POLITEHNICA UNIVERSITY OF BUCHAREST

Doctoral School of Electronics, Telecommunications and Information Technology

Decizion No. 973 from 08-12-2022

Ph.D. THESIS SUMMARY

Ing. Teodor-Pantelimon Tivig

CONTROL PLAN OPTIMIZATION IN SOFTWARE DEFINED NETWORKS

THESIS COMMITTEE

Prof. Dr. Ing. Bogdan IONESCU Univ. Politehnica din Bucureşti	President
Prof. Dr. Ing. Eugen BORCOCI Univ. Politehnica din Bucureşti	PhD Supervisor
Prof. Dr. Ing. Virgil DOBROTĂ Univ. Politehnica din Cluj-Napoca	Referee
Prof. Dr. Ing. Sorin ZOICAN Univ. Politehnica din Bucureşti	Referee
Dr. Ing. Cristian-Iulian RÎNCU Academia Tehnică Militară din Bucureşti	Referee

BUCHAREST 2023

Contents

Chapter 1

Introduction

In recent years, in the context of the development of large telecommunications networks, the concept of Software Defined Networks (SDN) has been introduced, which brings to the fore the networks formed by the control plane (SDN controller) and data traffic transmission nodes.

„SDN is seen as a new paradigm, which attempts to build an elastic, adaptable and programmable network model by decoupling the control plane from the data plane." This model, with the help of SDN programming, enables innovation in an agile way that has not been known before. Last but not least, the network paradigm, which promises to remove the specifics of a network equipment manufacturer, but also the specific requirements of a network administrator, by transforming network switching devices into simple packet forwarding devices that can be programmed through -an interface, placing at the same time the control plane of each network device in a logical centralized module [1].

SDN is based on logically centralized network control. This centralization, due to SDN, brings some challenges related to distributed control in traditional TCP/IP networks.

In computer networking technology, one of the strongest movements is the trend towards creating user-developed applications, as users are no longer viewed as passive consumers, but can be a vector of application development. Users can be represented by specific company departments or even companies in a shared environment and want to customize their applications and services. In this context, applications refer to network services and functions: traffic balancing to optimize network resources, routing, and even security. To be able to implement these applications, networks must change, become flexible, adaptable to user requirements, by applying requirements as services. This movement has induced the conceptualization and adoption of new network models such as Software Defined Networks [2], [3] and Network Function Virtualization [4], [5].

“The concept of Network Function Virtualization (NFV) is both a concept and a powerful technology that is constantly evolving. At the concept level, NFV proposes to disaggregate software from the hardware it runs on, by realizing in software as many network functions as possible, which have traditionally been built with hardware and software specifically designed to work in tandem.

The current evolution and trend is focused on a more intensive use and in a close connection of SDN and NFV technologies.

A solution for the efficiency of large networks is represented by the development of multicontroller networks, with a well-defined geography. The advantages of the multi-controller network are from the point of view of traffic, both through the possibility of dividing it between controllers, and through the taking over of the traffic of a controller by the other controllers when needed, the latter representing the redundancy of controllers in SDN networks. One of the major disadvantages of these networks is given by the latency involved in the communication between controllers and nodes, when inter-controller communication is carried out. Latency also occurs in inter-controller communication. The intensively studied solution to minimize the disadvantages of SDN networks is given by the optimal placement of controllers with an overview of the network.

1.1 Presentation of the field of the doctoral thesis

The main area of this thesis is the focus on SDN technology and the study as well as personal contributions to a problem that is still open: the synchronization of the state of the network with several controllers in case of failure of the master controller.

Based on the development software framework of the Ryu controller and the Mininet network emulator, the scope of the thesis mainly focuses on control plane optimization, in SDN software-defined networks. Based on the text, which identifies the challenges, the thesis creates experiments and personal contributions for concepts, software configurations, and integrates distributed databases, for network state consistency.In addition to those presented, 5G networks, NFV and software tools are also introduced in the study.

1.2 The purpose of the doctoral thesis

The starting point of this thesis was my professional experience corroborated with the interest in analyzed trends that loomed on the horizon in SDN, virtualization, NFV and cloud computing.

Moreover, the scientific research opportunity to contribute to a technology, which introduces automation in the field of networking, the field in which I work professionally, to solve one of the current challenges, which is to manually configure the multitude of existing equipment in a provider network of services and eliminating certain tasks, which are performed manually, both subject to errors from the human factor gave me the chance to know, work and contribute to the optimization of the control plane in software-defined networks through SDN.

.

1.3 The content of the doctoral thesis

This thesis is organized as follows:

Chapter 2 "Current State of the Field and Open Issues"presents the current status of classical IP networks, the challenges and limitations of this decentralized architecture, as well as the field of software-defined networks, but also its own contribution to the identification of open problems in SDN.

Chapter 3 "Quality of Service in Software Defined Networks"presents the specific QoS characteristics of software-defined networks and various researches that have dealt with the subject of QoS in SDN architecture, but also experiments involving the parameters used to evaluate the quality of service**.**

Chapter 4 "Application Implementations for SDN Controllers"focuses on the presentation of the components of the Ryu framework and the detailed explanation of the specific events, with the help of which it is possible to create management and control applications, but also your own contribution to the development of a traffic routing application**.**

Chapter 5 "Control plane optimization in SDN technology"focuses on the presentation and testing of an own mechanism, which creates a scalable and fault-tolerant control plan, being a possible solution to the problem of the presence of a single controller in the software-defined network**.**

Chapter 6 "Conclusions"- summarizes the main lessons learned along the way doctoral activity.

Chapter 2
Current state of the SDN field and open issues

This chapter presents the challenges encountered in classic IP networks. Also this chapter is dedicated to understanding the architecture of Software Defined Networks (SDN) together with a critical analysis of the placement of multi-controllers in SDN technology used in networks dispersed over large geographical areas. This analysis was used as a starting point for the architecture proposed in Chapter 5.

This critical approach is gonefrom our observation that there are relatively few requests that deal with the problem of SDN controller placement (CPP) with an emphasis on a single performance criterion but also the absence of mechanisms for open source SDN controllers to re-elect a new master controller in the scenario when the master controller current fails.

2.1 Problems encountered with classic IP networks

Traditional networks use a large amount of human resources (network operators) and equipment (routers, switches and intermediate equipment). This is why traditional networks present these challenges, while network operators have to configure the protocols, use high-level policies to configure; by using configuration commands it responds to different types of network events, constantly adapting to changing conditions [6].

Traditional networks are vertically integrated, which means that the control plane (the decision maker of network traffic) and the data plane (the sender of network traffic as decided by the control plane) are tightly integrated within the network. As seen in Figure 2.1 [7], networks consist of tree planes: data plane, control plane and management planes. The data plane corresponds to the network devices used to transmit packets, the control plane corresponds to the protocols that elaborate the forwarding tables, the management plane includes both software services [8], as well as network policies. Software services are used to oversee the control plan, and network policies defined in the management plan are used to transmit data according to the control plan [9].

The last two planes mentioned are located on the network devices. Their incorporation on the same devices and the close connection between them imposes limited flexibility on traditional IP networks, as these two planes are vertically integrated, resulting in a decentralized architecture. These features were important to the design of the Internet; also important in building networks was their resilience and performance, for line rate and port density [10].

Traditional networks can often contain misconfigurations and associated errors (more than 1000 BGP routers) [11], a single misconfigured device producing packet loss, forwarding loops, unintended forwarding paths, and contract violations [7].

Traditional IP networks are highly decentralized, with tightly coupled control and data planes, features that were important to the design of the Internet. Also important in building the network was network resilience and performance to rapidly increase line rate and port densities [7].

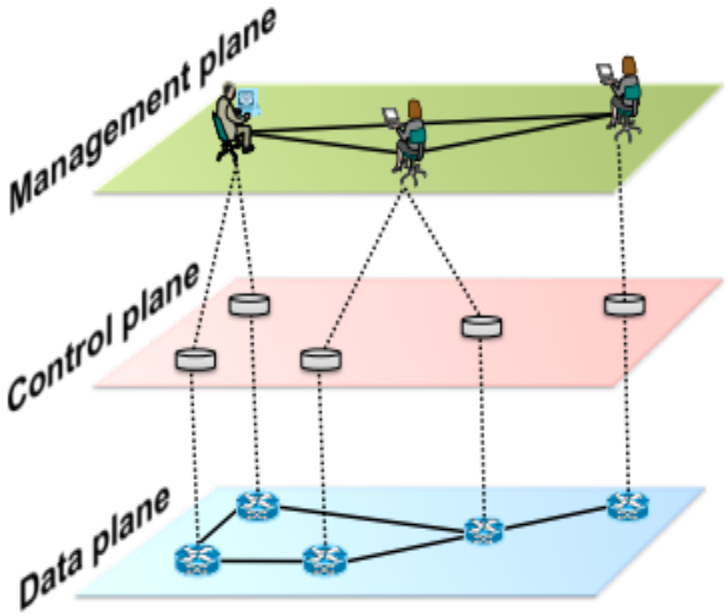

Figura 2. 1. *Layered view of network functionality in planes* [7]*, [Copyright © 2015, IEEE]*

2.2 Software-defined networks

The term SDN represents a recent paradigm in computer networking. It became better known with the appearance of the OpenFlow protocol in 2008 [10] [12]. The

separation of control and data planes of network equipment has been around since the 1980s [10], [13]. Early attempts at programmed networks have been around since the 1997s [10], [14].

Software-defined networking (SDN) features an architecture where the data plane and the control plane are separated or decoupled. The SDN architecture comprises three main elements or layers, which are: the application layer, the control layer and the forwarding layer, as shown in Figure [7].

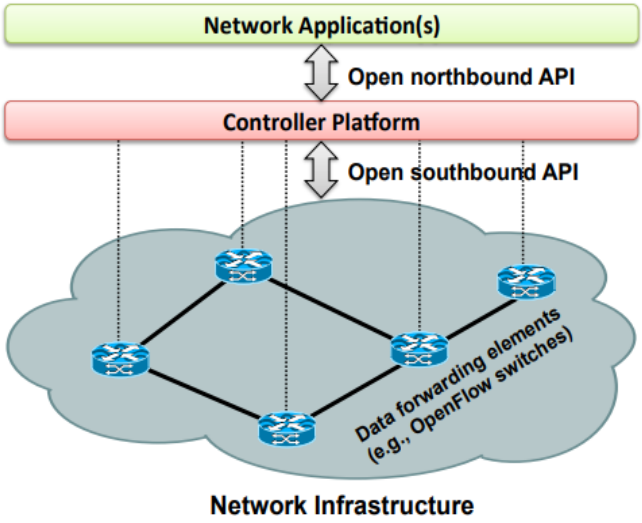

Figura 2. 2. *Vedere simplificată asupra arhitecturii definite software* [7]*, [Copyright © 2015, IEEE]*

A software-defined network architecture has four attributes [7]:

- Control and data planes are decoupled. Network devices represent only the switching elements, without further incorporating control decisions.
- Redirection decisions are based on flows. A stream is a set of packet value fields, composed of a match criterion and a set of instructions. In software-defined networking, a flow is defined as a set of packets between a source and a destination with the same service policies [15], [16].
- Logical control is referred to as SDN controller or Network Operating System (NOS). An NOS is software that runs on a server and schedules the switching devices based on centralized logic and an abstract view of the network.
- Network programmability is a primary feature of SDN, provided by management plane software applications.

2.2.1. Ryu controller

The chosen controller for this thesis is Ryu. In the SDN architecture, Ryu is described as component-based and open source software. The Python programming language is

used to implement Ryu, providing its software tools, which contain explicit APIs, for free.

The Ryu controller architecture is shown in Figure 2.6 [17].

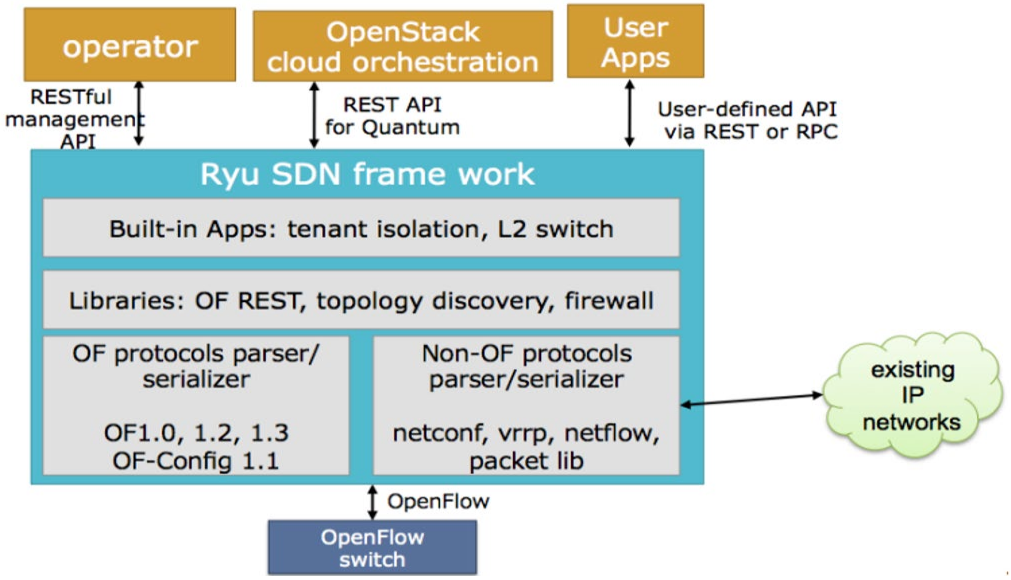

Figura 2. 3. *Ryu controller architecture [17]*

2.3 A critical analysis of the multi-controller placement problem in wide area SDN networks

In this section we compare the complexity of different CPP approaches, through critical analysis, using two broad criteria, i.e. static and elastic controller-to-switch mapping, aiming to highlight potential research gaps. SDN specific evaluation criteria will be: capacity, scalability and load balance. Many CPP research studies have considered a static mapping between switches and controllers. However, in real networks, the traffic load is dynamic with high bandwidth requirements; therefore, some studies have recently concluded that static mapping is not the best approach to meet the Service Level Agreements (SLAs) that govern many network services.

2.3.1 Limitation of a single control solution

„Early SDN architectures used a single controller to control the entire infrastructure. Meanwhile, the researchers concluded that a single controller cannot handle the high data throughput demand from the switches due to the limited capacity of the controller [18]. Also, a single controller is a single point of failure; the consequence is that without a controller, forwarding devices cannot learn how to treat newly arrived

packets. To alleviate these problems, a modern architectural design using a multi-controller scheme was proposed. In the new scheme [7] there are most concepts of master, slave and peer controller; any switch can connect to a single master and multiple peer or slave controllers. A peer controller is one that has full access to the switches and is a peer controller with another controller having the same role."

2.3.2 Analysis and discussion of dynamic and static SDN controller placement aspects

Controller placement has two strategic aspects: dynamic and static. In general, many works propose methods that have a known number of controllers and then optimally solve the CPP.

Especially in static and medium-sized networks, the optimal location of the controller is made accordingly, taking into account two parameters: the type of topology and the capacity of the controller, [19]. In this section, we try to perform a critical analysis of the CPP problem when the network conditions change.

2.3.2.1 Static approach to SDN controller placement

„Many studies have tried to optimize the flow processing time on the controller's network interfaces, thereby improving the response time of the controllers, while other researchers [20] they also used the message buffering model, but imposed that the number of streams processed by the controller would not exceed the controller's capacity.

Conversely, other studies have found that partitioning the network into subdomains will increase the number of controllers and halve the flow configuration. It is our view that any approach to improving setup time is valuable for CPP, which we will discuss in the rest of the section".

2.3.2 Critical analysis of open issues regarding CPP

„To date, the issue of migration selection for a switch or controller has not been fully addressed in any of the cited papers; there is still no clear approach to choosing a target switch or controller. If there are multiple underutilized controllers, there is no clear mechanism that should be selected. There are a few studies that consider the cost effectiveness of migration for a switch cluster. This lack of information should be expanded upon in future research. The development of a traffic behavior prediction method in order to estimate the future state in which the traffic is unpredictable is still

an open research problem. This is important for making the switch migration decision."

Chapter 3

Quality of Service in Software Defined Networks

With the rapid development of SDN, academic and industry researchers have been trying to achieve QoS provisioning, for more network applications nowadays. Various studies, which have addressed the subject of QOS, have described one or more of the categories, in which QoS can be placed as a benefactor of SDN architectures. These categoriescan be transformed into a simple scheme (Figure 3.1 [21]), each of the 7 elements, reflects a challenge for QoS in SDN that has been studied in various researches, to bring a solution to one of the 7 particular problems, in providing QoS:

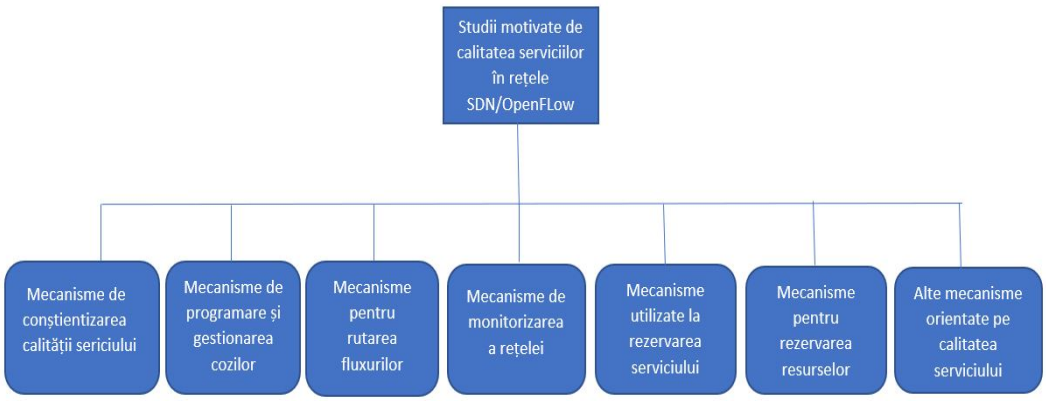

Figura 3. 1. Organization of studies based on quality of service in SDN/OpenFlow networks, modified by [21]

3.1 The relationship between SDN and QoS

The key purpose of QoS is to ensure traffic classification, which results from the use of QoS parameters [21]:

- bandwidth;
- delay;
- jitter;
- packet loss.

The relationship between SDN mechanisms and QoS delivery is listed below [21]:

- separation of the data plane and the control plane for networks, which improves the network controller, from a network control perspective;
- network applications are not forced to deal with low-level configurations of devices in the data plane and are provided with an abstract view of the network by controllers;
- controllers can get the global view of the network and its states
- SLA-urile and control policies can be specified by an administrator at a higher level of abstraction without the need for configuration on each forwarding device.
- the policy set and also the different flow classes are unrestricted allowing fine-tuning based on user needs;
- therefore, rules can be defined per flow (if needed) and the controller is tasked with applying them, appropriately to different network elements.

Table 3.1 [21] presents the advantages of SDN in correlation with the 7 categories of QoS provisioning.

Table 3.1. The main features of SDN, which are used in the surveyed papers and their relationship with the 7 categories of QoS provisioning, modified after [21]

Organizarea Caracteristicilor SDN	Arhitectură	Rutare	Management			Modelare	
	Mecanism de rutare a calității serviciului inter-domeniu	Mecanisme de rutarea a fluxurilor multimedia	Mecanisme de rezervarea resurselor	Mecanisme de monitorizarea retelei	Mecanisme pentru programarea si gestionarea cozilor	Mecanisme de constinetizarea a calității serviciului	Alte mecanisme de constientizare a calității serviciului
Actualizare dinamică a regulilor de flux	•	•	•		•	•	•
Monitorizarea traficului			•	•	•	•	•
Configurarea cozii		•			•	•	•
Redirectionare bazată pe flux	•	•				•	
Analiza traseului fluxului	•		•	•			•
Analiza pachetului/ fluxului	•	•	•	•	•	•	•

3.2 Performance evaluation experiments for video and VoIP traffic with RYU controller and Mininet framework

This section presents personal SDN experiments, the purpose of which is to create different scenarios to perform performance tests, at the node level. The tests use the following evaluation criteria: bandwidth, delay and bandwidth for data traffic and VoIP. The experiments use the Iperf software tool, the Mininet network emulator, and the Ryu SDN controller.

Traffic statistics are useful for obtaining information about network congestion in order to later send it to the controller to decide how to solve itthe bandwidth requirement for concurrent streams from subscribers connected to the network.

.

Chapter 4

Application Implementations for SDN Controllers

This section continues the work of Chapter 3 with a section on prototyping a control application for the Ryu controller. The current section implements, analyzes, explains, and tests the logic of a switch that performs packet forwarding using OSI stack layer 3 criteria.

These software components feature APIs that make it possible to create control applications in a customized manner, instructing the Ryu controller how the user wishes to use the equipment.

The applications for the Ryu controller are Python scripts, and the controller

will configure the network equipment using the Open Flow protocol or another protocol on the south interface.

4.1. Prototyping the Packet Forwarding Application

The proposed application will implement functionalities such as transferring packets from one network interface to another interface to the destination address, storing them in the buffer memory of the equipment and performing other actions specific to the packets. The code is based for the proposed application is in the application simple_switch.py [18], which we will build to create the functionalities of the proposed application.

„At this stage, the application will initialize the data that will be used throughout the entire infrastructure. To implement the key feature of equipment intelligence, the ryu.base.app_ .manager will be used, which inherits the features of the OSI Layer 2 switch".

```
clasa Layer3forwarding(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_5.OFP_VERSION]
    def __init__(self, *args, **kwargs):
            super(Layer3forwarding).__init__(*args,**kwargs)

                    self.mac_to_port = {}
```

4.1.2 Event Manager for newly registered switches

The event manager specifies the actions to be taken for a particular event, such as receiving a packet. The next two lines instantiate the actions to be executed. The set_ev_cls decorator is used to specify the event class. Every time the switch generates an event, which reaches the controller, it must respond with an appropriate message. The structure *ofp_event.EventOFPSwitch Features + Message_name* represents the event class name argument or dispatcher for the event manager.

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures,CONFIG_DISPATCHER)
        defswitch_features_handler(self, ev):
```

Information received by the switch shall be decoded using the following code data structure:

```
        datapath = ev.msg.datapath
```

```
ofproto = calea datelor.ofproto
parser = datapath.ofproto_parser
potrivire = parser.OFPMatch()
actiuni=[parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,dep
roto.OFPCML_ NO_BUFFER)]
self.add_flow(datapath, 0, match, actions)
```

4.1.3 Adding flow entries to the switch

The *add_flow* function adds a match condition, instructions, and also the effective time and input priority level for the target packet to each new flow entry. This function is invoked each time in response to a specific message.

```
def add_flow(self, datapath, priority, match, actions, buffer_id=None):
ofproto = calea datelor.ofproto
parser =datapath.ofproto_parser
inst=[parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
```

4.1.4 Construction of the flow change message

Flow Mods (FlowMods) messages instruct datapaths to add, delete, or modify a flow in its flow table.

```
if buffer_id:
   mod=parser.OFPFlowMod(datapath=datapath,buffer_id=buffer_id,
                         prioritate=prioritate,potrivire=potrivire,
                         instrucțiuni=inst)
                else:
                         mod=parser.OFPFlowMod(datapath
                         =datapath,priority=priority,match
                         =potrivire, instrucțiuni=inst)
                         datapath.send_msg(mod)
```

4.1.5 Parsing the packet and verifying the integrity of the message

The Ryu controller has its own built-in packet handling library that contains the rules for handling Packet-In notification messages.

```
    @set_ev_cls(ofp_event.EventOFPPacketIn,MAIN_EXPEDITOR)
        def _packet_in_handler(self, ev):
    if ev.msg.msg_len < ev.msg.total_len:
                        self.logger.debug("packet truncated:only %s of %s bytes",
v.msg.msg _len)
    msg = ev.msg
    datapath = msg.datapat
    ofproto = calea datelor.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']
    msg = ev.msg
    datapath = msg.datapat
    ofproto = calea datelor.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']
    pkt =packet.Packet(msg.data)
        eth =pkt.get_protocols(ethernet.ethernet)[0]
    if eth.ethertype == ether_types.ETH_TYPE_
                        LLDP:
                        return
    dst = eth.dst
    src = eth.src
    dpid = datapath.id
    self.mac_to_port.setdefault(dpid, {})
```

Metadatele necesare pentru pachet sunt înregistrate în scopul depanării.

```
    self.mac_to_port[dpid][src] = in_port
    if dst în self.mac_to_port[dpid]:
                out_port = self.mac_to_port[dpid][dst
        else:
                out_port = ofproto.OFPP_FLOOD
    actions = [parser.OFPActionOutput(out_port)]”
    if out_port != ofproto.OFPP_FLOOD:
                dacă L3.ethertype==ether_types.ETH_TYPE_IP:ip
                        =pkt.
                        obține _ protocol(ip4.ipv4)
                        srsip=ip.src
                        dstip=ip.dst
                        potrivire=parser.OFMatch(eth_type=ether_
```

Chapter 5

Control Plane Optimization in SDN Technology

This chapter presents the main scientific contributions to the creation of a scalable fault-tolerant control plan that represents a potential solution to the problem of a single controller representing a single point of failure in software-defined networks. The mechanism to create a scalable control plane uses Redis and Zookeeper. Software simulations are done with the Ryu controller and the Mininet emulator.

The general objectives of this chapter are:

- Introducing the SDN multi-controller placement problem and its intermediate problems:
    - the number of SDN controllers for optimal network operation.
    - and where to place the controllers on the network
- Proposing a scientific scenario as well as a practical solution to ensure a scalable control plane in SDN technology.
- The functional components and tools used are as follows:

- Python, Ryu, Mininet, Zookeeper and Redis.

5.1 SDN multicontroller placement problem

To solve the problem of placing multiple controllers in SDN technology, several metrics need to be improved, such as the latency between the controller and the forwarding elements and between the controllers, but also the resilience and reliability. Also, considering the latency for real network architectures, the optimal choice in fail-safe scenarios is to evaluate the global solution space with offline computations.

The control plane architecture of software-defined networks has been designed with a single controller or with multiple controllers, these being classified in turn into either centralized physical or logical architectures or distributed physical or logical architectures as shown in Figure 5.1.

In a multi-controller architecture, the controllers work simultaneously to achieve reasonable performance and scalability. A multi-controller SDN architecture can take many forms and aspects, with network segments divided into centralized or distributed logical or physical architectures and flat and hierarchical segment architectures.

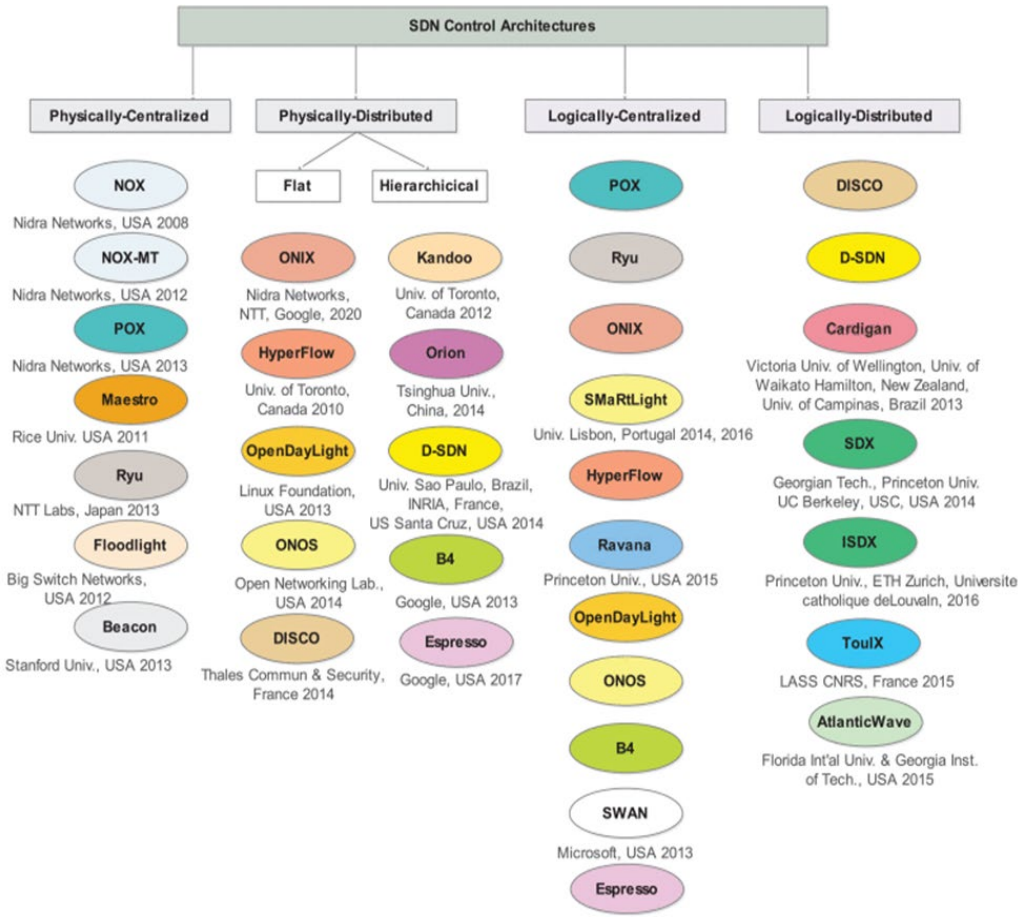

Figura 5. 1. *Classification of SDN control plane architectures SDN [22], [Copyright © 2018, IEEE]*

5.2 The problem of placing multiple controllers in de area SDN networks

This section is intended to introduce a number of constraints as well as objectives for deploying multicontrollers in wide area SDN networks as well as challenges that may arise in implementing them to achieve the objectives. Also, in large-area SDN networks, it is necessary to use algorithms to solve the multi-objective large-area SDN controller placement problem [23].

Constraints used when placing controllers include:

- controller capabilities;
- switch loads.

Applicable goals for large-scale SDN networks [23],

- latency between switches and controllers;
- inter-controller latency;
- load balancing;
- node, controller and link failures

5.3 Contributions to the creation of a scalable control plane in SDN technology

In this section, a mechanism for creating a control plane in SDN architecture is presented.

The main idea of this thesis is that it tries to create and present a live test of a robust mechanism that represents a potential solution to the problem of having a single point of failure in the SDN architecture.

The mechanism by which the SDN controllers managing the network have a symmetric topological basis in the scenario when the main controller fails is realized with Redis and Zookeeper. The mechanism allows multiple Ryu controllers to continue to perform their functions over the network in a manner tolerant of network events that can take the master controller out of service, through a centralized service called Redis that maintains network state information and Zookeeper is used to choosing the master controller.

5.3.1 Proposed solution and results

In this section, a fault-tolerant mechanism is presented to reduce the single point of failure in the SDN network, and consequently, the proposed solution will create a high degree of service availability and increase reliability and scalability.

From a high-level perspective this architecture consists of the following components:

- Ryu Controllers;
- Redis database for distributing topological information for the Ryu controller;
- Zookeeper used for fault detection and choice of master or slave controller;
- Mininet for constructing the topology;

- The Python language used to define the topology and the interaction of the components shown above.

The proposed solution consists of three individual Ryu controllers running in three terminals, each controller using different port numbers. So controller number 1 listens on port 6633, controller number 2 listens on port 6634 and controller number 3 listens on port 6635. The controllers will be used in each of the three roles of peer, master or slave. In the fourth terminal, a basic linear topology with two switches will run and each individual switch will be connected to the three controllers as shown in Figure 5.2 by running the command: sudo mn --controller=remote, ip=127.0.0.1:6633 - -controller=remote, ip=127.0.0.1:6634 controller=remote, ip=127.0.0.1:6635

In the controller simulation in Equal mode

The switches are connected as can be seen in Figure 5.2 and all controllers will have equal roles. This means that all controllers will receive a duplicate notification with the PACKET_IN message for each new packet that a switch receives. Having all controllers with the same role, the consequence being that the number of packets will multiply by the number of controllers present in the topology. The advantage is that control plane redundancy is achieved in case of failure of any controller and the resulting packet loss is zero. The disadvantage of this configuration is limited scalability, every controller receives the same PACKET_IN message, the same payload. The graph in Figure 5.3 shows the number of transmitted packets, received packets, and duplicate packets when the controllers are running in the equal role."

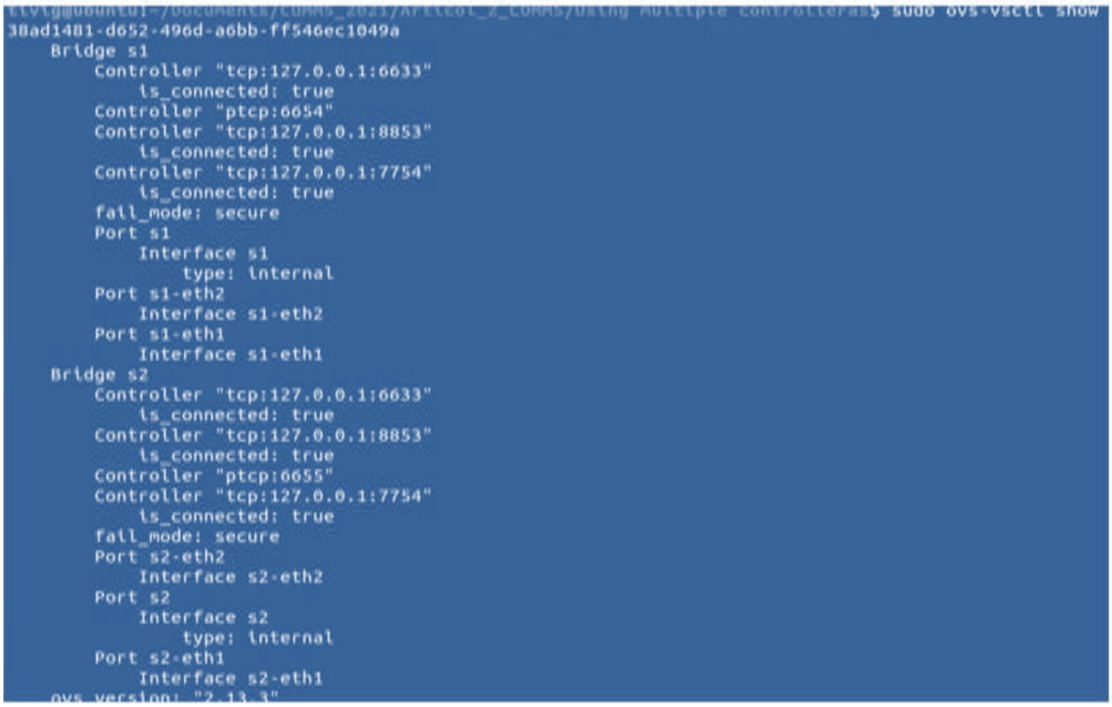

Figura 5. 2. *Evaluating the type of connectivity of the switches with all three controllers [24]*

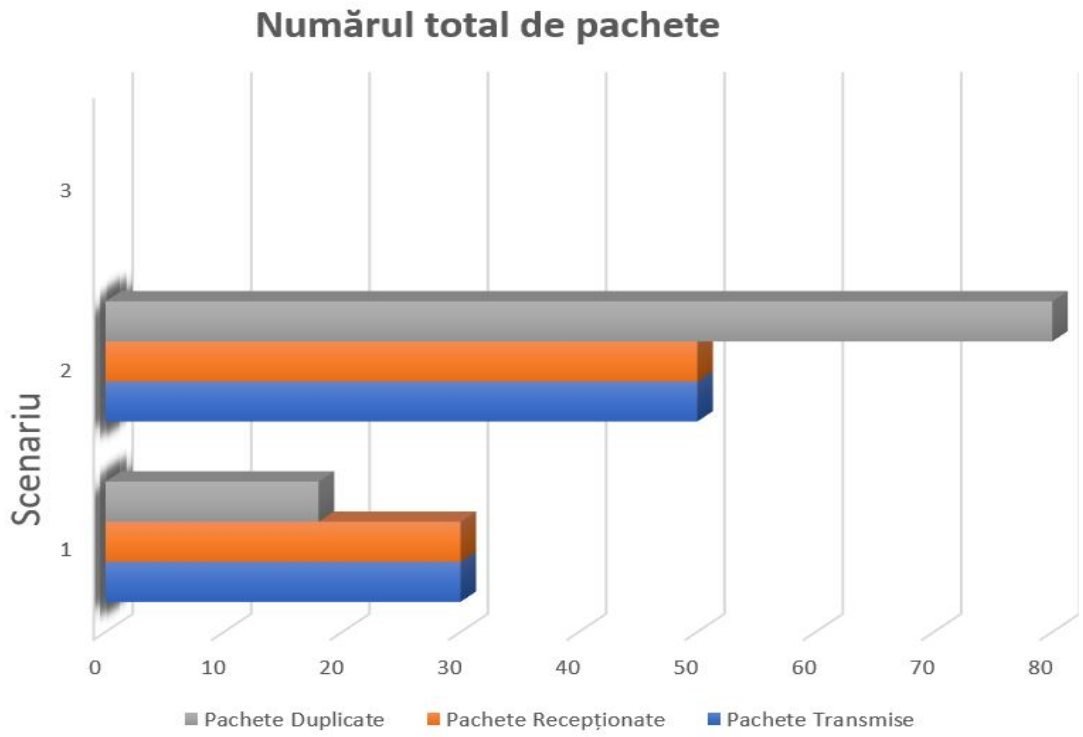

Figura 5. 3. Statistics for transmitted, received and duplicate packets [24]

Master and Slave role simulation

The master and slave configuration scenario will have the first controller running as the master and the remaining two controllers running as slaves. The two controllers working in the "SLAVE" role do not receive the PACET_IN message, only the number one controller which is the master, receives the PACKET_IN notification from all switches. After the functionality of the control plane represented by the first controller is verified, the test continues by stopping the main controller using the CTRL+C command in its terminal, which triggers the re-election of the master and slave controller. Any remaining controller can be chosen as the master.

The second controller becomes the master (according to a random criterion, which can be changed by the user), it receives all the topological information from the Redis database, receiving the PACKET_IN notification because it has the role of master, and the number three controller receives the role of SLAVE. Initiating a ping test using the pingall command from the first host in the topology evaluates the functionality of the control plane after reselection.

In the next step, the number one controller that was previously stopped is started by issuing the command: ryu-manager --ofp-tcp-listen-port 6633 master_slave.py and takes the role of slave, due to the fact that the master controller was previously chosen.

After that, we're going to shut down the master controller, controller number 2, and either controller becomes the master controller. The functionality of the controller that obtains the role of master (controller number 3) is tested, after which the controller number two that takes over the role of slave is restarted. In the last step, controller number 3 is turned off, thus triggering the re-election process of the master controller.

Thus, through successive steps of starting and stopping one of the three controllers with the master role, the controller that takes over the master role after the failure (stopping) of the current master controller, it is tested if the proposed mechanism that integrates the Redis database, provides the topological information in a optimal time to the new master-elected controller at each simulated failure of the master-role controller.

Chapter 6

Conclusions

SDN technology brings advantages but also major challenges such as: networks that have a single controller that can impact the network becoming unusable; but also at the level of networks dispersed over large geographical areas that require the presence of multi-controllers by: interconnecting these to create a fault-resistant control plan by providing the network status of the controller that takes control in case of failure of the master controller, but also very important is the location of these controls to satisfy specific parameters with a strong impact on the quality of services but also on scalability.

This thesis proposes a review of the works present in the specialized literature in order to identify guidelines regarding contributions to the optimization of the control plane both to identify an optimal solution for the placement of multi-controllers but also to achieve communication at the level of the data plane for the transfer of the

network state in scenario as the controller managing the network fails, but has not identified a complete solution.

6.1 Results achieved

Chapter 1 presents the thesis, its motivation and structure. The thesis area focuses very strongly on the control plane of SDN technology, developing personal experiments for quality of service (QoS), making applications for the Ryu controller using the Python programming language.

The software frameworks used are Ryu and Mininet.

Chapter 2 focuses on the state of the SDN field and the open issues, challenges and status of classic networks. It presents its own contributions regarding the identification of challenges and open issues belonging to SDN technology, but also the highlighting of critical aspects, which were presented at the ICCNE 2015 conference.

Chapter 3 is completely dedicated to presenting the quality of services of the SDN architecture at a high level and the experiments performed.

Chapter 4 results are based on the Ryu controller software framework and specialized software events used to create applications. There is an example of an application based on traffic routing where the code used is contribution.

Chapter 5 is entirely the author's work from theoretical and practical perspectives. The main focus is on the control plan of software-defined networks and the main challenges that can impact the scalability and availability of control but also the quality of service. Due to these challenges, an own mechanism and scenario was proposed for a scalable control plane that also avoids the problem of the presence of a single controller in the SDN topology.

Chapter 6 is devoted to conclusions and perspectives.

6.2 Summary of original contributions

Below is a brief description of the personal contributions presented in this thesis.

1. Collecting different points of view and trying to identify and highlight critical issues related to 5G technology.
   Contributions can be identified in: Chapter 2.6, Annex A.3, Publications [1-ICCNE].
2. Critical analysis of the multi-controller placement problem in wide area SDN networks.
   Contributions can be identified in: Chapter 2.7 Publications [2-COMMS].

3. Performance evaluation experiments for voice and video traffic with Ryu controller and Mininet framework.
   Contributions can be identified in: Chapter 3.8 Publications [3-Scientific Bulletin]
4. Analyzes, explains the components of the Ryu framework, as well as software-created events for use in creating management and control applications. Proposes a trick for driving traffic with explanations and steps needed to do it.
   Contributions can be identified in: Chapter 4.12 Publications [4-ISNCC].

5. Another experiment proposes a scenario for creating a scalable, fault-tolerant control plane that tries to solve the problem of having a single SDN controller for open software architecture controllers.
   Contributions can be identified in: Chapter 5.3 Publications [4-ISNCC].

6.3. Personal publications

6.3.1 List of Articles Published/Accepted for publication

1. "Pantelimon-Teodor Tivig, Silviu-Andrei Lazar,5G Mobile Technology between Requirements and Real Life Deployment, The 2nd International Conference on Communication and Network Engineering, 2015".
   Published in: International Journal of Future Computer and Communication (IJFCC).
   Abstracting/Indexingby Google Scholar, Engineering & Technology Digital Library, and Crossref, DOAJ, Electronic Journals Library, EI (INSPEC, IET).
   Conference location: Amsterdam, Netherlands.
   Conference date: December 10-11.

2. "Pantelimon-Teodor Tivig, Borcoci Eugen, Software Defined Networking Using Mininet for Agile Testing Network Scenarios, The 12th International Conference on Computer Science and Information Technology (ICCSIT 2019), 2019"
   Published in: Journal of Communications (JCM, ISSN: 1796-2021Online; 2374-4367 Print). Abstracting/Indexing: SCOPE; DBLP; ULRICH's Periodicals Directory; And so on
   Conference Location: Barcelona, Spain.
   Conference date: December 18-20.

3. Pantelimon-Teodor Tivig, Borcoci Eugen, Critical Analysis of Multi-Controller Placement Problem in Large SDN Networks, 13th International Conference on Communications (COMM), 2020.
   Published and Indexed: IEEE Xplore, Conf**Abstracting/Indexing:**ISI database.
   Conference location: Bucharest, Romania.
   Conference date: June 18-20.
   DOI: 10.1109/COMM48946.2020.9141969

4. Pantelimon-Teodor Tivig, Borcoci Eugen, Alexandru Brumaru, Layer 3 Forwarder Application - Implementation Experiments Based on Ryu SDN Controller, International Symposium on Networks, Computers and Communications (ISNCC), 2021.
   Published in: IEEE Xplore digital library effective 2021-11-25, DBLP and Thomson Reuters.
   Indexed in: IEEE BDI
   Conference location: Dubai – United Arab Emirates.
   Conference date: October 31 – November 2.
   DOI:10.1109/ISNCC52172.2021.9615685

5. Pantelimon-Teodor Tivig, Borcoci Eugen, Alexandru Brumaru, Performance evaluation experiments for video and voip traffic with Ryu controller and mininet framework-Part I, Scientific Bulletin – Universitatea Politehnică Bucharest, 2022.
   Accepted and published.
   Filing number 12260.
   Indexed in ISI and SCOPUS.

6. Pantelimon-Teodor Tivig, Borcoci Eugen, Alexandru Brumaru, Performance Evaluation Use Cases For Video And Voip Traffic Using Ryu Controller And Mininet Framework – Part II, Scientific Bulletin – Universitatea Politehnică Bucharest, 2022.
   Accepted and published.
   Filing number 12261
   Indexed in ISI and SCOPUS.

7. Pantelimon-Teodor Tivig, Alexandru Brumaru, Serban Georgica Obreja, "Creating Scalable Distributed Controlplane In SDN To Rule Out The Single Point Of failure", The 14th International Conference on Communications, 2022.
   Conference location: Bucharest, Romania.
   Conference date: June 16-18.
   Published and Indexed in: IEEE BDI Conference.
   Indexed in ISI.
   DOI:10.1109/COMM54429.2022.9817181

8. Pantelimon-Teodor Tivig, Eugen Borcoci, "Performance Assessments For SDN Control Plane Into Distinct Network Topologies", The 30th International Conference on Software, Telecommunications and Computer Networks.
   Published and Indexed in: IEEE Xplore, Scopus, and DBLP.
   Conference date: 22 - 22 September 2022 - Split, Croatia.
   DOI:10.23919/SoftCOM55329.2022.9911385

9. Pantelimon-Teodor Tivig, Eugen Borcoci, " **Dynamically Offloading The SDN Control Plane In Large Area Networks By Condition Aware Migration Of Forwarding Devices"**, 2022 25th International Symposium on Wireless Personal Multimedia Communications.
   Published and Indexed in: IEEE Xplore
   Conference date: 30 October - 2 November 2022 - Herning, Denmark.
   DOI:10.1002/dac.3101

10. Pantelimon-Teodor Tivig, Eugen Borcoci, Frank Y. Li, Indika Anuradha Mendis Balapuwaduge, Marius Vochin, "Sicing the 5G Core Network Based on SDN Ryu Controller for Everything as a Service", in progress.

6.3.4 List of Projects

Knowledge, innovation and development through doctoral scholarships - CID-Doc – 2014-1015

Investment endowment for the laboratories "Ad Net Market Media for research and development in future mobile communications" - FUTURE-NET-LAB, financed by the EU within the Operational Program Competitiveness (POC), priority axis 1, action 1.1.1:. Large infrastructures of Research and Development, November 2019.

A Massive MIMO Enabled IoT Platform with Networking Slicing for Beyond 5G IoV/V2X and Maritime Services (SOLID-B5G), January 2022.

Increasing the innovative competitiveness of SC Ad Net Market Media SRL through initial innovation investments in order to achieve a SmartDelta technology platform, within a newly established unit for the realization of CD activities in effective collaboration, February 2022.

Collaborator for the management of Master's thesis.

6.4 Future Objectives

The intention for the future is to make contributions in the area of interest of SDN and also in the field of NFV, which is strongly integrated with software-defined networks, but also in tangential technologies Cloud computer and Machine learning.

From an academic perspective, the possibility of entering the teaching staff hierarchy is taken into account.

Bibliography

[1] Feamster, H. Kim and N., „Improving network management with software defined networking," *Communications Magazine, IEEE,* vol. vol. 51, nr. no. 2, p. pp. 114–119, 2013.

[2] Qiang Duan, Nirwan Ansari, and Mehmet Toy, „Software-defined network virtualization: An architectural framework for integrating SDN and NFV for service provisioning in future networks," *IEEE Network,* 2016.

[3] N. Omnes, M. Bouillon, G. Fromentoux, and O. le Grand, „A programmable and virtualized network & IT infrastructure for the internet of things: How can NFV & SDN help for facing the upcoming challenges," *International Conference on Intelligence in Next Generation Networks,* p. pp. 64–69, Mar. 2015.

[4] ETSI Industry Specification Group (ISG) NFV, 2014b. GS NFV-MAN 001 - V1.1.1: Network Functions Virtualisation (NFV); URL:, Management and Orchestration. Technical Report., „http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf," [Interactiv].

[5] ETSI Industry Specification Group (ISG) NFV, a. ETSI GS NFV-EVE 005 V1.1.1: Network Functions Virtualisation (NFV); Ecosystem; URL:, Report on SDN Usage in NFV Architectural Framework. Technical Report. ETSI., „http://www.etsi.org/deliver/etsi_gs/NFV-EVE/001_099/005/01.01.01_60/gs_nfv-eve005v010101p.pdf.," [Interactiv].

[6] Bruno Nunes Astuto, Marc Mendonça, Xuan Nam Nguyen, Katia Obraczka, Thierry Turletti, „A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," 2013.

[7] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig,

„Software-Defined Networking: A Comprehensive Survey," *in Proceedings of the IEEE,* vol. vol. 103, nr. no. 1, pp. pp. 14-76, Jan. 2015.

[8] Presuhn, R., „Version 2 of the Protocol Operations for the SimpleNetwork Management Protocol (SNMP)," RFC 3416 (INTERNET STANDARD), Internet Engineering Task Force," Dec. 2002. [Interactiv]. Available: http://www.ietf.org/rfc/rfc3416.txt.

[9] CIARAN EGAN, Dr. Marco Ruffini, „End-to-end Capacity Reservation in Software Defined Networks," *A dissertation submitted in fulfillment of the requirements for the degree: Integrated Masters In Computer Science at the School Of Computer Science & Statistics, The University of Dublin,* May 2016.

[10] Balakrishnan, N. Feamster and H., „Detecting BGP configuration faults with static analysis," *in Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation, ser.NSDI'05. Berkeley, CA, USA: USENIX Association,* vol. Volume 2, p. pp. 43–56, 2005.

[11] McKeown, Nick et al., „OpenFlow: Enabling Innovation in Campus Networks," In: SIGCOMM Comput. Commun. Rev. 38.2, pp. 69–74. ISSN: 0146-4833, 2008. [Interactiv]. Available: http://doi.acm.org/10.1145/1355734.1355746.

[12] Sheinbein, D. and R. P. Weber, „800 Service Using SPC Network Capability," *In: The Bell System Technical Journal 61.7,* 1982.

[13] Tennenhouse, D.L. et al., „A survey of active network research," *In: Communications Magazine, IEEE 35.1, pp. 80–86. ISSN: 0163-6804. DOI: 10.1109/35.568214.,* 1997.

[14] P. Newman, G. Minshall, and T. L. Lyon, „Ip switching—atm under ip," *IEEE/ACM Trans. Netw., vol. 6, no. 2, pp. 117–129,* Apr.1998.

[15] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, „NOX: towards an operating system for networks," *Comp. Comm. Rev.,* 2008.

[16] [Interactiv]. Available: https://nsrc.org/workshops/2014/nznog-sdn/raw-attachment/wiki/WikiStart/Ryu.pdf.

[17] Srivastava, I.S. A. K. Singh and S., „A survey and classification of controller placement problem in sdn," *International Journal of Network Management,* 2018.

[18] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, „Applying nox to the datacenter," *in HotNets,* 2009.

[19] D. Zeng, C. Teng, L. Gu, H. Yao, and Q. Liang, „Flow setup time aware minimum cost switch-controller association in software-defined networks," *in Heterogeneous Networking for Quality, Reliability, Security and Robustness (QSHINE), 2015 11th*

International Conferenceon. IEEE, pp. 259–264, 2015.

[20] Murat Karakus, Arjan Durresi, „Quality of Service (QoS) in Software Defined Networking (SDN): A Survey," J. Netw. Comput. Appl. 80, C, pp. 200–218., February 2017. [Interactiv]. Available: https://doi.org/10.1016/j.jnca.2016.12.019.

[21] B. Isong, RRS Molose, AM Abu-Mahfouz și N. Dladlu, „Comprehensive Review of SDN Controller Placement Strategies," *în IEEE Access, vol. 8, p. 170070-170092,* 2020.

[22] V. Ahmadi, M. Khorramizadeh, „An adaptive heuristic for multi-objective controller placement in software-defined networks," *Computers & Electrical Engineering. 66. 10.1016/j.compeleceng.2017.12.043.,* 2017.

[23] P.-T. Tivig, A. Brumaru, S. G. Obreja, „Creating Scalable Distributed Control Plane In SDN To Rule Out The Single POINT Of Failure," *COMMS,* 2022.

[24] B. Eugen, A. Tudor, M. Vochina , „Multi-criteria based Optimization of Placement for Software Defined Networking Controllers and Forwarding Nodes," *The Fifteenth International Conference on Networks ICN 2016,* February 2016.