. Aming, and M. Ziefle, “How fear of crime affects needs for privacy safety: Acceptance of surveillance technologies in smart cities,” in *2016 5th International Conference on Smart Cities and Green ICT Systems (SMARTGREENS)*, pp. 1–12, 2016.

[87] Intel, “Smart building scenarios,” 2017. [Online]. Available: `https://www.intel.com/content/www/us/en/smart-buildings/smart-building-scenarios/overview.html`. [Accessed: Feb. 7, 2018].

[88] F. Cleveland, “IEC TC57 WG15: IEC 62351 security standards for the power system information infrastructure,” 2012.

[89] B. Ge and W.-T. Zhu, *Preserving User Privacy in the Smart Grid by Hiding Appliance Load Characteristics*, pp. 67–80. Cham: Springer International Publishing, 2013.

[90] S. Uludag, S. Zeadally, and M. Badra, *Techniques, Taxonomy, and Challenges of Privacy Protection in the Smart Grid*, pp. 343–390. Cham: Springer International Publishing, 2015.

[91] NIST, “Framework and roadmap for smart grid interoperability standards, release 2.0,” tech. rep. [Online]. Available: `http://goo.gl/4ZXB7y`. [Accessed: July 26, 2016].

[92] O. Hersent, D. Boswarthick, and O. Elloumi, *The Internet of Things: Key Applications and Protocols.* Wiley, 2012.

[93] S. Sarkar, G. Saha, G. Pal, and T. Karmakar, “Indian experience on smart grid application in blackout control,” in *2015 39th National Systems Conference (NSC)*, pp. 1–6, 2015.

[94] USAID, “The smart grid vision for india’s power sector,” white paper, PA Government Services, Inc., 2010. [Online]. Available: `http://pdf.usaid.gov/pdf_docs/pbaaa526.pdf`. [Accessed: Feb. 13, 2018].

[95] J. Minkel, “The 2003 northeast blackout–five years later,” 2008. [Online]. Available: `https://www.scientificamerican.com/article/2003-blackout-five-years-later/`. [Accessed: Feb. 11, 2018].

[96] UCTE, “Final report of the investigation committee on the 28 September 2003 blackout in Italy,” UCTE Report, 2004. [Online]. Available: `https://www.entsoe.eu/fileadmin/user_upload/_library/publications/ce/otherreports/20040427_UCTE_IC_Final_report.pdf`. [Accessed: Feb. 13, 2018].

[97] G. Bryant, “Getting across the smart home threshold,” 2016. [Online]. Available: `https://newsroom.intel.com/editorials/getting-across-threshold/`. [Accessed: Aug. 15, 2017].

[98] Gartner, “Gartner says 6.4 billion connected ’things’ will be in use in 2016, up 30 percent from 2015,” 2015. [Online]. Available: http://www.gartner.com/newsroom/id/3165317. [Accessed: Aug. 15, 2017].

[99] Deloitte, “Switch on to the connected home,” Consumer Review OC303675, Deloitte, 2016. [Online]. Available: https://www2.deloitte.com/content/dam/Deloitte/uk/Documents/consumer-business/deloitte-uk-consumer-review-16.pdf. [Accessed: Aug. 15, 2017].

[100] Intel, “Introducing the Intel smart ‘tiny house’: Exploring smart home technology in 210 square feet,” fact sheet, Intel, 2015. [Online]. Available: http://download.intel.com/newsroom/kits/iot/pdfs/IntelTinyHomeFactSheet.pdf. [Accessed: Aug. 25, 2017].

[101] Google, “Google Assistant SDK,” 2017. [Online]. Available: https://developers.google.com/assistant/sdk/. [Accessed: Sept. 1, 2017].

[102] WHO, “Global status report on road safety 2015,” Report ISBN 978 92 4 156506 6, World Health Organization, 2015. [Online]. Available: http://www.who.int/violence_injury_prevention/road_safety_status/2015/en/. [Accessed: Nov. 28, 2017].

[103] GSMA, “GSMA smart cities guide: Traffic management,” report, GSMA, 2016. [Online]. Available: https://www.gsma.com/iot/wp-content/uploads/2017/03/Traffic-Management-guide-webv2.pdf. [Accessed: Nov. 21, 2017].

[104] Huawei, “NB-IOT - enabling new business opportunities,” white paper, 2015. [Online]. Available: http://www.huawei.com/minisite/iot/img/nb_iot_whitepaper_en.pdf. [Accessed: Nov. 21, 2017].

[105] S. Bera, S. Misra, and A. V. Vasilakos, “Software-Defined Networking for Internet of Things: a survey,” *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1994–2008, 2017.

[106] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian, “A software defined networking architecture for the Internet-of-Things,” in *2014 IEEE Network Operations and Management Symposium (NOMS)*, pp. 1–9, 2014.

[107] Y. Bi, C. Lin, H. Zhou, P. Yang, X. Shen, and H. Zhao, “Time-constrained Big Data transfer for SDN-enabled Smart City,” *IEEE Communications Magazine*, vol. 55, no. 12, pp. 44–50, 2017.

[108] N. Kushalnagar, G. Montenegro, and C. Schumacher, “IPv6 over low-power wireless personal area networks (6LoWPANs): Overview, assumptions, problem statement, and goals,” 2007. [Online]. Available: https://tools.ietf.org/html/rfc4919. [Accessed: Aug. 18, 2016].

[109] M. M. Mazhar, M. A. Jamil, A. Mazhar, A. Ellahi, M. S. Jamil, and T. Mahmood, “Conceptualization of software defined network layers over internet of things for future smart cities applications,” in *Wireless for Space and Extreme Environments (WiSEE), 2015 IEEE International Conference on*, pp. 1–4, 2015.

[110] S. Chakrabarty and D. W. Engels, “A secure IoT architecture for Smart Cities,” in *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pp. 812–813, 2016.

[111] M. S. Munir, S. F. Abedin, M. G. R. Alam, N. H. Tran, and C. S. Hong, "Intelligent service fulfillment for software defined networks in smart city," in *2018 International Conference on Information Networking (ICOIN)*, pp. 516–521, 2018.

[112] N. Z. Bawany, J. A. Shamsi, and K. Salah, "DDoS attack detection and mitigation using SDN: methods, practices, and solutions," *Arabian Journal for Science and Engineering*, vol. 42, no. 2, pp. 425–441, 2017.

[113] A. Manzalini and A. Stavdas, "The network is the robot," *Communications & Strategies*, vol. 4th quarter 2014, no. 96, p. 73, 2014.

[114] NERC, "Cyber security — BES cyber system categorization," 2016. [Online]. Available: `http://goo.gl/QBs37A`. [Accessed: Aug. 12, 2016].

[115] A. Sydney, D. S. Ochs, C. Scoglio, D. Gruenbacher, and R. Miller, "Using GENI for experimental evaluation of software defined networking in smart grids," *Computer Networks*, vol. 63, pp. 5 – 16, 2014. Special issue on Future Internet Testbeds - Part II.

[116] GENI, "About GENI," 2016. [Online]. Available: `https://www.geni.net/?page_id=2`. [Accessed: July 26, 2016].

[117] N. Dorsch, F. Kurtz, H. Georg, C. Hägerling, and C. Wietfeld, "Software-defined networking for smart grid communications: Applications, challenges and advantages," in *Smart Grid Communications (SmartGridComm), 2014 IEEE International Conference on*, pp. 422–427, 2014.

[118] A. Sydney, J. Nutaro, C. Scoglio, D. Gruenbacher, and N. Schulz, "Simulative comparison of multiprotocol label switching and OpenFlow network technologies for transmission operations," *IEEE Transactions on Smart Grid*, vol. 4, pp. 763–770, June 2013.

[119] D. Gyllstrom, N. Braga, and J. Kurose, "Recovery from link failures in a smart grid communication network using OpenFlow," in *Smart Grid Communications (SmartGridComm), 2014 IEEE International Conference on*, pp. 254–259, 2014.

[120] A. Goodney, S. Kumar, A. Ravi, and Y. H. Cho, "Efficient PMU networking with software defined networks," in *Smart Grid Communications (SmartGridComm), 2013 IEEE International Conference on*, pp. 378–383, 2013.

[121] Y. J. Kim, K. He, M. Thottan, and J. G. Deshpande, "Virtualized and self-configurable utility communications enabled by software-defined networks," in *Smart Grid Communications (SmartGridComm), 2014 IEEE International Conference on*, pp. 416–421, 2014.

[122] X. Dong, H. Lin, R. Tan, R. K. Iyer, and Z. Kalbarczyk, "Software-defined networking for smart grid resilience: Opportunities and challenges," in *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, CPSS '15, (New York, NY, USA), pp. 61–68, ACM, 2015.

[123] J. Zhang, B.-C. Seet, T.-T. Lie, and C. H. Foh, "Opportunities for software-defined networking in smart grid," in *Information, Communications and Signal Processing (ICICS) 2013 9th International Conference on*, pp. 1–5, 2013.

[124] E. G. da Silva, L. A. D. Knob, J. A. Wickboldt, L. P. Gaspary, L. Z. Granville, and A. Schaeffer-Filho, "Capitalizing on SDN-based SCADA systems: An anti-eavesdropping case-study," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 165–173, 2015.

[125] B. Genge, F. Graur, and P. Haller, "Experimental assessment of network design approaches for protecting industrial control systems," *Int. J. Crit. Infrastruct. Prot.*, vol. 11, no. C, pp. 24–38, 2015.

[126] Y. Lopes, N. C. Fernandes, C. M. Bastos, and D. C. Muchaluat-Saade, "SMARTFlow: A solution for autonomic management and control of communication networks for smart grids," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, (New York, NY, USA), pp. 2212–2217, ACM, 2015.

[127] ONF, "OpenFlow," 2016. [Online]. Available: `https://www.opennetworking.org/sdn-resources/openflow`. [Accessed: Aug. 8, 2016].

[128] T. Pfeiffenberger, J. L. Du, P. B. Arruda, and A. Anzaloni, "Reliable and flexible communications for power systems: Fault-tolerant multicast with SDN/OpenFlow," in *2015 7th International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–6, 2015.

[129] S. Fang, Y. Yu, C. H. Foh, and K. M. M. Aung, "A loss-free multipathing solution for data center network using software-defined networking approach," in *APMRC, 2012 Digest*, pp. 1–8, 2012.

[130] Y. Y. Huang, M. W. Lee, T. Y. Fan-Chiang, X. Huang, and C. H. Hsu, "Minimizing flow initialization latency in software defined networks," in *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific*, pp. 303–308, 2015.

[131] E. Molina, E. Jacob, J. Matias, N. Moreira, and A. Astarloa, "Using software defined networking to manage and control IEC 61850-based systems," *Computers & Electrical Engineering*, vol. 43, pp. 142 – 154, 2015.

[132] A. Mckeown, H. Rashvand, T. Wilcox, and P. Thomas, "Priority SDN controlled integrated wireless and powerline wired for smart-home internet of things," in *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing, Autonomic and Trusted Computing, Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*, pp. 1825–1830, 2015.

[133] S. A. Lazar and C. E. Stefan, "Future vehicular networks: What control technologies?," in *2016 International Conference on Communications (COMM)*, pp. 337–340, 2016.

[134] W. F. Elsadek and M. N. Mikhail, "IP mobility management using software defined networking: A review," in *2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pp. 76–81, 2017.

[135] R. Hwang, H. Tseng, and Y. Tang, "Design of SDN-enabled cloud data center," in *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, pp. 950–957, 2015.

[136] P. Goransson and C. Black, *Software Defined Networks: A Comprehensive Approach.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1st ed., 2014.

[137] S. Lee, K. Li, K. Chan, J. YwiChi, T. Lee, W. Liu, and Y. Lin, “Design of SDN based large multi-tenant data center networks,” in *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, pp. 44–50, 2015.

[138] R. R. Krishnan and N. Figueira, “Analysis of data center SDN controller architectures: Technology and business impacts,” in *Computing, Networking and Communications (ICNC), 2015 International Conference on*, pp. 104–109, 2015.

[139] J. Hwang, J. Yoo, S. H. Lee, and H. W. Jin, “Scalable congestion control protocol based on SDN in data center networks,” in *2015 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2015.

[140] B. Andrus, J. V. Olmos, V. Mehmeri, I. Monroy, S. Spolitis, and V. Bobrovs, “SDN data center performance evaluation of torus and hypercube interconnecting schemes,” in *Advances in Wireless and Optical Communications (RTUWO), 2015*, pp. 110–112, 2015.

[141] L. Chen, M. Qiu, and J. Xiong, “An SDN-based fabric for flexible data-center networks,” in *Cyber Security and Cloud Computing (CSCloud), 2015 IEEE 2nd International Conference on*, pp. 121–126, 2015.

[142] B. Flavio, M. Rodolfo, Z. Jiang, and A. Sateesh, “Fog computing and its role in the internet of things,” in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC ’12, (New York, NY, USA), pp. 13–16, ACM, 2012.

[143] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, “Mobile edge computing a key technology towards 5G,” ETSI White Paper 11, European Telecommunications Standards Institute, 2015. ISBN: 979-10-92620-08-5.

[144] O. Salman, I. Elhajj, A. Kayssi, and A. Chehab, “Edge computing enabling the internet of things,” in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*, pp. 603–608, 2015.

[145] O. Salman, I. Elhajj, A. Kayssi, and A. Chehab, “An architecture for the internet of things with decentralized data and centralized control,” in *2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)*, pp. 1–8, 2015.

[146] A. Sonba and H. Abdalkreim, “Performance comparison of the state of the art OpenFlow controllers,” master thesis, Halmstad University, 2014. [online] Available: `http://goo.gl/YupmKF`, accessed Aug. 9, 2016.

[147] C. Nakasan, “Design and implementation of OpenFlow-based routing optimization for MPTCP,” master thesis, Nara Institute of Science and Technology, 2015. [Online]. Available: `https://goo.gl/EYX77q`. [Accessed: Aug. 9, 2016].

[148] R. M. Hirannaiah, “OpenFlow based traffic engineering for mobile devices,” dissertation, Wichita State University, 2014. [Online]. Available: `http://goo.gl/DgxxCA`. [Accessed: Aug. 9, 2016].

[149] I. Ku, “Software-defined mobile cloud,” dissertation, University of California, 2014. [Online]. Available: `http://goo.gl/rHh0Vm`. [Accessed: Aug. 9, 2016].

[150] Trunomi, “GDPR portal: site overview,” 2018. [Online]. Available: `https://www.eugdpr.org/`. [Accessed: May 21, 2018].

[151] Cisco, "Cisco Meraki — Mobile device management," 2018. [Online]. Available: https://meraki.cisco.com/solutions/mobile-device-management. [Accessed: Mar. 17, 2018].

[152] W. Wang, Y. Xu, and M. Khanna, "A survey on the communication architectures in smart grid," *Computer Networks*, vol. 55, no. 15, pp. 3604 – 3629, 2011.

[153] Intel, "Making buildings smart," tech. rep., Intel, 2015. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/making-buildings-smart-ebook-vol-1.pdf. [Accessed: Feb. 7, 2018].

[154] Y. V. P. Kumar and R. Bhimasingu, "Review and retrofitted architectures to form reliable smart microgrid networks for urban buildings," *IET Networks*, vol. 4, no. 6, pp. 338–349, 2015.

[155] W. Peat, C. Higgins, and J. Whyte, "Smart building potential within heavily utilised networks," *CIRED - Open Access Proceedings Journal*, vol. 2017, no. 1, pp. 1466–1470, 2017.

[156] ETSI SG Group, "Smart grid reference architecture," Technical Report. Mandate M/490 v3.0, CEN-CENELEC-ETSI Smart Grid Coordination Group, 2012. [Online]. Available: http://gridscientific.com/images/Smart_Grid_Reference_Artichtecture.pdf. [Accessed: Aug. 31, 2017].

[157] M. Kuzlu, M. Pipattanasompom, and S. Rahman, "A comprehensive review of smart grid related standards and protocols," in *2017 5th International Istanbul Smart Grid and Cities Congress and Fair (ICSG)*, pp. 12–16, 2017.

[158] IEC, "IEC 61850:2017 SER Series," 2017. International Standard. [Online]. Available: https://webstore.iec.ch/publication/6028#additionalinfo. [Accessed: Jan. 13, 2018].

[159] A. Lisowiec and A. Nowakowski, "Modern IED in today's smart grids," in *2013 International Conference on Clean Electrical Power (ICCEP)*, pp. 288–292, 2013.

[160] R. E. Mackiewicz, "Overview of IEC 61850 and benefits," in *2006 IEEE PES Power Systems Conference and Exposition*, pp. 623–630, 2006.

[161] H. Heine, P. Guenther, and F. Becker, "New non-conventional instrument transformer (NCIT) - a future technology in gas insulated switchgear," in *2016 IEEE/PES Transmission and Distribution Conference and Exposition (T D)*, pp. 1–5, 2016.

[162] I. Ali, M. S. Thomas, S. Gupta, and S. M. S. Hussain, "IEC 61850 substation communication network architecture for efficient energy system automation," *Energy Technology & Policy*, vol. 2, no. 1, pp. 82–91, 2015.

[163] IEEE, "IEEE Std 1646-2004 - IEEE standard communication delivery time performance requirements for electric power substation automation," 2005. [Online]. Available: https://standards.ieee.org/findstds/standard/1646-2004.html. [Accessed: Jan. 17, 2018].

[164] N. Komninos, E. Philippou, and A. Pitsillides, "Survey in smart grid and smart home security: Issues, challenges and countermeasures," *IEEE Communications Surveys Tutorials*, vol. 16, pp. 1933–1954, Fourthquarter 2014.

[165] B. Zhou, W. Li, K. W. Chan, Y. Cao, Y. Kuang, X. Liu, and X. Wang, “Smart home energy management systems: Concept, configurations, and scheduling strategies,” *Renewable and Sustainable Energy Reviews*, vol. 61, pp. 30 – 40, 2016.

[166] A. Kailas, V. Cecchi, and A. Mukherjee, “A survey of communications and networking technologies for energy management in buildings and home automation,” *Journal of Computer Networks and Communications*, 2012.

[167] B. Butler, “Worldwide enterprise WLAN market sees steady growth in full year and Q4 2017, according to IDC,” 2018. [Online]. Available: `https://www.idc.com/getdoc.jsp?containerId=prUS43599518`. [Accessed: Mar. 09, 2018].

[168] Cisco, “802.11ac: The fifth generation of Wi-Fi,” Technical White Paper C11-713103-05, Cisco, 2018. [Online]. Available: `https://www.cisco.com/c/dam/en/us/products/collateral/wireless/aironet-3600-series/white-paper-c11-713103.pdf`. [Accessed: Mar. 11, 2018].

[169] IEEE, “IEEE Std 1609.0-2013 - IEEE guide for wireless access in vehicular environments (WAVE) - architecture,” 2013. [Online]. Available: `https://standards.ieee.org/findstds/standard/1609.0-2013.html`. [Accessed: Jan. 10, 2018].

[170] B. Li, M. S. Mirhashemi, X. Laurent, and J. Gao, “Wireless access for vehicular environments,” final report, Chalmers University of Technology, 2011. [Online]. Available: `https://pdfs.semanticscholar.org/8702/4b92ac23dc6e1a4322f760b77e93914826ae.pdf`. [Accessed: Jan. 8, 2018].

[171] A. Filippi, K. Moerman, G. Daalderop, P. D. Alexander, F. Schober, and W. Pfliegl, “Ready to roll: Why 802.11p beats LTE and 5G for V2x,” white paper, NXP Semiconductors, Cohda Wireless, and Siemens, 2016. [Online]. Available: `http://www.eenewsautomotive.com/design-center/why-80211p-beats-lte-and-5g-v2x`. [Accessed: Nov. 28, 2017].

[172] Z. Hameed Mir and F. Filali, “LTE and IEEE 802.11p for vehicular networking: a performance evaluation,” *EURASIP Journal on Wireless Communications and Networking*, vol. 2014, no. 1, p. 89, 2014.

[173] F. Holik, J. Horalek, S. Neradova, M. Novak, and O. Marik, “Communication technology for smart heat metering,” in *2015 25th International Conference Radioelektronika (RADIOELEKTRONIKA)*, pp. 405–410, 2015.

[174] IEEE, “IEEE 802.3 Ethernet working group,” 2017. [Online]. Available: `http://www.ieee802.org/3/`. [Accessed: Dec. 29, 2017].

[175] ITU-T, “X.1035 - Password-authenticated key exchange (PAK) protocol,” 2007. [Online]. Available: `https://www.itu.int/rec/T-REC-X.1035-200702-I/en`. [Accessed: Apr. 11, 2018].

[176] IETF, “IPv6 over Low power WPAN (6lowpan),” 2017. [Online]. Available: `https://datatracker.ietf.org/wg/6lowpan/about/`. [Accessed: Jan. 4, 2018].

[177] IEEE, “IEEE 802.15 WPAN task group 1,” 2018. [Online]. Available: `http://www.ieee802.org/15/pub/TG1.html`. [Accessed: Jan. 7, 2018].

[178] Bluetooth SIG, "Bluetooth core specification v 5.0," 2016. [Online]. Available: `https://tinyurl.com/yd9ppyoa`. [Accessed: Jan. 7, 2018].

[179] Bluetooth SIG, "Bluetooth: Topology options," 2018. [Online]. Available: `https://www.bluetooth.com/bluetooth-technology/topology-options`. [Accessed: Jan. 7, 2018].

[180] A. Mulla, J. Baviskar, S. Khare, and F. Kazi, "The wireless technologies for smart grid communication: A review," in *2015 Fifth International Conference on Communication Systems and Network Technologies*, pp. 442–447, 2015.

[181] K. Naji, "Latency in LTE systems," 2012. [Online]. Available: `http://cwi.unik.no/images/Latency_in_LTE_comments.pdf`. [Accessed: Jan. 8, 2018].

[182] 3GPP, "About 3GPP home," 2018. [Online]. Available: `http://www.3gpp.org/about-3gpp/about-3gpp`. [Accessed: Jan. 8, 2018].

[183] 3GPP, "LTE," 2018. [Online]. Available: `http://www.3gpp.org/technologies/keywords-acronyms/98-lte`. [Accessed: Jan. 8, 2018].

[184] P. Burns, "LoRaWAN vs Haystack," dec 2016. [Online]. Available: `https://www.slideshare.net/haystacktech/how-to-disrupt-the-internet-of-things-with-unified-networking`. [Accessed: Feb. 1, 2018].

[185] M. Weyn, G. Ergeerts, L. Wante, C. Vercauteren, and P. Hellinckx, "Survey of the DASH7 alliance protocol for 433 MHz wireless sensor communication," *International Journal of Distributed Sensor Networks*, vol. 9, no. 12, 2013. [Online]. Available: `http://journals.sagepub.com/doi/full/10.1155/2013/870430`. [Accessed: Feb. 1, 2018].

[186] H. Lee, S. H. Chung, Y. S. Lee, and Y. Ha, "Performance comparison of DASH7 and ISO/IEC 18000-7 for fast tag collection with an enhanced CSMA/CA protocol," in *2013 IEEE 10th International Conference on High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pp. 769–776, 2013.

[187] ISO/IEC, "ISO/IEC 14543-3-10:2012," 2012. [Online]. Available: `https://www.iso.org/standard/59865.html`. [Accessed: Jan. 5, 2018].

[188] EnOcean, "EnOcean – the world of energy harvesting wireless technology," white paper, EnOcean. [Online]. Available: `https://www.enocean.com/fileadmin/redaktion/pdf/white_paper/White_Paper_Getting_Started_With_EnOcean.pdf`. [Accessed: Jan. 5, 2018].

[189] F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui, and T. Watteyne, "Understanding the limits of LoRaWAN," *IEEE Communications Magazine*, vol. 55, no. 9, pp. 34–40, 2017.

[190] LoRa Alliance, "LoRa Alliance Technology," 2017. [Online]. Available: `https://www.lora-alliance.org/technology`. [Accessed: Jan. 10, 2018].

[191] N. Sornin, M. Luis, T. Eirich, T. Kramp, and O.Hersent, "LoRaWAN Specification," 2015.

[192] S. Tabbane, "Internet of things: A technical overview of the ecosystem." Regional Workshop for Africa on "Developing the ICT ecosystem to harness Internet-of-Things (IoT)", 2017. [Online]. Available: https://www.itu.int/en/ITU-D/Regional-Presence/Africa/Documents/IoT_Technical%20Overview%20of%20the%20Ecosystem_v2_27062017.pdf. [Accessed: Feb. 17, 2018].

[193] Sigfox, "Sigfox technology overview," 2017. [Online]. Available: https://www.sigfox.com/en/sigfox-iot-technology-overview. [Accessed: Feb. 16, 2018].

[194] U. Raza, P. Kulkarni, and M. Sooriyabandara, "Low power wide area networks: An overview," *IEEE Communications Surveys Tutorials*, vol. 19, no. 2, pp. 855–873, 2017.

[195] Weightless SIG, "Weightless specification," 2015. [Online]. Available: http://www.weightless.org/about/weightless-specification. [Accessed: Feb. 17, 2018].

[196] Weightless SIG, "Weightless-P system specification," 2017. System specification, version 1.03.

[197] IEEE, "IEEE 802.11 Wireless local area networks," 2017. [Online]. Available: http://www.ieee802.org/11/. [Accessed: Jan. 5, 2018].

[198] IEEE, "The IEEE 802.16 working group on broadband wireless access standards," 2017. [Online]. Available: http://www.ieee802.org/16/. [Accessed: Jan. 8, 2018].

[199] H. Kassim and M. D. Baba, "Performance analysis of fixed and mobile WiMax networks using NCTUns tools," in *2011 IEEE Control and System Graduate Research Colloquium*, pp. 159–165, 2011.

[200] IEEE, "IEEE Std 802.15.4-2015 (revision of IEEE std 802.15.4-2011) - IEEE standard for low-rate wireless networks," 2015. [Online]. Available: https://standards.ieee.org/findstds/standard/802.15.4-2015.html. [Accessed: Jan. 5, 2018].

[201] Z. Alliance, "Zigbee 3.0," 2017. [Online]. Available: http://www.zigbee.org/zigbee-for-developers/zigbee-3-0/. [Accessed: Jan. 5, 2018].

[202] Zigbee Alliance, "Zigbee resource guide," resource guide, Zigbee Alliance, 2017. [Online]. Available: http://www.nxtbook.com/nxtbooks/webcom/zigbee_rg2017/. [Accessed: Jan. 5, 2018].

[203] P. Yi, A. Iwayemi, and C. Zhou, "Developing ZigBee deployment guideline under WiFi interference for smart grid applications," *IEEE Transactions on Smart Grid*, vol. 2, no. 1, pp. 110–120, 2011.

[204] ZWAVE, "How Z-Wave works.," 2018. [Online]. Available: http://www.z-wave.com/about#how-it-works. [Accessed: Jan. 5, 2018].

[205] EPRI, "IEC61850 part 7-2 - abstract common services interface (ACSI)," 2004. [online] Available: https://tinyurl.com/ybyov4oc. [Accessed: Jan. 14, 2018].

[206] IEEE, "IEEE Std 802.22a-2014," 2014. [Online]. Available: https://standards.ieee.org/findstds/standard/802.22a-2014.html. [Accessed: Jan. 16, 2018].

[207] V. Dehalwar, A. Kalam, M. L. Kolhe, and A. Zayegh, "Review of IEEE 802.22 and IEC 61850 for real-time communication in Smart Grid," in *2015 International Conference on Computing and Network Communications (CoCoNet)*, pp. 571–575, 2015.

[208] ISO, "Industrial automation systems — manufacturing message specification — part 1: Service definition," 2003. [Online]. Available: `https://www.iso.org/obp/ui/#iso:std:iso:9506:-1:ed-2:v1:en`. [Accessed: Jan. 14, 2018].

[209] W. Wang, M. Liu, X. Zhao, and G. Yang, "Shared-network scheme of SMV and GOOSE in smart substation," *Journal of Modern Power Systems and Clean Energy*, vol. 2, no. 4, pp. 438–443, 2014.

[210] D. Mills, "Simple network time protocol (SNTP) version 4 for IPv4, IPv6 and OSI," 2006. Request for Comments: 4330. [Online]. Available: `https://tools.ietf.org/html/rfc4330`. [Accessed: Jan. 14, 2018].

[211] IEEE, "IEEE Std 1588-2008 (revision of IEEE std 1588-2002) - IEEE standard for a precision clock synchronization protocol for networked measurement and control systems," 2008. [Online]. Available: `https://standards.ieee.org/findstds/standard/1588-2008.html`. [Accessed: Jan. 14, 2018].

[212] J. B. Kenney, "Dedicated short-range communications (DSRC) standards in the United States," *Proceedings of the IEEE*, vol. 99, no. 7, pp. 1162–1182, 2011.

[213] J. Wu, H. Lu, and Y. Xiang, "Measurement and comparison of sub-1GHz and IEEE 802.11p in vehicular networks," in *2017 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1063–1066, 2017.

[214] S. Demmel, A. Lambert, D. Gruyer, A. Rakotonirainy, and E. Monacelli, "Empirical IEEE 802.11p performance evaluation on test tracks," in *2012 IEEE Intelligent Vehicles Symposium*, pp. 837–842, 2012.

[215] G. Dimitrakopoulos, *Current technologies in vehicular communication*, vol. 1. Springer International Publishing, 2017.

[216] A. Abdeldime and L. Wu, "The physical layer of the IEEE 802.11p WAVE communication standard: The specifications and challenges," in *Lecture Notes in Engineering and Computer Science*, vol. 2, 2014.

[217] B. E. Bilgin and V. C. Gungor, "Performance comparison of IEEE 802.11p and IEEE 802.11b for vehicle-to-vehicle communications in highway, rural, and urban areas," *International Journal of Vehicular Technology*, 2013.

[218] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (CoAP)," 2014. Request for Comments 7252, ISSN: 2070-1721.

[219] J. Olenski, "SSL vs. TLS - what's the difference?," 2016. [Online]. Available: `https://www.globalsign.com/en/blog/ssl-vs-tls-difference/`. [Accessed: Aug. 24, 2017].

[220] H. Grace, "TLS/SSL vulnerabilities," 2017. [Online]. Available: `https://www.gracefulsecurity.com/tls-ssl-vulnerabilities/`. [Accessed: Aug. 24, 2017].

[221] NIST, "Framework for improving critical infrastructure cybersecurity," 2014. Cybersecurity Framework 1.0. [Online]. Available: https://www.nist.gov/sites/default/files/documents/cyberframework/cybersecurity-framework-021214.pdf. [Accessed: Sept. 4, 2017].

[222] OASIS, "MQTT version 3.1.1," 2014. OASIS Standard. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html. [Accessed: Aug. 24, 2017].

[223] C. Ellison, "UPnP security ceremonies design document," 2003. [Online]. Available: http://upnp.org/specs/sec/UPnP-sec-UPnPSecurityCeremonies-v1.pdf. [Accessed: Mar. 24, 2018].

[224] XMPP, "Internet of Things (IoT)," 2017. [Online]. Available: https://xmpp.org/uses/internet-of-things.html. [Accessed: Sept. 1, 2017].

[225] IETF, "Extensible messaging and presence protocol (XMPP): Core," 2011. [Online]. Available: https://tools.ietf.org/html/rfc6120. [Accessed: Sept. 1, 2017].

[226] OMG, "Data distribution service (DDS)," 2015. [Online]. Available: http://www.omg.org/spec/DDS/1.4. [Accessed: Sept. 4, 2017].

[227] IETF, "Hypertext transfer protocol – HTTP/1.1," 1999. [Online]. Available: https://tools.ietf.org/html/rfc2616. [Accessed: Aug. 24, 2017].

[228] OASIS, "OASIS advanced message queuing protocol (AMQP) Version 1.0," 2012. OASIS Standard. [Online]. Available: http://docs.oasis-open.org/amqp/core/v1.0/amqp-core-transport-v1.0.html. [Accessed: Aug. 24, 2017].

[229] E. Rescorla and N. Modadugu, "Datagram transport layer security version 1.2," 2012. Request for Comments: 6347, ISSN: 2070-1721. [Online]. Available: https://tools.ietf.org/html/rfc6347. [Accessed: Jan. 18, 2018].

[230] S. Frankel and S. Krishnan, "IP security (IPsec) and internet key exchange (IKE) document roadmap," 2011. Request for Comments: 6071. ISSN: 2070-1721. [Online]. Available: https://tools.ietf.org/html/rfc6071. [Accessed: Jan. 18, 2018].

[231] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," 1996. Request for Comments: 1889. [Online]. Available: https://tools.ietf.org/html/rfc1889. [Accessed: Jan. 18, 2018].

[232] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman, "The secure real-time transport protocol (SRTP)," 2004. Request for Comments: 3711. [Online]. Available: https://tools.ietf.org/html/rfc3711. [Accessed: Jan. 18, 2018].

[233] T. Dierks and E. Rescorla, "The transport layer security (TLS) protocol version 1.2," 2008. Request for Comments: 5246. [Online]. Available: https://tools.ietf.org/html/rfc5246. [Accessed: Jan. 18, 2018].

[234] C. Cerrudo, "An emerging US (and world) threat: Cities wide open to cyber attacks," white paper, IOActive, 2015. [Online]. Available: https://ioactive.com/pdfs/IOActive_HackingCitiesPaper_CesarCerrudo.pdf. [Accessed: Jan. 19, 2018].

[235] R. Nixon, "Homeland security dept. struggles to hire staff to combat cyberattacks," 2016. [Online]. Available: `https://goo.gl/URXbvn`. [Accessed: Mar. 25, 2018].

[236] E. Rosenberg and M. Salam, "Hacking attack woke up Dallas with emergency sirens, Officials Say," 2017. [Online]. Available: `https://goo.gl/cDY6NX`. [Accessed: Mar. 25, 2018].

[237] D. Eckhoff and I. Wagner, "Privacy in the smart city - applications, technologies, challenges, and solutions," *IEEE Communications Surveys Tutorials*, vol. 20, pp. 489–516, Firstquarter 2018.

[238] ICS-CERT, "Cyber-attack against Ukrainian critical infrastructure," 2016. [Online]. Available: `https://ics-cert.us-cert.gov/alerts/IR-ALERT-H-16-056-01`. [Accessed: Mar. 25, 2018].

[239] C. Kalloniatis, E. Kavakli, and E. Kontellis, "PRIS tool: A case tool for privacy-oriented Requirements Engineering," vol. 6, pp. 3–19, 01 2010.

[240] T. Myerson and M. Rogers, "Windows 10 privacy journey continues: more transparency and controls for you," 2017. [Online]. Available: `https://goo.gl/XU1BjL`. [Accessed: Feb. 23, 2018].

[241] A. Barbir, S. Murphy, and Y. Yang, "Generic threats to routing protocols," Request for Comments Request for Comments: 4593, IETF, 2006. [Online]. Available: `https://tools.ietf.org/html/rfc4593#section-4.5`. [Accessed: Jan. 27, 2018].

[242] D. Jacoby, "IoT: How I hacked my home," 2014. [Online]. Available: `https://securelist.com/iot-how-i-hacked-my-home/66207/`. [Accessed: Jan. 29, 2018].

[243] S. Malladi, J. Alves-Foss, and R. B. Heckendorn, "On preventing replay attacks on security protocols," in *In Proc. International Conference on Security and Management*, pp. 77–83, 2002.

[244] G. Kakarontzas, L. Anthopoulos, D. Chatzakou, and A. Vakali, "A conceptual enterprise architecture framework for smart cities: A survey based approach," in *2014 11th International Conference on e-Business (ICE-B)*, pp. 47–54, 2014.

[245] H. Nissenbaum, "Privacy as contextual integrity," *Washington Law Review*, vol. 79, no. 1, pp. 119–158, 2004.

[246] J.-H. Hoepman, *Privacy Design Strategies*, pp. 446–459. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014.

[247] ISO, "ISO/IEC 29100:2011," 2011.

[248] F. Alam, R. Mehmood, I. Katib, N. N. Albogami, and A. Albeshri, "Data fusion and IoT for smart ubiquitous environments: A survey," *IEEE Access*, vol. 5, pp. 9533–9554, 2017.

[249] ISO, "ISO/IEC 25010:2011," 2011.

[250] S. Djahel, R. Doolan, G. M. Muntean, and J. Murphy, "A communications-oriented perspective on traffic management systems for smart cities: Challenges and innovative approaches," *IEEE Communications Surveys Tutorials*, vol. 17, pp. 125–151, Firstquarter 2015.

[251] W. Zhou, L. Li, M. Luo, and W. Chou, "REST API design patterns for SDN northbound API," 2014.

[252] European Commission, "Cookies," 2016. [Online]. Available: `http://ec.europa.eu/ipg/basics/legal/cookies/index_en.htm`. [Accessed: Feb. 28, 2018].

[253] J. Mokráček, "Analysis of SDN controllers," bachelor thesis, University of Pardubice, 2017. [Online]. Available: `https://dk.upce.cz/bitstream/handle/10195/68099/MokracekJ_AnalyzaKontroleru_FH_2017.pdf?sequence=3&isAllowed=y`. [Accessed: Jan. 20, 2018].

[254] N. Naik, P. Jenkins, and D. Newell, "Choice of suitable identity and access management standards for mobile computing and communication," in *2017 24th International Conference on Telecommunications (ICT)*, pp. 1–6, 2017.

[255] A. Jøsang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decision Support Systems*, vol. 43, no. 2, pp. 618 – 644, 2007. Emerging Issues in Collaborative Commerce.

[256] M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about sdn flow tables," in *Passive and Active Measurement* (J. Mirkovic and Y. Liu, eds.), (Cham), pp. 347–359, Springer International Publishing, 2015.

[257] Aruba, "Aruba 3810 switch series," data sheet, 2017. [Online]. Available: `http://www.arubanetworks.com/assets/ds/DS_3810SwitchSeries.pdf`. [Accessed: Jan. 20, 2018].

[258] Hewlett-Packard, "HPE switch software OpenFlow v1.3 administrator guide K/KA/KB/WB 16.01," tech. rep., Hewlett-Packard, 2016. [Online]. Available: `https://support.hpe.com/hpsc/doc/public/display?sp4ts.oid=1827663&docLocale=en_US&docId=emr_na-c04943224`. [Accessed: Jan. 20, 2018].

[259] Visit Lancashire, "Lancaster," 2018. [Online]. Available: `https://www.visitlancashire.com/explore/lancaster`. [Accessed: May. 21, 2018].

Published papers in association with thesis

[1] F. Holik, J. Horalek: "Security Principles of Smart Grid Networks," in *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 2018, vol. 10, pp. 41-45. ISSN: 2180-1843

[2] J. Horalek, F. Holik: "Analysis of Threats and Attacks Impacts on Smart Grid Networks," in *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 2017, vol. 9, pp. 65-69. ISSN: 2180-1843

[3] F. Holik, J. Horalek: "Implementing SDN Into Computer Network Lessons," in *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 2017, vol. 9, pp. 65-69. ISSN: 2180-1843

[4] J. Horalek, F. Holik, V. Hurtova: "Implementation and testing of Cisco IP SLA in smart grid environments," in *International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, 2017, pp. 450-454.

[5] F. Holik, S. Neradova: "Vulnerabilities of modern web applications" in *International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, 2017, pp. 1256-1261.

[6] F. Holik, J. Horalek, S. Neradova, M. Novak, O. Marik: "Communication technology for smart heat metering," in *Proceedings of 25th International Conference Radioelektronika, RADIOELEKTRONIKA 2015*, art. no. 7128983, pp. 405-410, 2015. doi: 10.1109/RADIOELEK.2015.7128983

[7] J. Horalek, S. Neradova, S. Karamazov, F. Holik, O. Marik, S. Zitta: "Proposal to Centralize Operational Data Outputs of Airport Facilities," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 346-354, 2015. doi: 10.1007/978-3-319-24306-1_34

[8] J. Horalek, F. Holik, S. Karamazov, S. Neradova: "Effects of Integrating Cisco Networking Academy Program into Lessons on Computer Networks," in *International Conference on Computer Science and Environmental Engineering (CSEE). 1. Beijing, PEOPLES R CHINA: Destech Publicat Inc*, pp. 1560-1567, 2015. ISBN: 978-1-60595-240-6

[9] F. Holik, J. Horalek, O. Marik, S. Neradova, S. Zitta: "Implementing Multicast in Education Area of Computer Networks," in *International Conference on Computer Science and Environmental Engineering (CSEE). 1. Beijing, PEOPLES R CHINA: Destech Publicat Inc*, pp. 1568-1574, 2015. ISBN: 978-1-60595-240-6

[10] F. Holik, J. Horalek, O. Marik, S. Neradova, S. Zitta: "The methodology of measuring throughput of a wireless network," in *CINTI 2014 - 15th IEEE International Symposium on Computational Intelligence and Informatics, Proceedings*, art. no. 7028689, pp. 279-284, 2014. doi: 10.1109/CINTI.2014.7028689

Accepted papers in association with thesis

[1] F. Holik: "Requirements on SDN Applications for Smart Cities," in *5th International Conference on Communication and Computer Engineering (ICOCOE)*. Malaysia, July 2018.

[2] F. Holik: "SDN Security Protection Methods for IoT Networks," in *5th International Conference on Communication and Computer Engineering (ICOCOE)*. Malaysia, July 2018.

[3] F. Holik, S. Neradova: "Using Raspberry Pi as an SDN Enabled Switch," in *5th International Conference on Communication and Computer Engineering (ICOCOE)*. Malaysia, July 2018.

Note

These papers were accepted and presented in conferences, but they were not yet published. The papers are expected to be published in the following journals:

- Journal of Telecommunication, Electronic and Computer Engineering (SCOPUS)
- Journal of Advanced Manufacturing Technology (JAMT) (SCOPUS)

Attachments

Attachment A - SC Topology Script

This attachment shows the complete *Python* script, which creates the topology of the use case *Lancaster Smart City*.

```python
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import RemoteController, Host, OVSKernelSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info

def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8')

    info( '*** Adding controller\n' )
    cont=net.addController(name='lan_cont',
                      controller=RemoteController,
                      ip='127.0.0.1',
                      protocol='tcp',
                      port=6633)

    info( '*** Add switches\n')
    dtc1s = net.addSwitch('DTC1S', cls=OVSKernelSwitch, dpid='1')
    lan1s = net.addSwitch('LAN1S', cls=OVSKernelSwitch, dpid='2')
    lan2s = net.addSwitch('LAN2S', cls=OVSKernelSwitch, dpid='3')
    lan3s = net.addSwitch('LAN3S', cls=OVSKernelSwitch, dpid='4')
    mor1s = net.addSwitch('MOR1S', cls=OVSKernelSwitch, dpid='5')
    mor2s = net.addSwitch('MOR2S', cls=OVSKernelSwitch, dpid='6')
    mor3s = net.addSwitch('MOR3S', cls=OVSKernelSwitch, dpid='7')
    hey1s = net.addSwitch('HEY1S', cls=OVSKernelSwitch, dpid='8')
    hey2s = net.addSwitch('HEY2S', cls=OVSKernelSwitch, dpid='9')
    hey3s = net.addSwitch('HEY3S', cls=OVSKernelSwitch, dpid='A')
    sco1s = net.addSwitch('SCO1S', cls=OVSKernelSwitch, dpid='B')
    sco2s = net.addSwitch('SCO2S', cls=OVSKernelSwitch, dpid='C')
    sco3s = net.addSwitch('SCO3S', cls=OVSKernelSwitch, dpid='D')
    hwf1s = net.addSwitch('HWF1S', cls=OVSKernelSwitch, dpid='E')
    lss1s = net.addSwitch('LSS1S', cls=OVSKernelSwitch, dpid='F')
    hpp1s = net.addSwitch('HPP1S', cls=OVSKernelSwitch, dpid='10')

    info( '*** Add hosts\n')
    ser001_dtc = net.addHost('SER001_DTC', cls=Host, ip='10.1.0.1', mac='00:00:00:00:00:01')
    ser002_dtc = net.addHost('SER002_DTC', cls=Host, ip='10.1.0.2', mac='00:00:00:00:00:02')

    sap001_lan = net.addHost('SAP001_LAN', cls=Host, ip='10.2.1.1', mac='00:00:00:00:00:11')
    sip001_lan = net.addHost('SIP001_LAN', cls=Host, ip='10.2.2.1', mac='00:00:00:00:00:12')
    stl001_lan = net.addHost('STL001_LAN', cls=Host, ip='10.2.3.1', mac='00:00:00:00:00:13')
    stl002_lan = net.addHost('STL002_LAN', cls=Host, ip='10.2.3.2', mac='00:00:00:00:00:14')
    sip002_lan = net.addHost('SIP002_LAN', cls=Host, ip='10.2.2.2', mac='00:00:00:00:00:15')
    sap002_lan = net.addHost('SAP002_LAN', cls=Host, ip='10.2.1.2', mac='00:00:00:00:00:16')
    sap003_lan = net.addHost('SAP003_LAN', cls=Host, ip='10.2.1.3', mac='00:00:00:00:00:17')
    stl003_lan = net.addHost('STL003_LAN', cls=Host, ip='10.2.3.3', mac='00:00:00:00:00:18')

    stl001_mor = net.addHost('STL001_MOR', cls=Host, ip='10.3.3.1', mac='00:00:00:00:00:21')
    sps001_mor = net.addHost('SPS001_MOR', cls=Host, ip='10.3.4.1', mac='00:00:00:00:00:22')
    sps004_mor = net.addHost('SPS004_MOR', cls=Host, ip='10.3.4.4', mac='00:00:00:00:00:23')
    evs001_mor = net.addHost('EVS001_MOR', cls=Host, ip='10.3.5.1', mac='00:00:00:00:00:24')
    stl002_mor = net.addHost('STL002_MOR', cls=Host, ip='10.3.3.2', mac='00:00:00:00:00:25')
    sps002_mor = net.addHost('SPS002_MOR', cls=Host, ip='10.3.4.2', mac='00:00:00:00:00:26')
    evs002_mor = net.addHost('EVS002_MOR', cls=Host, ip='10.3.5.2', mac='00:00:00:00:00:27')
    sps003_mor = net.addHost('SPS003_MOR', cls=Host, ip='10.3.4.3', mac='00:00:00:00:00:28')

    sip001_hey = net.addHost('SIP001_HEY', cls=Host, ip='10.4.2.1', mac='00:00:00:00:00:31')
    sts001_hey = net.addHost('STS001_HEY', cls=Host, ip='10.4.7.1', mac='00:00:00:00:00:32')
```

```
sls001_hey = net.addHost('SLS001_HEY', cls=Host, ip='10.4.6.1', mac='00:00:00:00:00:33')
sip002_hey = net.addHost('SIP002_HEY', cls=Host, ip='10.4.2.2', mac='00:00:00:00:00:34')
sts002_hey = net.addHost('STS002_HEY', cls=Host, ip='10.4.7.2', mac='00:00:00:00:00:35')
sls002_hey = net.addHost('SLS002_HEY', cls=Host, ip='10.4.6.2', mac='00:00:00:00:00:36')
sls003_hey = net.addHost('SLS003_HEY', cls=Host, ip='10.4.6.3', mac='00:00:00:00:00:37')
sls004_hey = net.addHost('SLS004_HEY', cls=Host, ip='10.4.6.4', mac='00:00:00:00:00:38')

sls001_sco = net.addHost('SLS001_SCO', cls=Host, ip='10.5.6.1', mac='00:00:00:00:00:41')
ses001_sco = net.addHost('SES001_SCO', cls=Host, ip='10.5.8.1', mac='00:00:00:00:00:42')
sls002_sco = net.addHost('SLS002_SCO', cls=Host, ip='10.5.6.2', mac='00:00:00:00:00:43')
sls003_sco = net.addHost('SLS003_SCO', cls=Host, ip='10.5.6.3', mac='00:00:00:00:00:44')
sts001_sco = net.addHost('STS001_SCO', cls=Host, ip='10.5.7.1', mac='00:00:00:00:00:45')
sts002_sco = net.addHost('STS002_SCO', cls=Host, ip='10.5.7.2', mac='00:00:00:00:00:46')
sls004_sco = net.addHost('SLS004_SCO', cls=Host, ip='10.5.6.4', mac='00:00:00:00:00:47')
ses002_sco = net.addHost('SES002_SCO', cls=Host, ip='10.5.8.2', mac='00:00:00:00:00:48')

ctv001_hwf = net.addHost('CTV001_HWF', cls=Host, ip='10.6.9.1', mac='00:00:00:00:00:51')
ctv002_hwf = net.addHost('CTV002_HWF', cls=Host, ip='10.6.9.2', mac='00:00:00:00:00:52')

ses001_lss = net.addHost('SES001_LSS', cls=Host, ip='10.7.8.1', mac='00:00:00:00:00:61')
ses002_lss = net.addHost('SES002_LSS', cls=Host, ip='10.7.8.2', mac='00:00:00:00:00:62')

ctv001_hpp = net.addHost('CTV001_HPP', cls=Host, ip='10.8.9.1', mac='00:00:00:00:00:71')
ctv002_hpp = net.addHost('CTV002_HPP', cls=Host, ip='10.8.9.2', mac='00:00:00:00:00:72')
scd001_hpp = net.addHost('SCD001_HPP', cls=Host, ip='10.8.10.1', mac='00:00:00:00:00:73')

info( '*** Add links\n')
net.addLink(dtc1s, lan2s)
net.addLink(dtc1s, mor3s)
net.addLink(dtc1s, hey1s)
net.addLink(dtc1s, sco1s)
net.addLink(dtc1s, hwf1s)
net.addLink(dtc1s, lss1s)
net.addLink(dtc1s, hpp1s)
net.addLink(ser001_dtc, dtc1s)
net.addLink(ser002_dtc, dtc1s)

net.addLink(lan1s, lan2s)
# net.addLink(lan1s, lan3s)
net.addLink(lan2s, lan3s)
net.addLink(sap001_lan, lan1s)
net.addLink(sip001_lan, lan1s)
net.addLink(stl001_lan, lan1s)
net.addLink(stl002_lan, lan1s)
net.addLink(sip002_lan, lan2s)
net.addLink(sap002_lan, lan2s)
net.addLink(sap003_lan, lan3s)
net.addLink(stl003_lan, lan3s)

# net.addLink(mor1s, mor2s)
net.addLink(mor1s, mor3s)
net.addLink(mor2s, mor3s)
net.addLink(stl001_mor, mor1s)
net.addLink(sps001_mor, mor1s)
net.addLink(sps004_mor, mor1s)
net.addLink(evs001_mor, mor1s)
net.addLink(stl002_mor, mor2s)
net.addLink(sps002_mor, mor2s)
net.addLink(evs002_mor, mor3s)
net.addLink(sps003_mor, mor3s)

net.addLink(hey1s, hey2s)
net.addLink(hey1s, hey3s)
# net.addLink(hey2s, hey3s)
net.addLink(sip001_hey, hey1s)
net.addLink(sts001_hey, hey1s)
net.addLink(sls001_hey, hey1s)
net.addLink(sip002_hey, hey2s)
net.addLink(sts002_hey, hey2s)
net.addLink(sls002_hey, hey2s)
net.addLink(sls003_hey, hey3s)
net.addLink(sls004_hey, hey3s)
```

```
    net.addLink(sco1s, sco2s)
    net.addLink(sco1s, sco3s)
    # net.addLink(sco2s, sco3s)
    net.addLink(sls001_sco, sco1s)
    net.addLink(ses001_sco, sco1s)
    net.addLink(sls002_sco, sco2s)
    net.addLink(sls003_sco, sco2s)
    net.addLink(sts001_sco, sco2s)
    net.addLink(sts002_sco, sco3s)
    net.addLink(sls004_sco, sco3s)
    net.addLink(ses002_sco, sco3s)

    net.addLink(ctv001_hwf, hwf1s)
    net.addLink(ctv002_hwf, hwf1s)

    net.addLink(ses001_lss, lss1s)
    net.addLink(ses002_lss, lss1s)

    net.addLink(ctv001_hpp, hpp1s)
    net.addLink(ctv002_hpp, hpp1s)
    net.addLink(scd001_hpp, lss1s) # Virtual tunnel link

    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info( '*** Starting switches\n')
    net.get('DTC1S').start([cont])
    net.get('LAN1S').start([cont])
    net.get('LAN2S').start([cont])
    net.get('LAN3S').start([cont])
    net.get('MOR1S').start([cont])
    net.get('MOR2S').start([cont])
    net.get('MOR3S').start([cont])
    net.get('HEY1S').start([cont])
    net.get('HEY2S').start([cont])
    net.get('HEY3S').start([cont])
    net.get('SCO1S').start([cont])
    net.get('SCO2S').start([cont])
    net.get('SCO3S').start([cont])
    net.get('HWF1S').start([cont])
    net.get('LSS1S').start([cont])
    net.get('HPP1S').start([cont])

    info( '*** Post configure switches and hosts\n')

    ctv001_hwf.cmdPrint('ifconfig CTV001_HWF-eth0 inet6 add fc00::1/64')
    ctv002_hwf.cmdPrint('ifconfig CTV002_HWF-eth0 inet6 add fc00::2/64')

    ctv001_hpp.cmdPrint('ifconfig CTV001_HPP-eth0 inet6 add fc00::11/64')
    ctv002_hpp.cmdPrint('ifconfig CTV002_HPP-eth0 inet6 add fc00::12/64')

    ser001_dtc.cmdPrint('ifconfig SER001_DTC-eth0 inet6 add fc00::99/64')

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

Attachment B - SCPS Workflow

This attachment explains the main workflow of the SCPS.

1. Initialization of a new thread for statistics polling:

```
#File: scps_main_module.py, Class: MainSCPSModule
def __init__(self, *args, **kwargs):
    #... omitted ...
    fd_poller = FDPoller() #Thread for periodic polling of information from switches
    poller_thread = Thread(target=fd_poller.run, args=(self.SWITCH_POLL_TIMER,
                                                  self.datapath_dict))
    poller_thread.start()
```

2. Periodical sending of queries (flow statistics requests) to switches:

```
#File: fd_poller.py, Class: FDPoller
#The main function of the thread - periodically calls request in the defined interval
def run(self, poll_time, datapath_dict):
    while True:
        for dpid, datapath in datapath_dict.iteritems():
            self.send_flow_stats_request(datapath)
            time.sleep(poll_time)

#Function sends a FlowStatsRequest message for the content of flow table
def send_flow_stats_request(self, datapath):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser
    cookie = cookie_mask = 0
    match = ofp_parser.OFPMatch() #Empty match = request for the entire flow table
    req = ofp_parser.OFPFlowStatsRequest(datapath, 0, ofp.OFPTT_ALL, ofp.OFPP_ANY,
                                         ofp.OFPG_ANY, cookie, cookie_mask, match)
    datapath.send_msg(req)
```

3. The function for processing received statistics replies:

```
# File: scps_main_module.py, Class: MainSCPSModule
#Handler declaration for OpenFlow StatsResponse messages
@set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
def _flow_stats_reply_handler(self, ev):
    datapath = ev.msg.datapath
    flows = []

    for stat in ev.msg.body:
        flowdict = {} #Preparation of a new entry into flow dictionary
        flowdict['table_id'] = stat.table_id
        flowdict['priority'] = stat.priority
        flowdict['duration_sec'] = stat.duration_sec
        flowdict['idle_timeout'] = stat.idle_timeout
        flowdict['packet_count'] = stat.packet_count
        flowdict['match'] = str(stat.match)
        flowdict['instructions'] = str(stat.instructions)
        flowdict['dos_limit'] = self.DoS_PROTECTION_PPS_HIGH
        flowdict['bf_limit'] = self.BF_PROTECTION_PP10S

        #Custom format of a match (parsed from the string):
        flowdict['matchdict'] = self._create_match_dict(str(stat.match))

        #Check if the flow already exists
        previous_flowdict = self._flow_exists(datapath.id, flowdict)
        if previous_flowdict != 0:
            #Flow already exists, so the statistics will be saved
            flowdict = self._add_packet_count_history(previous_flowdict, flowdict)
            flows.append(flowdict)

        self.flow_tables_dict[datapath.id] = flows #Save the new flow

        #Protection modules functions:
        self._dos_protection(ev.msg)
        self._brute_force_protection(ev.msg)
```

4. Initialization of the REST API class and definition of the function, which returns the *flow table* for the switch specified in the *dpid* argument:

```
#File: scps_main_module.py, Class: SCPRestAPI
class SCPRestAPI(ControllerBase):
    def __init__(self, req, link, data, **config):
        super(SCPRestAPI, self).__init__(req, link, data, **config)
        self.sc_app = data[SC_REST_INSTANCE]

    #Function listens on URL: /sc/flows/{dpid} for HTML requests
    @route('sc', URL_FLOWS, methods=['GET'], requirements={})
    def list_flows(self, req, **kwargs):
        dpid = kwargs['dpid'] #Switch DPID from the HTML request argument
        flows = self.sc_app.get_flows(dpid)
        if flows == 0:
            return Response(status=404)
        body = json.dumps(flows) #Data is encoded into JSON format
        return Response(content_type='application/json', body=body)
```

5. Function of the web application, which is executed, when the URL: "*/dos_monitoring*" is accessed. The function sends a URL request to the REST API of the main module ("*/sc/flows*" with the *dpid* of the first switch). A successful result will render the *dos_protection.html* web page.

```
#File: scps_web_app.py, Web application (no class)
URL_FLOWS = "/sc/flows/"
#... omitted ...

@app.route('/dos_monitoring')
def dos_monitoring():
    data = get_data_from_connection_GET(URL_FLOWS + '0000000000000001')
    if data == -1:
        return render_template('page-error-500.html')
    else:
        return render_template('dos_protection.html', data=data,
                                nodes=getConnectedNodes(data), topo=sc_lan_topo)
```

6. Examples of returned data usage in a *html* file.

```
#File: dos_protection.html, Web application (no class)
<td>
    {% if flowrule['priority'] == 8 %}
        <div class="color-danger"> DoS attack detected! Blocking the traffic! </div>
    {% elif flowrule['packet_count_history']|length == 10 %}
        {% if  ((flowrule['packet_count_history'][9] - flowrule['packet_count_history'][8]) /
                            flowrule['dos_limit'] * 100) > 75 %}
        <div class="color-warning">Unusual traffic detected! Utilization over 75%. </div>
        {% else %}
            The device is working correctly.
        {% endif %}
    {% endif %}
</td>

... omitted ...

<td><span>
    {{ topo[flowrule['matchdict']['eth_src']][1] }}
</span></td>
```