A FRAMEWORK FOR ECONOMIC ANALYSIS OF NETWORK ARCHITECTURES

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Murat Karakus

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2018

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Arjan Durresi, Chair

Department of Computer and Information Science

Dr. Yao Liang

Department of Computer and Information Science

Dr. Mihran Tuceryan

Department of Computer and Information Science

Dr. Yuni Xia

Department of Computer and Information Science

Approved by:

Dr. Shiaofen Fang

Head of the Department Graduate Program

To my wife, kids, parents, and sister. Thank you!

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor, Dr. Arjan Durresi, for his invaluable guidance and unconditional support. Dr. Durresi is a professional, thoughtful and disciplined researcher. His encouragement, patience, ideas, and dedication helped me a lot when I had difficulties throughout my Ph.D. studies. He also gave me many valuable suggestions in my life. I enjoyed the time working with him. This thesis would not have been successful without his guidance and support.

In addition, I would like to thank all my committee members, Dr. Yao Liang, Dr. Mihran Tuceryan, and Dr. Yuni Xia for their willingness to serve in my committee and valuable feedback in my research work.

Last but not the least, I want to thank my lab mates, Yefeng Ruan and Lina Alfantoukh for their kind helps and inspiring discussions.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Karakus, Murat PhD, Purdue University, December 2018. A Framework for Economic Analysis of Network Architectures. Major Professor: Arjan Durresi.

This thesis firstly surveys and summarizes the state-of-the-art studies from two research areas in Software Defined Networking (SDN) architecture: (i) control plane scalability and (ii) Quality of Service (QoS)-related problems. It also outlines the potential challenges and open problems that need to be addressed further for more scalable SDN control planes and better and complete QoS abilities in SDN networks. The thesis secondly presents a hierarchical SDN design along with an inter-AS QoS-guaranteed routing approach. This design addresses the scalability problems of control plane and privacy concerns of inter-AS QoS routing philosophies in SDN. After exploring the roots of control plane scalability problems in SDN, the thesis then proposes a metric to quantitatively evaluate the control plane scalability in SDN. Later, the thesis presents a general framework for economic analysis of network architectures and designs. To this end, the thesis defines and utilizes two metrics, *Unit Service Cost Scalability* and *Cost-to-Service*, to evaluate how SDN architecture performs compared to MPLS architecture in terms of unit cost for a service and cost of introducing a new service along with giving mathematical models to calculate Capital Expenditures (CAPEX) and Operational Expenditures (OPEX) of a network. Moreover, the thesis studies the problem of optimal final pricing for services by proposing an optimal pricing scheme for a service request with QoS in SDN environment while aiming to maximize benefits of both service providers and customers. Finally, the thesis investigates how programmable network architectures, i.e. SDN, affect the network economics compared to traditional network architectures, i.e. MPLS, in case of fail-

ures along with exploring the economic impact of failures in different SDN control plane models.

1 INTRODUCTION

Traditional networks have reached their architectural limitations. Increasing cloud services, server virtualization, sharp growth of mobility, content-like video, and big data etc. have led researchers and network operators/administrator/analysts to rethink today's network architectures. In traditional architectures, network devices and appliances are complex and difficult for (re)configuration and (re)installation since they require highly skilled personnel. Adding or moving a device from a network requires extra costs. It is also time-consuming because IT people need to deal with multiple switches, routers, etc. and update ACLs, VLANs and other mechanisms [1]. Furthermore, as business demands or user needs increase day by day, application developers, carriers, and enterprises delve into evolving new services and facilities. However, vendor dependency is an obstacle deterring them from developing new networking applications and services for their networks due to slow equipment product cycle, application testing and deployment. Modifying forwarding hardware in networking devices like routers and switches renders new network applications possible. If developers become more capable of making changes to forwarding hardware in routers and switches, new applications can be developed without dependency of equipment vendors. Therefore, today's data centers, carriers, and campuses need more dynamic architectures.

Software Defined Networking (SDN) [2–6] architecture has emerged in response to the aforementioned limitations of traditional networking architectures. SDN aims to decouple the control plane and data plane. This separation provides network operators/administrators with efficient use of network resources and eases provisioning of resources. Also, SDN brings ease of programmability in changing the characteristics of whole networks. This simplifies the management of the network, since it is decoupled from the data plane. Therefore, network operators can easily and quickly manage,

configure, and optimize network resources with dynamic, automated and proprietary-free programs in SDN architecture [1, 7]. Google's datacenter WAN, B4 [8], is one of the examples for SDN adopted in a large-scale network with the aforementioned purposes . In addition, since the network is logically centralized in SDN, controllers have a global visibility of the whole network unlike conventional networking. Hence, they can dynamically optimize flow-management and resources.

1.1 Network Programmability

Network programmability has been proposed as a solution to mitigate the deficiencies as mentioned earlier for the traditional architectures. While there is no consensus on the definition of the network programmability, it is mostly accepted that a network is programmable when the control and data planes are separated by an interface that allows modification and monitoring of the network state through machine readable data-driven APIs. The value of the network programmability is three-fold: (a) the reduction in complexity enabled by centralized routing decisions, (b) the ability to allow applications to interact with the control plane, and (c) automation of network-related tasks such as network resource configuration/optimization through proprietary-free programs written by network administrators [9].

Network programmability has its own benefits in terms of both network management and network revenues. Today, networks are mostly configured manually for provisioning, which requires considerable time, effort, and expertise. This process is error-prone and can lead to many mistakes due to human interventions. Also, manual repetition of the same configuration across a large number of heterogeneous devices inevitably increase the possibility of having some errors in someplace. Network programmability provides automation that is a cure for large-scale repetition of common tasks that will result in saving time and making the network more error-free and available by reducing the Mean Time Between Mistakes (MTBM). Automating these

tasks can drive the bottom line down by reducing the OPerational EXpenditures (OPEX) required to run the network as well.

Moreover, programmable networks help increase the network utilization. Network traffic often follows daily, hourly, weekly, and seasonal patterns. These patterns may require rapidly moving of traffic around to less utilized links. Since this rapid modification to the network is not trivial in the traditional architectures, network administrators often over provisions their network resources, which increases the CAPital EXpenditures (CAPEX) in the network. For example, bandwidth is selected to support the highest traffic across a single link. During non-peak times, this bandwidth is not used and is, therefore, a cost that has no return on investment is incurred. In a programmable network, traffic can be engineered in near real time to adjust the increasing load on the network. This dynamic nature prevents network operators from unnecessary network capacity increases and, thus, produces CAPEX savings. Planning bandwidth usage in advance in this way is called bandwidth calendaring. While calendaring cannot replace the need for effective bandwidth planning, it can move the emphasis away from building for peak load and toward building for a number closer to the average load across longer time periods.

There have been different attempts to realize network programmability in the literature. Active Networks [10], software routers such as Click Router [11] and XORP [12], Overlay Networks [13], and Path Computation Element (PCE) [14] are just a few examples to name from both academy and industry. SDN is another recent networking paradigm to bring network programmability in use of networking industry.

1.2 Problem Statement

Increasing revenue for a network is crucial for its operations as well as future. Networks can increase their revenue by serving more end-users, reducing CAPEX/OPEX, introducing new services and so on. Network owners need wise plans before making decisions on investing money. These plans include deciding on which networking

architecture is appropriate for specific types of services, pricing those services, scalability aspects and programmability points of architectures and so on. These are all challenging decisions and have impact on sustainability of a network.

Network architectures, such as IntServ, DiffServ, and MPLS, have been proposed over the years based on the needs of end-users. Each network architecture has its own advantages and disadvantages. For example, IntServ is capable of per-flow resource reservation unlike the DiffServ architecture, IntServ architecture suffers from scalability problems. These pros and cons define the limits of the network architectures and have impact on how network services are provided to customers. In addition to network architecture, network topology designs, such as centralized, distributed, and hierarchical, can also have impact on a network services and operations due to some factors such as latency, bandwidth provisioning, overhead generation and so on. Therefore, it is important for a network owner to aware advantages and disadvantages of network architectures and topology designs regarding economic aspects of the network before investing on specific ones.

Also, having a scalable network is crucial for providing network services to customers. If the network is not scalable, some service interruptions or cuts can occur, which can then adversely affect the revenue of the network. Scalability proposals in literature study the network scalability problem in terms of improving the scalability of architectures or topology designs with respect to some network metrics such as throughput and latency and do not propose a metric to quantify the scalability itself. However, a metric to quantify scalability of a network architecture or design can provide network owners and administrators some insights while they construct their networks.

Moreover, determining the unit cost of a service that a network provides to its customers is vital for networks to sustain their businesses in the market. This process is followed by determination of an optimum final price to be charged by the networks for their services. On one hand, network owners want to charge more from customers to maximize their profit and revenues. On the other hand, if they determine the costs

and final prices for their services too low or high, they may have the risk of making loss or losing customers in the long run. Therefore, how to determine the cost and set the optimal prices for network services to make the revenue and profit maximized while attracting customers is a challenging problem. Although many solutions have been proposed from both industry and academia, pricing the Internet services is still an ongoing research problem for researchers. Both of these phases are crucial and require careful and correct mechanisms for a network's business future.

1.3 Dissertation Statement

Recently, SDN has emerged as a new way to architect networks by providing network programmability and exposing network APIs. SDN has got the attention of researchers from both academia and industry as a means in order to decrease network costs and generate revenue for service providers due to features it promises in networking [15–22]. SDN paradigm has several key attributes that have an impact on the CAPEX and OPEX equations of a network. Some of the main attributes of SDN are network programmability, hardware and software independence, virtualized software infrastructure, multi-tenancy, and resource pooling and so on. These features as well as how SDN can impact CAPEX and OPEX for a network are explained later in the thesis.

To address the aforementioned challenges, this thesis firstly introduces a scalable hierarchy-based SDN network design to mitigate the scalability problem of SDN architecture. The proposed design helps network administrators reduce their CAPEX and OPEX through the scalability-driven design philosophies adopted as explained later in the thesis. It also addresses QoS routing privacy problem of networks, which is one of the challenging issue of inter-AS QoS routing problem.

Secondly, this thesis presents a metric in order to evaluate the control plane scalability in SDN. The metric is introduced after carefully exploring the roots of control plane scalability problem in SDN. The thesis also gives mathematical models of the

proposed metric over different control plane designs. The metric realizes the real scalability performance of a network by not just focusing on typical network parameters such as throughput and latency. It also captures the relationship between the unit service cost and scalability as shown later in the thesis.

Thirdly, this thesis defines two metrics, "*Unit Service Cost Scalability*" and "*Cost-To-Service*, to measure the unit cost for a service request as well as cost of introducing a new service in the network in case of SDN architecture. The thesis compare the SDN architecture to MPLS architecture to evaluate how programmable networking performs compared to traditional networking using the two metrics. The thesis also presents mathematical models to calculate CAPEX and OPEX of an SDN network. Later, the thesis also presents an optimal final pricing scheme to charge users while maximizing network benefits of both the network an customers.

Finally, the thesis also investigates how programmable network architectures, i.e. SDN technology, affect the network economics compared to traditional network architectures, i.e. MPLS technology, under certain failure scenarios. This work also aims at being a useful primer to providing insights regarding how network architectures and control plane models perform with respect to network economics under failures for network owners to plan their investments accordingly.

1.4 Dissertation Organization

This chapter outlines the structure and presents a brief overview of the chapters.

Chapter 2 surveys scalability problems of the control plane (i.e. controllers) in SDN architectures as opposed to other general SDN surveys. It discusses the main causes that make the control plane suffer from scalability issues in an SDN architecture. It also presents characterizations and classifications of proposals based on the primary concepts exploited to alleviate the controller scalability issues.

Chapter 3 aims at making a picture of QoS-motivated literature in OpenFlow-enabled SDN networks by surveying relevant research studies. It organizes the related

studies into seven categories that are the most prominent ways in which QoS can benefit from the concept of SDN. These categories (i.e. problems) and related studies are explained (i.e. solutions) in corresponding sections. In addition, QoS capabilities of OpenFlow protocol is discussed by reviewing its versions along with some well-known, open-source, and community-driven control platform projects. Finally, it outlines the potential challenges and open problems that need to be addressed further for improved and complete QoS abilities in OpenFlow-enabled SDN networks.

Chapter 4 proposes a hierarchy-based network architecture along with an inter-AS routing approach with QoS. It exploits idea of levels in which networks with controllers reside and a main controller, which works like a broker, is on top of them to keep the global network state and view. The experiment results indicate that a controller in a hierarchic setting handles 50% less number of traffic than a controller in a non-hierarchic environment.

Chapter 5 firstly explores the roots of control plane scalability problem in SDN as well as proposed existing solutions. A metric is then proposed in order to evaluate the control plane scalability in SDN. This chapter also gives mathematical models of the proposed metric over different control plane designs. Furthermore, the performance of these control plane designs is compared by extensive experiments.

Chapter 6 investigates how programmable networking, i.e. SDN technology, affects the network economics compared to traditional networking, i.e. MPLS technology. To this end, this work defines two metrics *Unit Service Cost Scalability* and *Cost-to-Service* to evaluate how SDN architecture performs compared to MPLS architecture. Also, mathematical models are presented to calculate certain cost parts of a network. In addition, a comparison of different popular SDN control plane models, Centralized Control Plane (CCP), Distributed Control Plane (DCP), and Hierarchical Control Plane (HCP), are given to understand the economic impact of them with regards to the defined metrics. Video service with different traffic patterns, (1) 20% (inter-domain) - 80% (intra-domain), 2) 50% (inter-domain) - 50% (intra-domain), and 3)

80% (inter-domain) - 20% (intra-domain), has been used for the comparison due to its QoS requirements and the facts explained earlier.

Chapter 7 proposes an optimal pricing scheme for a service request with QoS in SDN environment using the Nash bargaining problem, which aims to maximize benefits of both service providers and customers. It integrates a new cost function and network connectivity degree factor into the proposed pricing scheme. In addition, it gives a general scheme of revenue and profit that a service provider makes. This scheme employs the idea of *penalty* for each request that the service provider cannot provide to its customers. Furthermore, this work applies these schemes in a scalable SDN-based hierarchic architecture and evaluate with extensive experiments.

Chapter 8 investigates how programmable network architectures, i.e. SDN technology, affect the network economics compared to traditional network architectures, i.e. MPLS technology, in case of failures. In addition, it explores the economic impact of failures in different SDN control plane models: Centralized (Single) Control Plane (CCP), Distributed (Flat) Control Plane (DCP), and Hierarchical Control Plane (HCP). This work exploits the predefined metric called *Unit Service Cost Scalability* to evaluate economic performances of SDN architecture along with aforementioned control plane models and MPLS architecture under different failure scenarios. It considers two different failure types: i) a random single data plane link failure and ii) a random controller (i.e. control plane) failure.

Finally, Chapter 9 concludes this dissertation with concluding remarks and provides directions for future work.

2 A SURVEY: CONTROL PLANE SCALABILITY ISSUES AND APPROACHES IN SOFTWARE-DEFINED NETWORKING (SDN)

2.1 Abstract

SDN architecture has emerged in response to limitations of traditional networking architectures in satisfying today's complex networking needs. In particular, SDN allows network administrators to manage network services through abstraction of lower-level functionality. However, SDN is a logically centralized technology. Therefore, scalability, and especially the control plane (i.e. controller) scalability in SDN is one of the problems that needs more attention. This survey study first discusses the scalability problems of controller(s) in an SDN architecture. It then comprehensively surveys and summarize the characterizations and taxonomy of state-of-the-art studies in SDN control plane scalability. The study organizes the discussion on control plane scalability into two broad approaches: Topology-related approaches and Mechanisms-related approaches. Topology-related approaches study the relation between topology of architectures and scalability issues. It has sub-categories of Centralized (Single) Controller Designs and Distributed approaches. Distributed approaches, in turn, have also sub-categories: Distributed (Flat) Controller Designs, Hierarchical Controller Designs, and Hybrid Designs. Mechanisms-related approaches review the relation between various mechanisms used to optimize controllers and scalability issues. It has sub-categories of Parallelism-based Optimization and Control Plane Routing Scheme-based Optimization. Furthermore, this study outlines the potential challenges and open problems that need to be addressed further for more scalable SDN control planes.

2.2 Introduction

Increasing cloud services, server virtualization, sharp growth of mobility and content-like video have led researchers to rethink today's network architectures. In traditional architectures, network devices and appliances are complex and difficult for (re)configuration and (re)installation since they require highly skilled personnel. Adding or moving a device from a network requires extra costs. It is also time-consuming because IT people need to deal with multiple switches, routers, etc. and update ACLs, VLANs and other mechanisms [1]. Furthermore, as business demands or user needs increase day by day, application developers, carriers, and enterprises delve into evolving new services and facilities. However, vendor dependency is an obstacle deterring them from developing new networking applications and services for their networks due to slow equipment product cycle, application testing and deployment. Therefore, today's data centers, carriers, and campuses need more dynamic architectures.

SDN architecture has emerged in response to the aforementioned limitations of traditional networking architectures. SDN aims to decouple the control plane and data plane. This separation provides network operators/administrators with efficient use of network resources and eases provisioning of resources. Also, SDN brings ease of programmability in changing the characteristics of whole networks. This simplifies the management of the network, since it is decoupled from the data plane. Therefore, network operators can easily and quickly manage, configure, and optimize network resources with dynamic, automated and proprietary-free programs in SDN architecture. Google's datacenter WAN, B4 [8], is one of the examples for SDN adopted in a large-scale network with the aforementioned purposes. In addition, since the network is logically centralized in SDN, controllers have a global visibility of the whole network unlike conventional networking. Hence, they can dynamically optimize flow-management and resources.

Despite the advantages of centralized control in SDN architectures, SDN faces some issues challenging its nature (i.e. centralized control) due to day by day increasing network demands. Although network operators enhance the performance of the network controllers, it still cannot be enough to meet the high network demands such as flow request and monitoring network statistics. For example, one of the earlier SDN controllers, NOX [23], can serve only 30K flow requests per second with a response time less than 10 ms. This insufficiency appears more in large-scale networks or data centers compared to small networks. Kandula et al. [24] report that a cluster of 1500 servers receives 100K flows per second on average. Also, Erickson [25] states that a network with 100 switches can result in 10 million flow arrivals per second in the worst case. These numbers indicate that the control plane in an SDN architecture is prone to suffer from scalability issues due to its centralized nature. Furthermore, Sezer et al. [1] state that one of the main challenges in SDN is the scalability issue, which especially needs more attention by researchers. Therefore, understanding and improving the scalability of the SDN control plane (i.e. controller) is a critical problem for successful adoption of SDN for large scale networks or networks with many flows.

2.2.1 Chapter Organization

This chapter surveys scalability problems of the control plane (i.e. controllers) in SDN architectures as opposed to other general SDN surveys. It discusses the main causes that make the control plane suffer from scalability issues in an SDN architecture. It also presents characterizations and classifications of proposals based on the primary concepts exploited to alleviate the controller scalability issues. In addition, the study points out the main challenges along with existing proposals in controller scalability.

The data plane scalability in SDN is not a part of this study's scope. However, as a brief note, data plane scalability in SDN is mostly dominated by (1) processing

power, (2) capacity of memory/buffer, and (3) software implementation of data plane devices. For a more detailed and comprehensive discussion on data plane scalability in SDN, the readers are referred to following studies: [26–33].

In the remaining sections of the chapter, Section 2.3 gives a light-weight overview of the SDN framework with OpenFlow protocol. Section 2.4 discusses the *Scalability* concept regarding its meaning and present some scalability metrics proposed in the literature to quantitatively measure the scalability of systems both in general and SDN context as well as contributors to scalability issues in SDN. Section 2.5 presents the organization of the studies over control plane scalability in SDN. Section 2.6 outlines the relation between topology of architectures and scalability issues while Section 2.7 discusses the relation between other mechanisms used to optimize the controller performance and scalability issues. Section 2.8 presents a comparative discussion over control plane scalability proposals. In Section 2.9, the study outlines the potential challenges and open issues that need to be addressed further for fully scalable SDN control planes in the future in a nutshell. Finally, Section 2.10 wraps the chapter up with concluding remarks.

2.3 An Overview of SDN Architecture and OpenFlow Protocol

SDN architecture with OpenFlow protocol enables network operators to treat flows in a finer-granular way compared to the traditional networks by means of controllers. In a traditional network, flows (or packets) are mainly treated based on a single or a few attribute combinations of packet headers, such as longest destination IP prefixes, destination MAC addresses, or a combination of IP addresses and TCP/UDP port numbers etc. SDN allows to manage flows based on more attributes of packet headers by means of a Controller-Data Plane Interface (C-DPI) such as OpenFlow protocol [34–37].

As shown in Fig. 2.1, Open Networking Foundation (ONF)[1] vertically splits SDN architecture into three main planes [38]:

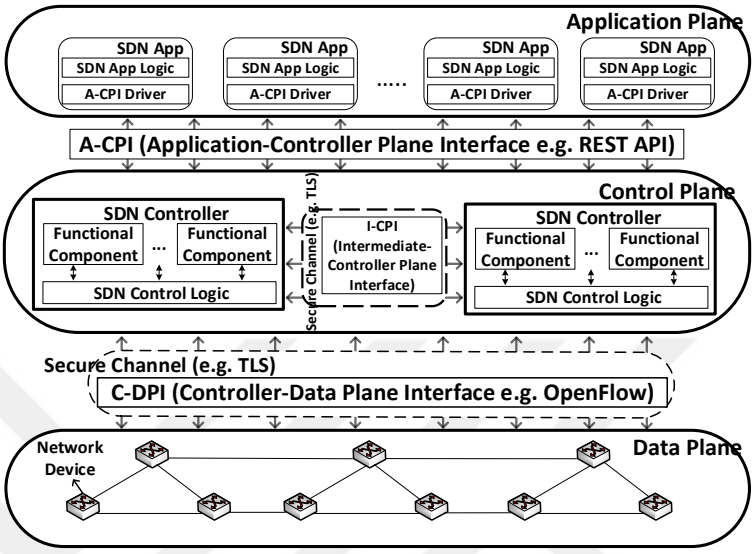


Figure 2.1.: An Overview of an SDN Control Plane. Main components in a control plane of an SDN network are a controller(s) and interfaces (e.g. A-CPI, C-DPI, and I-CPI).

- Data Plane: The data plane is the bottom plane and consists of network devices such as routers, physical/virtual switches, access points etc. These devices are accessible and managed through C-DPIs by SDN controller(s). The network elements and controller(s) may communicate through secure connections such as the TLS connection. OpenFlow protocol is the most prevalent standard C-DPI used for communication between controller(s) and data plane devices.

- Control Plane: An SDN control plane comprises a set of software-based SDN controller(s) to provide control functionality in order to supervise the network forwarding behavior through C-DPI. It has interfaces to enable communication among controllers in a control plane (Intermediate-Controller Plane Interface, i.e. I-CPI [39], optionally secured using the TLS), between controllers and network devices (C-DPI), and also between controllers and applications

[1]https://www.opennetworking.org/

(Application-Controller Plane Interface, i.e. A-CPI). An A-CPI[2] renders possible the communication between network applications/services and controller(s) for network security, management etc. A controller consists of two main components: Functional components and control logic. Controllers include more than one functional components such as Coordinator, Virtualizer etc. to manage controller behaviors. Furthermore, SDN control logic in a controller maps networking requirements of applications into instructions for network element resources [38].

- Application Plane: An SDN application plane consists of one or more end-user applications (security, visualization etc.) that interact with controller(s) to utilize an abstract view of the network for their internal decision making process. These applications communicate with controller(s) via an open A-CPI (e.g. REST API). An SDN application comprises an SDN App Logic and A-CPI Driver.

In an SDN network with OpenFlow protocol and OpenFlow-enabled switches, there are three main parts in a switch: Flow Table, Secure Channel, and OpenFlow Protocol. An OpenFlow switch maintains a number of flow tables containing a list of flow entries. Each flow entry consists of 3 parts: A "Rule" field to define the flow entry based on certain header attributes such as source/destination addresses, an "Action" field to apply on a packet matching the values in the "Rule" field, and a "Stats" field to maintain some counters for the entries [37]. A Secure Channel (e.g. TLS) is the interface that connects data plane elements to a remote controller. Switches are managed and configured by the controller over the secure channel. In addition, the controller receives events from the switches and sends packets out to switches through this channel.

In SDN, a controller can work in three operational modes to setup a new flow rule (a.k.a flow entry): reactive mode, proactive mode, and hybrid mode [40]:

[2]An A-CPI is mostly called "Northbound Interface" by the SDN community.

- Reactive Mode—In the reactive mode, when a new packet arrives to a network device (e.g. switch), the switch does a flow rule lookup in its flow tables. If no match for the flow is found, the switch forwards it to the controller using C-DPI so that the controller decides how to handle the packet. After the controller processes the packet according to the network policies, it creates and sends a flow entry to be installed in the network device. Future flows matching with this flow entry based on packet header attributes will be treated according to the corresponding matching rule.

- Proactive Mode—In the proactive mode, flow entries are setup in flow tables of the switches before new flows arrive at the switches. When a packet arrives at a switch, the switch already knows how to deal with that packet. In this case, the controller is not involved in any flow rule setup process.

- Hybrid Mode—In the hybrid mode, a controller benefits advantages of both reactive and proactive modes. It is quite possible that network administrators proactively install certain flow entries in data plane devices and the controller(s) reactively modify (delete/update) them or even add new flow entries based on incoming traffic.

While the proactive mode brings some concerns regarding inefficient use of switch memory, the reactive mode provides more agile, flexible, and dynamic environment for both controllers and switches [40].

2.3.1 Scalability Support in OpenFlow Protocol

There are also some scalability related improvements in OpenFlow specifications. One improvement is the "group table mechanism" specified in version 1.1 [41] and later. A group table consists of group entries. A group entry, in turn, consists of a group identifier, a group type, counters, and a list of action buckets. This mechanism enables multiple flow table entries to point to the same group identifier, so that the

group table entry is performed for multiple flows. For example, if you need to update the action on this set of flow table entries (all have the same action), the controller can only update the pointed group table entry action instead of updating the action of all flow table entries. Another improvement is that it provides multiple controller support as of its version 1.2 [42] through the "controller role change mechanism". This scheme enables a switch to establish communication with a single controller or multiple controllers in parallel under different controller roles such as master, equal, and slave.

2.4 Scalability and Its Causes in SDN

Scalability is a frequently-claimed attribute of various systems. It is a multi-dimensional topic. While the basic notion is intuitive, the term *scalability* does not evoke the same concept to everybody. Therefore, there is no general precise agreement on neither its definition nor content. While some people may refer to scalability as optimization of processing power to CPUs, others may define it as a measure of parallelization of applications across different machines. However, regardless of its meaning to someone, it is a desired property indicating positive sense regarding a system, architecture, algorithm and so on.

Furthermore, trade-offs concerning some concepts such as performance, resiliency, availability, reliability and flexibility have to be taken into account by network designers and managers while designing a network architecture [43]. A "solution" proposed as a scalability cure for a network may introduce trade-offs that harm other useful properties of the network. For example, in the context of SDN, proactive rule installation in SDN switches decreases the load of the controller, thus reducing the processing time and flow initiation overhead in the controller. However, this constraints the flexibility coming from reactive flow installation and reduces decision-making dynamicity of the controller and management of the network. Also, controller distribution is one way to overcome computational load on controller but it brings consistency and

synchronization problems as well. Therefore, scalability is not an independent problem that can be exclusively dealt with but is a combination of issues that introduces trade-offs to be explicitly stated while proposing a remedy.

2.4.1 Existing Scalability Metrics in General

There are several research efforts [44–50] proposing a metric to measure scalability of systems. Most of these metrics are for homogeneous environments. The majority of these proposals revolve around two major types of scalability metrics: Isospeed scalability and Isoefficiency scalability.

The Isospeed scalability is characterized by the fact that an achieved average unit speed of an algorithm on a given machine can remain constant with increasing number of processors and problem size for an algorithm-machine combination [44]. In [45], the authors present a metric to describe the scalability of an algorithm-machine combination in homogeneous environments. Their scalability function is defined as $\psi(p, p') = \frac{p'W}{pW'}$ where p and p' are the initial and scaled number of processors of the systems respectively, and W and W' are the initial and scaled problem size (workload) respectively.

The Isoefficiency scalability is described as the ability of parallel machine to keep the parallel efficiency constant when the system and problem size increase [46]. The parallel efficiency is defined as speedup over the number of processors, i.e. $E = \frac{S}{p}$. Speedup is also given by the ratio of problem size (W) and parallel execution time (T_p), i.e. $S = \frac{W}{T_p}$ where $T_p = \frac{W+T_0(W,p)}{p}$ with $T_0(W, p)$ extra communication overhead [47].

Pastor and Orero [48] define heterogeneous scalability by presenting a heterogeneous efficiency function. They attempt to extend the homogeneous *Isoefficiency* scalability model to heterogeneous computing and, therefore, their work inherits the limitation of parallel speedup, requiring the measurement of solving large-scale problem on single node. Sun et al. [49] propose a scalability metric called Isospeed-efficiency for general heterogeneous computing systems. This metric combines the

roots of both Isospeed scalability and Isoefficiency scalability metrics by means of a concept called "Marked Speed" to describe the computing power for a stand-alone node and a combined computing system.

2.4.2 Scalability in SDN

In SDN networks, controller performance is one of the primary concerns while designing more scalable networks. There are many studies exploring performances of controllers with respect to different network workload, implementations, architectures and so on [51–55]. Although studies evaluate scalability performance of controllers they propose regarding various performance metrics, such as path installation time, link utilization, and so on, depending on their target problem, the most prominent and considered metrics are control plane *throughput*, which refers to the number of flow requests handled per second, and (flow setup) *latency*, which refers to the delay to respond flow requests, in SDN context. In SDN, a controller needs to proactively or reactively set up (i.e. handle) and tear down flow-level forwarding state in OpenFlow switches. Once set up, the flow forwarding state remains cached on the OpenFlow switches so that this process is not repeated for subsequent packets in the same flow. This setup process includes a latency as well. It is perceived that this flow setup process is to be likeliest source of control plane (i.e. controller) performance bottleneck by the SDN community. Hence, the number of flow requests handled per second (throughput) and flow setup latency come into prominence in evaluation of control plane scalability performance. Therefore, the term *Scalability*, particularly control plane scalability in SDN context, is characterized by the aforementioned two metrics, throughput and flow setup latency, as well as in this work. A more detailed comparison of studies with respect to their scalability performance in terms of throughput and flow setup latency metrics is given in Section 2.8.

There are also few research efforts proposing a metric to quantify scalability of SDN networks. Hu et al. [56] present a metric for SDN control plane scalability. They

use the scalability metric, which is based on productivity of a distributed system, presented in [50] to quantify the scalability of SDN control plane by adapting to the SDN case. A similar work in [57] also proposes a metric to quantify control plane scalability by ratio of workload (number of flows entering the network through the data plane) and overhead (number of messages processed in the control plane). However, these metrics have been proposed recently. Therefore, their adoption by SDN research community may be seen as one of the metrics for scalability performance measurement by the time.

2.4.3 Contributors to Scalability Issues in SDN

SDN is a logically centralized architecture, therefore scalability is one of the crucial issues to be addressed in SDN as in many traditional networks [58]. However, in particular, scalability concerns of the control plane in SDN are intrinsic to SDN owing to its separated structure. This section points out the main reasons that make the control plane a scalability bottleneck in SDN.

- Separation of Control Plane and Data Plane: The separation of the data plane and control plane is a contributor to scalability issues of the SDN architecture, particularly control plane scalability, since this decoupling requires the management of network devices from a remote controlling mechanism (i.e. controller). Since data plane devices have no longer ability to make decisions about traffic packets a communication has to be established with controllers to receive corresponding decisions about the packets. This communication brings extra message burden for both controllers and data plane devices. Therefore, this separation may result in significant signaling overhead between control plane and data plane, depending on the network architecture (e.g. distributed, hierarchical etc.) and applications on top of the controller. Hence, this makes the control plane play a bottleneck role regarding the scalability of the system.

- Quantity of Events/Requests Handled by a Controller: This problem pertains more to the single controller designs than to the distributed (flat), hierarchical or hybrid designs since it results from the centralization of computation at a single central entity. An increase in the number of network devices reinforces the foregoing problem for controllers. As the network grows with respect to the size of the nodes (e.g. hosts, switches etc.), the controller will have to cope with more events and flow requests, which can make the controllers a bottleneck point due to its limited computation resources such as CPU and memory. Therefore, the number of control messages sent by data plane devices to the controller(s) becomes one point to be addressed because the controller may not be able to handle all the incoming requests. For example, a NOX controller can handle up to 30K requests/sec, which is enough for small to mid-size networks [59]. However, that number may not be enough for some network settings, such as data centers, depending on the number of servers and the switches [24,60]. This issue may also result in delay in programming of data-plane (devices) since it may increase flow rule setup process delay at controller, which eventually affects the speed of the network.

- Controller-Switch Communication Delay: As stated in [61], the controller's placement (distance between network devices and controller) is one of the factors that introduces latency into the flow setup time. Flow setup latency is typically determined by switch packet processing time, RTT (*round-trip-time*) between controller and switches, and controller packet processing time. If the controller-switch communication delay (determined by RTT) is high, then resulting flow setup latency becomes high too, which causes longer flow rule addition, deletion or update in switch flow tables. This, in turn, may result in congestion in both control plane level and data plane level and longer failover time in the network. Hence, scalability of the controller degrades. Although this delay depends on physical distance between controllers and data plane devices, a well-defined placement of controllers may minimize the delay. This particularly becomes

important in WAN compared to small scale networks. Azodolmolky et al. [62] outline a comprehensive analytical model for the behavior of a scalable SDN deployment regarding boundary performance of event processing delay and buffer space of SDN controllers by means of the network calculus as a mathematical framework.

SDN brings the possibility of various network innovations, but lacks uniform definitions and standard implantation in reality. Many essential issues of the controller (plane), however, need to be well addressed so as to improve the development and usages of SDN.

2.5 Classification of Control Plane Scalability Proposals

As discussed in Section 2.4, there is no consensus on the definition of scalability. Therefore, it is not easy to present an unified classification for scalability solutions. The organization presented reflects this study's own point of view over the proposed studies in SDN control plane scalability.

As shown in Fig. 2.2, this study organizes the discussion on control plane scalability into two broad approaches.

The first approach is Topology-related Approaches with sub-categories of Centralized (Single) Controller Designs and Distributed approaches. This category studies the relation between topology of architectures and scalability issues. Distributed approaches are Distributed (Flat) Controller Designs, Hierarchical Controller Designs and Hybrid Designs. Reducing the workload on a controller will result in a better performance of the controller regarding scalability. Therefore, distribution of control plane (i.e. controller) workload among controllers is one way related to the scalability. Hybrid designs represent the studies that leverage the data plane by devolving some limited control functions to the switches to partition the control plane workload. This approach is hybrid due to involvement of both the control plane and data plane in the network control. It differs from the distributed (flat) and hierarchical designs in

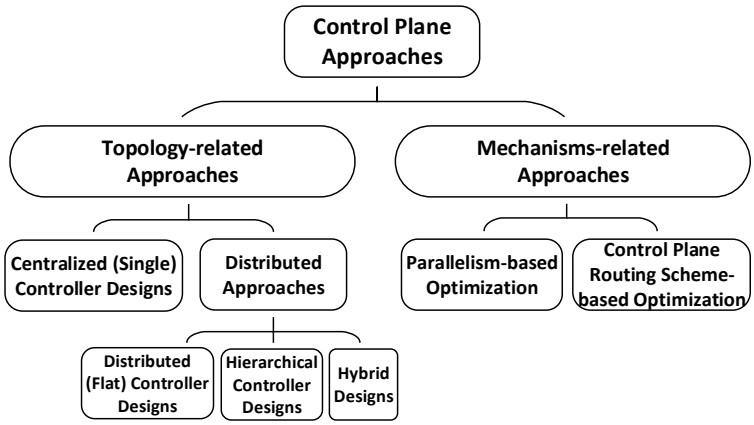


Figure 2.2.: Taxonomy of Control Plane Approaches in SDN. The proposed approaches are categorized into two categories with sub-categories. Topology-related approaches revolve around structure of the framework to distribute the total workload that the controllers handle. Mechanisms-related approaches offer different ways of optimization for controllers and application implementations.

the way that switches are involved in decision processing and network control. These approaches are explained further in the corresponding subsections throughout the chapter.

The second approach, Mechanisms-related Approaches, reviews the relation between various mechanisms used to optimize controllers and scalability issues. Enhancing controllers with respect to their performance by some optimization techniques results in better scalability performance too. In addition, reducing the events resulting from routing mechanism of a controller is another way to increase the scalability in control plane since routing process brings worth considering load to controller.

Also, some proposals may seem to belong more than one category. Hence, this work classifies and presents such proposals by mainly focusing on their primary approaches.

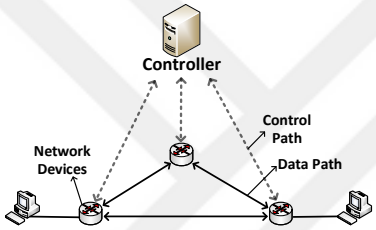
Table 2.1 shows the network types targeted by the studies. Most of the proposals target data centers, enterprise networks or WAN networks since these networks are more vulnerable to the control plane scalability issues.

Table 2.1.: Network types targeted by the studies. Most of the proposals target data centers (DC), enterprise networks or WAN networks.

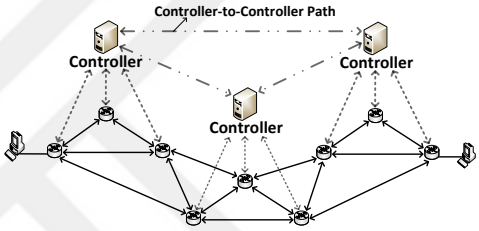
Network Types / **Proposals**	**Campus**	**Cloud**	**DC**	**Enterprise**	**WAN**
Beacon [25]				✓	
DEFO [63]					✓
DevoFlow [64]			✓		
DIFANE [65]				✓	
DISCO [66]			✓	✓	✓
D-SDN [67]					✓
ElastiCon [68]			✓	✓	
Ethane [69]	✓			✓	
Fibbing [70]	✓	✓	✓	✓	✓
FlowBroker [71]					✓
HyperFlow [72]		✓	✓		
Kandoo [73]	✓	✓	✓		
Logical xBar [74]					✓
Maestro [75]			✓		✓
McNettle [76]				✓	
NOX [23]			✓	✓	
NOX-MT [51]			✓	✓	
Onix [77]		✓	✓	✓	✓
ONOS [78]				✓	✓
Orion [79]					✓
Tavakoli et al. [80]			✓		
Tam et al. [81]			✓		
Yazici et al. [82]	✓		✓		
Bari et al. [83]					✓
Karakus et al. [84]					✓
Owens et al. [85]	✓	✓		✓	✓
Soliman et al. [86]					✓

2.6 Topology-related Approaches

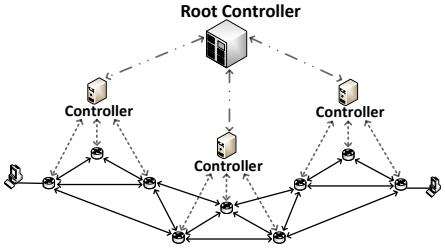
This approach reviews the relation between topology of architectures and scalability issues. The proposals that use different topology models, illustrated in Fig. 2.3, can be classified in four prevalent architectures: Centralized (Single) Controller Designs, Distributed (Flat) Controller Designs, Hierarchical Controller Designs, and Hybrid Designs. These designs have their own intrinsic advantages and disadvantages with respect to control plane scalability. These architectures are explained and the related studies are presented in corresponding subsections below.



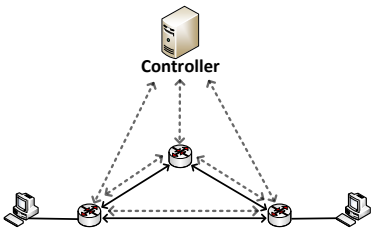
(a) Centralized (Single) Controller Design



(b) Distributed (Flat) Controller Design



(c) Hierarchical Controller Design



(d) Hybrid Design

Figure 2.3.: An Overview of Topology-related Architectures. The two-sided solid, dashed, and dashed-dotted arrows represent two-way data path among network devices, control path between controller and data devices, and controller-to-controller path among controllers, respectively. In 2.3a (Centralized (Single) Controller Design), there is one main controller with global network state. In 2.3b (Distributed (Flat) Controller Design), every controller is responsible for different sites/parts of network(s) with partial or full shared network view. In 2.3c (Hierarchical Controller Design), there are levels in which controllers are responsible for different sites (subdomains) and a Root controller on top with global network view for global applications like routing. In 2.3d (Hybrid Design), data plane devices are also involved in network control.

2.6.1 Centralized (Single) Controller Designs

This type of architecture settings revolve around a single central controller [23,69] with a global network view. The design of this architecture is simple and it is easy to manage the network. This design may meet the needs of small to mid-size networks. However, it is not efficient to handle the burden of environments such as data centers and large-scale networks due to number of events/requests that the controller must handle as stated in the Section 2.4. Therefore, a single controller design is considered less scalable compared to distributed (flat) controller, hierarchical controller and/or hybrid designs.

The authors in [69] develop a new networking architecture called "Ethane" that targets the enterprise networks although it is first deployed in campus network. In an Ethane network, network managers are able to define policies and each request that is not matching a flow entry has to traverse through the controller. There are three concerns that the authors address and resolve in this architecture. First, Ethane renders that high-level policies become the authority part to control the network. Second, the packet paths are managed by policies in order to have better control and global network view. Third, the Ethane network requires a precise binding between a packet and its origin to be able to identify where the packet coming is from.

NOX [23] is inspired by the need for a centralized and uniform programmatic interface that would make a network more manageable. NOX is a network operating system that is more than just a controller platform for a network. As in most SDN controller platforms, NOX treats the packets based on the first packet of a flow traversing through the controller. This flow-based method helps in having more granular control over the traffic in a network. In [80], the authors investigate whether generalized solutions such as NOX can handle characteristic requirements of specialized environments such as datacenters.

2.6.2 Distributed Approaches

This approach classifies and presents the studies [64–68, 70–74, 77–79, 81–84, 87] that distribute the control plane workload on controllers based on topological models, such as flat, hierarchical as well as hybrid designs. As using distributed controllers brings advantages such as load distribution and avoiding centralized (single) controller failure, it brings some challenges such as overhead from controller communication, latency due to state synchronization, and (policy/state) consistency among controller instances that are being addressed by researchers. These challenges are discussed in Section 2.9.

2.6.2.1 Distributed (Flat) Controller Designs

In this structure, each controller manages a sub-network/domain of the whole network. There are two strategies for distributed controller architectures to implement controller's network view. In the *local view strategy*, each controller has its own local network view and each of its neighboring local networks is abstracted as a logical node. In the *global view strategy*, on the other hand, each controller has a global view of the whole network. In both cases, the controllers need to communicate through controller-to-controller channels to exchange needed state information (e.g. reachability information) regarding their domains.

HyperFlow [72] is logically centralized albeit its distributed architecture is an event-based control plane for OpenFlow. In HyperFlow, the authors exploit local controllers, serving all requests for their own remote sites, due to an increase in the flow setup times and flow initiation rates. It is actually implemented as a NOX [23] application that is responsible for: (a) global network view synchronization between controllers, (b) communication to switches controlled by another controller from a different site, and (c) managing responses coming from switches in other sites to the request-originator controllers. A system called "publish/subscribe" message paradigm is exploited to accomplish these tasks through controllers from different sites.

In [77], the authors propose a new distributed network platform called "Onix" for large-scale networks in response to deficiencies (e.g. providing consistent network state distribution, global network view among network applications, and failure recovery mechanisms) in a common control platform. Onix instances propagate network states to other instances to be able to scale large networks. The authors follow three approaches to improve scalability in Onix architecture; (1) Network Information Base (NIB)[3] partitioning by controller instances for less work, (2) cluster aggregation for a hierarchical structure, and (3) consistency and durability of the network states for applications. A similar work, Software Transactional Networking (STN) [87], also proposes a distributed control plane along with a scheme with a middleware to resolve policy consistency among distributed controller over the data plane. While Onix expects application writers to provide the necessary logic to detect and resolve conflicts of network state due to concurrent control, STN propose concurrent policy composition mechanisms that can be used by any application in a general fashion.

Tam et al. [81] study the feasibility of using multiple controllers to improve scalability without global network view and limited network topology information stored in controllers in a data center environment. They leverage flow routing example to see practicability of these controllers and propose two approaches, *path-partition* and *partition-path*, for the corresponding purpose.

In [82], the authors propose a distributed cluster-based controller architecture and a framework to retain the communication and coordination between controllers to obtain a more scalable network. This cluster-based architecture brings flexibility to the network regarding adding or removing controllers since it does not involve network applications. The controllers select a master controller that is in charge of delineation between controllers and switches.

Distributed controller architectures are proposed to mitigate the scalability issues of SDN networks. However, distributed controller architecture may not achieve the planned scalability because of the unbalanced load across the controllers since network

[3]NIB is a data structure to store network state and is roughly analogous to the Routing Information Base (RIB) used by IP routers.

administrators decide which and how many switches connect to a controller when they setup the network. Therefore, this may cause an overload in the controller.

ElastiCon [68] distributes the workload evenly through the controllers by means of a controller pool. This elastic distributed controller architecture dynamically shifts the workload across the controllers by adding or removing controllers to the controller pool and/or rebalancing the load of an individual controller based on threshold values.

Phemius et al. [66] present the "DISCO" (DIstributed Sdn COntrol plane) framework consisting of multiple controllers controlling different SDN domains that share aggregated network-wide information for a consistent network view on each controller. The DISCO framework has two main parts. While the intra-domain part is responsible for controller's own domain functionalities, the inter-domain part manages the flows across the distributed networks by exchanging the aggregated network state information such as reservation, topology etc. The difference of the DISCO framework from the other distributed architectures is its capability of differentiation of intra-domain and inter-domain information along with heterogeneous inter-domain links such as MPLS tunnels and SATCOM links.

Bari et al. [83] address difficulties of deploying multiple distributed controllers in a large-scale WAN network. They present a framework that readjusts the required active controllers with some assigned switches in accordance with current network dynamics to reduce flow setup time, horizontal overhead (between controllers) and vertical overhead (between controllers and switches). Their proposed management framework is responsible for (re)assignment of switches to controllers in case of a need.

ONOS [78] is another distributed SDN control platform aimed at improving scalability, performance and availability of networks. ONOS addresses how a network OS can scale horizontally to avoid becoming a performance bottleneck and avoid being a single point of failure. In ONOS, a large-scale WAN network can be divided into multiple parts controlled by different ONOS instances. These distributed ONOS instances construct a global network view for the network.

It is worth to mention another collaborative, open-source controller platform, the "OpenDaylight" (ODL) project [88]. The ODL is a Linux Foundation collaborative project to promote use of SDN. The ODL community has come together to establish an open reference controller framework to freely program and control an SDN architecture.

2.6.2.2 Hierarchical Controller Designs

In hierarchical architectures [67, 71, 73, 74, 79, 84] local controllers handle local applications' requirements with frequent events, and a main more powerful controller, usually called as "Root", deals with non-local applications' needs requiring global network view and rare events as opposed to local controllers. Although controllers may have a global view of the whole network in the distributed (flat) controller designs, lower-tier controllers (which are more localized compared to upper-tier controllers) do not maintain a global view of the network in the hierarchical controller designs. Therefore, this design is different from the distributed (flat) design regarding network views of the controllers.

Kandoo [73] focuses on scaling a controller by decreasing the number of frequent events on the control plane since these events bring more overhead than others to the control plane. Kandoo's architecture comprises of two layers to sustain scalability. The bottom layer consists of local controllers which are not connected to each other and do not maintain a network wide state while the top layer is a logically centralized controller, connected to all bottom layer controllers, with the global network view. Frequent and resource-greedy events like flow arrivals are processed by the local controllers at the bottom layer, thereby preventing the root (top layer) controller from coping with more numbers of events.

McCauley et al. [74] discuss "Logical xBar" that is a recursive building block used to construct a centralized abstract hierarchical control plane. It exploits the idea of aggregating smaller units for forwarding into larger ones. The proposed control plane

design has two building blocks: 1) Logical xBar, which is a programmable entity that can switch packets between ports, and 2) Logical Server which handles the forwarding table management and the control plane computations. In the proposed design, the network itself does not necessarily need to be physically hierarchical, instead aggregation of logical xBars and logical Servers bring that abstracted hierarchy on the network.

Flat and hierarchical control plane structures may still suffer from certain issues. In flat control plane architecture, the controllers may face increasing computational complexity resulting from growing large size networks. On the other hand, the centralized hierarchical architectures suffer from path stretch problems [89].

In [79], the authors propose the "Orion", a hierarchical control plane for large-scale networks managed by the same administrator to alleviate the above-mentioned two problems. Orion has three layers: the bottom layer consists of network devices of areas; the middle layer consists of area controllers; and the top layer contains sub-domain controllers. Sub-domain controllers have global network views for their own domains and synchronize this information with each other by a distributed protocol.

In [67], the authors introduce Decentralize-SDN, D-SDN, framework that distributes a control plane not only physically but also logically in a SDN. D-SDN exploits the hierarchy of controllers in which main controllers (upper layer) delegate control to secondary controllers (bottom layer) to manage certain network devices.

Marconett and Yoo [71] propose the "FlowBroker" architecture for a better collaboration between multiple domains in terms of load balancing and network performance. The FlowBroker architecture exploits the idea of hierarchy with domain controllers and one or more super-controllers, called as Brokers, atop. Each domain controller may attach to more than one Broker according to their reputations that reflect performance of a Broker regarding load balancing and reliability. The FlowBroker architecture allows Brokers to cooperate between them to share abstracted network states coming from the domain controllers below level. They report that as the domain count increases from 6 to 10, the difference between utilizing 1 broker or

5 broker agents equals a 5 to 8% decrease in maximum link utilization, a 28 to 84% reduction in end-to-end delay, and 69 to 151% reduction in traffic loss.

Karakus and Durresi [84] propose a hierarchy-based network architecture along with an inter-AS routing approach with QoS. The authors use an idea of levels in which networks with controllers reside on top. There is also a main controller that works like a broker on top of networks to keep the global network state and view. Their experiment results indicate that a network controller in a hierarchic setting handles 50% less number of traffic than a network controller in a non-hierarchic environment.

2.6.2.3 Hybrid Approach

This approach differs from the distributed (flat) and hierarchical designs in a way that data plane devices are involved in decision processing and network control. Therefore, this approach is considered hybrid due to involvement of both the control plane and data plane in the network control. This subsection presents several SDN architectures [64,65,70] that leverage the data plane by devolving some limited control functions (such as sending rules to other network devices to be added, deleted or updated in their flow tables etc.) to network devices for control plane workload partitioning, thereby improving scalability. This might happen either by installing rules proactively or reactively in the switches. Also, it is obvious that keeping flows as much as possible in the data plane reduces overhead and improves the controller performance regarding throughput and latency.

“DIFANE (DIstributed Flow Architecture for Networked Enterprises)” [65] is an architecture that preserves traffic in the data plane through managing packets in switches called ”Authority Switches”. In DIFANE, the authority switches are assigned rules by means of the controller that maintains an algorithm to partition the rules and minimizes rule fragmentation along with the authority switches.

“DevoFlow (Devolved Flow)” [64] addresses frequent interactions between the control plane and the data plane for the sake of full control and global view over

the network. Since this redundant interaction on almost every flow setup brings extra overhead and delay, the authors propose DevoFlow to reduce the interaction while preserving the required amount of visibility by conveying some functionalities of the control plane to the data plane. More efficiency and scalability are achieved because the controller controls only significant, and long-lived flows such as elephant flows. Use of wild-card rules which aggregate multiple rules into one minimizes the controller-switch communication as well. DevoFlow lets the switches make local decisions through cloning rules, multi-path support, and re-routing. However, there are some issues that remain open in DevoFlow such as how to manage some network applications including QoS, security, and traffic engineering.

Fibbing [70, 90] is another hybrid SDN architecture that applies a central control over traditional distributed link-state protocols such as OSPF and IS-IS. In Fibbing architecture, the controller is still centralized and responsible for path computation based on requirements from operators as in SDN case. However, the actual computation of Forwarding Base Information (FIB) entries and their installation on data plane devices is done by the distributed control plane of traditional protocols run on the network. In this way, Fibbing takes advantages of centralized control (SDN) and distributed traditional protocols for scalability.

2.7 Mechanisms-related Approaches

Section 2.6 has discussed topology-related approaches. This section discusses other mechanisms used to optimize controller(s). Mechanisms-related approaches primarily exploit various optimization techniques in order to alleviate the foregoing scalability issues in SDN networks. They aim to empower the controller performance so that it can handle more packet flows per second (i.e. throughput), improve the latency, and reduce overhead. One way to increase throughput and improve latency is to exploit the parallelism in multi-core systems by means of some methods such as multi-threading, I/O batching etc. while another way is reducing the events processed

in the control plane. These events mostly result from routing decisions made by the controller(s). Some research efforts propose better optimized routing decision mechanisms to reduce events to be processed in the control plane.

2.7.1 Parallelism-based Optimization

Parallelism, such as multi-threading, I/O batching and so on, is an optimization technique to improve I/O performance, reduce overhead and memory consumption of the controllers [25, 51, 75, 76]. These help increase controler's performance and therefore improve the scalability of the control plane.

Maestro [75] uses a multi-core architecture to leverage the parallelism in order to increase controller speed along with a hassle-free programming model for application writers. Maestro uses the batching of packets to individual destinations to improve processing and communication efficiency besides multi-threading structure. It is designed to evenly partition the workload in cores to increase the performance (i.e throughput) by keeping all processor cores busy by means of the "pull" fashion instead of the "push" fashion. It is pointed out that Maestro can achieve 600K requests/sec which implies that a distributed architecture of Maestro is needed to meet today's data center requirements.

McNettle [76] exploits multi-core opportunities of the Glasgow Haskell Compiler (GHC)[4] [92] and the run-time system. A certain number of CPU cores supports the McNettle system to scale up and the control algorithms requiring a global network state of flow arrival rates. In McNettle, when a packet cannot be associated with a flow rule, a packet-miss message is sent by a corresponding switch to invoke the packet-miss function included in message handlers forming McNettle programs. The authors claim that McNettle may scale up to $5K$ switches with 46 cores over a single controller with up to 13M flows/sec.

[4]GHC is an open source compiler for Haskell [91] (a functional programming language).

NOX-MT [51], the successor of NOX, is also a multi-thread controller which surpasses its predecessor (NOX) regarding throughput and response time. It embodies the fact that performance of a controller can be improved to certain levels by exploiting some optimization techniques such as multi-threading, I/O batching, malloc implementations etc. The authors leverage a performance measurement benchmark, Cbench [93], to emulate the switches and compare results of three different controllers, Beacon, Maestro and NOX, with NOX-MT regarding controller responsiveness, throughput performance and controller latency. The NOX-MT outperforms the other controllers by handling 1.8M flow requests/sec with an average response time of 2 ms.

Erickson [25] reveals "Beacon" that provides an easy-to-handle environment for programmers, extra abilities to manage applications, and better performance. One incentive design decision behind the Beacon is to enable network operators/administrators in order to manage (adding and/or removing) applications while running the Beacon. The Beacon is reinforced for a high performance by multi-threaded designs: "Shared Queue" and "Run-To-Completion". In "Shared Queue" design, the pipeline threads take the messages from the shared queue in order to process by corresponding applications. In case of the "Run-To-Completion" design, on the other hand, there are no pipeline threads and each message is processed by I/O threads. The evaluation results show that the Beacon outperforms some other controllers such as Maestro [75], NOX etc. by responding 1.35M messages/sec with a single thread. It also scales linearly with 12 threads by responding more that 12.8M messages/sec.

2.7.2 Control Plane Routing Scheme-based Optimization

Reducing the processed events resulting from routing decisions of the controller(s) is another way to increase the scalability and performance of the control plane in an SDN architecture. In [85,86,94] the authors aim for a better and less event-producing

routing schemes managed directly by controller(s) in order to scale up the OpenFlow-based networks.

Gao et al. [94] leverage a Dynamically Reconfigurable Processor (DRP) to increase the scalability of the controller. The authors exploit an emulated network-on-chip, called "diorama network", to perform routing. In the diorama network, they send emulated packets from source nodes to destination nodes through the network in order to figure out the shortest path. Their study is motivated by the fact that since routing by controllers affects the performance of controllers, slow routing decisions will increase the response time of controllers to switches in the data plane. An issue that is not investigated by the authors is how the proposed design copes with link failures in the network.

Source routing and its variations are also utilized to increase controller scalability and performance in SDN [85, 86]. The underlying motivation behind these studies relies on reducing the state distributed by the controller to data plane devices. This state distribution on each switch on a path takes a long time and pushes the controller response time. Hence, it increases the delay and network convergence time. It exploits the idea of inserting path information in packet headers so that each node can acquire the next node information where the packets are to be sent without communicating with the controller. This approach is different from the traditional OpenFlow hop-by-hop routing model in which each node communicates to the controller to learn what to do and where to send the flows.

QuagFlow [95] and RouteFlow [96] (evaluation of the QuagFlow) are some other projects that aim at certain objectives: (1) utilization of cheap network devices with minimal embedded software, (2) enabling use of legacy IP routing protocols, OSPF, RIP, BGP etc., without re-writing in a centralized way, and (3) ensuring interoperability with legacy network devices. They provide a transparent unification of the Quagga routing software suite [97] and OpenFlow-enabled hardware. They run control logic of underlying OpenFlow switches through a virtual network composed by virtual machines (VMs), which execute a routing engine (e.g. Quagga). These

VMs are connected to each other to represent the physical topology. The virtual environment is kept in external servers communicating with a controller application. Decisions made by the legacy IP protocols are converted to flow rules by the controller application and installed to switch flow tables by the controller. Therefore, there is no requirement for modification of the existing routing protocols.

Scalability in carrier-grade networks also requires attentions from researchers due to some reasons such as number of and geographical distances between devices. Hartert et al. [63] propose a solution framework, DEFO (Declarative and Expressive Forwarding Optimizer), to achieve high scalability as well as robustness at carrier-grade networks. Their solution is based on two logical layers: connectivity layer and optimization layer. While the connectivity layer is responsible for default forwarding behavior and defines connectivity paths, the optimization layer defines exceptions to this default forwarding behavior and implements optimized paths, which are overwritten connectivity paths and computed by stitching connectivity paths together.

2.8 Comparison of Control Plane Scalability Proposals

Controllers are the main entities in decision-making processes in SDN networks. They perform crucial tasks affecting performance of the whole network. Currently, there exist more than 35 different publicly-available and proprietary SDN OpenFlow controllers created by different research groups, vendors, and organizations from both academia and industry, written in different languages, and having different performances. This rapid growing of controllers has raised questions regarding performance benchmarking of these controllers. Some research efforts [59, 98] have been proposed to evaluate performances of the controllers with respect to certain metrics. In [59], the authors present a limited analysis of controllers' performances by using a new benchmarking framework called "hcprobe". Similarly, Jarschel et al. [98] also introduce a tool called "OFCBenchmark" to benchmark OpenFlow controllers. As stated earlier, the performance of an SDN controller is characterized by several metrics,

Table 2.2.: Certain scalability related metrics such as control plane throughput in terms of the number of flows handled and flow setup latency by control plane from the studies.

Metrics / **Proposals**	**Throughput (Flows/sec)**	**Flow Setup Latency**
Beacon [25]	up to 12.8M	avg 24.7 μs
DevoFlow [64]	-	-
DIFANE [65]	up to 3M	min 0.4 ms
DISCO [66]	-	-
D-SDN [67]	-	-
ElastiCon [68]	up to 30K	min 1 ms
Ethane [69]	up to 11K	min 1.5 ms
Fibbing [70]	-	min 0.89 ms
FlowBroker [71]	-	-
HyperFlow [72]	-	-
Kandoo [73]	up to 1.3M	-
Logical xBar [74]	-	-
Maestro [75]	up to 3.5M	avg 55 ms
McNettle [76]	up to 13M	max 10 ms
NOX [23]	up to 30K	avg 49 ms
NOX-MT [51]	up to 1.8M	avg 2ms
Onix [77]	up to 200K	min 2 ms
ONOS [78]	up to 19K	avg 34 ms
Orion [79]	up to 50K	min 11 ms
Tam et al. [81]	-	-
Yazici et al. [82]	up to 36K	-
Bari et al. [83]	-	min 5 ms
Karakus et al. [84]	-	-
Owens et al. [85]	-	-
Soliman et al. [86]	-	-

but, throughput and flow setup latency latency are the most considered ones by the SDN research community. In terms of the control plane scalability, the *throughput* metric typically represents the number of flows that a control plane (i.e. controller) can handle in certain amount of time while the *flow setup latency* denotes the time elapsed from arrival of a "packet_in" message from a switch to installation of the corresponding flow rule in the switch flow table.

Table 2.2 shows performance related results of studies with respect to some scalability related metrics such as throughput and flow setup latency. Since some studies

evaluate performance of their systems regarding different metrics such as path installation time [81], ratio of elephant to mouse flows [73], link utilization [71] and so on, it is difficult to show all the metrics used in studies in a table. In addition, these numbers heavily depend on the evaluation environments. In other words, each study uses different network dynamics and parameters such as workload, network topology, number of controllers, applications for testing etc. during their experiments. Also, these controllers are designed for different problems. Therefore, it is highly recommended that readers individually examine the corresponding studies in order to rightly evaluate their scalability performances with respect to their characteristics.

Using different number of threads shows that single threaded controllers, such as Ethane and NOX, are very limited regarding the throughput because they cannot handle a large number of flows. However, the controllers that are multi-threaded, such as Beacon, Maestro, McNettle, and NOX-MT, can handle a large number of flows per second. The authors in [82] report that the average number of controller responses per second per switch when one, two, three, and four controllers are used are approximately $6K$, $12K$, $25K$, and $36K$, respectively. ElastiCon's throughput performance with respect to the number of controllers varies from $30K$ to $72K$, its response time performance for packet-in arrivals up to $2K$ packets/sec regarding 1-controller, 2-controllers, and 4-controllers cases varies from 1.1 ms to 13.8 ms, 1.0 ms to 4.3 ms, 1.0 ms to 2.2 ms, respectively. In Orion architecture, the total number of new flows that area controller(s) can handle per second varies from $8K$ to around $50K$ with respect to the number of area controllers. It is also reported that minimum average flow setup delay between areas is around 11 ms while maximum of which reaches to around 25 ms depending on the number of domain controllers, areas, and switches in an area. In [83], the authors state that their framework shows around 160 ms and 5 ms average flow setup time performance for 1-controller and n-controllers cases, respectively, on RF-I topology (79 nodes, 294 links) while it is 185 ms and 12 ms, respectively, on RF-II topology (108 nodes, 306 links). In ONOS, 45.2 ms and 34.1 ms latency values are reported for the time elapsed from a network event

detection to sending first corresponding `OFPT_FLOW_MOD` message for rerouting $1K$ flows and path installation, respectively. In DIFANE, packets experience 0.4 ms round-trip time at 100 single-packet flows/sec sending rate. While NOX-MT has an average response time of 2 ms, Beacon has the minimum average latency with 24.7 μs among the others.

Table 2.3.: Some features such as the controller that works with, used programming language and evaluation setup characteristics of the studies.

Features / **Proposals**	**Controller**	**Programming Language**	**Evaluation Setup**
Beacon [25]	Beacon	Java	Used Cbench for tests run on Amazon's Elastic Computer Cloud using a Cluster Compute Eight Extra Large instance containing 16 cores.
DEFO [63]	DEFO	Scala	Used many different real and realistic topologies with different number of nodes and links and compared it to Cisco MATE [99], a traffic engineering tool.
DevoFlow [64]	Any	Depends on controller used	Implemented a flow-level data center network simulator. Used a three-level Clos topology w/ 168 switches and a two-dimensional HyperX topology w/ 97 switches.
DIFANE [65]	Any	Depends on controller used	Used XORP [100] to run the link-state routing protocol and kernel-level Click-based OpenFlow switches as a authority switches.
DISCO [66]	Any	DISCO	Used Floodlight [101] controllers and Mininet [102] SDN simulator to create 3 SDN WANs w/ 4 switches each and connected to each other.
DRP [94]	Any	Depends on controller used	Constructed an emulated network (w/ 6 routers and 10 links) on a commercially available dynamically reconfigurable processor DAPDNA-2.
ElastiCon [68]	Any	Java	Used modified Floodlight controller, k=4 fat tree topology and a modified version on Mininet to run the Open vSwitch [103] instances on different hosts.
Ethane [69]	Ethane Controller	C++/Python	Deployed at Stanford's Computer Science department for over 4 months and managed over 19 switches and 300 hosts.

Table 2.3 continued

Features / Proposals	Controller	Programming Language	Evaluation Setup
Fibbing [70]	Fibbing Controller	Python/C	Used ISP topologies [104] whose sizes range from 80 nodes to over 300. All measurements were performed using OSPF on a Cisco ASR9K router equipped with 12GB of DRAM as well as on a Juniper M120 router equipped with 2GB of DRAM.
FlowBroker [71]	Any	Java	Used Floodlight controller and Mininet tool to test 5 different scenarios.
HyperFlow [72]	NOX	C++	Used 10 servers each equipped with a gigabit NIC and running as a storage node.
Kandoo [73]	Kandoo	C/C++/Python	Used a simple tree topology where each switch is controlled by one local controller and Kandoo root controller atop in modified version of Mininet and Open vSwitch.
Maestro [75]	Maestro	Java	Implemented an emulator to generate flow requests from hosts on a common 79-switch topology going to Maestro controller.
McNettle [76]	McNettle	Haskell	Used a modified version of Cbench and ran the controller on a DELL Poweredge R815 server with 48 cores.
NOX [23]	NOX	C++/Python	Ran it in their internal network of roughly 30 hosts for over 6 months.
NOX-MT [51]	NOX-MT	C++/Python	Used Cbench representing 100K hosts and 32 emulated switches.
Onix [77]	Onix	C++	Evaluated Onix in two ways: with micro-benchmarks to test Onix's performance as a general platform, and with end-to-end performance measurements of an in-development Onix application in a test environment.
ONOS [78]	Any	Java	Used Floodlight controller and connected a 6-node ONOS cluster to an emulated Mininet network of 206 software switches and 416 links. Also demonstrated in Internet2 [105] topology.
Orion [79]	Any	Java	Used Floodlight controller as area controllers. Conducted different experiments for different number of domain (from 1 to 2) and area controllers (from 1 to 6) and switches (from 20 to 120).
Tam et al. [81]	Any	Depends on controller used	Used 4 controllers on topology of an irregular network with 28 nodes and 66 links.
Yazici et al. [82]	Any	Java	Used Beacon controllers for the experimental setup with 4 controllers and 4 emulated switches to run Cbench instances.

Table 2.3 continued

Features / Proposals	Controller	Programming Language	Evaluation Setup
Bari et al. [83]	Any	Python	Used POX [106] controller and Mininet to simulate RF-I (79 nodes, 294 links) and RF-II (108 nodes, 306 links) ISP topologies.
Karakus et al. [84]	Any	Depends on controller used	Used a topology with 4 different autonomous domains with 4 switches each and a Broker conntected to domain controllers.
Owens et al. [85]	VSDN Controller	C/C++	Used NS-3 [107] tool to simulate a 6-node network with increasing connection requests for the controller.
Soliman et al. [86]	Any	Depends on controller used	Used Internet2 OS3E topology with 34-nodes.

Table 2.3 illustrates some features, such as the controller that works with, used programming language in controller implementation and evaluation setup characteristics, of the studies. Most of the proposals work with any SDN controller with some modification efforts. However, the Floodlight [101] controller is the most used one in the evaluation phase of the studies due to its good documentation, active community support, and integration with REST API. Also, Java is the prevalent programming language used in implementation of the studies due to their controller choice although some studies do not report which programming language they used.

Table 2.4 illustrates the approaches used by the studies to achieve control plane scalability. Topology-related approaches uses single, distributed (flat) or hierarchical controller designs. Mechanisms-related approaches exploit multi-threading, I/O batching, better routing schemes etc. There are also hybrid (i.e. both the control plane-centric and data plane-centric) studies. Some research efforts belong to more than one approach because they more or less exploit some other approaches too. However, they have been classified based on their main methods, which are discussed in corresponding sections.

Controller designers may consider two architectural design goals while designing their controllers to improve scalability performance: (1) they can utilize static *switch partitioning*—distribution and allocation of connected network devices to worker threads running in the controller—and *packet batching*—where multiple bytes are

read from or written to the underlying network using a socket buffer—techniques to achieve high throughput and (2) workload adaptive *packet batching* and *task batching*—a strategy used to allocate already received packets to the worker threads for processing, hence directly impacting the latency of the controller—to reduce flow setup latency.

Table 2.4.: Approaches used to achieve control plane scalability. Topology-related approaches utilizes central (single), distributed (flat), hierarchical controller and hybrid designs. Mechanisms-related approaches exploit multi-threading, I/O batching, better routing schemes etc. Some research efforts belong to more than one approach because they exploit some other approaches in their designs too. However, they have been classified based on their primary approaches, which are discussed in the corresponding (sub)sections.

Approaches / **Proposals**	**Topology-related Approaches**				**Mechanisms-related Approaches**	
	Centralized (Single) Controller Designs	**Distributed Approaches**			**Parallelism-based Optimization**	**Control Plane Routing-based Optimization**
		Distributed (Flat) Controller Designs	**Hierarchical Controller Designs**	**Hybrid Designs**		
Beacon [25]					✓	
DEFO [63]		✓				✓
DevoFlow [64]				✓	✓	
DIFANE [65]				✓		
DISCO [66]		✓				
DRP [94]					✓	✓
D-SDN [67]			✓			
ElastiCon [68]		✓				
Ethane [69]	✓					
Fibbing [70]	✓	✓		✓		
Marconett [71]			✓			
HyperFlow [72]		✓			✓	

Table 2.4 continued

Approaches / Proposals	Topology-related Approaches				Mechanisms-related Approaches	
	Centralized (Single) Controller Designs	Distributed Approaches			Parallelism-based Optimization	Control Plane Routing-based Optimization
		Distributed (Flat) Controller Designs	Hierarchical Controller Designs	Hybrid Designs		
Kandoo [73]			✓			
McCauley [74]			✓			
Maestro [75]					✓	
McNettle [76]					✓	
NOX [23]	✓				✓	
NOX-MT [51]	✓				✓	
Onix [77]		✓				
ONOS [78]		✓				
Orion [79]			✓			
Tavakoli [80]	✓					
Tam et al. [81]		✓			✓	
Yazici [82]		✓			✓	
Bari et al. [83]		✓				
Karakus [84]			✓			✓
Owens [85]						✓
Soliman [86]						✓

2.9 Challenges and Existing Proposals in SDN Control Plane

While SDN is becoming a mature technology, the control plane scalability issues deserve more research efforts from both academia and industry. This section discusses the general problems in an SDN control plane. However, each of these problems affects the scalability of the control plane in SDN. Therefore, these problems need to be taken care of by network operators while designing/operating their SDN networks. In the following, the main SDN control plane challenges along with existing proposals is stated.

- Controller(s) Failure: In a traditional network, when one or more network nodes fail, flows are routed through alternative paths/nodes to maintain the traffic continuity. However, in an SDN architecture, failure of the controller(s) may

result in a chaos for the specific part(s) of the network controlled by the failed controller(s) due to two main critical reasons: (i) The controllers are responsible for all configurations, operations, and validations of the network topologies, resources etc. and (ii) data plane devices lack an ability for an online "detour" of flows. This problem may become worse in the single controller design case. In addition, distributing the load of a failed controller to other controllers brings extra load on them, which reduces performances thereby their scalability. This distribution may even result in a cascading failures of controllers because it can exceed the capacity of them.

One way to address this problem is to enhance the network with backup/standby controllers [108, 109]. In case of the main controller failure, these backup controller(s) may take the responsibility of the network operations over from the main controller. In this case, controllers need to be synchronized to be in a consistent status regarding network states.

In [110], the authors present a disaster-aware control plane design to reduce controller-related interruptions. They model the problem of designing a disaster-resilient control plane problem regarding the number of controllers, their placement, and the control plane topology. Pashkov et al. [111] propose a fault-tolerant control plane design, High-Available Controller (HAC) architecture, to address the fast recovery of the control plane by adding an additional cluster middleware between the controller core and controller network services and applications.

- State/Policy Distribution/Consistency: Another important problem regarding scalability is the network state distribution and consistency between controllers of a control plane. This problem mainly happens in the distributed and/or hierarchical architectures due to distribution of network states among controller replicas. In addition, this distribution needs to be fast and reliable to provide the consistency between controller instances [112]. Moreover, policy consistency [87]

in a distributed control plane is required because network-wide policies do not come from a single component of a network, but rather, they are formed by different functional modules such as routing, monitoring, and access control as well as multiple human operators controlling different parts of the network. These conflicts may result in serious inconsistencies such as violation to another policy and wrong forwarding of the packet etc. on the data plane. Therefore, more efficient algorithms and mechanisms are needed to maintain state/policy consistency among the distributed controllers.

Distributing network state among local controllers in the same domain does not necessarily deal with security issues. However, the Internet consists of many networks managed by different authorities. Therefore, the logically centralized control model of SDN must be extended to account for inter-domain traffic. This extension requires peering, thereby state sharing, among different administrative domains to have a relative global network view in order to determine the next hop. However, this distribution has to be secure, private, and consistent. In addition, some other critical questions regarding this sharing are how and what to exchange with other domains. Yin et al. [113] state that the types of messages exchanged among controllers may be various such as reachability information, flow setup/tear-down/update requests, network parameters (bandwidth, delay, loss etc.), service-level agreements (SLAs), virtual network information and so on. In [39,114,115], the authors propose a West-East (WE) Bridge mechanism to enable different SDN administrative domains to securely peer and cooperate with each other.

- Flow Rule Setup Latency: This problem refers to the delay in new flow rule setup process in the context of control plane scalability [54]. As explained earlier, proactive mode and reactive mode are two prominent modes to setup a new flow rule. The proactive mode mode does not impose any latency in the flow rule setup from the controller's point of view. In the reactive mode,

the controller response time (i.e. delay) is crucial. Controllers having longer flow rule setup latencies may not meet requirements of certain applications such as fast fail-over and reactive routing of latency-sensitive flows. Therefore, such control planes cannot be scalable enough to satisfy the network needs. However, this delay can be relatively reduced by imposing more controller and switch resources such as CPU, memory etc. and devolving some control functions to the switches.

In [116], the authors conduct various setup experiments to test the latency of various controllers by changing the number of switches, number of threads, and controller workload. They conclude that adding more threads beyond the number of switches does not improve latency, and serving more switches than available CPUs increases controller response time.

Some studies [64,65] mitigate the flow rule setup latency by leveraging the idea of "Control Function Devolvement". This idea relies on the delegation of some of the control functions to the data plane so as to alleviate the load on the controller(s), thereby reducing controller-switch communication frequency.

- Controller Placement: In addition the number of controllers, placement of the controller(s) [61] has impacts on performance of the network as well. Suboptimal controller placement affects many other problems such as flow rule setup latency due to controller-switch communication delay, controller-controller communication delay, control plane overhead, fault tolerance, resiliency and so on. Although there are some studies addressing this problem in the literature [117–124], it is still an ongoing issue and needs further attention of researchers.

  Hu et al. [117] propose algorithms to automate the controller placement decisions given a physical network and the number of controllers. The main objective of these algorithms is to maximize resiliency of SDN to failures. In [118,119],

the authors address the controller placement problem to maximize the reliability of control networks.

Rath et al. [120] present a non-zero-sum game-based distributed technique to discuss optimal controller placement in SDN. Hock et al. [121] introduce the Pareto-based Optimal COntroller-placement (POCO) framework, which brings network operators a range of options to select the controller placement based on their particular needs regarding the metrics like latency, load balancing etc. In [122], the authors focus on the controller placement problem for WAN. They use the Spectral Clustering placement algorithm to partition a large network into several small SDN domains. Jimenez et al. [123] utilize the algorithm called "k-Critical" to discover the minimum number of controllers and their locations to create a robust control topology that deals with failures and balances the load among the selected controllers.

Furthermore, control plane overhead is affected by the placement of controllers due to traffic between switches and controllers (*packet_in* and *flow_mod* messages) and among controllers (e.g. state sharing). Obadia et al. [124] challenge the problem of minimizing control overhead by optimizing the number of controllers and their placement. This approach differs from others because they target minimization of control overhead instead of minimization of switch-controller delay.

2.10 Chapter Summary

SDN is a promising emerging architecture for many networking environments such as data centers, enterprise networks, campus networks, cloud networks, and WAN. The major advantages of SDN are its programmability and agility. However, the scalability issues in the control plane is one major problem in SDN that needs more research attention. This chapter has firstly given an overview of the SDN architecture and OpenFlow protocol along with its support mechanisms for scalability. It has dis-

cussed the scalability as a concept in general and presented various metrics proposed for quantification of scalability. There is no consensus on the definition of scalability. In other words, while the basic notion is intuitive, scalability does not evoke the same concept to everybody. In the context of SDN, scalability is characterized by the two prominent metrics, throughput and flow setup latency. Also, the study has pointed out the main reasons that make the control plane a scalability bottleneck in SDN: Separation of Control Plane and Data Plane, Quantity of Events/Requests Handled by a Controller, and Controller-Switch Communication Delay. Furthermore, it has presented the organization for taxonomy of scalability-centric studies in two broad approaches: Topology-related approaches and Mechanisms-related approaches. While the former reviews the relation between topology of architectures and scalability issues, the latter discusses the relation between various mechanisms used to optimize controllers and scalability issues. Finally, the chapter has outlined the potential challenges and open problems that need to be addressed further for more scalable SDN control planes: Controller(s) Failure, State/Policy Distribution/Consistency, Flow Rule Setup Latency, and Controller Placement.

3 QUALITY OF SERVICE (QOS) IN SOFTWARE DEFINED NETWORKING (SDN): A SURVEY

3.1 Abstract

Supporting end-to-end Quality of Service (QoS) in existing network architectures is an ongoing problem. Although researchers from both academia and industry have proposed many solutions to solve the QoS limitations of the current networking, many of them either failed or were not implemented. SDN paradigm has emerged in response to limitations of traditional networking architectures. Its main advantages are the centralized global network view, programmability, and separation of the data plane and control plane. These features have got attention of researchers to improve the QoS provisioning of today's various network applications. This survey chapter aims at making a picture of QoS-motivated literature in OpenFlow-enabled SDN networks by comprehensively surveying relevant research studies. It organizes the related studies according to the categories that are the most prominent ways in which QoS can benefit from the concept of SDN: Multimedia flows routing mechanisms, inter-domain routing mechanisms, resource reservation mechanisms, queue management and scheduling mechanisms, Quality of Experience (QoE)-aware mechanisms, network monitoring mechanisms, and other QoS-centric mechanisms such as virtualization-based QoS provisioning and QoS policy management etc. In addition, this study discusses QoS capabilities of OpenFlow protocol by reviewing its versions along with some well-known, open-source, and community-driven controller projects. Furthermore, it outlines the potential challenges and open problems that need to be addressed further for better and complete QoS abilities in SDN/OpenFlow networks and lessons learned during preparation of this survey study.

3.2 Introduction

With the growth of the Internet, new types of networking applications and services (e.g. web surfing, texting, VoIP, email, audio, video conferencing and streaming, online gaming, e-commerce etc.) have emerged for end users. These applications and services generate their own characteristic flows which need to be delivered by the Internet. However, all of these applications require different treatments for their own flows to make the delivery successful over a network [125]. For example, some applications such as video conferencing require a certain bandwidth for its flows while applications like VoIP are more sensitive to the delay over a network [126]. Addressing these requirements needs a well-defined Quality of Service (QoS) mechanism(s) in a network. However, today's de facto delivery model, best-effort, in the Internet is not capable of serving to all of the aforementioned services. In addition, proposed QoS solutions have not been successful enough to solve the QoS issues of the traditional networking paradigms.

The Internet Engineering Task Force (IETF) has defined various types of QoS architectures to support QoS provisioning. The Integrated Services (IntServ) model [127] is based on the per-flow concept. It utilizes the resource reservation protocol (RSVP) [128] to provide the QoS to end users. In IntServ model, resources are explicitly reserved through an end-to-end path and hence all routers store network states related to the service. Therefore, it suffers from the scalability and complexity issues. To mitigate that scalability issue, researchers have proposed the Differentiated Services (DiffServ) model [129], which is on flow-aggregation basis and exploits the hop-by-hop process. It classifies the incoming flows (using pre-configured classes) based on the Type of Service (ToS) field in the header of the packets. Since DiffServ treats packets in the same class identically, it is difficult to provide quantitative QoS to individual flows. It is strong on simplicity, but weak on guarantees. The Multiprotocol Label Switching (MPLS) [130–132] is another technology that is used to reduce the complex routing table lookups by labeling techniques. All of these

advantages and disadvantages show that the current QoS architectures are not truly successful at QoS support for service providers, enterprises and/or end users.

SDN is a new emerging architecture in recent years. SDN is described in Open Networking Foundation's [2] definitions as "In the SDN architecture, the control and data planes are decoupled, network intelligence and state are logically centralized, and the underlying network infrastructure is abstracted from the applications.". This separation provides network operators/administrators with efficient use of network resources and ease of resource provisioning. Also, SDN brings ease of programmability to change the characteristics of whole networks. This ability simplifies the management of the network since it is decoupled from the data plane. Therefore, network operators can easily and quickly manage, configure, and optimize network resources with dynamic, automated and proprietary-free programs written by themselves in SDN architecture.

In addition, since the SDN is logically centralized, controllers have a global visibility of the whole network unlike conventional networking. Hence, they can dynamically optimize flow-management and resources. Furthermore, per-flow or application-level QoS provisioning becomes easier and feasible for network administrators. For these reasons, SDN is drawing attention of companies, universities, data centers, and service providers to be deployed in their networks. Google's private WAN (B4 [8]), connecting Google data centers across various geographical location over the world, is one of the examples for SDN adoption in a large-scale network with the aforementioned purposes.

3.2.1 Chapter Organization

This survey chapter aims at making a picture of QoS-motivated literature in OpenFlow-enabled SDN networks by surveying relevant research studies. The scope of this work revolves around the term *QoS* characterized by network characteristics such as bandwidth, delay, jitter, and loss along with industry-wide set of standards

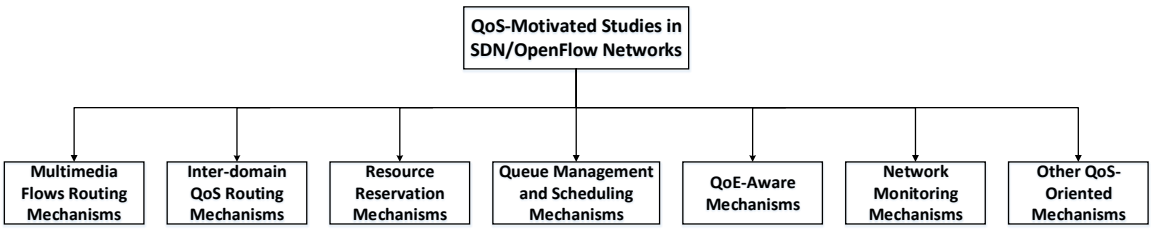


Figure 3.1.: The Organization of QoS-based studies in SDN/OpenFlow networks. The first two types of mechanisms are driven by the routing functionality. The third and fourth types of mechanisms are concentrated around resource reservation and queue management and packet scheduling for QoS support. The fifth type of the studies address the QoE of the system while the sixth group of the studies revolve around network monitoring frameworks. The last group of the mechanisms study miscellaneous QoS-related issues such as QoS policy management, QoS testbed extensions etc.

and mechanisms for ensuring high-quality performance for critical applications. Focus of studies presented in and the scope of this study are centered around aforementioned typical network characteristics.

As seen in Fig. 3.1, the study organizes the related studies into seven categories that are the most prominent ways in which QoS can benefit from the concept of SDN: Multimedia flows routing mechanisms, inter-domain routing mechanisms, resource reservation mechanisms, queue management and scheduling mechanisms, Quality of Experience (QoE)-aware mechanisms, network monitoring mechanisms, and other QoS-centric mechanisms such as virtualization-based QoS provisioning and QoS policy management etc. Each category itself in the organization reflects a problem/challenge for QoS in SDN. Therefore, the organization is indeed a taxonomy of the problems for QoS in SDN at the same time. These categories (i.e. problems) and related studies are explained (i.e. solutions) in corresponding sections. In addition, QoS capabilities of OpenFlow protocol is discussed by reviewing its versions along with some well-known, open-source, and community-driven control platform projects. Finally, it outlines the potential challenges and open problems that need to be addressed further for improved and complete QoS abilities in OpenFlow-enabled SDN networks.

This study gives an overview of the relations between QoS and SDN. This survey study may be a useful primer for a reader interested in studying QoS in/with SDN. After reading this survey chapter, the reader will be familiar with:

- A lightweight overview of the SDN Architecture
- QoS capabilities of specific OpenFlow protocol versions
- QoS support of some well-known, active, and open-source SDN controller projects
- QoS problems in SDN and related solutions from researchers
- Some other potential challenges and critical points for QoS in SDN requiring attention of research community

The rest of the chapter is organized as follows: Section 3.3 discusses the QoS capabilities of OpenFlow protocol in its different versions and that of (well-known) open-source SDN control platforms. Section 3.4 summarizes the role of SDN with relation to QoS. While Section 3.5 outlines multimedia flows-based routing mechanisms Section 3.6 presents inter-domain QoS routing frameworks. Section 3.7 introduces resource reservation based frameworks to provide QoS. Section 3.8 discusses frameworks focusing on queue management and packet scheduling. Section 3.9 states the QoE-oriented mechanisms. Section 3.10 presents network monitoring frameworks. Section 3.11 discusses miscellaneous QoS-related mechanisms. Section 3.12 outlines few potential challenges and open problems for QoS support in OpenFlow networks along with lessons learned while preparation of this survey. Finally, Section 3.13 wraps the chapter up with concluding remarks.

3.3 QoS Implementation in OpenFlow-Enabled SDN Networks

Although SDN and OpenFlow couple support some limited QoS capabilities it allows us to obtain per-flow QoS control in a more scalable, flexible and finer-granular way compared to the above traditional architecture. This section reviews the QoS

capabilities of OpenFlow protocol by looking at its different versions and that of (well-know) open-source SDN control platforms.

3.3.1 QoS in OpenFlow Protocol

Each OpenFlow specification version has brought some different features along with minor and major changes compared to their previous versions. In the following, the study highlights the QoS-related features and changes implemented in the different versions of OpenFlow specification.

OpenFlow 1.0—In OpenFlow 1.0 [133], there is an optional action called "enqueue", which has been renamed to "set_queue" in OpenFlow 1.1 and later versions, that forwards packet through a queue attached to a port. An OpenFlow-enabled switch can have one or more queues depending on its ports. An OpenFlow controller can query an information about queues of a switch. However, the behavior of the queue is determined outside the scope of OpenFlow, which can be configured through the OF-CONFIG protocol [134] but requires OpenFlow 1.2 and later versions. Also, header fields can include VLAN priority and IP ToS, so packets can be matched against rules and their associated header fields can be rewritten.

OpenFlow 1.1—OpenFlow 1.1 [41] performs matching and tagging of VLAN and MPLS labels and traffic classes. Prior versions of OpenFlow specification had limited VLAN support (only supported a single level of VLAN tagging with ambiguous semantic). The new tagging support has explicit actions to add, modify and remove VLAN tags, and can support multiple levels of VLAN tagging. This version also adds a similar support the MPLS shim headers.

OpenFlow 1.2—OpenFlow 1.2 [42] has added an ability that enables a controller to query all queues in a switch. It also has added experimenter queue property. Another QoS related improvement in this version is that it has added a max-rate queue property. In addition, this version specifies that queues can be attached to ports and be used to map flows on them.

OpenFlow 1.3—OpenFlow 1.3 [135] introduces the rate-limiting functionality by means of meter tables consisting of meter entries. A meter entry consists of “Meter Identifier”, “Meter Bands”, and “ Counters”. A Meter Band, in turn, consists of “Band Type” (e.g. drop or remark DSCP etc.), “Rate” (e.g. kb/s burst), “Counters”, and optional “Type specific arguments”, such as drop and DSCP remark, as seen in Fig. 3.2.

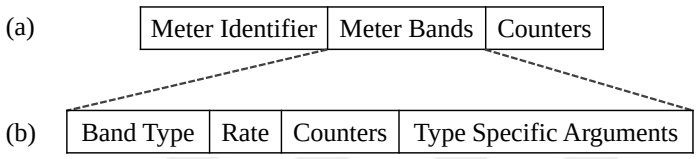


Figure 3.2.: Main components of a meter band (b) in a meter entry (a).

Counters may be maintained per-queue, per-meter, and per-meter band etc. They help controller collect statistics about the network. There may be one or more meter bands per meter table entry. Meters can be combined with the optional *set_queue* action, which associates a packet to a per-port queue in order to implement complex QoS frameworks such as DiffServ. These meters complement the queue framework already in place in OpenFlow by allowing for the rate-monitoring of traffic prior to output. More specifically, with meters, the ingress rate of traffic can be monitored as defined by a flow rule. Packets can be directed to a specific meter using the optional *meter*(meter_id) instruction, where the meter can then perform some operations based on the rate it receives packets.

OpenFlow 1.4—OpenFlow 1.4 [136] presents the flow monitoring framework that allows a controller to monitor the changes done by other controllers to any subsets of the flow tables in real time. To this end, a controller can define a number of monitors, each selecting a subset of the flow tables. Each monitor includes a table id and a match pattern that defines the subset monitored. When any flow entry is added, modified or removed in one of the subsets defined by a flow monitor, an event is sent to the controller to inform it about the change.

OpenFlow 1.5—OpenFlow 1.5 [137] replaces the "meter" instruction, which was used for metering in previous versions, with a *meter* action. As a result, multiple meters can be attached to a flow entry, and meters can be used in group buckets.

3.3.2 QoS in SDN Controllers

Since OpenFlow does not currently provide support for queue configuration in its specification, queue configuration is handled by specific OF-CONFIG and OVSDB (Open vSwitch Database Management Protocol) [138] protocols. The former is currently being standardized by ONF and the latter is already standardized by the IETF. Although OVSDB is already implemented in OVS switches, there is no available controllers providing a standardized management of queues. Currently, there are many different SDN controller platforms offering various features for users. Although there are many commercial and proprietary SDN controllers from different vendors, there also exist some collaborative and open-source projects with active development support from research community and industry. Below, the study discusses some of these active, open-source, and collaborative SDN controller projects with regards to their QoS support.

OpenDaylight—OpenDaylight (ODL) [88] is a community-led and open-source controller platform. It is a Linux Foundation collaborative project to promote use of SDN. The ODL community has come together to establish an open reference controller framework to freely program and control an SDN architecture. ODL project consists of many other sub-projects, such as southbound protocol plugins (e.g. OpenFlow, NetCONF, SNMP, and BGP) and applications (e.g. DDoS Protection and Virtualization Coordinator), complementing each other to compose a complete reference controller platform for heterogeneous networks. PCMM (PacketCable MultiMedia), presented in ODL-Lithium release in June 2015, plugin is another southbound plugin utilized to enable flow-based dynamic QoS for the DOCSIS infrastructure. Packet Cable MultiMedia (PCMM) provides an interface to control and management service

flow for CMTS network elements. Also, OVSDB southbound plugin has been introduced in ODL-Lithium release, which can manage and configure queues in switches. In addition, the Reservation module in ODL also aims at providing dynamic low-level resource reservations so that users can get network services, connectivity or a pool of resources (ports, bandwidth) for a specific period of time.

ONOS—ONOS (Open Network Operating System) [139] is a distributed SDN control platform aimed at improving scalability, performance and availability of networks for service providers. It is also an open-source platform with over 50 partners and collaborators that contribute to all aspects of the project. ONOS has limited QoS support currently. It supports OpenFlow metering mechanism, but this feature is rarely implemented in existing switches. The idea behind this support is based on implementation of OpenFlow *set_queue* functionality in ONOS. As another QoS support improvement attempt in ONOS, a new high-level instruction *SetQueueInstruction* has been implemented in org.onosproject.net.flow.instructions library and the corresponding references in ONOS libraries have been modified accordingly.

Floodlight—Floodlight [101] is a Java-based another open-source SDN controller that is supported by community developers including engineers from Big Switch Networks. There are community driven projects built on top of Floodlight proposing integrating/updating new/existing modules. QoS module [140] implemented for Floodlight controller aims at providing an application that does burden of matching, classification, flow insertion, flow deletion, and policy handling for QoS. The module utilizes OpenFlow 1.0 *enqueue* action and the network ToS bits. It controls tracking and storing services with their DSCP values, applying policies for services class, and tracking of policies in switches. The QueuePusher [141] extension utilizes OVSDB protocol integrated with northbound API of Floodlight to generate appropriate queue configuration messages. The QueuePusher module uses a CRUD (Create, Read, Update, Delete) API, exposed by Floodlight, that allows external entities to manage Open vSwitch.

3.4 Relationship between SDN and QoS

QoS is typically defined as an ability of a network to provide the required services for a selected network traffic. The primary goal of QoS is to provide priority with respect to QoS parameters including but not limited to:

- bandwidth
- delay
- jitter
- loss

characteristics. In order to provide QoS, differentiating application flows is needed since they battle for available network resources. These network resources have to be allocated to ensure the precedence of the higher-priority traffic for the appropriate network resource distribution. This process often requires knowledge of the current network states, so that the right decisions with regard to packet forwarding can be made.

Today, QoS provisioning mostly relies on Service Level Agreements (SLAs) between end users and service providers. This approach works well for best-effort service and does not support finer-granular traffic control. However, there are other types of applications, such as VoIP, online-gaming, and video conferencing, whose flows are sensitive to delay, jitter, and bandwidth, thereby requiring special handling. Also, "hop-by-hop" decision architecture of the Internet is sometimes difficult to monitor, mainly because of the many different vendor-specific firmwares at use. There is no standardized way for specifying high level traffic control policies and restrictions with regard to the depth of traffic differentiation exist.

QoS is mainly implemented in two approaches: hard QoS and soft QoS. The hard QoS method guarantees the QoS requirements of connections but it suffers from resource limitations. IntServ method is an example of this type of QoS guaranteeing approach. On the other hand, the soft QoS method is not as strict as hard QoS methods regarding QoS requirements. DiffServ is an example of the soft QoS method.

Table 3.1 illustrates the implemented QoS models (hard QoS vs. Soft QoS) with respect to QoS metrics considered in the survey studies. Certain metrics are considered target QoS metrics to be provided in the studies. Therefore, Table 3.1 also reveals a broad category of problems/challenges handled in the studies from the QoS metrics viewpoint.

Table 3.1.: QoS models implemented in the techniques. The hard QoS approach guarantees the network resources for flows sent from source to destination. IntServ mechanism is an example for this approach. On the other hand, the soft QoS method does not guarantee the QoS requirements of the flows sent from source to destination throughout the entire session. DiffServ method is an example of soft QoS method.

QoS Models / Techniques	Hard QoS				Soft QoS			
	Bandwidth	**Delay**	**Jitter**	**Loss**	**Bandwidth**	**Delay**	**Jitter**	**Loss**
Wallner et al. [140]					✓	✓		
Civanlar et al. [142]					✓	✓		✓
HiQoS [143]					✓	✓		
OpenQoS [144]					✓	✓		✓
VSDN [145]	✓	✓	✓					
RVSDN [146]	✓	✓	✓					
Tomovic et al. [147]	✓							
Egilmez et al. [148]					✓	✓		✓
Egilmez et al. [149]					✓	✓	✓	✓
ARVS [150]							✓	✓
Yilmaz et al. [151]							✓	✓
Egilmez et al. [152]							✓	✓
Egilmez et al. [153]					✓	✓	✓	✓
Karakus et al. [84]	✓	✓						
FlowBroker [71, 154]	✓	✓		✓	✓	✓		✓
Wang et al. [155, 156]	✓	✓						
Miao et al. [157]		✓		✓		✓		✓
CXP [158]	✓	✓			✓	✓		
Kim et al. [159]	✓	✓		✓				
NCL [160]	✓	✓	✓					
Duan et al. [161, 162]	✓	✓			✓	✓		

Table 3.1 continued

QoS Models / Techniques	Hard QoS				Soft QoS			
	Bandwidth	**Delay**	**Jitter**	**Loss**	**Bandwidth**	**Delay**	**Jitter**	**Loss**
FlowQoS [163, 164]	✓	✓			✓	✓		
Afaq et al. [165, 166]					✓			
QoSFlow [167]					✓			
OpenQFlow [168]	✓				✓			
Xu et al. [169]					✓	✓	✓	
Wang et al. [170]	✓	✓			✓	✓		
Caba et al. [171]	✓	✓			✓	✓		
Truong et al. [172]					✓	✓	✓	✓
Kumar et al. [173]	✓				✓			
Yiakoumis et al. [174]	✓	✓	✓		✓	✓	✓	
Kassler et al. [175]					✓	✓		
Q-POINT [176]	✓	✓		✓	✓	✓		✓
QFF [177]					✓			
Gorlatch et al. [178, 179]					✓	✓		
Jarschel et al. [180]	✓				✓			
Ayadi et al. [181]					✓	✓		
Q-Ctrl [182]	✓				✓			
PolicyCop [183]					✓	✓		
OpenCache [184, 185]					✓	✓		
Sonkoly et al. [186]	✓	✓	✓		✓	✓	✓	
SoIP [187]					✓			
ACDPA [188]					✓	✓		

SDN adopts separation of data plane and control plane for networks. This separation enhances the network controller with regard to control of the networks. Also, in SDN concept, the network applications are not forced to deal with low-level configurations of data plane devices and are provided with abstract view of the network by controllers. Controllers can obtain global network view and states, e.g. statistics, network resource availability, events, by sampling of packets. Using this information, control policies and SLAs can be specified (even dynamically be adjusted) by an administrator at a higher abstraction level without a need to reconfigure low-level settings at each of the forwarding devices. The set of policies and also the different flow classes are unrestricted, allowing for fine-grained tuning based on the needs of the user. The rules can, therefore, be defined per-flow (if necessary) and the controller

has the task to apply them properly to the different network elements. Without a doubt, all of these mechanisms are crucial for QoS.

QoS can benefit from advantages of SDN concept in different network functions. Table 3.2 shows some main features of SDN, which are used in the surveyed papers, and their relation with this work's organization. Flow based forwarding allows networks to route different application flows in different treatments (e.g. priorities). Dynamic flows rule update enables network operators to update rules installed in network devices on-the-fly without interrupting device operations. SDN also renders flow/packet analysis possible to acquire header fields of them. Since SDN provides global network view it is possible to maintain related states for a full path of a flow. Furthermore, monitoring network statistics based on different levels such per-flow, per-port, per-device and so on is achievable. Moreover, in OpenFlow-enabled SDN networks, queue management and scheduling operations are also possible by means of some other southbound plugins such as OF-CONFIG and OVSDB protocols.

- One function that SDN can help networks improve is QoS-motivated routing. With SDN architecture, per-flow routing (both intra-domain and inter-domain) becomes viable through more scalable, simpler and less time-consuming mechanisms compared to traditional architectures. OpenFlow enables network operators to use various routing algorithms (rather than the typical shortest path) within the controller to generate forwarding tables that govern different isolated flows, such as the QoS flows, in the data plane [189]. Also, dynamic routing of flows are viable by controllers due to decoupling of control and forwarding functions of devices. These abilities, per-flow and dynamic routing, allow network administrators to come up with more QoS-motivated routing mechanisms for their networks.

- Also, SDN can help network operators create powerful and easy-to-use automated QoS management frameworks by means of resource reservation and queue management and packet scheduling for their networks. QoS provisioning

Table 3.2.: Main features of SDN, which are used in the surveyed papers, and their relation with the organization of this survey.

Organization / SDN Features	Routing	Architecture	Management			Modeling	
	Multimedia Flows Routing Mechanisms	**Inter-domain QoS Routing Mechanisms**	**Resource Reservation Mechanisms**	**Queue Management and Scheduling Mechanisms**	**Network Monitoring Mechanisms**	**QoE-Aware Mechanisms**	**Other QoS-related Mechanisms**
Flow-based forwarding	✓	✓				✓	
Dynamic Flow Rule Update	✓	✓	✓	✓		✓	✓
Flow/Packet Analysis	✓	✓	✓	✓	✓	✓	✓
Flow Path Analysis		✓	✓		✓		✓
Traffic Monitoring			✓	✓	✓	✓	✓
Queue Configuration	✓			✓		✓	✓

for network applications require well-defined control mechanism due to dynamic nature of network resources. As SDN brings capabilities to obtain global view of network controlling QoS configuration becomes easier compared to traditional network architectures.

- Furthermore, user QoE improvement can also benefit from SDN capabilities. User satisfaction cannot be guaranteed just by providing certain QoS parameters since these low-level network parameters represent the network states in terms of numbers. However, real user satisfaction (i.e. QoE) may require different QoS parameters which can dynamically change over the time. SDN's ability to manage network flows in a finer-granular way by flow rules through an automated control can help improve user QoE in a network.

- Moreover, network monitoring task is another function that SDN can help within a network. Monitoring task is crucial for a network since it helps detect and respond threads, performance issues in real time, and predicting future behaviors in a network. SDN allows network managers to monitor network dynamics through counters at very low levels such as per-packet, per-port, per-table, per-queue, and per-meter.

- Finally, SDN can be utilized to provide QoS in some miscellaneous ways such virtualization-based QoS provisioning, QoS policy management, and content delivery mechanisms due to some of its features such as per-flow control concept and multi-header field based routing.

The aforementioned network functions/tasks mainly form the underlying logic of the organization in which the surveyed papers are presented. These categories are the most prominent ways in which QoS can benefit from the concept of SDN.

In the rest of the chapter, Table 3.3 shows some features of studies such as queuing and/or scheduling mechanisms, scaling domain, simulation and/or emulation environment, and the controller(s) exploited in the development stages. Table 3.4 illustrates

the corresponding SDN planes that the techniques impact. Table 3.5 illustrates the organization, based on the categories identified, of the studies surveyed in the chapter along with their short descriptions.

3.5 Multimedia Flows Routing Mechanisms

With proliferation of different applications (e.g. video conferencing, VoIP etc.) on the Internet, more sophisticated and efficient routing mechanisms are needed for these types of application to meet their QoS demands. However, routing in today's traditional networking is an ossified issue due to some unsolved issues such as network's limited global view, per-hop decisioning, and limited QoS abilities for flows. The SDN and OpenFlow couple is considered a prospective solution architecture for the routing problems of the current networking. Decoupling of control plane and data plane in SDN brings many opportunities to routing functionalities. Supporting QoS in SDN/OpenFlow networks becomes more feasible owing to a logically centralized controller component of the SDN. With OpenFlow, it is possible to use various routing algorithms with different objectives such certain delay limit or packet loss (rather than just shortest path routing) within a controller and generate flow tables accordingly in forwarding devices. Flows can be dynamically routed in a per-flow basis with end-to-end QoS over the paths by means of the controller. Further, it allows to utilize the network resources in a more efficient way compared to today's architectures.

QoS-greedy multimedia applications such as video conferencing, distance learning, and interactive gaming are becoming prevalent in recent years. Efficient delivery of streaming media over the Internet presents many challenges. Flows of multimedia streaming require steady network resources with little or no packet drop and delay variation depending on the application. For example, while VoIP data is delay sensitive HTTP data requires reliable transmission. This indicates that different types of media may have different quality impairments under the same network condition. Therefore, designing multimedia flows routing frameworks that can cope with varying

network conditions becomes important. Classification and prioritization of flows are the key points at designing such frameworks. QoS routing of video streaming over OpenFlow networks is studied in [142]. The authors introduce a formula based on linear programming aiming at reducing packet loss and keeping delay tolerable for Scalable Video Coding (SVC) base layer video flows while calculating routes for QoS flows. The idea is to keep the best-effort traffic on typical shortest paths and maintain a best-effort traffic table while video flows are routed on QoS-rich paths calculated by the proposed formula and maintaining QoS flows table for them. HiQoS application [143] exploits an SDN-based ECMP (Equal Cost Multipath Routing) algorithm, presented in [190], to find multiple paths between source and destination along with using queuing mechanisms to provide bandwidth guarantee for different classes of traffic. It differentiates different types of traffic and provide different bandwidth guarantees to different services through queuing mechanisms on the SDN switches. The multi path routing component finds multiple paths meeting certain QoS constraints between the source node and the destination node, and calculates the optimal path for each flow by real time monitoring of the network state.

An OpenFlow controller (OpenQoS) design for video streaming with QoS support is presented in [144]. The key concept in this architecture is the classification of the incoming flows as multimedia flows and data flows using packet header fields. These flows are dynamically routed on the QoS-supported paths while data flows are subject to best-effort routing. Another controller architecture and protocol (VSDN) for supporting QoS for video applications over SDN networks is presented in [145]. It allows video applications to request end-to-end guaranteed services (GS) from the network. They achieve guaranteed services by modifying limited switch capabilities provided by OpenFlow. The queue properties of OpenFlow, "ofp_queue_properties", has been modified to support GS based queuing as "ofp_queue_prop_gs_rate" to contain required fields for token bucket based traffic shaping. VSDN switch creates a token bucket shaping queue for each requested flow. The queuing process using GS regulates traffic per flow based on traffic specification provided by VSDN controller.

The study in [146] is an extension of the VSDN architecture to address reliable QoS support for video streaming by adding "reliability" constraint to the problem of path calculation for a requested QoS path. Classification of flows is exploited for different routing treatment in networks. Tomovic et al. [147] also propose an SDN controller architecture that performs route calculations and resource reservations based on flow specifications for priority flows in an automated manner. It uses an algorithm that avoids highly utilized links even if traffic passing over them is best-effort.

Finding a route that provides best QoS for flows is not an easy task. Also, calculating such a route is not enough since network resources can dynamically change anytime. Therefore, a certain path may not be a good route for a flow all the time. To this end, frameworks taking into account these network changes are needed to keep flows under QoS guaranteed routes and provide optimized QoS. QoS routing should optimize a different cost function than simply the path length. For example, routes that have larger capacity even with longer distances may be more preferable to shorter routes that may cause packet loss. In [148, 149], the authors propose an optimization framework for video streaming with dynamic rerouting capability on the OpenFlow controller. To this end, they introduce two optimization problems along with their formulations. In the first problem, only lossless QoS flows (the base layer of the SVC encoded video) are routed under congestion conditions with an aim of no packet loss. In the second problem, both lossless QoS flows and lossy QoS flows (enhancement layers of the SVC encoded video) are routed with goals of no packet loss and minimized loss, respectively. ARVS (Adaptive Routing Video Streaming) approach [150] also studies the same optimization problem for adaptive routing of video packets. In ARVS, if the shortest path does not satisfy the delay variation constraint, the base layer packets have the first priority to be rerouted to a calculated feasible path based on the available bandwidth of this path, and the enhancement layer packets will stay on the shortest path. However, if there is no available bandwidth in this path, the base layer packets will stay on the shortest path while the enhancement layer packets will be rerouted to this path.

Server load balancing can affect quality of video streaming for end users. Server load balance requires continuous monitoring of the load of each server and dynamically rerouting current or new service requests to available servers for lower delay and distortion in case of servers are overloaded. SDN can help mitigate this problem since it can provide global network view to users. For this problem, a load balancing application that reroutes flows of video streams is presented in [151]. When the application detects server overloading, it calculates cost metrics (packet loss and delay) for each route connecting the user to each server. The old flows are deleted and new flows are pushed to all switches along the new least cost route.

Providing QoS-guaranteed paths for flows in networks is a challenging task for network operators. This objective requires taking many restrictions (e.g. bandwidth, delay etc.) into account before supplying such paths. Researchers anticipate that SDN and OpenFlow couple can help network administrators make flow-based routing easier compared to current state of it since it can provide centralized and finer-granular flow management along with global network view. Therefore, they propose various routing frameworks that exploit advantages of SDN and OpenFlow to make QoS provisioning easier for network paths.

3.6 Inter-domain QoS Routing Mechanisms

A single controller solution in the current OpenFlow specification is not scalable for large-scale multi-domain networks due to the limitation in processing power of the single controller, latency resulting from distant network devices, and huge amount of overhead because of messaging between controller and switches. Therefore, there is need for a distributed control plane with multiple controllers so that each controller is responsible for a part (domain) of the network. Routing end-to-end QoS flows between these networks requires collecting up-to-date global network state information, such as delay, bandwidth, and packet loss rate for each link. However, over a large-scale network, this is a difficult task because of problem dimension (size) and network

operators' intent not to share internal precise network dynamics in detail. Therefore, distributed QoS routing models need to consider all these challenges to ensure optimal end-to-end QoS for applications.

A distributed control plane-based routing architecture for video streaming over OpenFlow networks is presented in [152, 153]. In this routing architecture, each domain controller aggregates internal network resource information for each border node pairs (called virtual links) and share with other domain controllers. In this way, each controller acquires a global view of whole network and becomes capable of calculating an end-to-end QoS optimized route for flows. Karakus et al. [84] propose a similar QoS routing architecture but it utilizes a hierarchy-based network architecture in which network controllers compose hierarchy-levels along with another controller, called "Broker", on the top level. Each network controller shares its summarized network state information with the Broker instead of other controllers. The Broker keeps the global network state and view to share necessary information with certain controller when needed. FlowBroker [71, 154] architecture also exploits Brokers for network performance enhancement and load balancing regarding flow coordination over multiple domains in SDN.

An important problem in inter-data center (IDC) traffic management is bandwidth allocation to competing applications while maximizing the overall network utilization and considering QoS metrics and fairness. MCTEQ [155, 156] model proposes a joint bandwidth allocation to multiple traffic classes. It uses SDN concept to give preference to higher priority traffic in grabbing bandwidth by associating its utility with a larger weight while considering end-to-end delay requirement of interactive applications. Miao et al. [157] exploit SDN paradigm's control plane to update the look-up-table (LUT) of OPS (Optical Packet Switching) nodes at data center networks by extending OpenFlow protocol. By this way, application flows are switched by the OPS at sub-ms hardware speed, decoupled from the slower (millisecond timescale) SDN control operation. Hence, with flows prioritization and faster speed, it is possible to guarantee QoS for flows for intra data center traffic.

Pathlets (i.e. partial paths) based models are also leveraged to provide inter-domain end-to-end QoS paths. In this model, pathlets with specific QoS properties from each autonomous domain are advertised to an independent external entity that manage them for an end-to-end route. Control Exchange Point (CXP) [158] exploits abstracted network paths to orchestrate the end-to-end stitching of slices (a flow space associated with a specific service and a virtual topology (e.g. pathlets)) that the ISPs provide. The task of the CXP is to admit requests for QoS-guaranteed end-to-end paths, embed paths in the inter-domain virtual topology, and monitor the provided QoS guarantees.

3.7 Resource Reservation Mechanisms

This type of frameworks typically exploits flow classification and a rate-shaping through some modules implemented in controllers. A classifier module uses packet header fields to classify the packet and assign a priority to the corresponding flow based on network QoS policies. The rate-shapers then manage the flow rates to install corresponding rules in switches over the path in order to reserve resources for flows needing QoS.

The rate-limiters and priority queues can also be used with high level service requirements for resource reservation to provide QoS. The architecture in [159] exploits extensions to the OpenFlow's QoS capabilities. The proposed QoS controller creates network slices for different applications and feeds them with required performance requirements. These network slices are set of services defined by certain QoS performance requirements such as max bandwidth, min delay, etc. for each network slice. The authors utilize a mechanism called "QoS APIs", an extension to OpenFlow, so as to control configuration and management of QoS parameters. The aggregated bandwidth usage is accomplished by the rate-limiter APIs and the queue mapping API is exploited to map flow(s) to priority queues in ports in order to cope with bandwidth and delay allocation.

Table 3.3 shows some features of studies such as queuing and/or scheduling mechanisms, scaling domain, simulation and/or emulation environment, and the controller(s) exploited in the development stages. Most of the studies target a single domain and do not modify (i.e. use available default queues) the queue mechanism(s) of associated data plane devices (e.g. switches) in their architectures. Moreover, albeit the most of the studies state that their frameworks work with any OpenFlow controller by little modification (if not necessary), the Floodlight controller has been also chosen due to its QoS support over other controllers, highly modular design, and rich set of APIs.

Table 3.3.: Some features of studies such as queuing and/or scheduling mechanisms, scaling domain, simulation/emulation environment, and the controller(s) exploited in the development stages of the techniques.

Features / **Techniques**	**Queuing/ Scheduling**	**Scale (Domain)**	**Simulation/Emulation Environment**	**Controller**
Wallner et al. [140]	Default	Single	Presentation of the architecture, no simulation or testbed implementation	Floodlight
Civanlar et al. [142]	Default	Single	Simple 4-forwarder OpenFlow testbed	NOX
HiQoS [143]	Default	Single	Used Mininet [102] tool and topology with 5 switches, 2 servers, and 11 clients	Floodlight
OpenQoS [144]	Default	Single	3 OpenFlow-enabled Pronto 3290 switches, 1 controller, 3 hosts	Floodlight
VSDN [145]	Token Bucket Shaping (TBS), Weighted Fair Queuing (WFQ)	Single	A topology of 6 nodes in NS-3 simulator	VSDN
RVSDN [146]	TBS and WFQ	Single	A topology of 6 nodes in NS-3 simulator	RVSDN
Tomovic et al. [147]	HTB	Single	6 Open vSwitch (OVS), 4 clients, 4 servers	POX
Egilmez et al. [148]	Default	Single	Used a simulator implemented using LEMON library, used 6 nodes	Any
Egilmez et al. [149]	Default	Multi	Used a simulator implemented using LEMON library, used 15 domains with 300 nodes total	Any

Table 3.3 continued

Features / Techniques	Queuing/ Scheduling	Scale (Domain)	Simulation/Emulation Environment	Controller
ARVS [150]	Default	Single	Used Mininet and a topology with 30 nodes, 20 Mbps bandwidth, 10 ms and 20 ms delays randomly of each link	Floodlight
Yilmaz et al. [151]	Default	Single	Used 2 servers, 2 switches, 1 controller and a traffic loader to test different scenarios	OpenDaylight
Egilmez et al. [152]	Default	Multi	Used a simulator implemented using LEMON library, used 6 domains with 30 nodes each	Any
Egilmez et al. [153]	Default	Multi	Used a simulator implemented using LEMON library, used 6 domains with 30 nodes each	Any
Karakus et al. [84]	Default	Multi	4 domains with 4 nodes each in simulation	Any
FlowBroker [71, 154]	Default	Multi	Used Mininet to test 5 different scenarios	Floodlight
Wang et al. [155, 156]	WFQ, RED, Self-clocked Fair Queueing (SCFQ)	Multi	Used Google's IDC backbone network (G-WAN) and IBM's global data center network (SoftLayer) for extensive simulations	Central Traffic Manager
Miao et al. [157]	Default	Single	Used virtual networks connected to each other with ToRs and Racks.	OpenDaylight
CXP [158]	Default	Multi	Used 5 IXPs data	Any
Kim et al. [159]	Priority Queuing (PQ)	Single	3 ProCurve 5406zl switches	NOX
NCL [160]	Default	Single/Multi	Presented a use case with description of the architecture	Floodlight
Duan et al. [161, 162]	Default	Single/Multi	Numerical Results with examples	Any
FlowQoS [163, 164]	Default	Single	OpenWrt router with OVS integration, Raspberry Pi as controller hardware	POX
Afaq et al. [165, 166]	Default	Single	Used Mininet and a linear topology with 4 OVSes connected to a host each	Floodlight
QoSFlow [167]	HTB, Stochastic Fairness Queuing (SFQ), RED	Single	Up to 3 TPLink 1043ND switches	Any

Table 3.3 continued

Features / Techniques	Queuing/ Scheduling	Scale (Domain)	Simulation/Emulation Environment	Controller
OpenQFlow [168]	Rate Controlled Static Priority (RCSP), CETA based scheduling	Single	Used a data plane module Cavium OCTEON CN5650 with multi-core processors, each assigned different tasks	Any
Xu et al. [169]	Pre-defined queues w/ priority levels	Single	A topology w/ 8 switches and 6 hosts in Mininet and also a psychical network w/ 3 switches	RYU
Wang et al. [170]	Weighted RED, PQ, Weighted Round-Robin (WRR) scheduling	Single	Used a testbed with 3 Dell R410 servers and 4 R710 servers	NOX
Caba et al. [171]	Priority Queuing (PQ)	Single	Used Distributed OpenFlow Testbed (DOT) [191] and topology consisting VMs containing controller and 4 OVS switches and 3 hosts	Floodlight
Truong et al. [172]	Default	Single/Multi	Implemented in a testbed infrastructure, consisting of 3 layers, set up in their lab	Any
Kumar et al. [173]	FIFO and HTB	Single	Emulated a small home network with TP-LINK WR1043ND gateway router and DELL PowerEdge R620 OVS switch as ISP access switch	Floodlight
Yiakoumis et al. [174]	Minimum-rate queuing and WFQ	Single	Implemented a minimal user-ISP with TP-LINK WR1043ND home gateway router and 48-port Pronto switch as ISP access switch	Any
Q-POINT [176]	WFQ	Single	Evaluated in a random topology with 4 nodes and CARnet-like topology with 9 nodes using IBM ILOG CPLEX Optimization Studio	Any
QFF [177]	Default	Single	A testbed recreating home network with TP-LINK WR1043ND home gateway router with Pantou and 3 clients	Any

Table 3.3 continued

Features / Techniques	Queuing/ Scheduling	Scale (Domain)	Simulation/Emulation Environment	Controller
Gorlatch et al. [178, 179]	Default	Single	Numeric experimental study for low-level and application-level QoS metrics	Any
Jarschel et al. [180]	Default	Single	A testbed with two Pronto 3290 switches and a Dell PowerEdge 860 server as controller platform	Floodlight
Ayadi et al. [181]	Default	Single	Numerical evaluation	Any
Q-Ctrl [182]	Default	Single	Real-time experimental setup with 2 PowerEdge T110 II servers, HP2920 and Pica8 switches and 6 VMs launched in servers	Floodlight
PolicyCop [183]	Default	Single	An experiment with 5 switches and 4 hosts	Floodlight
OpenCache [184, 185]	Default	Single	No complete experiments yet	NOX
SoIP [187]	Default	Single/Multi	Used 3 switches w/ 100 Mbps link capacities for edge network and 2 routers for core network	Any
ACDPA [188]	Default	Single	Used Mininet and a topology w/ randomly connected 20 switches and 30 hosts	OpenDaylight

SDN and Network as a Service (NaaS) paradigms can be cooperated to address the problem of providing QoS parameters for application requirements while providing end-to-end service provisioning. NCL (Network Control Layer) [160] framework supports the low-level network QoS provisioning for requirements of different types of data flows by means of resource reservation. While SDN brings the ability to flexibly manage and program the underlying network, the NaaS paradigm supply users secure and isolated access to the network. In addition, the NaaS paradigm provides ability to easily expand or shrink the network services. The proposed NCL architecture has two main parts: The QoS SDN Application (SDNApp) and the Monitor Module. The SDNApp accounts for adaptation of control plane to the providers' requirements and configures the data plane accordingly. while the SDN Monitor component is responsible for monitoring the network states and collecting statistics from switches by means of OpenFlow counters. Duan et al. [161] also present a NaaS-applied framework in SDN that enables network service orchestration for supporting inter-domain

end-to-end QoS. A high-level abstraction model for network service capabilities is proposed and a technique for determining required bandwidth in network services to achieve QoS guarantee is developed. Network calculus is exploited in the proposed modeling and analysis which makes the developed techniques general and applicable to networking systems consisting of heterogeneous autonomous domains. In [162], the authors extend the study presented in [161] to develop the idea of NaaS-SDN integration to propose a framework of a NaaS-based Service Delivery Platform (SDP) for a multi-domain SDN environment. This platform provides a high-level abstraction of each SDN domain as a network service and enables network service orchestration for end-to-end service delivery. They investigate two key technologies for achieving end-to-end QoS guarantee through this SDP, an abstract model for network service capabilities and a technique for end-to-end bandwidth allocation.

Making per-flow and application-based QoS allocation hassle-free is an important task in home networks using an SDN-based approach because home networking devices have less processing power than typical networking devices and the users are not skilled. FlowQoS [163, 164] is a system in which users of the broadband access network simply specify the high-level applications that should have higher priority (e.g., adaptive video streaming, VoIP) compared to others. The FlowQoS controller performs the appropriate application identification and QoS configuration for both upstream and downstream traffic to implement a user's preferences. For each flow, FlowQoS performs on-the-fly application identification. It also installs rules in the data plane that forward individual flows according to user-specified priorities for those applications. The system creates links in a virtual topology in the home router, configures each of these links with a user-specified rate, and assigns flows to these links to provide rate shaping per application.

Long-lived flows are mostly called elephant flows and are large transfer such as backups. These elephant flows can affect the performance of the network since network resources are consumed by them and they fill buffers end-to-end. Other flows may be affected from this tendency because they also use the same buffers with ele-

phants. Therefore, detecting elephant flows and satisfying their QoS needs is needed for a better network performance. In [165, 166], a QoS provisioning mechanism is proposed for elephant flows after their detection. In the proposed approach, flows over a specified threshold value, called elephant flows, are subject to QoS module application that routes them to rate-limited queues (e.g. max or min bandwidth) for traffic shaping QoS technique. The QoS module application enables the network to define a queuing policy which exploits the *enqueue* action in OpenFlow to enqueue certain types of flows in the network.

3.8 Queue Management and Scheduling Mechanisms

The order of some packets in a queue may have more priority than other packets which are ahead of them in the queue. This idea has impact on QoS along with the traffic shaping. Hence, the QoSFlow [167] model manipulates the multiple packet schedulers, i.e. not only FIFO, in Linux kernel in order to provide more flexible and manageable QoS control mechanisms in OpenFlow-enabled networks. The QoSFlow combines the Linux packet schedulers along with OpenFlow networks and supports the Hierarchical Token Bucket (HTB), Random Early Detection (RED), and Stochastic Fairness Queuing (SFQ) schedulers. The QoSFlow enriches the software switches of OpenFlow. The authors state that they use OpenFlow 1.0 because of its stability and ability to let users make use of different schedulers. The QoS module of QoSFlow has three components: Traffic Shaping, Packet Schedulers, and Enqueueing. The Traffic Shaping and Packet Schedulers are responsible together for manipulation of bandwidth size in queues. On the other hand, the Enqueueing component administrates the flow table messages of OpenFlow protocol and mapping flows to queues where maximum 8 queues is supported per switch port.

OpenQFlow architecture [168] is a variant of OpenFlow architecture that provides microflow-based QoS in a scalable manner. It divides classic flow table framework to three tables: flow state table, forwarding rule table, and QoS rule table. The flow state

table entries are used to maintain 128-byte micro-flow state information including forwarding, QoS, and statistics information. It is used to find the forwarding and QoS information base without rule table lookups. Therefore, this increases the scalability of OpenQFlow architecture. Each entry of forwarding rule table maintains a pointer to a forwarding information base that comprises of forwarding information such as forward and drop. Similarly, each QoS table entry has a pointer to a QoS information consisting of the traffic type, bandwidth, and priority information. OpenQFlow brings two packet scheduling schemes, BETA and CETA, that provide max-min fairness without the need of output queues per flow.

Queue-based classification techniques are used in [140] to achieve the QoS support in Floodlight-controlled SDN networks. To this end, traffic shaping (rate limiting) and DiffServ DSCP (Differentiated Services Code Point) approaches are exploited for QoS support in Floodlight-based SDN networks. The authors describe different class of services along with rate limiting paths between switches. In their approach, the main player is the "SDN module" that is responsible for packet matching, classification, and flow operations like insertion, deletion etc. This QoS component tracks and stores service classes with their DSCP values. The QoS module allows the network to define two different main policies: Queue-based policy and ToS/DSCP-based policy. The Queue-based policy exploits enqueueing mechanisms for flows while the ToS/DSCP-based policy uses class of services with a name (e.g. Expedited Forwarding, Best Effort etc.) and a corresponding DSCP value. An IPv4 ToS-based QoS mechanism is also proposed in [169]. It classifies flows as QoS flows and best flows and then assign them queues based on their priorities.

Another software defined automatic QoS management model is introduced in [170]. The proposed model includes certain QoS functions such as packet marking, queue management, and queue scheduling. It utilizes Weighted Random Early Detection (WRED) queue management algorithm, Priority Queuing (PQ), and Weighted Round-Robin (WRR) queue scheduling algorithms. It also proposes a Collaborative

Borrowing Based Packet-Marking (CBBPM) algorithm to improve the utilization rate of network resource.

OpenFlow alone is not enough to build more complex SDN services that require complete control and management of the data plane in terms of configurations of ports, queues, and so on. OVSDB protocol has been exploited to configure QoS capabilities of OVS switches in data plane in [171]. The proposed QoS Config API allows applications to configure priority queues on the ports of data plane devices by adding OVSDB at the D-CPI of a network controller. Hence, services and applications built on top of an SDN controller using the proposed QoS API can make use of the full set of QoS features available in OVS devices.

3.9 QoE-Aware Mechanisms

The requirements for network applications are diverse and today's networks try to support them based on QoS parameters. However, user satisfactions are not necessarily always met by just providing QoS for some applications like IPTV, real-time online interactive gaming, e-learning etc. since QoS is not powerful enough to express all features involved in a communication service [192]. Therefore, the performance of a specific application cannot be determined by simply relying on QoS metrics. Instead, user QoE is an alternative measurement of user satisfactions for those applications over the network. Therefore, a major challenge for future networks is to dynamically adapt QoE demands of the users to QoS parameters in the network. However, mapping user QoE to network QoS parameters is a challenging issue over the networks. This is especially true for networks with limited resources like today's access networks. To this end, there are some researches aiming to maximize QoE of users while providing required QoS in SDN/OpenFlow networks.

Table 3.4 illustrates the corresponding SDN planes that the techniques impact. Each study targets a main plane in the SDN architecture to implement the idea presented in the studies. Most of the techniques are conducted in control plane since

it provides the control functions of the SDN paradigm. It is important to notice that the techniques do not solely rely on a specific plane of SDN architecture to implement their ideas due to cooperation among planes.

Table 3.4.: Impact of the techniques on the SDN planes.

SDN Planes / Techniques	Application Plane	Control Plane	Data Plane	Management Plane
Wallner et al. [140]		✓		
Civanlar et al. [142]		✓		
HiQoS [143]	✓			
OpenQoS [144]		✓		
VSDN [145]		✓		
RVSDN [146]		✓		
Tomovic et al. [147]		✓		
Egilmez et al. [148]		✓		
Egilmez et al. [149]		✓		
ARVS [150]		✓		
Yilmaz et al. [151]	✓			
Egilmez et al. [152]		✓		
Egilmez et al. [153]		✓		
Karakus et al. [84]		✓		
FlowBroker [71, 154]		✓	✓	
Wang et al. [155, 156]	✓	✓	✓	
Miao et al. [157]		✓	✓	
CXP [158]		✓		
Kim et al. [159]		✓		
NCL [160]		✓		
Duan et al. [161, 162]	✓	✓		
FlowQoS [163, 164]	✓	✓		
Afaq et al. [165, 166]	✓	✓		
QoSFlow [167]			✓	
OpenQFlow [168]		✓	✓	✓
Xu et al. [169]		✓	✓	
Wang et al. [170]	✓		✓	✓

Table 3.4 continued

SDN Planes / Techniques	Application Plane	Control Plane	Data Plane	Management Plane
Caba et al. [171]		✓	✓	
Truong et al. [172]	✓			
Kumar et al. [173]	✓	✓		
Yiakoumis et al. [174]	✓	✓		
Kassler et al. [175]	✓	✓		
Q-POINT [176]	✓	✓		
QFF [177]		✓		
Gorlatch et al. [178, 179]	✓			
Jarschel et al. [180]	✓	✓		
Ayadi et al. [181]		✓		
Q-Ctrl [182]		✓		
PolicyCop [183]		✓		✓
OpenCache [184, 185]		✓		
Sonkoly et al. [186]			✓	
SoIP [187]		✓		
ACDPA [188]		✓	✓	
OpenNetMon [193]	✓	✓		
PayLess [194]	✓	✓		
Isolani et al. [195]	✓	✓		
Jose et al. [196]	✓	✓		
OpenSketch [197]	✓	✓		
OpenTM [198]	✓	✓		
OpenSAFE [199]	✓	✓		

IPTV is an emerging application recently in networking world. Controlling and implementing QoS policies on a network is an issue for IPTV services. The QoE-aware IPTV network architecture presented in [172] combines IP Multimedia Subsystem (IMS) and OpenFlow-based network to optimize the network resources and service characteristics according to user satisfactions. In this design, users are able to rate the services that they are receiving and the proposed architecture maps and provisions the network QoS parameters accordingly. The architecture consists of three layers. The Application Layer includes the IMS IPTV Client and QoS engine to predict the user satisfaction. The IMS Core Layer is responsible for signaling and session/service control. Finally, the Media Layer is the data plane consisting of OpenFlow switches for transportation of traffic in the unicast, multicast or broadcast manners.

Enabling users to gain some controls over bandwidth allocation of access links for their devices and applications at home networks can be used to improve user QoE in such networks. The study in [173] leverages the SDN paradigm in ISP network to make such control delegation possible for users. The authors state that such a control by users not only improve user QoE but also allows ISP to monetize their services and powerfully compete with other ISPs in the market. They design a GUI that allows a typical user to specify their requirements on a per-device and per-application basis. The GUI then translates these requests into the appropriate API calls exposed by the SDN controller hosted in the ISP network. Finally, the ISP's SDN controller determines an appropriate resource allocation for the request, which it then configures into the switching hardware associated with that user's access link. Yiakoumis et al. [174] also present a very similar idea that proposes allowing users to choose the relative priority of their applications, and indicate their preference to the ISP that then enforces the preference by an OpenFlow controller.

Optimized path assignment while improving the QoE level of user perception for multimedia services is studied in [175]. The proposed system aims to enable negotiation of service and network communication parameters between users and to find a path for delivering flows for corresponding communication. The system leverages OpenFlow to set up the networking paths for users in order to maximize QoE while considering network resources such as link capacities, delay etc. and network topology. The two principle components of the proposed system are QMOF (QoS Matching and Optimization Function) and PAF (Path Assignment Function). QMOF resides in the SDN application layer and conducts an initial parameter matching process to produce feasible service configurations. PAF is located in the SDN control layer and executed on an OpenFlow controller. It optimizes the network paths to meet the resource requirements of a currently active service configuration. In [176], the authors propose the "Q-POINT", a QoE-driven path optimization model, built on [175] by formulating and solving the multi-user domain-wide QoE optimization problem. Their aim is to find a best path for each media flow while maximizing

the aggregated user-expected QoE value over all users and service flows in an SDN network domain, subject to resource constraints and network topology. They present the problem as a mathematical model, which is formulated as a mixed integer linear program.

Dynamic adjustment of bit rate has been used to reduce pauses and buffering times in video playbacks in recent researches. This idea brings its own advantages for overall user experiences. However, that model has some issues such as unstable and bursty flows, network congestion owing to independent adoption strategy as well. Furthermore, user requests to maximize their satisfactions without knowledge of others on the network is another drawback of variable bit rate idea. The "OpenFlow-assisted QoE Fairness Framework (QFF)" [177] architecture aims at mitigating aforementioned problems. The QFF framework improves the QoE for all network and video streaming devices, thereby users, along with network resources and requirements. The QFF framework watches video streams in the network so that it can dynamically adapt the flow parameters to fairly increase the QoE for users. The QFF exploits the idea of sharing resources (particularly bandwidth) evenly among users because a user or device may have a very low bandwidth rate than another one although its resolution is much higher than the latter. This results in reduced QoE of users. An OpenFlow-enabled controller takes a place in the heart of the QFF framework to control its functionalities.

Real-Time Online Interactive Applications (ROIA) such as Real-Time Strategy (RTS) games (e.g. StarCraft) require highly dynamic QoS characteristics from a network. ROIA currently use the network on a best-effort basis, because of the lack of control over QoS in traditional networks. However, this results in a sub-optimal QoE by the end-user. Use of SDN technology to meet the dynamic network demands of ROIA, therefore improving QoE, is studied in [178, 179]. The study propose a Northbound API consisting of two parts, Base API and Application-level API, in order to differentiate and map application-oriented QoS metrics to network-oriented ones. The Base API is a bridge between SDN controller and SDN modules. It receives

applications' high-level QoS metrics and translates them to low-level network QoS metrics so that the controller can provide. The Application-level API is responsible for applications' high level QoS metrics and prevents developer from low-level details.

Application information, such as per-flow parameters, and application signatures, and related QoS levels offer greater flexibility in terms of supporting QoE than hard QoS parameters. However, using them may require an overhead of signaling effort compared to management at the network level. The study in [180] investigates how different kinds of information or application quality parameters can support a more effective network management in an SDN-enabled network. The authors examine the trade-off between the QoE improvement due to more detailed application information and corresponding signaling overhead in an SDN-enabled testbed for the application of YouTube streaming.

3.10 Network Monitoring Mechanisms

One of the benefits that SDN promises is efficient use of network resources and ease of resource provisioning. SDN renders these features possible by decoupling of data plane and control plane. This separation simplifies the management of the network. Network operators maintain a global view of a network from a central control mechanism (i.e. controller). They can dynamically optimize flow management and resources. Moreover, per-flow, and/or application-level QoS provisioning becomes easier and feasible for network administrators. However, making all these features possible requires well-designed network monitoring frameworks. Network monitoring is employed for many different applications such as QoS management, resource utilization, anomaly detection, traffic engineering and so on. It helps collect data from network components like switches, routers (through southbound APIs such as OpenFlow), and controllers (through west/eastbound APIs from other controllers). Monitoring frameworks should be able to gather, process and deliver monitored data at requested aggregation levels (such as per flow, port, table etc.) and frequency

without introducing too much monitoring overhead into the network. In addition, they should pay attention to the accuracy and timeliness of measurements.

OpenNetMon [193] is a network QoS metrics monitoring module written for POX controller. It is used to monitor per-flow QoS metrics by polling flow ingress and egress switches at an adaptive rate. It utilizes querying flow counters to obtain per-flow throughput. They subtract the increase of the packet counter at destination switch from the increase of the source switch packet counter in order to calculate per-flow packet loss. The idea to calculate the path delay is to inject probe packets traveling the same path (i.e. links, nodes, buffers etc.). However, as a disadvantage, injecting such probes can bring extra message overhead to the controller.

PayLess [194] is a network statistics gathering framework. The PayLess framework works as a moderator between network applications and controller. It translates the high-level monitoring requirements of network applications for controllers and prevents applications from low-level details of statistics collection and storage management. The authors of PayLess also propose an adaptive monitoring algorithm which takes into consideration polling frequency to reduce the monitoring message overhead as well as accuracy of monitored statistics by only monitoring important switches.

An interactive approach to SDN monitoring, visualization, and configuration is studied in [195]. The proposed monitoring manager retrieves information about the network and stores it in a local database through a module called "Infrastructure Synchronizer". This module gathers control and data information such as traffic statistics and network topology information and stores a history of these changes along with SDN-related configurations performed by the network administrator.

A traffic measurement framework for online large traffic aggregates based on an OpenFlow approach is introduced in [196]. The proposed model works on commodity OpenFlow switches and can be used for various measurement tasks. The hierarchical heavy hitters (HHH) traffic problem is exploited to understand the trade-off between accuracy and overhead in the proposed framework.

OpenSketch [197] is a measurement architecture that provides a three stage packet processing pipelines (hashing, filtering, and counting) in SDN. It helps operators by making understanding the complex switch implementations and parameter tuning easier in diverse sketches. It proposes a measurement library configuring the pipelines for different sketches and allocating switch memory across tasks to maximize accuracy. OpenTM [198] concentrates on measuring traffic matrix estimation by periodically polling one switch on each flow's path and then combining the measurements. In OpenTM, after a switch has been chosen it is constantly queried for gathering flow statistics. Polling a single switch does not impose significant load on the network but may affect accuracy if the switch is not carefully chosen. A disadvantage of OpenTM is that it is limited to generating traffic matrices for offline use and does not capture packet loss and delay. OpenSAFE [199] uses OpenFlow to enable flexible monitoring of network traffic for security problems. It directs spanned network traffic towards predefined sinks (e.g., IDS) according to pre-specified policies. While such an approach could be used to compute network utilization (by analyzing the redirected traffic), the overhead it creates by copying all network traffic is prohibitive. OpenSAFE requires hardware investments to perform the actual monitoring that network operators are reluctant to do.

3.11 Other QoS-related Mechanisms

QoS in SDN/OpenFlow networks is not bounded just by routing, queue management, and QoE-aware mechanisms. Studies have been conducted in many broad areas of networking by taking advantage of SDN concept. Virtualization-based QoS providing, QoS policy management, content delivery mechanisms, and testbed QoS extension are some of the other ongoing studies in the SDN/OpenFlow networks.

- Virtualization-based QoS Provisioning—In recent years, many research efforts focus on effectively virtualization of computation, storage, server and network resources that are provided as a service over a network. Although server and

storage virtualization show a great success regarding efficiency and performance, the network resource virtualization do not achieve the same success due to restricted access of network operators to control plane of network devices. Therefore, SDN brings capabilities that pave the way for virtualizing network resources in an on demand manner by abstraction of the underlying network infrastructure to the applications. Ayadi et al. [181] exploit the network virtualization and SDN paradigm to meet applications' QoS requirements. The proposed VNOS (Virtual Network Operating System) plane is the fundamental layer for the virtualization of the network. In terms of the SDN, they use a distributed approach which manages the flows for QoS requirements in each network. The "Network as a Service" framework is used on top of the SDN controllers for management of virtual network flows. NaaS framework brings the management of aggregated flows and creation of a logical virtual network. To manage aggregation of flows, a mechanism called "solver 1" is leveraged to categorize the flows regarding their QoS criteria such as availability, delay, capacity, and reliability. These criteria are associated with a degree of high, medium, and low for flows and then each flow is classified as a pre-defined class of service (CoS) for aggregation. After classification and aggregation of the flows, a logical virtual network is created by interaction of management, control, and data planes. Q-Ctrl [182], QoS Controller, is an architecture for programmatically attaining requested QoS constraints by users in a SDN-based cloud infrastructure. The Q-Ctrl system is able to execute in a virtual overlay network via Open vSwitch (OVS), physical network infrastructure equipped with an SDN controller, or a simulated SDN environment via Mininet. It regulates and allocates the bandwidth for the virtual machines running on the Cloud infrastructure.

- QoS Policy Management—In general, service level agreements (SLAs) are used to establish QoS parameters for traffic management of QoS-greedy applications such as online interactive gaming, video streaming, and video conferencing.

Each SLA consists of a set of Service Level Objectives (SLOs) that are used to derive network level policies, which are in turn translated into device level primitives (e.g. forwarding rules, queue configurations, packet dropping rate, and traffic shaping policies). In traditional network architectures like DiffServ and MPLS, managing these QoS related policies are difficult due to static traffic classes with a coarse granularity of QoS levels and installation requirement of specialized software or hardware components in the network. On the other hand, SDN promises a rich northbound API possibility and global network view, it enables network operators to implement wide-range of network policies and rapid service deployment. PolicyCop [183] project aims at bringing a flexible, easy-to-control, and vendor-independent management of QoS policies by means of SDN Northbound APIs in SDN/OpenFlow networks. PolicyCop fairly benefits the features of OpenFlow to make itself a good management framework. It provides per-flow control and on-demand aggregation thanks to OpenFlow. Traffic definition is easier by PolicyCop, compared to DiffServ and MPLS, due to no need of shutting down the network devices. It promises reduced operational overhead and is easy to deploy in a network because of its vendor-independent feature. PolicyCop architecture has 3 planes: data plane, control plane, and management plane. The data plane and control plane are classic SDN planes. The management plane is the heart of the PolicyCop framework. It is divided into two parts as well: Policy Validator and Policy Enforcer. The Policy Validator tracks and detects the policy violations while the Policy Enforcer component takes charge in case of any policy violations and maintains network policy rules.

- Content Delivery Mechanisms—The ability to shape and control data traffic is one of the primary advantages of SDN. Being able to direct and automate data traffic makes it easier to implement QoS for certain applications such as Video-on-Demand (VoD). The increase in the use of VoD services brings a huge demand on servers of content networks. However, this demand load floods network resources such as bandwidth, latency etc. in response to user

requests. OpenCache [184, 185] mitigates the duplication of traffic in cases if a user from network A gets a content from another network B and another user from network A requests the same content from network B. Therefore, the content needs to traverse the operator's network again. By OpenCache, the VoD content is cached within the network (i.e. network A) to avoid this duplication. Therefore, it reduces not only congestion and inefficiency but also increases throughput and response time to user requests. It still keeps the unicast delivery fashion so that existing infrastructure can be maintained. To this end, OpenCache exploits SDN's data plane and control plane separation philosophy in order to redirect user requests for the same content to a local cache. In the OpenCache framework, there is another controller called "Cache Controller" that is intermediary (i.e. connected) between SDN controller and cache instances. The cache controller allows connections of redirected requests. It also maintains a full global state of underlying network so that it can modify and manage the cache instances.

- Testbed QoS Extension—Vendor-dependent implementations of composite device structures regarding hardware and software requires more attention to QoS support of testbeds. Hence, QoS support in OpenFlow-based testbeds like OFELIA[1] will contribute and encourage to SDN/OpenFlow QoS research from both academia and industry. Therefore, Sonkoly et al. [186] extend the Ofelia's architecture to support more QoS features for OpenFlow experiments in a more flexible, user friendly, and easy-to-manage way by a comprehensive study of different QoS settings and use-cases. The authors study the QoS features of diverse devices used in the Ofelia project for a comprehensive QoS performance analysis. Also, they extend the OpenFlow switches by defining vendor specific queue properties to selected queue types. Middleton and Modafferi [200] present their experience over 2 years running SDN network experiments on

[1]http://www.fp7-ofelia.eu/

three classes of testbed facilities: commercial Amazon EC2[2], pre-commercial federated testbed of FIWARE Lab[3] instances,and experimental OFELIA. They focus on measuring how testbed features limit the ability to perform an idealized experiment, and how effectively that experiment can be executed using the testbed support apparatus provided. This study compares and gives results for three testbeds regarding some qualitative metrics such as QoS Monitoring, external IP addresses, network slice isolation reliability and so on.

- SDN over IP for QoS—Although IETF has proposed a series of architectures QoS in IP networks, none of them has been successful to be a unified adoption due to their deficiencies such as complex structures or lack of fine-grained control over flows. It is becoming clear that a future architectures such as SDN can be a solution for aforementioned problems. However, use of such a new architectures over existing networks requires some long-term and fundamental changes such as equipment, training of network operators etc. Therefore, interoperation of legacy networks and SDN is being researched to overcome such changes in short-term. SoIP (SDN over IP) [187] approach promises providing better QoS guarantee for end users and applications using SDN over IP concept. The basic idea of SoIP is to update or reconstruct the network edge and build SDN-based overlay networks to take advantage of its per-flow control over flows while the network core maintains the existing differentiated services based on the ToS field of IP protocol header. This approach not only preserves the existing infrastructure and network devices but also enhance resource utilization and QoS guarantee.

- SDN and Hadoop for QoS—Advanced Control Distributed Processing Architecture (ACDPA) [188] takes advantage of both SDN and Hadoop[4] software framework to provide better QoS for flows. It uses SDN for network abstrac-

[2]http://aws.amazon.com/ec2
[3]https://www.fiware.org/lab/
[4]http://hadoop.apache.org/

tion and control and Hadoop for processing large amount of data coming from data plane. In ACDPA, Wireshark packet sniffer is used to capture packets from the network. Hadoop is then used to process the captured packets regarding classification and the results are given to the controller. The SDN controller gives corresponding priorities to the flows and propagate associated flow rules to switches to provide QoS.

Table 3.5.: Organization and descriptions of the studies surveyed in SDN/OpenFlow networks. These categories are the most prominent ways in which QoS can benefit from the concept of SDN.

Description / Organization	Techniques	Description
Multimedia Flows Routing Mechanisms (Sec. 3.5)	[142]	A QoS-enabled routing architecture for scalable video streaming
	[143]	Design of HiQoS application for multi path routing and queueing mechanisms
	[144]	A controller design, OpenQoS", for QoS-enabled routing of multimedia traffic delivery
	[145, 146]	A QoS-enabled (reliable) routing architecture (R-VSDN) for video streaming
	[147]	A QoS routing framework to provide resource-guaranteed paths for multimedia applications
	[148–150]	A QoS-enabled dynamic optimization-based routing architecture for scalable video streaming
	[151]	Server load balancing application that reroutes flows of video streams
Inter-domain QoS Routing Mechanisms (Sec. 3.6)	[152, 153]	A distributed QoS routing architecture for scalable video streaming over multi-domain OpenFlow networks
	[84]	A hierarchic network architecture with an inter-AS QoS routing approach
	[71, 154]	Design of Broker-based FlowBroker architecture for QoS support
	[155, 156]	Design of MCTEQ model proposing a joint bandwidth allocation for trafffic classes
	[157]	Use of SDN and OPS nodes for QoS support
	[158]	Design of Control Exchange Points (CXPs)" for QoS routing among ISPs
	[159]	A network QoS control framework for management of converged network fabrics
	[160]	A Network Control Layer (NCL) based on SDN, OpenFlow, and NaaS for QoS requirements of applications

Table 3.5 continued

Organization \ Description	Techniques	Description
Resource Reservation Mechanisms (Sec. 3.7)	[161, 162]	A framework to apply NaaS in OpenFlow networks to enable network service orchestration for supporting inter-domain end-to-end QoS
	[163, 164]	A system, FlowQoS, enabling users to specify high-level application flow prioritization (e.g. VoIP etc.)
	[165, 166]	A QoS provisioning mechanisms for elephant flows
Queue Management and Scheduling Mechanisms (Sec. 3.8)	[167]	A QoS control framework (QoSFlow) using multiple packet schedulers
	[168]	A QoS-motivated SDN architecture (OpenQFlow) for scalable and stateful SDN/OpenFlow networks
	[140, 169]	ToS/DSCP-based classification approach for QoS
	[170]	A hierarchical autonomic QoS model by adopting SDN
	[171]	A QoS configuration API using OVSDB protocol
QoE-Aware Mechanisms (Sec. 3.9)	[172]	A QoE-Aware IPTV network architecture design over OpenFlow networks
	[173, 174]	A system to improve user QoE by bandwidth allocation management framework at home networks
	[175, 176]	Design of Q-POINT, a QoE-driven path optimization model
	[177]	An OpenFlow-assisted QoE Fairness Framework (QFF) to maximize the QoE of clients in a shared network
	[178, 179]	A Northbound API design for online applications to increase QoE of users
	[180]	A study investigating how different kinds of information such as per-flow parameters, application signatures etc. can improve network management
Network Monitoring Mechanisms (Sec. 3.10)	[193]	Design and implementation of OpenNetMon monitoring framework
	[194]	Design and implementation of PayLess monitoring framework
	[195]	Design and implementation of an interactive network monitoring framework
	[196]	Design and implementation of traffic measurement framework
	[197]	Design and implementation of OpenSketch monitoring framework
	[198]	Design and implementation of OpenTM monitoring framework
	[199]	Design and implementation of OpenSAFE monitoring framework
Other QoS-related Mechanisms (Sec. 3.11)	[181]	A language to express QoS requirements of applications when placing virtual network components
	[182]	A QoS controller architecture, Q-Ctrl, for programmatically attaining requested QoS constraints by users in an SDN-based cloud infrastructure

Table 3.5 continued

Organization \ Description	Techniques	Description
	[183]	Design of a QoS policy management framework called PolicyCop
	[184, 185]	A caching mechanism (OpenCache) to store content for VoD services
	[186]	An architectural extension for QoS-enabled experiments in Ofelia using OpenFlow
	[187]	Design of SoIP architecture showing interoperability of SDN and IP for better QoS
	[188]	Design of ACDPA architecture using SDN and Hadoop for better QoS support
	[200]	Report of 2 years-running SDN network experiments on 3 different testbeds

Table 3.5 illustrates the organization, based on the categories identified, of the studies surveyed in the study along with their short descriptions.

3.12 Discussion

3.12.1 Research Challenges

While SDN matures, QoS provisioning in SDN/OpenFlow networks deserves more research efforts from both academia and industry. This subsection explains few main issues that need further attentions to complete QoS abilities of SDN/OpenFlow environments.

- Inter-AS QoS Provisioning: Most of the current research studies have been focused on providing QoS in intra-domain. While single-domain problem is important, supporting QoS for flows at inter-domain level is arguably more crucial and difficult owing to two obvious reasons among others: Firstly, majority of the traffic in the Internet is between hosts which are part of different autonomous networks (i.e. inter-AS traffic). Secondly, network administrators eschew sharing their internal network-related configurations since they are proprietary.
  SDX (Software Defined Internet Exchange) project [201, 202] tries to realize the use of SDN for inter-domain routing in IXPs (Internet Exchange Points) for more expressive, flexible and destination-independent forwarding. It aims at

curing the deficiencies of today's *de facto* inter-AS routing protocol BGP [203] by utilizing SDN features. SDX can enable some applications that are difficult and complex (if not impossible) in today's routing infrastructure: domain-based or application specific peering, enforceable inter-domain routing policies, remote traffic control, preventing free-riding, time-based routing, wide-area server load balancing etc. SDX faces some challenges as well such as developing proper isolation mechanisms for AS route selection processes, backward compatibility, and resolution of potential policy conflicts among ASes. A similar study [158] demonstrates an architectural model, "Control Exchange Point (CXP)", which dynamically stitch partial paths (called *pathlets*) provided by ISPs and provisions end-to-end QoS for services. CXPs leverage SDN principles such as the clean decoupling of the routing control plane from the data plane and the consequent centralization of control. The task of the CXP is to admit requests for QoS-guaranteed end-to-end paths, embed paths in the inter-domain virtual topology and monitor the provided QoS guarantees.

Another approach used in SDN case to mitigate the inter-domain routing is to utilize a more powerful controller(s), mostly called "Broker", with a full global network view over different ASes. This controller(s) is connected to domain controllers of other ASes. FlowBroker [71, 154, 204] architecture proposes use of brokers connected with network controllers. These brokers collect network state updates from each associated domains and forms corresponding local and global link state tables. When an inter-as flow requests comes to a broker, the broker calculates a path satisfying network metrics (e.g. packet loss ratio, delay etc.) of the request and sends this path information to controllers over the path. The main concern with this broker approach is that network operators consider their internal configurations proprietary and are not willing to share them with a third party control mechanisms.

The eXtensible Session Protocol (XSP) [205] supports application-driven configuration of network resources across domains. XSP provides mechanisms that

enable the configuration of dynamic networks services in support of applications such as GridFTP. The XSP libraries and APIs consolidate applications with a standard interface to define parameters determining network paths. The realization of these paths is then managed by the XSP Daemon (XSPd) that signals the underlying provisioning service while providing feedback to the application.

- QoS Signaling Overhead:

  SDN is a logically centralized architecture. This structure results in gathering all QoS-related signaling messages (i.e. overhead) at control mechanism of the network (i.e. controller) by means of statistics messages from data plane elements to controller(s). OpenFlow enables network operators to collect statistics at different level of flows such as per-flow or aggregation of flows. However, each of these collection approaches comes at a cost. While per-flow approach brings finer granularity regarding QoS-related states, it suffers from the scalability issue. On the other hand, aggregation of statistics mitigates the scalability problem yet restrains the OpenFlow fine-granular flow independence semantics. Also, in an SDN/OpenFlow environment, a controller can poll a switch to collect statistics on the active flows. Alternatively, it can request a switch to push flow statistics (upon flow timeout) at a specific frequency (i.e. periodically). Moreover, one important issue is how often the QoS information should be sent from network elements to controller. Even though pulling statistics frequently from data plane help controller maintain up-to-date global vision of network states, it brings extra overhead to be handled by the controller owing to processing information. Therefore, this is process is a trade-off between measurement accuracy, timeliness and signaling overhead and thereby resulting in control plane scalability issue for controller [206]. The PayLess [194] framework provides different flow aggregation levels by a RESTful API for flow statistics collection. It uses an adaptive statistics collection algorithm that delivers highly accurate information in real-time without incurring significant network overhead. The algorithm

can achieve an accuracy close to constant periodic polling method while having up to 50% reduced messaging overhead compared to periodic poling strategy.

3.12.2 Lessons Learned

This survey experience has showed us several important points that require more attention from researchers to provide QoS in SDN networks.

QoS support for applications and service provisioning have been difficult tasks to achieve for quite a while even though newer applications such as video conferencing, VoIP etc. demand performance guarantees. Despite a large volume of work, QoS has not been completely deployed in today's networks. A primary reason for this is the complexity of proposed QoS solutions and largely manual per-device configuration of QoS knobs by network administrators. Supporting QoS for services and applications requires a well-defined automated QoS control and network management mechanisms in order to maintain the requested QoS performance over a network. A QoS control mechanism should provide an automated but fine-grained control for flow configurations. Also, it should be adaptive to dynamic workloads for dynamic QoS configurations based on network states. Furthermore, it should support legacy devices and large-scale networks like WANs. In addition, it should provide network-wide optimization in resource allocation by utilizing a global view of the network.

In recent years, some emerging applications, such as distance learning, video conferencing and so on, are becoming prevalent in networking world. Despite the advantages of these QoS-dependent applications for users, they still suffer from some issues regarding QoS or QoE requests of their users/customers. Firstly, today's QoS based applications take into account only the network parameters as a QoS performance. However this approach does not reflect the user's real satisfaction of provided services. Secondly, even if the user's satisfaction, i.e. QoE, is provided, converting this QoE indicators to network-based QoS parameters is another issue. Also, this conversion

needs to be in a dynamic and optimized way. Thirdly, controlling and implementing QoS policies on the network is another issue for IPTV services.

An A-CPI enables applications to communicate with the controller to express their needs including dynamically specifying the QoS parameters of applications. Since they provide crucial tasks between applications and controller, network operators should consider certain points while designing A-CPIs. An A-CPI should be able to tolerate slow modifications of networks such as resource allocation for applications. It should also allow for determining the requirements beforehand using the application if possible. Defining different kinds of network parameters for different data types should be possible by an A-CPI. An interface should make sense for application developers while providing application metrics. A desired interface should not involve any application-related metrics such as response time. Instead, it should be able to convert these application-oriented metrics to network-based metrics such as delay, bandwidth etc.

3.13 Chapter Summary

Providing QoS is still a hot research problem in existing networking architectures. The emerging applications in the Internet (e.g. video streaming, VoIP etc.) generate diverse flows which require different treatments for each one. However, providing QoS needs of these flows is not easy with today's networking models. Therefore, researchers has started exploiting the SDN paradigm and OpenFlow protocol since they bring centralized global network view, and more fine-granular flow management opportunities in networks. These features of SDN make it a better candidate in order to provide QoS for applications in easier and more flexible ways compared to traditional network architectures. This survey study has made a picture of QoS in OpenFlow-enabled SDN networks by surveying the current QoS-motivated studies in the field. It has organized the related studies according to the categories that are the most prominent ways in which QoS can benefit from the concept of SDN: Multimedia

flows routing mechanisms, inter-domain routing mechanisms, resource reservation mechanisms, queue management and scheduling mechanisms, Quality of Experience (QoE)-aware mechanisms, network monitoring mechanisms, and other QoS-centric mechanisms. It has also outlined the potential challenges and open problems that need to be addressed further for better and complete QoS abilities in SDN/OpenFlow networks and lessons learned during preparation of this survey chapter.

4 A SCALABLE HIERARCHIC SDN ARCHITECTURE

4.1 Abstract

All new networking architectures come with their own problems. Software Defined Networking (SDN) has its own challenges which are needed to be addressed by researchers as well. One of the crucial problems with SDN is the control plane scalability since it is a bottleneck for its evolution. As the network grows, the number of messages a controller receives also increases. This increase puts the controller scalability in the heart of problems of SDN. This chapter proposes a hierarchy-based network architecture along with an inter-AS routing approach with QoS. It exploits idea of levels in which networks with controllers reside and a main controller, which works like a broker, is on top of them to keep the global network state and view. The experiment results indicate that a controller in a hierarchic setting handles 50% less number of traffic than a controller in a non-hierarchic environment.

4.2 Introduction

Traditional networking is forcing its limits to meet the needs of today's users, enterprises and carriers due to its limited capabilities. Configuration or installation of network devices and appliances requires more trained people and increases costs and take time to do so. Vendor dependency is an obstacle for network application developers and IT people to develop new types of network applications [1]. Increases in network applications, such as virtualization, cloud services as well as mobility and video content, requires more dynamic architectures of data centers, carriers or ISP networks.

SDN aims to handle above-mentioned drawbacks of today's networking architectures. It brings the idea of separation of data plane and forwarding plane along with a controller to acquire the global view of the network. Network managers become more capable of efficient manipulating of network resources. SDN makes management of the network easy for network operators/administrators by providing flexible programmability resulted from decoupled forwarding and data planes. Network managers can easily manage their network resources by dynamic, automated and easy-to-handle applications. OpenFlow [34, 36] is the first standard protocol for communication of separated forwarding plane devices/applications (e.g. controller) and data plane dump devices (e.g. routers, switches). It removes the vendor-dependency of data plane devices and make them able to communicate with all kinds of controllers.

SDN is an evolving networking architecture and has not completed its evolution. Scalability, as in all new networking architectures, in SDN is one of the most important challenges that will complete evolution of the SDN. As stated in [1], the scalability issue in SDN has not been focused by researchers as much as it deserves. Decoupling of data and forwarding planes is the most important reason to the scalability issue, particularly control plane scalability, since it requires management of data plane devices from a remote point (i.e. controller) and therefore control plane scalability becomes a focal point for the system. Also, as the number grows regarding the number of network devices such as routers, switches etc., the controller will need to handle more events and flow requests. This increase requires the control plane to be scalable with respect to the network size. In addition, the placement of the controller in a network has effect on the scalability of the control plane since the distance between controller and data plane devices introduces latency into the system [61]. There are some proposals to mitigate the control plane scalability issue of SDN in the literature. They are mainly categorized in either central controller-based [23,69] or distributed controller-based [66,68,72,82,83] solutions. Optimization techniques-oriented proposals are other types of the solutions. These proposals are discussed in more detail in Section 4.3. However, very few of them revolve around

a hierarchic controller-based solution. In this type of architecture, every domain has its own controller with their own network view and there is a Broker with a more global network view which orchestrates those controllers. This type of solution is more efficient and mitigates the issue of control plane scalability of an SDN network. That is the motivation for us to propose a hierarchy-based network architecture in this chapter.

In the remaining of the chapter, an extensive survey of studies, which aim to improve the scalability of the SDN with different techniques, in the literature is given in Section 4.3. Section 4.4 explains the details of the architecture and routing approach proposed along with its components. Section 4.5 clarifies how the proposed architecture works while Section 4.6 discusses the experiment results. Finally, Section 4.7 wraps the chapter up with concluding remarks.

4.3 Related Work

The proposed solutions to control plane scalability issue of an SDN network can be classified in two broad categories. First, control plane itself is a vicinity to solve the scalability of the network by means of some networking architectures. Second category aims to exploit some well-known optimization techniques in order to alleviate the foregoing issue in an SDN network.

As a solution to improve the scalability of an SDN network, networking topologies like central controller architecture and distributed networking are famous settings for SDN networks. Central controller centric settings utilize a central controller which is powered with global network view, applications and policies.

[69] proposes a central controller based network targeting enterprises networks. An Ethane network comprises of a controller which accounts for routing task and simple and dump switches to forward packets on controller commands. Ethane network performs five main functions: registration, bootstrapping, authentication, flow setup, and forwarding. While Ethane brings ease of use and can scale to large networks, it

might be cheated by means of MAC addresses to send packets, which is an open issue to be addressed.

NOX [23] is a network operating system which is more than just a controller platform for a network. A NOX network has switches, server(s) running NOX software acting like a controller along with applications atop, and a database to keep network view for applications. The network view consists of network topology, users, endpoints, etc.. As in most SDN controller platforms, NOX treats the packets based on the first packet of a flow traversing through the controller. This flow-based method helps increase the scalability of a network.

In distributed type of architectures, the controllers share the same global view and cope with the traffic locally and coming from neighbor domains. The purpose of a distributed networking is to reduce the load on the central controller and avoid the failure of the central controller.

In HyperFlow [72], local controllers are utilized to serve all requests for their own remote sites and thereby flow setup times and flow initiation rates drop by the time. HyperFlow is logically centralized albeit its distributed architecture is an event-based control plane for OpenFlow and is actually implemented as a NOX application. Its main tasks are: global network view synchronization between controllers, communication with switches controlled by another controller from a different site, and managing responses coming from switches in other sites to the request-originator controllers.

[82] introduces a distributed cluster-based controller architecture to retain the communication and coordination between controllers to obtain a more scalable network. This cluster-based architecture brings flexibility to the network regarding adding or removing controllers since it does not bother network applications. In the proposed network, each switch is associated with a single controller via an IP network.

[77] proposes Onix, a distributed control platform, in responses to lack of a control platform which provides consistent network state and global network view for network devices and applications. Onix instances propagate network states to other instances

to be able to scale large networks. The authors follow three approaches for a better scalability in Onix architecture; (1) work partitioning by applications for less work on instances, (2) cluster aggregation for a hierarchical structure, and (3) consistency and durability of the network states for applications.

The architecture called ElastiCon in [68] aims to evenly distribute load in controllers by a controller pool since they may not be loaded equally due to static configuration. The proposed architecture has two main mechanisms to dynamically shift the workload across the controllers in a pool which is dynamically expanding or shrinking. Their framework considers the total load on controllers and may add or remove controllers based on the load exceeding or falling down a threshold. It may also move switches from one controller to another if the load on a controller is more than a pre-defined threshold.

Kandoo [73] focuses on scaling controller by decreasing the number of frequent events on the control plane since these events bring more overhead than others to the controller plane. Kandoo's setting is similar to this framework proposed but there are two differences. First, in Kandoo, the local controllers are handling less number of switches than the local controllers proposed in the setting proposed in this chapter. Second, routing with Quality of Service (QoS) for connection requests is considered in the architecture proposed. Users may specify their requested QoS values when they ask for a connection. On the other hand, the Orion [79] exploits a hybrid hierarchical architecture along with a routing method for large-scale networks in which a large size domain, managed by one administrator, is divided into sub-domains. The proposed framework differs from the Orion since routing with QoS for both intra-domain and inter-AS traffic is considered while the Orion considers for only intra-domain traffic.

Optimization-based designs aim to empower the controller performance so that it can handle more packet flows per second and reduce the latency and overhead. They achieve this by exploiting some techniques like parallelism, multi-threaded designs, batching, efficient routing decisions.

In [75], Maestro is proposed to increase scalability in SDN networks by empowering multi-core architecture to leverage the parallelism in order to increase controller speed along with hassle-free programming model for application writers. Maestro is designed to partition the workload evenly available in threads in cores to increase the performance (i.e throughput) by keeping all processor cores busy by means of "pull" fashion instead of "push" fashion. Maestro balances the memory consumption by keeping no data between stages in CPU cores' threads.

Beacon [25] is reinforced for a high performance by multi-threaded designs; "Shared Queue" and "Run-To-Completion". In "Shared Queue" design, each switch is connected to single I/O thread which reads the messages from the switches and then send to a shared queue. The pipeline threads take the message from the shared queue in order to process by corresponding applications. In the "Run-To-Completion" design, on the other hand, there is no pipeline threads and each message is processed by I/O threads.

[85,86] propose source routing based routing schema in SDN. They aim to reduce the number of events processed by the controller because each flow installation in hops across the path will create an event and make the controller busy. The routing schema in [85] and this chapter's are similar with respect to considering QoS for requests. However, the proposed setting is hierarchic and considers inter-AS traffic as well albeit they consider the traffic in a single domain.

4.4 Scalable Hierarchic Architecture

In this section, a scalable hierarchy-based proposal is presented along with its components in details. The proposed architecture consists of levels from bottom to up. The levels can be increased, as they are needed, through up. In this version of the framework, there are currently two level: Network-Level (bottom-level) consisting of independent domains/ISPs/ASes(Autonomous system) which are also SDN domains with their own local controllers, and Broker-level (up-level) consisting of a super

controller acting like a supervisor for the bottom-level controllers. These components are explained in corresponding subsections below.

4.4.1 Advertisement

This section explains how reachable addresses and available path advertisements are carried out in the proposed architecture. Every AS domain controller advertises their reachable addresses information (a method for this advertisement is out of this chapter's scope) only to the Broker controller so that when a request comes to the Broker, it will be able to determine the source and destination ASes.

Table 4.1.: All possible paths between R5 – R7 in AS2

Path ID	**Path Details**	**QoS Values (B, D)**
P21	R5 – R7	(28, 3)
P22	R5 – R8 – R7	(19, 8)
P23	R5 – R6 – R8 – R7	(7, 7)

Every local controller will advertise its border switches/routers (entering or exiting points) to the Broker with neighbor connectivity information (through which border router/switch) as well as inter-connecting links so that the Broker will know which AS is connected to which AS by which border node. Every AS controller will calculate all possible paths between their own border nodes (entering and exit points), e.g. in AS2, paths between R5 – R7, R5 − R8, and R7 − R8 (assuming links are bidirectional), with the corresponding QoS parameters. Then they make these QoS values a tuple as in the following:

$$\text{Tuple1} \rightarrow (\text{Available Bandwidth, Delay, Jitter, } \cdots)$$

For example: All possible paths between R5 − R7 in AS2 are like in the Table 4.1:

All possible path advertisements from other ASes are as in the Table 4.2. Here, the QoS values shown in Table 4.1 and 4.2 are for representation purpose. In other

Table 4.2.: All possible paths (advertisements) from all ASes.

Path ID	**Path Details**	**QoS Values (B, D)**
P11	R1 – R2 – R4	(10, 2)
P12	R1 – R2 – R3 – R4	(7, 4)
P13	R1 – R3 – R4	(11, 4)
P14	R5 – R7	(28, 3)
P14	R1 – R3 – R2 – R4	(7, 6)
P21	R5 – R7	(28, 3)
P22	R5 – R8 – R7	(19, 8)
P23	R5 – R6 – R8 – R7	(7, 7)
P24	R5 – R8	(21, 4)
P25	R5 – R6 – R8	(7, 3)
P26	R5 – R7 – R8	(19, 7)
P27	R7 – R8	(19, 4)
P28	R7 – R5 – R8	(21, 7)
P29	R7 – R5 – R6 – R8	(7, 6)
P31	R10 – R12	(8, 1)
P32	R10 – R9 – R2	(12, 10)
P33	R10 – R9 – R11 – R12	(16, 10)
P41	R13 – R14	(18, 4)
P42	R13 – R15 – R14	(20, 6)
P43	R13 – R15 – R16 – R14	(15, 7)
P44	R13 – R14 – R16	(16, 6)
P45	R13 – R14 – R15 – R16	(15, 11)
P46	R13 – R15 – R16	(15, 5)
P47	R13 – R15 – R14 – R16	(16, 8)
P48	R14 – R16	(16, 2)
P49	R14 – R15 – R16	(15, 7)
P410	R14 – R13 – R15 – R16	(15, 9)

words, certain bandwidth and delay values are assigned to the links between nodes by manually and the numbers shown in the corresponding tables and figures are calculated by concave feature of bandwidth (i.e. minimum) and additive feature of delay for the single paths.

4.4.2 Domain Controllers: Bottom Level

In the bottom level of the architecture, there are independent domains/ISPs/ASes with their own controllers (local). These networks are also SDN networks and operate independent of each other. Their controllers have all required applications and services to make their own decisions for local traffic flows.

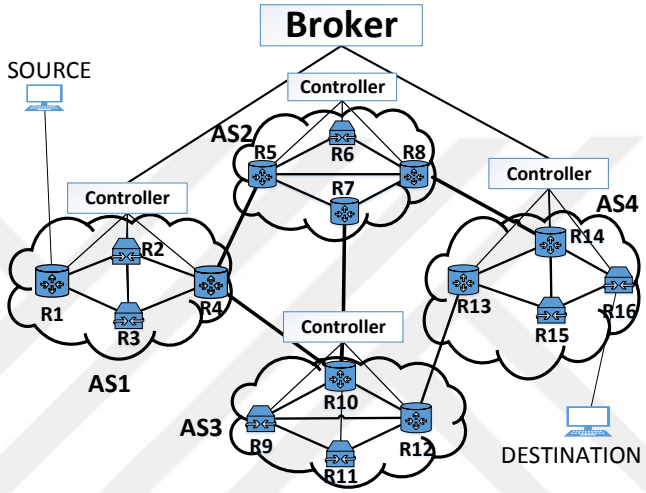


Figure 4.1.: An overview of the hierarchical network with multi-ASes. Every network has its controller and the Broker controller reside on top of them with connections to all of the bottom-level controllers.

The controllers are not connected to each other, thereby not able to communicate with other network controllers. Networks are connected to each other through interconnecting links between their entering/exiting border nodes in the data plane and are not necessarily connected to every other ASes as in today's Internet architecture. As shown in Fig. 4.1, AS1, AS2, AS3, and AS4 form the bottom-level of the architecture. AS1 is, for example, connected to AS2 and AS3 although AS2 has connection to all other ASes. R4 is an entering/exiting border node for AS1 and similiarly, R5, R7, and R8 are for AS2. The same idea of hierarchy can be applied in each ASes as the AS network size grows and is needed. Therefore, one can have many levels in this architecture.

4.4.3 Broker: Up Level

The Broker resides in the up level and acts like a super controller for all bottom level AS controllers. All the local controllers are connected to the Broker. They can communicate with the Broker in both direction. The Broker is responsible for finding an end-to-end route from source to destination specified in the request coming to the Broker. When a host wants to send a flow to a destination from another AS, the source AS controller will communicate to the Broker so that it will calculate an end-to-end route for the flow.

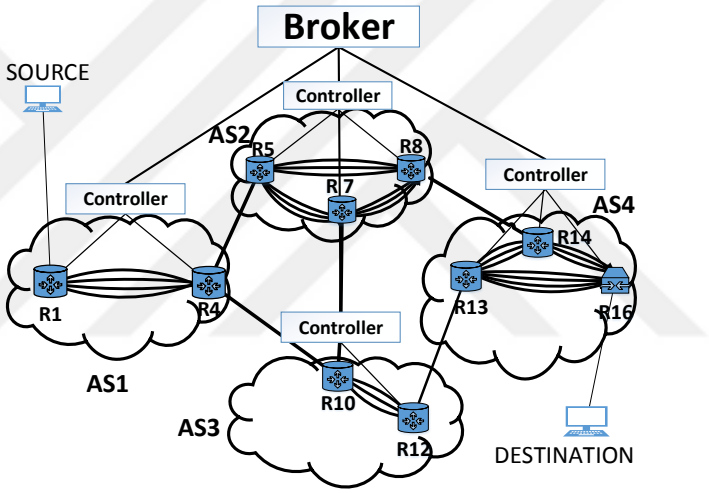


Figure 4.2.: Path level view of the Broker. It represents the available paths satisfying the requested QoS values between border nodes in an AS.

The paths in Table 4.1 will be advertised to the Broker in a file (one file for each border node pair) without the path details information, just (PathID, QoS). Therefore, the details of the path is hidden inside an AS. An AS controller sends just PathID and corresponding QoS values and hence the Broker will know that there are, for example, 3 paths from R5 to R7 with corresponding QoS values. The procedure in 4.4.1 is applied for all border node pairs in all ASes (as in Fig. 4.1) and the advertisements are obtained as in the Table 4.2 since it is needed for using an AS as a transit way. The Broker should know this so as to which ASes the flow should go

through. An update advertisement will be send to the Broker in case of any change regarding QoS values in the above information.

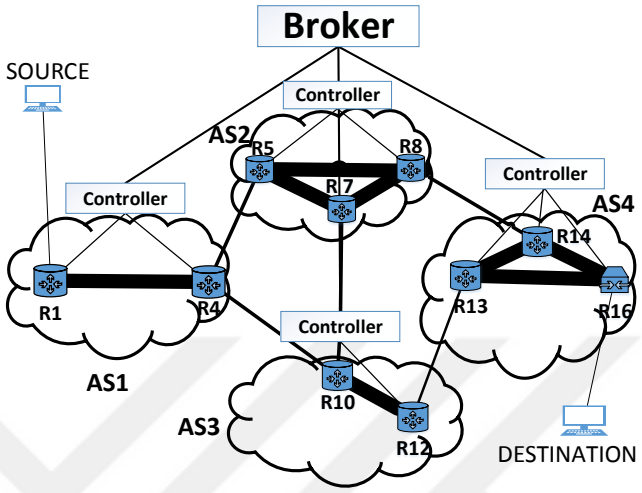


Figure 4.3.: AS level view of the Broker. It represents the available ASes satisfying the requested QoS values.

The Broker will also know the QoS values and connection information of inter-connecting links between ASes since the local controllers will advertise their connectivity information (neighbor ASes that they connect to) to the Broker so that the Broker will know which AS is connected to which one. After all of these advertisements, the Broker's view will be like in Fig. 4.2 when it gets a connection request with certain QoS values. It will just see the paths satisfying (eliminating the rest) the requested QoS values between border node pairs in ASes in case of an end-to-end path request comes to the Broker.

The total advertisements (files) from each AS will be $b_n * (b_n - 1)/2$ where b_n is the number of border nodes in an AS (all paths between a border node pair is advertised in a single file). For instance, for AS2 with 3 border nodes, there will be $3 * (3 - 1)/2 = 3$ advertisement files. Based on all these paths with QoS information between border node pairs from all ASes and inter-connecting link connection and their QoS values, the Broker will have a complete global network view, as in the Fig. 4.3, based on border node pairs from ASes.

4.5 How the Architecture Works

This section briefly explains how the proposed architecture works in an hierarchic environment.

When a connection request comes to a local controller (e.g. AS1's controller), it checks if the destination is from its domain. If so, then it handles the routing procedure locally based on its routing table.

If the destination is from another domain, then it will inform the Broker about the destination and QoS values in the request. The Broker will know the AS of the destination because of the advertised reachability addresses by all ASes. Then it will ask the source AS controller to calculate QoS paths advertisements between source switch/router and every border nodes (exit points) as indicated in the Section 4.4.1.

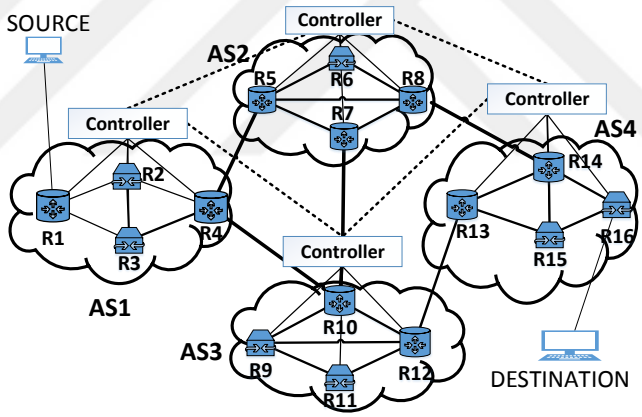


Figure 4.4.: A Distributed SDN Architecture without hierarchy. Each controller is connected to its neighbor controllers.

The Broker will also ask the destination AS to calculate QoS path advertisements between every border nodes (entering points) and the switch/router that destination host is connected to. These advertisements will be sent to the Broker. Here, it should be noted that these advertisements are different from the ones in Section 4.4.1 because these paths are from source router/switch to exiting border nodes, not between any border node pairs as in the Section 4.4.1. Based on these advertisements, other advertisements from other ASes (as in Table 4.2) and inter-connecting links'

QoS values, the Broker will have border node-level and AS-level views, as in the Fig. 4.2 and Fig. 4.3, respectively, and will determine the best routes meeting the requirements of the request. Upon calculating the best route for the request, the Broker will send the entering (ingress) and exiting (egress) border node points (switches) along with corresponding ports to corresponding ASes. Then the Broker will ask every AS controller across the path (including source AS, destination AS and transit ASes) to reserve the required QoS values. Upon getting the confirmation from them, the flow starts traveling.

4.6 Evaluation

This section analyzes the number of network events processed in the control planes (i.e. controllers) of each ASes in the proposed hierarchic architecture and as well as non-hierarchic (distributed, as shown in Fig. 4.4) architecture. For the distributed setting, state sharing and not-sharing case are considered in network setup phases. The following metric have been chosen to assess the scalability performance of the proposed hierarchic SDN architecture since the goal is to reduce the number of messages the controllers exchanges.

- *Number of Messages per Control Plane* - measures the number of messages handled per control plane in proposed hierarchic and distributed architecture.

In Fig. 4.5, the number of messages the Broker exchanges is not same with the number of connections it handles. So there is not a linear relation between the number of messages and number of connections at Broker. This happens because for each connection request from source to destination, the Broker calculates an optimal path. This optimal path may include more than 2 ASes (at least 2 ASes including source AS and destination AS). The flow (actually any flow) will go through at least 2 ASes (transiting 2 ASes). Therefore, the Broker will exchange 1 message for each transit AS controller on the path to set the path and make the reservation. Hence, the number of messages the Broker exchanges depend not only number of connections it

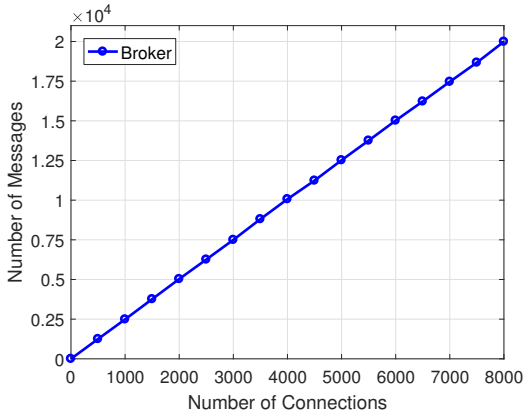


Figure 4.5.: Number Connections vs. Number of Messages at Broker in hierarchy. There is no sharing of state between controllers of ASes.

handles but also the number of ASes each path includes for each flow. For example, for a request from the source (in AS1) and destination (in AS4), the flow may go through AS1 – AS2 – AS4 which requires the Broker to exchange 3 messages with corresponding controllers (one for each).

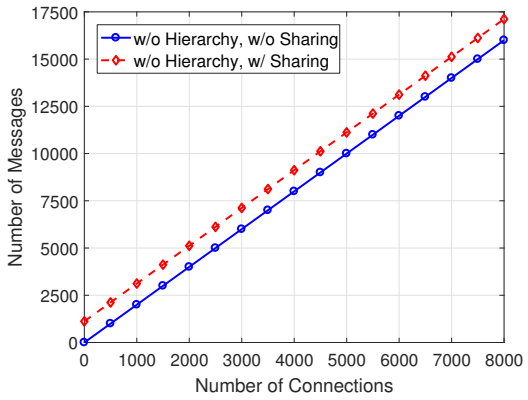


Figure 4.6.: Number of Connections vs. Number of Messages at controller of AS1. In this architecture, there is no hierarchy. There are two cases: 1) State sharing with each other controller, 2) No state sharing with each other controller.

Fig. 4.6 shows how the AS1 controller messages are affected in the case of no hierarchy, i.e. there is no Broker on top of the AS controllers. The architecture would be similar to the hierarchic setting. The only difference is not that there is Broker over the AS controllers and the AS controllers are connected to only their neighbor

AS controllers as in the Fig. 4.4. That setting is in distributed fashion and two cases have been considered: 1) w/o state sharing, 2) w/ state sharing. In the first case (i.e. w/o state sharing), the AS controllers do not exchange their states such as reachable network addresses in advance. Therefore they do not know anything about each other. They are able to communicate with each other in case of a connection request to out of their domains (i.e. inter AS traffic). When a connection request comes to the controller, it will talk to their neighboring AS controllers to check whether they are able to send the request to the destination. For example, as in the Fig. 4.4, the controller of AS1 is only connected to controllers of AS2 and AS3.

When a source from AS1 wants get a connection to a destination from AS4, the AS1 controller will just exchange messages with neighbor AS controller (i.e. A2 and AS3). In this case, the number of messages will be 2 times of number connections. In the second case (i.e. w/ state sharing), the controllers share some of their network states as indicated before (e.g. reachable network addresses). This state sharing brings the extra message exchanges for each controller in advance. Handling with a connection request is the same procedure as explained in the case of no sharing. They still need to talk to their neighbor AS controllers if they can send the request to the destination. As shown in the Fig. 4.6, there will be some number of messages that AS1 controller has already exchanged with others beforehand in case of sharing setting although there is no connection request.

Fig. 4.7 shows the comparison of the number of messages at controller of AS with hierarchy and without hierarchy. In case of an hierarchic architecture, the bottom level controllers like AS1 controller will not handle with their inter-AS traffic connection requests. If the destination in the request is from another AS, then the controller will forward the request to the Broker so that it calculate an optimal path for it since it has the global view of the entire architecture. Therefore, the AS1's controller exchanges only 1 message, which is forwarding it to the Broker. Hence, there is a one-to-one ratio between number of connection and number of messages the AS1's controller exchanges in a hierarchic architecture. The without hierarchy

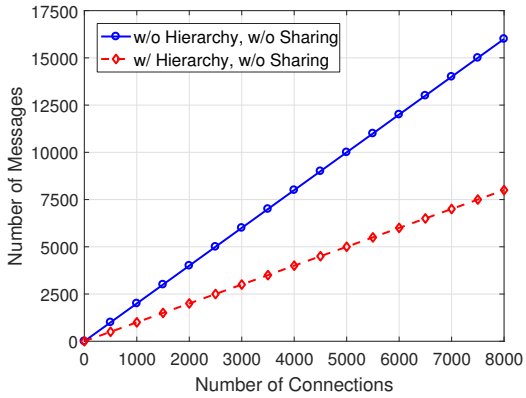


Figure 4.7.: Number of Connections vs. Number of Messages at controller of AS1 in the hierarchic and non-hierarchic architectures. There is no state sharing in each case.

case is the same with case explained above and there is no sharing. As shown in the Fig. 4.7, the number messages that AS1's controller exchange in the no hierarchy case is 2 times of the hierarchy case. A hierarchic architecture reduces 50% of the number of messages that a controller handles.

4.7 Chapter Summary

This chapter have presented a hierarchical SDN architecture and inter-AS QoS based routing approach. The purpose of the proposed architecture is to improve the scalability of the control plane (i.e. controller) in an SDN network by reducing the number of messages that a controller deals with. The experiment results have shown that a network controller will handle 50% less messages for inter-AS traffic in a hierarchic environment compared to non-hierarchic environment since they do not need to keep global network view and synchronize with other states. This situation reduces the number of messages although the number of connections increase in a network.

5 MEASURING SCALABILITY IN SDN

5.1 Abstract

SDN architecture promises to mitigate limitations of traditional networking architectures in order to satisfy today's complex networking needs. However, as all new networking architectures, SDN also presents several inevitable technical challenges to be addressed by researchers. Control plane scalability is one of the crucial issues deserving more attention from both academia and industry in SDN as well. There are many existing solutions proposing a way to alleviate the control plane scalability in SDN. However, one prominent common ground they share is that they measure the control plane scalability performance in terms of typical network QoS parameters such as throughput and latency. Although these metrics may be a good performance indicators for quality of service measurement in mid-term and long-term, they may not reflect real scalability performance of control planes in SDN environments. However, a metric for scalability of control plane in SDN can provide network administrators some insights while they construct their SDN networks. This chapter firstly explore the roots of control plane scalability problem in SDN as well as proposed existing solutions. A metric is then proposed in order to evaluate the control plane scalability in SDN. This chapter also gives mathematical models of the proposed metric over different control plane designs. Furthermore, the performance of these control plane designs is compared by extensive experiments.

5.2 Introduction

Traditional networks have reached their architectural limitations. Increasing cloud services, server virtualization, sharp growth of mobility and content-like video have

led researchers to rethink today's network architectures. In traditional architectures, network devices and appliances are complex and difficult for (re)configuration and (re)installation since they require highly skilled persons. Adding or moving a device from a network requires extra costs. It is also time-consuming because IT people need to deal with multiple switches, routers, etc. and update ACLs, VLANs and some other mechanisms. Furthermore, as business demands or user needs increase day by day, application developers, carriers, and enterprises need to delve into evolving new services and facilities. However, vendor dependency is an obstacle deterring them from developing new networking applications and services for their networks due to slow equipment product cycle, application testing and deployment. Therefore, data centers, carriers, and campuses need more dynamic architectures today.

SDN architecture has emerged in response to aforementioned limitations of traditional networking architectures. SDN aims to decouple the controller plane and data plane. This separation provides network operators/administrators efficient use of network resources and eases provisioning of resources. Also, SDN brings ease of programmability to change the characteristics of whole networks. This simplifies the management of the network since it is decoupled from the data plane. Therefore, network operators can easily and quickly manage, configure, and optimize network resources with dynamic, automated and proprietary-free programs written by themselves in SDN architecture. In addition, since network is logically centralized in SDN, controllers have a global visibility of the whole network unlike conventional networking. Hence, they can dynamically optimize flow-management and resources.

However, SDN also presents several technical challenges. Sezer et. al [1] states that these challenges can be classified in four different categories. The first one is how to deal with high-performance packet processing in a flexible/programmable manner. The second is interoperability or standardization that needs to be addressed in SDN infrastructure. The third is security issues in SDN. The final category is the scalability issue in SDN, which especially needs more attention by researchers.

Scalability proposals in SDN study the problem in terms of improving the scalability of control plane with respect to some network metrics such as throughput and latency and do not propose a metric to quantify the scalability [51, 59]. However, a metric for scalability of control plane in SDN can provide network administrators some insights while they construct their SDN network. This chapter firstly explores the roots of control plane scalability problem in SDN as well as proposed existing solutions. A metric is then proposed in order to evaluate the control plane scalability in SDN. This chapter also gives mathematical models of the proposed metric over different control plane designs. Furthermore, the performance of these control plane designs is compared by extensive experiments.

In the remaining of the chapter, the chapter digs out the roots of SDN control plane scalability issues and presents some existing solutions alleviating the problems in Section 5.3. In addition, the chapter gives a snapshot of several research attempts proposing a scalability metric to measure the scalability of systems. Section 5.4 describes the proposed scalability metric in a general perspective. In Section 5.5, the chapter models the metric by a mathematical methods over different SDN control plane designs throughout in corresponding subsections. After discussing the experimental results in Section 5.6, the chapter is summarized with concluding remarks in Section 5.7.

5.3 Control Plane Scalability in SDN

This section points out the main reasons that make control plane a scalability bottleneck and present some existing solutions alleviating the control plane scalability issue in SDN. In addition, this section exhibits several research attempts proposing a scalability metric to quantify the scalability of distributed systems. Below are some of the main reasons that make control plane a scalability bottleneck:

- Separation of Control Plane and Data Plane: This decoupling requires management of network devices from a remote controlling mechanism (i.e. software).

Separation of these planes may result in significant signaling overhead depending on the network type (e.g. distributed, hierarchical etc.) and applications on top of the controller.

- Quantity of Events/Requests Handled by Controller: As the network grows with respect to the size of the network elements, the controller will have to cope with more events and flow requests. Therefore, the number of control messages sent by data plane devices to controller(s) becomes one point to be addressed because the controller may not be able to handle all the incoming requests [59].

- Propagation and Processing Delay of/in Controller: The controller's placement (distance between network devices and controller) is one factor that introduces latency into the system along with controller processing power and communication among controllers, which affects the control plane scalability as well. [62] outlines a comprehensive analytical model for the behavior of a scalable SDN deployment regarding boundary performance of event processing delay and buffer space of SDN controllers by means of network calculus as a mathematical framework.

5.3.1 Existing Solutions for Control Plane Scalability in SDN

The existing solutions to the control plane scalability issue of an SDN network can be classified in three broad categories. First category is the control plane topology-oriented solutions such as single controller design, distributed controllers design, and hierarchical controller designs. Second category aims to exploit optimization techniques in order to alleviate the foregoing issue in SDN networks. Finally, some sate-of-the-art solutions are centered around the data plane of the SDN network by giving some limited control back to switches over flows.

In [69], the authors propose an architecture called "Ethane" which enables network managers to define policies and flow entries. There are three concerns that the authors address and resolve in this architecture. First, Ethane renders that high-level policies

become the authority part to control the network. Second, the packet paths are managed by policies in order to have better control and global network view. Third, the Ethane network requires a precise binding between a packet and its origin to be able to identify where the packet coming is from.

HyperFlow [72] is logically centralized albeit its distributed architecture is an event-based control plane for OpenFlow [34]. In HyperFlow, the authors exploit local controllers, serving all requests for their own remote sites, due to an increase in the flow setup times and flow initiation rates.

Kandoo [73] focuses on scaling a controller by decreasing the number of frequent events on the control plane since these events bring more overhead than others to the controller plane.

[84] proposes a hierarchy-based network architecture along with an inter-AS routing approach with QoS. The authors use an idea of levels in which networks with controllers reside on top. There is also a main controller which works like a broker on top of networks to keep the global network state and view.

In [75], a controller system called "Maestro" is proposed to increase scalability in SDN networks. Maestro uses a multi-core architecture to leverage the parallelism in order to increase controller speed along with a hassle-free programming model for application writers. Maestro uses the batching of packets to individual destinations to improve processing and communication efficiency besides multi-threading structure.

DIFANE [65] is an architecture that preserves traffic in the data plane through managing packets in switches called "Authority Switches". These authority switches keep OpenFlow rules. DIFANE is motivated by minimizing the number of packets traveling in the control plane since this journey becomes a bottleneck for the scalability issue.

5.3.2 Scalability Metric Proposals

There are several research efforts proposing a metric to measure scalability of systems. However, these metrics attempt measuring scalability performance of algorithms, parallel machines, and distributed systems. Also, most of these metrics are for homogeneous environments. Majority of these proposals revolve around two major types of scalability metrics: Isospeed scalability and Isoefficiency scalability.

The Isospeed scalability is characterized by the fact that an achieved average unit speed of an algorithm on a given machine can remain constant with increasing number of processors and problem size for an algorithm-machine combination [44]. In [45], the authors presents a metric to describe the scalability of an algorithm-machine combination in homogeneous environments. Their scalability function is defined as $\psi(p, p') = \frac{p'W}{pW'}$ where p and p' are the initial and scaled number of processors of the systems respectively, and W and W' are the initial and scaled problem size (workload) respectively.

The Isoefficiency scalability is described as the ability of parallel machine to keep the parallel efficiency constant when the system and problem size increase [46]. The parallel efficiency is defined as speedup over the number of processors, i.e. $E = \frac{S}{p}$. Speedup is also given by the ratio of problem size (W) and parallel execution time (T_p), i.e. $S = \frac{W}{T_p}$ where $T_p = \frac{W+T_0(W,p)}{p}$ with $T_0(W, p)$ extra communication overhead [47].

[48] defines heterogeneous scalability by presenting a heterogeneous efficiency function. They attempt to extend the homogeneous Isoefficiency scalability model to heterogeneous computing and, therefore, their work inherits the limitation of parallel speedup, requiring the measurement of solving large-scale problem on single node. [49] proposes a scalability metric called Isospeed-efficiency for general heterogeneous computing systems. This metric combines the roots of both Isospeed scalability and Isoefficiency scalability metrics by means of a concept called "Marked Speed" to describe the computing power for a stand-alone node and a combined computing system.

The work in [56] is the closest to the work presented in this chapter because it presents a metric for SDN control plane scalability. They apply the proposed metric in three typical control plane designs in SDN. Their work borrows the idea presented in [50] for distributed systems. They use the scalability metric, which is based on productivity of a distributed system, presented in [50] to quantify the scalability of SDN control plane by adapting to the SDN case. The work in this chapter is different than [56] in a way that, this work assumes that the QoS (i.e. average flow processing time in SDN case) is the same for a network when scaled from N_1 to N_2. Also, this scheme does not depend on cost of the system.

5.4 Scalability Metric

In this work, it is assumed that there are total of s switches, regardless of total number of local domains in a network setting such as distributed and hierarchical, in each different network settings. It is also assumed that there are h hosts connected to every single switch in all settings. Therefore, the total number of hosts is $\mathcal{H} = s \times h$ in each network architecture type. The matrix below specifies the data traffic between hosts in network setting:

$$\begin{array}{c} \text{i/j} \\ 1 \\ 2 \\ \vdots \\ sh \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & & 2 & \cdots & h & \cdots & 2h & \cdots & sh \end{array} \\ \begin{pmatrix} 0 & \lambda_{1,2} & \cdots & \lambda_{1,h} & \cdots & \lambda_{1,2h} & \cdots & \lambda_{1,sh} \\ \lambda_{2,1} & 0 & \cdots & \lambda_{2,h} & \cdots & \lambda_{2,2h} & \cdots & \lambda_{2,sh} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ \lambda_{sh,1} & \lambda_{sh,2} & \cdots & \lambda_{sh,h} & \cdots & \lambda_{sh,2h} & \cdots & 0 \end{pmatrix} \end{array}$$

where $\lambda_{i,j}$ is the flow sending rate from host i destined to host j, and $i, j = 1, 2, ..., sh$. It is assumed that these random $\lambda_{i,j}$ variables are independent identically and have the Poisson distribution.

This work defines the scalability metric for a network as the ratio of workload (W) over overhead (O) as in Eg. 5.1. The overhead refers to the number of messages

processed in the control plane by a controller(s) and workload to the number of flows entering the network through the data plane. The whole network is considered a black-box and is therefore assumed that any flow entering the network is part of the workload.

$$Scalability = f(W, O) = \frac{W}{O} \tag{5.1}$$

In SDN, when the first packet of a new flow enters a network through a switch, the switch starts a flow initiation request if there is no rule entry matching the packet in switch's flow table. This flow initiation request is then sent to the controller. The controller processes it and installs a rule for the flow in switches over the path calculated by the controller. Therefore, a rule-missing flow results in some other control-messages which are created, processed, and sent by a switch and/or controller. Also, a controller may deal with some other periodic messages, such as statistics, generated in the network but not related to rule installation process. These types of messages are categorized as a overhead message in this work. These overhead messages keep a controller busy and thereby cause the control plane (i.e. controller) to degrade scalability performance regarding some network QoS results such as throughput and processing delay. Therefore, scalability performance of a control plane depends on overhead resulted in the control plane. If these messages can be reduced, the scalability performance of a control plane upgrades with respect to workload.

5.5 Modeling of the Scalability Metric over Different Control Plane Designs in SDN

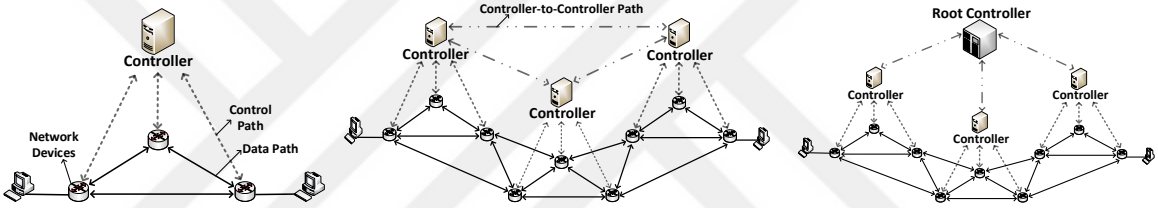
In an SDN network with OpenFlow protocol, there are three different types of messages between a controller and data plane devices: Controller-to-Switch, Asynchronous, and Symmetric messages. Each of these message type has its sub-types as well. Controller-to-Switch messages, Asynchronous messages, Symmetric messages are represented by α, β, and γ, respectively, throughout the chapter.

- Controller-to-Switch: These messages are initiated by the controller and may or not require a response from the switch. "features_request", "features_reply",

"set_configuration", and "flow_mod" messages are some main prevalent sub-types of this message type.

- Asynchronous: Switches send controllers asynchronous messages to denote a packet arrival or switch state change. "packet_in" and "flow_removed" messages are some most-used sub-message types of "Asynchronous" messages.

- Symmetric: Symmetric messages are sent without solicitation from a controller or switch, in either direction. "hello" and "echo" messages are prevalent sub-types of "Symmetric" messages.



(a) Centralized (Single) Control Plane (CCP) Design

(b) Distributed Control Plane (DCP) Design

(c) Hierarchical Control Plane (HCP) Design

Figure 5.1.: A representational overview of popular SDN control plane models. The two-sided solid, dashed, and dashed-dotted arrows represent two-way data path among network devices, control path between controller and data plane devices, and controller-to-controller path among controllers, respectively. In 5.1a (CCP), there is one main controller with global network state. In 5.1b (DCP), every controller is responsible for different sub-domains of the network(s) with partially shared network view. In 5.1c (HCP), there are layers where controllers reside and are responsible for different sites (sub-domains) and a master (or Root) controller on top with global network view for global applications like routing. The data plane and control plane topologies shown in this figure are just representative purposes.

This work characterizes the overhead in a control plane, i.e. a controller, as the function of these three message types of OpenFlow protocol, synchronization messages, and messages for global flows going through a domain because a controller is occupied with processing these messages. An architecture setting generating less of these messages results in a better scalability performance compared to the one

with more of these messages. Therefore, the number of overhead messages can be quantified as below:

$$\begin{aligned} O &= f(\alpha, \beta, \gamma, \omega, \Omega, \Psi) \\ &= \sum_{k=1}^{s} (\alpha_k + \beta_k + \gamma_k + \omega_k) + \Omega + \Psi \end{aligned} \tag{5.2}$$

The overhead function is formulated in each network setting in corresponding sections. The metric mainly measures the overhead messages based on "Controller-to-Switch", "Asynchronous", and "Symmetric" messages sent/processed from/at a controller with respect to a switch. Each of these message types for different control plane designs, shown in Fig. 5.1, is modeled in corresponding sections with respect to their prevalent sub-message types. The main sub-messages, for example, for "Controller-to-Switch" messages are δ and θ messages. For "Asynchronous" messages, they are σ and ρ sub-message types sent from a switch. Finally, for "Symmetric" messages, ϵ messages are the ones which are mostly sent between a controller and a switch. Further, the synchronization messages (ω and Ω) between domain controllers to obtain a fully or partly global view bring also overhead messages to be processed at a controller in distributed or hierarchical settings. Lastly, a flow originated from another domain and going through a certain domain (say n-th domain) also brings some burden (Ψ) to the (n-th) controller. Table 5.1 gives the associated notations used in the chapter for each of these message types. It is noted that there is no distinction between ${}_l\rho_k$ and ${}_g\rho_k$ in CCP setting since there is only one domain. Therefore, using just ρ_k is fine.

The workload model that is proposed in this work depends on both the number of hosts in the network and the flow sending rate between hosts. The whole network is considered a black-box and is therefore assumed that any flow entering the network is part of the workload.

$$W = f(\mathcal{H}, \lambda) = \sum_{i=1}^{\mathcal{H}} \sum_{j=1}^{\mathcal{H}} \lambda_{ij} \tag{5.3}$$

Table 5.1.: Definition of notations used in the chapter.

Notation	Meaning
s_k	Switch k
α_k	Number of *Controller-to-Switch* messages sent from a controller to s_k
β_k	Number of *Asynchronous* messages sent from s_k to a controller
γ_k	Number of *Symmetric* messages sent between a controller and s_k
δ_k	Number of *stat_requests* messages periodically sent from a controller to s_k
θ_k	Number of *flow_mod* messages sent from a controller to s_k
σ_k	Number of *flow_removed* messages sent from s_k to a controller
${}_l\rho_k$	Number of *packet_in* messages sent from s_k to a controller for local flows
${}_g\rho_k$	Number of *packet_in* messages sent from s_k to a controller for global flows
ϵ_k	Number of *echo* messages sent between a controller and s_k
ω_k	Number of synchronization messages sent by a controller to neighbor domain controllers for global flows of s_k
${}^c r$	Average number of switches over an end-to-end (e2e) path in CCP setting
${}^d r_n^l$	Average number of switches over an e2e path in n-th domain in DCP-LV setting
${}^d r_n^g$	Average number of switches over an e2e path in n-th domain in DCP-GV setting
${}^H r_n$	Average number of switches over an e2e path in n-th domain in HCP setting
${}^d\Omega_n^l$	Number of synchronization messages exchanged between n-th controller and its neighbor controllers for global flows going through them in DCP-LV setting
${}^d\Omega_n^g$	Number of synchronization messages periodically exchanged between n-th controller and other controllers in a unit time to obtain a global view of networks in DCP-GV setting
${}^H\Omega_n$	Number of synchronization messages exchanged between n-th domain controller and master controller for global flows originating from other domains and going through n-th domain in HCP setting.
${}^d\Psi_n^l$	Number of global flows originating from other domains and going through n-th domain in DCP-LV setting
${}^d\Psi_n^g$	Number of global flows originating from other domains and going through n-th domain in DCP-GV setting
${}^H\Psi_n$	Number of global flows originating from other domains and going through n-th domain in HCP setting
${}^d N_n^l$	Average number of neighbor domains of n-th domain in DCP-LV setting

This workload model does not change in any of different network designs unlike the overhead model because the total number of hosts ($\mathcal{H}$) and flow sending rate (λ) between hosts are the same in each network settings.

5.5.1 Centralized Control Plane (CCP) Design

This type of architecture setting revolves around a centralized single controller [23, 69] with a global network view. The design of this architecture is simple and it is easy to manage the network. However, it is not efficient to handle the burden of environments such as data centers and large-scale networks [60]. The overhead is defined in CCP design as following:

$$\begin{aligned} O &= f(\alpha, \beta, \gamma, \omega, \Omega, \Psi) \\ &= \sum_{k=1}^{s} (\alpha_k + \beta_k + \gamma_k + \omega_k) + \Omega + \Psi \end{aligned} \tag{5.4}$$

In the CCP design, there is no any kinds of synchronization messages (i.e. $\omega_k = 0$ where $k = 1, 2, \cdots, s$ and $\Omega = 0$). Further, because there is only one single centralized domain, there is no case in which a domain can route a flow through another domain (i.e. $\Psi = 0$). Every flow is just handled in one domain based on the global view of the controller. Therefore, these variables are zero and have no impact over the overhead in the CCP design. Since a CCP design consists of just one domain, the total number of switches in a domain is s.

The "Controller-to-Switch" messages are characterized as in the Eq. (5.5) for each switch s_k:

$$\begin{aligned} \alpha_k &= 3 + \delta_k + \theta_k \\ &= 3 + \delta_k + \rho_k \times {}^{c}r \\ &= 3 + \delta_k + \left[\sum_{i=(k-1)h+1}^{kh} \left(\sum_{j=1}^{(k-1)h} \lambda_{ij} + \sum_{j=kh+1}^{\mathcal{H}} \lambda_{ij} \right) \right] \times {}^{c}r \end{aligned} \tag{5.5}$$

The "flow_mod" messages (θ_k) depends on the "packet_in" messages (ρ_k) because every "packet_in" message makes the controller install a flow entry (by a "flow_mod" message) on every switch over an e2e path. Also, 3 is added for "features_request", "features_reply", and "set_configuration" messages sent and/or processed by/at controller after controller-switch channel establishment.

The Eq. (5.6) gives the total number of "Asynchronous" messages per switch. These messages are dominated by the "flow_removed" (σ_k) and "packet_in" messages.

$$\begin{aligned} \beta_k &= \sigma_k + \rho_k \\ &= \sigma_k + \sum_{i=(k-1)h+1}^{kh} \left(\sum_{j=1}^{(k-1)h} \lambda_{ij} + \sum_{j=kh+1}^{\mathcal{H}} \lambda_{ij} \right) \end{aligned} \tag{5.6}$$

Finally, the "Symmetric" messages sent between a switch (s_k) and a controller is defined as in Eq. (5.7):

$$\gamma_k = 2 + \epsilon_k \tag{5.7}$$

Here, 2 is added in the formula for "hello" messages sent and/or processed at/by controller (i.e. 1 message going from controller and 1 message coming from switch).

5.5.2 Distributed Control Plane Design with Local View (DCP-LV)

A distributed control plane with local view (DCP-LV) design consists of distributed controllers associated with switches. It is assumed that there are ${}^{d}m^{l}$ controllers, each of which connects to $\frac{s}{{}^{d}m^{l}}$ switches on average, in this distributed setting. In this structure, each controller manages a sub-network/domain of the whole network. In the DCP-LV design, each controller has its own local network view and each of its neighboring local networks is abstracted as a logical node. However, these controllers do not synchronize their states at all. The controllers need to communicate through controller-to-controller channels to exchange needed state information (e.g. reachability information etc.) regarding their domains. Therefore, a controller does not know about IP prefixes of other domains all the time.

A flow is categorized in two ways: a local flow and a global flow. If a flow's source host and destination host belong to the same domain, then it is called as local flow. Local flows are handled by the controller of that domain if a switch starts a flow initiation request for the flow. On the other hand, a global flow has a source host and destination host which belong to different domains. When a controller receives a flow initiation request for a rule-missing flow, it determines whether it is a global flow. If it is a global flow, the controller then starts asking its neighbor domain controllers if they can provide a path to the flow so that it can reach its destination. This process brings extra burden to the controller. The overhead in a sub-domain, say n-th network, in DCP-LV design is given as in the Eq. (5.8):

$$\begin{aligned} O &= f(\alpha, \beta, \gamma, \omega, \Omega, \Psi) \\ &= \sum_{k=(n-1)\frac{s}{d_{m^l}}+1}^{n\frac{s}{d_{m^l}}} (\alpha_k + \beta_k + \gamma_k + \omega_k) + {}^{d}\Omega_n^l + {}^{d}\Psi_n^l \times {}^{d}r_n^l \end{aligned} \tag{5.8}$$

where (subscript) n represents the n-th domain. For global flows going through neighbors of n-th domain, the n-th controller receives synchronization messages from neighbor controllers, as similar to case in which n-th controller asks to its neighbors for its global flows, stating that whether it can provide a path to those flows. Therefore this process will bring extra message burden (${}^{d}\Omega_n^l$) to a controller. Further, each global flow originating from other domains and going through n-th domain also makes the n-th controller install rules on switches over a path throughout its domain (${}^{d}\Psi_n^l$).

Since θ_k represents the number of *flow_mod* messages created and installed on switches by a controller for local and global flows originating within the domain and coming from hosts of s_k, it can be written that $\theta_k = ({}_{l}\rho_k + {}_{g}\rho_k) \times {}^{d}r_l^n$. It is multiplied by ${}^{d}r_l^n$ because regardless of local or global connection, a controller needs to install

rules for the connection to take it from ingress switch to egress switch. It should be noted that for any domain n, the subscript k is $(n-1)\frac{s}{d_{m^l}} + 1 \leq k \leq n\frac{s}{d_{m^l}}$.

$$\begin{aligned}\alpha_k &= 3 + \delta_k + \theta_k \\ &= 3 + \delta_k + ({}_l\rho_k + {}_g\rho_k) \times {}^d r_n^l \\ &= 3 + \delta_k + \left[\sum_{i=(k-1)h+1}^{kh} \left(\sum_{j=1}^{(k-1)h} \lambda_{ij} + \sum_{j=kh+1}^{\mathcal{H}} \lambda_{ij} \right) \right] \times {}^d r_n^l \end{aligned} \tag{5.9}$$

The *Asynchronous* messages can be calculated by the same idea as in case of CCP design. However, the number of *packet_in* messages are the total of $({}_l\rho_k + {}_g\rho_k)$ in DCP-LV design.

$$\begin{aligned}\beta_k &= \sigma_k + ({}_l\rho_k + {}_g\rho_k) \\ &= \sigma_k + \sum_{i=(k-1)h+1}^{kh} \left(\sum_{j=1}^{(k-1)h} \lambda_{ij} + \sum_{j=kh+1}^{\mathcal{H}} \lambda_{ij} \right) \end{aligned} \tag{5.10}$$

The *Symmetric* messages sent between a switch (s_k) and a controller is defined as in the Eq. (5.11):

$$\gamma_k = 2 + \epsilon_k \tag{5.11}$$

The Eq. (5.12) gives the number of synchronization messages created/sent by a controller for its global flows coming from hosts of s_k. A controller creates one message (to be sent) per neighbor domain controller for its one global flow and receives another message from a neighbor domain.

$$\begin{aligned}\omega_k &= 2 \times {}_g\rho_k \times {}^d N_n^l \\ &= 2 \times \left[\sum_{i=(k-1)h+1}^{kh} \left(\sum_{j=1}^{(n-1)\frac{s}{d_{m^l}}h} \lambda_{ij} + \sum_{j=n\frac{s}{d_{m^l}}h+1}^{\mathcal{H}} \lambda_{ij} \right) \right] \times {}^d N_n^l \end{aligned} \tag{5.12}$$

where ${}^d N_n^l$ is the average number of neighbor domains of n-th domain in DCP-LV design.

5.5.3 Distributed Control Plane Design with Global View (DCP-GV)

As in the DCP-LV setting, it is assumed that there are ${}^{d}m^{g}$ controllers, each of which connects to $\frac{s}{{}^{d}m^{g}}$ switches on average in DCP-GV setting as well. Each controller manages a sub-network/domain of the whole network. However, unlike DCP-LV setting, each controller has a global view of the whole network. This global view of the whole network is acquired by a controller because they periodically exchange their state information through controller-to-controller channels among themselves. Each controller, therefore, can process all the flow initiation requests by itself. There is no synchronization message exchange (i.e. $\omega_k = 0$ where $(n-1)\frac{s}{{}^{d}m^{g}} + 1 \leq k \leq n\frac{s}{{}^{d}m^{g}}$ for any n-th domain) between a controller and its neighbor controllers per global flow unlike the case in DCP-LV due to global periodic state synchronization. On the other hand, the periodic state synchronization (${}^{d}\Omega_n^g$) bring its own message burden to a controller. The overhead in n-th domain in DCP-GV design is given as in the Eq. (5.13):

$$\begin{aligned} O &= f(\alpha, \beta, \gamma, \omega, \Omega, \Psi) \\ &= \sum_{k=(n-1)\frac{s}{{}^{d}m^{g}}+1}^{n\frac{s}{{}^{d}m^{g}}} (\alpha_k + \beta_k + \gamma_k + \omega_k) + {}^{d}\Omega_n^g + {}^{d}\Psi_n^g \times {}^{d}r_n^g \end{aligned} \tag{5.13}$$

The α_k, β_k, and γ_k can be calculated in the same manner with DCP-LV design case as explained in Section 5.5.2. However, in calculation of α_k, ${}^{d}r_n^g$ is used, instead of ${}^{d}r_n^l$.

5.5.4 Hierarchical Control Plane (HCP) Design

This work assumes that the HCP design consists of two layers: The lower layer consisting of local domain controllers and the upper layer in which a master controller resides. In HCP design, it is also assumed that there are ${}^{H}m$ controllers, each of which controls to $\frac{s}{{}^{H}m}$ switches on average. The domain controllers manage their own

domains with a full control over. However, as in the DCP-LV design, a local controller does not main a global view of the whole network. The master controller has a full global view of the whole network by abstracting all domains as logical nodes. A flow is either a local flow or global flow as explained in DCP-LV design. Local flow initiation requests are handled by corresponding local controllers while global flow initiations are first handled by the master controller and then corresponding local controllers over the e2e path of the flow. The overhead in n-th network in HCP design is given as in the Eq. (5.14):

$$\begin{aligned} O &= f(\alpha, \beta, \gamma, \omega, \Omega, \Psi) \\ &= \sum_{k=(n-1)\frac{s}{H_m}+1}^{n\frac{s}{H_m}} (\alpha_k + \beta_k + \gamma_k + \omega_k) + {}^{H}\Omega_n + {}^{H}\Psi_n \times {}^{H}r_n \end{aligned} \tag{5.14}$$

where ${}^{H}\Omega_n$ can be calculated as ${}^{H}\Omega_n = 2 \times {}^{H}\Psi_n$. It is multiplied by 2 because when a new flow from another domain comes to n-th domain to pass through, the n-th domain controller sends a message to master controller stating that it has received a flow destined outside of its domain. Then the master controller sends a response message back to the n-th domain controller stating which egress switch and port that the flow should go through. Therefore, the Eq. (5.14) can be rewritten as:

$$\begin{aligned} O &= f(\alpha, \beta, \gamma, \omega, \Omega, \Psi) \\ &= \sum_{k=(n-1)\frac{s}{H_m}+1}^{n\frac{s}{H_m}} (\alpha_k + \beta_k + \gamma_k + \omega_k) + {}^{H}\Psi_n \times (2 + {}^{H}r_n) \end{aligned} \tag{5.15}$$

The Eq. (5.16) gives the number of synchronization messages created/sent by a controller for its global flows coming from hosts of s_k. A controller creates one message for its one global flow.

$$\begin{aligned}\omega_k &= {}_g\rho_k \times 2 \\ &= \left[\sum_{i=(k-1)h+1}^{kh} \left(\sum_{j=1}^{(n-1)\frac{s}{H_m}h} \lambda_{ij} + \sum_{j=n\frac{s}{H_m}h+1}^{\mathcal{H}} \lambda_{ij} \right) \right] \times 2 \end{aligned} \tag{5.16}$$

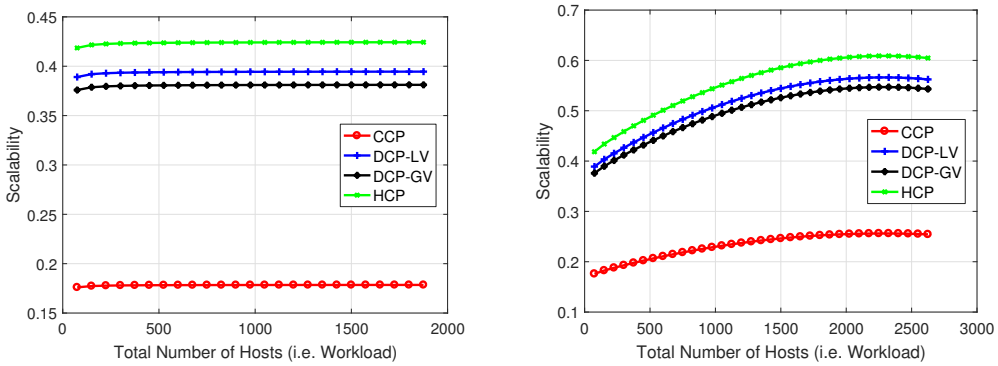
where ω_k is twice of ${}_g\rho_k$ because for each global flow originating from n-th domain there are 2 messages exchanged between the domain controller and master controller due to same philosophy with ${}^H\Omega_n$.

The α_k, β_k, and γ_k can be calculated in the same manner with DCP-LV design as explained in Section 5.5.2. However, in calculation of α_k, ${}^Hr_n$ is used, instead of ${}^dr_n^l$.

5.6 Evaluation

This section presents the experiments conducted using Mininet [102] tool to provide intuitive conclusions regarding scalability of four different SDN network control plane architectures. In each experiment, the number of hosts connected to a switch varies from 5 to 125 by increasing 5 at each iteration. The experiments have averaged 30 runs for each experiment to achieve and exceed 95% statistical significance.

Fig. 5.2 presents the scalability patterns with respect to increasing number of hosts (i.e. workload) in different control plane settings. In this experiment, a random topology with 15 switches have been deployed in each architecture settings. In Fig. 5.2a, the flow sending rate from host i destined to host j is 1 per second, i.e. $\lambda_{i,j} = 1$ where $i \neq j$, otherwise 0. The total number of domains in DCP-LV, DCP-GV, and HCP settings have been set to $m = {}^dm^l = {}^dm^g = {}^Hm = 3$ (in CCP, $m = 1$) with 5 switches each. In all settings, the scalability of architectures show a stable-like behavior. This happens because hosts send each other new flows. Therefore, while the total workload increases, the total overhead messages described in previous sections

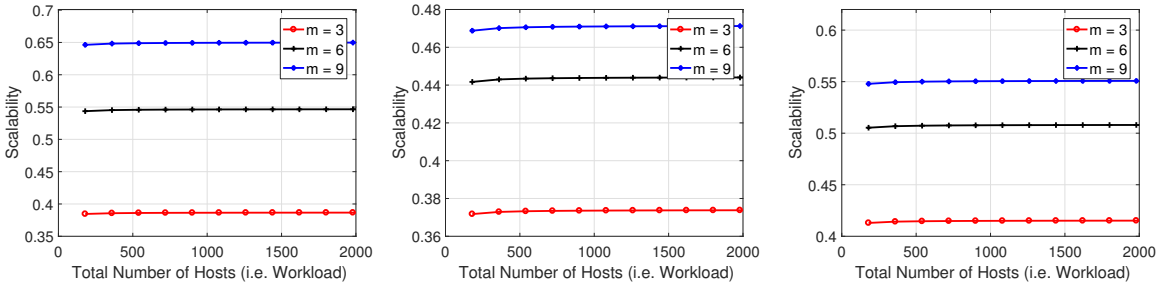


(a) $\lambda_{i,j} = 1$, $m = 3$ (b) $\lambda_{i,j} = 1$, $m = 3$

Figure 5.2.: Scalability vs. Total Number of Hosts (i.e. workload). In 5.2a, hosts send new flows to each other all the time while In 5.2b, hosts start sending the same flows to each other after they have sent first flows.

increase as well, thereby resulting in keeping the ratio of workload and overhead almost the same. On the other hand, in Fig. 5.2b, the experiment settings are the same except that hosts do not send each other new flows all the time. Instead, they keep sending the previously sent flows after they have sent first flows. This, therefore, does not let the overhead messages sent by controller(s) increase while the workload increases. Hence, the scalability show an increasing pattern by the time. In both figures, it can be seen that hierarchical control plane (HCP) provides the best scalability while the centralized control plane (CCP) has the worst scalability. Also, DCP-LV architecture shows a better performance compared to DCP with global view (DCP-GV) regarding scalability.

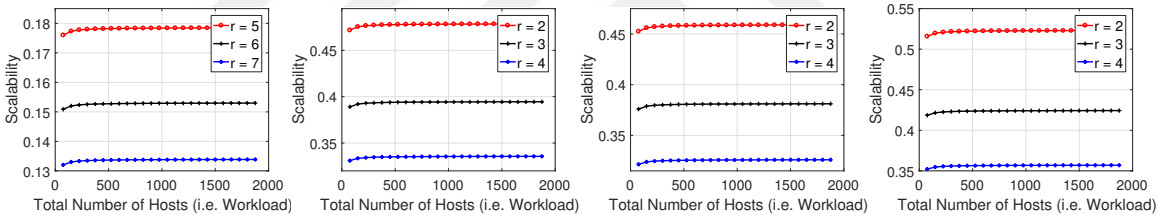
Fig. 5.3 captures the scalability and workload (through total number of hosts) relation with respect to number of domains in DCP-LV, DCP-GV, and HCP settings. The same topology with 36 switches have been used. In each different architecture, $\lambda_{i,j} = 1$ is per second (hosts send new flows to each other). In all settings, the number of domains have been changed from $m = {}^{d}m^{l} = {}^{d}m^{g} = {}^{H}m = 3$ to 9 as in Fig. 5.3a, 5.3b, and 5.3c, respectively. As the figures indicates, the more number of domains results in better scalability result. This happens due to the fact that when you partition a whole network into smaller domains, the total number of switches and



(a) $\lambda_{i,j} = 1$, $m = 3$ to 9, DCP-LV (b) $\lambda_{i,j} = 1$, $m = 3$ to 9, DCP-GV (c) $\lambda_{i,j} = 1$, $m = 3$ to 9, HCP

Figure 5.3.: Scalability vs. Total Number of Hosts (i.e. workload) with respect to number of domains in different control plane architectures.

hosts in a domain will be less. This reduces the total overhead that a controller deals with in the domain.



(a) $\lambda_{i,j} = 1$, $m = 1$,(b) $\lambda_{i,j} = 1$, $m = 3$,(c) $\lambda_{i,j} = 1$, $m = 3$,(d) $\lambda_{i,j} = 1$, $m = 3$, $r = 5$ to 7, CCP $r = 2$ to 4, DCP-LV $r = 2$ to 4, DCP-GV $r = 2$ to 4, HCP

Figure 5.4.: Scalability vs. Total Number of Hosts (i.e. workload) with respect to average length of a path (i.e. total switches over a path) in different control plane architectures.

Fig. 5.4 shows the scalability and total number of hosts (i.e. workload) relation regarding average length of a path (i.e. number of switches over a path) in all control plane settings. The same topology with 36 switches have been deployed as well. In each different architecture, $\lambda_{i,j} = 1$ is per second (hosts send new flows to each other) and the total number of domains in DCP-LV, DCP-GV, and HCP settings to $m = {}^{d}m^{l} = {}^{d}m^{g} = {}^{H}m = 3$ (in CCP, $m = 1$). In CPP (Fig. 5.4a), the average path length have been changed from $r = {}^{c}r = 5$ to 7 while in DCP-LV, DCP-GV, and HCP the average path length varies from $r = {}^{d}r^{l} = {}^{d}r^{g} = {}^{H}r = 2$ to 4 as in Fig.

5.4b, 5.4c, and 5.4d, respectively. In all control plane architectures, a shorter average path length (i.e. less number of switches over a path) results in better scalability result. This happens due to the fact that when a new flow comes to a controller, the controller needs to calculate a path for the flow and install necessary rules on switches over the calculated path. Therefore, the longer a path would cause a controller to create and install the more rules in switches over a path.

5.7 Chapter Summary

Although SDN brings many advantages regarding network and flow management, it still has several issues that need further attention from researchers. Control plane scalability in SDN is one of the crucial issues to be addressed in SDN as well. Many existing solutions propose a way to mitigate the control plane scalability in SDN. However, they quantify the control plane scalability performance in terms of some network QoS parameters such as throughput and latency. This chapter has firstly examined the roots of SDN control plane scalability issues and presented some existing solutions alleviating the problems. The chapter has also given a snapshot of several research attempts proposing a scalability metric to measure the scalability of systems. Finally, a scalability metric has been described and modeled by mathematical methods over different SDN control plane designs.

6 A FRAMEWORK FOR ECONOMIC ANALYSIS OF NETWORK ARCHITECTURES AND DESIGNS

6.1 Abstract

Economical and operational facets of networks drive the necessity for significant changes towards fundamentals of networking architectures. Recently, the momentum of programmable networking attempts illustrates the significance of economic aspects of network technologies. SDN has got the attention of researchers from both academia and industry as a means to decrease network costs and generate revenue for service providers due to features it promises in networking. This work investigates how programmable network architectures, i.e. SDN technology, affect the network economics compared to traditional network architectures, i.e. MPLS technology. This study defines two metrics, *Unit Service Cost Scalability* and *Cost-to-Service*, to evaluate how SDN architecture performs compared to MPLS architecture. Also, mathematical models are presented to calculate certain cost parts of a network. In addition, a comparison of different popular SDN control plane models, Centralized Control Plane (CCP), Distributed Control Plane (DCP), and Hierarchical Control Plane (HCP), are given to understand the economic impact of them with regards to the defined metrics. A video traffic with different patterns is used for the comparison. This work aims at being a useful primer to providing insights regarding which technology and control plane model are appropriate for a specific service, i.e. video, for network owners to plan their investments.

6.2 Introduction

Traditional networks are forcing their limits to meet the needs of today's users, enterprises and carriers due to their limited capabilities. Increasing cloud services, server virtualization, the sharp growth of mobility, and content-like video have led researchers to rethink today's network architectures. In traditional architectures, network devices and appliances are complex and challenging for (re)configuration and (re)installation since they require highly skilled persons. Adding or moving a device from a network brings extra costs. It is also time-consuming because IT people need to deal with multiple switches, routers, etc. and update ACLs, VLANs and some other mechanisms. Furthermore, as business demands or user requirements increase day by day, application developers, carriers, and enterprises need to delve into evolving new services and facilities. However, the software and the hardware in network equipment are vertically integrated and proprietary. Therefore, vendor dependency is an obstacle deterring them from developing new networking applications and services for their networks due to slow equipment production cycle, long protocol standardization process, application testing, and deployment. As a result, dynamicity in networking becomes an inevitable and crucial feature to meet the needs of today's end users.

6.2.1 Chapter Organization

This study investigates how programmable networking, i.e. SDN technology ([207, 208]), affects the network economics compared to traditional networking, i.e. MPLS technology. The MPLS technology has been chosen as the traditional architecture for comparison because it has the concept of flows similar to SDN architecture although MPLS-based flows (FEC+LSP) are not as generic and flexible as the SDN flow abstraction in terms of the match definitions and forwarding actions. Also, MPLS is the most implemented and accepted architecture by service providers to provide QoS among others named earlier. To this end, this work defines two metrics *Unit Service Cost Scalability* and *Cost-to-Service* to evaluate how SDN architecture performs

compared to MPLS architecture. Also, mathematical models are presented to calculate certain cost parts of a network. In addition, a comparison different popular SDN control plane models, Centralized Control Plane (CCP), Distributed Control Plane (DCP), and Hierarchical Control Plane (HCP), are given to understand the economic impact of them with regards to the defined metrics. Video service with different traffic patterns, (1) 20% (inter-domain) - 80% (intra-domain), 2) 50% (inter-domain) - 50% (intra-domain), and 3) 80% (inter-domain) - 20% (intra-domain), has been used for the comparison due to its QoS requirements and the facts explained earlier. This work aims at being a useful primer to providing insights regarding which technology and control plane model are appropriate for a specific service, i.e. video, for network owners to plan their investments. Furthermore, it should be noted that the economic analysis framework proposed in this study is independent of an architecture. SDN and MPLS architectures are the ones considered in this study. It can be applied to any architecture in order to evaluate its economic promises.

In the rest of the chapter, Section 6.3 presents a snapshot of papers that study cost models and expenses for network providers. While 6.4 gives an overview of SDN technology, Section 6.5 discusses value proposition of SDN over network expenditures. Section 6.6 presents MPLS in general, and Section 6.7 discusses the economics of MPLS technology. Section 6.8 analyzes network costs and presents two metrics: *Unit Service Cost Scalability* and *Cost-to-Service* along with the experiment results, Section 6.9 summarizes the study with concluding remarks.

6.3 Related Work

The studies related to network expenditures can be mainly classified into three general categories: 1) identification of CAPEX and OPEX for a network in general, 2) economic analysis of mobile networks, and 3) cost determination using different methods for a network.

The first category of network expenditures related studies considers identification of CAPEX and OPEX for a network in general. Verbrugge *et al.* [209] introduce a cost model to identify expenditures of telecom operators. They discuss the relation between CAPEX and OPEX for telecom networks. The authors mainly split CAPEX into four categories and OPEX into three general parts with carious subparts, respectively. They also discuss activity-based descriptions of identified operational processes for telecom networks. [210] proposes an operational cost model to calculate actual OPEX cost for telecom operators. Their identified CAPEX and OPEX parts are the same. In their cost model, rented infrastructure (e.g. building and equipment) costs do not contribute to CAPEX but OPEX. However, Swisscom *et al.* [211] state that all costs related to infrastructure should be considered as CAPEX. In [212], the authors study the impact of the resilience schemes on both CAPEX and OPEX for a network operator using process-based approach. The authors in [213, 214] evaluate how GMPLS technologies impact network operators' processes using a quantitative technique, and provide a calculation of the expected OPEX savings.

The second category of network expenses related studies mostly concerns with economic analysis of mobile networks. Naudts *et al.* [215] perform a techno-economic analysis of SDN for mobile networks in different architecture cases: a classic scenario in which a distributed network is considered, an SDN scenario with centralized network architecture, and a network architecture shared based on SDN among several network operators through FlowVisor [216] controller. The authors state that the benefits of SDN outweigh its extra costs according to the quantitative analysis conducted. In [217], the authors present a general qualitative study on how SDN/NFV (Network Function Virtualization) affects OPEX for service provider networks. The authors summarize that SDN/NFV is expected to reduce service provider OPEX due to consolidating and optimizing the network and surrounding operating model. Zhang and Hammainen [20] also study the SDN impact on network expenditures using a Finnish LTE reference network. Their findings show that SDN reduces the network related annual CAPEX by 7.72% and OPEX by 0.31% compared to non-SDN LTE.

Knoll [218,219] uses a model called "Life-cycle Cost (LCC)" to investigate a detailed techno-economic structure of SDN/NFV based mobile networks. The LCC-based model considers the life-cycle phases of a network from the idea to set up a certain product or service, followed by the installation and operation of the network up to the decommissioning of the equipment. Bouras *et al.* [21] also present a cost model to estimate the CAPEX, OPEX, and TCO (Total Cost of Ownership) of SDN/NFV based mobile 5G networks and compare them with a traditional network architecture. In [220], the authors study the determination of unit cost for a service with QoS parameters over various SDN network topologies. They characterize the unit cost for a service with respect to CAPEX, OPEX, and workload of a network in a certain time period.

Finally, the last category consists of the studies concerned about cost determination using different methods for a network. Kwak *et al.* [221] propose a cost estimation method based on ABC (Activity-Based Costing) procedure to reduce network OPEX and, thus, the general cost. The authors also present several useful use cases of the suggested cost estimation method and describe expected effects. [222] proposes a method to analyze the implementation of network design optimization by validating network cost models. The authors' key validation technique is the balance between the total network cost calculated from traffic and which of summing the cost of modules. In [223], the authors use a cost mode named "Total Element based Long Run Incremental Cost (TELRIC)" to calculate and distribute the cost of a network element according to the usage that each user type makes of in Next Generation Networks (NGNs). In [224], the authors present a technique for estimating and comparing the costs of different data center architectures and analyze costs of these architectures based on the proposed methodology. Casier *et al.* [225] propose a cost allocation model based on a combination of resource usage and peak capacity to understand how the different CAPEX and OPEX cost parts for a service provider can be allocated to the services. Bailey [226] discusses the economic realities of migrating to the cloud and virtualized networking. His conclusion is that once the network was a

service, now the data center is the new network for the purpose of economic modeling and business insights.

As can be seen from the studies presented above, this study differs from them in a way that it defines and presents two metrics to investigate how programmable networking, i.e. SDN technology, impact the unit cost for a service and service introduction cost compared to traditional networking, i.e. MPLS technology. It calculates CAPEX and OPEX as a means to use in calculation of unit service cost metric. It also includes network performance with regards to total number of satisfied requests (i.e. workload) in the model proposed unlike aforementioned studies.

6.4 SDN Overview

SDN allows managing flows in a finer-granular way based on more attributes of packet headers by means of a Controller-Data Plane Interface (C-DPI) such as OpenFlow protocol [34]. As shown in Fig. 2.1, Open Networking Foundation (ONF) [227] vertically splits SDN architecture into three main planes [38]:

6.4.1 Data Plane

The data plane is the bottom plane and consists of network devices such as routers, physical/virtual switches, access points, etc. These devices are accessible and managed through C-DPIs by SDN controller(s). The network elements and controller(s) may communicate through secure connections such as the TLS connection.

6.4.2 Control Plane

An SDN control plane comprises a set of software-based SDN controller(s) to provide control functionality in order to supervise the network forwarding behavior through C-DPI. It has interfaces to enable communication among controllers in a control plane (Intermediate-Controller Plane Interface, i.e. I-CPI [39], optionally

secured using the TLS), between controllers and network devices (C-DPI), and also between controllers and applications (Application-Controller Plane Interface, i.e. A-CPI).

As illustrated in Fig. 5.1, some of the popular control plane models used in SDN technology are CCP, DCP and HCP. These models have their own intrinsic advantages and disadvantages with respect to the different concepts such as control plane scalability [228], resiliency [229], better manageability and so on. These are the control plane models in SDN that have been considered while conducting analysis to understand their impact over network economics. However, they are not the only control plane models in SDN.

6.4.2.1 Centralized Controller Plane Model (CCP)

This type of settings revolves around a single central controller [23, 69] with a global network view. The model is simple and it is easy to manage the network. This design may meet the needs of small to mid-size networks.

6.4.2.2 Distributed Controller Plane Model (DCP)

This model [66, 72] consists of distributed controllers associated with switches. Each controller manages a subnetwork/domain of the whole network and has its own local network view, which is, in turn, abstracted as a logical node to its neighboring controllers. These controllers communicate with each other (i.e. connected neighbors) when they receive a packet destined out of its domain in order to set up an end-to-end path.

6.4.2.3 Hierarchical Controller Plane Model (HCP)

An HCP design [79, 84] consists of two control plane layers minimum: The lower-layer, consisting of local domain controllers, and the upper-layer where another con-

troller, usually called “master”, resides. The domain controllers manage their own domains with full control and are not connected to each other but the master controller. However, a local controller does not maintain a global view of the whole network. Instead, the master controller has a full global view of the entire network by abstracting all domains as logical nodes.

6.4.3 Application Plane

An SDN application plane consists of one or more end-user applications (security, visualization, etc.) that interact with a controller(s) to utilize an abstract view of the network for their internal decision making process. These applications communicate with a controller(s) via an open A-CPI (e.g. REST API). An SDN application comprises an SDN App Logic and A-CPI Driver.

6.5 SDN Value Proposition

This section discusses the values of SDN architecture that are results of programmable networking. It is important for a network owner to identify and understand these values while evaluating economic position of an architecture before investing money on the architecture.

6.5.1 SDN Impacts over CAPEX

Virtualization and flexible placement of network functions, fine-grain network traffic optimization, and efficient resource utilization through orchestration associated with SDN provide an intuitive indication of potential CAPEX reduction [217]. SDN can influence CAPEX of a network in different ways. Some of the key factors that affect the potential CAPEX changes include:

- Simpler Network Devices: In an SDN network, each network device will be simpler because complex features such as proprietary software implemented by vendors are not needed. The devices will be simpler and cheaper white-boxes.

- Extra Components: In an SDN scenario, the network will have other elements that a traditional network does not have. These elements include a controller(s) hardware and controller software licenses (if not used an open-source software).

- Network Dimensioning: Since network controller(s) can have global network view in SDN case, this leads to better network resource utilization by means of some methods such as load balancing. Therefore, there may not be a need for over-provisioning the network, which can reduce the capital expenditures.

6.5.2 SDN Impacts over OPEX

[230] reports that OPEX for service providers are up to 5 times higher than CAPEX according to the financial analysis conducted. This increased cost leads service providers to pay more attention to the OPEX part of their expenses. SDN promises to lower some of the main OPEX components for service providers using its various features:

- Energy-Related Costs: In an SDN network, switches will not have an embedded control plane, which consumes the most of the total energy that a switch needs. Also, since SDN allows more efficient traffic optimization over the network devices, this reduces the total number of needed devices.

- Maintenance Costs: SDN creates a homogeneous network environment on hardware and software. There is no case in which different vendor-dependent devices need to be managed and maintained independently.

- Reparation Costs: SDN provides better testing opportunities, identifying bugs, and so on before reaching out the actual production traffic. Software related is-

sues can be remotely fixed without touching network devices since these devices are simple and SDN is software-centric.

- Service Provisioning Costs: Service provisioning cost in SDN scenario is expected to be lower due to automated configuration of network devices, less personnel need for network tasks due to automation, reduced manual configuration and so on.

6.6 MPLS Overview

Although the original idea behind the development of MPLS was to facilitate fast packet switching, currently its main goal is to support traffic engineering and provide quality of service. The goal of traffic engineering is to facilitate efficient and reliable network operations and at the same time optimize the utilization of network resources. Routers that support MPLS are known as Label Switching Routers (LSRs). When an LSR identifies the Forwarding Equivalent Class (FEC)[1] associated with the packet, it selects a label from a pool of free labels, and makes an entry in a table referred to as the Label Forward Information Base (LFIB) [231]. This table contains information regarding the incoming and outgoing labels associated with an FEC and the output interface, i.e. the FEC's next hop router. The LSR also saves the label in its FIB in the entry associated with the FEC. A Label Switched Path (LSP) is referred to as a path from the ingress node to the egress node of an MPLS domain followed by packets with the same label.

6.6.1 MPLS-TE

MPLS-TE model mainly consists of Path Management, Traffic Assignment, Network State Information Dissemination, and Network Management components [232]. Path Management is a mechanism by which MPLS network manages the packet for-

[1]A FEC is a class which comprises the group of packets which are treated in the same manner by the LSR.

warding, which includes choosing the right path for the specific packet, maintaining the existing path and finding new paths due to failure or addition of links. Traffic Assignment performs the assignment of traffic to the established tunnel by path management to do load distribution. Network State Information Dissemination component conducts the advertisement of the topological and state information of the network to all the nodes of the MPLS network. The final component, Network Management, is responsible for configuration and fault management functions.

6.6.2 Signaling Protocols in MPLS-TE

There are two main signaling protocols that support TE in MPLS networks: Constraint-based Label Distribution Protocol (CR-LDP) [233] and Resource Reservation Protocol-Traffic Engineering (RSVP-TE) [234].

CR-LDP is the extension of the signaling protocol LDP. It is extended from LDP with the additional support to explicitly route the information about the traffic parameters for the reservation of the resources along the LSPs. CR-LDP is a hard state protocol as it sends the signaling messages only once without refreshing.

RSVP uses the direct routes to set up the Constraint-based Routed LSPs (CR-LSPs) (also known as ER-LSPs). It uses UDP for resource reservation and label allocation. RSVP supports Integrated Service (IntServ) model of QoS. The TE supported RSVP, which is an extended version of RSVP and known as RSVP-TE, supports loop detection, periodization, reordering of a path, and strict/loose CR-LSPs.

6.7 Economic Position of MPLS

Service providers continue to use MPLS to transparently carry legacy services as part of their evolutionary service strategy. This factor results in an opportunity for both service providers and customers to exploit MPLS as a new service opportunity. In terms of the service provider, MPLS can speed the service delivery window for customers who subscribe to these services. For the customers, MPLS can reduce WAN

costs or offer services internally to various departments or subsidiaries. On the other hand, enterprise organizations are using MPLS to develop virtualized architectures to scale WAN/LAN, campus, and data center resources [132].

MPLS attracts service providers as a business opportunity due to cost savings and revenue generation factors it can provide. This business opportunity is ultimately translated to deploying a global ubiquitous network and to developing services that are based on this technology. Moreover, MPLS-based faster TTM service delivery windows of new services are critical for the service providers. In addition, MPLS can provide any-to-any service constructs that customers can utilize.

Furthermore, service providers expect operational savings by deploying new IP/MPLS-based services. Applications such as voice, once implemented in circuit-based networks, are perceived by service providers to be less expensive to deploy over IP. These cost savings come from the opportunity to consolidate multiple infrastructures (PSTN for voice and video, and data over IP). The consolidation can be facilitated by such mechanisms as a differentiated class of service (CoS).

Finally, controlling costs while supporting existing and new services, and transitioning multiple networks to a consolidated packet-based service-oriented technology, such as MPLS, are indeed requirements for service providers.

6.8 Network Economic Performance Indicators

A final price that end users pay for their use of services from networks mostly rely on a unit cost for the service charged by the network. However, these types of internal cost calculations are highly proprietary since networks typically do not share their internal cost structures in current practices. On the other hand, understanding of these internal price calculations is crucial as researchers try to come up with an optimal final pricing scheme for both users and service providers.

Determining cost for a QoS service in a network is not an easy task for network operators. If the cost of a service in a network is miscalculated, two consequences

are possible for the network in mid-term and long-term. One consequence is that the network may start making a loss if customers pay lower than real cost for services they use based on the cost calculated by the network operators. Another possible consequence is that a network may lose customers, thereby losing revenue, in long-term if the network calculates higher cost than the real cost for services. In this case, the network may seem to be making profit in short-term but customers would eventually stop receiving service from the network. As a result, cost calculation is a crucial stage for a network in terms of reflecting a real cost structure for services so that both networks and customers can utilize an optimum service-cost relationship for their future businesses. We note that our cost calculation scheme reflects the minimum cost that a network should charge for the service. It neither leads to a loss nor a profit for a network. Keeping cost at a higher or lower level depends on a network's market strategies.

To this end, we analyze network costs and present two metrics: *Unit Service Cost Scalability* metric to evaluate unit cost performance of a network technology for service requests with respect to increasing service request workload and *Cost-to-Service* metric to economically quantify the cost of introducing a new service in a networking technology in this section. We also present CAPEX and OPEX calculations, which is used in determination of unit cost for a service with QoS parameters in a network. We only consider bandwidth QoS parameter for service requests along with multiple numerical service tiers. We should note that the unit cost calculation scheme proposed in this paper reflects the minimum cost that a network should charge for the service to compensate its expenditures. Therefore, it neither leads to an financial loss nor a profit for a network. Keeping cost at a higher or lower amount depends on network's market strategy and it is out of this paper's scope.

6.8.1 Unit Service Cost Scalability Metric

Scalability is a frequently-claimed attribute of various systems. It is a multi-dimensional topic. While the basic notion is intuitive, the term *Scalability* does not evoke the same concept to everybody. While some people may refer to scalability as optimization of processing power to CPUs, others may define it as a measure of parallelization of applications across different machines. Therefore, there is no general, precise agreement on neither its definition nor content. However, regardless of its meaning to someone, it is a desired property indicating positive sense for a system, algorithm, network and so on.

There are several research efforts [44–50] proposing a metric to measure scalability of systems. Most of these metrics are for homogeneous environments. The majority of these proposals revolve around two major types of scalability metrics: *Isospeed* scalability and *Isoefficiency* scalability. The *Isospeed* scalability is characterized by the fact that an achieved average unit speed of an algorithm on a given machine can remain constant with increasing number of processors and problem size for an algorithm-machine combination [44]. In [45], the authors present a metric to describe the scalability of an algorithm-machine combination in homogeneous environments. Their scalability function is defined as $\psi(p, p') = \frac{p'W}{pW'}$ where p and p' are the initial and scaled number of processors of the systems respectively, and W and W' are the initial and scaled problem size (workload) respectively. The *Isoefficiency* scalability is described as the ability of parallel machine to keep the parallel efficiency constant when the system and problem size increase [46]. The parallel efficiency is defined as speedup over the number of processors, i.e. $E = \frac{S}{p}$. Speedup is also given by the ratio of problem size (W) and parallel execution time (T_p), i.e. $S = \frac{W}{T_p}$ where $T_p = \frac{W+T_0(W,p)}{p}$ with $T_0(W, p)$ extra communication overhead [47]. Pastor and Orero [48] define heterogeneous scalability by presenting a heterogeneous efficiency function. They attempt to extend the homogeneous *Isoefficiency* scalability model to heterogeneous computing and, therefore, their work inherits the limitation of parallel speedup, requiring the

measurement of solving large-scale problem on single node. Sun et al. [49] propose a scalability metric called *Isospeed-efficiency* for general heterogeneous computing systems. This metric combines the roots of both *Isospeed* scalability and *Isoefficiency* scalability metrics by means of a concept called "Marked Speed" to describe the computing power for a stand-alone node and a combined computing system.

Scalability means not just the ability to operate, but to operate efficiently and with adequately quality of service, over the given range of configurations. There are certain questions that need answers from researchers. One interesting example for such questions is that whether the cost of the system to provide service affect the system scalability. Jogalekar and Woodside [50] state that increased capacity should be in proportion to the cost of the system, and quality of service should be maintained. Moreover, when discussing network scalability, a large number of influencing factors have to be taken into account to arrive at a full picture. Behringer et al. [235] state that TCO is one example for such factors.

There are several research efforts proposing a metric to measure scalability of systems while considering the cost of the systems. [236] proposes a scalability metric, called "P-Scalability", taking into account the cost of the system in distributed systems. It utilizes a concept called "power" measure and the cost of the system to provide service at a scale factor k. It is defined as:

$$P - Scalability(k_1, k_2) = \frac{P(k_2) \cdot Cost(k_1)}{P(k_1) \cdot Cost(k_2)}$$

where $P(k) = Throughput/ResponseTime$. This metric combines capacity and response time (both are present in the power P) with the cost of the system. [50] defines scalability around "productivity" of the system in distributed systems. Productivity $F(k)$ is the value delivered per second, divided by the cost per second:

$$F(k) = \frac{\lambda(k) \cdot f(k)}{C(k)}$$

where $\lambda(k)$ is throughput in response/sec at scale k, $f(k)$ is the value of QoS at scale k, and $C(k)$ is the cost of the system at scale k.

As proposed in these studies, the cost of providing service while preserving QoS for each service request in a system should be considered as evaluating the scalability of the system. In this context, we define a metric called *Unit Service Cost Scalability* to evaluate unit cost performance of a network for service requests with respect to increasing service request workload. This metric takes into account the network workload and different expenditures incurred to preserve the QoS at the same level for all service requests in the network. We expect that more scalable architectures result in less unit service cost in networks because such networks are able to handle more workload with respect to the same amount of network expenses compared to others.

This study characterizes the unit cost for a service (request) from a service tier as a function of network CAPEX, OPEX, and Workload over a certain time period. Workload is referred to service requests of all service tiers coming from users/customers to and *satisfied* by the network. The general unit cost framework for a service (request) with one QoS parameter (bandwidth) from a service tier is shown in Eq. 6.1. This formula implies that the unit service cost for a request from certain service tier is the ratio of TCO (CAPEX + OPEX) over workload in a given period.

$$\mathcal{C}_{bw_j} = f(\mathbb{C}, \mathbb{O}, W) = \begin{cases} \frac{\mathbb{C}+\mathbb{O}}{\sum_{j=1} w_j \cdot |bw_j|} \cdot |bw_j| & \text{before } \delta \\ \frac{\mathbb{C}+\mathbb{C}_\delta+\mathbb{O}+\mathbb{O}_\delta}{\sum_{j=1} (w_j+w_{\delta_j}) \cdot |bw_j|} \cdot |bw_j| & \text{after } \delta \end{cases} \tag{6.1}$$

where bw_j, $|bw_j|$, $\mathcal{C}_{bw_j}$, $\mathbb{C}$, $\mathbb{O}$ represent the type of (i.e. bandwidth) service with tier j, the numerical value of the service tier bw_j, the unit cost of the service bw_j, CAPEX, and OPEX in a time period (e.g. month, year), respectively. w_j and w_{δ_j} represent the workload and possible additional workload of service bw_j and $W = \sum_{j=1} w_j$ and $W_\delta = \sum_{j=1} w_{\delta_j}$. Similarly, $\mathbb{C}_\delta$, $\mathbb{O}_\delta$, and W_δ represent possible extra CAPEX, OPEX, and workload, respectively, incurred after introducing different kinds of changes/upgrades (represented as δ) in the network.

6.8.1.1 Calculation of CAPEX

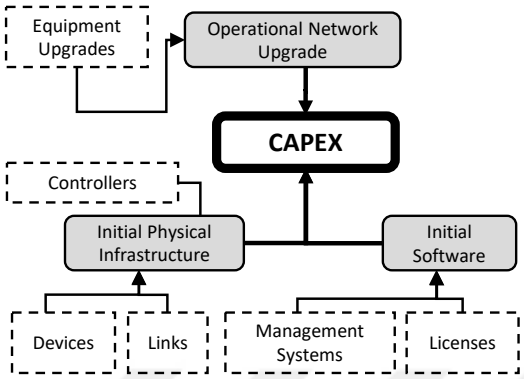


Figure 6.1.: CAPEX cost groups and corresponding input costs. The dashed rectangles represent the input costs (in CAPEX) to the corresponding cost groups represented as gray rounded rectangles. The arrows point to the direction of the input.

CAPEX consists of expenses that are made for acquiring or upgrading a fixed, physical and non-consumable assets of a company. They are needed to expand the services to the customers. As seen in Fig. 6.1, CAPEX is mainly determined by the initial physical infrastructure expenses ($\mathcal{H}$), initial software expenses ($\mathcal{S}$), and operational network upgrade costs ($\mathcal{A}$). Therefore, CAPEX is a function of these expenses and can be written as in Eq. 6.2:

$$\mathbb{C} = f(\mathcal{H}, \mathcal{S}, \mathcal{A}) = \mathcal{H} + \mathcal{S} + \mathcal{A} \tag{6.2}$$

6.8.1.1.1 Initial Physical Infrastructure Expenses ($\mathcal{H}$) These expenses are related to hardware and the infrastructure of a network. The initial physical infrastructure ($\mathcal{H}$) of a network primarily consists of cables, network devices, and also extra hardware such as a server that controller(s) installed on (in SDN case). Therefore, $\mathcal{H}$ can be written as in Eq. 6.3:

$$\mathcal{H} = \sum_{i=1}^{|l|} \mathcal{C}_{l_i} + \sum_{j=1}^{|d|} \mathcal{C}_{d_j} + \sum_{k=1}^{|c|} \mathcal{C}_{c_k} \tag{6.3}$$

where $|l|$, $|d|$, $|c|$ represent the total number of links, network devices, controller hardware and $\mathcal{C}_{l_i}, \mathcal{C}_{d_j}, \mathcal{C}_{c_k}$ represent the cost of the corresponding link, network device, and controller hardware, respectively.

6.8.1.1.2 Initial Software Expenses ($\mathcal{S}$) Similarly, the initial software expenses ($\mathcal{S}$) such as the purchase of management systems, licenses for proprietary controllers (in SDN case) can be calculated as in Eq. 6.4:

$$\mathcal{S} = \sum_{m=1}^{|s|} \mathcal{C}_{s_m} \tag{6.4}$$

where $|s|$ and $\mathcal{C}_{s_i}$ represent the total number of paid software used in the network and the cost of corresponding software, respectively.

6.8.1.1.3 Operational Network Upgrade Costs ($\mathcal{A}$) These expenses are incurred from the ongoing network upgrade activities, represented as δ, such as adding, deleting, upgrading controller(s) (in SDN case), network device(s), link(s), and so on in the network. These expenses correspond the $\mathbb{C}_\delta$ and $\mathbb{O}_\delta$ in the Eq. 6.1. They are considered because, after foregoing modifications, extra CAPEX and OPEX may be incurred in the network.

Finally, substituting Eq. 6.3 and Eq. 6.4 in Eq. 6.2, the new CAPEX equation becomes as in Eq. 6.5:

$$\begin{aligned} \mathbb{C} &= f(\mathcal{H}, \mathcal{S}, \mathcal{A}) \\ &= \sum_{i=1}^{|l|} \mathcal{C}_{l_i} + \sum_{j=1}^{|d|} \mathcal{C}_{d_j} + \sum_{k=1}^{|c|} \mathcal{C}_{c_k} + \sum_{m=1}^{|s|} \mathcal{C}_{s_m} + \mathcal{A} \end{aligned} \tag{6.5}$$

This study considers only the initial physical infrastructure expenses ($\mathcal{H}$), the initial software ($\mathcal{S}$) expenses, and the operational network upgrades costs ($\mathcal{A}$) as the main drivers for CAPEX for both SDN and MPLS cases in this study.

6.8.1.2 Calculation of OPEX: Overhead-based Approach

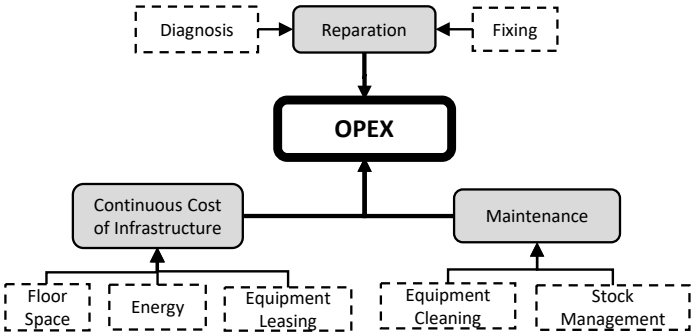


Figure 6.2.: OPEX cost groups and corresponding input costs in overhead-based approach. The dashed rectangles represent the input costs (in OPEX) to the corresponding cost groups represented as gray rounded rectangles. The arrows point to the direction of the input.

OPEX is more complicated to calculate than CAPEX because it requires more information about internal network dynamics. However, such information is proprietary and highly hidden by network owners. As seen in Fig. 6.2, Main drivers for OPEX in overhead-based approach are (i) infrastructure energy expenses ($\mathcal{K}$), (ii) maintenance expenses ($\mathcal{M}$), and (iii) reparation expenses ($\mathcal{R}$) in a network. Therefore, it can be stated, in general, that OPEX is a function of these expenses and can be written as in Eq. 6.6:

$$\mathbb{O} = f(\mathcal{K}, \mathcal{M}, \mathcal{R}) = \mathcal{K} + \mathcal{M} + \mathcal{R} \tag{6.6}$$

6.8.1.2.1 Infrastructure ($\mathcal{K}$), Maintenance ($\mathcal{M}$), and Reparation ($\mathcal{R}$) Costs

It is difficult to simulate these expenses in an artificial simulation environment without real and accurate parameters from networks. This study assumes that they have a relation with the total messages handled in the network in SDN case in order to model them without data from a real network. These messages are internal overhead messages (O) generated in the network and service requests (W) (i.e. workload) entering the network. The idea behind this assumption is that the more messages handled

in the network result in the more infrastructure energy expenses (e.g. due to more power and energy consumption) and require the more maintenance, and reparation, which brings more expenses as well, in the network. In order to monetize continuous infrastructure, maintenance and reparation expenses for OPEX, this study assumes that every single message processed in the network brings $\$\varepsilon$ cost to the network. Therefore, the total of these expenses becomes as in Eq. 6.7:

$$\mathcal{K} + \mathcal{M} + \mathcal{R} = \mathcal{C}_{pw}\left(\sum_{j=1}^{|d|} pw_{d_j} + \sum_{k=1}^{|c|} pw_{c_k}\right) + \varepsilon\left(\sum_{j=1} w_j + O\right) \tag{6.7}$$

where $\mathcal{C}_{pw}$, pw_{d_j}, pw_{c_k} represent cost of KWh electricity power, energy consumption of network device d_j and controller c_k per hour, respectively.

On the other hand, in MPLS case, MTBF (Mean Time Between Failures) and MTTR (Mean Time To Repair) values are used to calculate Reparation ($\mathcal{R}$) costs of network devices as in Eq. 6.8:

$$\mathcal{R} = |d| * \frac{\mathbb{T}}{MTBF} * MTTR * p \tag{6.8}$$

where $\mathbb{T}$ and p represent a time period that the OPEX is calculated for and pay rate for the employee who repairs a device, respectively. It is assumed that each device is repaired by one employee.The same formula, as in Eq. 6.7, is also used for energy expenses ($\mathcal{K}$) in MPLS too:

$$\mathcal{K} = \mathcal{C}_{pw}\left(\sum_{j=1}^{|d|} pw_{d_j} + \sum_{k=1}^{|c|} pw_{c_k}\right) \tag{6.9}$$

where the number of controllers ($|c|$) is zero in MPLS case. It is also assumed that total of Maintenance ($\mathcal{M}$) costs are the half of the Reparation ($\mathcal{R}$) costs, $\mathcal{R} = \frac{\mathcal{M}}{2}$.

6.8.1.2.2 Overhead Messages As defined in Chapter 5, overhead is referred to the messages processed in the control plane by a controller(s). In SDN, when the first packet of a new flow enters a network through a switch, the switch starts a flow initiation request if there is no rule entry matching the packet in switch's

flow table. This flow initiation request is then sent to the controller. The controller processes it and installs a rule for the flow in switches over the path calculated by the controller. Therefore, a rule-missing flow results in some other control-messages which are created, processed and sent by a switch and/or controller. Also, a controller may deal with some other periodic messages, such as statistics, generated in the network but not related to rule installation process. These types of messages are categorized as an overhead message. In an SDN network with OpenFlow protocol, there are three different types of messages between a controller and data plane devices: "Controller-to-Switch", "Asynchronous, and "Symmetric" messages. Each of these message types has its sub-types as well.

This work characterizes the overhead in a control plane, i.e. a controller, as the function of these three message types of OpenFlow protocol and synchronization messages among controllers. A control plane model generating less of these messages results in less OPEX. As explained in Chapter 5, the number of overhead messages are quantified as in Eq. 5.2.

A total number of overhead messages in a control plane model such as CCP, DCP, and HCP can be different. Therefore, the OPEX in each corresponding model can be different. The details of overhead messages calculation and discussion is in Chapter 5.

6.8.1.3 Calculation of OPEX: Overhead-based Approach - Evaluation

This section presents numerical results to provide some insights in order to understand the economic impact of SDN architecture with different control plane models (CCP, DCP, and HCP) and MPLS architecture on unit service cost using the overhead-based approach. In this context, this work analyzes unit service cost and service introduction cost for the video service by total number of satisfied requests, CAPEX and OPEX in different SDN models and MPLS architecture.

6.8.1.3.1 Experimental Setup

6.8.1.3.1.1 SDN Setup Mininet emulator [102] with POX controller [106] has been used in SDN case. While there is one controller in CCP model, the whole network has been divided into 4 fully-connected sub-networks with a controller for each in DCP and HCP models with a varying number of switches in different simulation cases shown in the figures. There is also a master controller on top of local domain controllers in HCP model.

6.8.1.3.1.2 MPLS Setup Regarding MPLS setting, ns3 [107] network simulator has been used. A signaling protocol such as RSVP-TE or CR-LDP has been needed to support constraint-based routing in MPLS. Since none of them has been implemented in ns3 at the time of this writing and it is time-consuming and effort-greedy to implement them in ns3, extra packets have been generated between network elements to mimic link state advertisements and state refresh messages for LSPs from aforementioned signaling protocols in MPLS.

6.8.1.3.1.3 Shared Setup The experiments have used 3 *Mbps* flow sending rate for all service requests. Therefore, there is only one service tier and $|bw| = 3$ for all requests. Other numerical calculations have been done using MATLAB platform. Also, a modified version of Waxman [237] random topology generator defined by Erdos-Renyi random graph model has been used to randomly create the networks while preserving connectivity degrees of nodes (i.e. switches) as three in all switch cases and models. Furthermore, a heuristics, i.e. A*Prune Algorithm [238], has been used to find a feasible path through the network because constraint-based routing with two or more constraints has been shown to be an NP-hard [239]. A*Prune algorithm combines A*-search with a correct pruning technique. A*Prune algorithm can be used to solve finding the K shortest paths subject to multiple constraints (KMCSP). Finally, all experiments were performed on Ubuntu 14.04 in Oracle VirtualBox using an Intel Core i7-5500 system with 12GB RAM.

The scheme explained in Chapter 5 has been used to get overhead messages with respect to number of requests in SDN models. The experiments have used three different traffic patterns: 1) 20% (inter-domain) - 80% (intra-domain), 2) 50% (inter-domain) - 50% (intra-domain), and 3) 80% (inter-domain) - 20% (intra-domain). In each pattern, the source and destination switches are chosen randomly for service request while preserving the traffic pattern condition. Furthermore, all calculations are based on one year period. Finally, it has averaged 15 runs for each experiment to achieve and exceed 95% statistical significance.

Table 6.1.: List of parameters and their values used in calculation of CAPEX and OPEX (overhead-based approach).

Parameter	**Value**			
	CCP	**DCP**	**HCP**	**MPLS**
$\|d\|$	4, 8, 12, 16, 20, 24, 28, 32	4, 8, 12, 16, 20, 24, 28, 32	4, 8, 12, 16, 20, 24, 28, 32	4, 8, 12, 16, 20, 24, 28, 32
$\mathcal{C}_d$	\$1000	\$1000	\$1000	\$2000
$\|l\|$	6, 12, 18, 24, 30, 36, 42, 48	6, 12, 18, 24, 30, 36, 42, 48	6, 12, 18, 24, 30, 36, 42, 48	6, 12, 18, 24, 30, 36, 42, 48
$\mathcal{C}_l$	\$500	\$500	\$500	\$500
$\|c\|$	1	4	$4+1$	N/A
$\mathcal{C}_c$	$\frac{\$500*\|d\|}{\|c\|}$	$\frac{\$500*\|d\|}{\|c\|}$	$\frac{\$500*\|d\|}{\|c\|}$	N/A
$\|s\|$	1	4	$4+1$	N/A
$\mathcal{C}_s$	$\frac{\$500*\|d\|}{\|c\|}$	$\frac{\$500*\|d\|}{\|c\|}$	$\frac{\$500*\|d\|}{\|c\|}$	N/A
ε	$\$10^{-8}$	$\$10^{-8}$	$\$10^{-8}$	N/A
$\mathcal{C}_{pw}$	\$0.116 KWh	\$0.116 KWh	\$0.116 KWh	\$0.116 KWh
pw_d	48 W	48 W	48 W	60 W
pw_c	400 W	400 W	400 W	N/A
$\mathbb{T}$	1 year	1 year	1 year	1 year
$MTBF$	N/A	N/A	N/A	2000 hours
$MTTR$	N/A	N/A	N/A	10 hours

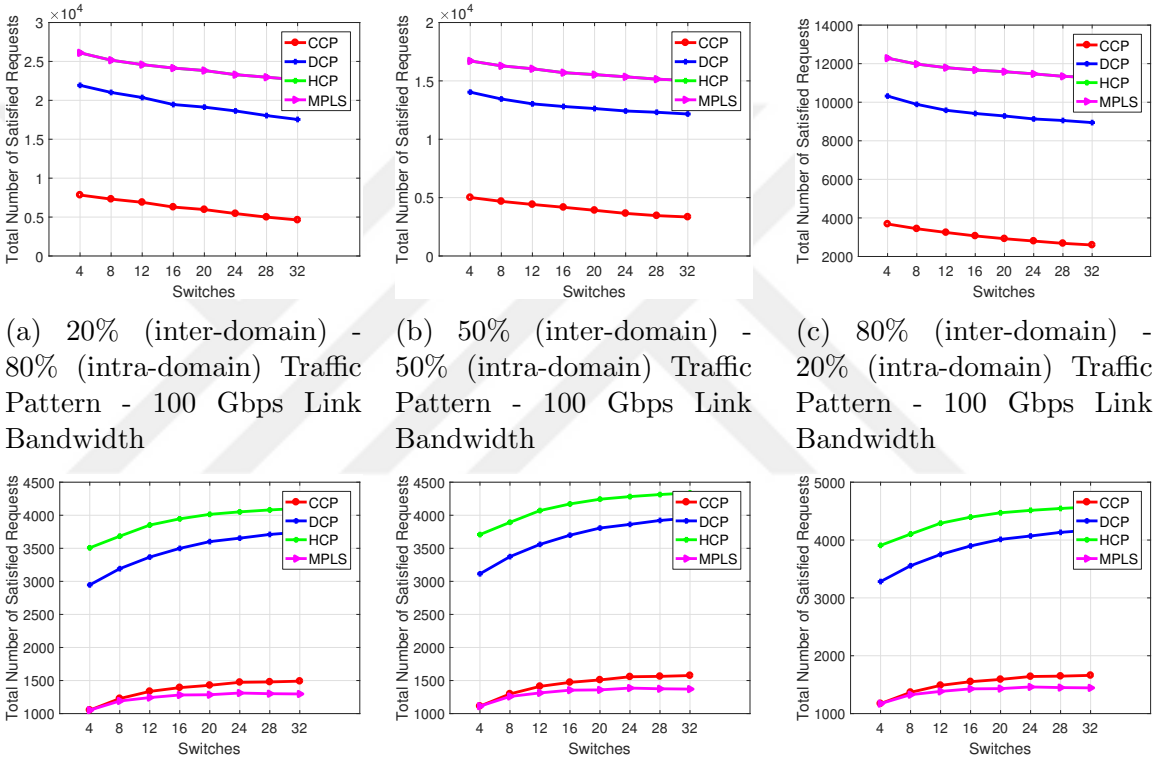
Table 6.1 lists the parameters and their values used in calculation of CAPEX and OPEX (overhead-based approach). It is difficult to gather precise input values for some of these parameters because they are proprietary and companies are not willing to publicly share them. These value assumptions constitute an average of each

parameter based on Internet research, literature review ([20, 21, 209, 212, 240–249]), and research discussions. Although these input numbers may not be reflecting precise and/or realistic values, they should not impact the nature of the calculation framework since these values are very relative for every network company. The same cost has been used for all links ($\mathcal{C}_l$) in both SDN models and MPLS case. However, it has been assumed that \$100 and \$500 for a link cost in 1 Gbps and 100 Gbps link bandwidth cases, respectively. It has been assumed that device cost is double ($\mathcal{C}_d$) in MPLS case since traditional network equipment is expected to be more expensive than SDN equipment due to integrated control plane (i.e. proprietary software implementation). It has also been assumed the same cost, which is proportional to the number of network devices ($|d|$) and that of controllers ($|c|$) in different device number cases, for a controller hardware ($\mathcal{C}_c$) and software ($\mathcal{C}_s$) in all models. Regarding device energy consumption, these values depend on many factors such as number of ports, capacity of port, memory type used, number of coming flows/packets and so on. Therefore, an average value has been used from these studies ([240,244–247,249]) for both SDN and MPLS devices. Furthermore, it has used the same number of employees and pay rate in service introduction steps in both SDN and MPLS cases for all switch cases. The time spent by an employee has been made proportional to the number of network devices and controllers in service introduction steps of SDN models. These time ratios are based on customer feedback from SDN use cases explained in [243]. In MPLS, this time is much more compared to SDN case due to mostly manual configurations over multiple heterogeneous devices. Finally, these values are the same for the different traffic patterns.

6.8.1.3.2 Experimental Results These results are based on the CAPEX calculation formulas presented in Subsection 6.8.1.1, OPEX calculation formulas presented in Subsection 6.8.1.2, and values presented in Table 6.1.

Fig. 6.3 shows the relation between the total number of satisfied QoS-based requests regarding the different switch numbers in SDN models and MPLS under the

different traffic patterns. This experiment consists of two parts: 100 Gbps and 1 Gbps link bandwidth parts. In the first part, (Fig. 6.3a, 6.3b, and 6.3c), enough bandwidth (100 Gbps) has been provided in links so that there is no service request rejection due to network resource limitations, while the link bandwidth has been reduced to 1 Gbps to see their performances under network resource limitations in the second part of the experiment (Fig. 6.3d, 6.3e, and 6.3f).



(a) 20% (inter-domain) - 80% (intra-domain) Traffic Pattern - 100 Gbps Link Bandwidth

(b) 50% (inter-domain) - 50% (intra-domain) Traffic Pattern - 100 Gbps Link Bandwidth

(c) 80% (inter-domain) - 20% (intra-domain) Traffic Pattern - 100 Gbps Link Bandwidth

(d) 20% (inter-domain) - 80% (intra-domain) Traffic Pattern - 1 Gbps Link Bandwidth

(e) 50% (inter-domain) - 50% (intra-domain) Traffic Pattern - 1 Gbps Link Bandwidth

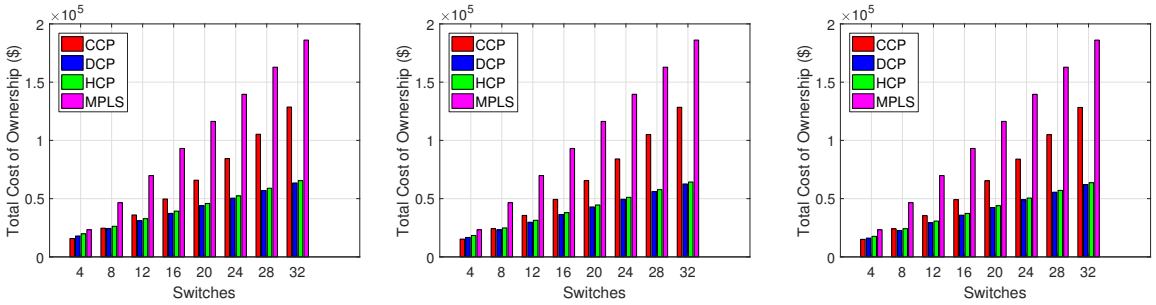
(f) 80% (inter-domain) - 20% (intra-domain) Traffic Pattern - 1 Gbps Link Bandwidth

Figure 6.3.: Total number of satisfied (controller) requests (i.e. Workload) with QoS in terms of the different switch numbers in SDN models and MPLS under the different traffic patterns with 100 Gbps and 1 Gbps link bandwidth.

In SDN models, the satisfied requests numbers represent the total number of requests that have been serviced in the corresponding models in a second by all controller(s) before rejecting a request in each switch number case. This rejection happens due

to controller message handling capacity. In each traffic pattern, the total number of satisfied requests in all SDN models show reduction while switch number increases because the network paths that are set up by the controllers become longer. Therefore, controllers need to handle more overhead messages per path setup. In addition, as traffic becomes more inter-domains, total satisfied request numbers also reduce in all SDN models because paths become longer, which also results in more overhead in control planes that controllers need to deal with. In MPLS case, the network is able to serve requests at least HCP model in 100 Gbps case, which gives the best result in SDN case. Although MPLS could serve more due to no controller capacity constraint and enough bandwidth on links, it has been left at the same number as HCP model. These numbers also depend on SDN controller performance. However, comparing controllers performance is out of this chapter's scope. In 1 Gbps link bandwidth case, the link bandwidth resource is exhausted before controllers reach their message handling capacity in SDN models. In this part of the experiment, the total number of satisfied requests show the tendency of increase in SDN and MPLS cases while switch number increases because adding more switches in the topology results in more link connectivities. This increases the number of possible end-to-end paths from a source to destination that can be used for a request. However, the number of satisfied requests is less than CCP because the flooding of available link bandwidth information (i.e. link state advertisements) and tunnel refresh messages (i.e. overhead) also consumes usable link bandwidth in MPLS case. The difference between CCP and MPLS regarding the total number of satisfied requests increases as the switch and link numbers increase because the aforementioned advertisement and refresh messages increase as well.

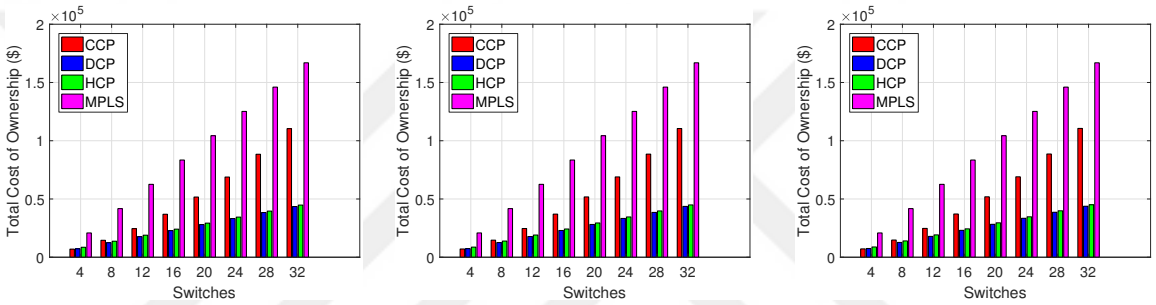
Fig. 6.4 shows the relation between the TCO with respect to the different switch numbers in SDN models and MPLS under the different traffic patterns. This experiment also consists of two parts: 100 Gbps (Fig. 6.4a, 6.4b, 6.4c) and 1 Gbps (Fig. 6.4d, 6.4e, 6.4f) link bandwidth parts. Main drivers of TCO are CAPEX and OPEX. Therefore, it is considered that TCO is the sum of CAPEX and OPEX. OPEX cost



(a) 20% (inter-domain) - 80% (intra-domain) Traffic Pattern - 100 Gbps Link Bandwidth

(b) 50% (inter-domain) - 50% (intra-domain) Traffic Pattern - 100 Gbps Link Bandwidth

(c) 80% (inter-domain) - 20% (intra-domain) Traffic Pattern - 100 Gbps Link Bandwidth



(d) 20% (inter-domain) - 80% (intra-domain) Traffic Pattern - 1 Gbps Link Bandwidth

(e) 50% (inter-domain) - 50% (intra-domain) Traffic Pattern - 1 Gbps Link Bandwidth

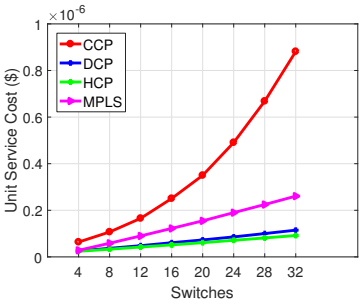
(f) 80% (inter-domain) - 20% (intra-domain) Traffic Pattern - 1 Gbps Link Bandwidth

Figure 6.4.: Total Cost of Ownership (TCO) with respect to the different switch numbers in SDN models and MPLS under the different traffic patterns with 100 Gbps and 1 Gbps link bandwidth.

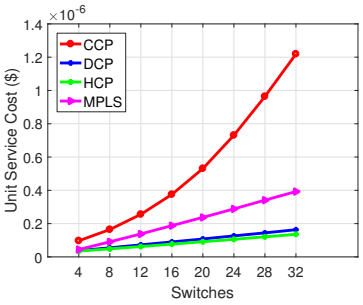
is mostly dominated by the total number of messages (overhead + workload) handled by the controller(s) in SDN case as discussed in Subsection 6.8.1.2.1. In this experiment, when controllers reach their maximum throughput point, which happens in the first part of the experiment, more switches have been added to the network and started sending traffic. It has been assumed that a controller's port number is the same as the number of switches it manages. Therefore, when new switches are added to the network, the current controller either needs to be upgraded (i.e. replaced with a new one with enough ports) or a new controller needs to be added, depending on the model, in the network. Since CCP model has only one controller,

then the current controller needs to be upgraded. However, this upgrade brings exponential CAPEX addition since the previous controller is not used anymore. On the other hand, in DCP and HCP models, current controllers can still be used while adding new controllers in the network. This brings fewer expenses compared to CCP model case. This fact is the reason for a fast increase in TCO of CCP model under all traffic patterns and link bandwidth types, which is an example for $\mathbb{C}_{\Delta_\delta}$ as well discussed in Subsection 6.8.1.1.3. Furthermore, TCO of HCP model is more than DCP model because HCP model handles more workload (W) than DCP model, see Eq. 6.7, and there is extra master controller cost in HCP model. On the other hand, MPLS shows more TCO compared to all SDN models because of its OPEX, which is not programmable and does not bring any cost reduction. In the second part of the experiment, TCO shows reduction in all SDN models and MPLS under all traffic patterns because CAPEX reduces due to link cost and OPEX reduces due to the number of workload and overhead.

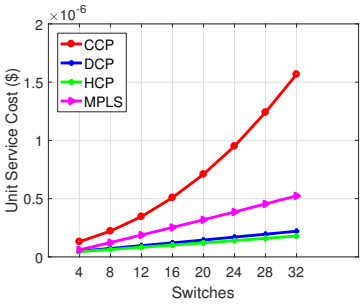
Fig 6.5 shows the relation between the unit service cost with respect to the different switch numbers in SDN models and MPLS under the different traffic patterns. This experiment has two parts as well: 100 Gbps and 1 Gbps link bandwidth parts. In the 100 Gbps link bandwidth part (Fig. 6.5a, 6.5b, and 6.5c), while CCP shows the highest unit service cost among all models, DCP gives higher unit service cost than HCP based on one-to-one point comparison of curves. Although MPLS and HCP have the same and the highest number of satisfied requests, MPLS gives higher unit service cost than DCP and HCP because its CAPEX and OPEX is higher than that of DCP and HCP under all traffic patterns. Furthermore, both SDN models and MPLS result in lower unit service costs as the traffic becomes more local (i.e. intra-domain) because more requests are satisfied as explained previously. The unit service cost increases while the number of switches increase because CAPEX and OPEX increase and total satisfied number of requests decreases in both SDN models and MPLS as implied by the Eq. 6.1. In 1 Gbps link bandwidth part (Fig. 6.5d, 6.5e, and 6.5f), MPLS gives higher unit service cost than SDN models because it shows the lowest



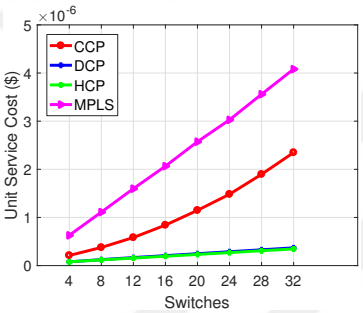
(a) 20% (inter-domain) - 80% (intra-domain) Traffic Pattern - 100 Gbps Link Bandwidth



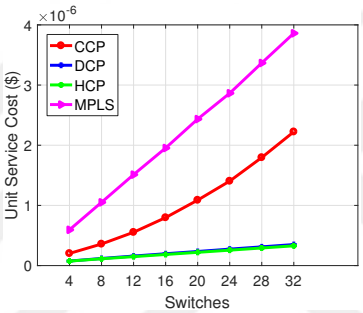
(b) 50% (inter-domain) - 50% (intra-domain) Traffic Pattern - 100 Gbps Link Bandwidth



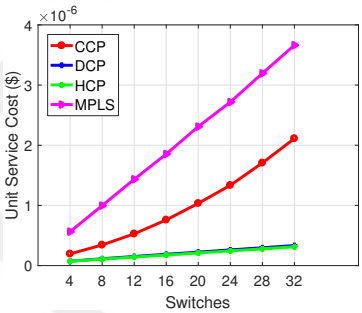
(c) 80% (inter-domain) - 20% (intra-domain) Traffic Pattern - 100 Gbps Link Bandwidth



(d) 20% (inter-domain) - 80% (intra-domain) Traffic Pattern - 1 Gbps Link Bandwidth



(e) 50% (inter-domain) - 50% (intra-domain) Traffic Pattern - 1 Gbps Link Bandwidth



(f) 80% (inter-domain) - 20% (intra-domain) Traffic Pattern - 1 Gbps Link Bandwidth

Figure 6.5.: Unit service cost with respect to the different switch numbers in SDN models and MPLS under the different traffic patterns with 100 Gbps and 1 Gbps link bandwidth.

number of satisfied requests. CCP gives the highest unit service cost among SDN models while HCP gives slightly lower cost than DCP under all traffic patterns. Both SDN models and MPLS unit service cost results are similar (due to a similar number of satisfied requests) under all traffic patterns because link bandwidth is exhausted before controller capacity. Therefore, traffic pattern has little to no effect on unit service cost in 1 Gbps link bandwidth case. The CAPEX and OPEX increase ratio is faster compared to total number of satisfied requests in both SDN models and MPLS as the number of switches increase. Therefore, the unit service cost also increases while the number of switches increase. Finally, the unit service cost is higher for

both SDN models and MPLS in 1 Gbps link bandwidth case compared to 100 Gbps link bandwidth case in each switch case due to the total number of satisfied requests, CAPEX, and OPEX results.

6.8.1.4 Network OPEX Activities

Table 6.2 shows the activities along with their *duration* and *frequency* attributes and their values in a network. These activities are those that can be different in terms of their duration and frequency in a programmable network and traditional (non-programmable) network.

Table 6.2.: List of activities along with their attributes (i.e. duration and frequency) and their values considered in calculation of OPEX (Activity-based approach).

Activity Name	**Duration**		**Frequency**	
	SDN	**MPLS**	**SDN**	**MPLS**
Service Provisioning	1 hour	8 hours	per connection	per connection
Device-Level Configuration	1 hour	4 hours	per 9K connections per device	per 2K connections per device
Topology-based Modification	1 hour	4 hours	when new resource added	when new resource added
Fault Detection	30 mins	2 hours	Every 9K hours per device	Every 2K hours per device
Fault Reparation	2 hours	10 hours	Every 9K hours per device	Every 2K hours per device

It is difficult to gather precise input values for these parameters because they are proprietary and relative and companies are not willing to publicly share them either. These value assumptions constitute an average of each parameter based on Internet research, literature review ([250–254]), and our discussions. “Duration” refers to the time that an activity takes to be finished while “Frequency” refers to the number of times that an activity occurs in a network. In other words, duration refers to “how long”, frequency refers to“how many times” questions, respectively.

"Service Provisioning" activity refers here to providing a service first time to an end-user (or customer) while the service had already been implemented/created in the network. This activity includes all tasks that are necessary to start, update, test and/or stop the customer's service. Sample tasks in this activity may be, not limited to, updating necessary databases for the customer, traveling to site to configure corresponding cabinets on the field, and so on. This activity is applied for each connection request in a network. We assume this activity takes much less time in a SDN network compared to MPLS network due to the automation feature that programmability adds to the network.

"Device-level Configuration" activity can be characterized by converting network-wide policies to device-level configurations. These configurations are different than and/or independent of customer service provisioning requests and applied on the current network devices. Some of the sample tasks in this activity may be, not limited to, routing protocol-based configurations such as updating BGP preferences, or OSPF/IS-IS routing algorithm etc., resource management-based updates to devices, updating ACLs, VLANS for packet filtering, testing devices, and so on. In SDN case, network devices are simple devices without any intelligence (i.e. no routing protocols etc.). Also, they are accessible and configurable from a remote control points (i.e. controllers). Therefore, we assume it takes less time to configure a network device compared to traditional network devices. In traditional networking case, devices are more complex due to the intelligence. It is not easy, if not impossible, to access and configure from a remote point. Also, configuring a device may require some other configurations in neighboring devices for a complete and accurate configuration. Therefore, we assume it takes more time to configure a network device compared to SDN devices. In addition, we assume that these types of configurations may be needed more frequent in traditional network devices case compared to SDN case because traditional devices are complex. Therefore, they may require more interventions per device from network administrators to efficiently operate in the network.

Tasks in "Topology-based Modifications" activity can be the similar or same as in the "Device-level Configuration" case. However, this activity is triggered only when a new device and/or link is added (connected), removed, and replaced in the network. Also, any modification in the network topology can require configurations by administrators in other (particularly nearby/neighbor) devices to ensure the coordination among the network devices. As explained in the "Device-level Configuration" activity, we assume it takes longer in MPLS case compared to SDN case due to the same reasons. We assume that certain number of connections in the network result in topology based modifications (adding/replacing new device(s)/link(s)) in the network. This modifications can be due to limited memory size of the devices, or adding network resources such as bandwidth etc., due to inefficient resource management capabilities of the network architecture and so on. We trigger this activity in this study when there is not enough resources and therefore new links and devices are added to the network in the experiments.

"Fault Detection" activity does not refer to the path failure detection problem, which is well-studied and done in milliseconds in both SDN and traditional networks. Detecting a network failure is the first step to recover from it. However, the failure detection or identifying the failure (i.e. reason(s), location, type, time etc.) may not be a straightforward task in a network. Although there are certain dedicated software tools (e.g. SNMP) to help detect a failure, they may not help pinpoint the aforementioned attributes of the failure since they usually throw a simple alarm(s) when certain monitored network parameters exceed predefined thresholds. However, this process can be done in different duration and with(out) human-device interaction depending on the network architecture. For example, since SDN provides real-time global full network view on network devices and links, this process can be shorter and easier compared to a traditional architecture. Some sample tasks in this activity may be, not limited to, checking network cable(s)/devices with physically or through software tools using probing techniques etc. to pinpoint it, checking network log files, and so on. Frequency attribute of this activity represents the MTBF (Mean Time

Between Failure) and we assume longer MTBF value for traditional devices owing to the complexity they come with and overhead they handle in a network compared to SDN devices.

"Fault Reparation" activity includes tasks necessary to fix a detected failure in the network. After being informed and detected a failure, following tasks may be applied depending on the network architecture: Analysis (interacting multiple devices personally on the site or using a controller), travel by technicians (to the place of failure), fixing the failure (with device configuration), testing devices to verify the repair, and so on. Duration attribute of this activity represents the MTTR (Mean Time To Repair) and we assume longer MTTR value for traditional devices due to the complexity and proprietary status compared to SDN devices.

These activities are not the only ones in a network and may not be precise regarding their duration and frequency. Network administrators can classify and define different activities depending on their network. Furthermore, the granularity level of activities may result in discrepancy in calculation of OPEX cost in the network.

6.8.1.5 Calculation of OPEX: Activity-based Approach

OPEX are the ongoing costs and contribute to the operational costs of a network to keep its operations (e.g. technical, commercial, and administrative) running on a daily basis. These expenses widely vary depending on the network. OPEX is more complicated to calculate than CAPEX because it requires more information about internal network dynamics. However, such information is proprietary and highly hidden by network owners.

This method of OPEX calculation utilizes an activity-centric approach to calculate OPEX in a network. Such an activity-based approach would help network administrators capture savings on OPEX of their networks owing to the reasons explained earlier. To this end, this approach defines a set of cost groups, G, as follows:

$$G = \{g_1, g_2, ..., g_n\} = \{g_i | 1 \leq i \leq n\}$$

where g_i is a cost group considered in OPEX calculation in a network. There are four cost groups in OPEX, as seen in Fig. 6.6, in this study: Service Provisioning and Management cost group, Maintenance cost group, First Time Setup cost group, and Reparation cost group.

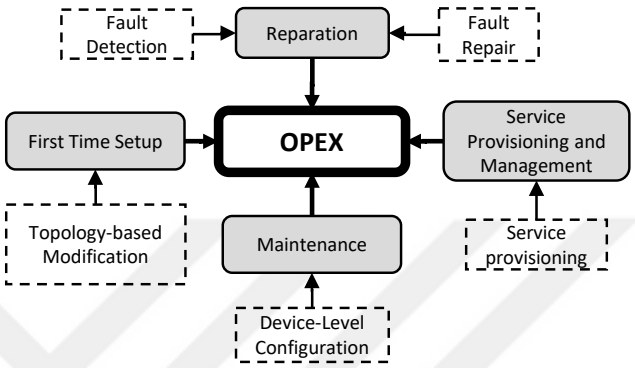


Figure 6.6.: OPEX cost groups and corresponding activities in activity-based approach. The dashed rectangles represent the input activities (in OPEX) to the corresponding cost groups represented as gray rounded rectangles. The arrows point to the direction of the input.

Similarly, this study defines and describes activities, as seen in Table 6.2, that take place to keep the network operational. These activities can be various and different in terms of time it takes to complete or number of occurrences in a network. A set of activities is defined, A, as follows:

$$A = \{a_1, a_2, ..., a_m\} = \{a_j | 1 \leq j \leq m\}$$

where a_j is an activity considered in OPEX calculation in a network. There are five activities in this study. Fig. 6.6 also shows which activities are input (i.e. involved) into which cost groups in this study.

Finally, another set, E, is defined for the employees of the network as follows:

$$E = \{e_1, e_2, ..., e_r\} = \{e_k | 1 \leq k \leq r\}$$

where e_k is an employee of the network.

After defining these sets, cost of an activity can be calculated as in Eq. (6.10):

$$\mathcal{C}_{a_j} = \sum_{k \in e_{a_j}}^{|e_{a_j}|} {}_{a_j}\Delta t_{e_k} * p_{e_k} \tag{6.10}$$

where $\mathcal{C}_{a_j}$, e_{a_j}, $|e_{a_j}|$, ${}_{a_j}\Delta t_{e_k}$, and p_{e_k} represent total cost of an activity a_j, set of employees who have done activity a_j, number of employees who have done activity a_j, amount of time that employee e_k spends in/for activity a_j, and pay rate of employee e_k, respectively. ${}_{a_j}\Delta t_{e_k}$ represents the duration attribute of an activity shown in Table 6.2.

On the other hand, after substituting Eq. (6.10) in Eq. (6.11), cost of a cost group can be calculated as in Eq. (6.11):

$$\begin{aligned} \mathcal{C}_{g_i} &= \sum_{j \in a_{g_i}}^{|a_{g_i}|} \mathcal{C}_{a_j} * {}_{g_i}\Delta f_{a_j} \\ &= \sum_{j \in a_{g_i}}^{|a_{g_i}|} \sum_{k \in e_{a_j}}^{|e_{a_j}|} {}_{a_j}\Delta t_{e_k} * p_{e_k} * {}_{g_i}\Delta f_{a_j} \end{aligned} \tag{6.11}$$

where $\mathcal{C}_{g_i}$, a_{g_i}, $|a_{g_i}|$, and ${}_{g_i}\Delta f_{a_j}$ represent total cost of a cost group g_i, set of activities involved in expenditure group g_i, number of activities involved in expenditure group g_i, and how many times an activity a_j is applied in expenditure group g_i, respectively. ${}_{g_i}\Delta f_{a_j}$ represents the frequency attribute of an activity shown in Table 6.2.

Finally, after substituting Eq. (6.11) in Eq. (6.12), OPEX can be calculated as the summation of these cost groups as in the Eq. (6.12):

$$\begin{aligned} \mathbb{O} &= \sum_{i=1}^{|G|} \mathcal{C}_{g_i} \\ &= \sum_{i=1}^{|G|} \sum_{j \in a_{g_i}}^{|a_{g_i}|} \sum_{k \in e_{a_j}}^{|e_{a_j}|} {}_{a_j}\Delta t_{e_k} * p_{e_k} * {}_{g_i}\Delta f_{a_j} \end{aligned} \tag{6.12}$$

where $|G|$ is the number of cost groups involved in OPEX calculation for the network.

6.8.1.6 Calculation of OPEX: Activity-based Approach - Evaluation

This section presents numerical results to provide some insights in order to understand the economic impact of SDN architecture with different control plane models (CCP, DCP, and HCP) and MPLS architecture on unit service cost. In this context, this work analyzes unit service cost and service introduction cost for the video service using total number of satisfied requests, CAPEX and OPEX in different SDN models and MPLS architecture.

6.8.1.6.1 Experimental Setup Since the experimental setup is the same as in the Subsection 6.8.1.3.1, the details are not given here.

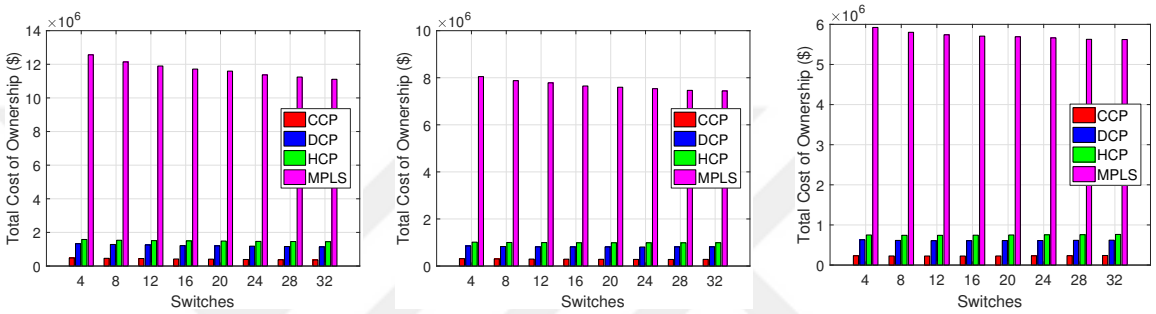
Table 6.3 lists the parameters and their values used in calculation of CAPEX and OPEX (activity-based approach) in this study. It is difficult to gather precise input values for some of these parameters because they are proprietary and companies are not willing to publicly share them. These value assumptions constitute an average of each parameter based on Internet research, literature review ([20, 21, 209, 212, 240–242]), and research discussions. Although we should note that these input numbers may not be reflecting precise and/or realistic values, they should not impact the nature of the calculation framework since these values are very relative for every network company. We have used the same cost for all links ($\mathcal{C}_l$) in both SDN models and MPLS case. However, it has been assumed that \$100 and \$750 for a link cost in 1 Gbps and 100 Gbps link bandwidth cases, respectively. We have assumed that device cost is double ($\mathcal{C}_d$) in MPLS case since traditional network equipment is expected to be more expensive than SDN equipment due to integrated control plane (i.e. proprietary software implementation). We have also assumed the same cost, which is proportional to the number of network devices ($|d|$) and that of controllers ($|c|$) in different device number cases, for a controller hardware ($\mathcal{C}_c$) and software ($\mathcal{C}_s$) in all models. We have assumed that an activity a_j is always done by 2 employees and the pay rate of an employee is \$60 per hour in any activity and SDN and MPLS cases. We have used the data provided in [250] for the total time spent in corresponding service introduction

Table 6.3.: List of parameters and their values used in calculation of CAPEX and OPEX (activity-based approach).

Parameter	**Value**			
	CCP	**DCP**	**HCP**	**MPLS**
$\|d\|$	4, 8, 12, 16, 20, 24, 28, 32	4, 8, 12, 16, 20, 24, 28, 32	4, 8, 12, 16, 20, 24, 28, 32	4, 8, 12, 16, 20, 24, 28, 32
$\mathcal{C}_d$	\$750	\$750	\$750	\$2000
$\|l\|$	6, 12, 18, 24, 30, 36, 42, 48	6, 12, 18, 24, 30, 36, 42, 48	6, 12, 18, 24, 30, 36, 42, 48	6, 12, 18, 24, 30, 36, 42, 48
$\mathcal{C}_l$	\$500	\$500	\$500	\$500
$\|c\|$	1	4	$4+1$	N/A
$\mathcal{C}_c$	$\frac{\$500*\|d\|}{\|c\|}$	$\frac{\$500*\|d\|}{\|c\|}$	$\frac{\$500*\|d\|}{\|c\|}$	N/A
$\|s\|$	1	4	$4+1$	N/A
$\mathcal{C}_s$	$\frac{\$500*\|d\|}{\|c\|}$	$\frac{\$500*\|d\|}{\|c\|}$	$\frac{\$500*\|d\|}{\|c\|}$	N/A
$\|G\|$	4	4	4	4
$\|A\|$	5	5	5	5
$\|e_{a_j}\|$	2	2	2	2
p_{e_k}	\$60	\$60	\$60	\$60
$MTBF$	9000 hours	9000 hours	9000 hours	2000 hours
$MTTR$	2 hours	2 hours	2 hours	10 hours
e^I, p^I, t^I	2, \$60, 14	2, \$60, 14	2, \$60, 14	2, \$60, $6 * \|d\|^{\left(\frac{\log_{10} 0.7}{\log_{10} 2}+1\right)}$
e^E, p^E, t^E	2, \$60, 3.5	2, \$60, 3.5	2, \$60, 3.5	2, \$60, $4 * \|d\|^{\left(\frac{\log_{10} 0.7}{\log_{10} 2}+1\right)}$
e^T, p^T, t^T	2, \$60, 3.5	2, \$60, 3.5	2, \$60, 3.5	2, \$60, $3 * \|d\|^{\left(\frac{\log_{10} 0.7}{\log_{10} 2}+1\right)}$

steps of SDN models. Authors in [250] state that these values would be the same for different switch cases and/or control plane models owing to the automation feature in SDN. In MPLS, we have used a 70% learning curve-based [255] and switch number proportional timing values for each service introduction step in all switch cases to make it more realistic. Time values in MPLS are more compared to SDN case due to mostly manual configurations over multiple heterogeneous devices. Finally, these values are the same for the all traffic patterns and link bandwidth cases in both SDN models and MPLS.

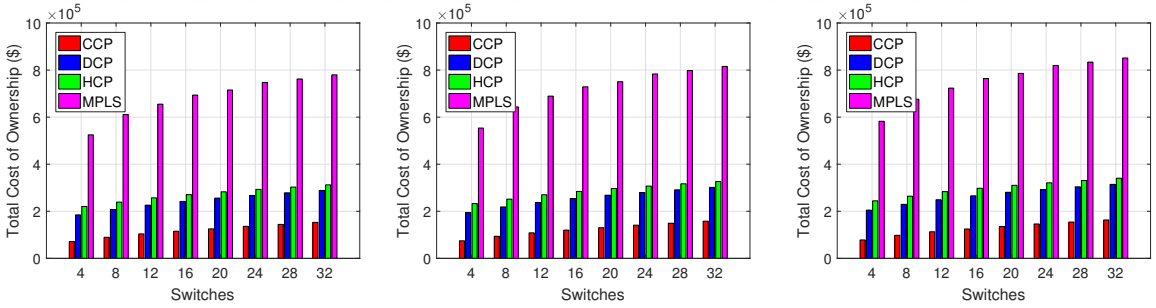
6.8.1.6.2 Experimental Results These results are based on the CAPEX calculation formulas presented in Subsection 6.8.1.1, OPEX calculation formulas presented in Subsection 6.8.1.5, and values presented in Table 6.2 and Table 6.3. The total number of satisfied requests are the same as in Fig. 6.3 for each switch cases under all traffic patterns and link bandwidth cases due to the same experimental setup. Therefore, the results are not presented here again.



(a) 20% (inter-domain) - 80% (intra-domain) Traffic Pattern - 100 Gbps Link Bandwidth

(b) 50% (inter-domain) - 50% (intra-domain) Traffic Pattern - 100 Gbps Link Bandwidth

(c) 80% (inter-domain) - 20% (intra-domain) Traffic Pattern - 100 Gbps Link Bandwidth



(d) 20% (inter-domain) - 80% (intra-domain) Traffic Pattern - 1 Gbps Link Bandwidth

(e) 50% (inter-domain) - 50% (intra-domain) Traffic Pattern - 1 Gbps Link Bandwidth

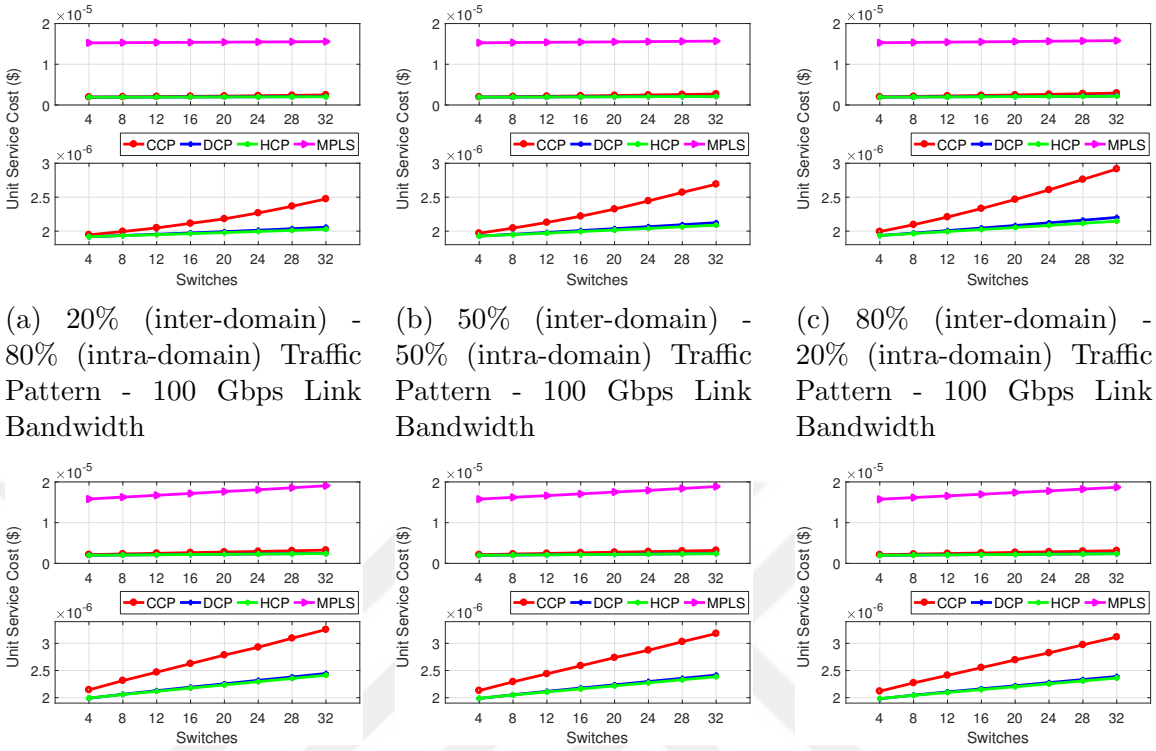
(f) 80% (inter-domain) - 20% (intra-domain) Traffic Pattern - 1 Gbps Link Bandwidth

Figure 6.7.: Total Cost of Ownership (TCO) with respect to the different switch numbers in SDN models and MPLS under the different traffic patterns with 100 Gbps and 1 Gbps link bandwidth.

Fig. 6.7 shows the relation between the TCO with respect to the different switch numbers in SDN models and MPLS under the different traffic patterns. This exper-

iment also consists of two parts: 100 Gbps (Fig. 6.7a, 6.7b, 6.7c) and 1 Gbps (Fig. 6.7d, 6.7e, 6.7f) link bandwidth parts. Main drivers of TCO are CAPEX and OPEX. Therefore, we consider that TCO is the sum of CAPEX and OPEX. In this experiment, when controllers reach their maximum throughput point, which happens in the first part of the experiment, we have added more switches to the network and started sending traffic. In 100 Gbps link bandwidth case, TCO of SDN models and MPLS shows a reduction because OPEX reduces in each switch case. The reason behind the OPEX reduction is the decrease on the total number of satisfied connections as shown in Fig. 6.3 and dependency of total cost of activities on the connections. TCO of SDN models and MPLS also decreases as traffic becomes more inter-domain centric because the total number of satisfied connections shows reduction as well. TCO of HCP model is higher than DCP and CCP models because it handles more connections (i.e. workload (W)) among all and has extra master controller hardware and software cost. Also, although it has the same number of connections MPLS reflects more TCO than all SDN models because activities in MPLS cost more due to longer duration compared to all SDN models. In 1 Gbps link bandwidth case, TCO of SDN models and MPLS are less compared to 100 Gbps link bandwidth case because CAPEX is less due to link cost and the total number of connections decrease in this case. However, TCO of SDN models and MPLS shows an increase as the switch number increases because total connections increase which result in more OPEX in the network. Finally, as explained in the previous link bandwidth case, TCO in MPLS is more than all SDN models because activities in MPLS cost more due to longer duration compared to all SDN models.

Fig 6.8 shows the relation between the unit service cost with respect to the different switch numbers in SDN models and MPLS under the different traffic patterns. In each figure, while the top-subfigure shows the corresponding results to compare for all SDN models and MPLS, the bottom-subfigure shows the corresponding results to make the differences visible only for SDN models due to figure scaling. This experiment has two parts as well: 100 Gbps and 1 Gbps link bandwidth parts. In



(a) 20% (inter-domain) - 80% (intra-domain) Traffic Pattern - 100 Gbps Link Bandwidth

(b) 50% (inter-domain) - 50% (intra-domain) Traffic Pattern - 100 Gbps Link Bandwidth

(c) 80% (inter-domain) - 20% (intra-domain) Traffic Pattern - 100 Gbps Link Bandwidth

(d) 20% (inter-domain) - 80% (intra-domain) Traffic Pattern - 1 Gbps Link Bandwidth

(e) 50% (inter-domain) - 50% (intra-domain) Traffic Pattern - 1 Gbps Link Bandwidth

(f) 80% (inter-domain) - 20% (intra-domain) Traffic Pattern - 1 Gbps Link Bandwidth

Figure 6.8.: Unit service cost with respect to the different switch numbers in SDN models and MPLS under the different traffic patterns with 100 Gbps and 1 Gbps link bandwidth. In each figure, while the top-subfigure shows the corresponding results to compare for all SDN models and MPLS, the bottom-subfigure shows the corresponding results to make the differences visible only for SDN models due to figure scaling.

the 100 Gbps link bandwidth part (Fig. 6.8a, 6.8b, and 6.8c), unit service cost for all SDN models and MPLS increases in each switch case although total TCO and satisfied connections decrease as well. This happens due to the fact that the decrease ratio of workload is more than that of TCO. Although MPLS and HCP have the same and the highest number of satisfied requests, MPLS gives higher unit service cost than all SDN models because its CAPEX and OPEX is higher than that of SDN models under all traffic patterns. While CCP shows the highest unit service

cost among all SDN models, DCP gives slightly higher unit service cost than HCP based on one-to-one point comparison of curves due to differences in total connections handled in corresponding models. Furthermore, both SDN models and MPLS result in very slightly higher unit service costs as the traffic becomes more non-local (i.e. inter-domain) because less requests are satisfied as explained previously. In 1 Gbps link bandwidth part (Fig. 6.8d, 6.8e, and 6.8f), MPLS also gives a higher unit service cost than SDN models because it shows the lower number of satisfied requests and higher TCO compared to SDN models. CCP gives the highest unit service cost among SDN models while HCP gives slightly lower cost than DCP under all traffic patterns. Both SDN models and MPLS unit service cost results are similar (due to a similar number of satisfied requests) under all traffic patterns because link bandwidth is exhausted before controller capacity. Therefore, traffic pattern has little to no effect on unit service cost in 1 Gbps link bandwidth case. The CAPEX and OPEX increase ratio is faster compared to total number of satisfied connections in both SDN models and MPLS as the number of switches increase. Therefore, the unit service cost also increases while the number of switches increase. Finally, the unit service cost is higher for both SDN models and MPLS in 1 Gbps link bandwidth case compared to 100 Gbps link bandwidth case in each switch case due to the total number of satisfied requests, CAPEX, and OPEX results as implied by Eq. 6.1.

6.8.2 Cost-to-Service Metric

Programmable networks (e.g. SDN) bring standardized and programmatic interfaces (e.g. OpenFlow) that provide automation in network operations such as configuration across multiple, heterogeneous devices and flow management for efficient resource utilization. They also minimize human intervention in network operations, which helps in reducing network OPEX. This automation increases service velocity, streamlines service introduction, and fosters innovative applications and services. Also, if it is combined with NFV, very agile service creation can be achieved,

as network functions can be dynamically started and logically chained to compose customized end-to-end network services. On the other hand, in the traditional non-programmable networks (e.g. MPLS), there are a number of devices requiring different skill sets, including different technicians, programmers, and customer care personnel due to lack of standard programmable interfaces. They do not provide the flexibility necessary to make dynamic network changes and create new service offerings. Any changes to these networks are difficult, slow, and risky. Therefore, programmable networking helps reduce the costs of network operations and time required for introduction of a new service.

Network operators may decide to introduce new services for their users for different purposes such as generating new revenue opportunities. This process includes various steps before making the service fully operational. We define these steps as Service Design/Implementation (I), Service Testing (E), and Service Tuning Up (T) in this study.

Service Design/Implementation phase mostly includes planning the configuration specifics for each network entities and sites. This planning involves design details of network elements and sites including port mappings, interface naming, host naming, IP addressing, VLAN addressing and many more. A proper design process is crucial for the continuity and timeliness of the whole service introduction without any errors. In addition, implementing planned design may require necessary coding over various network entities such as network devices, controllers, databases, and management systems depending on the technology, installation, and configuration of equipment and files. In SDN case, these actions/behaviors can be minimized because such actions do not necessarily have to be taken for all network entities. Instead, applying necessary actions once centrally and then distributing them to the relevant network devices saves number of employees and time spent for the service introduction. Service Testing phase aims at detecting network configuration issues causing faulty service functioning and service quality degradation, which may result in revenue losses. This testing process can be automatically conducted by exploiting network programming

or manually as the technology allows. In programmable networking, it is possible to program and test every single element in the network remotely and quickly compared to traditional networking, which also saves the number of employees necessary and time to spend for testing purposes for a fast service introduction. Finally, Service Tuning Up phase includes final touches necessary to fix the detected issues from the testing phase and maximize the service quality to generate/retain the revenue from the service. Programmable networks also save costs in this phase due to the automation it provides as in the previous step.

Each step explained above brings its own expenses to the total cost of service introduction process. In this context, we describe a metric called *Cost-to-Service* to economically quantify the cost of introducing a new service in a network technology. To monetize the cost of the service introduction, we define this metric as the total cost of each step described above where the cost of each is the function of number of employees (e) (worked), employee pay rate per hour (p), and time in hours (t) (spent by employees) in each step. Therefore, this metric can be written as in Eq. 6.13:

$$\begin{aligned} \mathfrak{S}_{\mathfrak{C}} &= f(e,p,t) = \mathcal{C}^I + \mathcal{C}^E + \mathcal{C}^T \\ &= \sum_{i=1} e_i^I p_i^I t_i^I + \sum_{j=1} e_j^E p_j^E t_j^E + \sum_{k=1} e_k^T p_k^T t_k^T \end{aligned} \tag{6.13}$$

where subscripts i, j, and k represent the corresponding employees involved in and $\mathcal{C}^I$, $\mathcal{C}^E$, and $\mathcal{C}^T$ are the cost of each corresponding steps.

6.8.2.1 Cost-to-Service Metric Evaluation

The values used for each steps descbribed aobve are shown in Table 6.3. Fig. 6.9 shows the service introduction cost with respect to the different switch numbers in SDN models and MPLS. The cost results are the same for the different traffic patterns and link bandwidth cases as well because they have no effect on service introduction steps described in Subsection 6.8.2. In MPLS, the service introduction cost increases

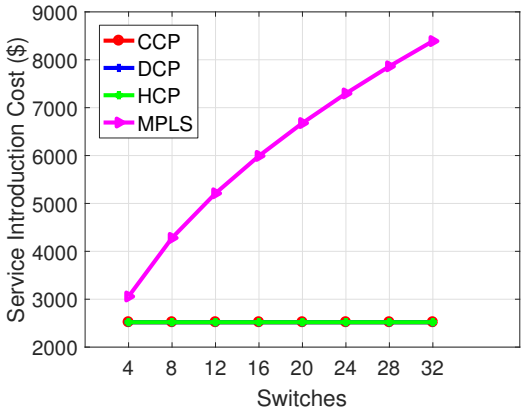


Figure 6.9.: Service introduction cost with respect to the different switch numbers in SDN models and MPLS. The results are the same for the different traffic patterns and link bandwidth cases as well.

as the number of switches increase because we have used a 70% learning curve-based and number of devices ($|d|$) proportional time frames spent in corresponding steps by each employee. The time spent in corresponding service introduction steps are the same according to the study presented in [250] in SDN models. Therefore, they give the same results for each switch case. Based on the values assumed, MPLS gives the highest service introduction costs among all due to lack of automation and programmability, which is reflected in time.

6.9 Chapter Summary

SDN paradigm has several key attributes that have an impact on the CAPEX and OPEX equations of a network. It has got the attention of researchers from both academia and industry as a means to be leveraged in order to decrease network costs and generate revenue for service providers due to features it promises in networking. This study has investigated how programmable network architectures, i.e. SDN technology, affects the network economics compared to traditional network architectures, i.e. MPLS technology. To this end, this work has defined two metrics, *Unit Service Cost Scalability* and *Cost-to-Service*, to evaluate how SDN architecture performs com-

pared to MPLS architecture. It has also presented mathematical models to calculate certain cost parts of a network. In addition, it has compared different popular SDN control plane models, Centralized Control Plane (CCP), Distributed Control Plane (DCP), and Hierarchical Control Plane (HCP), to understand the economic impact of them with regards to the defined metrics. The simulation results have revealed that MPLS shows more TCO compared to all SDN models because of its OPEX, which is not programmable and does not bring any cost reduction. TCO of HCP model is more than DCP model because HCP model handles more workload than DCP model and there is extra master controller cost in HCP model. In addition, CCP shows the highest unit service cost because it results in more CAPEX and least workload among all models. On the other hand, HCP results in the lowest unit service cost because it handles the most number of workload. The results have also demonstrated that as the number of switches increases the unit service cost increases as well because the total number of satisfied requests are decreasing due to longer paths in both SDN models and MPLS. Furthermore, it has been shown that MPLS gives the higher service introduction costs compared to SDN models owing to lack of automation and programmability, which is reflected in time. In SDN case, on the other hand, the number of network elements, such as controllers, impact the total cost of service introduction because of the time spent in each service introduction step. These results have pointed out that programmability has great impact on network economics.

7 PRICING END-USERS FOR NETWORK SERVICES IN SDN

7.1 Abstract

Although many solutions have been proposed from both industry and academia, pricing the Internet services is still an ongoing research problem for researchers. This study proposes an optimal pricing scheme for inter-AS traffic requests with QoS made by customers in SDNs. This pricing scheme exploits the Nash bargaining problem which aims to maximize benefits of both service providers and customers. This work integrate a new cost function and network connectivity degree factor into the proposed pricing scheme. Also, it gives a general scheme of revenue and profit that a service provider makes. This scheme employs the idea of *penalty* for each request that a service provider cannot provide for its customers. Furthermore, it applies these schemes in the scalable hierarchic architecture, presented in Chapter 4, with extensive experiments.

7.2 Introduction

The Internet consists of thousands of networks which are directly or indirectly interconnected to each other. Exchanging traffic between two networks in the Internet are regulated by the economic arrangements like “peering” and “transit”. In a peering case, two or more independent networks directly connect to each other with a promise of no charging for traffic exchanged among them. In the latter case, a network (usually smaller) purchases a service from another network to carry its traffic to others in return for a service fee.

Today, there are mainly two types of charging systems available by networks. In the first case, customers (i.e. end-users) are charged based on a flat rate, mostly

called "capacity-based pricing", regardless of the amount of traffic they receive or send over a time period (e.g. a month). An ISP prices its service tiers based on a QoS parameter(s) (usually bandwidth), and customers are allowed to receive and/or send traffic up to that QoS rate. In this type of pricing scheme, ISPs that sell traffic services to others may not be receiving balanced traffic requests from all purchasing-networks. This may happen because users of a purchasing-ISP may load the upstream ISP up with traffic. On the other hand, the second type of charging system is based on usage of customers, called "usage-based pricing", and mostly uses the 95-percentile pricing method. In this case, the bandwidth consumed by a customer is measured in time intervals (e.g. 5-minutes) and are sorted according to the Mbps rates consumed. The 5% of the highest rates are discarded and the Mbps values that have to be paid are determined. Therefore, a customer only pays for 95% of his/her usage [256].

Charging an end user is still a research problem from a service provider perspective since it requires consideration of many aspects of a network (e.g. cost), user satisfaction, and other networks' services over an end-to-end (e2e) path purchased by the service provider. This charging should satisfy both end-users and service providers with optimum prices for corresponding services. An optimal price is usually a result of a negotiation process between customers and service providers. On the other hand, charging between service providers (p2p) is similar to the customer-service provider (c2p) case since they are also customers of each other.

The study in this chapter exploits and builds on the idea presented in [257]. This chapter proposes an optimal pricing scheme for a service request with QoS in SDN environment using the Nash bargaining problem, which aims to maximize benefits of both service providers and customers. It integrates a new cost function and network connectivity degree factor into the proposed pricing scheme. Although the idea in [257] is for end users (customers)-service provider (c2p) relation, this work applies the same idea to service provider (customer)-service provider (p2p) relation and adapt a new cost function. It also integrates the idea of network connectivity degree factor into the pricing scheme. In addition, it gives a general scheme of revenue

and profit that a service provider makes. This scheme employs the idea of *penalty* for each request that the service provider cannot provide to its customers. Furthermore, this work applies these schemes in the scalable SDN-based hierarchic architecture [84] and evaluate with extensive experiments.

The next section gives a snapshot of papers that study pricing schemes of providers on the Internet. Section 7.4 describes the SDN-centric scalable hierarchic architecture in which the proposed pricing scheme is applied. Section 7.5 first defines the price optimization problem in the context of the Nash bargaining process, and adapt the cost function and network connectivity degree parameter into the final price in corresponding subsections. Then, it show sa general scheme for network revenue and total profit. After discussing the experiment results in Section 7.6, the study is summarized with concluding remarks in Section 7.7.

7.3 Related Work

[258] proposes a protocol called "Border Pricing Protocol (BPP)" which is similar to Border Gateway Protocol (BGP) to exchange the pricing information among Autonomous Systems (AS) on the Internet. ASes can request pricing information from other ASes and store the pricing information for destinations as in BGP's path information case for destinations. The authors state that ASes need to store two different information bases: The Pricing Information Base (PIB) for price information of particular destination and Charging Information Base (CIB) for charging information of each sender.

In [259, 260], the authors study the relation between the number of ISPs and prices in localized regions. They conclude that ISPs, which are co-located, involve an inevitable customer fight due to Nash reversion as the number of local ISPs increase in the region. They also state that even if the ISPs are not located in the same regions, thereby not involved in a price war directly, they still get into process of setting transit prices for each other.

The study in [261] focuses on networks offering tiered services and prices for their customers. The authors firstly propose a solution to identify the optimal service tiers for customers. They then discuss the optimal prices for corresponding tiered services by exploiting game theoretical methods to satisfy both users and service providers. [262] also studies the similar problem for service providers and end-users. A model for transit traffic is proposed so that ISPs can arrange their service tiers and corresponding prices. They point out that tiered services, which are organized based on the cost of traffic flows, is sub-optimal.

[263] examines the relation between time and prices of service providers. Their study summarizes that service providers can achieve maximum revenue and social welfare if they differentiate prices across users and time. [264] states that network resources can be used more efficiently by users through exploiting price differentiation on volume usage-based connectivity pricing. The authors claim that unbundling services might be better in access networks than core networks.

Altmann et al. [265] provides an approach to find out most economical interconnection for ISPs, especially small and medium-size ISPs, that also needs to buy services from upstream providers to provide services to their users. This purchasing gives them pressure regarding their service prices and directs them to reduce their interconnection costs. The model they propose leverages AS-level topology information, measurement information regarding bits/bytes, and pricing scheme information.

7.4 Scalable Hierarchic Architecture

This section briefly describes the SDN-based architecture and explains how it works in an hierarchic environment. The hierarchic architecture, as explain in Chapter 4, consists of levels from bottom to up. The levels can be increased, as they are needed, through up. In this version of the framework, as shown in Fig. 4.1, there are currently two level: Network-Level (bottom-level) consisting of independent ISPs/ASes which are also SDN domains with their own local controllers, and Broker-level (top-level)

consisting of a super controller, called "Broker", which acts like a supervisor for the bottom-level controllers.

The Broker receives following information from all AS controllers: 1) the advertisements for paths between border node pairs including their QoS values and Path IDs, 2) unit prices for bandwidth, 3) unit prices for delay classes, and 4) inter-connecting links between ASes. Once the Broker gets aforementioned information, it builds its own general border node-level view and AS-level view as in the Fig. 4.2 and Fig. 4.3, respectively. The Fig. 4.2 represents the available paths satisfying the requested QoS values between border nodes in ASes. On the other hand, Fig. 4.3 is the final abstract global view of the domains and represents the available ASes that have paths satisfying the requested QoS values. These views are updated by AS controllers when a change happens.

When an inter-AS request comes to an AS controller, the controller forwards the request with QoS values to the Broker (due to out of domain destination). The Broker first figures out the paths satisfying the requested QoS values based on the advertisements coming from all ASes. Then, it updates the border node pair-level view and AS-level view based on the paths satisfying the requested QoS values. Upon having AS-level view, the Broker finds complete e2e paths by stitching those paths satisfying the request from domain for the request. It also calculates prices for these e2e paths according to the pricing scheme proposed in Section 7.5.

7.5 Pricing Framework

It is assumed that $U(x)$ and $C(x)$ are non-decreasing a utility (measure of the value that a customer receives from the service) and cost (for carrying a customer traffic) functions, respectively [257]. Let P_1 denote the price that customers pay for the service and P_2 denote the amount that the service provider receives for the service and $P_2 \leq P_1$. P_1 and P_2 are not necessarily the same due to transactional cost G ($P_1 = P_2$ if $G = 0$). Also, let $Y_1 = U - P_1$ and $Y_2 = P_2 - C$ be surplus of customers

and service provider, respectively. It can be represented that the *net social surplus* as a union of *customer surplus* and *provider surplus* minus the transaction cost. The goal is to increase customer and provider satisfactions by optimal division of the *net social surplus* between the customers and the provider. Furthermore, it is assumed that β, where $0 \leq \beta \leq 1$, and $1 - \beta$ represent the bargaining power of the customers and that of the provider, respectively. Then, it can be defined that the Nash product as $\Omega = Y_1^{\beta} Y_2^{1-\beta}$ in the bargaining process [266–268]. At this point, the goal turns into maximizing the Nash product by finding optimal values for Y_1 and Y_2. This can be represented as a *price optimization* problem and can be formulated as in the Section 7.5.1.

7.5.1 Price Optimization Problem

Let P_{high} be the maximum price that customers would pay for a service where $P_{high} \leq U$, and P_{low} be the minimum price that a service provider would accept for the service where $C \leq P_{low}$. Given U, C, P_{high}, P_{low}, G, β, and $1 - \beta$, the objective is to maximize the Nash production function:

$$\begin{aligned} \underset{Y_1, Y_2}{\text{maximize}} \quad & \Omega = Y_1^{\beta} Y_2^{1-\beta} \\ \text{subject to} \quad & Y_1 + Y_2 \leq U - C - G, \\ & U - P_{high} \leq Y_1, \\ & P_{low} - C \leq Y_2. \end{aligned} \tag{7.1}$$

The objective function Ω is the function of Y_1 and Y_2 and obtains its maximum value while $Y_1 + Y_2 = U - C - G$ where Y_1 and Y_2 are optimal values and represented as ${}^*Y_1$ and ${}^*Y_2$. ${}^*Y_1$ and ${}^*Y_2$ can be found by rewriting the optimization problem in Equation (7.1) as:

$$\max_{Y_1, Y_2} \quad \Omega' = Y_1^{\beta} Y_2^{1-\beta} + \eta(Y_1 + Y_2 - U + C + G) \tag{7.2}$$

where η is the Langrange multiplier. After taking partial derivatives of Ω' with respect to Y_1 and Y_2 and set them to zero along with the fact that $Y_1^* + Y_2^* = U - C - G$, then it is obtained:

$$^*Y_1 = \beta(U - C - G) \tag{7.3}$$

$$^*Y_2 = (1 - \beta)(U - C - G) \tag{7.4}$$

Equation (7.3) implies that net user surplus is directly proportional to the bargaining power of customers (β), while equation (7.4) implies a similar proportion on net provider surplus. Using the definitions of Y_1 and Y_2, the optimal prices $^*P_1$ and $^*P_2$ can be obtained as below:

$$\begin{aligned} ^*Y_1 &= U - {}^*P_1 = \beta(U - C - G) \\ ^*P_1 &= U - \beta(U - C - G) = (1 - \beta)U + \beta(C + G) \end{aligned} \tag{7.5}$$

$$\begin{aligned} ^*Y_2 &= {}^*P_2 - C = (1 - \beta)(U - C - G) \\ ^*P_2 &= C + (1 - \beta)(U - C - G) = (1 - \beta)(U - G) + \beta C \end{aligned} \tag{7.6}$$

These optimal prices, $^*P_1$ and $^*P_2$, represent the optimal amount that customers would pay for the service and the optimal amount that the service provider would receive for the service, respectively.

7.5.2 Cost Function

The optimum price that customers pay for a request/service, P_1^*, in (7.5) is the final price of the service including the cost and profit, $^*P_1 = cost + profit = C + f$. Subtracting the cost, C, from (7.5) would give the profit (f) portion of the optimal price for the service as in below:

$$
\begin{aligned}
{}^{*}P_1 &= C + f \\
f &= {}^{*}P_1 - C = (1-\beta)U + \beta(C+G) - C \\
f &= (1-\beta)(U-C) + \beta G
\end{aligned}
\tag{7.7}
$$

Therefore, it can be obtained;

$$
\begin{aligned}
{}^{*}P_1 &= C + f \\
{}^{*}P_1 &= C + (1-\beta)(U-C) + \beta G
\end{aligned}
\tag{7.8}
$$

This work proposes that absolute cost function for a service request made by a customer should be a function of the QoS parameters (e.g. bandwidth, delay) of the request, time (Δt) that the resources are used, and unit prices for the QoS parameters. These unit prices for QoS parameters are determined by service providers internally. This work does not study this determination but only consider the bandwidth and delay regarding QoS parameters in this chapter.

Let $\mathcal{N} = \{N_i \mid N_i \textit{ is an independent network/provider, } 1 \leq i \leq n\}$ be the set of all networks. This study defines the new cost function $C(b, d, \Delta t)$ as follows:

$$
C(b, d, \Delta t) = \frac{b \times \Delta t \times p_B^i(b)}{p_D^i(d)} \tag{7.9}
$$

where b and d are the bandwidth and delay values respectively in the request, $p_B^i(b)$ is the unit price for bandwidth value in a provider N_i, and $p_D^i(d)$ is the unit price for delay value based on the associated delay segment (class) in the provider N_i. $p_B^i(b)$ and $p_D^i(d)$ might be different for each network since they determine their own unit prices internally. Here, for the sake of simplicity, it is assumed $p_B^i(b)$ and $p_D^i(d)$ are independent of time, that is, they do not change throughout the day. As shown in Fig. 7.1, delay segments represent the delay intervals with certain numbers determined by the providers internally and the determination of those intervals and numbers are out of this chapter's scope.

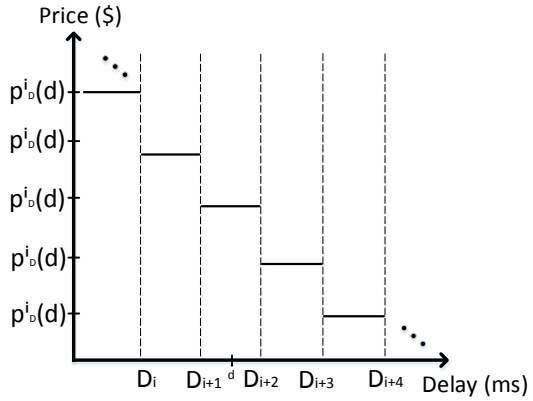


Figure 7.1.: Delay segments vs. unit prices for the segments. Delay parameter (i.e. value) in a request will be in an interval of delay segments. Each delay segments has a corresponding price. The better delay request requires the more price.

Delay segments reflect that the better delay value request will result in an increased cost in return, or vice-versa. This work uses segments/intervals for delay unit pricing since the increased ratio of the delay parameter does not necessarily reflect the same ratio on price. For example, a delay request with 1 ms should not cost two times more than a request with 2 ms.

This cost function captures the impact of requested QoS values from customers by increasing or decreasing the cost incurred by the provider by means of time and unit prices of bandwidth and delay. Obviously, more bandwidth, time, and better delay will affect the cost of the service for the customers accordingly.

7.5.3 Network Connectivity Degree

Service providers invest money to build their connectivity with other providers to improve their QoS or provide more services for their customers. A network with many connections (by means of border nodes with inter-connecting links) to other networks involves more end-to-end (e2e) paths from a source host to a destination host compared to a network with less connections. More centric networks (which have more border nodes connecting to more networks) will profit more due to the increased

number of e2e paths they mediate. This idea leads us to think the connectivity degree of a provider should also be part of the profit that a provider makes for each service it provides to its customers. Therefore, this work proposes these networks should exploit their connectivity in their pricing schemes because they have more chance to be used as a transit network in a path. This exploitation will increase the price they charge for the transit service, thereby affecting its chance of being chosen in a path. Yet if the sender selects another path that does not involve that network, the number of paths will be less, which leads to less path choices for the sender. This choice is a trade-off between price and the probability of having more paths with better QoS.

This study defines the network connectivity degree factor for a network, N_i, as $\mathcal{X}^i = \alpha \times \bar{b}^i$ where α is the control factor and $\bar{b}^i$ is the number of border nodes in N_i. It is assumed that α is the same for all networks, but $\bar{b}^i$ might be different for each N_i. Therefore, each network may have different $\mathcal{X}^i$ for itself.

7.5.4 Final Price for a Service

After integration of the cost function and network connectivity degree factor ($\mathcal{X}^i$) into the optimal price in (7.8), the new final optimal price charged by a provider for a service becomes as follows:

$$^*P_1 = \frac{b \times \Delta t \times p_B^i(b)}{p_D^i(d)} + \Big[(1-\beta)(U - \frac{b \times \Delta t \times p_B^i(b)}{p_D^i(d)}) + \beta G\Big] \times \mathcal{X}^i \qquad (7.10)$$

This new updated optimal price in (7.10) captures both QoS values in a service request, duration of the request and network connectivity degree factor of a provider.

The formula provided in (7.10) reflects the price for a service request by one provider. However, a flow on the Internet mostly passes through many providers until it reaches its final destination. This journey requires many transit services from other providers. Obviously, a source provider may not be able to directly connect to all other networks over an e2e path. Therefore, other providers need to charge the network that forwards the flow. This charging happens until the flow reaches its

destination. As a result of this process, the final price, ${}^*P$, for a service throughout an e2e path for customers can be calculated as follows:

$$\begin{aligned} {}^*P &= \sum_{N_i \in \Psi} {}^*P_1^i \\ &= \sum_{N_i \in \Psi} C^i + \Big[(1-\beta)(U - C^i) + \beta G^i\Big] \times \mathcal{X}^i \\ &= \sum_{N_i \in \Psi} \left[\frac{b \times \Delta t \times p_B^i(b)}{p_D^i(d)} + \Big[(1-\beta)(U^i - \frac{b \times \Delta t \times p_B^i(b)}{p_D^i(d)}) + \beta G^i\Big] \times \mathcal{X}^i\right] \end{aligned} \tag{7.11}$$

where Ψ is the set of networks over the e2e path. The equation in (7.11) implies that the final price for a certain e2e path consists of the sum of the partial prices from networks over the e2e path. As a result, the total charge will be split among the networks over the e2e path. This splitting happens by recursive payments from one provider to another ending with the destination provider (by analogy provider A passes the data to provider B and so on).

7.5.5 Network Revenue and Total Profit

It has been stated that the formula in (7.10) is calculated to find the price that is paid by a customer for a service request in a network. This price includes cost and profit for the service provided. A service provider selling services to other providers receives payments in return of the services provided. On the other hand, it also pays to other service providers that it purchases transit services from. Also, peering with other networks does not bring a huge payment burden to an ISP but it incurs costs to maintain the peering links between peered networks. In addition, this work proposes inclusion of a *penalty* factor for a service request that is not satisfied by the service provider. This factor financially penalizes the service provider for each service that the service provider cannot provide to a customer. This unsatisfied service can happen due to many reasons such as inadequate network resources like bandwidth, delay etc. A penalization for an unsatisfied service happens based on the QoS values,

i.e. bandwidth, delay etc., of the request. Furthermore, every ISP spends money for maintenance of their own local resources to keep providing and improving services to its customers. All of these payments and/or costs form the revenues of an ISP. Therefore, the profit of a network N_i can be expressed as in the equation (7.12).

Let $\mathcal{C}^i$ be the set of customers, $\mathcal{P}^i$ be the set of providers and $\mathcal{R}^i$ be the set of peer networks of a network N_i. Then, the total profit of a service provider would be:

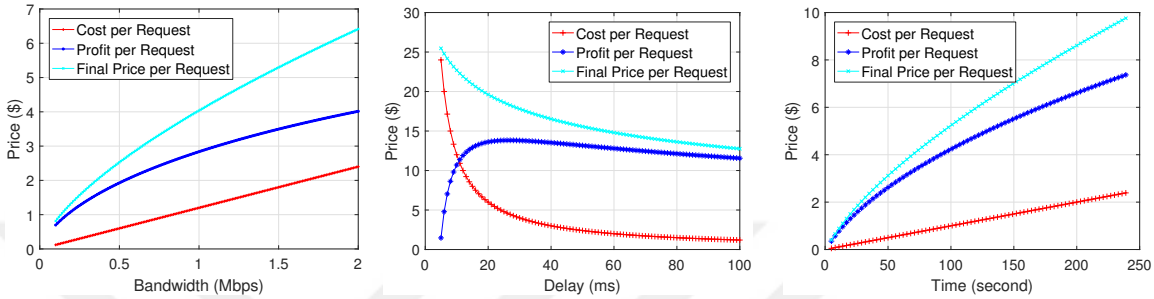
$$F^i = \sum_{c \in \mathcal{C}^i} S(t_c^i) - \sum_{p \in \mathcal{P}^i} S(t_i^p) - \sum_{r \in \mathcal{R}^i} S(c_r^i) - \sum_{c \in \mathcal{C}^i} P(q_i^c) - \mathcal{I}^i \tag{7.12}$$

where F^i is the total profit of the provider N_i, $S(t_c^i)$ is the payments received from customer c based on the total traffic t_c^i exchanged between two networks N_c and N_i, $S(t_i^p)$ is the payments made by network N_i to its upstream provider N_p based on the total traffic t_i^p exchanged between two networks, $S(c_r^i)$ is the cost for maintenance of peering links between networks N_r and N_i, $P(q_i^c)$ is the penalty payments made by N_i to another network N_c due to unsuccessful QoS provisioning, and finally $\mathcal{I}^i$ is the financial investment payments to improve infrastructure/hardware (increased link bandwidth, more switches/routers etc.) of the provider.

7.6 Evaluation

This section has analyzed how cost, profit, price, and number of unsatisfied service requests in a provider (here AS1) are responding to the changes in the bandwidth, delay and time parameters of a request made by a customer. In the experiments, requests have been made from any source host in a provider to any destination host in another provider as an inter-AS traffic with QoS. The experiments have used the utility function as $U(x) = \lambda x^{\gamma} \log \theta x$. This utility function is the same for all providers. The cost function is shown in equation (7.9) and the same for all providers as well. The utility function is an increasing, strictly concave, and continuously differentiable function of QoS parameters. It has also been considered in the context

of elastic traffic [269]. The parameters λ, γ, and θ can be used to control the slope of the utility function. It has also assumed that the transactional cost, G, is zero for simplicity. It has averaged 60 runs for each experiment to achieve and exceed 95% statistical significance.



(a) Bandwidth (Mbps) vs. Price ($) per request relation. Delay = 100 *ms* and Time = 120 *seconds* are constant.

(b) Delay (ms) vs. Price ($) per request relation. Bandwidth = 1.0 *Mbps* and Time = 120 *seconds* are constant.

(c) Time (seconds) vs Price ($) per request relation. Bandwidth = 1.0 *Mbps* and Delay = 100 *ms* are constant.

Figure 7.2.: (a) shows the relation between bandwidth and price change regarding cost, profit, and final price per request while time and delay are constant. (b) compares cost, profit and final price per request based on delay and price change while bandwidth and time are constant. (c) evaluates cost, profit and final price per request by means of time and price while bandwidth and delay are constant.

Fig. 7.2a has analyzed the relation between bandwidth and price regarding impact of the proposed new cost function and network connectivity degree factor over cost, profit and final price per request. In this experiment, it has used $\lambda = 0.5$, $\gamma = 0.5$, and $\theta = 1000$ as it has kept constant as $d = 100$ *ms* and $\Delta t = 120$ *seconds*. It has changed the bandwidth (b) value from $b = 0.1$ *Mbps* to $b = 2.0$ *Mbps*. As seen in Fig. 7.2a, cost per request is increasing by an increase of the bandwidth value. This cost will be reflected to the user as (s)he requests more or less bandwidth for his/her requests.

Fig. 7.2b compares cost, profit and final price per request while delay is increasing and bandwidth and time are constant. It has used $\lambda = 25$, $\gamma = -0.5$, and $\theta = 1$ as it has kept $b = 1.0$ *Mbps* and $\Delta t = 120$ *seconds*. It has varied the delay (d) value

from $d = 5\ ms$ to $d = 100\ ms$. As seen in Fig. 7.2b, cost per request is decreasing by an increase of the delay value since the smaller delay value (better delay) results in the less number of paths satisfying the delay value. On the other hand, the bigger delay values will increase the chance for more paths satisfying the requested delay. The profit per request shows a sharp increase up to around 20 ms and then shows a stable behavior. This depends on the utility of the users because the utility customers receive goes down as the delay increases over time.

Fig. 7.2c evaluates cost, profit and final price per request by means of time and price. In this experiment, it has used $\lambda = 0.1$, $\gamma = 0.5$, and $\theta = 1$ as it has kept $b = 1.0\ Mbps$ and $d = 100\ ms$ and increased the time (Δt) value from $\Delta t = 5\ seconds$ to $\Delta t = 240\ seconds$. As expected, the cost increases as the time increases because the longer QoS requests bring more burden to service providers. The profit also shows an increasing behavior with some slope depending on utility function.

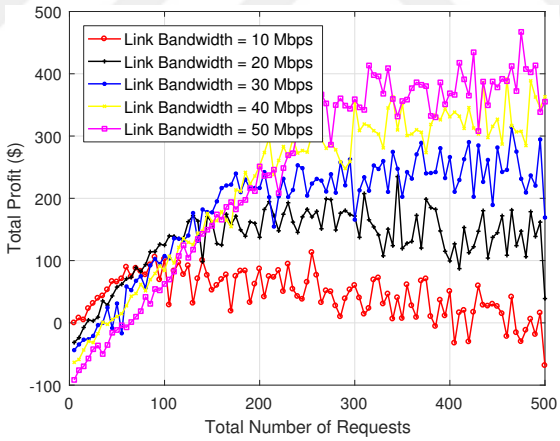


Figure 7.3.: Total Number of Requests vs. Total Profit ($) with respect to investment on link bandwidth capacities and penalties for each unsatisfied service request. Internal link bandwidth capacities are compared with different values, as a result of local internal investment, starting from 10 $Mbps$ to 50 $Mbps$ while number of requests are increasing.

Fig. 7.3 captures the effect of financial investment and penalty on the total profit with respect to number of total requests. The investment can be improvement on network infrastructure like better links with increased link bandwidth or delay ca-

pacity. Service providers can invest money on their networks to improve the QoS for user requests or provide more services for their users. These type of investments help improve user satisfactions and reduce the number of unsatisfied requests. These unsatisfied requests reduce (or keep stable) service provider's total profit since each unsatisfied request brings penalty for the service provider. Therefore, investment on a network and the number of unsatisfied requests maintain an inverse ratio. As shown in Fig. 7.3, the total profit at the beginning is negative since the service provider spends money on links. Obviously, the investment on links with 50 $Mbps$ capacity requires more money compared to that of 10 $Mbps$. In addition, while the total profit with 50 $Mbps$ link capacity is less than 10 $Mbps$ link capacity at the beginning, 50 $Mbps$ link capacity case makes more profit than 10 $Mbps$ case over time. This happens since the number of unsatisfied requests with 50 $Mbps$ links is less than that of 10 $Mbps$ links. In other words, 10 $Mbps$ case brings more penalties (i.e. unsatisfied requests) by the time and reduces total profit. This relation is a trade-off between financial investments and the number of unsatisfied requests. Therefore, service providers should plan their investments carefully regarding the total requests that they receive (or expect) from customers. A careless plan for a service provider can even result in service provider's bankruptcy. For example, a service provider invests more money for links with 50 $Mbps$ bandwidth capacity and therefore starts with a negative total profit as shown in Fig. 7.3. The provider expects that it will make more profit by the time as the number of requests increases since there will be less unsatisfied requests, thereby less penalty. However, if the service provider does not receive the expected number of requests from its customers, assuming just around 50 requests as in Fig. 7.3, it will not be able to make more profit to compensate its investments. Hence, service providers should wisely plan their financial investment strategies.

Fig. 7.4 also reflects the same relation and characteristics as Fig. 7.3 in case of investment on link delay capacity. Links with better delay capacity, e.g 5 ms, require more investment compared to links with worse delay. Although 5 ms link capacity

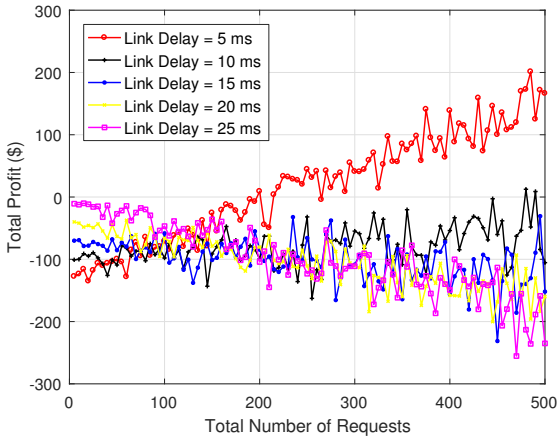


Figure 7.4.: Total Number of Requests vs. Total Profit ($) with respect to investment on link delay capacities and penalties for each unsatisfied service request. Internal link delay capacities are compared with different values starting from 5 ms to 25 ms while number of requests are increasing.

case starts with a less total profit compared to 25 ms link capacity case, it makes more total profit by the time owing to less unsatisfied requests.

7.7 Chapter Summary

The optimal final price for a service request is usually a result of negotiation process between end users (customers) and service provider. On the other hand, charging between service providers (p2p) is similar to end users-service provider (c2p) case since they are also customers of each other. This study has defined the price optimization problem from the perspective of Nash bargaining process and adapted the cost function and network connectivity degree parameter into the final price. It has also given a general scheme of revenue and profit a provider makes with integration of investment and penalty factors. It has finally applied these schemes in the scalable hierarchic architecture. The experiment results have showed that both cost and profit per request reflect the characteristics of the QoS parameters used in the pricing schemes.

8 ECONOMIC ANALYSIS OF SDN UNDER VARIOUS NETWORK FAILURE SCENARIOS

8.1 Abstract

Failures are inevitable in an operational network. They can happen anytime in different sizes and components of a network. They impact the network economics regarding CAPEX, OPEX, revenue lost due to service provisioning cut and so on. In order to mitigate the damages resulting from these failures, reactions of network architectures and designs are crucial for the future of the network. Recently, SDN has got the attention of researchers from both academia and industry as a means in order to increase network availability and reliability due to features, such as centralized automated control and global network view, it promises in networking. This study investigates how programmable network architectures, i.e. SDN technology, affect the network economics compared to traditional network architectures, i.e. MPLS technology, in case of failures. In addition, it explores the economic impact of failures in different SDN control plane models: Centralized (Single) Control Plane (CCP), Distributed (Flat) Control Plane (DCP), and Hierarchical Control Plane (HCP). This work exploits the predefined metric called *Unit Service Cost Scalability* to evaluate economic performances of SDN architecture along with aforementioned control plane models and MPLS architecture under different failure scenarios. It considers two different failure types: i) a random single data plane link failure and ii) a random controller (i.e. control plane) failure. This work also aims at being a useful primer to providing insights regarding how network architectures and control plane models perform with respect to network economics under failures for network owners to plan their investments accordingly.

8.2 Introduction

Availability in networks is one of the crucial attributes for the future of a network. When a failure happens in a network, it is important that service disruption for customers of the network are minimal because customers and the services are sources for the revenue/economics of the network. Therefore, detection of and recovery from a failure as quickly as possible has importance to mitigate the service and performance degradation in networks.

Failures are inevitable in an operational network. They can happen anytime in different sizes and components of a network. Regardless of the network size and the type of services and business, they impact the productivity, network economics regarding CAPEX, OPEX, revenue lost due to service provisioning cut and so on. They can have a major financial impact on service providers. According to [270], an hourly cost of downtime for computer networks is USD 42,000. A company, for example, suffering from an average downtime of 100 hours a year can lose more than $4 million per year. Also, the study in [271] states that cloud networks from 28 cloud providers amass losses estimated at USD 273 million and 1,600 hours of disruptions due to application and infrastructure failures.

Impacts of failures can be different depending on the architecture, programmable e.g. SDN or traditional e.g. MPLS, as well as topology used in networks. In order to mitigate the damages resulting from these failures, reactions of network architectures and designs are crucial for the future of the network. Recently, SDN has got the attention of researchers from both academia and industry as a means in order to increase network availability, reliability, and revenue and as well as decrease network costs, thereby service costs, due to features, such as centralized automated control and global network view, it promises in networking.

This study investigates how programmable network architectures, i.e. SDN technology, affect the network economics compared to traditional network architectures, i.e. MPLS technology, in case of failures. In addition, it explores the economic

impact of failures in different SDN control plane models: Centralized (Single) Control Plane (CCP), Distributed (Flat) Control Plane (DCP), and Hierarchical Control Plane (HCP). This work exploits the predefined metric called *Unit Service Cost Scalability* to evaluate economic performances of SDN architecture along with aforementioned control plane models and MPLS architecture under different failure scenarios. It considers two different failure types: i) a random single data plane link failure and ii) a random controller (i.e. control plane) failure. This work also aims at being a useful primer to providing insights regarding how network architectures and control plane models perform with respect to network economics under failures for network owners to plan their investments accordingly.

In the rest of the chapter, Section 8.3 gives a quick snapshot of the papers that study failure detection and recovery. Section 8.4 explains the method along with experimental details and the metric used in economic analysis of the foregoing network architectures and control plane models in case of certain failure scenarios. While Section 8.5 presents economic performances of both SDN models and MPLS in the data plane link failure scenario, Section 8.6 discusses economic performances of SDN models in the control plane (i.e. controller) failure scenario. The study is summarized with concluding remarks in Section 8.7.

8.3 Related Work

Failure detection is the first step of dealing with a failure in a network. A simple and inefficient approach is to probe each switch on each link in the network utilizing some kind of control messages such as hello messages (e.g. LLDP) [272]. However, this solution suffers from some problems such as imprecise detection of the failed device and scalability issues. Kozat et al. [273] propose a more scalable approach that revolves around the controller computing an Eulerian cycle across all links under its responsibility. Xu et al. [274] utilize Monitoring Flow Entries-based Link Failure Detection (MLFD) instead of LLDP-based failure detection. MLFD mechanism forms

a monitoring tree path consisting of switches, which is probed using Link Monitoring (LM) packets. In [275], the authors propose an OpenFlow-like pipeline design called SPIDER that provides a detection mechanism based on switches' periodic link probing. Bidirectional Forwarding Detection (BFD) [276] with fast failover group type of OpenFlow is another approach exploited in failure detection.

After the detection of a failure, the network has to deal with recovery of paths (i.e. recomputation of new paths), that are broken down, in order to fulfill the flows that are affected by the failure. Failure recovery has two schemes: Protection and restoration. While protection involves a proactive behavior by installing backup paths before a failure occurs, the forwarding decisions are made after a failure happens in the restoration case. In [277], the authors exploit the fast failover group type and BFD to switch between two disjoint paths (working and protected) before a failure occurs in the network. Ramos et al. [278] utilize source routing to compute a secondary path for every path and storing it in the packet header along with the primary path. [279] introduces a framework, CORONET, which is a system for recovery from multiple link failures in data plane.

8.4 Economic Analysis of Network Failures

This work studies possible economic effects of different types of network failures in SDN networks and MPLS. It investigates two different failure scenarios: a random data plane link failure and a random control plane (i.e. controller) failure. These failure types are possible common failures in an SDN or MPLS networks. They also provide network administrators with insights to understand economic impact of failures in different domains in a network.

Carrier-grade networks require sub 50 ms failure recovery time not to cause a significant loss in service connectivity, customers subscriptions, and network revenue. Studies have shown that preserving such a fast recovery time is more possible with protection schemes without involving controllers in online decision-making process

[277]. However, the goal of this chapter's study is not to propose a new standalone failure recovery mechanism. It aims to economically analyze SDN architecture along with some popular control plane models used in SDN and MPLS architecture in case of different types of network failures scenarios. Therefore, this work keeps the following points in mind (for all scenarios where applicable) while conducting this study in order to economically evaluate the foregoing architectures and control plane models:

- It is not interested in or concerned with the speed of a detection/recovery mechanism since the goal is not to introduce a standalone network failure detection and recovery framework, which is out of this work's scope. It exploits a failure detection/recovery mechanism for each failure scenario from the literature and use the same framework for all control plane models to obtain the time of failure detection/recovery while conducting the economic analysis.

- This study concerns about the economic impact of the failures in different control plane models. Therefore, using a framework with lower detection/recovery time affects economic values at the same ratio for all SDN models and MPLS network.

- Also, there is usually (a possibility for) another study introducing lower failure detection/recovery time. Thus, there is no end to find the best framework to name and use in a study similar to this.

In addition, as introduced in the Section 6.8, this work exploits the metrics, *Unit Service Cost Scalability*, to evaluate economic performances of corresponding SDN control plane models and MPLS in the analysis. For more details, readers are referred to the study presented in Chapter 6.

8.4.1 Experimental Setup

As shown in Fig. 5.1, the SDN control plane models considered in this study are CCP, DCP and HCP while conducting analysis to understand their impact over network economics in case of some failure scenarios. These models have their own intrinsic advantages and disadvantages with respect to the various concepts such as control plane scalability, resiliency, better manageability and so on. The data plane and control plane topologies shown in the Fig. 5.1 are just representational and do not reflect the data plane topology used in the study.

Centralized (Single) Control Plane Model (CCP): CCP setting revolves around a single centralized controller with a global network view. The model is simple and it is easy to manage the network.

Distributed (Flat) Control Plane Model (DCP): This model consists of distributed controllers associated with switches. Each controller manages a sub-network/domain of the whole network and has its own local network view, which is, in turn, abstracted as a logical node to its neighboring controllers. These controllers communicate with each other (i.e. connected neighbors) when they receive a packet destined out of its domain in order to set up an end-to-end path.

Hierarchical Control Plane Model (HCP): An HCP design consists of two control plane layers minimum: The lower-layer(s), consisting of local domain controller(s), and the top-layer where another controller, usually called "Root", resides. The domain controllers manage their own domains with full control and are not connected to each other but the Root controller. However, a local controller does not maintain a global view of the whole network. Instead, the Root controller has a full global view of the entire network by abstracting all domains as logical nodes.

This work has used Mininet emulator with POX controller in SDN models. While there is one controller in CCP model, it has divided the whole network into 4 fully-connected sub-networks with a primary controller for each in control planes of DCP and HCP models and provided 16 switches and 24 links in data planes of all SDN

models and MPLS. The data plane topology is the same in all SDN models and MPLS cases. There is also a Root controller on top of local domain controllers in HCP model. Regarding MPLS setting, it has used ns3 network simulator. It needed to use a signaling protocol such as RSVP-TE or CR-LDP to support constraint-based routing in MPLS. Since none of them has been implemented in ns3 at the time of this writing and it is time-consuming and effort-greedy to implement them in ns3, it has generated extra packets between network elements to mimic link state advertisements and state refresh messages for LSPs from aforementioned signaling protocols in MPLS.

In the experiments, it has used 1 Mbps flow sending rate for all service requests. Also, it has used a modified version of Waxman [237] random topology generator defined by Erdos-Renyi random graph model to randomly create the networks while preserving connectivity degrees of nodes (i.e. switches) as three in all models. Furthermore, it has conducted a heuristics, i.e. A*Prune Algorithm [238], to find a feasible path through the network. A*Prune algorithm can be used to solve finding the K shortest paths subject to multiple constraints (KMCSP). In all scenarios, in the first part of the experiments, it has provided enough bandwidth (100 Gbps) in links so that there is no service request rejection due to network resource limitations, while, in the second part of the experiments, it has reduced the link bandwidth to 1 Gbps to see their performances under network resource limitations. In the experiments, the corresponding failure type occurs at the 2nd second in both failure scenarios. Moreover, it has averaged 15 runs for each experiment to achieve and exceed 95% statistical significance in all scenarios. Finally, all experiments were performed on Ubuntu 14.04 in Oracle VirtualBox using an Intel Core i7-5500 system with 12GB RAM.

8.5 Scenario 1: Data Plane (Link) Failure

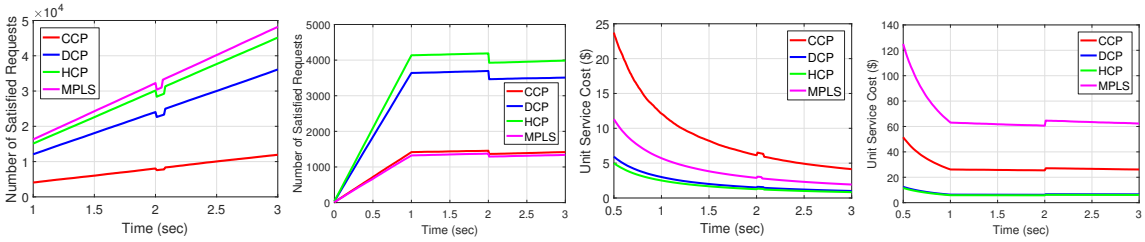
Data plane (link) failure is one of the failure types that we have investigated in our analysis for economic analysis of failures in different SDN control plane models

and MPLS. There exist many proposals for link failure detection/recovery methods in a network. Some of the prevalent methods exploited for failure detection are BFD sessions, Loss of Signal (LOS), and LLDP packets. In this scenario, we only consider a random single link failure.

For the link failure scenarios, we have utilized the methods proposed in [280]. This study implements two well-known mechanisms of failure recovery, i.e. protection and restoration, in OpenFlow networks. In the case of protection, the alternative paths are reserved before the failure occurs in the network. In the case of restoration, alternative paths are not established until a failure occurs. The controller in restoration must notify all the affected switches about a recovery action immediately. For implementation of the protection scheme, the "Group Table" concept specified for OpenFlow in its version 1.1 is used. OpenFlow introduces the fast-failover group type in order to perform fast failover without needing to involve the controller. Any group entry of this type consists of two or more action buckets with a well-defined order. The status of the bucket can be changed by the monitored port going into the "down" state or through other mechanisms such as BFD. In the study, BFD was used to detect the failures. Once BFD declares the failure in the working link, the action bucket associated with this link in the group table is made unavailable by changing the value of the alive status. For the restoration method, once the failure has been detected, the controller is notified about it in order to calculate new paths for the affected flows. The controller takes then the necessary actions such as path recomputation for flows affected by the failure and installation of new flow rules for newly computed paths in the corresponding switches over the new paths.

As to the MPLS case, we have used a similar method to the one explained above for the SDN models. In the protection case, we have utilized a link protection approach by pre-programming next-hop port values into the router FIB awaiting activation, which happens in milliseconds following the failure detection. In the restoration case, on the other hand, the head end of each path for each flow on the failed links recomputes a new path following the failure detection. Regarding the failure detection

and recovery activation, BFD sessions were established between routers for each link in the networks.

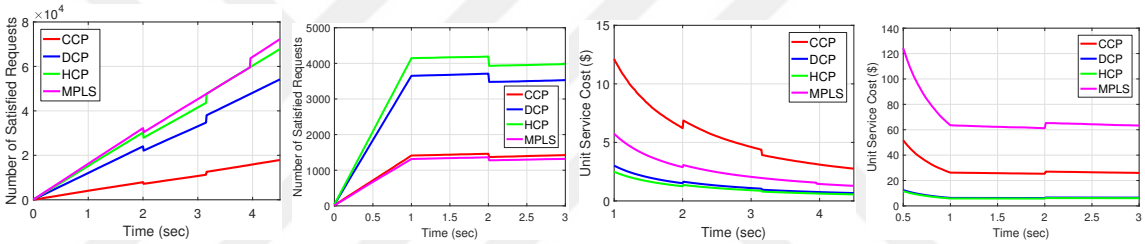


(a) Total number of satisfied requests vs. Time - 100 Gbps link bandwidth - Protection case

(b) Total number of satisfied requests vs. Time - 1 Gbps link bandwidth - Protection case

(c) Unit service cost vs. Time - 100 Gbps link bandwidth - Protection case

(d) Unit service cost vs. Time - 1 Gbps link bandwidth - Protection case



(e) Total number of satisfied requests vs. Time - 100 Gbps link bandwidth - Restoration case

(f) Total number of satisfied requests vs. Time - 1 Gbps link bandwidth - Restoration case

(g) Unit service cost vs. Time - 100 Gbps link bandwidth - Restoration case

(h) Unit service cost vs. Time - 1 Gbps link bandwidth - Restoration case

Figure 8.1.: Total number of satisfied requests and respective unit service cost performances of networks in case of data plane (link) failure with use of protection and restoration schemes under 100 Gbps and 1 Gbps link bandwidth cases.

Fig. 8.1 shows the total number of satisfied requests and unit service cost performances in case of a random single data plane link failure with use of protection (Fig. 8.1a, 8.1b, 8.1c, 8.1d) and restoration (Fig. 8.1e, 8.1f, 8.1g, 8.1h) schemes under 100 Gbps and 1 Gbps link bandwidth cases. We have used the protection and restoration schemes explained earlier for link failure detection and recovery. In 100 Gbps link bandwidth case, the total number of satisfied requests in the network increases in both protection (Fig. 8.1a) and restoration (Fig. 8.1e) cases for all SDN control plane

models and MPLS until the failure happens because there are enough bandwidth to use in the links. Once the failure happens, the number of satisfied flows in the network shows a reduction because the flows served over the failed link are not satisfied anymore. The reduction is the lowest in CCP and highest in HCP (465 in CCP, 1392 in DCP, 1750 in HCP, and 1862 in MPLS) among all SDN models and MPLS in both protection and restoration cases owing to the total number of satisfied flows in the networks. However, the total number of satisfied requests for all models continue to increase immediately after the reduction because there are enough bandwidth in the links for the upcoming requests. On the other hand, in 1 Gbps link bandwidth case, the number of satisfied requests in the network increase until links become loaded in both protection (Fig. 8.1b) and restoration (Fig. 8.1f) cases for all SDN control plane models and MPLS. After links become loaded, it shows a steady-like behavior until the failure. Once the failure happens, it shows the lowest reduction in CCP while the highest is in HCP (91 in CCP, 229 in DCP, 261 in HCP, 84 in MPLS) as in the previous link bandwidth case in both protection and restoration schemes. The total number of satisfied requests for all models stay steady, unlike the 100 Gbps link bandwidth case, during the recovery phase because new flows cannot be satisfied in the network since the links are loaded in both SDN models and MPLS. However, the recovery phase cannot be completed in 1 Gbps link bandwidth case because the flows over the failed link cannot be rerouted from failed link to the other links due to the fact that all links are loaded/full and cannot handle more flows. Therefore, the number of flows do not increase again after the failure unlike the 100 Gbps link bandwidth case.

Regarding the unit service cost, in 100 Gbps link bandwidth case, the results reveal that it decreases as the number of satisfied flows increase until the failure occurs for all SDN models and MPLS in both protection (Fig. 8.1c) and restoration (Fig. 8.1g) cases. While the unit service cost increase ratio is the highest in CCP ($\sim$6.1% in CCP, $\sim$3.1% in DCP, $\sim$1.9% in HCP, and $\sim$4.6% in MPLS), HCP shows the lowest increase ratio among all models in protection and restoration schemes. After the

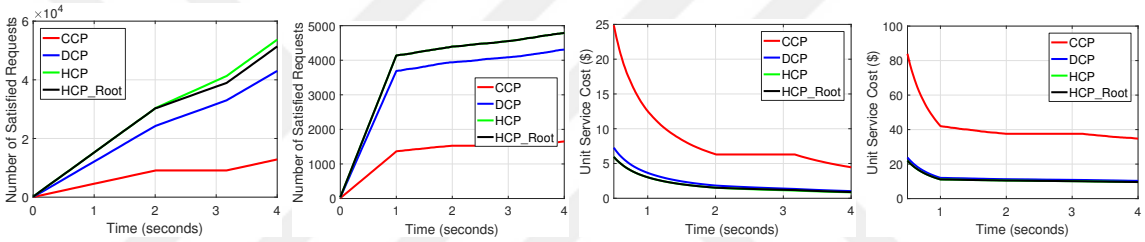
recovery is complete, the unit service cost starts showing a reduction again in both protection and restoration schemes. In 1 Gbps case, the unit service cost decreases as the total number of satisfied requests in the network increase until the links become loaded in both protection (Fig. 8.1d) and restoration (Fig. 8.1h) cases. Then, it reflects a steady-like behavior until the failure happens. The link failure results in a sudden increase (∼5.7% in CCP, ∼2.2% in DCP, ∼1.6% in HCP, and ∼6.5% in MPLS) in protection and restoration schemes in the unit service cost as in the 100 Gbps link bandwidth case. However, the unit service cost do not show a reduction again in either of protection and restorations schemes unlike the 100 Gbps link bandwidth case since the recovery cannot be completed due to the loaded links.

8.6 Scenario 2: Control Plane (Controller) Failure

As SDN brings many advantages to networking, it also suffers from various problems. One of the serious problems of SDN is that the controller may be a critical point of failure, which can result in an overall network unavailability. Therefore, the design of a fault-tolerant control plane is a must for a SDN-based network. There might be varying number of reasons for failure of a controller: hardware failure (e.g. controller server hardware), software failure/bug in the server operating system and/or controller software, power outages and so on. One basic solution for a controller failure is use of redundant controller(s) (i.e. backup/standby controller) in order to automatically take critical responsibilities over network infrastructure control and data flows management from the failed primary controller in case of a controller failure. While this procedure is called controller failover, the reverse of the procedure (i.e. restoring the primary controller) is called controller failback. As of OpenFlow protocol version 1.2.0, it provides the possibility to configure one or more backup controllers which can assume the network control in case of failure using controller role change mechanism, but OpenFlow does not provide any coordination mechanism between the primary and the backup controllers. Therefore, it is network administrators' responsibility to

provide such a synchronization method for consistency among them to handle controller failure and add resiliency without happening any fatal damage to network services and customer satisfactions.

In the control plane failure case, it has utilized the method described in [109]. In this scenario, every network domain is managed by a single controller, and another controller is used as backup for every domain controller that can take over its role in case the primary fails. In case a failure occurs, and to ensure a smooth transition to a new primary, in this particular instance of a fault-tolerant architecture, the controller store the network and application related state in a shared data store.



(a) Total number of satisfied requests vs. Time - 100 Gbps link bandwidth

(b) Total number of satisfied requests vs. Time - 1 Gbps link bandwidth

(c) Unit service cost vs. Time - 100 Gbps link bandwidth

(d) Unit service cost vs. Time - 1 Gbps link bandwidth

Figure 8.2.: Total number of satisfied requests and respective unit service cost performances of networks in case of controller failure under 100 Gbps and 1 Gbps link bandwidth cases.

Fig. 8.2 shows the total number of satisfied requests and respective unit service cost performances of networks in case of a controller failure in SDN models (CCP, DCP, and HCP) under 100 Gbps and 1 Gbps link bandwidth cases. It also evaluates Root controller failure case in HCP model. In 100 Gbps link bandwidth case (Fig. 8.2a), the total number of satisfied requests in the networks show an increase in DCP, HCP, HCP_Root controller failure cases all the time. However, because there is only one controller in CCP model, the total number of satisfied requests in the network stay the same once a controller failure happens. The total number of satisfied requests in the network start increasing after the backup controller takes over the network

management responsibilities. On the other hand, a primary controller failure results in just some reduction in increase ratio of the total number of satisfied requests in the network in DCP and HCP models. In addition, this reduction is more since all inter-domain connection requests are affected in case of HCP_Root controller failure. In 1 Gbps link bandwidth case (Fig. 8.2b), the total number satisfied requests in the network increase until the links become loaded and then shows a steady-like behavior in all control plane models. In CCP model, it does not increase in controller failure case since there is only one domain controlled by one controller although it shows very little increase in DCP and HCP models because some requests may find enough remaining bandwidth in the links depending on source and destination pairs. Also, HCP and HCP_Root failure cases do not make any difference because the total number of satisfied requests in the network are the same for both failure cases. It is the same because the network becomes loaded before the controllers reach their requests handling capacities.

Regarding the unit service cost, it is shown that it decreases as the number of satisfied requests are increasing in both 100 Gbps (Fig. 8.2c) and 1 Gbps (Fig. 8.2d) link bandwidth cases. In Fig. 8.2c, the unit service cost stays steady in CCP model when the controller failure occurs because the satisfied requests number in the network do not increase while the unit service cost shows a continuous decreasing behavior in DCP and HCP models in that phase. In Fig. 8.2d, the unit service cost decreases fast until the links become loaded and then shows a slow reduction in all models. In both 100 and 1 Gbps link bandwidth cases, HCP_Root controller failure does not make a noticeable difference compared to a controller failure in HCP with respect to the unit service cost.

8.7 Chapter Summary

This work has investigated how programmable network architectures, i.e. SDN technology, affect the network economics compared to traditional network architec-

tures, i.e. MPLS technology, in case of failures. In addition, it has explored the economic impact of failures in different SDN control plane models: CCP, DCP, and HCP. It has considered two different failure types: i) a random single data plane link failure and ii) a random controller (i.e. control plane) failure. The experiments have revealed that the unit service cost shows an increase in case of data link failure in all SDN models and MPLS. While CCP model shows the highest increase among all SDN models and MPLS, MPLS shows the least increase ratio among all. Also, this increase ratio is more for both SDN models and MPLS in use of restoration scheme compared to use of protection scheme. In control plane (i.e. controller) failure scenario, the unit service cost stays steady in CCP model once the controller failure occurs because the satisfied requests number in the network do not increase while the unit service cost shows a continuous decreasing behavior in DCP and HCP models in that phase in case of 100 Gbps link bandwidth case. In addition, the unit service cost decreases fast until the links become loaded and then shows a slow reduction in all models in case of 1 Gbps link bandwidth case. Moreover, in both 100 and 1 Gbps link bandwidth cases, HCP_Root controller failure does not make a noticeable difference compared to a controller failure in HCP with respect to the unit service cost.

9 CONCLUSIONS

This thesis has first surveyed and summarized the state-of-the-art studies in terms of the characterizations and taxonomy of two research areas in Software Defined Networking (SDN): (i) scalability-related problems and (ii) Quality of Service (QoS)-related problems. It has also outlined the potential challenges and open problems that need to be addressed further for more scalable SDN control planes and better and complete QoS abilities in SDN networks. It then has proposed a hierarchy-based network architecture along with an inter-AS routing approach with QoS considering scalability and routing privacy. Later, a metric has been proposed in order to evaluate the control plane scalability in SDN along with mathematical models of the proposed metric over different control plane designs. After that, this thesis has defined two metrics *Unit Service Cost Scalability* and *Cost-to-Service* to evaluate how SDN architecture performs compared to MPLS architecture. Also, mathematical models have been presented to calculate certain cost parts of a network. In addition, a comparison of different popular SDN control plane models, Centralized Control Plane (CCP), Distributed Control Plane (DCP), and Hierarchical Control Plane (HCP), have been given to understand the economic impact of them with regards to the defined metrics. Furthermore, the thesis has proposed an optimal pricing scheme for a service request with QoS in SDN environment using the Nash bargaining problem, which aims to maximize benefits of both service providers and customers. The scheme integrates a new cost function and network connectivity degree factor into the proposed pricing scheme. Finally, the thesis has investigated how programmable network architectures, i.e. SDN, affect the network economics compared to traditional network architectures, i.e. MPLS, under certain failure scenarios: (i) a random single data plane link failure and (ii) a random controller (i.e. control plane) failure.

REFERENCES

REFERENCES

[1] S. Sezer, S. Scott-Hayward, P.K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao. Are we ready for sdn? implementation challenges for software-defined networks. *Communications Magazine, IEEE*, 51(7):36–43, 2013.

[2] Software-Defined Networking: The New Norm for Networks. Technical report, Open Networking Foundation (ONF), April 2012.

[3] K. Kirkpatrick. Software-defined networking. *Commun. ACM*, 56(9):16–19, September 2013.

[4] F. de Oliveira Silva, J.H. de Souza Pereira, P.F. Rosa, and S.T. Kofuji. Enabling future internet architecture research and experimentation by using software defined networking. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 73–78, 2012.

[5] G. Goth. Software-defined networking could shake up more than packets. *Internet Computing, IEEE*, 15(4):6–9, 2011.

[6] W. Xia, Y. Wen, C.H. Foh, D. Niyato, and H. Xie. A survey on software-defined networking. *Communications Surveys Tutorials, IEEE*, PP(99):1–1, 2014.

[7] K. Bakshi. Considerations for software defined networking (sdn): Approaches and use cases. In *Aerospace Conference, 2013 IEEE*, pages 1–9, 2013.

[8] 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. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 3–14, 2013.

[9] R. White and J. Tantsura. *Navigating Network Complexity: Next-generation routing with SDN, service virtualization, and service chaining*. Addison-Wesley Professional; 1st edition, 2015.

[10] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A survey of active network research. *Comm. Mag.*, 35(1):80–86, January 1997.

[11] Click Router Project. https://github.com/kohler/click. (Date Last Accessed: 2018-10-17).

[12] XORP Platform. http://www.xorp.org/. (Date Last Accessed: 2018-10-17).

[13] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, pages 131–145, New York, NY, USA, 2001. ACM.

[14] JP. Vasseur and JL. Le Roux. RFC 5440 : Path Computation Element (PCE) Communication Protocol (PCEP), 2009.

[15] Business Cases for Brocade Software-Defined Networking Use Cases, White Paper, Brocade.

[16] Operator Network Monetization Through Openflow-Enabled SDN, White Paper, 2013, Open Networking Foundation (ONF).

[17] A. Manzalini, R. Saracco, C. Buyukkoc, P. Chemouil, S. Kuklinski, A. Gladisch, M. Fukui, E. Dekel, D. Soldani, M. Ulema, et al. Software-defined networks for future networks and services. In *White Paper based on the IEEE Workshop SDN4FNS*, 2013.

[18] The Top Five Reasons to Deploy Software-Defined Networks and Network Functions Virtualization, White Paper, 2014, ZK Research.

[19] Operationalizing SDN and NFV Networks, White Paper, 2015, Deloitte Consulting LLP.

[20] N. Zhang and H. Hmminen. Cost efficiency of sdn in lte-based mobile networks: Case finland. In *2015 International Conference and Workshops on Networked Systems (NetSys)*, pages 1–5, March 2015.

[21] C. Bouras, P. Ntarzanos, and A. Papazois. Cost modeling for sdn/nfv based mobile 5g networks. In *2016 8th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 56–61, Oct 2016.

[22] N. C. Nguyen, P. Wang, D. Niyato, Y. Wen, and Z. Han. Resource management in cloud networking using economic analysis and pricing models: A survey. *IEEE Communications Surveys Tutorials*, PP(99):1–1, 2017.

[23] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. Nox: Towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, July 2008.

[24] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of data center traffic: Measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '09, pages 202–208, New York, NY, USA, 2009. ACM.

[25] D. Erickson. The beacon openflow controller. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 13–18, 2013.

[26] Y. Luo, P. Cascon, E. Murray, and J. Ortega. Accelerating openflow switching with network processors. In *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '09, pages 70–71, New York, NY, USA, 2009. ACM.

[27] V. Tanyingyong, M. Hidell, and P. Sjödin. Using hardware classification to improve pc-based openflow switching. In *2011 IEEE 12th International Conference on High Performance Switching and Routing*, pages 215–221. IEEE, 2011.

[28] R. Narayanan, S. Kotha, G. Lin, A. Khan, S. Rizvi, W. Javed, H. Khan, and S. A. Khayam. Macroflows and microflows: Enabling rapid network innovation through a split sdn data plane. In *2012 European Workshop on Software Defined Networking*, pages 79–84, Oct 2012.

[29] G. Lu, R. Miao, Y. Xiong, and C. Guo. Using cpu as a traffic co-processing unit in commodity switches. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 31–36, New York, NY, USA, 2012. ACM.

[30] K. Kannan and S. Banerjee. *Compact TCAM: Flow Entry Compaction in TCAM for Power Aware SDN*, pages 439–444. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[31] N. Kang, Z. Liu, J. Rexford, and D. Walker. Optimizing the "one big switch" abstraction in software-defined networks. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 13–24, New York, NY, USA, 2013. ACM.

[32] N. Katta, O. Alipourfard, J. Rexford, and D. Walker. Infinite cacheflow in software-defined networks. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 175–180, New York, NY, USA, 2014. ACM.

[33] Q. Zuo, M. Chen, K. Ding, and B. Xu. On generality of the data plane and scalability of the control plane in software-defined networking. *China Communications*, 11(2):55–64, Feb 2014.

[34] 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.*, 38(2):69–74, March 2008.

[35] F. Hu, Q. Hao, and K. Bao. A survey on software defined networking (sdn) and openflow: From concept to implementation. *Communications Surveys Tutorials, IEEE*, PP(99):1–1, 2014.

[36] S. J. Vaughan-Nichols. Openflow: The next generation of the network? *IEEE Computer*, (8):13–15.

[37] OpenFlow Switch Specification (1.5.1). Open Networking Foundation, March 2015.

[38] SDN architecture. Technical report, Open Networking Foundation (ONF), June 2014.

[39] P. Lin, J. Bi, S. Wolff, Y. Wang, A. Xu, Z. Chen, H. Hu, and Y. Lin. A west-east bridge based sdn inter-domain testbed. *Communications Magazine, IEEE*, 53(2):190–197, Feb 2015.

[40] M.P. Fernandez. Comparing OpenFlow Controller Paradigms Scalability: Reactive and Proactive. In *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, pages 1009–1016, March 2013.

[41] OpenFlow Switch Specification (1.1.0). Open Networking Foundation, February 2011.

[42] OpenFlow Switch Specification (1.2.0). Open Networking Foundation, December 2011.

[43] B. J. van Asten, N. L. M. van Adrichem, and F. A. Kuipers. Scalability and resilience of software-defined networking: An overview. *CoRR*, abs/1408.6760, 2014.

[44] Kai Hwang and Zhiwei Xu. *Scalable Parallel Computing: Technology, Architecture, Programming*. McGraw-Hill, Inc., New York, NY, USA, 1998.

[45] X. H. Sun and D. T. Rover. Scalability of parallel algorithm-machine combinations. *IEEE Trans. Parallel Distrib. Syst.*, 5(6):599–613, June 1994.

[46] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1994.

[47] A. Y. Grama, A. Gupta, and V. Kumar. Isoefficiency: Measuring the scalability of parallel algorithms and architectures. *IEEE concurrency*, (3):12–21, 1993.

[48] L. Pastor and J.L. Bosquwe Orero. An efficiency and scalability model for heterogeneous clusters. In *Cluster Computing, 2001. Proceedings. 2001 IEEE International Conference on*, pages 427–434, Oct 2001.

[49] X. H. Sun, Yong Chen, and Ming Wu. Scalability of heterogeneous computing. In *Proceedings of the 2005 International Conference on Parallel Processing*, ICPP '05, pages 557–564, Washington, DC, USA, 2005. IEEE Computer Society.

[50] P. Jogalekar and M. Woodside. Evaluating the scalability of distributed systems. *IEEE Trans. Parallel Distrib. Syst.*, 11(6), June 2000.

[51] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood. On controller performance in software-defined networks. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Hot-ICE'12, pages 10–10, 2012.

[52] S. A. Shah, J. Faiz, M. Farooq, A. Shafi, and S. A. Mehdi. An architectural evaluation of sdn controllers. In *2013 IEEE International Conference on Communications (ICC)*, pages 3504–3508, June 2013.

[53] F. Benamrane, R. Benaini, et al. Performances of openflow-based software-defined networks: an overview. *Journal of Networks*, 10(6):329–337, 2015.

[54] K. He, J. Khalid, S. Das, A. Gember-Jacobson, C. Prakash, A. Akella, L. E. Li, and M. Thottan. Latency in software defined networks: Measurements and mitigation techniques. In *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '15, pages 435–436, New York, NY, USA, 2015. ACM.

[55] P. Isaia and L. Guan. Performance benchmarking of sdn experimental platforms. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 116–120, June 2016.

[56] J. Hu, C. Lin, X. Li, and J. Huang. Scalability of control planes for software defined networks: Modeling and evaluation. In *Quality of Service (IWQoS), 2014 IEEE 22nd International Symposium of*, pages 147–152, May 2014.

[57] M. Karakus and A. Durresi. A Scalability Metric for Control Planes in Software Defined Networks (SDNs). In *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pages 282–289, March 2016.

[58] S.H. Yeganeh, A. Tootoonchian, and Y. Ganjali. On scalability of software-defined networking. *Communications Magazine, IEEE*, 51(2):136–141, 2013.

[59] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky. Advanced study of sdn/openflow controllers. In *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, CEE-SECR '13, pages 1:1–1:6, New York, NY, USA, 2013. ACM.

[60] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 267–280, New York, NY, USA, 2010. ACM.

[61] B. Heller, R. Sherwood, and N. McKeown. The controller placement problem. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 7–12, 2012.

[62] S. Azodolmolky, P. Wieder, and R. Yahyapour. Performance evaluation of a scalable software-defined networking deployment. In *Software Defined Networks (EWSDN), 2013 Second European Workshop on*, pages 68–74, Oct 2013.

[63] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, and P. Francois. A declarative and expressive approach to control forwarding paths in carrier-grade networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 15–28, New York, NY, USA, 2015. ACM.

[64] A. R. Curtis, J. C. M., J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: scaling flow management for high-performance networks. *SIGCOMM Comput. Commun. Rev.*, 41(4):254–265, August 2011.

[65] M. Yu, J. Rexford, M. J. Freedman, and J. Wang. Scalable flow-based networking with difane. *SIGCOMM Comput. Commun. Rev.*, 41(4):–, August 2010.

[66] K. Phemius, M. Bouet, and J. Leguay. Disco: Distributed multi-domain sdn controllers. *CoRR*.

[67] M.A.S. Santos, B.A.A. Nunes, K. Obraczka, T. Turletti, B.T. de Oliveira, and C.B. Margi. Decentralizing sdn's control plane. In *Local Computer Networks (LCN), 2014 IEEE 39th Conference on*, pages 402–405, Sept 2014.

[68] A. A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella. Elasticon: an elastic distributed sdn controller. In *Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems*, pages 17–28. ACM, 2014.

[69] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: taking control of the enterprise. *SIGCOMM Comput. Commun. Rev.*, 37(4):1–12, August 2007.

[70] S. Vissicchio, O. Tilmans, L. Vanbever, and J. Rexford. Central Control Over Distributed Routing. *SIGCOMM Comput. Commun. Rev.*, 45(4):43–56, August 2015.

[71] D. Marconett and S. J. B. Yoo. Flowbroker: A software-defined network controller architecture for multi-domain brokering and reputation. *Journal of Network and Systems Management*, 23(2):328–359, 2015.

[72] A. Tootoonchian and Y. Ganjali. Hyperflow: A distributed control plane for openflow. In *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, INM/WREN'10, pages 3–3, 2010.

[73] S. H. Yeganeh and Y. Ganjali. Kandoo: a framework for efficient and scalable offloading of control applications. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 19–24, 2012.

[74] J. Mccauley, A. Panda, M. Casado, T. Koponen, and S. Shenker. Extending sdn to large-scale networks. In *ONS*, 2013.

[75] Z. Cai, A. L. Cox, and T. S. E. Ng. Maestro: A System for Scalable OpenFlow Control. Technical report, Rice University, 2010.

[76] A. Voellmy and J. Wang. Scalable software defined network controllers. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pages 289–290, New York, NY, USA, 2012. ACM.

[77] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: a distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, 2010.

[78] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar. Onos: Towards an open, distributed sdn os. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 1–6, New York, NY, USA, 2014. ACM.

[79] Y. Fu, J. Bi, K. Gao, Z. Chen, J. Wu, and B. Hao. Orion: A hybrid hierarchical control plane of software-defined networking for large-scale networks. In *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pages 569–576, Oct 2014.

[80] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker. Applying nox to the datacenter. In *Proc. of workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.

[81] A. S. W. Tam, X. Kang, and H. J. Chao. Use of devolved controllers in data center networks. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pages 596–601, April 2011.

[82] V. Yazici, O. M. Sunay, and A. O. Ercan. Controlling a software-defined network via distributed controllers. In *2012 NEM Summit Conference Proceedings*, NEM Summit'12, 2012.

[83] M. F. Bari, A. R. Roy, S. R. Chowdhury, Z. Qi, M. F. Zhani, R. Ahmed, and R. Boutaba. Dynamic controller provisioning in software defined networks. In *Network and Service Management (CNSM), 2013 9th International Conference on*, pages 18–25, Oct 2013.

[84] M. Karakus and A. Durresi. A Scalable Inter-AS QoS Routing Architecture in Software Defined Network (SDN). In *Advanced Information Networking and Applications (AINA), 2015 IEEE 29th International Conference on*, pages 148–154, March 2015.

[85] H. Owens and A. Durresi. Explicit routing in software-defined networking (ersdn): Addressing controller scalability. In *Network-Based Information Systems (NBiS), 2014 17th International Conference on*, Sept 2014.

[86] M. Soliman, B. Nandy, I. Lambadaris, and P. Ashwood-Smith. Source routed forwarding with software defined control, considerations and implications. In *Proceedings of the 2012 ACM Conference on CoNEXT Student Workshop*, CoNEXT Student '12, pages 43–44, 2012.

[87] M. Canini, P. Kuznetsov, D. Levin, and S. Schmid. Software transactional networking: Concurrent and consistent policy composition. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 1–6, New York, NY, USA, 2013. ACM.

[88] OpenDaylight Project. https://www.opendaylight.org/. (Date Last Accessed: 2018-10-17).

[89] L. J. Cowen. Compact routing with minimum stretch. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '99, pages 255–260, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.

[90] S. Vissicchio, L. Vanbever, and J. Rexford. Sweet Little Lies: Fake Topologies for Flexible Routing. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, HotNets-XIII, pages 3:1–3:7, New York, NY, USA, 2014. ACM.

[91] Haskell, A Functional Programming Language. https://www.haskell.org/. (Date Last Accessed: 2018-10-17).

[92] The Glasgow Haskell Compiler. https://www.haskell.org/ghc/. (Date Last Accessed: 2018-10-17).

[93] Cbench: An OpenFlow Controller Benchmarker. https://github.com/mininet/oflops/tree/master/cbench. (Date Last Accessed: 2018-10-17).

[94] S. Gao, S. Shimizu, S. Okamoto, and N. Yamanaka. A high-speed routing engine for software defined network. *Cyber Journals: Multidisciplinary Journals in Science and Technology, Journal of Selected Areas in Telecommunications (JSAT)*, JSAT August Edition(1):1–7, 2012.

[95] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, and M. F. Magalhães. Quagflow: Partnering quagga with openflow. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 441–442, New York, NY, USA, 2010. ACM.

[96] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, C. N. A. Corrêa, S. C. de Lucena, and M. F. Magalhães. Virtual routers as a service: The routeflow approach leveraging software-defined networks. In *Proceedings of the 6th International Conference on Future Internet Technologies*, CFI '11, pages 34–37, New York, NY, USA, 2011. ACM.

[97] GNU Quagga Project. http://www.nongnu.org/quagga/. (Date Last Accessed: 2018-10-17).

[98] M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries. A Flexible OpenFlow-Controller Benchmark. In *2012 European Workshop on Software Defined Networking*, pages 48–53, Oct 2012.

[99] Planning and Designing Networks with the Cisco MATE Portfolio, White Paper, 2013, Cisco.

[100] M. Handley, O. Hodson, and E. Kohler. XORP: An Open Platform for Network Research. *SIGCOMM Comput. Commun. Rev.*, 33(1):53–57, January 2003.

[101] Floodlight. http://www.projectfloodlight.org/floodlight/. (Date Last Accessed: 2018-10-17).

[102] Mininet. http://mininet.org/. (Date Last Accessed: 2018-10-17).

[103] Openvswitch. http://openvswitch.org/. (Date Last Accessed: 2018-10-17).

[104] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. Measuring isp topologies with rocketfuel. *IEEE/ACM Trans. Netw.*, 12(1):2–16, February 2004.

[105] Internet2. http://www.internet2.edu/. (Date Last Accessed: 2018-10-17).

[106] Pox. https://github.com/noxrepo/pox. (Date Last Accessed: 2018-10-17).

[107] ns-3 Network Simulator. https://www.nsnam.org/. (Date Last Accessed: 2018-10-17).

[108] P. Fonseca, R. Bennesby, E. Mota, and A. Passito. A replication component for resilient openflow-based networking. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 933–939, April 2012.

[109] F. Botelho, A. Bessani, F. M. V. Ramos, and P. Ferreira. On the design of practical fault-tolerant sdn controllers. In *Software Defined Networks (EWSDN), 2014 Third European Workshop on*, pages 73–78, Sept 2014.

[110] S. S. Savas, M. Tornatore, M. F. Habib, P. Chowdhury, and B. Mukherjee. Disaster-Resilient Control Plane Design and Mapping in Software-Defined Networks. *ArXiv e-prints*, September 2015.

[111] V. Pashkov, A. Shalimov, and R. Smeliansky. Controller failover for sdn enterprise networks. In *Science and Technology Conference (Modern Networking Technologies) (MoNeTeC), 2014 International*, pages 1–6, Oct 2014.

[112] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann. Logically centralized?: State distribution trade-offs in software defined networks. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 1–6, New York, NY, USA, 2012. ACM.

[113] H. Yin, H. Xie, T. Tsou, D. R. Lopez, P. A. Aranda, and R. Sidi. SDNi: A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains. Internet-Draft draft-yin-sdn-sdni-00, Internet Engineering Task Force, December 2012. Work in Progress.

[114] P. Lin, J. Bi, and Y. Wang. Webridge: west-east bridge for distributed heterogeneous sdn noses peering. *Security and Communication Networks*, 8(10):1926–1942, 2015.

[115] L. Pingping, B. Jun, C. Ze, W. Yangyang, H. Hongyu, and X. Anmin. Webridge: West-east bridge for sdn inter-domain network peering. In *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, pages 111–112, April 2014.

[116] J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A.ndrew R. Curtis, and S. Banerjee. Devoflow: Cost-effective flow management for high performance enterprise networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 1:1–1:6, New York, NY, USA, 2010. ACM.

[117] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng. On the placement of controllers in software-defined networks. *The Journal of China Universities of Posts and Telecommunications*, 19, Supplement 2:92 – 171, 2012.

[118] Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan. Reliability-aware controller placement for software-defined networks. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 672–675, May 2013.

[119] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng. On reliability-optimized controller placement for software-defined networks. *Communications, China*, 11(2):38–54, Feb 2014.

[120] H. K. Rath, V. Revoori, S. M. Nadaf, and A. Simha. Optimal controller placement in software defined networks (sdn) using a non-zero-sum game. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, pages 1–6, June 2014.

[121] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia. Pareto-optimal resilient controller placement in sdn-based core networks. In *Teletraffic Congress (ITC), 2013 25th International*, pages 1–9, Sept 2013.

[122] P. Xiao, W. Qu, H. Qi, Z. Li, and Y. Xu. The sdn controller placement problem for wan. In *Communications in China (ICCC), 2014 IEEE/CIC International Conference on*, pages 220–224, Oct 2014.

[123] Y. Jimenez, C. Cervello-Pastor, and A. J. Garcia. On the controller placement for designing a distributed sdn control layer. In *Networking Conference, 2014 IFIP*, pages 1–9, June 2014.

[124] M. Obadia, M. Bouet, J. L. Rougier, and L. Iannone. A Greedy Approach for Minimizing SDN Control Overhead. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pages 1–5, April 2015.

[125] Y. Liu, Y. Li, Y. Wang, and J. Yuan. Optimal scheduling for multi-flow update in software-defined networks. *Journal of Network and Computer Applications*, 54:11 – 19, 2015.

[126] S. Yang, S. Ho, Y. Lin, and C. Gan. A multi-rat bandwidth aggregation mechanism with software-defined networking. *Journal of Network and Computer Applications*, pages –, 2015.

[127] R. Braden, D. Clark, and S. Shenker. RFC 1633 : Integrated Services in the Internet Architecture: an Overview, 1994.

[128] L. Zhang, S. Berson, S. Herzog, and S. Jamin. RFC 2205 : Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification, 1997.

[129] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC 2475 : An Architecture for Differentiated Service, 1998.

[130] E. Rosen, A. Viswanathan, and R. Callon. RFC 3031 : Multiprotocol Label Switching Architecture, 2001.

[131] B. S. Davie and A. Farrel. *MPLS: Next Steps*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[132] J. W. Evans and C. Filsfils. *Deploying IP and MPLS QoS for Multiservice Networks: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[133] OpenFlow Switch Specification (1.0.0). Open Networking Foundation, December 2009.

[134] OpenFlow Management and Configuration Protocol 1.2 (OF-Config 1.2). Technical report, Open Networking Foundation (ONF), 2014.

[135] OpenFlow Switch Specification (1.3.0). Open Networking Foundation, June 2012.

[136] OpenFlow Switch Specification (1.4.0). Open Networking Foundation, October 2013.

[137] OpenFlow Switch Specification (1.5.0). Open Networking Foundation, December 2014.

[138] B. Pfaff and B. Davie. RFC 7047: The Open vSwitch Database Management Protocol, 2013.

[139] ONOS Project. http://onosproject.org/. (Date Last Accessed: 2018-10-17).

[140] R. Wallner and R. Cannistra. An sdn approach: Quality of service using big switchs floodlight open-source controller. In *Proceedings of the Asia-Pacific Advanced Network*, volume 35, pages 14–19, 2013.

[141] D. Palma, J. Gonçalves, B. Sousa, L. Cordeiro, P. Simoes, S. Sharma, and D. Staessens. The queuepusher: Enabling queue management in openflow. In *Proceedings of the 2014 Third European Workshop on Software Defined Networks*, EWSDN '14, pages 125–126, Washington, DC, USA, 2014. IEEE Computer Society.

[142] S. Civanlar, M. Parlakisik, A. M. Tekalp, B. Gorkemli, B. Kaytaz, and E. Onem. A qos-enabled openflow environment for scalable video streaming. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 351–356, Dec 2010.

[143] Y. Jinyao, Z. Hailong, S. Qianjun, L. Bo, and G. Xiao. Hiqos: An sdn-based multipath qos solution. *Communications, China*, 12(5):123–133, May 2015.

[144] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp. Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks. In *Signal Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific*, pages 1–8, Dec 2012.

[145] H. Owens and A. Durresi. Video over software-defined networking (vsdn). In *Network-Based Information Systems (NBiS), 2013 16th International Conference on*, pages 44–51, Sept 2013.

[146] H. Owens, A. Durresi, and R. Jain. Reliable video over software-defined networking (rvsdn). In *2014 IEEE Global Communications Conference*, pages 1974–1979, Dec 2014.

[147] S. Tomovic, N. Prasad, and I. Radusinovic. Sdn control framework for qos provisioning. In *Telecommunications Forum Telfor (TELFOR), 2014 22nd*, pages 111–114, Nov 2014.

[148] H. E. Egilmez, B. Gorkemli, A. M. Tekalp, and S. Civanlar. Scalable video streaming over openflow networks: An optimization framework for qos routing. In *Image Processing (ICIP), 2011 18th IEEE International Conference on*, pages 2241–2244, Sept 2011.

[149] H. E. Egilmez, S. Civanlar, and A. M. Tekalp. An optimization framework for qos-enabled adaptive video streaming over openflow networks. *Multimedia, IEEE Transactions on*, 15(3):710–715, April 2013.

[150] Y. Tsung-Feng, K. Wang, and H. Yi-Huai. Adaptive routing for video streaming with qos support over sdn networks. In *2015 International Conference on Information Networking (ICOIN)*, pages 318–323, Jan 2015.

[151] S. Yilmaz, A. M. Tekalp, and B. D. Unluturk. Video streaming over software defined networks with server load balancing. In *Computing, Networking and Communications (ICNC), 2015 International Conference on*, pages 722–726, Feb 2015.

[152] H. E. Egilmez, S. Civanlar, and A. M. Tekalp. A distributed qos routing architecture for scalable video streaming over multi-domain openflow networks. In *Image Processing (ICIP), 2012 19th IEEE International Conference on*, pages 2237–2240, Sept 2012.

[153] H. E. Egilmez and A. M. Tekalp. Distributed qos architectures for multimedia streaming over software defined networks. *Multimedia, IEEE Transactions on*, 16(6):1597–1609, Oct 2014.

[154] D. Marconett and S. J. B. Yoo. Flowbroker: Market-driven multi-domain sdn with heterogeneous brokers. In *Optical Fiber Communication Conference*, pages Th2A–36. Optical Society of America, 2015.

[155] J. M. Wang, Y. Wang, X. Dai, and B. Benasou. Sdn-based multi-class qos guarantee in inter-data center communications. *Cloud Computing, IEEE Transactions on*, PP(99):1–1, 2015.

[156] J. M. Wang, Y. Wang, X. Dai, and B. Benasou. Sdn-based multi-class qos-guaranteed inter-data center traffic management. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pages 401–406, Oct 2014.

[157] M. Wang, F. Agraz, P. Shuping, S. Spadaro, G. Bernini, J. Perello, G. Zervas, R. Nejabati, N. Ciulli, D. Simeonidou, H. Dorren, and N. Calabretta. Sdn-enabled ops with qos guarantee for reconfigurable virtual data center networks. *Optical Communications and Networking, IEEE/OSA Journal of*, 7(7):634–643, July 2015.

[158] V. Kotronis, R. Klöti, M. Rost, P. Georgopoulos, B. Ager, S. Schmid, and X. Dimitropoulos. Stitching inter-domain paths over ixps. In *Proceedings of the Symposium on SDN Research*, SOSR '16, pages 17:1–17:12, New York, NY, USA, 2016. ACM.

[159] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S. Lee, and P. Yalagandula. Automated and scalable qos control for network convergence. In *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, INM/WREN'10, pages 1–1, 2010.

[160] I. Bueno, J. I. Aznar, E. Escalona, J. Ferrer, and J. Antoni Garcia-Espin. An opennaas based sdn framework for dynamic qos control. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pages 1–7, Nov 2013.

[161] Q. Duan. Network-as-a-service in software-defined networks for end-to-end qos provisioning. In *Wireless and Optical Communication Conference (WOCC), 2014 23rd*, pages 1–5, May 2014.

[162] Q. Duan, C. Wang, and X. Li. End-to-end service delivery with qos guarantee in software defined networks. *arXiv preprint arXiv:1504.04076*, Apr 2015.

[163] M Said Seddiki, Muhammad Shahbaz, Sean Donovan, Sarthak Grover, Miseon Park, Nick Feamster, and Ye-Qiong Song. Flowqos: Per-flow quality of service for broadband access networks. Georgia Institute of Technology, 2015.

[164] M. S. Seddiki, M. Shahbaz, S. Donovan, S. Grover, M. Park, N. Feamster, and Y. Song. Flowqos: Qos for the rest of us. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 207–208, New York, NY, USA, 2014. ACM.

[165] M. Afaq, S. U. Rehman, and W. Song. A framework for classification and visualization of elephant flows in sdn-based networks. *Procedia Computer Science*, 65:672 – 681, 2015.

[166] M. Afaq, S. U. Rehman, and W. Song. Visualization of elephant flows and qos provisioning in sdn-based networks. In *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific*, pages 444–447, Aug 2015.

[167] A. Ishimori, F. Farias, E. Cerqueira, and A. Abelem. Control of multiple packet schedulers for improving qos on openflow/sdn networking. In *Software Defined Networks (EWSDN), 2013 Second European Workshop on*, pages 81–86, Oct 2013.

[168] K. Nam-Seok, H. Hwanjo, and P. Jong-Dae, P.and Hong-Shik. Openqflow: Scalable openflow with flow-based qos. *IEICE transactions on communications*, 96(2):479–488, 2013.

[169] C. Xu, B. Chen, and H. Qian. Quality of service guaranteed resource management dynamically in software defined network. In *Journal of Communications*, volume 10, pages 843–850, 2015.

[170] W. Wang, Y. Tian, X. Gong, Q. Qi, and Y. Hu. Software defined autonomic qos model for future internet. *Journal of Systems and Software*, 110:122 – 135, 2015.

[171] C. Caba and J. Soler. Apis for qos configuration in software defined networks. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pages 1–5, April 2015.

[172] T. Huong-Truong, N. H. Thanh, N. T. Hung, J. Mueller, and T. Magedanz. Qoe-aware resource provisioning and adaptation in ims-based iptv using openflow. In *Local Metropolitan Area Networks (LANMAN), 2013 19th IEEE Workshop on*, pages 1–3, April 2013.

[173] H. Kumar, H. H. Gharakheili, and V. Sivaraman. User control of quality of experience in home networks using sdn. In *Advanced Networks and Telecommuncations Systems (ANTS), 2013 IEEE International Conference on*, pages 1–6, Dec 2013.

[174] Y. Yiakoumis, S. Katti, T. Huang, N. McKeown, K. Yap, and R. Johari. Putting home users in charge of their network. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 1114–1119, New York, NY, USA, 2012. ACM.

[175] A. Kassler, L. Skorin-Kapov, O. Dobrijevic, M. Matijasevic, and P. Dely. Towards qoe-driven multimedia service negotiation and path optimization with software defined networking. In *Software, Telecommunications and Computer Networks (SoftCOM), 2012 20th International Conference on*, pages 1–5, Sept 2012.

[176] O. Dobrijevic, A. J. Kassler, L. Skorin-Kapov, and M. Matijasevic. Q-point: Qoe-driven path optimization model for multimedia services. In *Wired/Wireless Internet Communications*, volume 8458, pages 134–147. Springer International Publishing, 2014.

[177] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race. Towards network-wide qoe fairness using openflow-assisted adaptive video streaming. In *Proceedings of the 2013 ACM SIGCOMM workshop on Future human-centric multimedia networking*, FhMN '13, pages 15–20, 2013.

[178] S. Gorlatch, T. Humernbrum, and F. Glinka. Improving qos in real-time internet applications: from best-effort to software-defined networks. In *Computing, Networking and Communications (ICNC), 2014 International Conference on*, pages 189–193, Feb 2014.

[179] S. Gorlatch and T. Humernbrum. Enabling high-level qos metrics for interactive online applications using sdn. In *Computing, Networking and Communications (ICNC), 2015 International Conference on*, pages 707–711, Feb 2015.

[180] M. Jarschel, F. Wamser, T. Hohn, T. Zinner, and P. Tran-Gia. Sdn-based application-aware networking on the example of youtube video streaming. In *Software Defined Networks (EWSDN), 2013 Second European Workshop on*, pages 87–92, Oct 2013.

[181] I. Ayadi, G. Diaz, and N. Simoni. Qos-based network virtualization to future networks: An approach based on network constraints. In *Network of the Future (NOF), 2013 Fourth International Conference on the*, pages 1–5, Oct 2013.

[182] K. Govindarajan, K. C. Meng, H. Ong, W. M. Tat, S. Sivanand, and L. S. Leong. Realizing the quality of service (qos) in software-defined networking (sdn) based cloud infrastructure. In *Information and Communication Technology (ICoICT), 2014 2nd International Conference on*, pages 505–510, May 2014.

[183] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba. Policycop: An autonomic qos policy enforcement framework for software defined networks. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pages 1–7, Nov 2013.

[184] M. Broadbent, D. King, S. Baildon, N. Georgalas, and N. Race. Opencache: A software-defined content caching platform. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pages 1–5, April 2015.

[185] M. Broadbent and N. Race. Opencache: exploring efficient and transparent content delivery mechanisms for video-on-demand. In *Proceedings of the 2012 ACM conference on CoNEXT student workshop*, CoNEXT Student '12, pages 15–16, 2012.

[186] B. Sonkoly, A. Gulyas, F. Nemeth, J. Czentye, K. Kurucz, B. Novak, and G. Vaszkun. On qos support to ofelia and openflow. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 109–113, Oct 2012.

[187] C. Hu, Q. Wang, and X. Dai. Sdn over ip: Enabling internet to provide better qos guarantee. In *Frontier of Computer Science and Technology (FCST), 2015 Ninth International Conference on*, pages 46–51, Aug 2015.

[188] A. Desai and K. S. Nagegowda. Advanced control distributed processing architecture (acdpa) using sdn and hadoop for identifying the flow characteristics and setting the quality of service(qos) in the network. In *Advance Computing Conference (IACC), 2015 IEEE International*, pages 784–788, June 2015.

[189] S. Tomovic, I. Radusinovic, and N. Prasad. Performance comparison of qos routing algorithms applicable to large-scale sdn networks. In *EUROCON 2015 - International Conference on Computer as a Tool (EUROCON), IEEE*, pages 1–6, Sept 2015.

[190] H. Zhang, X. Guo, J. Yan, B. Liu, and Q. Shuai. Sdn-based ecmp algorithm for data center networks. In *Computing, Communications and IT Applications Conference (ComComAp), 2014 IEEE*, pages 13–18, Oct 2014.

[191] A. R. Roy, F. Bari, M. F. Zhani, R. Ahmed, and R. Boutaba. Dot: Distributed openflow testbed. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 367–368, New York, NY, USA, 2014. ACM.

[192] M. Fiedler, K. Kilkki, and P. Reichl. 09192 executive summary – from quality of service to quality of experience. In Markus Fiedler, Kalevi Kilkki, and Peter Reichl, editors, *From Quality of Service to Quality of Experience*, number 09192 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2009. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[193] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers. Opennetmon: Network monitoring in openflow software-defined networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–8, May 2014.

[194] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba. Payless: A low cost network monitoring framework for software defined networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9, May 2014.

[195] P. H. Isolani, J. A. Wickboldt, C. B. Both, J. Rochol, and L. Z. Granville. Interactive monitoring, visualization, and configuration of openflow-based sdn. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 207–215, May 2015.

[196] L. Jose, M. Yu, and J. Rexford. Online measurement of large traffic aggregates on commodity switches. In *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Hot-ICE'11, pages 13–13, Berkeley, CA, USA, 2011. USENIX Association.

[197] Y. Minlan, J. Lavanya, and M. Rui. Software defined traffic measurement with opensketch. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 29–42, Lombard, IL, 2013. USENIX.

[198] A. Tootoonchian, M. Ghobadi, and Y. Ganjali. Opentm: Traffic matrix estimator for openflow networks. In *Proceedings of the 11th International Conference on Passive and Active Measurement*, PAM'10, pages 201–210, Berlin, Heidelberg, 2010. Springer-Verlag.

[199] J. R. Ballard, I. Rae, and A. Akella. Extensible and scalable network monitoring using opensafe. In *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, INM/WREN'10, pages 8–8, Berkeley, CA, USA, 2010. USENIX Association.

[200] S. E. Middleton and S. Modafferi. Experiences monitoring and managing qos using sdn on testbeds supporting different innovation stages. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pages 1–5, April 2015.

[201] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett. SDX: A Software Defined Internet Exchange. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 551–562, 2014.

[202] N. Feamster, J. Rexford, S. Shenker, R. Clark, R. Hutchins, D. Levin, and J. Bailey. SDX: A Software Defined Internet Exchange. *Open Networking Summit*, page 1, 2013.

[203] Y. Rekhter and T. Li. RFC 1771: A Border Gateway Protocol 4 (BGP-4). IETF, 1995.

[204] S. J. B. Yoo. Multi-domain cognitive optical software defined networks with market-driven brokers. In *Optical Communication (ECOC), 2014 European Conference on*, pages 1–3, Sept 2014.

[205] E. Kissel, G. Fernandes, M. Jaffee, M. Swany, and M. Zhang. Driving software defined networks with xsp. In *Communications (ICC), 2012 IEEE International Conference on*, pages 6616–6621, 2012.

[206] M. Moshref, M. Yu, and R. Govindan. Resource/accuracy tradeoffs in software-defined measurement. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 73–78, New York, NY, USA, 2013. ACM.

[207] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou. A Roadmap for Traffic Engineering in SDN-OpenFlow Networks. *Computer Networks*, 71:1 – 30, 2014.

[208] H. Farhady, H. Lee, and A. Nakao. Software-Defined Networking: A survey. *Computer Networks*, 81:79 – 95, 2015.

[209] S. Verbrugge, D. Colle, M. Pickavet, P. Demeester, S. Pasqualini, A. Iselt, A. Kirstädter, R. Hülsermann, F. J. Westphal, and M. Jäger. Methodology and input availability parameters for calculating opexand capex costs for realistic network scenarios. *J. Opt. Netw.*, 5(6):509–520, Jun 2006.

[210] S. Verbrugge, S. Pasqualini, F. J. Westphal, M. Jager, A. Iselt, A. Kirstadter, R. Chahine, D. Colle, M. Pickavet, and P. Demeester. Modeling Operational Expenditures for Telecom Operators. In *Conference on Optical Network Design and Modeling, 2005.*, pages 455–466, Feb 2005.

[211] Swisscom AG, Tele Danmark A/S, Telefónica S.A., and Telenor AS. Extended Investment Analysis of Telecommunication Operator Strategies. volume 2, 1999.

[212] S. Verbrugge, D. Colle, M. Jäger, R. Huelsermann, F. Westphal, M. Pickavet, and P. Demeester. Impact of resilience strategies on capital and operational expenditures. In *Proceedings of ITG-Fachtagung Photonical Networks 2005*, pages 109–116, 2005.

[213] S. Pasqualini, S. Verbrugge, A. Kirstädter, A. Iselt, D. Colle, M. Pickavet, and P. Demeester. Influence of a control plane on network expenditures. In *Asia-Pacific Optical Communications*, pages 60220I–60220I. International Society for Optics and Photonics, 2005.

[214] S. Pasqualini, A. Kirstadter, A. Iselt, R. Chahine, S. Verbrugge, D. Colle, M. Pickavet, and P. Demeester. Influence of gmpls on network providers' operational expenditures: a quantitative study. *IEEE Communications Magazine*, 43(7):28–38, July 2005.

[215] B. Naudts, M. Kind, F. J. Westphal, S. Verbrugge, D. Colle, and M. Pickavet. Techno-economic Analysis of Software Defined Networking as Architecture for the Virtualization of a Mobile Network. In *2012 European Workshop on Software Defined Networking*, 2012.

[216] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Flowvisor: A Network Virtualization Layer. In *OpenFlow Switch Consortium, Tech. Rep*, volume 1, page 132, 2009.

[217] E. Hernandez-Valencia, S. Izzo, and B. Polonsky. How will NFV/SDN Transform Service Provider Opex? *IEEE Network*, 29(3):60–67, May 2015.

[218] T. M. Knoll. A Combined CAPEX and OPEX Cost Model for LTE Networks. In *Telecommunications Network Strategy and Planning Symposium (Networks), 2014 16th International*, pages 1–6, Sept 2014.

[219] T. M. Knoll. Life-cycle cost modelling for nfv/sdn based mobile networks. In *2015 Conference of Telecommunication, Media and Internet Techno-Economics (CTTE)*, pages 1–8, Nov 2015.

[220] M. Karakus and A. Durresi. Service Cost in Software Defined Networking (SDN). In *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pages 468–475, March 2017.

[221] E. J. Kwak, G. e. Kim, and J. H. Yoo. Network operation cost model to achieve efficient operation and improving cost competitiveness. In *ICACT, 2011*, pages 1107–1112, Feb 2011.

[222] R. G. Addie, Y. Peng, M. Albdair, C. Xing, D. Fatseas, and M. Zukerman. Cost modelling and validation in network optimization. In *ITNAC, 2015 International*, pages 11–16, Nov 2015.

[223] L. Rodrguez de Lope, K. Hackbarth, A. E. Garca, T. Plueckebaum, and D. Ilic. Cost models for next generation networks with quality of service parameters. In *Networks 2008 - The 13th International Telecommunications Network Strategy and Planning Symposium*, volume Supplement, pages 1–9, Sept 2008.

[224] L. Popa, S. Ratnasamy, G. Iannaccone, A. Krishnamurthy, and I. Stoica. A Cost Comparison of Datacenter Network Architectures. In *Proceedings of the 6th International Conference*, Co-NEXT '10, pages 16:1–16:12, New York, NY, USA, 2010. ACM.

[225] K. Casier, S. Verbrugge, R. Meersman, J. V. Ooteghem, D. Colle, M. Pickavet, and P. Demeester. A fair cost allocation scheme for capex and opex for a network service provider. *Proceedings of CTTE2006, the 5th Conference on Telecommunication Techno-Economics*, 2006.

[226] R. Bailey. Elements of techno-economic modelling planning, provisioning and operation of virtualised networks. In *Telecommunications Network Strategy and Planning Symposium (Networks), 2016 17th International*, pages 25–30. IEEE, 2016.

[227] Open Networking Foundation (ONF). https://www.opennetworking.org. (Date Last Accessed: 2018-10-17).

[228] M. Karakus and A. Durresi. A Survey: Control Plane Scalability Issues and Approaches in Software-Defined Networking (SDN). *Computer Networks*, 112:279 – 293, 2017.

[229] A. S. da Silva, P. Smith, A. Mauthe, and A. Schaeffer-Filho. Resilience Support in Software-Defined Networking: A Survey. *Computer Networks*, 92, Part 1:189 – 207, 2015.

[230] M. Walker. A Growth Opportunity for Vendors: Telco Opex, OVUM, Oct 31 2012.

[231] H. G. Perros. *Connection-Oriented Networks: SONET/SDH, ATM, MPLS and Optical Networks*. Wiley, 2005.

[232] J. Kharel and D. Adhikari. Performance evaluation of voice traffic over mpls network with te and qos implementation. Master's thesis, Blekinge Institute of Technology, Sweden, November 2011.

[233] B. Jamoussi, L. Andersson, R. Callon, R. Dantu, L. Wu, P. Doolan, T. Worster, N. Feldman, A. Fredette, M. Girish, et al. RFC 3212: Constraint-based LSP setup using LDP, 2002.

[234] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. RFC 3209: RSVP-TE: extensions to RSVP for LSP tunnels, 2001.

[235] M. Behringer, A. Retana, R. White, and G. Huston. RFC 7980: A Framework for Defining Network Complexity, 2016.

[236] P. P. Jogalekar and C. M. Woodside. A scalability metric for distributed computing applications in telecommunications. *Teletraffic Science and Engineering*, 2:101–110, 1997.

[237] B. M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, Dec 1988.

[238] Gang L. and K. G. Ramakrishnan. A*prune: an algorithm for finding k shortest paths subject to multiple constraints. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, volume 2, pages 743–749 vol.2, 2001.

[239] O. Younis and S. Fahmy. Constraint-based routing in the internet: Basic principles and recent research. *Commun. Surveys Tuts.*, 5(1):2–13, July 2003.

[240] J. Lahteenmaki, H. Hammainen, N. Zhang, and M. Swan. Cost modeling of a network service provider cloud platform. In *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*, pages 148–153, 2016.

[241] Data Center SDN Strategies Global Service Provider Survey. Technical report, IHS Technology, Infonetics, October 2015.

[242] Data Center SDN Strategies North American Enterprise Survey. Technical report, IHS Technology, Infonetics, February 2015.

[243] NEC SDN Customer Cases. https://www.nec.com/en/case/index.html?q=sdn. (Date Last Accessed: 2018-10-17).

[244] H. Hlavacs, G. Da Costa, and J. M. Pierson. Energy consumption of residential and professional switches. In *2009 International Conference on Computational Science and Engineering*, volume 1, pages 240–246, Aug 2009.

[245] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan. A power benchmarking framework for network devices. In *Proceedings of the 8th International IFIP-TC 6 Networking Conference*, NETWORKING '09, pages 795–808, Berlin, Heidelberg, 2009. Springer-Verlag.

[246] K. Xie, X. Huang, S. Hao, M. Ma, P. Zhang, and D. Hu. e^3mc: Improving energy efficiency via elastic multi-controller sdn in data center networks. *IEEE Access*, 4:6780–6791, 2016.

[247] D. B. Rawat and S. R. Reddy. Software defined networking architecture, security and energy efficiency: A survey. *IEEE Communications Surveys Tutorials*, 19(1):325–346, Firstquarter 2017.

[248] Cost of KWh Electricity Power. https://www.eia.gov/electricity/?t=epmt_5_6_a. (Date Last Accessed: 2018-10-17).

[249] F. Kaup, S. Melnikowitsch, and D. Hausheer. Measuring and modeling the power consumption of openflow switches. In *Network and Service Management (CNSM), 2014 10th International Conference on*, pages 181–186. IEEE, 2014.

[250] L. Cominardi, C. J. Bernardos, P. Serrano, A. Banchs, and A. Oliva. Experimental evaluation of sdn-based service provisioning in mobile networks. *Comput. Stand. Interfaces*, 58(C):158–166, May 2018.

[251] D. J. Stern. Software Defined Networking Everything: Evolving to the DoD Information Core, 2016, Defense Information Systems Agency (DISA).

[252] Provisioning New Service Paths in Minutes with SDN https://www.packetdesign.com/blog/provisioning-new-service-paths-in-minutes-with-sdn/. (Date Last Accessed: 2018-10-17).

[253] S. Fernandes, E. Tavares, M. Santos, V. Lira, and P. Maciel. Dependability assessment of virtualized networks. In *2012 IEEE International Conference on Communications (ICC)*, pages 2711–2716, June 2012.

[254] S. Mahankali and S. Rungta. Adopting Software-Defined Networking in the Enterprise, White Paper, April 2014, Intel IT.

[255] J. R. Martin. What is a learning curve? Management And Accounting Web. http://maaw.info/LearningCurveSummary.htm. (Date Last Accessed: 2018-10-17).

[256] L. Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Transactions on Networking (ToN)*, 9(6):733–745, 2001.

[257] G. N. Rouskas. *Internet Tiered Services: Theory, Economics, and Quality of Service*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[258] V. Oberle, H. Ritter, and K. Wehrle. Bpp: A protocol for exchanging pricing information between autonomous systems, 2001.

[259] S. Shakkottai and R. Srikant. Economics of network pricing with multiple isps. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 1, pages 184–194 vol. 1, March 2005.

[260] S. Shakkottai and R. Srikant. Economics of network pricing with multiple isps. *IEEE/ACM Trans. Netw.*, 14(6):1233–1245, December 2006.

[261] Q. Lv and G. N. Rouskas. an economic model for pricing tiered network services. *Communications*, 201:09, 2009.

[262] V. Valancius, C. Lumezanu, N. Feamster, R. Johari, and V. V. Vazirani. How many tiers?: pricing in the internet transit market. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 194–205. ACM, 2011.

[263] L. Jiang, S. Parekh, and J. Walrand. Time-dependent network pricing and bandwidth trading. In *Network Operations and Management Symposium Workshops, 2008. NOMS Workshops 2008. IEEE*, pages 193–200. IEEE, 2008.

[264] G. Kesidis, A. Das, and G. de Veciana. On flat-rate and usage-based pricing for tiered commodity internet services. In *Information Sciences and Systems, 2008. CISS 2008. 42nd Annual Conference on*, pages 304–308. IEEE, 2008.

[265] J. Altmann and D. Goel. Economizing isp interconnections at internet exchange points. *Journal of Integrated Design and Process Science*, 10(2):21–34, 2006.

[266] J. Nash. Two-person cooperative games. *Econometrica: Journal of the Econometric Society*, pages 128–140, 1953.

[267] J. F. Nash et al. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.

[268] J. F. Nash. The bargaining problem. *Econometrica: Journal of the Econometric Society*, pages 155–162, 1950.

[269] S. Shenker. Fundamental design issues for the future internet. *Selected Areas in Communications, IEEE Journal on*, 13(7):1176–1188, 1995.

[270] Q&A: How Much Does an Hour of Downtime Cost? Technical report, Gartner, September 20009.

[271] C. Cérin, C. Coti, P. Delort, F. Diaz, M. Gagnaire, Q. Gaumer, N. Guillaume, J. Lous, S. Lubiarz, J. Raffaelli, et al. Downtime statistics of current cloud solutions. *International Working Group on Cloud Computing Resiliency, Tech. Rep*, 2013.

[272] Ieee standard for local and metropolitan area networks - station and media access control connectivity discovery. *IEEE Std 802.1AB-2016 (Revision of IEEE Std 802.1AB-2009)*, pages 1–146, March 2016.

[273] U. C. Kozat, G. Liang, and K. Kkten. On diagnosis of forwarding plane via static forwarding rules in software defined networks. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 1716–1724, April 2014.

[274] H. Xu, L. Yan, H. Xing, Y. Cui, and S. Li. Link failure detection in software defined networks: an active feedback mechanism. *Electronics Letters*, 53(11):722–724, 2017.

[275] C. Cascone, D. Sanvito, L. Pollini, A. Capone, and B. Sansò. Fast failure detection and recovery in sdn with stateful data plane. *International Journal of Network Management*, 27(2), 2017.

[276] D. Katz and D. Ward. RFC 5880: Bidirectional Forwarding Detection (BFD), 2010.

[277] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester. A demonstration of fast failure recovery in software defined networking. In *International Conference on Testbeds and Research Infrastructures*, pages 411–414. Springer, 2012.

[278] R. M. Ramos, M. Martinello, and C. E. Rothenberg. Data center fault-tolerant routing and forwarding: An approach based on encoded paths. In *2013 Sixth Latin-American Symposium on Dependable Computing*, pages 104–113, April 2013.

[279] H. Kim, M. Schlansker, J. R. Santos, J. Tourrilhes, Y. Turner, and N. Feamster. Coronet: Fault tolerance for software defined networks. In *2012 20th IEEE International Conference on Network Protocols (ICNP)*, pages 1–2, Oct 2012.

[280] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester. Openflow: Meeting carrier-grade recovery requirements. *Computer Communications*, 36(6):656 – 665, 2013.

VITA

VITA

Murat Karakus

Indiana University - Purdue University Indianapolis (IUPUI)

Department of Computer and Information Science

Email: muratkarakus60@gmail.com

Education

Ph.D. in Computer Science *August 2013 - December 2018*

Department of Computer and Information Science

Indiana U. - Purdue U. Indianapolis (IUPUI), Indianapolis, IN, USA

M.Sc. in Computer Science, GPA 8.00/9.00 *April 2013*

Department of Computer Science, Engineering and Physics

University of Michigan-Flint, Flint, MI, USA

B.S. in Mathematics, GPA 3.34/4.00 *June 2009*

Department of Mathematics

Suleyman Demirel University, Isparta, Turkey

Research Interests

New network architectures (e.g. Software-Defined Networking (SDN)); Scalability; Service Pricing; Quality of Service (QoS); Routing; Localization and tracking in Wireless Sensor Networks (WSN); Communications and state estimation automation in Smart Grid; introducing programming to non-CS majors.

Refereed Publications

[1] **M. Karakus** and A. Durresi, "Economic Viability of Software Defined Networking (SDN)", Computer Networks, vol. 135, pp. 81 - 95, 2018.

[2] **M. Karakus** and A. Durresi, "Economic Impact Analysis of Control Plane Architectures in Software Defined Networking (SDN)", in IEEE ICC 2018 Communications Software, Services, and Multimedia Applications Symposium, May 2018.

[3] **M. Karakus** and A. Durresi, "Service Cost in Software Defined Networking (SDN)", in 2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA), pp. 468 - 475, March 2017.

[4] **M. Karakus** and A. Durresi, "Software-Defined Networking (SDN) Value Proposition over Different Network Architectures", in 1st International Balkan Conference on Communications and Networking (BalkanCom), pp. 175 - 182, May 2017.

[5] **M. Karakus** and A. Durresi, "Quality of Service (QoS) in Software Defined Networking (SDN): A Survey", Journal of Network and Computer Applications, vol. 80, pp. 200 - 218, 2017.

[6] **M. Karakus** and A. Durresi, "A Survey: Control Plane Scalability Issues and Approaches in Software-Defined Networking (SDN)", Computer Networks, vol. 112, pp. 279 - 293, 2017.

[7] **M. Karakus** and A. Durresi, "Economic Viability of QoS in Software Defined Networks (SDNs)", in 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA), pp. 140 - 146, March 2016.

[8] **M. Karakus** and A. Durresi, "A Scalability Metric for Control Planes in Software Defined Networks (SDNs)", in 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA), pp. 282 - 289, March 2016.

[9] **M. Karakus** and A. Durresi, "A Scalable Inter-AS QoS Routing Architecture in Software Defined Network (SDN)", in Advanced Information Networking and Applications (AINA), 2015 IEEE 29th International Conference on, pp. 148 - 154, March 2015.

[10] S. Uludag, **M. Karakus**, and E. Guler, "Low-Complexity 3D Target Tracking in Wireless Aerial Sensor Networks", in Communications (ICC), 2014 IEEE International Conference on, pp. 373 - 378, June 2014.

[11] S. Uludag, **M. Karakus**, E. Guler, S. W. Turner, and A. Kita, "Assessment of a Frugal, Virtual and Green Computing Lab Infrastructure of the Future", in Frontiers in Education (FIE), 2012 International Conference, Oct 2012.

[12] **M. Karakus**, S. Uludag, E. Guler, S. W. Turner, and A. Ugur, "Teaching Computing and Programming Fundamentals via App Inventor for Android", in Information Technology Based Higher Education and Training (ITHET), 2012 International Conference on, pp. 1 - 8, June 2012.

[13] E. Guler, S. Uludag, **M. Karakus**, and S. W. Turner, "Virtualized Lab Infrastructure on a Budget for Various Computing and Engineering Courses", in Information Technology Based Higher Education and Training (ITHET), 2012 International Conference on, pp. 1 - 7, June 2012.

[14] S. Uludag, E. Guler, **M. Karakus**, and S. W. Turner, "An Affordable Virtual Laboratory Infrastructure to Complement a Variety of Computing Classes," J. Comput. Sci. Coll., vol. 27, pp. 158 - 166, May 2012.

[15] S. Uludag, **M. Karakus**, and S. W. Turner, "Implementing IT0/CS0 with Scratch, App Inventor for Android, and Lego Mindstorms", in Proceedings of the 2011 ACM conference on Information technology education, SIGITE'11, ACM, 2011.

The paper has been awarded by "Meritorious Paper Award".

Grants

The EU Commission Grant for ERASMUS Program *Jan. 2008 - Jun. 2008*

Awarded money to pursue one semester abroad education in Technical University of Lodz by ERASMUS Student Exchange Program in Europe, Lodz, Poland

€1,325

Travel Grants

ACM SIGITE 2011 Conference *October, 2011*

Presented a paper at the conference.

$200 (Grantor: ACM SIGITE)

$500 (Grantor: Office of Research at the University of Michigan - Flint)

$400 (Frances Ann Frazier Student Travel Scholarship from UM-Flint)

IEEE FIE 2012 Conference *October, 2012*

Presented a paper at the conference.

$500 (Grantor: Office of Research at the University of Michigan - Flint)

$800 (Frances Ann Frazier Student Travel Scholarship from UM-Flint)

ACM CCSC-CP 2012 Conference *March, 2011*

Presented a paper at the conference.

$700 (Frances Ann Frazier Student Travel Scholarship from UM-Flint)

Service

University of Michigan-Flint Chess Club *May 2011 - May 2013*

Formed and served as the president of the club.

Turkish Student Association (TSA) at UM-Flint *May 2011 - May 2013*

Served as the vice-president of the TSA.

Super Science Friday Event *May 2012*

Performed a Lego MindStorm Robotic demo, coded by me, to get K-12 students interested in computer science.

Academic Article Reviewer *January 2016 - Present*

Computer Networks, Elvesier

Journal of Network and Computer Applications, Elvesier

Scholarships and Awards

M.Sc and Ph.D. Scholarships, Turkey *December 2009 - December 2018*

Awarded a scholarship to study M.Sc and Ph.D. in the US by the Republic of

Turkey Ministry of National Education, among all the university graduates in 2010 (Only 948 of all graduates in Turkey received the scholarship)

Assistantships

Teaching Assistant *August 2014 - Present*

Indiana U. - Purdue U. Indianapolis (IUPUI), Indianapolis, IN

Research Assistant *August 2013 - August 2014*

Indiana U. - Purdue U. Indianapolis (IUPUI), Indianapolis, IN

Graduate Student Research Assistant *September 2012 - August 2013*

Studied in the project, "Wide Area Situational Awareness modeling of the Smart Grid using Multivariate Statistical Tools", as a Research Assistant at the Univer sity of Michigan-Flint, Flint, MI

Graduate Student Research Assistant *September 2011 - September 2012*

Studied in the project, "Developing Multivariate Stochastic Composite Metric for Routing in Wireless Mesh Networks", as a Research Assistant at the University of Michigan-Flint, Flint, MI

Grader and Lab Assistant *June 2011 - May 2013*

Worked as a Grader and a Lab Assistant for various undergraduate courses at the University of Michigan-Flint: Telecomm & Comp Networks (Fall 2011), Fluency w/ Info Tech & Computing (Summer 2011), Using a Unix Computer System (Winter 2012), Prob Solving & Programming II (Summer 2012).

Professional Affiliations

ACM Student Member *2015 - Present*

IEEE Student Member *2015 - Present*

IEEE ComSoc Member *2015 - Present*

IEEE Young Professionals Member *2015 - Present*

Employment History and Experience

Teacher *June 2009 - January 2010*

Worked as a full-time high school mathematics teacher in Isparta, Turkey

Extra Curricular Activities

Soccer Referee *January 2011 - Present*

Serving as a USSF-licensed Grade 7 level soccer referee

Soccer Player *2002 - 2005*

Played soccer in the youth team of Tokatspor soccer club, a professional soccer club playing in 2^{nd} league in Turkey