QUALITY OF SERVICE IN SOFTWARE DEFINED NETWORKS

BY

AHMED SAEED AHMED BINSAHAQ

A Dissertation Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

In

COMPUTER ENGINEERING

APRIL 2021

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This dissertation, written by **AHMED SAEED AHMED BINSAHAQ** under the direction of his dissertation adviser and approved by his dissertation committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **DOCTOR OF PHILOSOPHY IN COMPUTER ENGINEERING**.

Dissertation Committee

Dr. Tarek R. Sheltami (Adviser)

Dr. Shokri Z. Selim (Member)

Dr. Sadiq M. Sait (Member)

Dr. Khaled H. Salah (Member)

Dr. Ashraf S. Mahmoud (Member)

Dr. Aiman H. El-Maleh
Department Chairman

Dr. Suliman S. Al-Homidan
Dean of Graduate Studies

Date

To my family

ACKNOWLEDGMENTS

First and foremost, I am grateful to Allah, the Most Merciful and the Most Gracious, for His guidance and blessings without which this dissertation would not have been possible. My greatest appreciation to my advisor, Dr. Tarek Sheltami for his guidance, patient, and valuable help to complete this work. A special thanks to my committee members for their support and valuable feedback.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	Application Programming Interface
CDPI	Controller-Data Plane Interfaces
CSP	Constraint Shortest Path
DiffServ	Differentiated Services
DPI	Deep Packet Inspection
DRL	Deep Reenforcement Learning
DSCP	Differentiated Services Code Point
FIFO	First In First Out
IntServ	Integrated Services
LARAC	Lagrange Relaxation based Aggregated Cost
LLDP	Link Layer Discovery Protocol
MCSP	Multiple Constrained Shortest Path
MOS	Mean Opinion Square
NFV	Network Function Virtualization
ONF	Open Networking Foundation
OVSDB	Open vSwitch Database Management Protocol

PSNR	Peak Signal to Noise Ratio
QoS	Quality of Service
RSVP	Resource Reservation Protocol
RTP	Real-time Transport Protocol
RTSP	Real Time Streaming Protocol
SBI	South-Bound Interfaces
SDN	Software Defined Network
SFC	Service Function Chain
SLA	Service Level Agreement
SLO	Service Level Object
SVC	Scalable Video Coding
ToS	Type of Service
VNF	Virtual Network Functions
VoIP	Voice over IP

DISSERTATION ABSTRACT

NAME: Ahmed Saeed Ahmed BinSahaq

TITLE OF STUDY: Quality of Service in Software Defined Networks

MAJOR FIELD: Computer Engineering

DATE OF DEGREE: April 2021

Software-Defined Networking (SDN) is a fast emerging networking paradigm with greater network control flexibility that promises to provide end-to-end Quality of Service (QoS) guaranteeing. In the literature, there are many proposals for developing QoS frameworks in SDN. However, many of these proposals adopt per-flow sequential resource allocation. Such behavior may hold in situations where there is auxiliary of resource's availability or some degree of traffic homogeneity. Otherwise, it may reduce resource utilization and cause fairness problems even within the same traffic class. In this dissertation, Firstly, we propose a Multi-Class Group-based Resource allocation and Refinement (MCGRR) algorithm. For each traffic class, it solves the flow-path and bandwidth assignment problem for a group of flows. The algorithm performs two tasks: a fast initial resources allocation and a post-allocation refinement process where a group of flows can reassign resources in between for fair allocation. Moreover, the

algorithm reduces the resource allocation process time by reducing the search space using small-sized state information from previous allocation attempts to cut down the search process as fast as possible. Secondly, we propose a LAgrange Relaxation-based Aggregated Cost (LARAC) based algorithm called MODLARAC to solve the Delay Constrained Least Cost (DCLC) problem with less computation time. We improved the solution feasibility search state by exploiting LARAC's lower-bound paths before the approximation process start and modify the stop condition to avoid extra non-useful Dijkstra calls. Obtained results showed better performance compared to recent research works.

ملخص الرسالة

الاسم: أحمد سعيد أحمد بن سحاق

عنوان الدراسة: جودة الخدمة في الشبكات المعرفة برمجيا

التخصص: هندسة الحاسب الالي

تاريخ الدرجة العلمية: أبريل 2021

تعد الشبكات المعرفة برمجيا (SDN) نموذجًا سريعًا للشبكات الناشئة مع قدر أكبر من المرونة في التحكم في الشبكة والذي يعد بتقديم ضمان جودة الخدمة (QoS) من البداية إلى النهاية. في الأدبيات ،هناك العديد من المقترحات لتطوير أطر جودة الخدمة (QoS) في الشبكات المعرفة برمجيا. ومع ذلك ،فإن العديد من هذه المقترحات تعتمد تخصيص موارد الشبكة بترتيب تسلسلي لكل تدفق او طلب خدمة. مثل هذا السلوك قد يكون مقبول في الحالات التي يكون فيها هناك وفرة في الموارد أو درجة معينة من تجانس نوع الطلبات الخدمة. خلاف ذلك ،قد يقلل من استخدام الموارد ويسبب مشاكل في الإنصاف حتى بين الطلبات من نفس الفئة. في هذه الرسالة ، أولاً ، نقترح خوارزمية تخصيص الموارد وتحسينها على أساس مجموعة متعددة الفئات (MCGRR). لكل فئة من فئات طلبات الخدمة ،الخوارزمية المقترحة تحل مشكلة ايجاد مسلك او مسار لتدفق بيانات الطلب و الموارد المناسبه لكل طلب. فالخوارزمية تؤدي مهمتين: تخصيص مبدئي سريع للموارد وعملية تحسين ما بعد التخصيص حيث يمكن لمجموعة من طلبات الخدة إعادة تعيين الموارد فيما بينهما لتحقيق التوزيع العادل للموارد. علاوة على ذلك ، تقلل الخوارزمية من وقت عملية تخصيص الموارد عن طريق تقليل مجال او نطاق البحث باستخدام معلومات محفوظة من محاولات التخصيص السابقة لتقليص عملية البحث في أسرع وقت ممكن. ثانيًا ، نقترح خوارزمية تسمى MODLARAC لحل مشكلة ايجاد مسلك او مسار لتدفق بيانات داخل الشبكة بأقل تكلفة مع وقت التقيد بايجاد سقف زمني لذلك المسار. لقد قمنا بتحسين حالة البحث عن مسارات البيانات من خلال استغلال المسارات المحدده مسبقا فيي بدايه العملية والحد من عمليات البحث الإضافية الغير المفيدة. أظهرت النتائج التي تم الحصول عليها أداءً أفضل مقارنة بالأعمال البحثية

الحديثة.

CHAPTER 1

INTRODUCTION

Today's Internet shows a massive increase in Internet-connected devices and the diversity of network killer services and applications such as multimedia streaming, network storage, and online gaming. In 2017, Cisco reported that the global annual IP traffic reached 1.2 Zettabytes in 2016 and it is expected to reach 3.3 Zettabytes per year by 2021 [2]. In addition 82% of the consumers' Internet traffic will be IP video traffic by 2021. From the overall Internet video traffic, 13% will be live video traffic [2]. Also, the ratio of Internet-connected devices will be three times the global population.

With this pressure on today's Internet, service providers were between two choices, either put in extra resources into their networks, which adds extra costs, or apply strict policies, which may not satisfy their customers. In addition, they are obligated to provide services with certain quality as it is agreed to in the Service Level Agreement (SLA). However, the massive increase in the numbers of Internet-connected devices (e.g. user's mobile devices, servers, Things, etc) is promising a lot of money at stake for service providers from now and upon[2]. At that point, providing services with certain Quality of Service (QoS) guarantee while efficiently utilizing the network resources

becomes a challenging task for many service providers or network operators [3].

There are two models proposed to support QoS over the Internet, namely: Integrated Services (IntServ) [4] and Differentiated Services (DiffServ) [5]. The earliest one is designed to reserve sufficient resources along the path that the service packets use. The work is evolved into a design of a signaling protocol called the Resource Reservation Protocol (RSVP) [6] that was designed to work over IP protocol. However, it has a scalability problem since it needs to keep state on each network node about each going traffic flow. In *DiffServ*, packets are tagged to differentiate between the service level they require (i.e. VoIP requires a 150ms maximum delay) or receive. Therefore, each packet is treated differently based on its tag or class and the QoS policy is configured on each node. By doing that, different QoS classes can be defined for variant services. *DiffServ* is designed to provide end-to-end QoS support by assuring QoS application on each intermediate node within the network.

DiffServ cannot provide guarantees like those offered by *IntServ*, and suffers from dynamic changes of application requirements. Moreover, the necessity to reconfigure each node within the network to reflect a change in QoS policy or new SLA requirement causes a huge headache for network operator probability of mistakes is high. The usage of automated tools could be a solution. However, due to the diversity of network vendors and the complexity of the current architecture (i.e. traditional networks with different types of equipment) makes it worse. To alleviate those problems, a new architecture that supports generality and programmability of network elements is defined by Open Networking Foundation (ONF) [7]. By separating the control for the data planes it provides agility and flexibility for the network operator. The new

architecture is called Software Defined Network (SDN) [8].

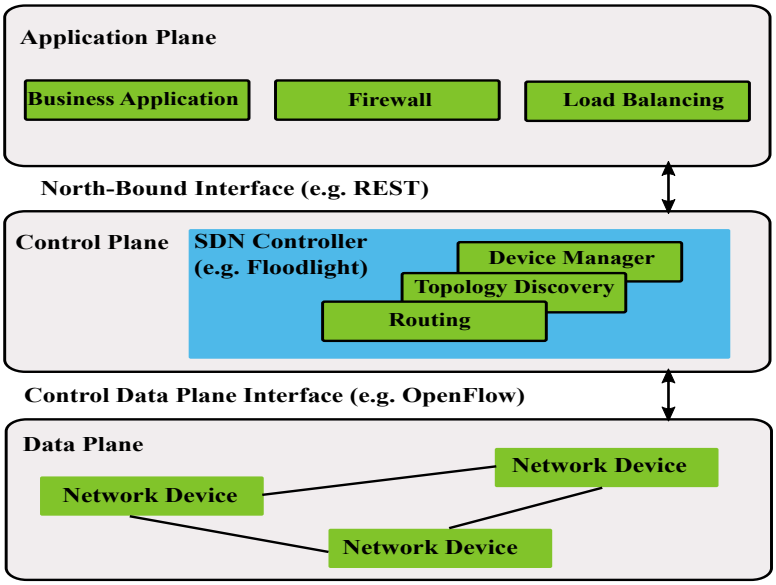


Figure 1.1: Software Defined Networking architecture

1.1 Software Defined Networking

SDN is a new network architecture that decouples control plane from data plane differently than the current network architecture [8]. The idea behind it was to use flow tables inside network devices and a standard interface for flow table configuration, management, and manipulation. The standard protocol to accomplish this task was called OpenFlow [9]. Those flow tables contain rules that match packet header fields and actions to perform on those matched packets [9]. OpenFlow protocol has become the de-facto protocol for SDN by which a central server controls network devices through OpenFlow primitives to install and delete flow table entries.

Fig. 1.1 shows an overview of the SDN architecture. Control and data planes are physically separated differently from today's widely deployed network architecture. In

which the network device (e.g., router) controls the forwarding tasks through routing protocol implemented within it. In the SDN, there are three separate planes:

- **Data Plane**. This plane consists of OpenFlow compliant forwarding devices of either physical or virtual switches. They can be considered as dump forwarding devices with OpenFlow firmware installed on. OpenVswitch[10] is an example of such platforms. Many vendors of network devices made their new products compatible with OpenFlow protocol as Table 1.1 shows. The communication between SDN controller and the switch is performed using a South-Bound Interfaces (SBI) or Controller-Data Plane Interfaces (CDPI) by which the controller can install, modify or delete flow tables rules.

- **Control Plane**. In this plane, the control of the whole network devices resides. Many other functionalities such as forwarding, topology discovery, fault detection and many others could be performed in this plane. SDN controllers are the embodiment of the SDN architecture. The controller provides an abstraction of the underlying network infrastructure and services to the network applications. Many network applications such as load balancer can obtain information through through North-Bound Interfaces (NBI) from the controller. For instance, a load balancer can retrieve information about the status of the network switches or links since they are maintained and monitored by the controller. The main drawback of the SDN is that the control functionality is centralized which makes it a single point of failure. However, many researchers have proposed solutions to overcome such problem by using multiple distributed controllers [11].

Table 1.1: Sample of OpenFlow Compatible switches

Ref	Switch	H/S	Management Protocol	Provider	QoS Support
[10]	OpenVSwitch	S	OpenFlow (1.1, 1.2), OVSDB	Linux Foundation	Traffic Queuing and Shaping
[?]	ofsoftswitch13	S	OpenFlow (1.3)	Ericsson Innovation Center	Traffic Queuing and Shaping
[?]	Indigo	S	OpenFlow (1.0)	Big Switch Networks.	–
[?]	Arista 7150	H	OpenFlow (1.0), SNMP	Arista	Traffic Queuing and Shaping
[?]	NEC PF5240 Switch	H	OpenFlow (1.0,1.3.1)	NEC	Rate limiting, bandwidth control
[?]	Pica8 P-3297	H	OpenFlow (1.4)	Pica8	–

- **Application Plane**. In this plane, network and business applications reside. Programmability of the SDN architecture allows these applications to interact with the controller without concerning of complexity of the network due to the abstract view that SDN provides and availability of network data through northbound Application Programming Interface (API). Which separate complex network configuration and from application developments.

The global view and control of the whole network allows operator to implement applications that adjust networks behavior to meet new needs [12]. The simplicity of service and network provisioning was the most important driver for investments in SDN deployment by many service providers [13] to achieve organization goals[14].

1.2 QoS support in OpenFlow

By design, SDN provides network operators with granular control over traffic flows. Forwarding rules installed within the flow table targeting a specific flow(s) using packet header information (i.e., 2nd-level, 3rd-level or 4th-level of TCP/IP stack). In its early specifications, OpenFlow allows actions to be taken on a flow's packet after it matches a rule within the flow table. In OpenFlow 1.0 [15], an *enqueue* action is defined by which a packet can be attached to one of the output port queues. It allows

Table 1.2: QoS in Traditional and SDN networks.

	Traditional		**SDN**
	DiffServ	**IntServ**	
Granularity	Per Class	Per Flow	Supports Both
Admition Control	NA	At each node	At controller
Policing (drops)	At edge or intermediated nodes	Edge of the network and at all source merge points	At edge or intermediated nodes
Shaping	At edge nodes	At intermediate nodes	At edge nodes
Metering	Edge nodes	At edge of the network and at all source merge points	Configurable meters (i.e. meter table) at edge or intermediate nodes
Scheduling	At intermediate nodes, based on port configuration	NA	At intermediate nodes, based on port configuration, packets steered using set-queue action
Complexity	At edge nodes with long-term setup	At all nodes with per-flow setup	At controller with dynamic per-flow setup
Signaling	NA	RSVP protocol	OpenFlow protocol
State	Low, per-class at the intermediate nodes and per-aggregate at the edge nodes	Soft, need refreshing per-flow at each node	Configurable hard/time-based per-flow at each node
Provisioning time	Time consuming	Signaling time	Automatic configuration and short provisioning-time with fast rollback capability due to its programmability
Scalability	High	Low	Medium
Packet matching	Multi-Field	Defined at the reservation setup as *FilterSpec*	Multi-Field
Resource Utilization	Affected by routing	Affected by routing	Can be Optimized through dynamic central control of QoS and routing configuration
Service Scope	Domain	End-To-End	Domain

the controller to assign different flow traffic such as those to belong to VoIP service into a specific queue (i.e., for sensitive delay requirements). This feature is limited compared to what is already defined in the traditional network devices. Moreover, queues configuration task is assigned to another protocol that is later defined called *OF-Config* [**?**]. In OpenFlow 1.1 [16], the ability of multi-level VLAN/MPLS and traffic classes (i.g. ToS/DSCP) matching was added. Also, an action was specified to add, remove or modify those labels (or tags). Moreover, group tables are introduced which allows flows aggregated action to be performed. A flow entry(s) in the flow table can point to an entry in the group table for more processing or to aggregate statistics separately than forwarding rules. In OpenFlow 1.2 [17], the controller becomes able to query switches for queues configuration such as max-rate. In OpenFlow 1.3 [18], the ONF working group guidelines offer to implement rate limiting by using meter tables with its structure shown in Fig. 1.3. Meter table contains meter entries each one is identified with a 32bit unsigned integer identifier. A meter can be attached to flow entry. Counters within meter to count packets that been processed by the meter. For rate limiting, meter bands are used to compute the rate for flows attached to that meter aggregately. Band type specifies what to do such as marking the packet with DSCP value or drop it (i.e., by adding it to the action or instruction set to execute). Therefore, if the current flow(s) rate is below configuration, the meter band is not executed. Multiple meter band can be defined, and the band is executed when the rate configuration is reached. This feature can be used to monitor packet entering the ingress port and apply metering or rate-limiting on their flows. In OpenFlow 1.4 [19], the controller is supported with a scheme to monitor a subset of the flow table(s)

in the switch. This feature is added due to the support of distributed controllers, in which many controllers can manage the same network. Therefore, a controller can install monitors on a subset of the flow table, and whenever an addition, modification or removal or flows happen, an event packet is sent to the controller to inform it about that change. OpenFlow 1.5 [20] introduce new features such as the egress tables in output ports. It also adds idle time statistic to count the time this flow entry is idle. Also, TCP flags matching is added (i.e., such as SYN, ACK and FIN), in which it helps to detect the start/end of TCP connection. Moreover, flow statistics triggers are added to allow the switch to push-up statistics (i.e. beside the default poll-based) toward the controller when their configured threshold/timeout is reached. Furthermore, port configuration changes notification message is introduced to alert other controllers in a distributed deployment. Table 1.2 compares main features of QoS in traditional and SDN networks.

Each network system should know existing services and the state of its components to support QoS provisioning. SDN uses distance-based monitoring of network elements since the control is separated from those devices and located in a logically centralized place. OpenFlow provides within its specification a messaging mechanism that allows the controller to poll statistics (i.e., about flows, tables, ports, and queues) from those compatible devices [19].

In SDN, the monitoring function can be implemented within or above the controller to observe the network state. The controller internally keeps three different views of each device; capabilities, configurations, and statistics as shown in Fig. 1.2. Upon establishing the connection with the controller, each network device sends its

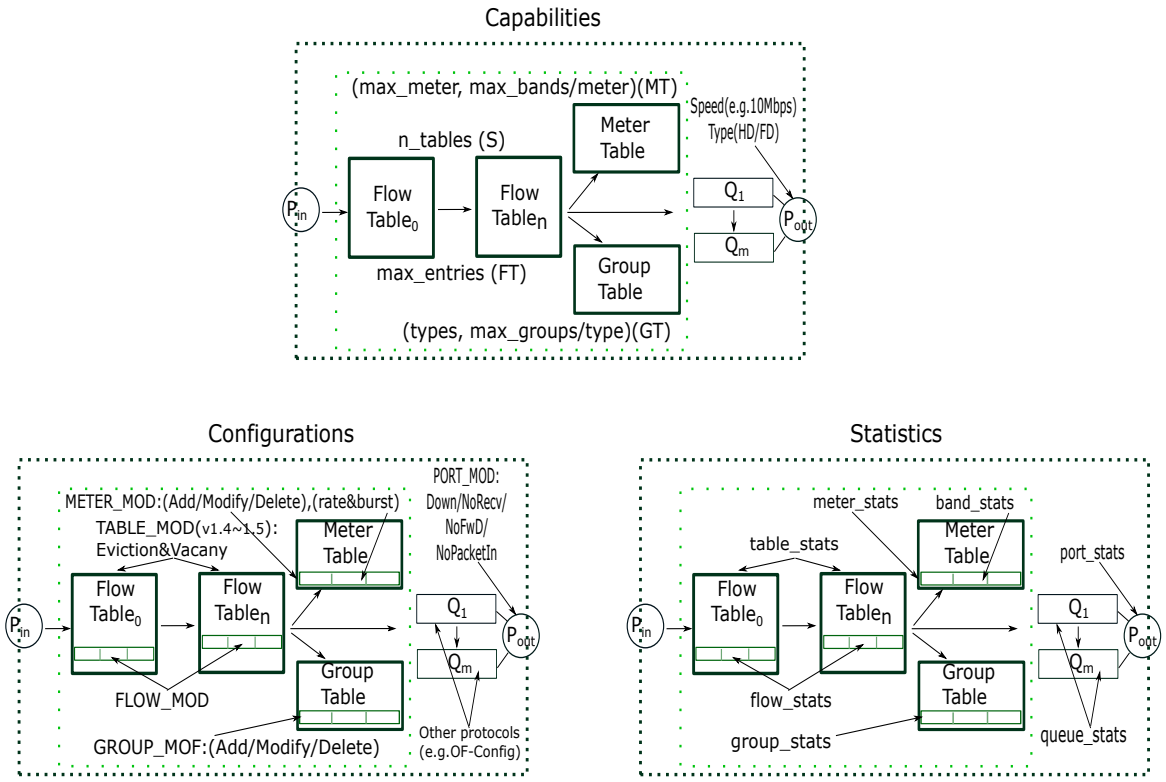


Figure 1.2: Three views of OpenFlow (OF) Switch from the controller point of view

capabilities(i.e., actually the controller asks for it by sending a feature request to the switch). Features such as the maximum number of flow tables, and if the switch supports metering or grouping of flows and how many flow entries each table can hold. Moreover, it knows the speed, and the configuration of connected links and the number of queues each line has as shown in Fig. 1.2. This helps the controller to know what the network can do. Beside capabilities, the controller can get the current configuration of each device. To know the state of the network, the controller can send *StatReq* messages to poll statistics from those network devices as shown in Fig. 1.3.

Fig. 1.3 depicts the format of the control messages between a switch and the controller. Whenever a new flow arrives, a *PacketIn* message is sent to the controller

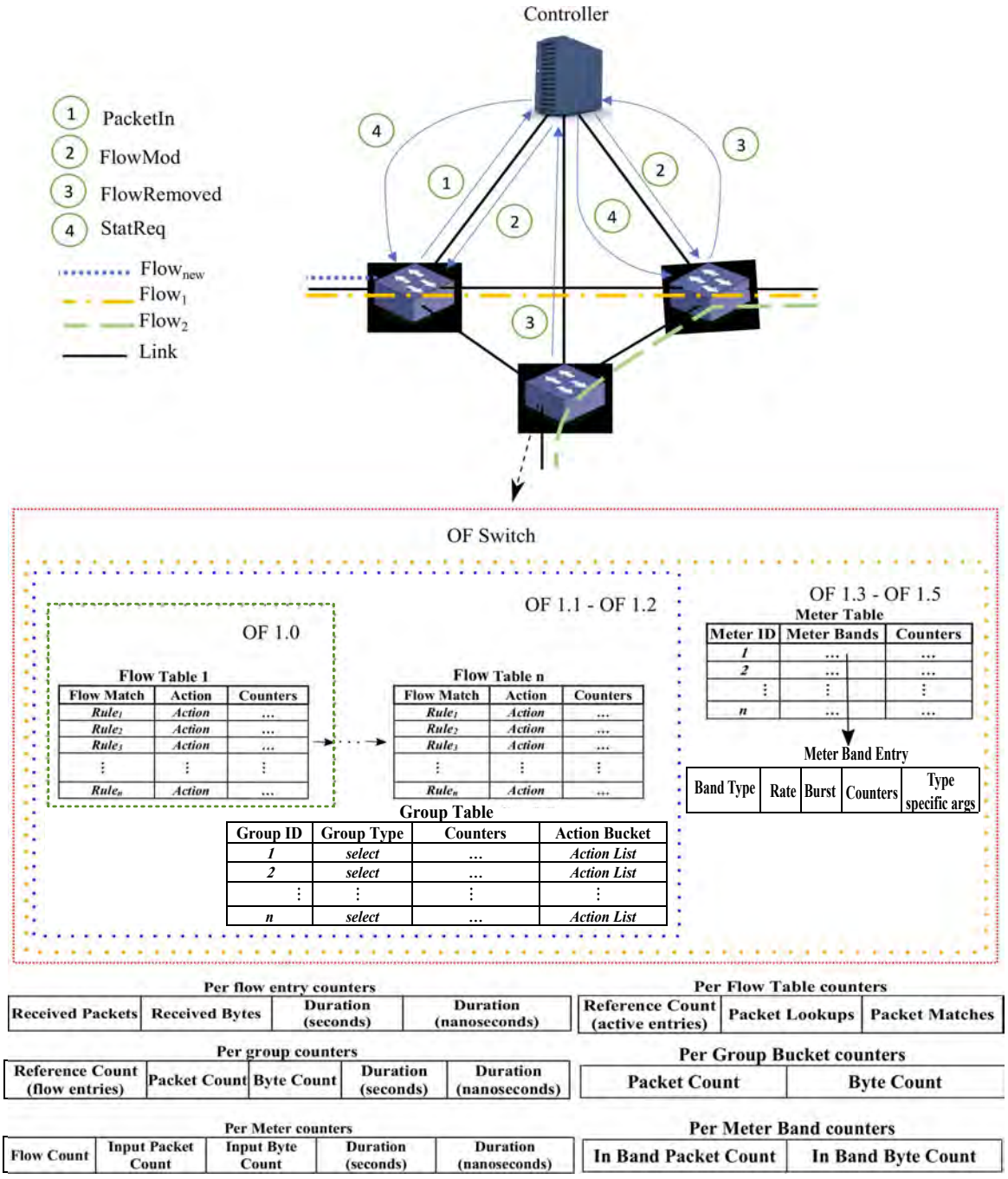


Figure 1.3: ONF architecture for OpenFlow

informing it about this unrecognized flow packets. Upon reception of that packet, the controller replies by *FlowMod* message(s) to create a new entry for that flow within switches along the specified flow path. A flow entry holds counters for packets matching that flow. Moreover, other counters for that flow can coexist in other tables in that switch such as the meter table that is used to band the traffic generated by that flow. An entry expires after certain time of inactivity called *IdleTimeout* or after a *HardTimout* elapsed even if the flow still active. At which, the switch removes expired flow entry and informs the controller by sending a *FlowRemoved* message (i.e., a flow-removed flag is used along with that flow entry). *FlowRemoved* packet contains statistics about that dead flow. Fig. 1.3 shows the structure of those counters in each switch and which OpenFlow version support.

1.3 Motivation and Problem Statement

Support of QoS is extensively addressed in the literature for traditional networks. However, the emergence of SDN opens the door to tackle that problem again. Promising features, such as the decoupling of network control from forwarding, also the global view of the whole network, gives SDN an advantage to support QoS. In the literature, there are many proposals for developing and improving QoS frameworks in SDN [21] [22][23]. From the QoS metric perspective, bandwidth is the primary QoS metric demanded by many network services and multimedia applications. Limitation of resources causes network congestion, service level degradation. However, throwing more resources into the network is also expensive and reduces the business revenue. There-

fore, the efficient use of resources that maximize revenue is an important goal for the service provider. In most QoS provisioning frameworks, we noted that requests are admitted as they received, in the sense of first come, first served(i.e., a flow per flow). This sequential treatment may hold in situations where there is auxiliary of resource's availability or some degree of traffic homogeneity, which not the case in the current Internet. One problem of flow-per-flow resource allocation is that it does not respect traffic class prioritization since low priority traffic may receive more resource due to their early arrival, leading to extra control effort by calling resources evacuation to accommodate future demands[24]. Moreover, resource allocation may not fairly be allocated between flows within the same class. A traffic engineering effort may help in solving post-admission problems by re-optimizing resources allocation[25]. However, such proposals may suffer from high computation overhead since they deal with many flows. Moreover, besides the messaging overhead, changing the current control configuration need to be carefully performed since it can cause service interruption and may lead to congestion in some network parts. Therefore, a group-based treatment might prioritize benefits and increase service providers' revenue to focus on the most profitable requests.

Moreover, an issue with SDN and its central control is that controller has to process all new incoming flows. This makes the usage of fast algorithms is a must. In QoS provisioning, bandwidth is a concave metric in which an optimal solution (i.e., for a single flow or request) can be easily found by pre-processing network graph. For instance, a residual resources graph can be obtained after pruning unsatisfying links from the original network graph. However, this problem becomes critical in other

additive metrics such as delay. Therefore, a design of a fast and QoS guaranteeing routing algorithm is still needed.

In this dissertation, we cover those issues and propose solutions to enrich the literature of QoS provisioning in SDN networks.

1.4 Research Objectives

This dissertation's main objective is to propose and implement solutions that improve QoS management and provisioning in the SDN network. We can summarize the objective as in the following:

- Conduct an extensive literature review in the field of QoS provisioning and management in SDN networks. We addressed this objective in Chapter 2, and as an output, we published a research article as in [23].

- Develop, implement and evaluate a resource allocation approach that improves resource utilization, fairness, and provider revenue. We address this objective in Chapter 3.

- Develop, implement, and evaluate a QoS-aware routing algorithm that is suitable for SDN networks. We address this objective in Chapter 4.

- Implement a mechanism to improve network state to support QoS in SDN. We address this objective in Chapter 5.

1.5 Dissertation Organization

The remainder of this Dissertation is organized as follows. In Chapter 2, we review the literature from the perspective of QoS provisioning life cycle in SDN network. Then, in Chapter 3, we introduce our proposed resource allocation approach, and evaluate its performance in comparison of recent related approaches. Next, in Chapter 4, we present our proposed algorithm to solve the Delay Constrained Least Cost (DCLC). We also experimentally evaluates its performance against opponent algorithms in the literature. In Chapter 5, we investigate SDN network state behavior and proposed a utility-based approach to improve the link state update process. Lastly, in Chapter 6 we conclude our dissertation, and future research directions.

CHAPTER 2

LITERATURE REVIEW

In the literature there are many review works discussed the development in SDN. For instance authors in [26] did a comprehensive survey in SDN that covers many subjects while authors in [27] conducted a systematic review in SDN. QoS provisioning in SDN also received its part of that research effort. In the literature, there are three surveys about QoS in SDN [21][22][23]. In [21], the authors discuss the concept of QoS and how it can be improved via SDN. They focused on four main fields, namely; resources reservation, flow-based routing, queue management, and policy enforcement. The work in [22] is more comprehensive than the previous one. The authors discussed more recent research work done aforementioned four fields. Beside of that, they focused on the user experience of QoS and network monitoring which specifically been discussed in the survey work in[28]. In [23], authors discussed the QoS provisioning and management from an autonomic functional point of view.

Figure 2.1 shows a conceptional view of the QoS provisioning process in SDN. In this chapter, we give a general review on the cycle of QoS provisioning in SDN. As it shown in Figure 2.1, the end-to-end process of QoS provisioning involve many modules

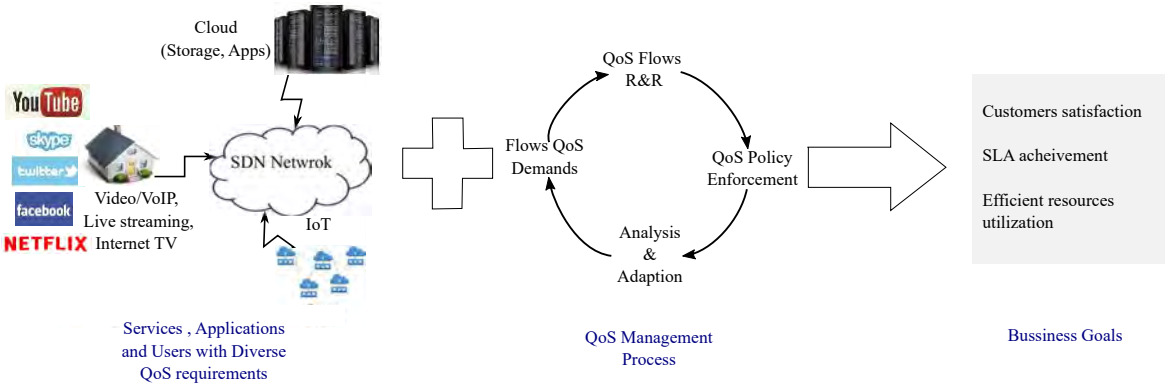


Figure 2.1: QoS Process Life Cycle

or functions that may performed by many parties. It can be summarized into four main functions, namely; 1) Flows QoS demands identification, 2) QoS-aware Routing and Resource allocation, 3) QoS policy enforcement, 4) Analysis and Adaption of current QoS flows. According to those four modules we organize this chapter and underneath we discuss the research literature in QoS provisioning and management in SDN.

2.1 Flows QoS Demands Identification

This module is the first entry point of any QoS management framework. It can be located at the control layer or splitted between control and edge of the network for load balancing. It identifies each QoS flow requirements or demands along with its priority or user subscription plan (i.e., Premium) and filter those QoS requests for subsequent modules. There are many research works in the literature related to this module. For instance, in [29] the authors use a Deep Packet Inspection (DPI) based traffic classification for getting flow information at the edge of the network and sent it to the controller. Based on the information received from the DPI classifier, the controller

knows to which service group a flow belongs and assign it to a service specific (i.e., VoIP sensitive to delay) configured queues within the network switches. Similarly, the authors in [30] uses a classification module to detect traffic type that traverses the SDN network and report the criticality of such flows (i.e. multimedia or video traffic) to the control plane. In [31], the authors used traffic classification at the edge and tags classified flow packets. For each flow, a service class reveals its QoS requirements (i.e.max delay, rate). For instance, at the controller, to admit the addition of a new flow to an existing path, the maximum delay required by its class is compared with the current path delay. Instead of using DPI classification, authors in in[32] depends on client side flow type identification through DSCP that is recognized at the control layer.

Instead of using DPI classification, authors in [33] utilize machine learning (ML) based flow characterization. An ML-based classifier (i.e., C4.5 Decision Tree) is used to classify flows. The controller uses the first 50 packets to extract flow features. Then, based on the produced flow class with application prioritization, the controller chose the appropriate path based on cost, and the application requirements (i.e., bandwidth and delay).

Other approaches try to intercept the end-to-end negotiation between intended communicated entities to identify the QoS level expected for that flow. For instance, the authors in [34] proposed an SDN-based architecture with a QoS module that match the QoS requirements or the ability between the end users. This module called QoS Matching and Optimization Function (QMOF). After negotiation using the SIP protocol, an optimized QoS profile is produced for the expected flow. Similarly, in

[35], the authors used Real Time Streaming Protocol (RTSP)/Real-time Transport Protocol (RTP) streaming setup packets and exploits include media information to setup QoS routes for their video flows. The authors in [36] proposed an SDN-based home or broadband traffic (i.e., user's HTTP based video traffic) management. They use DNS information to classify web applications such as HTTP(s) or any built upon it, and another DPI-based classifier to classify non-HTTP traffic.

Instead of inferring QoS requirements through traffic classification (ML or DPI based methods), users services can communicate with the controller through the north-bound interfaces. A network application can specify networks requirement to the controller to create flows rules. Whenever there are no more needs, the SDN controller can delete and install new rules to roll back the network to its previous state. For instance, a user wants to perform a conference call to another one or attend an online course. The user application will negotiate service and network communication parameters between end users before starting the flow of the service (e.g., audio or video). For instance authors in [37][38], allows a streaming service (e.g.,YouTube) or application to contact the SDN controller through north-bound API to tell about the type of information its flow contains(i.e., the buffer playtime threshold). It allows the controller to put a high priority of such flows over other existing flows.

Authors in [39] implemented a northbound API in the SDN controller to allow developers of video streaming application to request QoS requirements such as bandwidth, delay and so on. Similarly, the authors in [40] [41] proposed an SDN-based architecture to support the need for Real-Time Online Interactive Applications (ROIA) users such as gaming applications. The satisfaction of users in these services depends

on the responsiveness in which loss and delay of action (i.e., transferred through packets) is a significant concern. These user or application' needs should be specified dynamically to the network. Authors used the SDN due to its dynamic configuration through northbound API to accommodate these needs. Authors implemented an SDN module (within their proposed architecture) that is responsible for the communication between the SDN controller and the ROIA process that resides in the application server. Users are connected to this process and keep communicating with it during the game session time. The application requirement such as response time is transmitted as QoS policy for either one or aggregated flows between the ROIA process and ROIA client(s) toward the SDN controller through the north API.

The authors in [42] target these challenges in the field of IP Multimedia Subsystem (IMP) while designing an architecture for IPTV service. They use users rating of the services to reconfigure QoS network parameters. A QoE engine at the client side tries to learn about end user by using different profiles (action movie, talk show) within the ongoing time session and the user rating of the service quality. A prediction of the user's level of satisfaction is mapped into QoS network parameters base on these collected data (users rating) and the category of the service. They used linear regression to map between QoE into network level QoS parameters which been reflected into changes on the network rules that are enforced through the OpenFlow protocol. Similarly, the authors in [43] tried to give a score for QoE in which lower score means terrible services at the user side. The used an agent that monitors the service on the user side (e.g., IPTV networks or Video services such as Netflix) by recording network parameters such bandwidth, jitter and delay from the received packets on the user

side. Then it pushes these data to the service provider side where the QoE score is computed where actions are taken on the SDN controller to improve the QoS parameters for that user(s) flows. In [44], the authors propose a video streaming based on SDN (VSDN). The architecture allows the QoS application to request resources from the controller. It is a reservation based proposal by which a sender can request QoS for a specific video specification through QoS API to allow the controller to reserve the resources.

In QoS provisioning, it is an advantage if users' QoS requirements are translated automatically into low-level network policy if they have the right of that and the network can handle it (e.g., the user pays extra money for extra Video quality). It will help both users and service providers specially with the increased demand of value-added services in which the user has a choice to receive special treatment for their traffic over basic Internet quality as they pay extra money and it will help provider increase their revenue. However, this requires more investment in API standardization, trust policies to alleviate possible security problems.

2.2 QoS-aware Routing and Resources Allocation

This module represent the a joint process of finding an end-to-end path(s) that comply with the QoS demands, and efficiently utilize network resources that increase the business revenue. In this section we will try to cover each sub-process as a stand alone module and discuss the underneath research work.

2.2.1 QoS-aware Routing

In the SDN architecture, the controller maintains an abstract view of the network. Most of SDN controllers contain a module that provides routing and rules installation service. For instance, the Floodlight [45] controller implements the Shortest Path(SP) routing or what is called the Dijkstra's algorithm [46]. That module also computes a set of k shortest paths between all network pairs and cache them for fast future use. Due to the central view and up-to-date state of the network, it is easier to compute the shortest path within the SDN. The dynamic changes of network state and flows arrival rate affects the routes computation frequency. For instance, routes may be computed once in a proactive mode (i.e., off-line routing or planning) with the advance knowledge of all flows, and their demands are fixed or merely changing. In this dissertation We consider online routing that a path search is initiated for every arriving flow(s).

In QoS provisioning, routing plays a significant role to guarantee the required QoS demand(s). It tries to find path(s) that satisfies QoS requirement for flows. However, the problem is more complicated since the routing algorithm should satisfy the flows requirements. Otherwise, the service quality may start degrading if the used routes are congested. Based on the QoS requirements, different routing problems can be defined:

Shortest Path (SP)

In shortest path variant algorithms, a single QoS metric is considered while optimizing path computation. In [47], the proposed approach implements a modified version of the shortest path algorithm in the ONOS controller. It selects the least load shortest path to route QoS traffic. While in [48], the authors use the shortest path for non-QoS traffic with the hop count as the path cost. For the QoS traffic, the algorithm computes the shortest path from a residual resources graph. Links that do not satisfy the new bandwidth requirement are removed from the original network graph. The outcome is the shortest path that satisfies the bandwidth requirement.

In [35], the authors used an SDN-based approach that intercepts RTSP/RTP streaming setup packets and exploits them to setup QoS routes for their video flows. They used the shortest path with link weights. Links that have higher weight have a priority over others to be chosen in the QoS route setup. The link weight depends on the utilization of that link. Similarly, in [49], the routing module produces a set of available paths and their costs. The cost of a link is computed as the sum of received and sent bytes by the switch's port. Based on that cost, the shortest path is used. In [50], the link cost metrics are computed based on received network state information. Specifically, the port sent and received bytes at both arc nodes. Then, data and loss rates of the link are used with its configured capacity to compute the link cost value dynamically. After that, the routing module computes the shortest path that has the minimum cost.

In [51] authors define two traffic classes, a priority traffic which require bandwidth

guarantee (e.g., video traffic) and best effort traffic that represent the majority of Internet traffic. They used a modified Dijkstra routing algorithm that finds the shortest path that has sufficient bandwidth for the QoS flow. By using bandwidth as the constraint, their goal was to minimize the delay QoS metric as it is the most critical for multimedia applications. To avoid degradation of best effort traffic, they used weight metric for links that compute how much-utilized link is. They attempted to avoid those highly utilized link by routing best effort traffic on links less utilized or have more bandwidth.

The routing in [52] is a combination of the shortest path and the shortest-widest path. It finds the widest path and compares its bandwidth required by the flow. If it is less than what is feasible for the flow, then it is chosen as the bandwidth the flow will have. Otherwise, it uses the flow's feasible or required bandwidth. Based on the chosen bandwidth's value, a run of the shortest path is performed to find the path which can offer that bandwidth demand. By doing that, they attempt to save more rich paths for the future if the required bandwidth is small and another path can serve it. In general, the approach finds the shortest bandwidth feasible path. The approached proposed in [53] uses bandwidth information from the monitoring unit for each port. For a QoS flow, it checks if there is an alternative path with better bandwidth than the shortest path. If such a path is found, use it for that flow.

The work in [54] applied off-line and online routing. Routes are computed (off-line) when the network is initialized and recomputed when there is a change in the topology. When a new QoS request arrives, the online routing takes place. It first checks if an existing path (i.e., from off-line computed paths) can satisfy the bandwidth demand.

Then, the feasible shortest path is chosen. If no off-line bandwidth sufficient path exists, the shortest path is used to get a path from the residual graph where links below the bandwidth demand are filtered out. If such a path could not found, the request is rejected.

In [55], the proposed approach monitors the link bandwidth utilization. Upon detection of link congestion (i.e., utilization over a certain threshold), the routing module is invoked to find alternative paths that satisfy flows bandwidth requirements. The shortest path with available bandwidth is chosen as the new route to resolve such a situation.

In [56], the authors proposed an algorithm to route layered Scalable Video Coding (SVC) based video streaming in SDN. They classify the traffic into three classes; the base layer of the video as lossless QoS, the enhancement layer as a lossy QoS and the other traffic as a best effort class. The routing algorithm may choose different paths for the video layers. They used a modified version of Dijkstra algorithm with a weighted metric composed of the mean link loss and delay. Upon reception of a client request for a video service, the controller finds a path toward the video server and pick a Type of Service (ToS) value for each video layer. Then, the server uses these values when sending the video layers packets (i.e., tagged with agreed ToS value). By doing that, the controller knows which layer these packets belong to by just inspecting the one-byte ToS value. This method helps in avoiding the usage of deep packet inspection. However, ToS value may be changed by other parties the packets traverse their networks.

The shortest path algorithm is used extensively in the literature. A well-designed

QoS-aware cost function may add to the fast and simplicity of path computation provided by the shortest path algorithm. However, it does not guarantee QoS for each of those combined metrics.

(Multi-)Constrained Shortest Path (CSP and MCSP)

The shortest path gives an optimal path according to a specified cost metric (i.e., in QoS optimal path, the cost is a QoS metric). However, and in many cases, other QoS metrics may need to be considered and maintained within acceptable bounds as it is specified in the SLA contract. Therefore, another form of QoS routing problem is reported to find an optimal path according to one QoS metric while controlling another within required bounds.

In [57], the authors optimized the path selection for videoconferencing flows based on its delay as the QoS perused metric. They computed the cost metric as the weighted sum of lost packets and delay. Their objective is to adapt between the delay and the cost while computing paths for those flows. In [58], the authors propose an architecture to support QoS video flows in SDN that differentiated between two layers in the SVC encoded video. A base layer that should receive a particular QoS in which packet loss is forbidden and an enhancement layer that is classified as the best effort traffic. The approach directs the base layer traffic to non-congested routes due to its primary effect on the video quality. The model formulated by the authors consider packet loss as the primary constraint for path selection in which a zero loss should be achieved. Similarly, in [59] authors divided the video stream into two layers: a base layer served as QoS-traffic that should have higher priority and an enhancement layer as a second-

level QoS traffic. They performed optimization on route selection and formulated the QoS-aware routing as a Constraint Shortest Path (CSP) problem. They showed that using that dynamic routing for QoS traffic improved the overall video streaming quality. The packet loss measurement is defined similarly to [58] but with separation of the QoS traffic into two levels. Also, they solve the CSP problem two times to find routes for 1-level and 2-level QoS traffic respectively which may make it run slower. Also authors in [60] divide the SVC encoded streaming video into layers. However, they modified the problem to consider enhancement as QoS traffic with a possibility of packet drop. They showed that the second modification of the problem improved the video quality which is expected since extra frames can be received at the receiver side better than when they considered them as best of effort traffic.

In [61] authors define two types of traffic, a QoS traffic, and the best effort. They modeled the problem as Constrained Shortest Path and used Lagrange Relaxation based Aggregated Cost (LARAC) [62] algorithm to solve it. The algorithm tries to minimize the cost function while keeping path's delay below a maximum constraint value. The cost function calculated as a weighted sum of link delay variation and packet loss. However, their work uses QoS routing only when congestion has already happened which may affect flows that have a short lifetime.

In [63], the authors classify the traffic as multimedia and data traffic. They used the LARAC algorithm for computing the routes for multimedia flows dynamically, in which congestion and delay are used as the routing decision metric. They constrained the routing problem with predefined delays. They consider the link is congested if its utilization is above that value. Links with less than 70% utilization their congestion

measurement is zero, which link delay as the main factor for the routing decision. By doing that, almost congested links can be chosen if the statistics packets got dropped or update interval is long. Another concern is that they put the values of the delay in the cost calculation equals to 1 (i.e., they said it is because the current OpenFlow switch implementations do not have any support for collecting delay related statistics) which make the algorithm works based on hop count when the links are not congested as its used by the shortest path algorithm. Also authors in [64] model the routing as delay constrained least cost problem. They solved it by using kLAM approximation.

In [65] authors targeted video steaming over SDN. They divide the video stream into two layers, 1-level QoS is the base layer which has high priority and intolerant to packet loss or delay variation. Moreover, another 2-level QoS which is the enhancement layer. The third type of traffic is the best effort which is the majority. The authors propose a reroute approach for the base layer traffic (1-level QoS). The routing module computes a feasible path if this path satisfies the delay variation constraint and has a bandwidth that is enough to add the 1-level traffic to it, then the 1-level (base layer) is flow is rerouted using this selected path. Otherwise, if the bandwidth is not enough, the enhancement layer is rerouted, and the base layer stays using the shortest path. However, if the delay variation constraint is not satisfied, none of the base or the enhancement layer is rerouted.

Previously, only one metric besides the cost is considered. A flow may require more than one QoS metrics to be within specific values or constraints which makes the path computation process more complicated and time-consuming. In [66] authors built their work based on the work in *VSDN* [44]. They added reliability as an extra

constraint besides those previously used (i.e., bandwidth, delay, and jitter) in *VSDN* for video traffic path computation. Therefore, links are labeled with these four metrics and reliability has high priority in path selection. They argue that they modeled the problem as Multiple Constrained Shortest Path (MCSP) and used A*Prune [67] algorithm for solving it. They showed that *VSDN* could support a low number of satisfied requests when a high-reliability value is requested compared to *RVSDN*. They did not show in their work mathematical computation of the cost metrics, or the optimization process. However, this work is interesting in which it adds reliability as a constraint for path selection.

In [31], the authors define a deterministic network model to estimate the maximum delay of the network and the delays of link's queues are estimated. For admitting any new flow on a specific path, they compare with the network parameters. For each flow, a service class reveals its QoS requirements (i.e.max delay, rate). For instance, to admit the addition of a new flow to an existing path, the maximum delay required by its class is compared with the current path delay. Thus, for other demands such the rate parameter or burst value for instance. The problem is modeled as Multi-Constrained Shortest Path and solved as a mixed integer program.

The authors in [68] proposed a way to maximize the aggregated QoE for all uses and services flows. They used session configuration information received from a SIP server to informs the controller of those sessions will be established. The most interesting, they estimate the Mean Opinion Square (MOS) value of each of the traffic types (i.e., audio, video, data) using the loss QoS. However, they used end-to-end loss and the delay QoS to compute the MOS value for audio traffic. They used queuing

model (M/M/1/K) to compute the average delay and loss probability for each node. Their work shows significant performance (i.e., cumulated MOS) over the shortest path when the number of flow requests is increased. They modeled the routing as Multi-Constrained Shortest Path problem, and they used Integer linear programming implemented by IBM Optimization Language to solve it.

Multi-Constrained Path (MCP)

In some cases, it is not necessary to find the shortest path, but the chosen path should just cope the certain constraints to fulfill QoS requirements.

In [69] authors assume that each flow has QoS requirements represented as a vector of delay, loss rate, and bandwidth values. The QoS module uses a QoS-ware routing to find the best fit path that satisfies those metrics. The routing problem is a modeled a multi-constrained path problem. They proposed an algorithm to find the best QoS path by using a simulated annealing technique. The algorithm starts by finding the shortest path. A cost value is computed as the sum of weighted satisfaction ratios of each required demands. It adjusts the weight value by the miss rate of that QoS demand. It means higher weight for demands that less satisfied in the past. It tries to balance the satisfaction of those demands when choosing future routes. The algorithm continues to find a neighbor path(s) for $T = 16$ iterations (i.e., the stop threshold). The candidate neighbor path replaces the shortest path if its cost is smaller or the simulated annealing probability of the costs difference is high. It makes the chosen path does not always stick with the least cost local path (i.e., they use probability to avoid local optima). Simulated annealing depends on the design of the probability

function and the number of iteration used to abort the search process.

In [70], the authors consider critical flows' routing in real-time systems. Such flows have tight delay requirements in which packets received after a deadline is useless. Critical flows are prioritized based on their sensitivity to the delay requirements. For such path assignments, each flow demands two QoS metrics, delay, and bandwidth. They formulate the routing problem as Multi-Constrain Path (MCP) in which the path of a critical flow should cope with delay and bandwidth demands or constraints. To find a solution, they used relaxation in which one of the two constraints is relaxed at a time to find a heuristic solution. For instance, the proposed algorithm relaxes the bandwidth constraint and to find a path that complies with the delay constraint and vice versa. If both attempts could not find a solution, the MCP has no solution and the algorithm return no solution is found.

AI-based Routing Optimization

In [76], a QoS-aware adaptive routing (QAR) is proposed based on Reinforcement Learning (RL). RL is an ML-based technique that uses information from the system to re-adapt its behavior. An RL-based agent receives the state of the system, and a reward/punishment based on its actions. It tries to increase the long-term reward/revenue by exploiting its previous actions while exploring new ones. Authors designed a QoS-aware reward function as a tunable composite of QoS-based metric(i.e., link and queue delays, link loss, bandwidth) functions. According to, a value close to one is preferable while the harmful effect of an action (i.e., on other switches operation) gives negative values decreases the reward value. A quality function keeps

Table 2.1: QoS routing In SDN

Ref	Objective	Constraint	Formulation	QoS traffic	Controller	Domain	Network	Control	SDN Plane
[58]	Minimize weighted sum of the route length and the packet loss	loss of QoS traffic l	CSP	SVC video	NOX	Single	Campus	Single	Control
[59]	Minimize weighted sum of packet loss measure and delay variation	delay variation	CSP with LARAC	SVC Video	–	Multi	–	Single	Control
[60]	Minimize weighted sum of the route length and the packet loss	path length	CSP	Video	LEMON Simulation	Single	–	Single	Control
[61]	Minimize weighted sum of loss and delay variation	delay variations	CSP with LARAC	Video streaming	RYU	Single/ Multi	Cloud	Single	Control
[63]	Minimize sum of congestion and delay	delay variations	CSP with LARAC	Video streaming	Floodlight	Single	–	Single	Control
[57]	Minimize weighted sum of loss and delay	delay	DCLC	Video conferencing	Floodlight	Single	Campus	Single	Application
[51]	Minimize the delay	bandwidth	-	Video streaming	POX	Single	Campus	Single	Control
[71]	weighted cost of delay and loss	delay	CSP with LARAC	SVC base layer	LEMON Simulation	Multi	–	Single/ Multiple	Control
[72]	loss ratio	delay	CSP with LARAC	SVC video	OpenDaylight	Single	Enterprise	Single	Control
[73]	Minimize delay	delay variation	CSP with LARAC	Video streams	Floodlight	Single	Campus	Single	Control
[74]	delay	bandwidth and max (H) hops	CSP	delay sensitive	RYU	Single	Enterprise	Single	Control
[75]	per-hop cost	delay	DCLP	–	Simulation	Single	Campus	Single	Control
[64]	–	delay	DCLC with kLAM	–	Simulation	Single	Campus	Single	Control
[66]	Reliability	Reliability, bandwidth, delay	MCP with A*	Video streaming	–	Single	Campus	Single	Control
[68]	maximize MOS		MCSP	Video, Voice, Data	Simulation	Single	Campus	Single	Application
[69]	Weighted sum of the satisfaction ratios	delay, loss, bandwidth	MCP with SA, T=16	NA	Floodlight	Single	–	Single	Control

assessing the expected rewards received from actions within the current state. For a decision to be taken (i.e., chose next hop), the most quality returning action is chosen. The algorithm keeps updating its quality knowledge-base (i.e., state-action $\rightarrow$ reward table-like) using past chosen actions. This approach tries to find a feasible path according to the flow QoS requirements. Deep Reenforcement Learning (DRL) is investigated for routing optimization in the SDN-based network by the authors in [77]. A trained DRL-based agent is used to provide a one-step routing configuration. Moreover, DRL is used in [78] to regenerate link weights based on network state(i.e., Traffic Matrices or load). Thus, based on the trained DRL-based agent, the computed paths optimize the network delay.

In [79] a neural network-based routing is proposed, called *NeuRoute*. *NeuRoute* uses Deep Feed Forward Neural Network (DFFNN) to learn and match demands with routes. It collects outputs or path decisions history from an already running heuristic-based routing algorithm (i.e., for learning), combines them with the network state and the predicted traffic matrix (i.e., as in [80]) as an input to train its neural routing network. The time it takes to collect samples or to train the neural network is a performance decision. The designer should make that decision. The network state consists of all links costs and their available capacities at time t. Once the trained model is ready, it can be used to route new arriving flows. Therefore, based on the traffic matrix and the network state as an input to the model, it can produce an exact route as an output.

A genetic-based routing algorithm for enhancing video streaming in SDN networks (i.e., called GA-SDN) is proposed in [81]. Video flows and the level of network links

utilization (i.e., uses ports stat) are inspected periodically. Whenever link congestion(i.e., threshold-based) is likely to happen, *GA-SDN* tries to find a better path for video flows. *GA-SDN* reassembles the network graph of the path as a chromosome. The path cost equals to the fitness function of the chromosome. The fitness function takes one of two values: zero if not feasible, or $1/path(cost)$, which means increasing the path cost will decrease the fitness value. The proposed approach shows better Peak Signal to Noise Ratio (PSNR) and throughput performance values compared with the Bellman-Ford [82] routing algorithm.

The authors in [83] introduced a QoE-centric routing. They assess the effect of network condition (i.e., delay and loss rate) in the QoE values. QoE requirements are passed to the controller, i.e., northbound API. According to those demands, they formulate the routing as a multi-constrained problem. They used an Ant-Colony meta-heuristic algorithm to find a solution. The algorithm tries to maximize the flow(s) QoE value (e.g., MOS) and accommodating flows constraints.

Table 2.1 shows a summary of reviewed routing works been done in SDN for QoS provisioning. We did not show the shortest path problems due to space limitation.

2.2.2 Resource Allocation

It is necessary to manage network resources in a way that maximizes its utilization. Competition between services on those limited resources complicates such task. SDN shows potential to resolve such issue with its dynamic programmability which allows network owner adapts to new requirements efficiently. Resource allocation usually are jointly discussed with routing, however, due to specific characteristics in resources

reservation (e.g. admission control), QoS guarantee, and their effect of resource utilization we separate this module.

Single vs Group

In this section we differentiate in how the resource allocation module serves flows. A single approach that flows are served as first come first service without considering next upcoming requests versus group-based processing. For instance, authors in [51] modified Dijkstra [46] routing algorithm to finds the shortest path that has sufficient bandwidth for the QoS flow. Network traffic is classified into two classes, a priority traffic which require bandwidth guarantee (e.g., video traffic) and the rest as a best effort traffic. QoS requests that cannot be satisfied are rejected and the admission control module is supposed to notify end users through north-bound APIs. To avoid degradation of best effort traffic, they used link congestion threshold (i.e.,80% of link utilization). So that routing best effort traffic will avoid overused links. In [84], the resource allocation module searches the network graph for the shortest widest path for bandwidth requests and similarly did the authors in [52]. However, in [52], the admission control module first finds the widest path and compares its available bandwidth by the demand of incoming flow. If it is less than what is feasible for that flow, then user demand is downgraded to the current available bandwidth. Otherwise, it uses the flow's requested bandwidth. Based on the chosen demand value, a run of the shortest path algorithm is performed to find the path which can offer that bandwidth demand. In general, the approach finds the shortest bandwidth feasible path, however, it performs two search processes for each bandwidth request. In[85],

authors propose an optimization of bandwidth allocation in a hybrid SDN network. In such a network, few nodes are controller by the controller. The system contains flow managers at the edge of the network. The flow manager is responsible for path allocation requests and maintains the available network bandwidth, flow demand, and current flow-path bandwidth assignments. To increase network resource utilization, the authors adopt a multi-pathing approach. Therefore, the path computation module computes a set of paths to every flow at the controller if its demand is not satisfied by only one path. Then, it returns the computation output to the flow manager that either accepts the obtained solution or requests a reallocation if demands are still not satisfied. This process of flow manager and path computation opportunistically occurs when a new request or violation of bandwidth. In [54] , the proposed approach proactively, and only upon changes on network topology, computes and caches a set of paths between network pairs. When a new QoS request arrives, the routing module first checks if an existing path can satisfy the bandwidth demand. Then, the feasible shortest path is chosen. If no cached bandwidth sufficient path exists, the shortest path algorithm is used to search the network resources residual graph. In which links below the bandwidth demand are filtered out. If such a path could not be found, the request is rejected. Due to the fact that bandwidth is a concave QoS metric, most of bandwidth provisioning QoS frameworks do a prepossessing on network graph by removing unsatisfying links to produces a new residual resources graph. As its the case in [54][64] [48][86] [87][88][89], the required bandwidth is assured from residual resources graph. In [88][89], authors used ECMP similar approach for fast admission and load balancing. If the initial chosen path can's meet the QoS requirements,

the QoS-aware routing uses Dijkstra over network residual resources graph to find an alternative path with adequate bandwidth and reroute flow. Similar to [88][89], authors in [90] implement a weighted load balancing similar to ECMP at the network edge. Load balancing is implemented using OpenFlow SELECT group tables. Three traffic classes are defined, and fore each a group table is created at ingress switch with the appropriate entries and load weights. For fast admission, request is accepted if current configuration can handle its QoS requirement (i.e., only traffic class and QoS, cost is not considered). Authors assume the knowledge of a set of static paths or tunnels between each network pairs(i.e., ingress and egress switches). Therefore, a class traffic between each pair is splitted over those tunnels for load balancing. Since path cost is not considered at admission, authors proposed to periodically update those load-balancing weights. They try to optimize the overall cost while constrained with a budget of reconfiguration rate. It seems they adopted a hybrid approach since the SDN resides only at the edge of the network for admission and load-balancing reconfiguration, and maybe MPLS-based tunnel is used in core network.

One problem of flow-per-flow resource allocation that it does not respect traffic classes prioritization since low priority traffic may receive more resources due to early arrival. Which it may leads to extra control effort to accommodate future demands. For instance, authors in [24] obligate the SDN traffic management do resources evacuation when new demands are received and it detects a degradation of the network performance. In which, low priority flows are evacuated to another paths and use their share for high priority flows. Such changes will cause services interruption which affects users satisfaction and increase SDN control load. Besides classes prioritization,

resource allocation fairness is another issue. For instance within the same traffic class, bandwidth may not fairly allocated between the same class flows. Authors in [91][92] introduced a different approach for resource allocation targeting fairness. They propose to use a max throughput or resource bucket for each application. A new flow is scheduled only if the max throughput is not reached for that application. If not, a flow is inserted into the flow table within the switch. In order to achieve that, the SDN controller needs to keep track of a lot of information about each flow and each application. And even it try to control fairness between different application, still mice flows may starve in the competition with elephant ones.

To alleviate problems such as resource allocation unfairness, traffic prioritization and to efficiently utilize network resources, part of the state of art processes flows in a group or batch.

Authors in[93] proposed a Software-Defined WAN (SWAN) resource allocation scheme for inter-Data Centers traffic. The algorithm try to efficiently, and fairly share network resource between different priority traffic classes. In their architecture, a broker submits bandwidth request in behave of internal data center services. Then, the SDN controller perform the allocation process among set of flow requests with different priority classes. The algorithm first start with the highest priority class flows to allocate bandwidth. Within each priority class, the algorithm performs two procedures, first it try to allocate bandwidth for flows in a way that maximize the overall throughput of current class flows. Then, it iteratively try to reassign flows bandwidth to maximize fairness between flows within the same class. To control computation time, they define a max parameter T to stop the fairness approximation

which was set to 10 iterations in their experiments. Within each t iteration, they try to allocate b_i bandwidth value (i,e., they assume the knowledge of all shortest paths between network pairs) for the current flow. The value of b_i is bounded within interval $[b_{low}, min(d_i, b_{high})]$. Parameters b_{low}, b_{high} are passed to the allocation procedure to incrementally increase the assigned b_i value within each iteration. Parameter d_i is the requested bandwidth demand which will be bounded with to avoid over provisioning flows. Satisfied flows (i.e., $b_i = d_i$) are not entering the next round or iteration since they already receive what they need. By doing that, authors try to approximate max-min fairness between flows within the same class and limit its execution by $T = 10$ to avoid excessive computation time. According to this algorithm behavior, all flows are treated with soft QoS service level, since flows may be accepted even if they receive a small portion of the required bandwidth. Moreover, within their approach and to increase network utilization, authors allows proportional bandwidth assignment in which the flow demand can be assured by multiple paths. Splitting flow traffic over multiple paths causes packets out of order problem due to the unavoidable delay variance between used paths, which requires extra effort at the end edge of communicating services. Assigned resources are subtracted from current network graph that will be used to allocate resources for lower priority class in next class iteration. At the end, services are informed with the amount of rate or bandwidth each assigned to comply with it for the next time interval (i.e., 5 minute interval is used).

Authors in[94] proposed a resource allocation for value-added video services framework in SDN. In their proposed framework, users have the ability to request and knows multiple service levels differs than basic services. Users can pay extra money for guar-

anteed quality video streaming services. Such model also helps the service providers to increase their revenue from such users. Therefore, and differently than in[93], within such system model two types of service levels coexists, Assured and Improved Quality levels (AQ , IQ). Flows within AQ class accepted only when request bandwidth is guaranteed by the network (i.e., AQ requires hard QoS, while IQ receives soft QoS). To solve the resource allocation, authors use group batch flow admission (e.g., every 100ms) instead of processing requests as they arrive. Before the optimization process start, flows are sorted by their service level (i.e., AQ class has higher priority), and within each service level or class by required bandwidth. Flows with small demands are prioritized within the same class similarly as in [93] to avoid mice flows starvation. Similarly to previous work, shortest path algorithm is used over residual resources graph to obtain flow-path resource allocation. If there is no path found for AQ flow, the request is rejected. However, in case of IQ flows, the requested bandwidth demand d_i is decremented by a step size δ value within each iteration until a path is found or the flow is rejected. For fairness allocation of resources between IQ flows, flows that share the same source network switch are scaled their demand d_i down by a factor w^*. That factor computed as the ratio of the sum of all bandwidth demands of that set of flows over the switch remaining capacity.

Authors in [95] proposed a Multi-Class Traffic Engineering for QoS-guaranteed (MCTEQ) inter-DC communication. The multi-class traffic is managed through a central server which is the SDN controller. A bandwidth broker periodically (e.g., every 5 minute) assess the need for communication rate changes. So if any changes occur within that time interval or when a new flows arrive, the broker sends band-

width requests toward the controller. Then, based on output of the bandwidth allocation module, old tunnels or paths are torn down and new ones are created for flows group(FG). Similarly to SWAN, traffic of each FG can be splitted over set of paths or tunnels each one provide a portion of total FG bandwidth demand. Authors used Yen[96] algorithm which is a k shortest path algorithm (i.e., a max k =6) to obtain the set of candidate paths between communicating data centers pair. Differently than in SWAN, class prioritization is not reserved through strict order, however, they used a weighted class utility log function, $log(1 + \text{bandwidth allocated})$. They give a large weight value for the highest class priority traffic (i.e., 1000 compared to 50 and 1 for other two classes). By the usage of utility based bandwidth allocation, authors try to approximate proportional fairness between competing flows within the same class. Their work benefits from the soft treatment of traffic priority classes in the form of an increase in the total throughput received from low priority large volume flows compared with those high priority traffic classes.

Authors in [32] proposed Service and Resource Aware Flow Management (SRAFM) scheme for multi-flow management over a campus SDN network. Similar to previous works, resources are allocated for a batch of flows that are optimized together. However, delay-sensitive services such as interactive flows are treated proactively where resources for such services are allocated in advance. Similar to GCSP[94], mice flows are prioritized over bandwidth starving flows. They identify the type of the flow at the user side and use Differentiated Services Code Point (DSCP) tagging to identify at the control layer. SRAFM try to minimize overall system cost. Therefore, for each flow, selected path should minimized the combined weighted function of operational

cost (i.e., proportional to link load) and loss rate, and comply with the QoS constrains such as the required bandwidth. Two weight parameters, α, β, are used to reflect the importance of either metric on the overall system cost. In order to do that, authors used Lagrangian relaxation in which the constraints are Incorporated into the cost function. Therefore, a new Lagrangian cost function is defined in which a λ_e variable is defined for each link or graph edge. Beside the Lagrangian multiplier, authors defined another variable δ, which reflect the ratio of consequent rate assignments and used as a penalty for path cost value to avoid congested paths. Therefore, the main procedure of SRAFM is the path computation module that finds (i.e., for each flow) a path that minimizes the Lagrangian cost. At end of each iteration, and similar to work in [94], a single path and rate value is assigned to each flow. Then, values of λ_e links Lagrangian multipliers are update, and used in the next round of rate assignment process. The role of Lagrangian multiplier is direct the path search process either to rate or cost oriented. This procedure stop if the difference between two subsequent λ is less an ϵ_1 value. According to the authors, this procedure is the main part of SRAFM which try to find sub-optimal path and rate flow assignment. It iteratively called after δ value is update which perform the role of adjusting the behavior of the main procedure. SRAFM stops when it find a subsequent path rate-flows assignments are less than an ϵ_2 value. According to SRAFM algorithm behavior, its computation time depending on configuration of ϵ_1, ϵ_2 values. Therefore, it should be carefully chosen to avoid high computation overhead even in small scale network. Moreover, authors did not mention how initial values for network links Lagrangian multipliers values is chosen.

Authors in [97] propose, SWAY, a QoS-aware routing for SDN-based Internet of Things(IoT) traffic. They classify traffic into two classes, delay and loss sensitive. Therefore, for each class, they designed a cost function consider the corresponding QoS metric and minimize the number of activated switches. For each flow, and based on the corresponding class cost function, the algorithm search for K sorted least cost paths using Yen algorithm [96]. Then, the least path that comply with QoS requirement (i.e., bandwidth) is selected. Similar to the work in [94], SWAY uses single path routing and flows processed based on explicit prioritization. However, due to the usage of K-shortest path, its performance will suffer high computation time especially in large scale networks .

Other Optimization Approaches

We need to mention here, that there are other forms of optimizing allocated resource. For instance, the authors in [98] formulated the online virtual links resource allocation as an Integer Linear Problem. In which requests for virtual links arrive sequentially to the system with specific requirements such as bandwidth. The objective of the system is to distribute the network traffic with appropriate network resources utilization taken into consideration. The output of the algorithm is a route(s) and set of flow table rules that should be installed on the physical nodes or switches to support the creation of virtual link(s). They argue that the distribution of virtual links creation among network node increase the admission rate because nodes and links are reasonably utilized.

Similarly, the work in [99] formulated the problem of virtual link creation on top

of physical network as a Multi-objective Integer Programming (MIP). However, each virtual link request is associated with a QoS class that is defined within the QoS profile (i.e., specific attributes with absolute values). They applied three general link sharing policies when creating a virtual link over the physical link. Full sharing policy allows the physical link to be shared with as many virtual link requests with no constraint on the QoS class that request belongs to unless the max link capacity is reached. The second policy is called *full split* in which each the QoS class can have a maximum share of the physical link. It is for the equitable utilization of the physical resource between different QoS class. In this policy, different QoS class can share the physical link. The last one called the Russian Dolls Model in which the physical link is shared in a hierarchal way between the different QoS classes. This policy support priority between classes in bandwidth sharing which like ToS definition with lower values means higher priority. This division of QoS class and its constraints on creating virtual links can be considered such as allocating specific virtual links for certain types of traffic to satisfies its requirement. It is necessary due to the burst nature or unpredicted behavior traffic of those services or applications.

The authors in [100] modified the NUM (Network Utility Maximization) framework to support multi-class services. The objective of the NUM framework is to maximize the sum of the network flows utilities within the limitation of links capacities. They used two different utility functions for both elastic and inelastic flows. Each flow's utility function has a priority parameter to reflect the QoS class for that flow. For a new flow, and after routing path is computed, the controller uses a sub-gradient projection algorithm to estimate or allocate the rate of that flow. The only considered

constraint here is the link capacity. For scalability purpose, the authors assume usage of multiple controllers. They used *Opnet* to evaluate their work.

The authors in [101] [102] leverage traffic prediction for QoS-aware resource reallocation. They proposed ways to minimize the packet loss ratio while maintaining the delay and the bandwidth under control. They model the problem as Binary Linear Programming and propose two schemes to solve it. An exact optimal solution (QRTP) which solve the problem using CVX toolbox in Matlab. However, to reduce the optimization time, they made two relaxations on the original solution. The relaxed scheme is called *RQRTP*. First, an aggregation of flows is performed. Instead of granular-based flow optimization, they join flows that share the same source, destination and have a total bandwidth less than a certain threshold (i.e., forwarding table compression). Then, an upper bound for the objective function is computed to minimize the total link utilization. The approach computes the link utilization as the sum of the flows traversing that link. The resource reallocation process is ignited only when congestion is detected/predicted, or the time for the periodic resources optimization is reached.

Another interesting subject is the resource sharing (i.e., auction/pricing). For instance authors in [103] exploited *FlowVisor*[104] and virtual network slicing to present an auction-based resource allocation. *FlowVisor* is a network virtualization framework, that makes a layer between control and data planes which intercept the communication in between. Authors assumed existence of multiple controllers with no interaction between them. Each controller managed and control a slice or a virtual topology defined by *FlowVisor*. Therefore, flows belonging to a specific controller'

user(s) follows a certain set of virtual links. The problem they address is that controller may need extra resources to support incoming or existing flows of its users. *FlowVisor* performs the role of auctioneer, and interesting controllers behave like bidders. Due to its knowledge of the status of the network and existing network slices, *FlowVisor* behave as a proxy for all controllers in which it receives bids from each one and release shared resources to an interested controller. However, each controller will pay the cost to get those resources and the *FlowVisor* makes the auction decision. Authors here assumed the existence of multiple controllers, and even it may be not always applicable. However, this approach can be used to allocate resource for users sharing the network resource. Each user wants extra resources must pay the extra cost, and it may get those resources from a similar user that has underutilized private/reserved resources.

2.3 QoS Policy Enforcement

The work in [51] utilized weighted metric to compute the utilization of end-to-end paths (ingress-egress) and use this metric for calculating routes for QoS traffic. The exciting part is that when the bandwidth requirement for the QoS cannot be achieved, a resource evacuation is performed. The approach starts gradually removing the best effort flows from that path to allocate extra bandwidth for the QoS flow(s). It starts with a single best effort flow, if it was not enough multiple flows are rerouted. For resource allocation, the controller configures outgoing ports along the chosen path to handle the QoS flow requirements (e.g., max or min rates). The controller uses

OF-Config protocol [?] for this task since it is beyond the scope of the OpenFlow protocol. Authors assume the existence of northbound APIs for users QoS requests, however, the QoS flows are predefined within a file that is checked by the controller dynamically.

Similarly, the work in [105] uses link utilization as a cost metric for path selection for QoS flows. Whenever a routing decision cannot meet QoS requirement, queuing techniques are used to enforce prioritization of the QoS flow to meet its requirements. They reserve bandwidth on queues along the path the QoS flow will use. Authors ignore the overhead of monitoring the status of link utilization when they are polling statistic from the network element. This issue is vital since the accuracy of the network affects routing decisions and it is a trade-off between overhead and the accuracy (i.e., actually they did not show how statistics is polled).

In [29], the authors proposed an approach for network resource allocation based on different requirements of multiple services or applications. Their design consists of four modules: a traffic classification at the edge of the network (i.e., DPI-based software for getting flow information and send it to the controller), a queuing module, status collection (i.e., utilization of queues), and path calculation. After the controller knows which service group a flow belongs to (i.e., based on the information received from the DPI classifier), the queuing module configures queues within the network switches to support specific requirements for each service (i.e., bandwidth and delay are the QoS metrics). Therefore, the controller maps each flow to the queue(s) that will satisfy its requirement depending on the service type that flow belongs. For path calculation, they modeled that problem as constrained shortest path (CSP) and used

the LARAC algorithm to solve it. They optimize the path for each group of services based on their most important QoS metric (i.e., VoIP sensitive to delay). Classification at the network edge is like what is used in today networks. However, usage of deep packet inspection may consume more resources at the edge switch. Moreover, instead of tagging the packets, they send the information to the controller.

In [61], the authors leverage SDN to support resources allocation for cloud users. They utilized QoS route optimization combined with queuing to ensure resource allocation for high priority users. They divided the flows into QoS, and best of effort flows using ToS field in the IP header. Both flow types can share the same path(s) when the network is not congested. They poll statistic from network switched to monitor the network state. Optimized LARAC-based routing is used to compute a path for QoS flows based on their requirements. Where there is no feasible path found, they do queues manipulation through the Open vSwitch Database Management Protocol (OVSDB)[**?**] protocol to enforce the resource allocation. Authors ignored the communication overhead between the controller and switches while statistics is collected or when rerouting is needed.

The authors in [106] propose SDN-based real-time dynamic traffic shaping and bandwidth allocation for clients in home networks. When a client requests a video streaming, controller intercepts the DASH video streaming communication to get meta-data such as the video length and the bit-rate. Then, a QoE optimizer module tries to find the optimal allocation of bandwidth between clients to maximize the bit-rate each receive and equitably share the bandwidth between client as it is specified by network policy (i.e., they use equal bandwidth). Based on that, a traffic shaping

module enforces the output of the optimizer into rules installed by the controller at the home network gateway.

In [107], the authors used OpenFlow meter table for multimedia bandwidth assurance. They used the meter table entries to do metering for flows, while traffic is classified based on the incoming ports and the IP destinations. If link's congestion is detected a rerouting is performed to relieve that link. The controller periodically collects statistic from the SDN switches. They predict a bandwidth value for each flow and compare it with the corresponding real-time bit-rate reported from the SDN switch. If the predicted value less than the real value, a multiplicative increase is performed, else an additive decrease is performed to release part of the reserved bandwidth. It looks likes the congestion avoidance mechanism used by TCP (AIMD) but reversed. Links 'congestion can be detected from collected statistics. Authors used a rerouting algorithm that distributes low priority flows traffic along others link. The number of the packet forwarded to each of the new routes depends on the amount of available bandwidth that link has. They used OpenFlow group table to perform this multi-pathing. This work depends mainly on two things, the detection of congestion in the network which depends heavily on statistic collection which means higher accuracy requires a higher rate of statistic packets either pushed or polled from switches which may add extra overhead. The other thing is the prediction equation used for bandwidth reservation. From the results, they showed that they predict the peak traffic, but between peaks, degradation is slow which waste resources reserved for multimedia traffic.

The authors in [108] derived a model for evaluating the performance of SDN ar-

chitecture. They also proposed the usage of two queues; one has a high priority for packets from the SDN controller, and another low priority queue for other packets. The purpose of the high priority queue is to reduce the delay when flow miss-match happen for packets. They provide higher priority for those packets coming from the SDN controller. Their purpose was to maximize the fairness between packets with little scarification of performance (packet loss). They evaluated their approach using mathematical model and simulations. The results show lower delay for packets that suffer from flow miss match better than the traditional First In First Out (FIFO) scheduling. This work is like what existing in current traditional network elements by providing high priority for important packets such as Voice over IP (VoIP). Therefore, the authors did not change much but used this feature to help early flow's packets that suffer delay when a decision from the controller is required.

The authors in [109] propose a queue migration scheme. They used a model for predicting the delay of queues. If the queue delay is high and expected to violate the deadline constraint, packets are migrated to different queues.

In [110], the authors did network slicing and provisioning for these slices to meet application requirements — the controller reserve resources for customers based on high-level slices specification defined by the administrator for customers or services. The slices can be applied on aggregated or granular flows. They proposed schemes to solve the problem of meeting high-level requirements with low-level configurations. They configure rate limiter at network edge (i.e., bandwidth usage policies) and priority queues at network nodes (i.e., for bandwidth and delay reservation). This procedure is similarly done in current network architecture, but manually. They leverage

SDN controller to automate these tasks. They still do manual configuration of the application flows classification. Therefore, a dynamic detection (e.g., application profiling) will improve this work and reduce errors.

2.4 Analysis and Adaption

Along the network lifetime, a continuous process for monitoring network performance is mostly exist. Detection of network QoS violation is an essential part of QoS provisioning. For instance, the approach used in [111] monitors link utilization and produces a table of links utilization as an output of the monitoring process. That table is used as an input to the analysis module to detects congested links. A simple threshold-based (above 70%) detection is used. Link congestion can lead to service degradation. Therefore an action is performed to resolve that link contention by rerouting long-lived flows to other non-congested links. Moreover, massive bandwidth consumption needs to be detected due to the criticality of network resources to the QoS support. Elephant flows consume bandwidth and fill switches buffers which are a problem for a short-life flows (i.e., called mice flows) or delay sensitive ones as in VoIP flows. Therefore, detection and redistribution of them along the network or taking the appropriate action defined by the QoS policy (e.g., rate-limiting). The authors in [112] focus on visualizing such flows for the network operator. In [113], the authors put more attention on the adaption of the configuration of detection or decision thresholds. The authors in [114] used a learning model to detect heavy hitters. Heavy hitter consumes a large portion of network capacity that could be to service

nature or suspicious users. A predefined mark list by the network administrator of known heavy hitter is provided to the model in conjunction with flows statistics. Two threshold values , $\triangle_1 < \triangle_2$, is used to compare flows rate with history values. If flows rate is above $\triangle_2$, it is considered as a heavy hitter and added to the predefined list. If the rate between the threshold values, it is considered suspicious and more focus put on that flow. Feedback is reported to the monitoring unit to increase the sampling interval or install more fine-granular monitoring rules for that flow. Flows can be removed from the list if their behavior becomes normal, in which negative feedback is observed, and the flow is removed. Similarly authors in [115], [116], [117], [118], [119], [120], [121], [122] proposed schemes to detect and mitigate elephant flows. The reader can refer to [123] for more information about elephant flow detection methods.

The authors in [124] proposed a predictive model for monitoring SLA maintenance for cloud tenant applications. They used historical data collected from the flows of the application and use it as an input to a model that compute two values: the first one is the mean throughput and the second is a scaled score for application behavior. A forecast engine receives these values as input. It periodically monitors the flows of the application and uses the inputs from the model to compute a metric value for volatility detection. If this value above a certain threshold the rate of the application is limited.

Author in [125] [126] utilize ML in SLA management in NFV and SDN. They proposed an SLA enforcement architecture. It consists of thee parts, the data collector(from virtual machines, functions, and switches), data processing (i.e., preparation and feature extraction) and the smart engine which, based on the collected and pre-

processed data, can predict violation in SLA. Prediction depends on a forecasting module that takes the processed data and tries to predict or expect future values. Then, the prediction module can identify which Service Level Object (SLO) expected to be violated. Then, It issues a warning message to the administration layer, and the SLA enforcement module is notified to do proper management action such as allocating more resources.

In [55], the proposed approach monitors the link bandwidth utilization. Upon detection of link congestion (i.e., utilization over a certain threshold), the routing module is invoked to find alternative paths that satisfy flows bandwidth requirements. The shortest path with available bandwidth is chosen as the new route to resolve such a situation.

In [58], the authors propose an architecture to support QoS video flows in SDN that differentiated between two layers in the SVC encoded video. A base layer that should receive a particular QoS in which packet loss is forbidden and an enhancement layer that is classified as the best effort traffic. The approach directs the base layer traffic to non-congested routes due to its primary effect on the video quality. The process is as follows: the streaming server contacts the SDN controller with a QoS request message, the SDN controller configure the path of the QoS traffic according to the QoS agreed contract and return a QoS id for the streaming server which starts streaming. Upon reception of queues statistics or network error, the SDN detect that the path is congested so it cannot handle QoS traffic. Therefore, the rerouting algorithm works to find another path with the appropriate capacity even if it is longer than the shortest path (i.e., the best effort always uses the shortest path) but with a

delay or length that is tolerable by the streaming application or less max value. After that, flow modification is sent to nodes to configure the new path. The result in their work shows that when a new UDP traffic is entered the network, and by rerouting, the QoS traffic does not suffer much. The model formulated by the authors consider packet loss as the primary constraint for path selection in which a zero loss should be achieved. In this approach may not be applicable for another type of traffic such as VoIP which is tolerant toward packets lost and sensitive for delay or jitter.

In [61] authors define two types of traffic, a QoS traffic, and the best effort. They primarily used the shortest path for both traffic. However, when congestion is detected, and the requirement for QoS flow cannot be satisfied, a routing optimization algorithm starts working to find a path that satisfies the QoS metric requirement. They use the delay variation as the metric. They modeled the problem as Constrained Shortest Path and used LARAC [62] algorithm to solve it. The algorithm tries to minimize the cost function while keeping path's delay below a maximum constraint value. The cost function calculated as a weighted sum of link delay variation and packet loss. Similarly, in [63], used predefined 70% utilization threshold of the link capacity. They consider the link is congested if its utilization is above that value. The authors in [127] proposed schemes to offer guaranteed network resources for cloud users. They mainly focus on the implementation of their approach to be in the OpenVSwitch by exploiting multi-path support. They define a QoS metric consist of the sum of the ratios, the minimum values required by the user over the real measured ones. They belong to three metrics that are defined in the SLA which are the bandwidth, the delay, and the number of hops (i.e., path length). Therefore, whenever any of these

values do not meet what defined in the SLA, an alternative path is chosen for the QoS-flow. They define two types of flow, QoS-flow, and best effort. Even if the authors used OpenFlow compliant switches, their approach was not clear about the role of the SDN controller. Also, this highly dynamic change of the route due to nonacceptance of one of the three metrics they consider will cause high fluctuation of the QoS traffic between different flows. Moreover, they do not consider queue occupation in their metric which may lead to loss of packets.

In[89] the QoS framework monitor network links usage, if a used link is congested (i.e., 90% link bw utilization) and it been used for QoS routing, a local routing change algorithm is used to fix that path. It first start with evacuating flows with smaller bandwidth demand and change their forwarding path to a new one that guarantee required QoS requirements.

In summary, in this Chapter, we review the state-of-art in the life cycle of QoS provisioning. Mainly, flows classification (i.e., ML, DPI, or user-defined) is used to identify flows'QoS demands with a significant part that used controller-defined API to specify QoS requirements. QoS-aware routing and resource allocation is the primary process of any QoS framework in SDN. Despite many proposed research, we notice there is an ignorance of computation overhead in the central control of SDN. In the analysis module, many reviewed research approaches use simple threshold-based analysis of the QoS performance. Moreover, we notice that most of the research uses static policies or configurations. The modification only occurs once, maybe due to the specific scenarios applied in such researches. There are mainly two QoS implementation methods on the data plane: either traffic shaping through queues or traffic policing

through metering. SDN controller has full support on meter creation, modification, and control which is not the case with queues. The controller can only direct flows to already created and configured queues. Creation and modification of queuing disciplines or profiles still use vendor-specific protocols such as OVSDB[**?**]. Therefore, in this work, we focus our research effort to be on the control layer. To decouple it from the data plane SDN implementation and make it work in any SDN platform.

CHAPTER 3

MULTI-CLASS GROUP-BASED RESOURCE ALLOCATION AND REFINEMENT FOR QOS PROVISIONING IN SDN NETWORKS

The fundamental purpose of any QoS framework is to find an end-to-end path(s) that comply with the QoS flows demands and efficiently utilize network resources to increase the business revenue. Bandwidth is the primary QoS metric demanded by many network services and multimedia applications. In the literature, there are many proposals for developing and improving QoS frameworks in SDN [21] [22][23]. However, most requests are admitted as they are received, in the sense of first come,

first served(i.e., a flow per flow). This sequential treatment may hold in situations where there is auxiliary resource availability or some degree of traffic homogeneity, which is not the case in the current Internet. One problem of flow per-flow resource allocation is that it does not respect traffic class prioritization since low priority traffic may receive more resources due to their early arrival, leading to extra control effort by calling resources evacuation to accommodate future demands[24]. Moreover, resource allocation may not fairly be allocated between flows within the same class. A traffic engineering effort may help in solving post-admission problems by re-optimizing resource allocation[25]. However, such proposals may suffer from high computation overhead since they deal with many flows. Moreover, besides the messaging overhead, changing the current control configuration needs to be carefully performed since it can cause service interruption and lead to congestion in some network parts. This work adopts a different approach; we use group-based admission where a small group of flows is processed together in a short time interval. In [94], authors showed the advantage of this method over conventional flow-per-flow admission to improve business revenue where more valuable flows are prioritized over others. However, there are still two issues: the computation overhead due to the iterative flow-path bandwidth assignment search (i.e., resource allocation) process. Usually, a predefined step size up or down is used to reduce the search space. Our proposal adopts a similar step-down search process using small state information from previous iterations to cut down the search space as fast as possible. The second issue is that even group-based processing may reduce the sequential allocation effect; still, there is some sequentiality in flow processing within the same traffic class. Therefore, a flow-path bandwidth allocation

decision in earlier iterations may affect the amount of resources feasible for later ones, affecting fairness and reducing network utilization. Therefore, we develop this idea of performing fast initial allocation and then do a post-allocation refinement process where a group of flows can scavenge unassigned residuals and fairly share and reassign resources in between.

3.1 Related Work

Part of the state of art processes flows in a group or batch to alleviate problems such as resource allocation unfairness, traffic prioritization, and efficiently utilizing network resources.

Authors in [93] proposed a Software-Defined WAN (SWAN) resource allocation scheme for inter-Data Centers traffic. The algorithm tries to efficiently and fairly share network resources between different priority traffic classes. In their architecture, a broker submits bandwidth requests in behave of internal data center services. Then, the SDN controller performs the allocation process among a set of flow requests with different priority classes. The algorithm first starts with the highest priority class flows to allocate bandwidth. Within each priority class, the algorithm performs two procedures. First, it tries to allocate bandwidth for flows to maximize the overall throughput of current class flows. Then, it iteratively tries to reallocate flow bandwidth to maximize fairness between flows within the same class. They define a max parameter T to stop the fairness approximation, which was set to 10 iterations in their experiments, to control computation time. Each t iteration tries to

allocate b_i bandwidth value (i,e., they assume the knowledge of all shortest paths between network pairs) for the current flow. The value of b_i is bounded within interval $[b_{low}, min(d_i, b_{high})]$. Parameters b_{low}, b_{high} are passed to the allocation procedure to incrementally increase the assigned b_i value within each iteration. Parameter d_i is the requested bandwidth demand which will be bounded with to avoid over-provisioning flows. Satisfied flows (i.e., $b_i = d_i$) are not entering the next round or iteration since they already receive what they need. By doing that, the authors try to approximate max-min fairness between flows within the same class and limit its execution by $T = 10$ to avoid excessive computation time. According to this algorithm, all flows are treated with a soft QoS service level since they may be accepted even if they receive a small portion of the required bandwidth. Moreover, within their approach and to increase network utilization, the authors allow proportional bandwidth assignment in which multiple paths can assure the flow demand. However, splitting flow traffic over multiple paths may cause packets to arrive out of order due to the unavoidable delay variance between used paths, which requires extra effort at the end edge of communicating services[128]. Therefore, assigned resources are subtracted from the current network graph in the next class iteration for the lower priority class. In the end, services are informed of the rate or bandwidth amount assigned to comply with it for the next interval (i.e., a 5-minute interval).

Authors in[94] proposed a resource allocation for value-added video services framework in SDN. In their proposed framework, users can request and know multiple service levels that differ from essential services. As a result, users can pay extra money for guaranteed quality video streaming services. Such a model also helps the service

providers to increase their revenue from such users. Therefore, and differently than in[93], within such system model, two types of service levels coexist, Assured and Improved Quality levels (AQ, IQ). Flows within the AQ class are accepted only when the network guarantees request bandwidth (i.e., AQ requires hard QoS, while IQ receives soft QoS). To solve the resource allocation, authors use group batch flow admission (e.g., every 100ms) instead of processing requests as they arrive. Furthermore, they proposed a heuristic procedure called Group Constrained-Shortest Path (GCSP) for optimization. Before the optimization process starts, a set of flows (i.e., newly arrived and may include existing flows) are sorted by their service level (i.e., AQ class has high priority) and within each service level class by the required bandwidth. Flows with small demands are prioritized within the same class similarly as in [93] to avoid mice flows starvation. Like previous work, the shortest path algorithm is used over the residual resources graph to obtain flow-path resource allocation. If there is no path found for AQ flow, the request is rejected. However, with IQ flows, the requested bandwidth demand d_i is decremented by a step size δ within each iteration until a path is found or the flow is rejected. For fairness allocation of resources between IQ flows, those that share the same source switch scale their demand d_i down by a factor w^*. Which is computed as the sum of all demands of that set of flows over the switch remaining capacity.

Authors in [95] proposed a Multi-Class Traffic Engineering for QoS-guaranteed (MCTEQ) inter-DC communication. The multi-class traffic is managed through a central server which is the SDN controller. A bandwidth broker periodically (e.g., every 5 minutes) assesses the need for communication rate changes. So if any changes

occur within that time interval or when new flows arrive, the broker sends bandwidth requests toward the controller. Then, based on the bandwidth allocation module's output, old tunnels or paths are torn down, and new ones are created for the flows group(FG). Like SWAN, each FG traffic can be split over a set of paths or tunnels. Each provides a portion of the total FG bandwidth demand. Authors used Yen[96] algorithm, which is a k shortest path algorithm (i.e., a max k =6) to obtain the set of candidate paths between communicating data centers pair. Unlike in SWAN, class prioritization is not reserved through strict order; however, they used a weighted class utility log function, $log(1 + \text{bandwidth allocated})$. As a result, they give an enormous weight value for the highest class priority traffic (i.e., 1000 compared to 50 and 1 for the other two classes). Using utility-based bandwidth allocation, authors approximate proportional fairness between competing flows within the same class. Their work benefits from the soft treatment of traffic priority classes in the form of an increase in the total throughput received from low priority large volume flows compared with those high priority traffic classes.

Authors in [32] proposed Service and Resource Aware Flow Management (SRAFM) scheme for multi-flow management over a campus SDN network. Similar to previous works, resources are allocated for a batch of flows that are optimized together. However, delay-sensitive services such as interactive flows are treated proactively where resources for such services are allocated in advance. Similar to GCSP[94], mice flows are prioritized over bandwidth starving flows. They identify the flow type at the user side and use Differentiated Services Code Point (DSCP) tagging to identify at the control layer. The proposed approach in SRAFM tries to minimize overall system cost.

Therefore, for each flow, the selected path should minimize the combined weighted function of operational cost (i.e., proportional to link load) and loss rate and comply with the QoS constraints such as the required bandwidth. Two weight parameters, α, β, are used to reflect the importance of either metric on the overall system cost. The authors used Lagrangian relaxation in which the constraints are incorporated into the cost function. Therefore, a new Lagrangian cost function is defined in which a λ_e variable is defined for each link or graph edge. Besides the Lagrangian multiplier, the authors defined another variable δ, reflecting the ratio of consequent rate assignments and used as a penalty for path cost value to avoid congested paths. Therefore, the main procedure of SRAFM is the path computation module that finds (i.e., for each flow) a path that minimizes the Lagrangian cost. At the end of each iteration, a single path and rate value is assigned to each flow. Then, values of λ_e links Lagrangian multipliers are updated and used in the next round of the rate assignment process. The Lagrangian multiplier role is to direct the path search process either to rate or cost-oriented. This procedure stops if the difference between two subsequent λ is less than an ϵ_1. According to the authors, this procedure is the central part of SRAFM, which tries to find sub-optimal path and rate flow assignment. Iteratively called after the δ value is updated, which performs the role of adjusting the main procedure's behavior. SRAFM stops when it finds a subsequent path rate-flows assignments are less than an ϵ_2 value. According to SRAFM algorithm behavior, its computation time depending on the configuration of ϵ_1, ϵ_2 values. Therefore, it should be carefully chosen to avoid high computation overhead, even in small-scale networks. Moreover, the authors did not mention how initial values for network links lagrangian multipliers are chosen.

Authors in [97] propose SWAY, a QoS-aware routing for SDN-based Internet of Things(IoT) traffic. They classify traffic into two classes, delay and loss-sensitive. Therefore, they designed a cost function for each class to consider the corresponding QoS metric and minimize the number of activated switches. For each flow, and based on the corresponding class cost function, the algorithm search for K sorted least-cost paths using Yen algorithm [96]. Then, the least path that complies with the QoS requirement (i.e., bandwidth) is selected. Similar to the work in [94], SWAY uses single path routing and flows processed based on explicit prioritization. However, due to K-shortest path usage, its performance will suffer high computation time, especially in large-scale networks. Table 3.1 shows summary of the related work.

In general, part of the proposals that consider group-based admission in the literature uses multi-path routing, which suffers the issue of out-of-order arrival of packets that need more processing and requires larger re-sequencing buffers at the end of communication entities[128]. Others do not strictly prioritize traffic classes and define weighted costs function to increase throughput from low priority large volume flows. Another issue is the computation overhead; some works ignored that issue which is a killer in central SDN network environments [129]. In this chapter, we consider all those issues, and we try to efficiently and fairly allocating resources between traffic flows while preserving class prioritization. We do that by using a fast resource allocation process that reduces the search space as fast as possible and refine allocated resources to increase fairness in the same traffic class.

Table 3.1: Related Work

Ref	Main Objective	Network	BW Broker	Path	Class Priority	SL
[51]	Satisfy QoS flows demand, and reduce congestion and delay	–	north-bound APIs	S	NA	Both
[54]	Maximize utilization of network resources	Nation-wide backbone network	–	S	NA	Hard
[84]	Increase resource utilization efficiency	–	–	S	NA	Soft
[85]	Increase network utility	–	Flow Manager at edge	M	NA	Soft
[90]	Optimize the overall cost and reduce configuration overhead	ISP Network	–	M	Strict	Hard
[93]	Increase network utilization	Inter-data center	Broker	M	Strict	Soft
[94]	Maximize network service provider's revenue	ISP network	Similar	S	Strict	Both
[95]	Maximize network utilization and maintain delay	inter-data center	Broker	M	Not Strict	Soft
[32]	Minimize overall system cost (includes loss ratio)	campus SDN network	–	S	Not strict	Both
[97]	Maximize the overall QoS performance for delay and loss-sensitive applications	SDN-based Internet of things network	–	S	Strict	Hard

3.2 Resource Allocation Framework

There are two perspectives in the resource allocation problem: one is from the user side and another from the service provider. The user always wishes to receive the best service quality, while the service provider's goal is to utilize available resources to increase revenue efficiently. Most service providers apply fair use on network capacity for users. The service provider will serve the user's flows to a possible extent; however, the service level may be degraded when the network is congested. In this work, we assume that the customer may have the ability to pay extra costs for special treatment and avoid the fair use policy (e.g., for a live football match or important video conference). With the limited network resources and high competition between concurrent services, the sequential flow-per-flow admission is unprofitable in such situations. Furthermore, it may cause some important requests rejected due to resources are allocated to less important ones early arrived. Therefore, a group-based flow processing in short time intervals may accommodate some important requests that may be denied otherwise.

3.2.1 System Model

In this work and similarly to research works in [93] [94], we differentiate and classify traffic into a set of l strictly prioritized classes, $QoS_1, ..., QoS_l$. In general, such classification is defined by a network operator, who can also define each class's service level, either hard or soft. In a hard state, the requested bandwidth should be guaranteed; otherwise, the request is rejected. In the soft state, the resource allocation framework

will try to accommodate such requests with possible bandwidth it can offer with fairness between those class flows taken into account. The resource allocation algorithm resides on top of an SDN network controller. A statistics and topology management module within the SDN controller maintains an up-to-date global view of the network. Given a network graph, $G(V, E)$ consists of V switches and E set of links connecting them. Each link e is initialized with a max capacity e_c that ongoing traffic cannot exceed. For each QoS traffic class flow,f_i, there is a corresponding bandwidth demand, d_i, and the resource allocation process has to find a path, p_i, from source s to destination t, that satisfy the following constraints:

$$0 < b_i \leq d_i \qquad (3.1$$

$$\sum_{f_i \in F_e} b_i \leq e_c \qquad (3.2$$

In which b_i is the allocated bandwidth for flow f_i that requests d_i demand. While in hard service level, the allocated bandwidth b_i should equal the demand d_i, otherwise the flow is rejected since the network cannot guarantee the required bandwidth. The constraint in 3.2 implies that, the total allocated bandwidth for all flows , F_e, passing link e should not exceeds its capacity, e_c.

The objective of the allocation framework is to maximize a total revenue value

TR, which calculate as :

$$TR = \sum_{f_i \in F} b_i * m_{SL} \qquad (3.3$$

m_{SL} is a monetary value that the network operator defines as per data unit cost (e.g., per megabit). It will be more for users requesting hard service levels since their traffic will be specially treated.

The main job of the resource allocation framework is to use network resources efficiently. In the sense of serving as much as possible of high priority first-class flows and reasonably and fairly sharing the remaining resources between low priority flows, with more concentration in achieving such goals with low computation overhead, a killer issue in central SDN environments. Therefore, in the next section, we propose our scheme to implement group-based flow admission to solve the bandwidth allocation problems.

3.3 Proposed Work

In this section, we propose a Multi-Class Group-based Resource allocation and Refinement (MCGRR) approach. In which it strictly prioritizes traffic classes, and a group of flows or requests are processed together. Algorithm 1 shows how MCGRR works. As input, it takes an up-to-date network graph, G. A group F of flows or requests to be processed, and their corresponding demands set D. Moreover, an operator-defined step-down demand downgrading value, Γ. Flows within F are classified into

Algorithm 1 Proposed MCGRR Algorithm

Input: G, F, D, Γ
Output: P, R
Initialization: set $FSDP = \{\phi\}$, set , $TD = getAverageDemand(D)$
for $SL = QoS_1, QoS_2, \ldots, QoS_l$ **do**
    set $SDP = \{\phi\}$, $R_{Ref} = \{\phi\}$, $F_{Ref} = \{\phi\}$
    for $f_i \in F_{SL}$ **do**
        $p_i = null$, $b_i = TD.get(f_i)$, $d_i = D.get(f_i)$, sdp_i = (src,dst) pair of f_i
        $skipFlag = false$
        if $FSDP.contains(sdp_i)$ and $FSDP.get(sdp_i) <= b_i$ **then**
            if $SL_{QoS} = Hard$ **then**
                $skipFlag = true$
            else
                if $FSDP.get(sdp_i) > \Gamma$ **then**
                    $b_i = max(0, FSDP.get(sdp_i) - \Gamma)$
                else
                    $skipFlag = true$
                end if
            end if
        end if
        if $skipFlag = false$ **then**
            if $SL_{QoS} = Hard$ **then**
                $RRG = getRRGraph(CopyOf(G), b_i)$
                $p_i = getShortestPath(RRG, sdp_i)$
            else
                while p_i is $null$ and $b_i > 0$ **do**
                    $RRG = getRRG(CopyOf(G), b_i)$
                    $p_i = getShortestPath(RRG, sdp_i)$
                    if p_i is $null$ **then**
                        if $notFSDP.contains(sdp_i)$ or $FSDP.get(sdp_i) > b_i$ **then**
                            $FSDP.put(sdp_i, b_i)$
                        end if
                        $b_i = max(0, b_i - \Gamma)$
                    end if
                end while
            end if
        end if

```
        if p_i is null or skipFlag = true then
            if notFSDP.contains(sdp_i) or FSDP.get(sdp_i) > b_i then
                FSDP.put(sdp_i, b_i)
            end if
            b_i = 0
            R.put(f_i, b_i)
        else
            P.put(f_i, p_i), R.put(f_i, b_i)
            UpdateGraphByDecrRes(G, p_i, b_i)
        end if
        if SL_QoS = Soft and b_i ≠ d_i then
            R_Ref.put(f_i, b_i), SDP(sdp_i).append(f_i), F_Ref.append(f_i)
        end if
    end for
    if SL_QoS = Soft then
        RefResAlloc(G, F_Ref, D, P, R, R_Ref, SDP)
    end if
end for
```

Table 3.2: MCGRR Mathematical Notations

Symbol	**Definition**
G	Nework Graph
F	Set of flows
D	Set of flows'demands
Γ	Step-down downgrading value
P	Set of ouput paths
f_i	Flow i
sdp_i	Source and destination pair of f_i
SDP	Set of sdp_i pairs
$FSDP$	Set of failed sdp_i pairs
TD	Set of temparary demands
R	Set of allocated bandwidth values
SL	Priority ordered traffic classes
F_{SL}	Sef of F flows shares same SL class
SL_{QoS}	Service level (Hard/Soft)
R_{ref}	Set of flows'bandwdith for refinement
F_{ref}	Set of flows to be refined
d_i	Request demand for f_i
b_i	Allocated bandwidth for f_i
p_i	Found path with b_i values for f_i
$EXBW$	Temparary extra available bandwidth
$friends$	Set of flows shares the same sdp_i

Algorithm 2 RefResAlloc Algorithm

1: **function** RefResAlloc($G, F_{Ref}, D, P, R, R_{Ref}, SDP$)
2: **for** $f_i \in F_{Ref}$ **do**
3: $b_i = R_{Ref}.get(f_i)$, $d_i = D.get(f_i)$, sdp_i = (src,dst) pair
4: **if** $b_i < d_i$ **then**
5: **if** $b_i \neq 0$ **then**
6: $p_i = P.get(f_i)$
7: $availBW = getAvailBW(G, p_i, (d_i - b_i))$
8: $R.put(f_i, (b_i + availBW))$
9: $UpdateGraphByDecrRes(G, p_i, availBW)$
10: **end if**
11: $R_{Ref}.remove(f_i)$
12: *continue*
13: **end if**
14: $EXBW = \{\phi\}, EXBW.put(f_i, (b_i - d_i))$,$R.put(f_i, d_i)$,$friends = SDP(sdp_i)$
15: **for** $f_j \in frineds$ **do**
16: **if** $f_j = f_i$ or $not R_{Ref}.contains(f_j)$ **then**
17: *continue*
18: **end if**
19: $b_j = R_{Ref}.get(f_j)$, $d_j = D.get(f_j)$, $p_i = P.get(f_i)$
20: **if** $b_j = 0$ **then**
21: share b_i on p_i with the rest of $friends$, update R and P
22: $R_{Ref}.remove(friends)$, $EXBW.remove(f_i)$
23: *break*;
24: **else**
25: $p_j = P.get(f_j)$
26: **if** $b_j > d_j$ **then**
27: **if** $p_i = p_j$ **then**
28: $extraBW = (b_j - d_j) + EXBW.get(f_i)$
29: $EXBW.put(f_i, extraBW)$
30: **else**
31: $EXBW.put(f_j, (b_j - d_j))$
32: **end if**
33: $R.put(f_j, d_j)$, $R_{Ref}.remove(f_j)$
34: **else**
35: **if** $p_i = p_j$ **then**
36: $R.put(f_i, d_i)$, $R_{Ref}.remove(f_i)$
37: $R.put(f_j, (b_j + EXBW.get(f_i))$, $R_{Ref}.put(f_j, (b_j + EXBW.get(f_i))$
38: $EXBW.remove(f_i)$
39: *break*
40: **else**

```
                    R.put(f_j, b_i), P.put(f_j, p_i), R_Ref.put(f_j, b_i)
                    if b_j > d_i then
                        R.put(f_i, d_i), P.put(f_i, p_j), EXBW.remove(f_i)
                        UpdateGraphIncrRes(G, p_j, (b_j − d_i))
                    else
                        R.put(f_i, b_j), P.put(f_i, p_j), EXBW.remove(f_i)
                    end if
                    R_ref.remove(f_i)
                    break
                end if
            end if
        end if
    end for
    for f_x ∈ EXBW do
        UpdateGraphIncrRes(G, P.get(f_x), EXBW.get(f_x))
    end for
  end for
end function
```

set of priority ordered classes $SL = \{QoS_1, ..., QoS_l\}$. F_{SL} is the set of flows in F belongs to the same class. Within each class group, F_{SL} flows are ordered by their demands. Flows with low demands are prioritized over others to protect mice flows from bandwidth starving elephant ones. Traffic class can request hard or soft service level, SL_{QoS}. At the hard service level, the request demand should be guaranteed; otherwise, it is rejected. Its left for the operator to define which traffic classes require soft or hard service level; however, we assume that the highest prioritized class would receive hard service's level. The output of MCGRR is two sets, P and R. R contains all processed F flows along with their allocated bandwidth. While P contains flows' found paths with the corresponding allocated bandwidth in R. Rejected flow receives zero allocation, and no path is assigned for it. Before the algorithm starts, it defines two sets, $FSDP$, and TD. The set $FSDP$ is used to keep state information at each iteration for failed allocation attempts between a specific source and destination

pairs,*sdp*. The set TD contains normalized demands. For flows that request a soft service level and have the same source and destination pairs, *sdp*, we modify each flow demand to be the average of all flows demands with the same *sdp*. Other flows (i.e., with hard service level or singleton *sdp*) demands are not affected and stays the same. To make earlier flows borrow resources for the next upcoming flows shares the same source and destination. Later on, allocated resources will be re-arranged between flows with the same *sdp* pair. In general, we can divide the process into three part: 1) search space reduction (Lines 9-19), 2) resource allocation(Lines 20-36, 3) resource refinement (Algorithm 2). The algorithm starts processing at line 4, and it loops through priority classes. It defines three empty sets, SDP, R_{ref}, F_{ref}, which will be used to store information about flows that will enter the refinement process. Set F_{SL} contains all flows with the same traffic class, ordered by their requested demands, d. For each processed flow f_i, variables p_i, b_i, d_i, sdp_i are defined and initialized. p_i is the network path, b_i is the initial bandwidth will be allocated for flow f_i. d_i is the original flow demand value, and sdp_i is the source and destination pair of flow f_i. To reduce the search time (line 9), it checks if a previous flow between the current pair sdp_i has been processed and failed at some point. If set $FSDP$ holds an entry for sdp_i and the corresponding failed bandwidth request was equal or smaller than current allocation request b_i. This means that b_i will fail, so we need to downgrade the requested allocation b_i value. However, if the service level is hard, we immediately skip (line 11) the flow since no resources are available. Otherwise, it downgraded the requested value if the previously failed values were bigger than step-size Γ (Lines 13-18). If the flow is not skipped, the algorithm allocates the required resource b_i

for that flow (Lines 20-36). It first generates residual resources graph RRG and then finds the shortest path between source and destination pair sdp_i. If it could found a path p_i, then the requested allocation, b_i, is satisfied. In the case of a flow with a hard service level, this process is performed once, while in a soft case, the process is iterative and keeps downgrading the value of b_i by Γ value (line 32). In soft state, and when the allocation process fails at value b_i, the algorithm store (or update previous larger values) that value with the corresponding current pair, sdp_i (Lines 29-31). This state information is the key to reducing computation time since this value will be examined in subsequent iterations (line 9). Based on the output of the allocation process or the search reduction(line 37), the results are stored at sets R and P. If the path could not be found or its processing is skipped due to previously failed allocation, the allocated is set to zero at set R, indicating flow rejection. Otherwise, the flow, the found path, and the current allocation b_i are stored at P, R respectively (line 44). Then, allocated resources are subtracted from the current graph, and its state is updated(line 45). Flows with the soft service level and unsatisfied demands,d_i, are saved for later refinement (Lines 47-49). At each class iteration, and if this class requires soft service level, flows that belong to this class and are not satisfied, F_{ref} are passed for further resource refinement(line 52). Where current graph G, along with current state stored at $D, P, R, F_{ref}, R_{ref}, SDP$ sets. Algorithm 2 show the resources refinement process. G is the current state of the network graph, and D contains flows demands. P are the set of found paths, and R are the set of current flows' allocation values. F_{ref} is the list of flows that need to be refined appear in the same order they have processed in Algorithm 1, R_{ref} is the set of currently allocated resources for flows will be refined,

and SDP contains a group of flows sharing the same source and destination pairs, *sdp*.

Algorithm 2 process only flows with the soft service level, and it tries to scavenge any unallocated resources (Lines 4- 13) and share resources between the group of flows with the same source and destination pair, *sdp* (Lines 14-53). Doing that tries to use resources to the possible extent and fairly shares resources between flows. Set R_{ref} contains two types of flows. Flows that are over-provisioned due to demand normalization in Algorithm 1, but it will share it with other flows in the same *sdp* that are not satisfied yet. The others are flows with allocated resources values, b_i, but their demands d_i are not satisfied. We need to mention here that if the current f_i flow is *sdp* singleton, its only available option is to scavenge unused resources along its path(Lines 5-10) and update its final allocation in set R. Then, it will be removed from R_{ref} (line 11), and the process skips to the next flow in F_{ref}. Due to the ordering of flows in $F_{ref}$and the normalized demand in Algorithm 1, if the current f_i flow belong to a group of the same *sdp*, thier allocated resources b_i will be either equal or more of its next-upcoming group friends. Therefore, in line 14, the algorithm defines a set $EXBW$ to hold extra resources borrowed in previous allocation iteration for subsequent *sdp* group if needed. Otherwise, the resources are released at the end of this loop iteration(Lines 54-56), in which other flows can scavenge later on. Set $friends$ contains all flows that share the same sdp_i pair with current flow f_i and preserve the same order they processed within Algorithm 1. Lines 15 to 52 perform the resource re-assignment or re-arrangement between the same sdp_i group. First, it checks if the current candidate flows f_j, is not the same as current flow f_i, and it still

needs refinement as it still exists within R_{ref}. Local variable, b_j, d_j, p_i is define and their current values are retrieved from R_{ref}, D, P sets (line 19). If current allocated resource of f_j is zero(line 20). It means that the current candidate flow and the reset of flows within $friends$ would receive the same value, and they are rejected in the first allocation process performed in Algorithm 1. Therefore, it shares the reserved bandwidth or resource of current flow f_i with reset of flows in the $friends$ group. Also, value stored at $EXBW$ is removed and all flows are removed from R_{ref}, and sets P, R are updated(Lines 21-23). However, if current candidate f_j received and amount b_j along path p_j, the algorithm checks the value of b_j and compare it with its demand d_j (line 26). If it has more than what it needs, then its state is updated, and extra resources are added to $EXBW$, and it is removed from the R_{ref} set(Lines 26-33). Otherwise, if the candidate flows f_j is still not satisfied, the algorithm checks if they share the same path as f_i, then it uses the extra resources hold by f_i along path p_i for flow f_j, and remove satisfied flow f_i from Set R_{ref}(Lines 35-39). The flow f_j will enter another round since it comes in a place after f_i in F_{ref} list. If both f_i and f_j does not share the same path, paths and resource are swapped between flows and sets R_{ref}, R and P are updated(Lines 41-49). Figure 3.1 shows a flow chart for the working process of MCGRR algorithm.

3.3.1 Example

To how MCGRR works in comparison of GCSP, explain a simple example using network graph in Figure 3.2. It shows a subset of an ISP network consist of nine switches connected with links each set labeled with the cur-

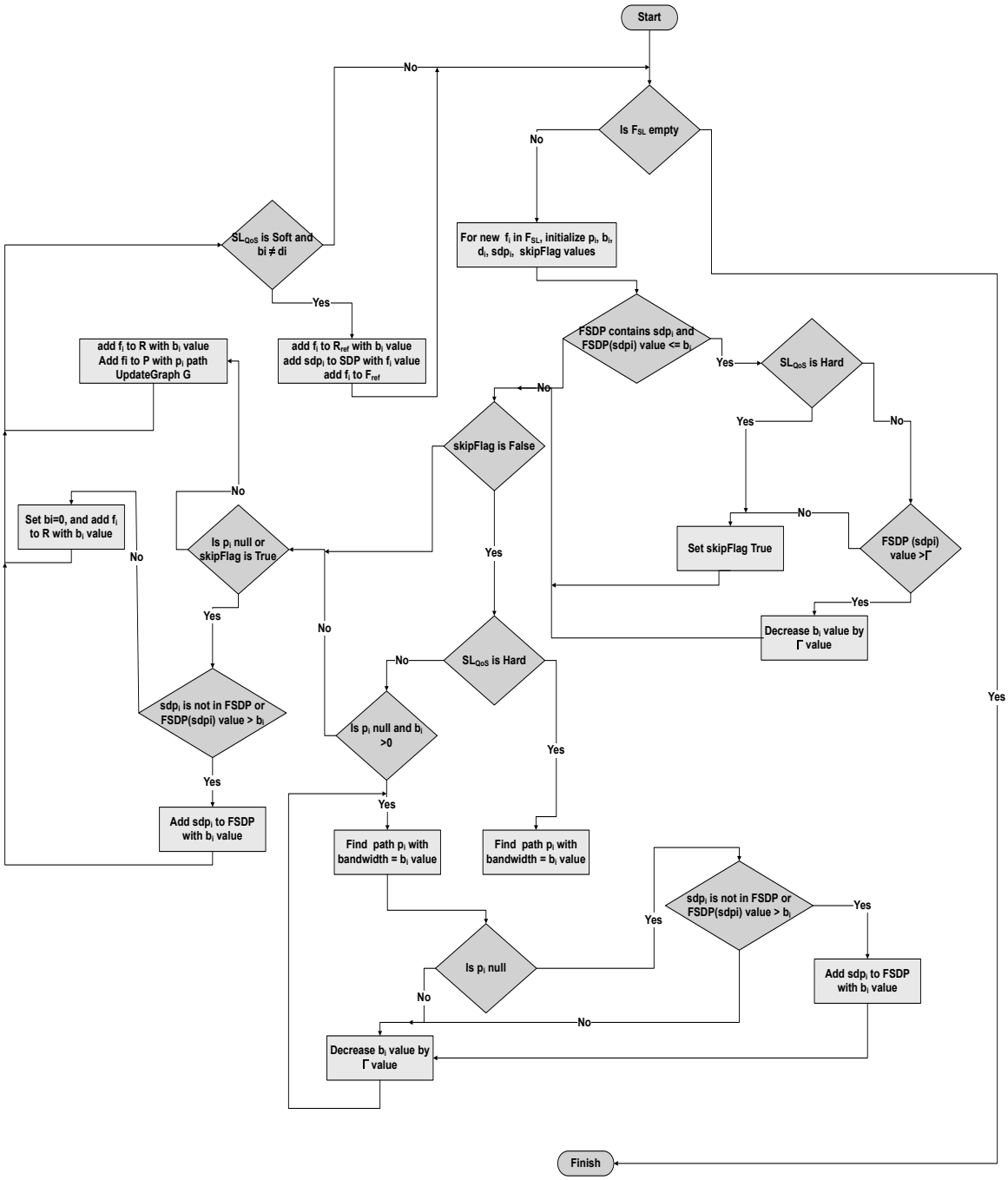


Figure 3.1: MCGRR Algorithm flow chart

rent available with capacity data unit (e.g., mbps). Assume a set of QoS_2 (i.e., both GCSP and MCGRR share the same behavior on QoS_1 flow) flows $F = \{f_1(1,7), f_2(2,8), f_3(3,9), f_4(2,8), f_5(3,9), f_6(1,7), f_7(2,8)\}$ with a a set $D = \{11, 12, 15, 20, 21, 24, 26\}$ of corresponding bandwidth demands data units. Both GCSP and MCGRR sort flows by their demand and use a step-size $\Gamma = 5$ data units.

We assume in this example that path cost is based on hop-count for simplicity. For instance, the shortest path from node 2 to node 8 is $2 \rightarrow 3 \rightarrow 5 \rightarrow 8$. GCSP will start with flow f_1 and ends up with flow f_7. According to GCSP, flows f_1 to f_5 will receive full demand allocation along paths $p_1 : 1 \rightarrow 3 \rightarrow 4 \rightarrow 7$, $p_2 : 2 \rightarrow 3 \rightarrow 5 \rightarrow 8$, $p_3 : 3 \rightarrow 5 \rightarrow 9$, $p_4 : 2 \rightarrow 3 \rightarrow 5 \rightarrow 8$, $p_.3 \rightarrow 4 \rightarrow 7 \rightarrow 9$. Due to decrease of resources, and after downgrading demands, flows f_6, f_7 receives 14, and 11 data units respectively on paths $p_6 : 1 \rightarrow 3 \rightarrow 4 \rightarrow 7$ and $p_7 : 2 \rightarrow 3 \rightarrow 5 \rightarrow 8$, with a total allocated bandwidth of 104 data units out of possible 110.

On the hand, MCGRR uses a different approach as in Algorithms 1 and 2. It first generate $TD = \{17, 19, 18, 19, 18, 17, 19\}$, normalized demands computed as the average of total demand between flows sharing the same source and destination. Starting the allocation at line 25, and storing information of initial allocation at data structures $P, R, R_{ref}, F_{ref}, SDP$. Flows initially allocated with values in $R, R_{ref} = \{17, 19, 18, 19, 18, 12, 4\}$ along paths: $p_1 : 1 \rightarrow 3 \rightarrow 4 \rightarrow 7$, $p_2 : 2 \rightarrow 3 \rightarrow 5 \rightarrow 8$, $p_3 : 3 \rightarrow 5 \rightarrow 9$, $p_4 : 2 \rightarrow 3 \rightarrow 5 \rightarrow 8$, $p_5 : 3 \rightarrow 4 \rightarrow 7 \rightarrow 9$, $p_6 : 1 \rightarrow 3 \rightarrow 4 \rightarrow 7$, $p_7 : 2 \rightarrow 3 \rightarrow 5 \rightarrow 9$.

These initial configuration is used along the execution of resource refinement in

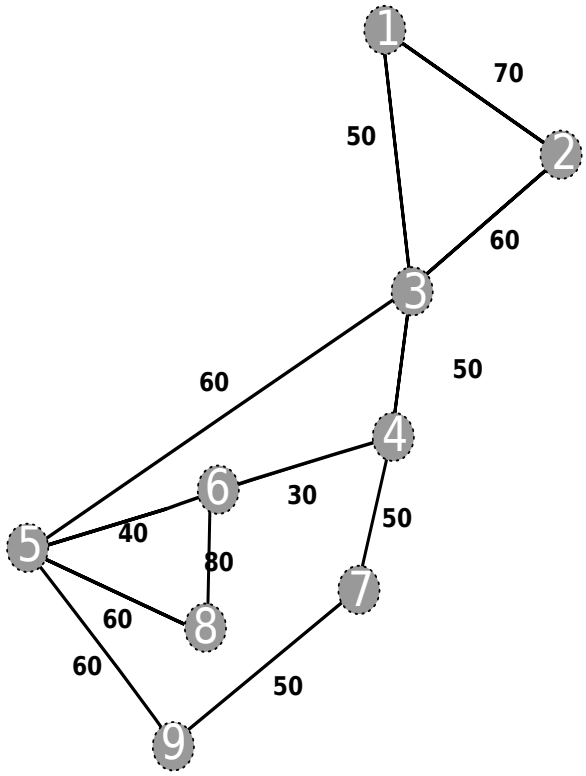


Figure 3.2: Subset of IP Sprint backbone network with link available bandwidth

Algorithm 2 to obtain the final output of allocation as in $R = \{11, 12, 15, 20, 18, 24, 10\}$ along paths in P: $p_1 : 1 \rightarrow 3 \rightarrow 4 \rightarrow 7$, $p_2 : 2 \rightarrow 3 \rightarrow 5 \rightarrow 8$, $p_3 : 3 \rightarrow 4 \rightarrow 7 \rightarrow 9$, $p_4 : 2 \rightarrow 3 \rightarrow 5 \rightarrow 8$, $p_5 : 3 \rightarrow 5 \rightarrow 9$, $p_6 : 1 \rightarrow 3 \rightarrow 4 \rightarrow 7$, $p_7 : 2 \rightarrow 3 \rightarrow 5 \rightarrow 9$. With total allocation equals 110 of 110 possible data units capacity. Before refinement process start, flows f_1 to f_3 are over-provisioned. An since flows in F_{ref} appears in the same order of F, flows f_1, and f_2 passes extra resources to their friend flows f_6 and f_4 respectively as in Algorithm 2, Lines 35-40. Flow f_4 then will be over-provision, so it will pass the extra 6 data units to flow f_7. Flow f_3 swaps allocation with its friends f_5 as in Lines 41-49. Since b_j value is bigger than d_i value (line 42), 3 data units are returned to the network path $3 \rightarrow 4 \rightarrow 7 \rightarrow 9$, which already has 3 data units left after the initial allocation not used. These 6 data units will be scavenged by flow f_6

Table 3.3: Experiment parameter

Parameter	**Value**
Network emulation	Mininet 2.3.0
Topology	IP Sprint Europe (12 nodes)
OS	Ubuntu 14.04
Controller	FloodLight SDN Controller
OpenFlow	Version 1.3
Γ	50kb
Link rate	1Mbps
Link cost	$\in$ [1,15]
Flow demand	$\in$ [100,300] kpbs
Traffic classes	Two classes, QoS_1=12%, and QoS_2=88%
SL_{QoS}	Hard for QoS_1 and Soft for QoS_2

when turn to refinement comes but its allocation is less its original demand. So it scavenge the possible bandwidth on its assigned path as in Lines 4-12, Algorithm 2. This achieves 100% utilization at this example, which will be reflected in the average utility each flow receives and the fairness between QoS_2 flows.

In the next section, we conduct experiments to test the performance of both MCGRR and GCSP as group-based flow admission and resource allocation algorithms.

3.4 Performance evaluation

In this section, we evaluate the performance of our proposed work.

3.4.1 Experiment Setup

For comparison, we implement our proposal, MCGRR, and the resource allocation algorithm proposed in [94] which is called GCSP. Both adopt the group-based flow admission in short time intervals. Moreover, we compare both algorithms with the

default bandwidth Constrained Shortest Path(CSP) algorithm. In CSP, only a single flow is processed instead of a group, as in MCGRR and GCSP. Like GCSP and MCGRR, CSP uses a step-size Γ (e.g., we use 50 kb) to downgrade the requested demand when initial allocation fails. Moreover, optimal allocation is approximated and reported by examining the possible allocations, and those with the best revenue outcome are reported along with other approaches. All routing and resource allocation algorithms are implemented on top of Floodlight SDN controller [45]. Floodlight is a Java-based SDN controller used by many QoS provisioning research works [23]. The underline SDN network data plane is emulated using Mininet [130]. OpenFlow v1.3 [131] is used as the southbound interface supported by Floodlight and Mininet virtual switches. We used a realistic Internet Service Provider (ISP) topology from Sprint [132]. Figure 3.3 shows the Sprint IP backbone network topology in Europe. Each node represents an OpenFlow enabled OpenVSwitch[133], with three attached hosts. Each link is configured with a rate of 1Mb/s a cost values are uniformly distributed within an interval $[1, 15]$ [62]. We use such a rate due to the processing limitation of the experimental setup. Traffic flows demand is uniformly distributed in the interval [100,300] kbps and generated using the iperf tool [134]. For comparison purposes with GCSP as in [94], we use only two classes, QoS_1, QoS_2, with hard and soft service levels, respectively. The first traffic class, QoS_1, represents about 12% of the total traffic, similar to the ratio of live video traffic reported in [2] compared to total Internet video traffic.

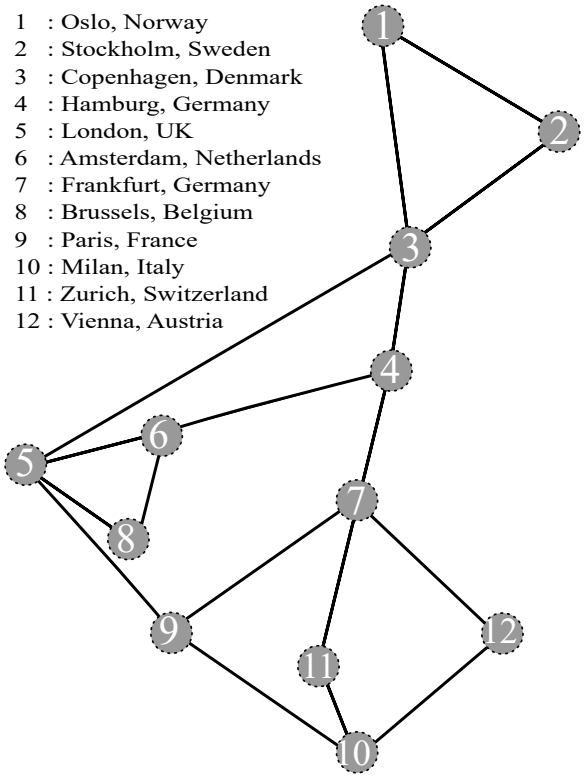


Figure 3.3: Topology from Sprint IP backbone Network

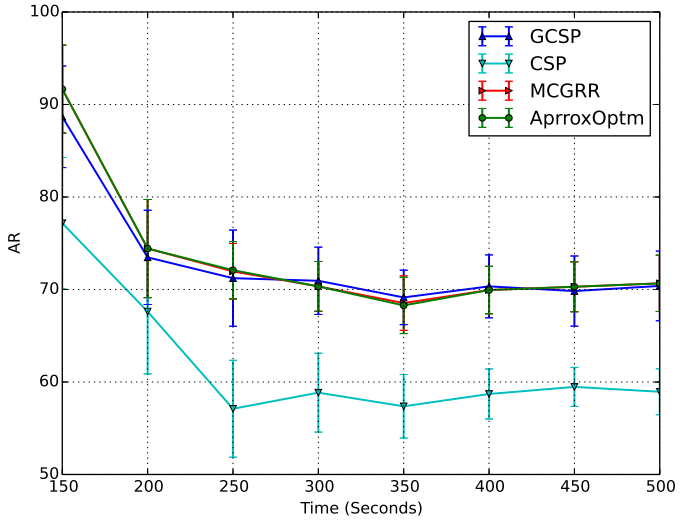


Figure 3.4: Admission ratio of QoS_1 traffic class flows

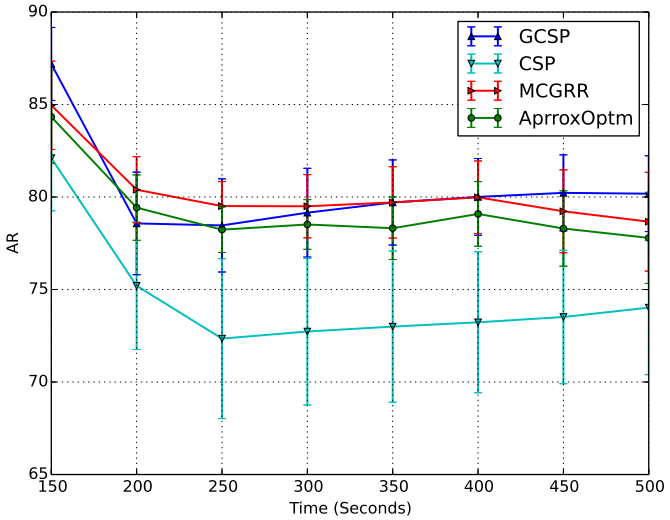


Figure 3.5: Admission ratio for all traffic classes flows

3.4.2 Results and Discussion

In this section, we present the obtained results and compare algorithms' behavior. We used five metrics to evaluate performance: the average admission ratio(AR), utility ratio(UR), fairness ratio (FS), total revenue (TR), and computation overhead of the resource allocation process. AR is calculated as the ratio of accepted requests over the total number of requests. UR defines the utility of each QoS_2 admitted flow received as the allocated bandwidth's allocation over requested demand. Moreover, FS defines fairness of resource allocation between QoS_2 flows. It is calculated as $100 * (1 - dev_{ur})$ where dev_{ur} is the deviation between utility values, ur, of admitted QoS_2 flows. TR represents the collected revenue from each method according to equation 3.3. Computation overhead computed as the count of shortest path algorithm calls over residual resource graph per accepted flow. Presented results are obtained over an average of 10 runs of experiments. Each experiment duration is 500 seconds, in

which traffic starts after the first 100 seconds. We used these 100 seconds to gradually starts the SDN controller, then the network.

Admission Ratio(AR)

Figures 3.4 and 3.5 shows the average admission ratio of QoS_1 class flows and the whole traffic flows respectively. As we mentioned before, from the services provider perspective, to increase the service revenue, the network should serve as many as possible of those QoS_1 valuable flows. Figure 3.4 shows the advantage of using group-based flow processing, as it is clear that both MCGRR and GCSP outperform the flow-per-flow behavior in CSP, as it clear in the ratio of admitted QoS_1 flows. Both MCGRR and GCSP treat QoS_1 flows similarly. They are prioritized over the other class due to their strict hard service level needs and importance to the business revenue. As expected, at the initial experiment time when network load is low(<200 seconds), the admission rate is high for all methods. When the load is increased, there is a sharp decrease in the ratio of admitted flows for all methods. After that, each method tries to adapt its behavior. In MCGRR and GCSP, mice flows are prioritized, and since they are ordered according to, they are processed first compared to CSP, where a flow with a great demand may be admitted earlier, leading to more rejection of flows. However, the behavior of MCGRR achieves about 1% lower AR compared with GCSP, which is the effect of QoS_2 flows that represent 88% of the total traffic. The main reason is that GCSP uses a scaling factor, which sometimes harshly decreases the ratio each flow should receive, decreasing the required demands that are later reflected on the number of admitted flows. The effect of this scaling factor appears

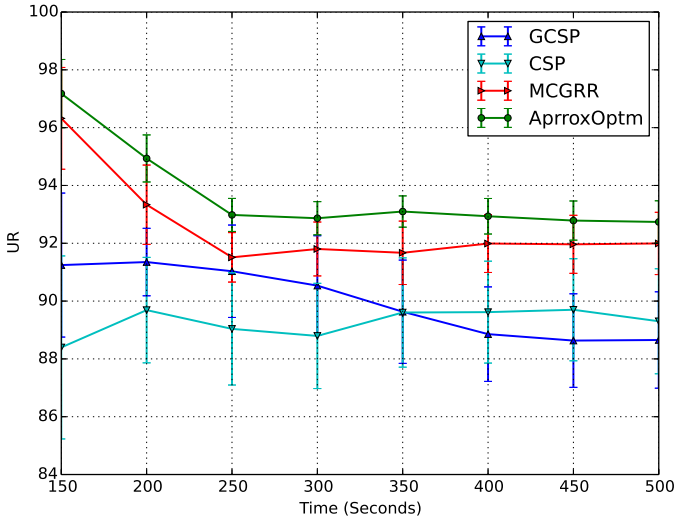


Figure 3.6: Average QoS_2 flow utility ratio, computed as the ratio of allocated bandwidth over demand

clearly at the utility ratio each flow receives.

Utility Ratio(UR) and Fairness(FS)

Figure 3.6 shows the average utility ratio per accepted flows. It indicates the number of resources each accepted flow receives. Both MCGRR and GCSP try to utilize the available resources in a better way than in CSP. The mice flows are prioritized, which increases the utility ratio each receives even when the resource is limited. In GCSP, the behavior of increasing the AR for QoS_2 (see Figure 3.5) affects the utility ratio of GCSP as it decreases drastically even compared to CSP. Because GCSP scarifies the amount of resources assigned to elephant QoS_2 flows, not the admission rate, and tries to hold the best admission ratio of QoS_1 flows. CSP does not efficiently utilize the resources, and even it does not achieve high AR values of QoS_1, which appear in fairness values as in Figure 3.7. It keeps similar behavior in a margin of

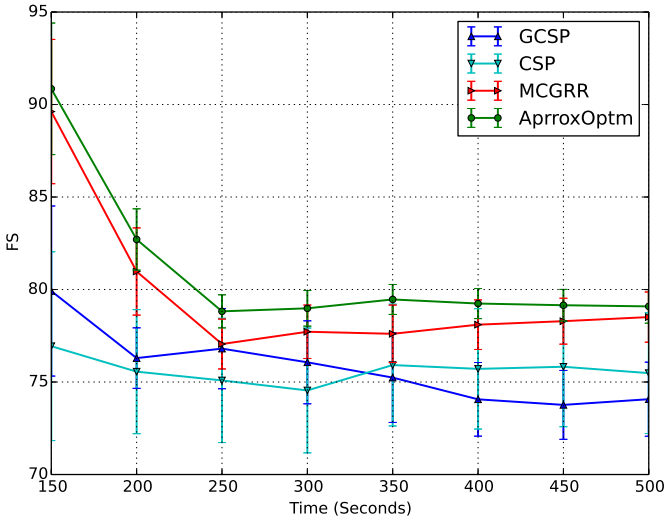


Figure 3.7: Average of resource allocation fairness score between QoS_2 flows

1% utility ratio up and down. The best behavior was received from MCGRR due to two reasons. First, MCGRR normalizes the demand of a group of flows sharing the same *sdp* at the beginning to save some resources for the last flows on the list. This means that both mice and elephant flow will receive a reasonable, proportional amount of resources compared to their demand after the resources are re-arranged in the refinement process. We also noticed that this method helps utilizing resources when refinement in Algorithm 2 takes effect. The second reason is the use of the step-down Γ parameter. We utilized a value in similar proportion to the value used in [94] compared to traffic rate (i.e., 1/6 max flow demand, 1/2 lowest flow demand). Downgrading the demand with a configured step size is inevitable in any resource allocation to reduce the computation overhead. Small values will improve resource utilization but increase computation overhead, a killer issue in major SDN networks. We alleviate the effect of less resource utilization caused by the step-size or any other reason by allowing unsatisfied flows to scavenge as much as possible still unused bandwidth along found

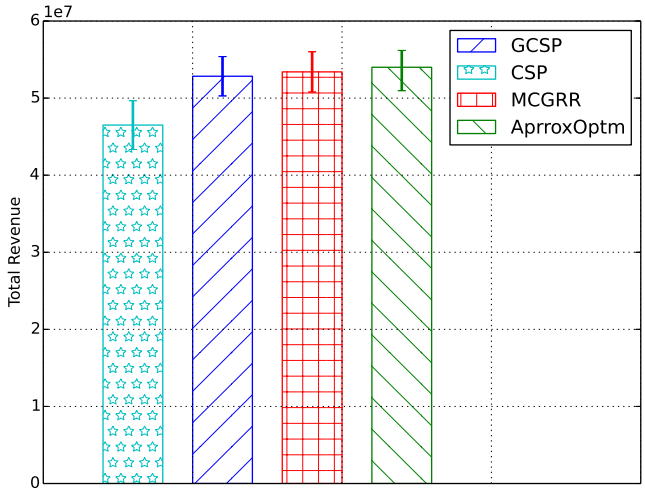


Figure 3.8: Total revenue collected by each method, computed as in equation 3.3

paths shown in Algorithm 2 (Lines 5 - 10). Figure 3.7 shows fairness between QoS_2 flows for all methods. Fairness is affected by the deviation of utility values, degradation in utility values recorded by GCSP at the second half of Figure 3.6 reflected on GCSP fairness. As the utility ratio decreases and the admission increase, the gap between mice and elephant flows utility ratio increases. GCSP tries to enforce fairness by using the scaling factor, which is not enough. This was not the case in MCGRR, since it makes mice flow reserve or borrow some resource in advance for elephant flows. Then, in the refinement process, make these flows share their additional bandwidth or swap paths with the large flow. Scavenging resources that are missed due to downgrading operation made MCGRR uses available resources to the possible extent reflected on both utility and fairness ratios compared to both GCSP and CSP.

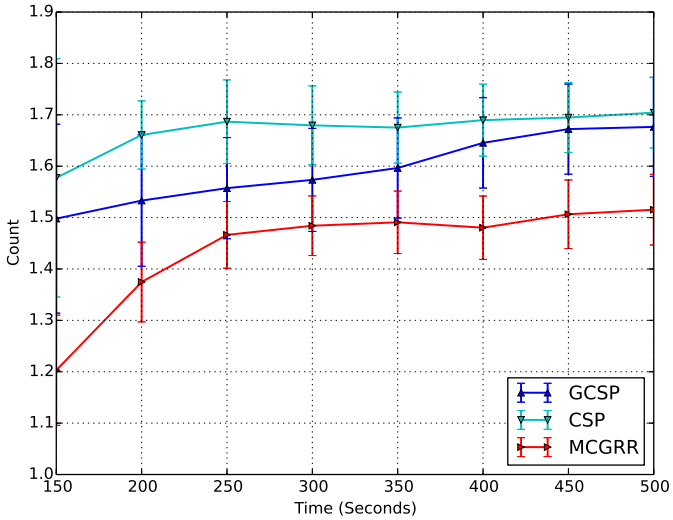
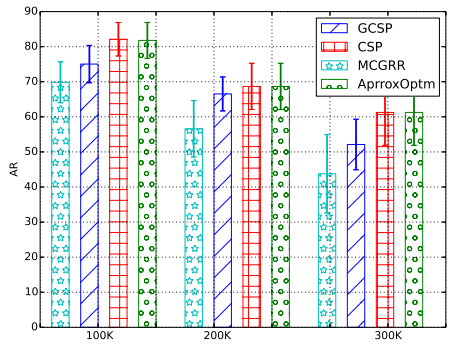


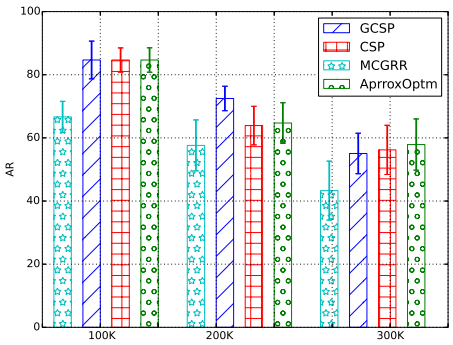
Figure 3.9: Computation overhead computed as the count of shortest path algorithm calls over residual resource graph per flow

Total Revenue(TR)

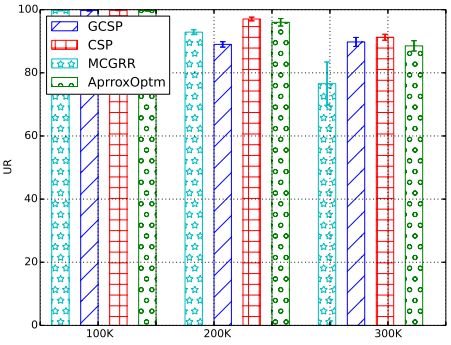
Figure 3.8 shows the total revenue obtained at the end of experiments time. Equation 3.3 is used to compute the total revenue value, where m_{SL} value was set as been used in [94], 1.8 and 1 for QoS_1 and QoS_2 respectively. In general, MCGRR and GCSP outperform CSP due to their high delivery ratio of QoS_1 flows, indicating the advantage of group-based processing. It makes the resource allocation module able to prioritize the most profitable flows. Even GCSP has a slightly higher admission ratio of QoS_2 flows than MCGRR; its obtained revenue is a little low with almost 1.13%. This is due to its behavior that sharply degrades the flows demand to increase the admission ratio, which affects obtained utility ratio as in Figure 3.6. Total revenue is mainly affected by both AR and UR metrics. Therefore, the resource allocation module needs to balance in between.



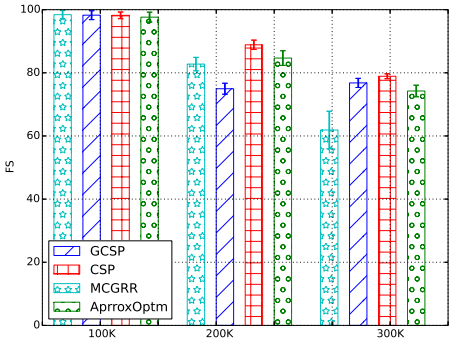
(a) AR of QoS_1



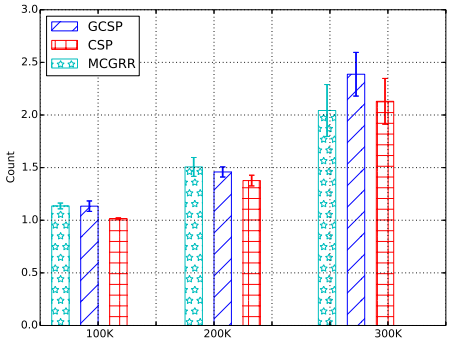
(b) Admission ratio for all



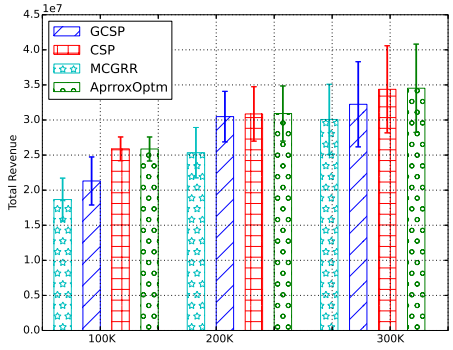
(c) UR



(d) FS



(e) Count



(f) TR

Figure 3.10: Performance with different network load

Computation Count

Figure 3.9 shows the computation overhead. A single resource allocation attempt performs two tasks. First, it generates a residual resource graph from the original graph according to the chosen allocation value b_i. Then, it searches for the shortest path from source to destination, sdp_i. Figure 3.9 shows the average count of those attempts consumed for each admitted flow by all three methods. In general, the behavior of all methods consists of the relation of resource availability.

The count values increase as more resources are consumed, making the resource allocation method more time to find a path with available b_i resources. This mainly applies to QoS_2 flows since the admission of QoS_1 is binary, and a single attempt is enough to decide whether to accept or reject the flow. Furthermore, the scaling factor's effect is more activated with group processing since degrading will be applied over all the flows at once, while it only will be applied at late-arriving flows at CSP. So degrading the demand is reducing the number of resource allocation calls within GCSP against CSP. MCGRR outperforms both GCSP and CSP due to its mechanism in reducing search space as in Algorithm 1, Lines 9-19. In addition, it stores information from previous resource allocation attempts, which reduces computational overhead compared to both GCSP and CSP with reductions up to 10% and 13%, respectively.

Figure 3.10 shows obtained results for the same set of metrics, but with different traffic loads (i.e., flow demand). All approach's performance is affected by the increase in traffic load (i.e., decreases as the higher load is used). However, the performance is consistent with the obtained results in previous experiments. CSP achieves the worst

performance, while MCGRR achieves results almost similar to the approximated optimal solution. However, when the workload is the highest, the computation count increases because MCGRR tries to increase the acceptance ratio of the second QoS class by fairly sharing resources between flows. The computation count for the approximated optimal solution is not shown since it is expected to consume much time compared to the existing approach.

In summary, the obtained results showed improvement up to a 10% reduction in computation overhead and an improvement of 4%, 6% in per-flow utility, fairness ratios, and 1.13% in total service provider revenue compared to GCSP.

CHAPTER 4

BOOTSTRAPPED LARAC ALGORITHM FOR FAST DELAY-SENSITIVE QOS PROVISIONING IN SDN NETWORKS

Software-Defined Networking (SDN) is a fast emerging paradigm that offers abstraction, programmability, and greater central network control flexibility compared to traditional networking. Moreover, it promises to provide services with a QoS guarantee while efficiently utilizing the network resources and maintain QoS consistency[135]. More specifically, for killer services such as audio and video streaming, online gaming [136], or for critical delay-sensitive applications as in medical (e.g., Telesurgery)[137],

or industrial environments [138] [139].

In the architecture of SDN, the controller maintains an abstract view of the network and contains a module that provides flow routing service. Other functions may be implemented on top of the SDN controller (e.g., Anomaly Detection, Load Balancing) in plug-and-play fashion in coexistence with QoS functions benefiting from north-bound interfaces [140]. Most SDN controllers implement Dijkstra [46] as their default for flow routing. For QoS provisioning, Dijkstra's performance depends on how the cost function is defined [141]. For instance, QoS requirements may be merged in a single weighted sum cost function. Moreover, to balance the satisfaction of QoS future demands, weights are dynamically adjusted [56][69]. However, this usually does not lead to routes that satisfy all needs. Minimizing a path cost while maintaining specific QoS metrics (e.g., delay, jitter, bandwidth) within demands is known to be an NP-hard problem[142][143].

The authors in [1] surveyed and evaluated the running time and cost efficiency of 26 unicast QoS-aware routing algorithms for SDN. More specifically, those that solve the delay constrained the least cost (DCLC) problem. They evaluated algorithms' performance against different criteria such as the topology type, size for scalability, and the delay constraint. They found that an algorithm called LARAC (LAgrange Relaxation based Aggregated Cost) [62] had the best performance among them. The basic idea of LARAC is to use the Lagrangian relaxation of the delay constraint along with Dijkstra to approximate or search for the optimal path. LARAC is used in many works for QoS provisioning in SDN. For instance, authors in [144] used LARAC for delay constrained SDN-based smart grid communication network. Authors in

[71][72] used LARAC to find routes for QoS video traffic (i.e., for video streaming). Also, authors in [145] used LARAC for QoS support in SDN and Network Function Virtualization (NFV). They adapt LARAC to route flows through Service Function Chain (SFC) in which the returned path should traverse a given set of nodes (i.e., Virtual Network Functions (VNF)) and maintain the required QoS constraints in the SDN network.

The controller is responsible for processing routing flows according to SDN architecture, which becomes more challenging when QoS guarantees are required. Therefore, the routing algorithm should consume low computation time while searching for QoS-constrained paths. In this work, we are motivated by the performance of LARAC reported in [1] and the growing demands for fast QoS-aware routing algorithms in SDN. To improve LARAC run time, we first make the algorithm search for a new start point (i.e., find the best fit solution), exploiting both LARAC's lower bound paths (i.e., cost and delay). Then, we modified LARAC to stop and avoid non-useful extra Dijkstra calls within the approximation phase.

4.1 Related Work

LARAC algorithm is proposed in [62] to solve the DCLC problem. Its main idea is based on relaxing the constraint by including it in the cost function. In the newly defined cost, $c_\lambda = c + \lambda * d$, values of c, d are link's cost and delay respectively. The Lagrangian multiplier, λ, determines the delay metric's weight while searching the network graph by Dijkstra. The value of λ is in interval $[0, +\infty)$, where, if $\lambda = 0$ least-

Table 4.1: Related Work

Ref	Main Contribution	Constraint	Network	Controller	Sub-routine algs
[146], [59], [71]	Use LARAC for video Streaming with weighted cost combined of loss and delay	delay	Multi-domain	Simulation	Dijkstra
[61]	Use LARAC for video Streaming with weighted cost of loss and delay variation	delay variations	Cloud	RYU	Dijkstra
[57], [73]	Use LARAC for video Streaming with weighted cost loss and delay	delay and delay variation	Campus	Floodlight	Dijkstra
[72]	Use LARAC for lossless video traffic	delay	Enterprise	OpenDaylight	Dijkstra
[73]	Use LARAC for video Streaming with delay as cost	delay variation	Campus	Floodlight	Dijkstra
[147]	Modify the Lagrangian multiplier, λ, by multiply it with another value x to make search delay-oriented.	delay	–	Simulation	k Shortest Path (kSP)
[148]	Extend LARAC to support hop-count constraint	delay and hop-count	–	–	Dijkstra
[149]	Extended LARAC to support for M criteria with combined $c\lambda_M$ cost. At each iteration, it may substitute a $c\lambda_M$-minimal path with a one of candidate paths generated according to each metric type. To get a final shortest path that comply with other $(M-1)$ metrics	$M-1$ constraints	–	Simulation	Dijkstra
[150]	Extend LARAC for smart grid communication, they used Floyd–Warshall algorithm instead of Dijkstra at first optimization step to reduce computation for multiple s, t pairs, while Dijkstra with c_λ within optimization loop	delay and packet loss	Smart community networks	–	Floyd–Warshall and Dijkstra
[151]	Minimize the global energy consumption. They use LARAC within k-shortest paths (kSP) algorithm to get k paths with energy cost	delay	Multi-hop wireless network	Simulation	kSP and LARAC
[145]	Adapt LARAC to to get a route passes through a specific set of nodes with. They use same c_λ as in original LARAC	delay, set of NFV nodes	–	Simulation	modified Dijkstra
[86]	Try to reduce the computation time by using destination-based routing. Extend LARAC for M metrics, a vector of c_λ values is repeatedly computed until a compliant path found.	delay, loss, jitter	–	ONOS	Dijkstra
[152]	They compute and choose one of two values of λ, one is similar to original LARAC, and another one as $\tan\theta, \theta \in [0, \frac{\pi}{2})$ and use a special algorithm called mDijkstra, to get two c_λ-minimal paths if they exist.	delay	–	Simulation	modified Dijkstra

cost path, p_c, is returned while least delay path, p_d, is obtained when λ approaches $+\infty$. Usage of weighted cost function alone does not guarantee the optimality of the solution (i.e., least cost path) or the required delay (i.e., the new path's delay may be above the constraint, $\triangle$) even if the path is the shortest according to c_λ. Therefore, LARAC iteratively adapts the value of λ until the best path is found. It keeps track of the best feasible and infeasible paths found yet, p_d, and p_c, respectively. When a newly obtained path, r (i.e., new λ), has the same c_λ cost value of the best infeasible path, p_c, then the algorithm stops, and the current best feasible path, p_d is returned as the solution (see Algorithm 3).

Authors in [147] proposed a LARAC similar algorithm. They change the Lagrangian multiplier, λ, by multiplying it with another value x. This new value indicates the tightness of the constraint, the more constraining the value of $\triangle$, the higher value of x. That increases the Lagrangian multiplier and makes the path search more delay-oriented. They also used a k shortest paths (kSP) algorithm within the optimization loop to achieve optimality, increasing runtime, especially in dense networks.

In [148], authors proposed extended LARAC (E-LARAC) to support hop count as an additional constraint besides delay; nonetheless, it is not usually considered in QoS provisioning.

The authors in [149] extended LARAC support for M criteria (i.e., cost and other $M-1$ QoS constraints) and called it MLARAC. Like LARAC, the shortest path is found for the first criteria (i.e., cost), and others' satisfaction is evaluated. If it violates any of those demands, a set of candidate paths is generated according to each metric type. A new path is obtained within each approximation loop according to a

combined $c\lambda_M$ cost function. It substitutes one of the candidate paths found in the earlier stage. The new cost, $c\lambda_M$, is similar to c_λ used within LARAC, however, it combines multiple constraints along with different multipliers, $\lambda_1, \ldots, \lambda_{M-1}$. Later on, in [153], authors evaluated different path substitution methods (e.g. random selection) in MLARAC. However, they found that the substitution method has no significant impact and reasoned that to the low number of iterations. Support of multiple constraints increases the running time of any QoS-aware algorithm. Some QoS metrics, especially concave ones such as bandwidth, can be guaranteed by running LARAC or any variant algorithm on residual resources graph in which bandwidth demand non-satisfying links are pruned from the original network graph.

Authors in [150] proposed a LARAC-based algorithm for smart community networks where a large number of smart devices pairs need QoS-aware communication. Since large pairs are communicating, LARAC needs to be executed repeatedly for each. Therefore, authors try to optimize the computation time by replacing Dijkstra used within LARAC with an extended version of Floyd–Warshall algorithm[154] at first steps. It reduces computation time due to its ability to work with multiple pairs and get a set of shortest path(s) between all pairs. However, within the approximation phase, the behavior is similar to the original LARAC. Similarly, in [151], the authors propose an algorithm called K-LARAC. It recursively calls LARAC to get a set of delay constrained least energy-consuming (i.e., energy consumption is the cost) paths for each user in an SDN-based multi-hop wireless network. It works similarly as in k-shortest paths (kSP) algorithm. However, and instead of calling only Dijkstra, it calls LARAC recursively with the required constraint. Similarly, authors in [145]

modified Dijkstra used within LARAC to get a route passes through a specific set of nodes (i.e., Service Function Chain). However, the computation of λ and the stop condition left the same as in the original LARAC.

Similar to [150], authors in [155] [86] try to reduce the computation time by using destination-based routing in overlay networks. For every destination and QoS policy combination, they generate a Direct Acyclic Graph (DAG). It requires that packets hold their QoS information in their header (i.e., delay and jitter) and updated at every hop forwarded. The graph is a destination-based least-cost tree. In which there is a path from every source at that graph toward the graph destination. That tree only holds nodes that have a path satisfying the QoS policy requirements. The cost of each link is the price to deliver traffic over it. The required bandwidth is assured by from residual resources graph. To cope with the constraints while building DAG, they utilized LARAC similar search. For other constraints (i.e., delay, jitter, loss), similar to [150], a vector of lambda values is repeatedly computed. The algorithm stops if the difference between the feasible path QoS metric value and the constraint reaches zero (i.e., stop loop). It is required to modify the data plane to make the QoS policy value replaces the flow source in the flow table entry. It may lead to flow granularity loss presented in SDN. The algorithm also enters the approximation phase immediately without examining lower bound paths (i.e., p_c and p_d) and solution feasibility as performed by most previous works. Also, without justification, they use initial values for the multiplier λ (e.g. $\lambda = 2$ or $\lambda = 3$) which is updated after each Dijkstra call.

The authors of LARAC mentioned that it might fail to obtain an optimal solution

Algorithm 3 LARAC algorithm

```
1: procedure LARAC(s, t, c, d, △)
2:     p_c := Dijkstra(s, t, c)
3:     if d(p_c) ≤ △ then
4:         return p_c
5:     end if
6:     p_d := Dijkstra(s, t, d)
7:     if d(p_d) > △ then
8:         return "There is no solution"
9:     end if
10:    repeat
11:        λ := (c(p_c) − c(p_d)) / (d(p_d) − d(p_c))
12:        r := Dijkstra(s, t, c_λ)                    ▷ c_λ := c + λ * d
13:        if c_λ(r) = c_λ(p_c) then
14:            return p_d
15:        else if d(r) ≤ △ then
16:            p_d := r
17:        else
18:            p_c := r
19:        end if
20:    end repeat
21: end procedure
    ▷ where Dijkstra(s, t, c) returns a c-minimal path between the nodes s and t
```

when there are paths with the same c_λ cost for a specific λ value. They argued that this case is not likely to occur in real networks. However, authors in [152] built their work based on that issue and proposed BiLAD algorithm. They assumed that for a fixed value of λ, there are more than one c_λ-minimal paths. Therefore, and in order to reduce the computation time, they modify Dijkstra, and called it mDijkstra, to get two c_λ-minimal paths if they exist. The first one, r_c, has the least cost (c), while the other, r_d, has the least delay (d). Both paths have the same c_λ cost value and may also be the same path when the assumption fails. They claimed this will save more than one Dijkstra calls. Moreover, they adopted a hybrid scheme for computing the Lagrangian multiplier λ that alternate between the original method of LARAC and another one depends on an angle θ value. In the (d, c)-plane, c_λ draw a line where c_λ-minimal path(s) (i.e. based on a λ value) lie on, and θ is the supplementary angle of the slope angle of that line (see Figure 4.2). In which there is a relation between λ and θ such that $\lambda = \tan\theta, \theta \in [0, \frac{\pi}{2})$. Therefore, they ensure that angle, θ, kept within an interval $[\underline{\theta}, \bar{\theta}]$. In LARAC, value of λ depends mainly on the delay and cost values of recently found best feasible and infeasible path (p_d, p_c). In BiLAD, before adopting λ value, they check the location of the new θ (i.e. $\arctan\lambda$). If it is too close to one of the edges of interval $[\underline{\theta}, \bar{\theta}]$, they recompute λ value based on θ as in $\lambda = \tan\frac{\bar{\theta}+\underline{\theta}}{2}$, which is the hybrid scheme they proposed. Value of either $\underline{\theta}$ or $\bar{\theta}$ is replaced with the current θ value at the end of each approximation loop. The algorithm stops when the constraint $\triangle$ is within the interval $[d(r_d), d(r_c)]$, or the size of interval $[\underline{\theta}, \bar{\theta}]$ becomes lower than a parameter value ϵ (see Algorithm 4). Table 4.1 shows summary of discussed related works.

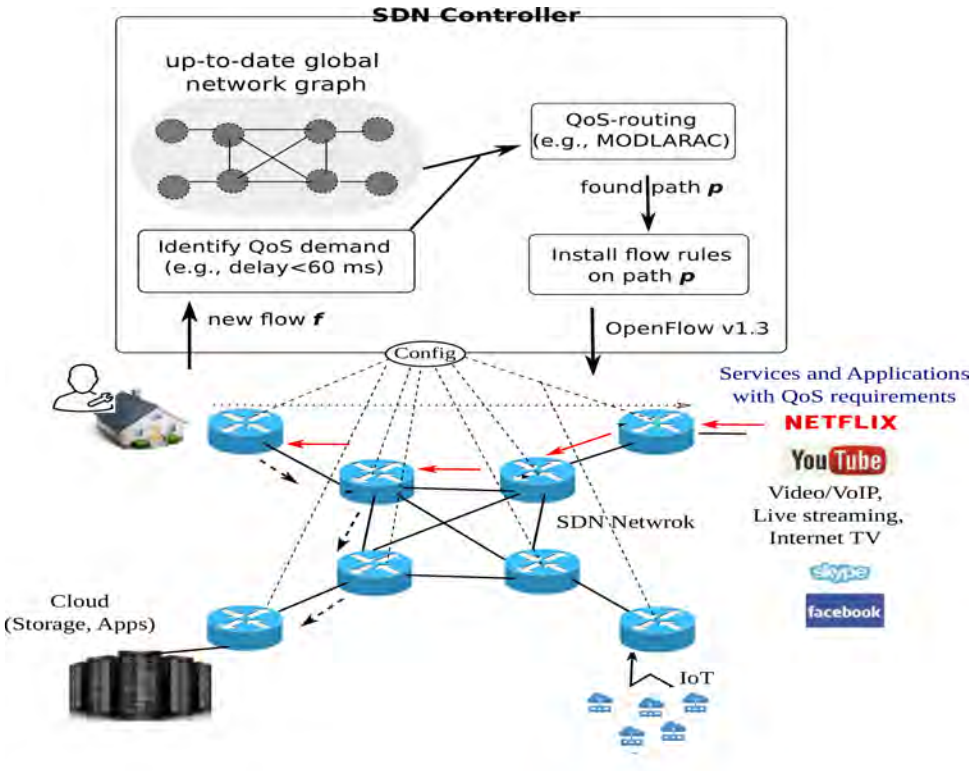


Figure 4.1: Delay constrained QoS routing in SDN.

In general, and for QoS provisioning, most of the reviewed works either adapt LARAC to be used in specific scenarios (i.e., smart networks, NFV), support multiple constraints, or focus on the exact optimal solution. Thus, with little focus on the computation time cost, especially when kSP algorithms are used to find the optimal solution. Furthermore, many of them depend on a special or modified version of Dijkstra to reduce computation time.

This work is similar to LARAC in which it is compatible with any shortest path algorithm. However, we focus on reducing computation time by exploiting lower bound paths (p_c, p_d) and stopping the search process when extra Dijkstra calls are not useful. The next section presents our proposed work and compares it against both LARAC and BiLAD algorithms.

Algorithm 4 BiLAD algorithm

procedure BiLAD($s, t, c, d, \triangle$)
  Set constant $\gamma \in (0, 1/2)$
  $p_c :=$ argmin $c(p)$ with the lowest delay, and $\underline{\theta} := 0$.
  if $d(p_c) <= \triangle$ **then**
    return p_c
  end if
  $p_d :=$ argmin $d(p)$ with the lowest cost, and $\bar{\theta} := \frac{\pi}{2}$.
  if $d(p_d) > \triangle$ **then**
    return "there is no solution"
  end if
  if $d(p_d) = \triangle$ **then**
    return p_d
  end if
  repeat
    if $\bar{\theta} - \underline{\theta} \leq \epsilon$ **then**
      return p_d
    end if
    $\lambda := \frac{c(p_d)-c(p_c)}{d(p_c)-d(p_d)}$ and $\theta := \arctan \lambda$.
    if $|\theta - \frac{\bar{\theta}+\underline{\theta}}{2}| > (\frac{1}{2} - \gamma)(\bar{\theta} - \underline{\theta})$ **then**
      $\theta := \frac{\bar{\theta}+\underline{\theta}}{2}$, $\lambda := \tan \theta$
    end if
    $r_c, r_d :=$ mDijkstra(s, t, c, d, c_λ) ▷ $c_\lambda := c + \lambda * d$
    if $d(r_d) \leq \triangle \leq d(r_c)$ **then**
      $p_c := r_c$, $p_d = r_d$
      if $(d(p_c) - \triangle) = 0$ **then**
        return p_c
      else
        return p_d
      end if
    else if $d(r_d) > \triangle$ **then**
      $p_c := r_d$ and $\underline{\theta} := \theta$.
    else if $d(r_c) < \triangle$ **then**
      $p_d := r_c$ and $\bar{\theta} := \theta$.
    end if
  end repeat
end procedure

▷ where mDijkstra(s, t, c, d, c_λ) returns two c_λ-minimal paths (r_c, r_d) between the nodes s and t.
They maybe the same path. r_c is a least cost, and r_d is a least delay.

4.2 Delay-Sensitive QoS-Aware Routing

In this section, we give a brief background on DCLC, LARAC and present our proposed work

4.2.1 DCLC Problem Formulation

In the SDN controller, a topology management module provides and maintains an updated network graph $G = (V, E)$. V is the set of network nodes (i.e., OpenFlow switches), and E is the set of links connecting them. In such graph, $e = (u, v)$ represent the link from node u to node v. Each link e is characterized with two values, delay $d(e)$ and cost $c(e)$. The cost can be simply hop-count, monetary value, or specifically defined cost function by the network owner or operator. In DCLC definition, these two values are non-negative, and the total path cost should be minimized. Also, its total delay should comply with a certain demand by the arriving flow(s), $\triangle$. The problem can be formulated as:

$$\min_{p \in P(s,t)} \sum_{e \in p} c(e) \qquad (4.1$$

Where e is a link along the feasible path p and $P(s, t)$ is the set of possible paths from s to t where each path's delay is bounded by delay constraint $\triangle$ as in 4.2:

$$\sum_{e \in p} d(e) \leq \triangle \qquad (4.2$$

In DCLC, a path is optimal if it has the least total cost, as in 4.1, and its delay is below or equal constraint $\triangle$, as in 4.2. DCLC is a well-known NP-hard problem[142][143]. Many heuristic algorithms are proposed in the literature to find the optimal solution for it. LARAC proves its outstanding performance over other opponents in run time and cost inefficiency, as reported in [1].

4.2.2 LARAC

Algorithm 3 shows how LARAC works. It can be divide into two phases:

1. Feasibility phase (Lines 2-9): In this phase, the algorithm checks the feasibility of a solution. First, it gets the shortest path, p_c, according to c cost function. Then, it checks compatibility with the delay constraint, $\triangle$. If the path delay is below or equals $\triangle$, then p_c is the optimal path according to the definition of DCLC problem, and the algorithm stops and returns p_c. Otherwise, it checks if there is a possible path in whatever cost by calling Dijkstra again with delay as the cost function (Line 6, Algorithm 3). If the returned path, p_d, has a delay above the constraint value, no solution exists, and the algorithm stops there. Some of the proposed DCLC heuristics use these two steps, sometimes in a different order. The advantage of getting p_c and p_d paths is the definition of lower bounds for cost and delay values. If the delay of path p_d is not bigger than the constraint, the algorithm enters the next phase to search for the best path.

2. Optimization or Approximation phase (Lines 10-20): LARAC uses p_c and p_d

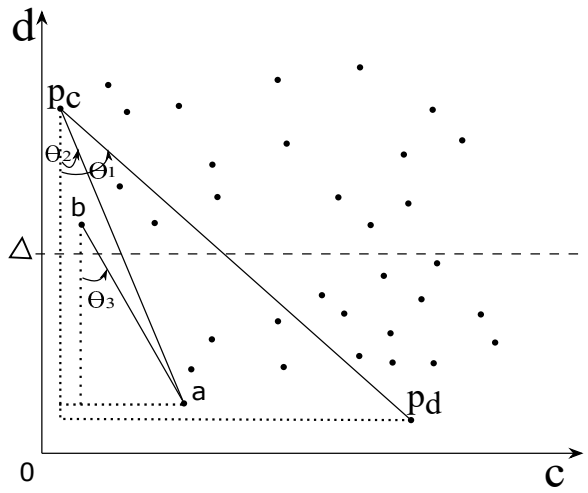


Figure 4.2: LARAC in cost and delay dimensions [1].

to compute λ multiplier as in line 11. This value will define the direction of the search using the newly defined cost function c_λ. Lower values direct toward cost-oriented search and ignore the delay and vice versa. Inside the optimization loop (Lines 10-20, Algorithm 3), the algorithm maintains and updates p_c and p_d as the best infeasible and feasible solutions found yet, respectively. For each loop, it update λ and searches for a possible path r according to c_λ, as in line 12. The algorithm stops if the condition in line 13 becomes true. Value of λ in line 11 assures that $c_\lambda(p_c) = c_\lambda(p_d)$ (i.e. additive cost). If the $c_\lambda(r)$ equals any of those of p_c or p_d (i.e. they use p_c) means r may be the same path as p_c or p_d. Then, the algorithm stops and returns the current best feasible path p_d based on the optimal multiplier λ value. Otherwise, update either, p_d or p_c depending on the delay of path r (Lines 15-19, Algorithm 3). A possible case that path r is a different new path, but its combined cost c_λ is equal to $c_\lambda(p_c)$. Authors in [62] referred to as a case where LARAC may fail to find an optimal solution.

Table 4.2: MODLARAC Mathematical Notations

Symbol	**Definition**
G	Nework Graph
s, t	Source and destination nodes
c	Link cost
d	Link Delay
$\triangle$	Delay constraint
P_c	Least cost path from s to t
P_d	Least delay path from s to t
λ	Computed lambda value
c_λ	Cost function computed using λ
r	Least path using c_λ cost
VC	Set of common nodes between P_c and P_d except s, t
P_{bf}	Best fit path of P_c and P_d
v, v'	Current and next common nodes in VC
$SFPath$	Current best sub-path obtrained
SFD	Delay of $SFPath$ sub-path

Figure 4.2 shows an example illustrates the operation of LARAC in both cost and delay space. It first finds p_c and p_d (i.e., p_c violate constraint $\triangle$ while p_d not). Then, it finds path a after using λ (i.e. first loop) computed from p_c and p_d and similarly b is found. Then, either a or b will be recomputed, and the algorithm will stop. As we mentioned, smaller values of λ direct toward a cost focus in the search process. We can also look at the value of λ as an equivalent to the tangent value of Θ angle as used in [152]. This example costs 5 Dijkstra calls to get path a as the final solution. In this chapter, we focus on improving the computation time of LARAC as we discuss in the next section. Figure 4.1 shows an illustration of how QoS-routing is working on top of the SDN controller.

4.2.3 Proposed Work

In this section we discuss changes we made to improve the running time of LARAC.

Algorithm 5 Modified LARAC algorithm

```
1: procedure MODLARAC(s, t, c, d, △)
2:     p_c := Dijkstra(s, t, c)
3:     if d(p_c) ≤ △ then
4:         return p_c
5:     end if
6:     p_d := Dijkstra(s, t, d)
7:     if d(p_d) > △ then
8:         return "There is no solution"
9:     end if
10:    VC := (p_c − {s, t}) ∩ (p_d − {s, t})                ▷ common nodes
11:    if VC ≠ φ then
12:        p_bf: = BestFit(s, t, p_c, p_d, △, VC)           ▷ find best fit
13:        if c(p_bf) < c(p_d) then
14:            p_d :=p_bf
15:        end if
16:    end if
17:    repeat
18:        λ := (c(p_c) − c(p_d)) / (d(p_d) − d(p_c))
19:        if λ ≠ 0 then
20:            r := Dijkstra(s, t, c_λ)                     ▷ c_λ := c + λ ∗ d
21:        end if
22:        if (λ = 0 or (c_λ(r) = c_λ(p_c))) then
23:            return p_d
24:        else if d(r) ≤ △ then
25:            if c(r) < c(p_d) then
26:                p_d := r
27:            else
28:                return p_d                               ▷ no improvement
29:            end if
30:        else
31:            p_c := r
32:        end if
33:    end repeat
   ▷ where Dijkstra(s, t, c) returns a c-minimal path between the nodes s and t
34: end procedure
```

Search Starting Point

Algorithm 5 shows modified LARAC, we call it MODLARAC. Similarly to LARAC (Lines 2-9, Algorithm 3) and BiLAD (Lines 3-13, Algorithm 4), MODLARAC checks the solution feasibility, (Lines 2-9, Algorithm 5). Differently than in LARAC and BiLAD, we propose to search for a sample of the solution space from previously computed two paths, p_c and p_d (Lines 2 and 6, Algorithm 5), respectively. We exploit already known information about each (i.e., delay, cost) and similarities between them. It likes bootstrapping MODLARAC from a better feasibility state. The new path should be constraint compatible and has a better cost than path p_d. In addition, we need to ensure that this process consumes little or minimum effort compared to searching the network graph. Therefore, we use the BestFit search procedure presented in Algorithm 6. But first, we get a list, VC, consist of common nodes between p_c and p_d excepting source and destination pair (Line 10, Algorithm 5). If VC is not empty, we execute the procedure in Algorithm 6.

Algorithm 6 shows the BestFit procedure. First, it obtains the first common node, v, from the list VC (i.e. list of common nodes received from MODLARAC) as in line 2, Algorithm 6. It uses node v to find the best sub-path from s to v (Lines 3-8, Algorithm 6). $p_c(s, v)$ is the sub-path from s to v on path p_c, while $p_d(s, v)$ is the sub-path from s to v on path p_d. In line 3 the algorithm checks superiority of sub-path $p_c(s, v)$ in both delay and cost values. If it is better than sub-path $p_d(s, v)$, we accept it as the best so far computed sub-path, $SFPath$, as in line 4. Otherwise, sub-path $p_d(s, v)$ is chosen as the best so far path as in line 6 $SFPath$ is the best sub-path

Algorithm 6 BestFit algorithm

```
1: procedure BestFit (s, t, c, d, p_c, p_d, △, VC)
2:     v := VC.Pop(0)                                   ▷ get and remove first node
3:     if (d(p_c(s,v)) ≤ d(p_d(s,v))) and (c(p_c(s,v)) ≤ c(p_d(s,v))) then
4:         SFPath := p_c(s,v)                                          ▷ so far path
5:     else
6:         SFPath := p_d(s,v)
7:     end if
8:     SFD := d(SFPath)                                              ▷ so far delay
9:     repeat
10:        if SFD + d(p_c(v,t)) ≤ △ then
11:            return (SFPath + p_c(v,t))
12:        end if
13:        if VC ≠ φ then
14:            v' := VC.Pop(0)
15:            if (d(p_c(v,v')) ≤ d(p_d(v,v'))) and (c(p_c(v,v')) ≤ c(p_d(v,v'))) then
16:                SFPath + := p_c(v,v')
17:            else
18:                SFPath + := p_d(v,v')
19:            end if
20:            SFD := d(SFPath)
21:            v = v'
22:        else
23:            return (SFPath + p_d(v,t))
24:        end if
25:    end repeat
26: end procedure
    ▷ This procedure needs at most |VC| (i.e., number of common nodes) iterations.
```

obtained so far and SFD is its delay (i.e., so far delay). Condition in line 10 checks if the total delay of sub-path $SFPath$ (i.e., SFD) in addition to the delay of sub-path from current node, v, to destination, t, along p_c (i.e., $p_c(v,t)$), is less than or equal constraint, $\triangle$. If so, the current $SFPath$ is joined with the sub-path $p_c(v,t)$ from current node, v, to t along path p_c as in line 11. The algorithm stops and return the obtained path. Otherwise, it check first if the list VC is not empty and still there are other common nodes, as in line 13. Then, it obtains the next common node v' and get the best sub-path from current v to v' as in lines 14 to 20 and update both $SFPath$ and SFD. After that, v' becomes the current common node. If list VC is empty, the algorithm stop and returns the joint of current $SFPath$ and the rest of path p_d, from v to t as in line 23. The consumed time by BestFit procedure depends on the number of common nodes in both p_c and p_d. Therefore, it has a max number of iteration equals the size of VC. We use BestFit algorithm only to get a better path of p_c and p_d if such exist. Any other procedure can be used since it has a shorter run time than Algorithm 6. It should find a path better (i.e, constraint compatible and less cost) than or at the worst case the same as p_d.

Getting back to Algorithm 5, the condition in line 11 checks if the common node(s) list, VC, is not empty. If so, the BestFit procedure is executed as in line 12, and p_{bf} path is obtained. If it has a lower cost than p_d, we accept it as the new best feasible solution without using λ or Dijkstra search (Line 14, Algorithm 5). From this point in the solution space, MODLARAC algorithm starts its optimization phase (Lines 17-33, Algorithm 5). If no common node(s) exist, the algorithm benefits only from improvement based on the stop condition, as we explain next.

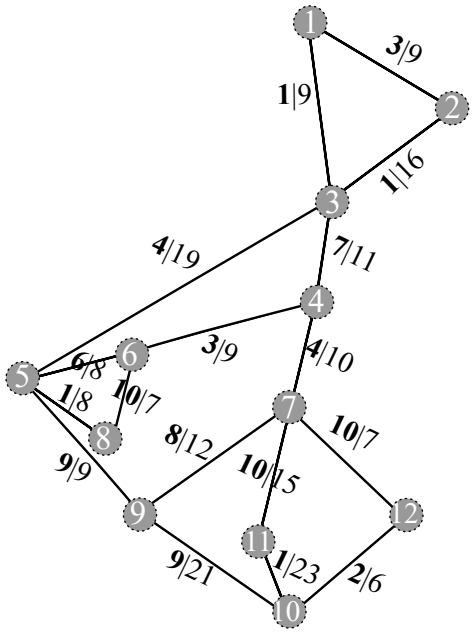


Figure 4.3: The QoS flow from source 1 to destination 10 requests $\triangle \leq 65ms$. State of $\lambda = 0$ encountered.

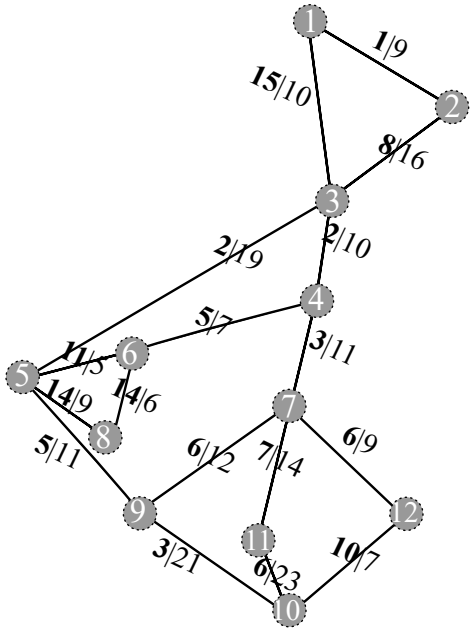


Figure 4.4: The QoS flow from source 1 to destination 10 requests $\triangle \leq 75ms$.

Stop Condition

In DCLC, the optimality condition is that the discovered path should has the least cost and a delay lower than or equals to the constraint. Therefore, more than one optimal path may coexist. During LARAC operation, we may get into a state that the current p_d and p_c paths have the same cost value. Therefore, a zero-valued λ is computed, which will lead to a path r (Line 12, Algorithm 3) equals to the first p_c and goes back to the starting point of the algorithm. Therefore, we changed LARAC to stop if such a state encountered and return the best feasible path, p_d since its cost equals the best infeasible solution, p_c. Changes are reflected in MODLARAC (Lines 19-23, Algorithm 5) by modifying the original stop condition and start r path search only if $\lambda \neq 0$. Figure 4.3 shows an example of such state ($\lambda = 0$). The QoS flow from source 1 to destination 10 demands a total path delay should be less or equals 65 milliseconds ($\triangle = 65ms$). Each graph link is associated with two values, cost (i.e. bold font) and delay ($\mathbf{c}|d$). By Using LARAC, first least cost path p_c is found as :$1 \rightarrow 3 \rightarrow 4 \rightarrow 7 \rightarrow 11 \rightarrow 10$ with total cost and delay equal 23, 68 respectively (Line 2 Algorithm 3). Since this p_c delay violates the required constraint (Line 3 Algorithm 3), LARAC searches for the least delay path p_d (Line 6 Algorithm 3). It obtains p_d path is in : $1 \rightarrow 3 \rightarrow 4 \rightarrow 7 \rightarrow 12 \rightarrow 10$ with a total cost and delay equal 24, 43 respectively. Delay of p_d is less than the constraint, therefore, LARAC enter the approximation or optimization loop (Lines 10-20, Algorithm 3). First, it computes the Lagrangian multiplier, $\lambda = 0.04$. Then, it search for the least c_λ-cost path, r (Line 12,

Algorithm 3) and as in: $1 \rightarrow 3 \rightarrow 5 \rightarrow 9 \rightarrow 10$, with a total cost and delay equal 23, 58 respectively. Since $c_\lambda(r) \neq c_\lambda(p_c)$(Line 13, Algorithm 3), it needs to adapt λ and search for another feasible path. p_d is substituted with r path since r has less delay than the constraint as shown in lines 15-16, Algorithm 3. At this moment, LARAC already called Dijkstra three times (i.e., for p_c, p_d, r). However, when LARAC enter the second loop, costs of both path p_c and p_d are equals which lead to a zero λ value. LARAC should stop here since p_d already the optimal path, however, it does not. Using Dijkstra with c_λ and zero λ value will make Dijkstra to go back toward the first produced least cost path p_c since the delay metric totally ignored (i.e, $c_\lambda = c + \lambda * d$). Therefore, the newly produced r, in the second loop, is the same as path stored in p_c. This makes the condition in line 13 evaluated to be true and the algorithm returns the path stored in p_d as the best found solution and LARAC stops there with a total Dijkstra calls equal 4 instead of 3. If we look at MODLARAC (Lines 19-23 Algorithm 5), we made a condition to avoid the situation LARAC fall in when λ equals zero (Line 19, Algorithm 5) and also modified the stop condition to stop in such situation (Line 22, Algorithm 5). MODLARAC returns the same path returned by original LARAC with one less Dijkstra run. This state that LARAC may falls in, increase the running time and lead to extra non useful Dijkstra calls.

Moreover, we strictly limit MODLARAC to update p_d only when a better cost r path is found (Lines 25-26). Otherwise, the algorithm stops and returns the current p_d path as no more improvement is found. Therefore, we think that LARAC should keep looking for lower-cost solutions, or it should stop the search process and take advantage of reducing computation time, especially in the central control scenario, as

in SDN[129].

Figure 4.4 shows another behavior trace instance. The QoS flow from 1 to 10 demands a total path delay 75 milliseconds or less ($\triangle = 75ms$). Similarly, LARAC (Lines 2-9, Algorithm 3), BiLAD (Lines 3-13, Algorithm 4) and MODLARAC (Lines 2-9, Algorithm 5) check the solution feasibility. First, p_c path is obtained as in: $1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 9 \rightarrow 10$ with total cost and delay equal 19, 76 respectively. Since p_c delay violate the constraint (Lines 3, 4, and 3 in Algorithms 3, 4, and 5 respectively), a search for the least delay path, p_d is started. Path p_d is obtained as: $1 \rightarrow 3 \rightarrow 4 \rightarrow 7 \rightarrow 12 \rightarrow 10$ with total cost and delay equal 36, 47 respectively. Since there is a path, p_d, exists with a delay less than the required flow demand, the approximation begin. Before that, BiLAD initializes it θ's lower and upper bounds ($\underline{\theta} = 0, \bar{\theta} = \frac{\pi}{2} \approx 1.57$). At this point, both LARAC and MODLARAC behave similarly. However, in our proposal we try to benefit from the existence of p_c and p_d. We used procedure in Algorithm 6 to find a different path, p_{bf}, that comply with the delay constraint by combining both p_c and p_d. Therefore, MODLARAC (Line 10 in Algorithm 5) finds a list VC of common nodes between p_c and p_d except source and destination pair. From the procedure in Algorithm 6, MODLARAC finds a BestFit path, p_{bf} as in: $1 \rightarrow 3 \rightarrow 5 \rightarrow 9 \rightarrow 10$ with total cost and delay equal 25, 61 respectively. Since p_{bf} has a better cost than p_d, MODLARAC replaces p_d with p_{bf} and proceed to optimization loop as in line 14, Algorithm 5. Since VC has only one node (i.e., node 3), we get the BestFit path, p_{bf}, of both p_c and p_d in only one iteration which is a neglectable overhead compared with searching the whole network graph.

At the beginning of each iteration, BiLAD checks the size of interval $[\underline{\theta}, \bar{\theta}]$ if it less or equals a parameter, ϵ (Line 15, Algorithm 4). We set $\gamma = 0.05$ as in [152] and $\epsilon = \frac{\pi}{9} (\approx 0.349)$. Since the current interval size, $(\bar{\theta} - \underline{\theta}) = 1.57$, is bigger than ϵ, BiLAD proceeds and computes values of $\lambda = 0.586$, and $\theta = 0.53$ (Line 18, Algorithm 4). Both values are accepted since condition in line 19 is not evaluated to be true ($|\theta - \frac{\bar{\theta}+\underline{\theta}}{2}| = 0.255 < (\frac{1}{2} - \gamma)(\bar{\theta} - \underline{\theta}) = 0.7065$). In line 22 of Algorithm 4, BiLAD calls a modified version of Dijkstra, mDijkstra, that return two c_λ-minimal paths, r_c (i.e., c_λ-minimal path with the least c cost) and r_d (i.e., c_λ-minimal path with the least d delay). In this iteration, mDijkstra fails to get two different paths. Both r_c and r_d are the same as in : $1 \rightarrow 3 \rightarrow 5 \rightarrow 9 \rightarrow 10$, with a total cost and delay equal 25, 61 respectively. Since r_c delay is below the constraint, $\triangle$, condition in line 32 is evaluated to be true and therefore, values of p_d and $\bar{\theta}$ is updated (Line 33, Algorithm 4). BiLAD proceeds to next iteration and check the value of $(\bar{\theta} - \underline{\theta}) = 0.53$ which is still bigger than ϵ. Therefore, BiLAD computes a new value of $\lambda = 0.4$ and $\theta = 0.38$ (Line 18, Algorithm 4). As previously, value of λ is accepted and used within mDijkstra to get r_c, r_d (Line 22, Algorithm 4). In this iteration, mDijkstra find two different r_c, r_d $c\lambda$-minimal paths as :$1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 9 \rightarrow 10$ and $1 \rightarrow 3 \rightarrow 5 \rightarrow 9 \rightarrow 10$ respectively. Path r_c has cost and delay values 19, 76, and path r_d has cost and delay 25, 61 respectively. The constraint value is in between delays of both r_c and r_d (Line 23 Algorithm 4). Therefore, values of p_c and p_d is updated as in line 24. Then, the algorithm check if delay of p_c equals the constraint value which is not. Therefore, BiLAD return p_d as the best path and stops after four Dijkstra calls (i.e., more specifically mDijkstra). Path $1 \rightarrow 3 \rightarrow 5 \rightarrow 9 \rightarrow 10$ is the obtained path by BiLAD. Getting back to LARAC,

it computes $\lambda = 0.568$ value in line 11 and uses Dijkstra to get r path (Line 12, Algorithm 3). obtained path r as in :$1 \rightarrow 3 \rightarrow 5 \rightarrow 9 \rightarrow 10$ has total cost and delay equal 25, 61 respectively. Since $c_\lambda(r) \neq c_\lambda(p_c)$ (Line 13, Algorithm 3), it needs to adapt λ and search for another feasible path. p_d is substituted with r path since r has less delay than the constraint as shown in lines 15-16, Algorithm 3. LARAC enters the next iteration and computes a newer $\lambda = 0.4$ value. It gets new r path as in: $1 \rightarrow 3 \rightarrow 5 \rightarrow 9 \rightarrow 10$ total cost and delay equal 25, 61 respectively. This the same path as the current p_d which makes the condition in line 13 to be evaluated true. Then LARAC stops here and return the current p_d path which is $1 \rightarrow 3 \rightarrow 5 \rightarrow 9 \rightarrow 10$, after four Dijkstra calls. Even they have different stop condition and work slightly in a different way, Both LARAC and BiLAD behave similarly. They got the solution path after 4 Dijkstra calls. However, our MODLARAC it gets the same solution with 3 Dijkstra calls. As discussed, MODLARAC obtained a BestFit path, p_{bf} and updated p_d before entering the optimization loop (i.e., it pushes MODLARAC into better state). Current, p_c and p_d paths are as in :$1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 9 \rightarrow 10$ and $1 \rightarrow 3 \rightarrow 5 \rightarrow 9 \rightarrow 10$ respectively. In the first iteration, MODLARAC computes $\lambda = 0.4$ as in line 18, Algorithm 5. In line 20, it searches and obtains r path using Dijkstra (i.e., since $\lambda \neq 0$, Line 19). Found r path as in: $1 \rightarrow 3 \rightarrow 5 \rightarrow 9 \rightarrow 10$ with total cost and delay 25, 61 respectively. Since $c_\lambda(r) = c_\lambda(p_c)$, MODLARAC stops and return current p_d path ($1 \rightarrow 3 \rightarrow 5 \rightarrow 9 \rightarrow 10$) as its solution which the same as path found by BiLAD and LARAC but with lower Dijkstra calls.

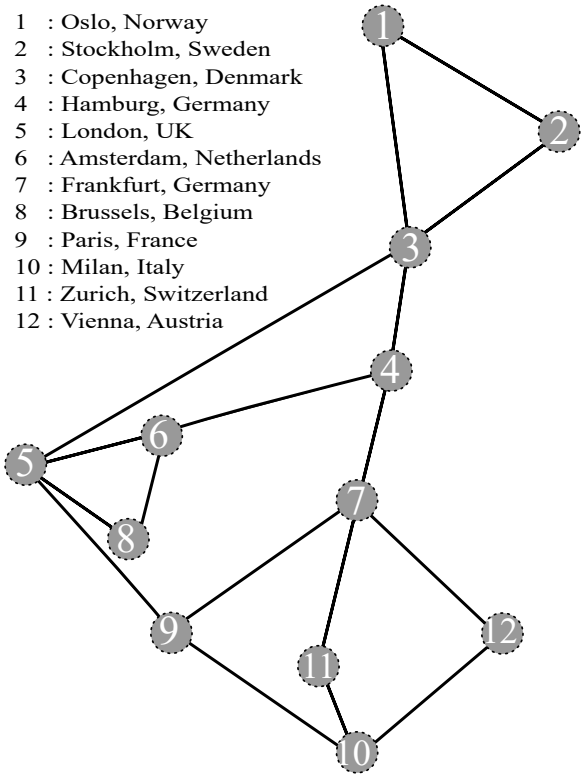


Figure 4.5: Topology from Sprint IP backbone Network

Table 4.3: Experiment parameter

Parameter	**Value**
Network emulation	Mininet 2.3.0
Topology	IP Sprint Europe (12 nodes)
OS	Ubuntu 14.04
Controller	FloodLight SDN Controller
OpenFlow	Version 1.3
Link rate	1Mbps
Link cost	$\in$ [1,15]
Flow rate	$\in$ [18,300] kpbs
Flow delay demand	$\in$ [50,80] milliseconds
γ	0.05
ϵ	$\frac{\pi}{9} = 20^o \approx 0.349$

4.3 Performance evaluation

In this section, we evaluate the performance of our proposed work.

4.3.1 Experiment Setup

For comparison with MODLARAC, we implement LARAC and BiLAD within the Floodlight SDN controller [45]. Floodlight is a Java-based SDN controller used in many QoS provisioning research works [23]. The underline SDN network data plane is emulated using Mininet [130]. OpenFlow v1.3 [131] is used as the southbound interface supported by Floodlight and Mininet virtual switches. We used a realistic Internet Service Provider (ISP) topology from Sprint [132]. Figure 4.5 shows the Sprint IP backbone network topology in Europe. Each node represent an OpenFlow enabled OpenVSwitch[133]. Each link is configured with a rate of 1Mb/s. We use such a rate due to the processing limitation of the experimental setup. QoS traffic flows between two nodes, from 1 and 10, with rates between 18 and 300 kbps. In the background, non-QoS traffic flows between other network nodes with rates uniformly distributed within the interval [18,300] kbps. We used the iperf tool to generate such traffic [134]. The least cost paths are used to route non-QoS traffic, while tested algorithms (i.e., LARAC, BiLAD, and MODLARAC) are used for QoS-flows. Throughout all experiments, the delay constraint, $\triangle$, requested by each QoS flow is uniformly distributed within the interval [50,80] milliseconds[156]. Each experiment runs for 500 seconds. Each link is characterized with two metrics, delay and cost. The link's delay is composed of the propagation and queuing delay. We initially configured links' prop-

agation delay similar to delay values reported in Sprint IP network performance online statistics[132], while queuing delay is dynamic and affected by the traffic passing along that link. Cost values are uniformly distributed within an interval $[1, 15]$ as used in [62] where LARAC is proposed. For BiLAD, we try to use the same configuration as in [152]. We set parameter $\gamma = 0.05$, and $\epsilon = \frac{\pi}{9} = 20^o \approx 0.349$.

4.3.2 Results and Discussion

In this section, we present the obtained results and compare algorithms' behavior. We used three metrics to evaluate performance: the average of path cost, run time as in the count of Dijkstra calls, and path delay. We need to mention that all results presented are for situations where the required delay demand is not satisfied by the least cost path, p_c. Therefore, more search is needed, and in which the performance of QoS-aware algorithms becomes noticeable. Obtained results are averaged over 20 runs with a 95% confidence interval.

Average Path Cost

Figure 4.6 shows the average path cost, along with different flow's delay demands. As expected, low delay requests cause any QoS-aware algorithm to suffer for such demands satisfaction. Therefore, as in Figure 4.6, average path cost is higher with low constraint values (e.g. when $\triangle \leq 65ms$) and drops down as the requested delay is relieved. In such situations, algorithms try to sacrifice cost for constraint compliance. In general, all three algorithm gets almost similar cost values. In many cases, they

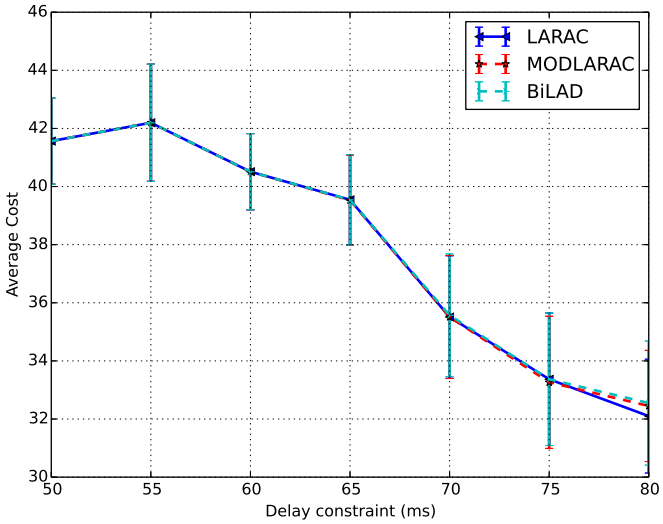


Figure 4.6: Average Path Cost

obtain the same path (e.g., when $\triangle \leq 70ms$). However, MODLARAC, in some cases, gets a little bit lower cost paths than both LARAC and BiLAD. This is due to its BestFit procedure that tries to get the best cost of p_c and p_c. While LARAC and BiLAD miss that due to their behavior that depends only on the λ multiplier to traverse the search space. Also, it may miss some better paths for computation time reduction. In general, and as it is clear from Figure 4.6, we can say the proposed modification to LARAC, in MODLARAC, achieve not much degradation in obtained paths cost.

Average Run-time Count

Figure 4.7 shows the average Dijkstra calls used by LARAC, BiLAD, and MODLARAC to get paths satisfying the constraint. As we mentioned before, results are shown only for those demands not satisfied by p_c. Therefore, at least three Dijkstra

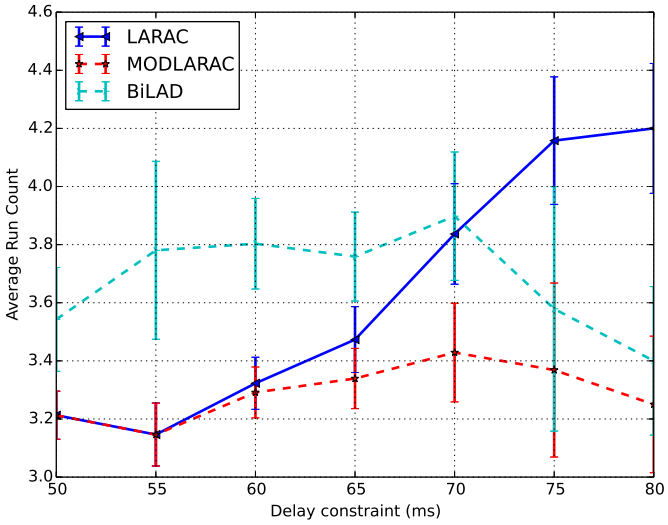


Figure 4.7: Average run time as in Dijkstra calls count

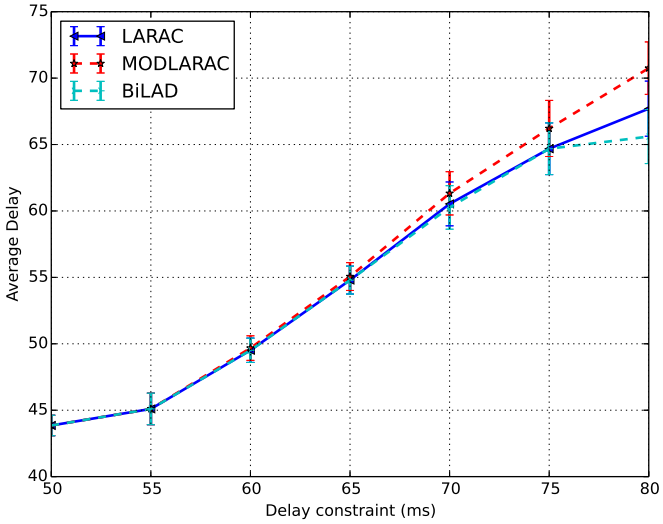


Figure 4.8: Average Path Delay

calls are performed to get a solution by LARAC, MODLARAC, and two by BiLAD if the condition in Line 11, Algorithm 4 evaluated to be true. Two of them for the lower bounds, p_c and p_d paths. The extra one(s) are for subsequent r path(s) exploration. The first section of curves, when $\triangle \leq 60ms$, shows almost the same behavior of both LARAC and MODLARAC. This is due to the little room for improvement caused by small constraints to find a better path than p_d. This is reflected in BiLAD's performance. It Consumes more mDijkstra calls before it stops because mDijkstra fails to find two different r paths, r_c, r_d. This means that obtained r_c and r_d are the same paths, which makes the condition in Line 23, Algorithm 4 evaluated to be false. Therefore, BiLAD depends on the value of ϵ, in Line 15, Algorithm 4 to stop.

However, when higher constraints are requested, the performance difference between them becomes noticeable. LARAC keeps calling Dijkstra with different λ multiplier values to obtain lower costs paths in its original implementation. There is a chance that LARAC faces a situation where $\lambda = 0$, which leads to falling back toward p_c direction search. This case worsened when LARAC changed the value of path p_c (i.e., substituted p_c when $d(r) > \triangle$, Line 18 Algorithm 3) before it reached a zero λ situation while MODLARAC avoids it (Line 19, Algorithm 5). The BestFit procedure cuts running time by finding a constraint-compliant path from both p_c and p_d. In many cases, this path might be found in later approximation iterations after readjusting the λ value in LARAC or BiLAD. Which saves MODLARAC a few Dijkstra calls by jumping to that solution. It merges p_c, p_d, and gets the best benefit or fit of them with a negligible computation cost. In general, as in Figure 4.7, it is clear that LARAC consumes more computation time if the least cost path p_c fails to

satisfy delay, especially when the delay gap between p_c and p_d is high. While BiLAD performance is affected clearly by the constraint tightness (≤ 70). Smaller constrain values make BiLAD depends totally on ϵ value to stop. Average mDijkstra calls drop-down as the QoS flow requests higher demand. Even it outperformed LARAC when constraint requests were above 70 milliseconds. This is due that BiLAD gets benefits from mDijkstra as there is a higher probability of getting two different paths, r_c, r_d which makes BiLAD stops earlier than LARAC. On the other hand, MODLARAC behavior is consistent and is not affected since it benefits from such auxiliary when high constraint values are used to get the best fit of p_c, p_d (i.e., if they share common nodes) and cut down the running time. Moreover, similarly to LARAC, it is not affected by low constraints, as is the case in BiLAD.

Average Path Delay

Figure 4.8 shows the average path delay, along with the required flow's delay constraints. When delay constraints are higher than $65ms$, there is a minor increase in path delay obtained by MODLARAC. This is because MODLARAC does not usually return the exact path as LARAC (i.e., see Figure 4.10). This is due to the BestFit procedure that tries to get the best of p_c and p_d. Therefore, more paths are closer to p_c are returned, which affects the obtained paths delay. However, all obtained paths are satisfying the constraint. Even if those paths returned by MODLARAC have a higher delay, they are constraint compliant with a save distance from the max allowed delay, $\triangle$ (i.e., 8ms in its worst case).

Figure 4.10 shows the average path exactness of both BiLAD and MODLARAC

with LARAC. If the algorithm finds the same path as LARAC did, exactness takes a value of 1 and 0 otherwise. As it appears, LARAC and BiLAD, most of the time, return the same paths, which explains why they have almost the same cost and delay values as in Figures 4.6 and 4.8 respectively. Beside that, Figure 4.10 shows the average cost and run count ratios. It is the average of MODLARAC, BiLAD values normalized by those corresponding to LARAC. Similarly, as in Figures 4.6 and 4.7, BiLAD in Figure 4.10, has more run time in values of Dijkstra calls ratio compared to LARAC when constraint is less than $70ms$. MODLARAC has the best performance than both LARAC, BiLAD with no much increase in path cost.

Moreover, Figure 4.10 shows values of two factors, $P1$ and $P2$ computed as in equations 4.3 and 4.4 respectively. Both values are computed before LARAC, MODLARAC or BiLAD do any approximation, more specifically after p_d is firstly computed and before the optimization phase.

$$P1 = \frac{\triangle - d(p_d)}{d(p_c) - d(p_d)} \quad (4.3)$$

$$P2 = \frac{d(p_d)}{\triangle} \quad (4.4)$$

Both $P1$ and $P2$ measures the constraint tightness but in different ways. $P1$ computes the ratio of the distance between p_d toward $\triangle$ over the variance of p_c and p_d delays. $P2$ computes the ratio of delay constraint with the least path delay p_d. $P1$ measures how var both p_c and p_d from the delay constrain line (see Figure 4.9). $P2$ only affected by the distance between p_d and the constraint, $\triangle$. Beside that, $P1$ is also consider delay distance between the lower bound paths, p_c and p_d. Small $P1$ values mean the delay

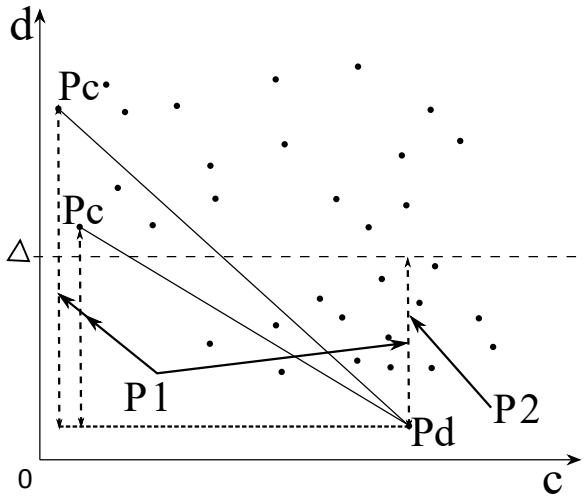


Figure 4.9: Constraints tightness factors, P1 and P2.

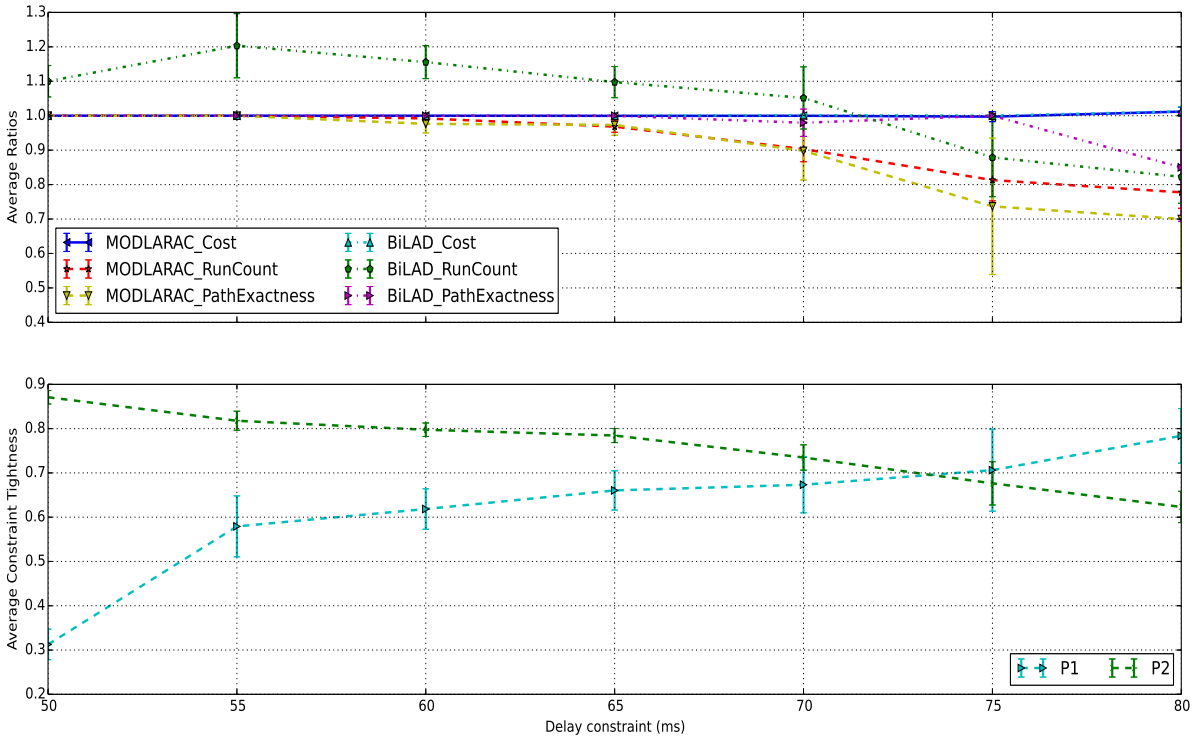


Figure 4.10: Average run-time, and path cost Ratios

is constrained and has high tightness, similar to high $P2$ values. This indicates the small available room for finding a better path. Therefore, p_d is usually returned as the solution of LARAC, BiLAD, and MODLARAC. This is shown clearly in Figure 4.10 when $\triangle \leq 65ms$. This relation becomes clear when both $P1, P2$ value becomes closer to each other. The performance of LARAC, BiLAD, and MODLARAC differs clearly. This is due to lower constraint tightness values. Even we did not use those factors in our work. However, they have a good indication of solution search feasibility. For instance, we may watch the distance between these two factors and decide whether to proceed or stop the search process and get enough with the current obtained best feasible path, p_d. It is a design trade-off between accuracy and fast computing QoS-aware algorithm. In summary, the obtained results showed improvement up to a 20% reduction in Dijkstra calls count compared to original LARAC and BiLAD algorithms without a significant increase in path's cost or delay metrics. Furthermore, it reached only a 3% increase in path cost and 7% in path delay in its worst case with a safe distance of 11% lower than the delay's demand.

CHAPTER 5

ADAPTIVE UTILITY-BASED LINK STATE

An up-to-date network state is a critical requirement for QoS provisioning in SDN. In which a miss-match in network state between the control and data planes would cause bad QoS decisions. In SDN, the monitoring function can be implemented within or above the controller to observe the network state. The network switch keeps different counters per table and entries (e.g., flow, port). Then, the controller uses specific OpenFlow messages (i.e., *StatReq*) to poll statistics from those network devices. Such counters and primitives are well-defined in the OpenFlow specifications. Therefore, any OpenFlow compatible switch should implement all of them as specified in the corresponding OpenFlow version. However, there is a limitation in OpenFlow specification for monitoring links delay. Therefore, many research works use specially crafted packets to probe the link latency or exploit the existing Link Layer Discovery Protocol (LLDP) protocol (i.e., discovery module) to examine link delay.

5.1 Related Work

Most of the existing implementation of link delay monitoring adopts packet probing mechanisms, either specially crafted or LLDP-based. For instance, the authors in [74] propose an approach for link latency monitoring. It consists of two modules namely; *LLDP discovery* and *Echo monitoring*. The first module tests the link latency between two switches using Link Layer Discovery Protocol (LLDP) packets. For the directional test, the controller sends an LLDP packet to the first switch, which already guides by the controller to forward that it the second switch as in Figure 5.1. It has no rules to tell how to deal with this packet (i.e., or guided to forward such packets to the controller as in Figure 5.1). Therefore, it send a *packetIn* message to the controller. By this process, the controller records the link latency between the two switches in one direction only, if the opposite direction is tested, another process is performed. The second module is used to calculate the propagation delay from the controller toward a network switch by sending times-tamped Echo messages towards that switch. Upon reception of that packet, the switch returns it to the controller, see Figure 5.1. By doing that, the controller computes the link latency between any switches. Similarly, did the authors in [157] by exploiting LLDP packets for link delay estimation and using Echo messages to tell the propagation delay between the controller and switches. In [158] implemented a link discovery approach similar to LLDP but with smaller packets. LLDP works as in Figure 5.1 in its simple form. The authors used that discovery procedure to measure links delay and periodically repeat that task. In [159], the authors injected a packet into the network that loops

along the test path and goes back to the controller to test the latency, and this is called the TTL loop. Then, similar to all probe-based approaches, it needs to insert rules for matching those packet TTL, decrements its value, and forward toward the controller when $TTL = 0$. Similarly, the work done in [65], [160], [161], [162] use similar approach to compute the link delay. In [160], the authors use probes arrival delays and RTT (Round Trip Time) values to compute path delay. Using the same way, the authors in [161] measures the flow's end-to-end delay. Each probe packet travels the path from the first to the last switch and back to the controller. This process is repeated periodically for each monitored flow. Upon its reception, the flow delay is estimated after subtracting the RTT from the controller to both switches. However, in [163], authors added a coefficient of proportionality varying from 0 to 1. They considered the Round Trip Time from the first and last switched toward the controller.

Similarly, in [164] delay is measured using the probe-based approach for sub-paths. Link delay is inferred from those sub-paths delays unless it is uncovered. Then, it is tested similarly to previously discussed works. The authors in [65] proposed a framework called *SLAM* to monitor path latency in SDN-based data center networks in order to detect high delay network segments. Authors use customized probe packets to trigger *PacktInt* notification packets toward the controller when a new unmatched packet is received in the network switch. Then, it measures the latency based on the arrival times of the *PacktInt* message to the controller.

For path delay, approach used in *OpenNetMon* [165] injected packets that travel between edge switches along the same interested flow's path. The delay is computed as

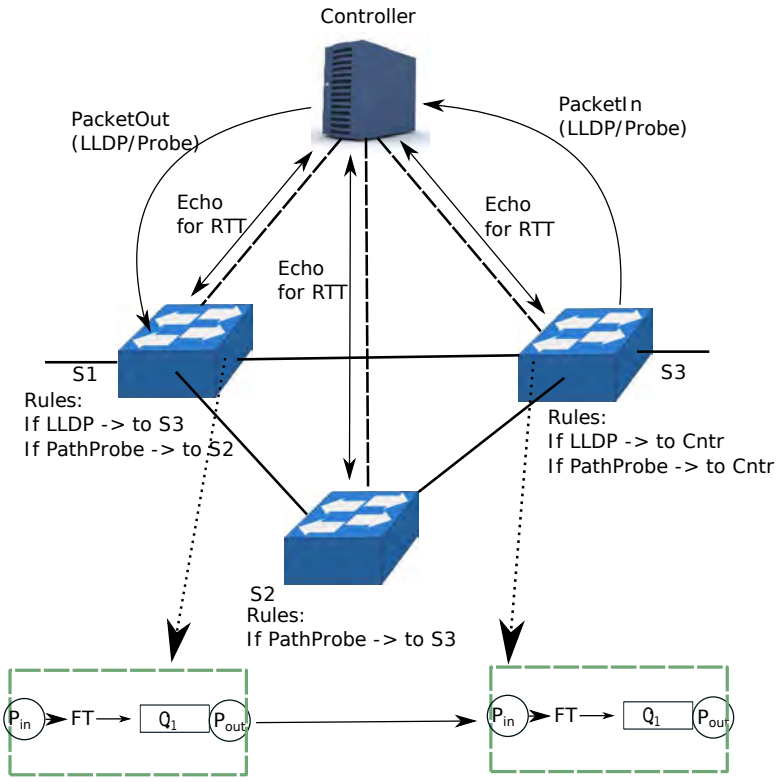


Figure 5.1: Link delay measurements in SDN

the difference between arrival (i.e., at egress switch) and departure (i.e., from ingress switch) of that injected packet. The difference of probe packets arrival and departure is also used in [166] to compute link latency. The authors in [167] utilize the same probe-based approach (i.e. similar to LLDP packets in Figure 5.1) to obtain the link delay.

For testing the queue delay, authors in [168] utilize the same probe-based approach, where a special packet is sent to the first switch, which is configured to send it to the next one. The receiving switch sends the packet to the controller since it does not know what to do. The difference in this work is in the first switch, along with the output action, the tested queue is selected by using *enqueue* action (i.e., changed to *set_queue* in later OF versions). By that, queue delay could be tested according to

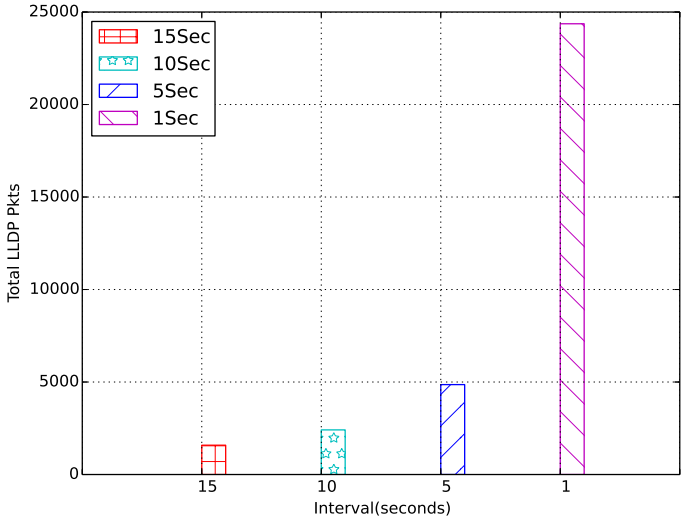


Figure 5.2: LLDP overhead in relation with probe interval

that approach, as in Figure 5.1 the queue's delay between *S1* and *S2*.

In summary, we can say that most of these methods are link-oriented, in which to measure the delay of a link, the controller needs to send a probe packet for that specific link which might cause network overhead depending on the probe rate and level of required accuracy, which is different than in OpenFlow statistics. A switch can reply to a *StatReq* message with a single message that contains all switch port statistics. Also, we note that both link's delay state is decoupled from the switch state captured by the OpenFlow statistic messages. Therefore, in this chapter, we discuss the problem of link-delay state accuracy at the control level and try to benefit from existing OpenFlow statistics to improve the whole process.

5.2 Proposed work

One of the severe issues in LLDP-based latency monitoring is the overheard since for each tested link, one or more LLDP packets are sent (i.e., *PacketIn* packets to inform the controller). Figure 5.2 shows sent LLDP packets in the network according to different probe intervals for a small network with six links. As it is clear, an increase in the probe rate will exponentially increase the overhead. At the end of each interval and for each network link, the controller obtains a latency sample. In this work, we assume a History Queue (HQ) with a size M where specific link latency samples are stored for each link. This queue is used to compute the average latency of that link. During the whole network lifetime, each link latency is computed from that queue. It behaves like a history window with size M through the whole link latency population during network life. In Floodlight, link latency is updated if the change ratio between the current latency value and the average of the HQ values is above a certain threshold. Such threshold-based updates may lead to a miss-match between network and controller state, especially with configuration long discovery intervals. For instance, a link with high latency due to congestion may not be detected, affecting other QoS functions such as QoS routing. On the other hand, a high delayed link might be chosen in a path for a delay-sensitive flow that requires constrained delay values. Therefore, we propose to adapt this process by exploiting link utilization statistics collected by OpenFlow-based messages due to its accuracy and short acquisition intervals (i.e., usually every 5 seconds).

Figure 5.3 shows the proposed approach. The process starts when a new link

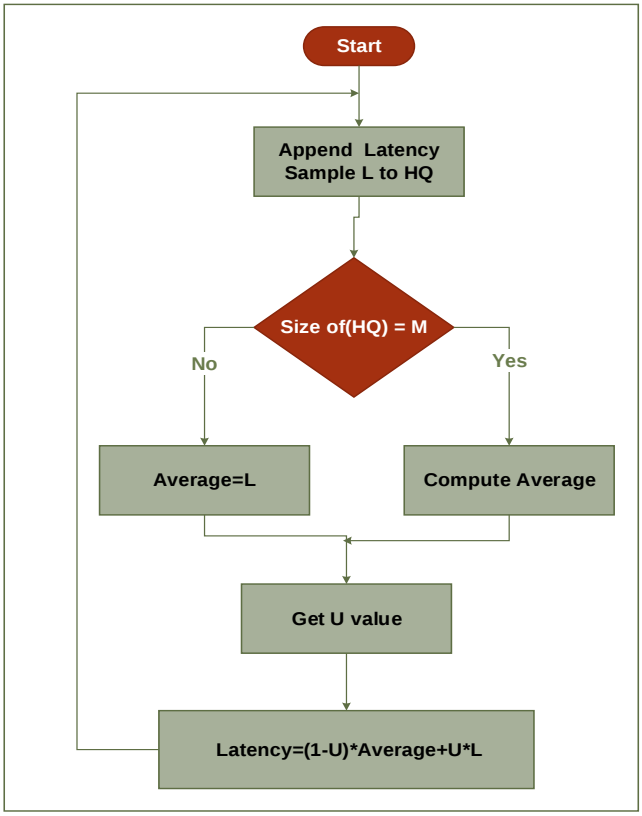


Figure 5.3: Adaptive link latency update process

Table 5.1: Experiment parameter

Parameter	**Value**
Network emulation	Mininet 2.3.0
Topology	IP Sprint Europe (12 nodes)
OS	Ubuntu 14.04
Controller	FloodLight SDN Controller
OpenFlow	Version 1.3
Link rate	1Mbps
Link cost	$\in$ [1,15]
Flow rate	$\in$ [100,300] kpbs
Port statistic interval	5 seconds
LLDP interval	10 seconds
Delay agent statistic interval	2.5 seconds using ping

sample is observed. The new latency value, L, will be appended to the HQ. If there are enough samples (i.e., the queue length is M), an average value is computed; otherwise, it uses the new sample as the average. Then, the process gets the utilization ratio, U, of that link from the current values collected by the OpenFlow statistic messages. This value, is with an interval [0,1]. A small value means the link is idle and underutilization, and vice versa. Then, based on these three values, L, $Average$, and U, a weighted link latency value is computed using equation 5.1.

$$Latency = (1 - U) * Average + U * L \qquad (5.1)$$

By doing that, we incorporate the link's state collected by OpenFlow messages into those collected by LLDP in the form of the weight value U.

5.3 Performance evaluation

In this section, we evaluate the performance of our proposed work.

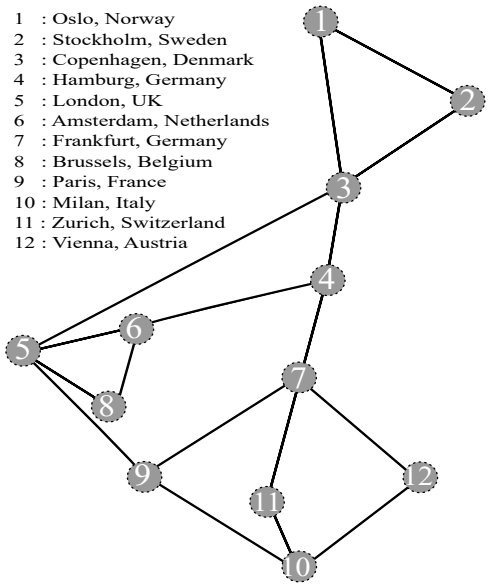


Figure 5.4: Topology from Sprint IP backbone Network

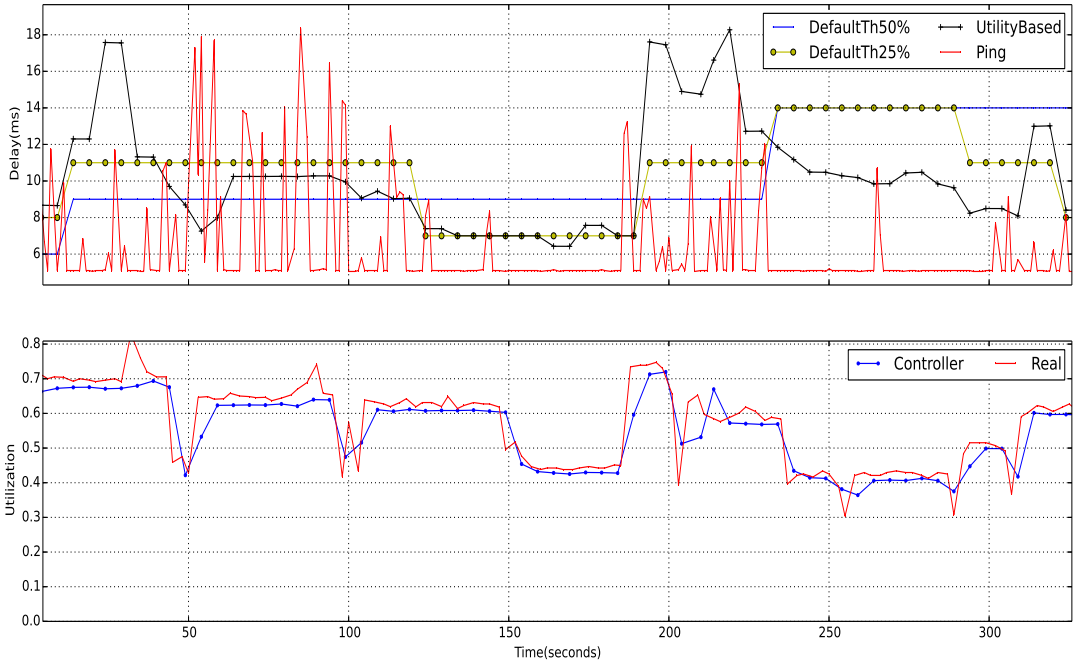


Figure 5.5: Link-delay update process, threshold-based with values %25, %50, vs utility-based update

5.3.1 Experiment Setup

We implement the proposed approach in Floodlight SDN controller [45]. Floodlight is a Java-based SDN controller used by many QoS provisioning research works [23]. The underline SDN network data plane is emulated using Mininet [130]. OpenFlow v1.3 [131] is used as the southbound interface supported by Floodlight and Mininet virtual switches. We used a realistic Internet Service Provider (ISP) topology from Sprint [132]. Figure 5.4 shows the Sprint IP backbone network topology in Europe. Each node represents an OpenFlow enabled OpenVSwitch[133]. For link statistics, the controller collected link port statistics every 5 seconds using OpenFlow-based statistic messages. At the same time, LLDP is used to test link latency with an interval of 10 seconds. Moreover, we install agents to capture real port statistics in a short interval (e.g., 2.5 seconds) to capture real statistics. For real link latency, we ping each link every second during the whole experiment time.

5.4 Results and Discussion

Figure 5.5 shows a time trace of link latency during the simulation time. The top part of the figure shows the LLDP-based link latency state, while the bottom part shows the OpenFlow-based link utilization state. It is clear that OpenFlow-based link's state (i.e., link utilization ratio) is more accurate in capturing the actual behavior of the link. Therefore, there is a high degree of state matching in both control and data planes. That is the reason that motivates us to use that information while updating the link latency state. In the same figure (i.e., top part), the effect of the update threshold

on the link latency values is apparent. For instance, with 50% (i.e., the default value in Floodlight controller) change threshold, the current value changed only two times, more specifically at seconds 10, 235. Decreasing the threshold value leads to more responsive latency values. The current link latency value is changed at time values 10, 120, 190, 235, 290, and 320 seconds. Even with this low threshold value, it does not always capture the actual behavior of the link latency. For instance, in the interval from 235 to 290, the actual link's delay is decreased when link utilization is reduced (i.e., below 40%). However, the sampled data values still do not affect or propagate through HQ's history queue, which is not the case in our utility-based update process. In the same time interval, the link latency is decreased. Even in some cases, our approach records high latency values. It can capture the change intervals in the link latency during the simulation time, especially when there is a decrease in the link utilization rate.

In general, we can say that link latency state monitoring still needs more investigation. Our proposed update process detects changes in link latency state faster than in threshold-based methods, which suffers from the steady-state effect of the history queue and threshold configuration.

CHAPTER 6

CONCLUSION AND FUTURE WORK

Software-Defined Networking is a revolutionary network architecture that came up with many benefits that facilitated operation overhead and reduced the service provisioning time. In this dissertation, we focused our effort on the aspect of QoS provisioning in the SDN architecture. We extensively surveyed the literature in fields that support QoS and cover QoS's whole life cycle in SDN. Furthermore, we studied different QoS problems and proposed solutions to improve the overall performance. More specifically, at the aspect of resource allocation and QoS-aware routing. We implemented and evaluated a Multi-Class Group-based Resource allocation and Refinement (MCGRR) that improved resource utilization and fair resource allocation to increase the business revenue. Moreover, we implemented and evaluated a LARAC-based QoS-aware algorithm, called MODLARAC, that solves the DCLC problem, a well-known NP-hard problem, and finds a solution in a reasonably fast time compared

to existing algorithms.

In the future, and to improve the proposed solutions, we plan to incorporate multi-path support in the MCGRR algorithm and evaluate its effect on network utilization and load balancing. Moreover, we plan to do extra investigation in benefiting from both constraint tightness factors in MODLARAC. We plan to exploit them and design a probability function to make an early decision to perform the optimization process or not. Moreover, we can use it to direct solution space search, either to be cost or delay-oriented. We expect that it will help in reducing the computation time and reduce controller overhead.

In general, and for research in the field of QoS support in SDN, we identified the following possible future direction:

- There is a limitation in standardization of the resources configuration (i.e., hardware-level such as queues creation) proposals and implementation in SDN. Almost all QoS research works use vendor-specific protocols.

- SDN uses a central control architecture responsible for processing a tremendous number of flows in which the computation overhead is a severe issue. Therefore, more research effort needs to be taken to propose faster resource allocation approaches and flows routing.

- SDN reasonably provides the right tools for automation that will reduce the operational cost. However, there is a lack of autonomic QoS end-to-end QoS management in SDN, which is still an open issue and requires more literature effort.

- Incorporating Machine Learning (ML) in QoS management and decision making. ML can learn from customers' behavior and help the network adapt its resource according to customers and service requirements. Also, its learning ability from actions taken by the network operator to resolve issues helps exploit human experience in the form of machine-defined policies that can resolve the same or similar future problems autonomically. However, ML has training issues that require large datasets that span long time intervals, and which techniques should be uses needs more investigation in the field of QoS support.

REFERENCES

[1] J. W. Guck, A. Van Bemten, M. Reisslein, and W. Kellerer, "Unicast QoS Routing Algorithms for SDN: A Comprehensive Survey and Performance Evaluation," *IEEE Communications Surveys and Tutorials*, vol. 20, no. 1, pp. 388–418, 2018.

[2] Cisco, "Cisco Visual Networking Index: Forecast and Methodology 2016–2021," 2017.

[3] R. Balakrishnan, *Advanced QoS for Multi-Service IP/MPLS Networks*, 2012.

[4] R. Braden, D. Clark, and S. Shenker, "Rfc1633: Integrated services in the internet architecture: an overview," 1994.

[5] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," 1998.

[6] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "Rsvp: A new resource reservation protocol," *IEEE network*, vol. 7, no. 5, pp. 8–18, 1993.

[7] ONF, "Open networking foundation," 2021. [Online]. Available: https://www.opennetworking.org/

[8] Open Networking Foundation (ONF), "SDN Architecture Overview," 2013. [Online]. Available: http://opennetworking.wpengine.com/wp-content/uploads/2013/02/SDN-architecture-overview-1.0.pdf

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM computer communication review*, vol. 38, no. 2, pp. 69–74, 2008.

[10] OpenVSwitch, "Open virtual switch," 2021. [Online]. Available: http://www.openvswitch.org/

[11] Y. E. Oktian, S. Lee, H. Lee, and J. Lam, "Distributed sdn controller system: A survey on design choice," *computer networks*, vol. 121, pp. 100–111, 2017.

[12] N. Feamster and J. Rexford, "Why (and how) networks should run themselves," *arXiv preprint arXiv:1710.11583*, 2017.

[13] M. Howard, "RESEARCH NOTE - IHS Markit: SDN deployed by 78% of global service providers at end of 2018," 2019. [Online]. Available: https://techblog.comsoc.org/2019/01/28/ihs-markit-sdn-deployed-by-78-of-global-service-providers-at-end-of-2018/

[14] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Holzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined WAN," in *SIGCOMM 2013 - Proceedings of the ACM SIGCOMM 2013 Conference on Applications,*

Technologies, Architectures, and Protocols for Computer Communication, 2013, pp. 3–14. [Online]. Available: https://dl.acm.org/citation.cfm?id=2486019

[15] ONF, "OpenFlow Switch Specification 1.0." [Online]. Available: https://opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf

[16] ONF, "OpenFlow Switch Specification 1.1." [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.1.0.pdf

[17] ONF, "OpenFlow Switch Specification 1.2." [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.2.pdf

[18] ONF, "OpenFlow Switch Specification 1.3." [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf

[19] ONF, "OpenFlow Switch Specification 1.4." [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.4.0.pdf

[20] ONF, "OpenFlow Switch Specification 1.5." [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.0.pdf

[21] A. Mirchev, "Survey of concepts for qos improvements via sdn," *Future internet (FI) and innovative internet technologies and mobile communications (IITM)*, vol. 33, no. 1, 2015.

[22] 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.

[23] A. Binsahaq, T. R. Sheltami, and K. Salah, "A survey on autonomic provisioning and management of qos in sdn networks," *IEEE Access*, vol. 7, pp. 73 384–73 435, 2019.

[24] H.-c. Chu and T.-h. Lin, "An Adaptive User-Defined Traffic Control Mechanism for SDN," in *Lecture Notes in Electrical Engineering*, 2018, vol. 425, pp. 609–619. [Online]. Available: http://link.springer.com/10.1007/978-981-10-5281-1_67

[25] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in software defined networks," *Computer Networks*, vol. 71, pp. 1–30, 2014.

[26] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[27] I. M. Alsmadi, I. AlAzzam, and M. Akour, "A systematic literature review on software-defined networking," *Information fusion for cyber-security analytics*, pp. 333–369, 2017.

[28] P.-W. Tsai, C.-W. Tsai, C.-W. Hsu, and C.-S. Yang, "Network monitoring in software-defined networking: A review," *IEEE Systems Journal*, vol. 12, no. 4, pp. 3958–3969, 2018.

[29] H. Cui, W. Lai, L. Zheng, and Y. Liu, "Accurate Network Resource Allocation in SDN according to Traffic Demand," in *Proceedings of the 4th International*

Conference on Mechatronics, Materials, Chemistry and Computer Engineering 2015, no. Icmmcce, 2015, pp. 1166–1175.

[30] A. Rego, A. Canovas, J. M. Jiménez, and J. Lloret, "An intelligent system for video surveillance in iot environments," *IEEE Access*, vol. 6, pp. 31 580–31 598, 2018.

[31] J. W. Guck and W. Kellerer, "Achieving end-to-end real-time quality of service with software defined networking," in *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*. IEEE, 2014, pp. 70–76.

[32] Y. Njah, C. Pham, and M. Cheriet, "Service and Resource Aware Flow Management Scheme for an SDN-Based Smart Digital Campus Environment," *IEEE Access*, vol. 8, pp. 119 635–119 653, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9127419/

[33] O. Dobrijevic, M. Santl, and M. Matijasevic, "Ant colony optimization for qoe-centric flow routing in software-defined networks," in *2015 11th international conference on network and service management (CNSM)*. IEEE, 2015, pp. 274–278.

[34] A. Kassler and L. Skorin-Kapov, "Towards QoE-driven multimedia service negotiation and path optimization with software defined networking," in *SoftCOM 2012, 20th International Conference on Software, Telecommunications and Computer Networks*, 2012, pp. 1–5.

[35] M. Karl, J. Gruen, and T. Herfet, "Multimedia optimized routing in openflow networks," in *2013 19th IEEE International Conference on Networks (ICON)*. IEEE, 2013, pp. 1–6.

[36] M. S. Seddiki, M. Shahbaz, S. Donovan, S. Grover, M. Park, N. Feamster, and Y.-q. Song, "FlowQoS: Per-Flow Quality of Service for Broadband Access Networks," *Proceedings of the third workshop on Hot topics in software defined networking - HotSDN '14*, pp. 207–208, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2620728.2620766

[37] 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 *2013 Second European Workshop on Software Defined Networks*. IEEE, 2013, pp. 87–92.

[38] T. Zinner, M. Jarschel, A. Blenk, F. Wamser, and W. Kellerer, "Dynamic application-aware resource management using software-defined networking: Implementation prospects and challenges," in *2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–6.

[39] C. R. Vasconcelos, R. C. M. Gomes, A. F. Costa, and D. D. C. da Silva, "Enabling high-level network programming: A northbound api for software-defined networks," in *2017 International Conference on Information Networking (ICOIN)*. IEEE, 2017, pp. 662–667.

[40] S. Gorlatch, T. Humernbrum, and F. Glinka, "Improving qos in real-time internet applications: from best-effort to software-defined networks," in *2014 international conference on computing, networking and communications (ICNC)*. IEEE, 2014, pp. 189–193.

[41] S. Gorlatch and T. Humernbrum, "Enabling high-level qos metrics for interactive online applications using sdn," in *2015 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2015, pp. 707–711.

[42] 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 *2013 19th IEEE Workshop on Local & Metropolitan Area Networks (LANMAN)*. IEEE, 2013, pp. 1–3.

[43] H. Balwani, P. Wagh, R. Vadaje, S. Shivgunde, and P. Wakode, "Implementation of qoe/qos mapping insdn," *Int. Res. J. Eng. Technol.*, vol. 4, no. 4, pp. 957–959, 2017.

[44] H. Owens II and A. Durresi, "Video over software-defined networking (vsdn)," *Computer Networks*, vol. 92, pp. 341–356, 2015.

[45] Floodlight, "Floodlight OpenFlow Controller," available at https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview, Accessed On 2020-08-23. [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview

[46] R. Walker, "Implementing discrete mathematics: combinatorics and graph theory with mathematica, steven skiena. pp 334. 1990. isbn 0-201-50943-1 (addison-wesley)," *The Mathematical Gazette*, vol. 76, no. 476, pp. 286–288, 1992.

[47] A. Kucminski, A. Al-Jawad, P. Shah, and R. Trestian, "Qos-based routing over software defined networks," in *2017 IEEE international symposium on broadband multimedia systems and broadcasting (BMSB)*. IEEE, 2017, pp. 1–6.

[48] S. Ren, Q. Feng, and W. Dou, "An End-to-End QoS Routing on Software Defined Network Based on Hierarchical Token Bucket Queuing Discipline," in *Proceedings of the 2017 International Conference on Data Mining, Communications and Information Technology - DMCIT '17*. New York, New York, USA: ACM Press, 2017, pp. 1–5. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3089871.3089883

[49] J. Huang, L. Xu, M. Zeng, C.-C. Xing, Q. Duan, and Y. Yan, "Hybrid Scheduling for Quality of Service Guarantee in Software Defined Networks to Support Multimedia Cloud Services," in *2015 IEEE International Conference on Services Computing*. IEEE, jun 2015, pp. 788–792. [Online]. Available: https://ieeexplore.ieee.org/document/7207433/

[50] Á. L. Valdivieso Caraguay, J. A. Puente Fernández, and L. J. García Villalba, "Framework for optimized multimedia routing over software defined networks," *Computer Networks*, vol. 92, pp. 369–379, dec 2015. [Online]. Available:

https://www.sciencedirect.com/science/article/pii/S1389128615003230https://linkinghub.elsevier.com/retrieve/pii/S1389128615003230

[51] S. Tomovic, N. Prasad, and I. Radusinovic, "SDN control framework for QoS provisioning," in *2014 22nd Telecommunications Forum Telfor (TELFOR)*. IEEE, nov 2014, pp. 111–114. [Online]. Available: http://ieeexplore.ieee.org/document/7034369/

[52] A. M. Al-Sadi, A. Al-Sherbaz, J. Xue, and S. Turner, "Routing algorithm optimization for software defined network WAN," in *2016 Al-Sadeq International Conference on Multidisciplinary in IT and Communication Science and Applications (AIC-MITCSA)*. IEEE, may 2016, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/7759945/

[53] S. Shamim and Z. Fei, "Evaluating a qos aware path selection service using the geni network," in *2016 IEEE International Conference on Ubiquitous Wireless Broadband (ICUWB)*. IEEE, 2016, pp. 1–4.

[54] S. Tomovic and I. Radusinovic, "Fast and efficient bandwidth-delay constrained routing algorithm for SDN networks," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. IEEE, jun 2016, pp. 303–311. [Online]. Available: http://ieeexplore.ieee.org/document/7502426/

[55] M.-T. Kao, B.-x. Huang, S.-j. Kao, and H.-w. Tseng, "An Effective Routing Mechanism for Link Congestion Avoidance in Software-Defined Networking," in

2016 International Computer Symposium (ICS). IEEE, dec 2016, pp. 154–158. [Online]. Available: http://ieeexplore.ieee.org/document/7858461/

[56] A. Gangwal, M. Gupta, M. S. Gaur, V. Laxmi, and M. Conti, "Elba: Efficient layer based routing algorithm in sdn," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, 2016, pp. 1–7.

[57] D. E. Henni, A. Ghomari, and Y. Hadjadj-Aoul, "Videoconferencing over OpenFlow networks: An optimization framework for QoS routing," *Proceedings - 15th IEEE International Conference on Computer and Information Technology, CIT 2015, 14th IEEE International Conference on Ubiquitous Computing and Communications, IUCC 2015, 13th IEEE International Conference on Dependable, Autonomic and Se*, pp. 491–496, 2015.

[58] S. Civanlar, M. Parlakisik, A. M. Tekalp, B. Gorkemli, B. Kaytaz, and E. Onem, "A QoS-enabled openflow environment for scalable video streaming," *2010 IEEE Globecom Workshops, GC'10*, pp. 351–356, dec 2010. [Online]. Available: http://ieeexplore.ieee.org/document/5700340/

[59] H. E. Egilmez, S. Civanlar, and A. M. Tekalp, "An Optimization Framework for QoS-Enabled Adaptive Video Streaming Over OpenFlow Networks," *IEEE Transactions on Multimedia*, vol. 15, no. 3, pp. 710–715, apr 2013. [Online]. Available: http://ieeexplore.ieee.org/document/6376227/

[60] H. E. Egilmez, B. Gorkemli, A. M. Tekalp, and S. Civanlar, "Scalable video streaming over openflow networks: An optimization framework for qos routing,"

in *2011 18th IEEE International Conference on Image Processing*. IEEE, 2011, pp. 2241–2244.

[61] C. Xu, B. Chen, P. Fu, and H. Qian, "A Dynamic Resource Allocation Model for Guaranteeing Quality of Service in Software Defined Networking Based Cloud Computing Environment," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer, Cham, 2015, pp. 206–217. [Online]. Available: http://link.springer.com/10.1007/978-3-319-27051-7_18

[62] A. Juttner, B. Szviatovski, I. Mécs, and Z. Rajkó, "Lagrange relaxation based method for the qos routing problem," in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, vol. 2, 2001, pp. 859–868 vol.2.

[63] 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 *Proceedings of the 2012 Asia Pacific signal and information processing association annual summit and conference*. IEEE, 2012, pp. 1–8.

[64] J. W. Guck, M. Reisslein, and W. Kellerer, "Function Split Between Delay-Constrained Routing and Resource Allocation for Centrally Managed QoS in Industrial Networks," *IEEE Transactions on Industrial Informatics*,

vol. 12, no. 6, pp. 2050–2061, dec 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7517394/

[65] C. Yu, C. Lumezanu, A. Sharma, Q. Xu, G. Jiang, and H. V. Madhyastha, "Software-defined latency monitoring in data center networks," in *International Conference on Passive and Active Network Measurement*. Springer, 2015, pp. 360–372.

[66] H. Owens, A. Durresi, and R. Jain, "Reliable video over software-defined networking (rvsdn)," in *2014 IEEE Global Communications Conference*. IEEE, 2014, pp. 1974–1979.

[67] Gang Liu 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)*, vol. 2, 2001, pp. 743–749 vol.2.

[68] O. Dobrijevic, A. J. Kassler, L. Skorin-Kapov, and M. Matijasevic, "Q-POINT: QoE-Driven Path Optimization Model for Multimedia Services," 2014, pp. 134–147. [Online]. Available: message:%3C4FD6F406CF829D40B1B48CD9A9CA5F2B0FFD7E@MAIL4.fer.hr%3E%5Cnpapers3://publication/uuid/34DD56BB-7965-4C49-85C1-36DD0E9C2A57http://link.springer.com/10.1007/978-3-319-13174-0_11

[69] C. Lin, K. Wang, and G. Deng, "A QoS-aware routing in SDN hybrid networks," *Procedia Comput. Sci.*, vol. 110, pp. 242–249, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.procs.2017.06.091

[70] R. Kumar, M. Hasan, S. Padhy, K. Evchenko, L. Piramanayagam, S. Mohan, and R. B. Bobba, "End-to-end network delay guarantees for real-time systems using sdn," in *2017 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2017, pp. 231–242.

[71] H. E. Egilmez and A. M. Tekalp, "Distributed QoS architectures for multimedia streaming over software defined networks," *IEEE Transactions on Multimedia*, vol. 16, no. 6, pp. 1597–1609, 2014.

[72] W. E. Liang and C. A. Shen, "A high performance media server and QoS routing for SVC streaming based on Software-Defined Networking," *2017 International Conference on Computing, Networking and Communications, ICNC 2017*, pp. 556–560, 2017.

[73] T. F. Yu, K. Wang, and Y. H. Hsu, "Adaptive routing for video streaming with QoS support over SDN networks," *International Conference on Information Networking*, vol. 2015-Janua, pp. 318–323, 2015.

[74] X. Zhang, W. Hou, L. Guo, S. Wang, Q. Zhang, P. Guo, and R. Li, *Joint optimization of latency monitoring and traffic scheduling in software defined heterogeneous networks*, 2018, vol. 234 LNICST. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-3-319-78078-8.pdf#page=113

[75] J. W. Guck, M. Reisslein, and W. Kellerer, "Model-based control plane for fast routing in industrial qos network," in *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*. IEEE, 2015, pp. 65–66.

[76] S. C. Lin, I. F. Akyildiz, P. Wang, and M. Luo, "QoS-aware adaptive routing in multi-layer hierarchical software defined networks: A reinforcement learning approach," in *Proceedings - 2016 IEEE International Conference on Services Computing, SCC 2016*, 2016, pp. 25–33.

[77] G. Stampa, M. Arias, D. Sánchez-Charles, V. Muntés-Mulero, and A. Cabellos, "A deep-reinforcement learning approach for software-defined networking routing optimization," *arXiv preprint arXiv:1709.07080*, 2017.

[78] C. Yu, J. Lan, Z. Guo, and Y. Hu, "DROM: Optimizing the Routing in Software-Defined Networks With Deep Reinforcement Learning," *IEEE Access*, vol. 6, pp. 64 533–64 539, 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8502806/

[79] A. Azzouni, R. Boutaba, and G. Pujolle, "NeuRoute: Predictive Dynamic Routing for Software-Defined Networks," sep 2017. [Online]. Available: http://arxiv.org/abs/1709.06002

[80] A. Azzouni and G. Pujolle, "NeuTM: A neural network-based framework for traffic matrix prediction in SDN," in *IEEE/IFIP Network Operations and Management Symposium: Cognitive Management in a Cyber World,*

NOMS 2018. IEEE, apr 2018, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/8406199/

[81] Y.-S. Yu and C.-H. Ke, "Genetic algorithm-based routing method for enhanced video delivery over software defined networks," *International Journal of Communication Systems*, vol. 31, no. 1, p. e3391, 2018.

[82] D. Cavendish and M. Gerla, "Internet qos routing using the bellman-ford algorithm," in *International Conference on High Performance Networking*. Springer, 1998, pp. 627–646.

[83] O. Dobrijevic, M. Santl, and M. Matijasevic, "Ant colony optimization for qoe-centric flow routing in software-defined networks," in *2015 11th international conference on network and service management (CNSM)*. IEEE, 2015, pp. 274–278.

[84] N. Yuan, Z. Zhang, W. Li, F. Qi, S. Guo, X. Qiu, and W. He, "Design of Real-Time Resource-Aware Network Resource Allocation Mechanism Under SDN Background," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11633 LNCS. Springer, Cham, jul 2019, pp. 642–652. [Online]. Available: http://link.springer.com/10.1007/978-3-030-24265-7_55

[85] X. Huang, T. Yuan, and M. Ma, "Utility-Optimized Flow-Level Bandwidth Allocation in Hybrid SDNs," *IEEE Access*, vol. 6, pp. 20 279–20 290, 2018. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8327805/

[86] N. Varyani, Z.-l. Zhang, and D. Dai, "QROUTE: An efficient Quality of Service (QoS) routing scheme for software-defined overlay networks," *IEEE Access*, pp. 1–1, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9096304/

[87] D. Wu, Z. Li, J. Wang, Y. Zheng, M. Li, and Q. Huang, "Vision and Challenges for Knowledge Centric Networking," *IEEE Wireless Communications*, 2019. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8685777

[88] M. Zaher, A. H. Alawadi, and S. Molnar, "Class-based Flow Scheduling Framework in SDN-based Data Center Networks," in *2020 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*. IEEE, aug 2020, pp. 51–56. [Online]. Available: https://ieeexplore.ieee.org/document/9231052/

[89] W. Sun, Z. Wang, and G. Zhang, "A QoS-guaranteed intelligent routing mechanism in software-defined networks," *Computer Networks*, vol. 185, no. November 2020, p. 107709, feb 2021. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1389128620313050

[90] S. Tomovic and I. Radusinovic, "Toward a Scalable, Robust, and QoS-Aware Virtual-Link Provisioning in SDN-Based ISP Networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 1032–1045, sep 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8764418/

[91] T. Feng, J. Bi, and K. Wang, "Joint Allocation and Scheduling of Network Resource for Multiple Control Applications in SDN," *China Communications*,

vol. 12, no. 6, pp. 85–95, 2014.

[92] T. Feng, "Allocation and scheduling of network resource for multiple control applications in SDN," *China Communications*, vol. 12, no. 6, pp. 85–95, jun 2015. [Online]. Available: http://ieeexplore.ieee.org/document/7122483/

[93] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven WAN," *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM - SIGCOMM '13*, p. 15, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2486001.2486012

[94] K. T. Bagci and A. M. Tekalp, "Dynamic resource allocation by batch optimization for value-added video services over SDN," *IEEE Transactions on Multimedia*, vol. 20, no. 11, pp. 3084–3096, nov 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8332510/

[95] J. M. Wang, Y. Wang, X. Dai, and B. Bensaou, "SDN-Based Multi-Class QoS Guarantee in Inter-Data Center Communications," *IEEE Transactions on Cloud Computing*, vol. 7, no. 1, pp. 116–128, 2019. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7299623/computer.org/tcc

[96] D. Eppstein, "Finding the k shortest paths," *SIAM Journal on computing*, vol. 28, no. 2, pp. 652–673, 1998.

[97] N. Saha, S. Bera, and S. Misra, "Sway: Traffic-Aware QoS Routing in Software-Defined IoT," p. 1, 2018. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8385144/

[98] M. Capelle, S. Abdellatif, M. J. Huguet, and P. Berthou, "Online virtual links resource allocation in Software-Defined Networks," *Proceedings of 2015 14th IFIP Networking Conference, IFIP Networking 2015*, 2015.

[99] R. Trivisonno, R. Guerzoni, I. Vaishnavi, and A. Frimpong, "Network resource management and qos in sdn-enabled 5g systems," in *2015 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2015, pp. 1–7.

[100] M. T. Thi, T. Huynh, M. Hasegawa, and W. J. Hwang, "A Rate Allocation Framework for Multi-Class Services in Software-Defined Networks," *Journal of Network and Systems Management*, vol. 25, no. 1, pp. 1–20, 2017.

[101] M. M. Tajiki, B. Akbari, and N. Mokari, "Qrtp: Qos-aware resource reallocation based on traffic prediction in software defined cloud networks," in *2016 8th International Symposium on Telecommunications (IST)*. IEEE, 2016, pp. 527–532.

[102] M. Tajiki, B. Akbari, M. Shojafar, and N. Mokari, "Joint QoS and Congestion Control Based on Traffic Prediction in SDN," *Applied Sciences*, vol. 7, no. 12, p. 1265, 2017. [Online]. Available: http://www.mdpi.com/2076-3417/7/12/1265

[103] S. D'Oro, L. Galluccio, P. Mertikopoulos, G. Morabito, and S. Palazzo, "Auction-based resource allocation in openflow multi-tenant networks," *Computer Networks*, vol. 115, pp. 29–41, 2017.

[104] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, vol. 1, p. 132, 2009.

[105] C. Xu, B. Chen, and H. Qian, "Quality of Service Guaranteed Resource Management Dynamically in Software Defined Network," *Journal of Communications*, vol. 10, no. 11, pp. 843–850, 2015. [Online]. Available: http://www.jocm.us/index.php?m=content&c=index&a=show&catid=153&id=883

[106] R. M. Abuteir, A. Fladenmuller, and O. Fourmaux, "An SDN approach to adaptive Video Streaming in Wireless home networks," in *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, sep 2016, pp. 321–326. [Online]. Available: http://ieeexplore.ieee.org/document/7577078/

[107] T.-N. Lin, Y.-M. Hsu, S.-Y. Kao, and P.-W. Chi, "OpenE2EQoS: Meter-based method for end-to-end QoS of multimedia services over SDN," in *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. IEEE, sep 2016, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/7794972/

[108] W. Miao, G. Min, Y. Wu, and H. Wang, "Performance Modelling of Preemption-Based Packet Scheduling for Data Plane in Software Defined Networks," *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, pp. 60–65, 2015. [Online]. Available: http://ieeexplore.ieee.org/document/7463702/

[109] G. S. Aujla, R. Chaudhary, N. Kumar, R. Kumar, and J. J. P. C. Rodrigues, "An Ensembled Scheme for QoS-Aware Traffic Flow Management in Software Defined Networks," in *2018 IEEE International Conference on Communications (ICC)*, vol. 2018-May. IEEE, may 2018, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/document/8422596/

[110] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S.-J. Lee, and P. Yalagandula, "Automated and scalable QoS control for network convergence," *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, p. 1, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863133.1863134

[111] J. L. Chen, Y. W. Ma, H. Y. Kuo, C. S. Yang, and W. C. Hung, "Software-Defined Network Virtualization Platform for Enterprise Network Resource Management," *IEEE Transactions on Emerging Topics in Computing*, vol. 4, no. 2, pp. 179–186, 2016.

[112] M. Afaq, S. U. Rehman, and W. C. Song, "Visualization of Elephant Flows and QoS Provisioning in SDN-Based Networks," *17th Asia-Pacific Network*

Operations and Management Symposium: Managing a Very Connected World, APNOMS 2015, vol. 65, no. Iccmit, pp. 444–447, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.procs.2015.09.011

[113] C. Bi, X. Luo, T. Ye, and Y. Jin, "On precision and scalability of elephant flow detection in data center with SDN," in *2013 IEEE Globecom Workshops, GC Wkshps 2013*, 2013, pp. 1227–1232. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6825161/

[114] Z. Wang, C. Zhou, Y. Yu, X. Shi, X. Yin, and J. Yao, *Fast detection of heavy hitters in software defined networking using an adaptive and learning method*, 2018, vol. 11065 LNCS. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-00012-7{_}5

[115] J. Liu, J. Li, G. Shou, Y. Hu, Z. Guo, and W. Dai, "SDN based load balancing mechanism for elephant flow in data center networks," in *International Symposium on Wireless Personal Multimedia Communications, WPMC*, vol. 2015-Janua, 2015, pp. 486–490. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7014867/

[116] X. Huang, T. Yuan, M. Ma, and P. Zhang, "Utility-Based Network Bandwidth Allocation in the Hybrid SDNs," in *2017 IEEE Global Communications Conference, GLOBECOM 2017 - Proceedings*, vol. 2018-Janua, 2017, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8254751/

[117] S. Bashir and N. Ahmed, "VirtMonE: Efficient detection of elephant flows in virtualized data centers," in *25th International Telecommunication Networks and Applications Conference, ITNAC 2015*, 2015, pp. 280–285. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7366826/

[118] R. B. Basat, "Optimal Elephant Flow Detection," 2017.

[119] Z. Liu, D. Gao, Y. Liu, and H. Zhang, "An enhanced scheduling mechanism for elephant flows in SDN-based data center," in *IEEE Vehicular Technology Conference*, 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7880900/

[120] C. Xing, K. Ding, C. Hu, and M. Chen, "Sample and Fetch-Based Large Flow Detection Mechanism in Software Defined Networks," *IEEE Communications Letters*, vol. 20, no. 9, pp. 1764–1767, 2016. [Online]. Available: https://ieeexplore.ieee.org/iel7/4234/5534602/07501550.pdf

[121] P. Xiao, W. Qu, H. Qi, Y. Xu, and Z. Li, "An Efficient Elephant Flow Detection with Cost-Sensitive in SDN," in *Proceedings of the 1st International Conference on Industrial Networks and Intelligent Systems*, 2015. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7157818/ http://eudl.eu/doi/10.4108/icst.iniscom.2015.258274

[122] L. Yang, B. Ng, and W. K. Seah, "Heavy hitter detection and identification in software defined networking," in *2016 25th International Conference on*

Computer Communications and Networks, ICCCN 2016, 2016. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7568527/

[123] B. Wang and J. Su, "A survey of elephant flow detection in SDN," in *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*, 2018, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8355352/

[124] A. Adegboyega, "An Adaptive Resource Provisioning Scheme for Effective QoS Maintenance in the IaaS Cloud," *Proceedings of the International Workshop on Virtualization Technologies*, pp. 2:1—-2:6, 2011. [Online]. Available: http://doi.acm.org/10.1145/2835075.2835078

[125] J. Bendriss, I. G. Ben Yahia, and D. Zeghlache, "Forecasting and anticipating SLO breaches in programmable networks," in *Proceedings of the 2017 20th Conference on Innovations in Clouds, Internet and Networks, ICIN 2017*. IEEE, mar 2017, pp. 127–134. [Online]. Available: http://ieeexplore.ieee.org/document/7899402/

[126] J. Bendriss, I. Yahia, P. C. D. .-D. of …, and U. 2017, "AI for SLA management in programmable networks," *ieeexplore.ieee.org*, vol. 2017, pp. 130–137, 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7993445/

[127] A. V. Akella and K. Xiong, "Quality of service (qos)-guaranteed network resource allocation via software defined networking (sdn)," in *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*. IEEE, 2014, pp. 7–13.

[128] S. K. Singh, T. Das, and A. Jukan, "A survey on internet multipath routing and provisioning," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2157–2175, 2015.

[129] C. C. Marquezan, X. An, Z. Despotovic, R. Khalili, and A. Hecker, "Identifying latency factors in SDN-based Mobile Core Networks," in *2016 IEEE Symp. Comput. Commun.*, vol. 2016-Augus. IEEE, jun 2016, pp. 484–491. [Online]. Available: http://ieeexplore.ieee.org/document/7543785/

[130] Mininet, "Mininet openflow virtual network," available at mininet.org/, Accessed On 2020-08-23. [Online]. Available: mininet.org/

[131] ONF, "OpenFlow Switch Specification," available at https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf/, Accessed On 2020-08-23. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf/

[132] Sprint, "Sprint IP Network," available at https://www.sprint.net/network_maps.php, Accessed On 2020-08-23. [Online]. Available: https://www.sprint.net/network{_}maps.php

[133] OpenVSwitch, "Open Virtual Switch," available at openvswitch.org/, Accessed On 2020-08-23. [Online]. Available: openvswitch.org/

[134] iPerf, "iPerf: The ultimate speed test tool for TCP, UDP and SCT," available at https://iperf.fr/, Accessed On 2020-08-23. [Online]. Available: https://iperf.fr/

[135] D. E. Henni, A. Ghomari, and Y. Hadjadj-Aoul, "A consistent qos routing strategy for video streaming services in sdn networks," *International Journal of Communication Systems*, vol. 33, no. 10, p. e4177, 2020.

[136] M. Bezahaf, D. Hutchison, D. King, and N. Race, "Internet Evolution: Critical Issues," *IEEE Internet Comput.*, vol. 6, pp. 1–1, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9113495/

[137] M. R. Parsaei, H. R. Boveiri, R. Javidan, and R. Khayami, "Telesurgery qos improvement over sdn based on a type-2 fuzzy system and enhanced cuckoo optimization algorithm," *International Journal of Communication Systems*, vol. 33, no. 11, p. e4426, 2020.

[138] R. Chaudhary, G. S. Aujla, S. Garg, N. Kumar, and J. J. Rodrigues, "Sdn-enabled multi-attribute-based secure communication for smart grid in iiot environment," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 6, pp. 2629–2640, 2018.

[139] J. W. Guck, A. Van Bemten, and W. Kellerer, "DetServ: Network models for real-time QoS provisioning in SDN-based industrial environments," *IEEE Trans. Netw. Serv. Manag.*, vol. 14, no. 4, pp. 1003–1017, 2017.

[140] S. Garg, K. Kaur, N. Kumar, and J. J. Rodrigues, "Hybrid deep-learning-based anomaly detection scheme for suspicious flow detection in sdn: A social multimedia perspective," *IEEE Transactions on Multimedia*, vol. 21, no. 3, pp. 566–578, 2019.

[141] E. Akin and T. Korkmaz, "Comparison of Routing Algorithms With Static and Dynamic Link Cost in Software Defined Networking (SDN)," *IEEE Access*, vol. 7, pp. 148 629–148 644, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8864963/

[142] Y. Xiao, K. Thulasiraman, X. Fang, D. Yang, and G. Xue, "Computing a most probable delay constrained path: NP-hardness and approximation schemes," *IEEE Transactions on Computers*, vol. 61, no. 5, pp. 738–744, 2012.

[143] A. Frangioni, L. Galli, and M. G. Scutellà, "Delay-Constrained Shortest Paths: Approximation Algorithms and Second-Order Cone Models," *J. Optim. Theory Appl.*, vol. 164, no. 3, pp. 1051–1077, 2015.

[144] J. Zhao, E. Hammad, A. Farraj, and D. Kundur, "Network-aware QoS routing for smart grids using software defined networks," in *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, vol. 166, sep 2016, pp. 384–394. [Online]. Available: http://dx.doi.org/10.1080/01422419908228843

[145] A. Van Bemten, J. W. Guck, P. Vizarreta, C. M. Machuca, and W. Kellerer, "LARAC-SN and Mole in the Hole: Enabling Routing through Service Function Chains," *2018 4th IEEE Conf. Netw. Softwarization Work. NetSoft 2018*, no. NetSoft, pp. 168–176, 2018.

[146] H. E. Egilmez, S. Civanlar, and A. M. Tekalp, "A distributed QoS routing architecture for scalable video streaming over multi-domain OpenFlow

networks," in *2012 19th IEEE International Conference on Image Processing*. IEEE, sep 2012, pp. 2237–2240. [Online]. Available: http://ieeexplore.ieee.org/document/6467340/

[147] L. Santos, J. Coutinho-Rodrigues, and J. R. Current, "An improved solution algorithm for the constrained shortest path problem," *Transp. Res. Part B Methodol.*, vol. 41, no. 7, pp. 756–771, 2007.

[148] H. Agrawal, M. Grah, and M. Gregory, "Optimization of QoS routing," *Proc. - 6th IEEE/ACIS Int. Conf. Comput. Inf. Sci. ICIS 2007; 1st IEEE/ACIS Int. Work. e-Activity, IWEA 2007*, no. Icis, pp. 598–602, 2007.

[149] K. Stachowiak, J. Weissenberg, and P. Zwierzykowski, "Lagrangian relaxation in the multicriterial routing," *IEEE AFRICON Conf.*, vol. 0, no. September, pp. 13–15, 2011.

[150] M. Ben Attia, K. K. Nguyen, and M. Cheriet, "QoS-aware software-defined routing in smart community network," *Comput. Networks*, vol. 147, pp. 221–235, 2018. [Online]. Available: https://doi.org/10.1016/j.comnet.2018.10.015

[151] Z. Liu, G. Xu, P. Liu, X. Fu, and Y. Liu, "Energy-Efficient Multi-User Routing in a Software-Defined Multi-Hop Wireless Network," *Future Internet*, vol. 11, no. 6, p. 133, jun 2019. [Online]. Available: https://www.mdpi.com/1999-5903/11/6/133

[152] C. Kou, D. Hu, J. Yuan, and W. Ai, "Bisection and Exact Algorithms Based on the Lagrangian Dual for a Single-Constrained Shortest Path Problem,"

IEEE/ACM Trans. Netw., vol. 28, no. 1, pp. 224–233, feb 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8935448/

[153] K. Stachowiak and P. Zwierzykowski, "Lagrangian Relaxation and Linear Intersection Based QoS Routing Algorithm," *Int. J. Electron. Telecommun.*, vol. 58, no. 4, pp. 307–314, dec 2012. [Online]. Available: http://journals.pan.pl/dlibra/publication/101207/edition/87224/content

[154] R. W. Floyd, "Algorithm 97: shortest path," *Communications of the ACM*, vol. 5, no. 6, p. 345, 1962.

[155] N. Varyani, Z. L. Zhang, M. Rangachari, and D. Dai, "LADEQ: A fast lagrangian relaxation based algorithm for destination-based QoS routing," *2019 IFIP/IEEE Symp. Integr. Netw. Serv. Manag. IM 2019*, pp. 462–468, 2019.

[156] P. Schulz, M. Matthe, H. Klessig, M. Simsek, G. Fettweis, J. Ansari, S. A. Ashraf, B. Almeroth, J. Voigt, I. Riedel, A. Puschmann, A. Mitschele-Thiel, M. Muller, T. Elste, and M. Windisch, "Latency Critical IoT Applications in 5G: Perspective on the Design of Radio Interface and Network Architecture," *IEEE Commun. Mag.*, vol. 55, no. 2, pp. 70–78, 2017.

[157] Y. Li, Z.-P. Cai, and H. Xu, "Llmp: exploiting lldp for latency measurement in software-defined data center networks," *Journal of Computer Science and Technology*, vol. 33, no. 2, pp. 277–285, 2018.

[158] R. B. Santos, T. R. Ribeiro, and C. D. A. Cesar, "A network monitor and controller using only OpenFlow," in *LANOMS 2015 - 8th Latin American*

Network and Operations Management Symposium, 2015, pp. 9–16. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7332663/

[159] D. Sinha, K. Haribabu, and S. Balasubramaniam, "Real-time monitoring of network latency in software defined networks," in *2015 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS)*. IEEE, 2015, pp. 1–3.

[160] V. Altukhov and E. Chemeritskiy, "On real-time delay monitoring in software-defined networks," in *2014 International Science and Technology Conference (Modern Networking Technologies)(MoNeTeC)*. IEEE, 2014, pp. 1–6.

[161] M. Selmchenko, M. Beshley, O. Panchenko, and M. Klymash, "Development of monitoring system for end-to-end packet delay measurement in software-defined networks," in *2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET)*. IEEE, 2016, pp. 667–670.

[162] K. Phemius and M. Bouet, "Monitoring latency with OpenFlow," in *2013 9th International Conference on Network and Service Management, CNSM 2013 and its three collocated Workshops - ICQT 2013, SVM 2013 and SETM 2013*, 2013, pp. 122–125.

[163] Q. He and S. Wang, "A Low-Cost Measurement Framework in Software Defined Networks," *International Journal of Communications, Network and System Sciences*, vol. 10, no. 05, pp. 54–66, 2017. [On-

line]. Available: http://file.scirp.org/pdf/IJCNS{_}2017052709354034.pdfhttp://www.scirp.org/journal/doi.aspx?DOI=10.4236/ijcns.2017.105B006

[164] A. M. Allakany and K. Okamura, "Latency monitoring in software-defined networks," in *Proceedings of the 12th International Conference on Future Internet Technologies*, 2017, pp. 1–4.

[165] N. L. Van Adrichem, C. Doerr, and F. A. Kuipers, "Opennetmon: Network monitoring in openflow software-defined networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–8.

[166] S. Ramanathan, Y. Kanza, and B. Krishnamurthy, "SDProber: A Software Defined Prober for SDN," *Proceedings of the Symposium on SDN Research - SOSR '18*, no. March, pp. 1–7, 2018. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3185467.3185472

[167] W. Zhang, X. Zhang, H. Shi, and L. Zhou, "An efficient latency monitoring scheme in software defined networks," *Future Generation Computer Systems*, vol. 83, pp. 303–309, 2018.

[168] S. Rowshanrad, S. Namvarasl, and M. Keshtgari, "A queue monitoring system in openflow software defined networks," *Journal of Telecommunications and Information Technology*, vol. 2017, no. 1, pp. 39–43, 2017.

VITAE

- Name: Ahmed Saeed Ahmed BinSahaq
- Nationality: Yemeni
- Date of Birth: 11 June 1986
- Email: *ishaq1900@gmail.com*
- Phone: +966533899293
- Permanent Address: Al-Qatn, Hadhramout, Yemen

Publication

- Binsahaq, Ahmed, Tarek R. Sheltami, and Khaled Salah. "A survey on autonomic provisioning and management of QoS in SDN networks." IEEE Access 7 (2019): 73384-73435.
- A. BinSahaq, T. Sheltami, A. Mahmoud, N. Nasser, "Bootstrapped LARAC Algorithm for Fast Delay-Sensitive QoS Provisioning in SDN Networks", International Journal of Communication Systems, 25 April 2021, (accepted)
- Ahmed Binsahaq, Tarek R. Sheltami, Ashraf Mahmoud and Khaled Salah., "Multi-Class Group-based Resource Allocation and Refinement for QoS Provisioning in SDN Networks", Elsevier Computer Networks , 2021, (to be submitted)