QoS-BASED TRAFFIC ENGINEERING IN SOFTWARE DEFINED NETWORKING

NWE THAZIN

UNIVERSITY OF COMPUTER STUDIES, YANGON

November, 2019

QoS-based Traffic Engineering in Software Defined Networking

Nwe Thazin

University of Computer Studies, Yangon

A thesis submitted to the University of Computer Studies, Yangon in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

November, 2019

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

…..…….…….…… ……..……

Date Nwe Thazin

ACKNOWLEDGEMENTS

I would like to express my gratitude to all who get this deep and precious research work done.

First of all, I would like to thank Ministry of Education for giving me the opportunity to study PhD Course by providing support that allowed me to perform the research in University of Computer Studies, Yangon, Myanmar.

Secondly, I especially thanks to Dr. Mie Mie Thet Thwin, Rector, the University of Computer Studies, Yangon, for allowing me to develop this thesis and giving me general guidance during the period of my study.

I would like to express my deepest gratitude to my supervisor, Professor. Dr. Khine Moe Nwe, Course-coordinator of the Ph.D. 9th Batch, the University of Computer Studies, Yangon, for her excellent guidance, for patience supervision, motivation, and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis. My Ph.D. study experience could not be considered as a complete one without her support.

Besides, I would also like to extend my special appreciation to Professor. Dr. Yutaka Ishibashi, Nagora Institute of Technologies, for the continuous support of my Ph.D. study and related research, and providing me with excellent ideas throughout the study of this thesis.

I would like to express my respectful gratitude to all my teachers for not only their insightful comments and encouragement but also for their challenging questions, which give incentives to me to widen my research from various perspectives. My thanks go to Daw Aye Aye Khine, Associate Professor, Department of English, for valuable supports and editing my thesis from the language point of view.

I would like to express my sincere gratitude to Dr. Zin May Aye, Professor, Head of Cisco Network Lab, the University of Computer Studies, Yangon, for her caring, for the useful comments and advice, and insight which are invaluable to me. I thank my fellow lab mates for the stimulating discussions, for the memorable time we were working together before deadlines, and for all the fun we have had in the last five years. I also thank all of my friends from the Ph.D.9th Batch for their co-operation and encouragement. I am very grateful for everything they shared with me and helped me to strive towards my goal.

Last but not least, I would like to express special thanks to my family. Words cannot express how grateful I am to my grand-mother, grand-father, and my little sister, for all of the sacrifices that you have made on my behalf. They are always supportive of me during my period of studies, especially for this Doctorate Course.

ABSTRACT

Quality of Service (QoS) is the overall performance of a computer network, particularly the performance seen by the users of the network. By managing the delay, bandwidth, and packet loss parameters, it allows us to allocate our available resources between applications in a reasonable way. In today's life, it is general that many applications run at the same time. As a critical type of resource, bandwidth would be shared without principle which leads to interference. Therefore, “important” applications cannot get enough bandwidth to transmit data. To solve these problems, the end-to-end bandwidth resource allocation scheme is proposed to support Quality of Service (QoS) for various types of traffic based on the user QoS demand.

The main goal of this system is to provide high QoS performance for high priority flows. In order to meet this goal, this system considers the flow priority and the dynamic characteristics of the network. In addition, feasible paths are calculated for all the traffic flows that can satisfy the user bandwidth demands. In order to mitigate the flow performance degradation and congestion, the controller checks the link bandwidths by reserving the required bandwidths for incoming flows. If the link bandwidth is smaller than the predefined threshold value, the link is defined as the bottleneck link and reroute the highest priority flow from the bottleneck link to an alternative link that has enough bandwidth for the rerouting flow. Furthermore, to improve the performance and to ensure the QoS of the high priority flows, a queue mechanism is used which is provided by the OpenFlow at the data link level. This research will try to accommodate as much traffic as possible, and study the effect of routing on a rather general mix of QoS traffic types.

The effectiveness of the proposed scheme is described on the emulated SDN network. The proposed scheme is compared with the conventional shortest path scheme, multipath routing scheme in the various network topology. Furthermore, the performance of the proposed scheme is compared with the popular flow scheduling scheme Hedera in the data center network topology. The improvement of QoS traffic type is quantified in terms of throughput, delay, jitter and packet loss rate, respectively. Based on the experiments, the researcher observed that the proposed method (QoS_based Traffic Engineering Approach) QT offers a significant improvement compared to a static, traditional IP network and the multipath network environment by

providing better performance in terms of packet loss rate for the QoS traffics and great improvement in link utilization.

TABLES OF CONTENTS

LIST OF FIGURE

LIST OF TABLES

LIST OF EQUATIONS

CHAPTER 1

INTRODUCTION

With the rapid growth of network technologies, the development of the domain is shifting from providing connectivity to providing a number of services and applications with desirable quality and reliability. These applications and services have different service features and their own quality demand. For example, telephony services like VoIP need extremely bandwidth and delay-sensitive. The packet needs to reach its destination before a specific delay threshold, otherwise, the service becomes useless for the VoIP transmissions. Furthermore, the retransmission of lost packets is also worthless for real-time applications. On the other hand, the data transfer applications like File Transfer Protocol (FTP) services are more robust in packet loss than real-time applications. Therefore, implementing a QoS-enable network becomes a very hard challenge and it requires a significant effort to provide accessible performance for all traffic types. In general, network providers offer Service-Level Agreements (SLAs) with guarantees on different network performance metrics such as bandwidth and delay. SLAs express the availability of the network function with percentage and provide the required quality assurance of applications such as delay and/or packet loss.

Quality of Service (QoS) is defined by Cisco [104] as “the capability of a network to provide better service to selected network traffic over various technologies with the primary goal to provide priority including dedicated bandwidth, controlled jitter and latency, and improved loss characteristics.” QoS deals with providing end-to-end guarantees to the users. There are many ways in which such assurances can be obtained. One can use one or any combinations of these technologies to implement QoS. A network operating system may exploit various services like resource reservation and allocation, prioritized scheduling, queue management, routing, etc.

The traditional network was not initially designed with QoS in mind, it was later supplemented by many techniques to achieve the desired performance tuning. These techniques allowed the Internet Service Providers (ISP) to fine-tune the internet as required. However, the traditional internet is facing new challenges with every new emerging technology. The increasing number of devices, the growing volume and velocity of traffic, big data and cloud computing are some of the problems that the traditional internet is finding hard to cater to.

Software Defined Networking (SDN) provides a solution to these challenges by making the internet flexible and programmable. SDN [97] is an emerging architecture that may play a critical role in future network architectures. SDN can provide a global network view of the network resources and their performance indicators such as link utilization and the network congestion level. The main idea of SDN is to separate the network intelligence from the forwarding device and logically place it in the external entity which is called the controller. OpenFlow protocol (OF) [71] is used to exchange data between controller and forwarding devices in SDN architecture. In the data plane, the simple packet forwarding elements match the incoming flows in the flow table and apply the specified action on the matching flows. The control plane lies above the data plane, and this is where all the routing decisions are made. Once a decision is reached for a "family" of flow, it is updated to all the flow tables, thus saving time for subsequent flows. In other words, the networking decisions are now taken in software rather than hardware. Due to its flexibility and responsive to rapid changes, SDN is proper for an emerging technology like 5G and cloud data center network.

As more and more vendors are accepting SDN as the new networking paradigm, the demands on SDN is changing with time. One of the biggest advantages of SDN is its vendor-agnostic and open-source nature which has led to rapid acceptance and involvement of research communities worldwide, both in academia as well as the industry. The primary solution that SDN provided over the traditional internet was that of flexibility and programmability. This means that, for SDN to be deployed on the worldwide internet, it needs to support fine-grained QoS, equivalent to what is possible in the traditional internet, if not better. By having strong QoS control included in SDN, the future Internet might have native QoS support.

This chapter highlights prominent research challenges in SDN and presents the objectives of the present research along with a description of the research structure. The remainder of this chapter is organized as follows. Section 1.1 introduces the problem statement in SDN. The motivation of the research is highlighted in section 1.2. In section 1.3, the objectives of the research are presented. Section 1.4 describes the research contributions. The organization of this thesis is presented in section 1.5.

1.1 Problem Statement

According to the traditional single-path routing scheme, all of the traffic shares the same link and compete over the network link bandwidth. Congestion happens when the traffic load exceeds the network link bandwidth, and it can seriously impact on the Quality of Service (QoS) parameters of the applications. For example, when the network faced packet loss as a consequence of congestion, the packet transmission rate drastically reduced which may negatively influence the quality of the provided services and lowers the network throughput. On the other hand, there may be more than a single path to reach a particular destination, and some paths may be underutilized. Suitable path selection from among the multiple paths to optimize the overall network performance is one of the critical issues in the network area. However, the routing behavior in traditional networks is rather static and cannot be altered programmatically on short notice. This makes it practically impossible to react to changing traffic characteristics.

Another major challenge is the dynamic link bandwidth allocation with congestion management that can support the QoS requirements for each traffic and improve the service degradation for high priority flows. Hence, increasing the bandwidth would not solve the problem. The reason for the losses can be found in the burst nature of the network traffic which causes congestion when multiple traffics flows transmitted on the same link produce high peaks simultaneously.

One more challenge of the current network architecture is to provide the satisfaction of network users (customers). From the network operator point of view, the network operator needs to use the available network resources and make sure the negotiated SLAs are still met.

From the above problem statements, the following questions for this research work can be gathered:

- How to provide bandwidth guarantee to QoS flow?
- How to improve the link utilization in SDN?
- How to steer the traffic to reduce network congestion while adhering to constraints given by QoS parameters for different service types?

It should be noted that these research questions are closely related to each other and all of them have the same destination. The main idea behind these research questions is

providing a performance guarantee for the high priority network traffic by supporting network resources according to their QoS demands and the current network status.

1.2 Motivation of the Research

The Quality of Service (QoS) demand for network applications is increasingly fast with global Internet traffic growing year per year. Providing higher bandwidth is just not enough because many of the applications not only have bandwidth requirements, but also require other QoS guarantees, such as end-to-end delay, jitter, or packet loss probability.

The traditional network architecture was designed to provide Best Effort (BE) service only. Therefore, it is not easy to provide a guarantee for requiring QoS performance in current traditional networks. In the last decades, the Internet Engineering Task Force (IETF) [81] has proposed two major QoS control architectures, i.e.: Integrated Service (IntServ) and Differentiated Service (DiffServ. The former one, IntServ [14] provides a way to deliver the end-to-end QoS solution. It uses bandwidth reservation and admission control at each network element. The IntServ uses a resource reservation system to ensures the portion of bandwidth reserved in every link for a particular flow. However, the Resource Reservation Protocol (RSVP) is not fit for a large network like the Internet due to its scalability problem. RSVP requires a periodic reservation-state refresh. In a large network with RSVP, each network component needs to store flow states, and it grows rapidly with the increasing number of flows and network components. Following, the reserved bandwidth can only be used by the reserving flow which may cause low bandwidth utilization in the network. On the other hand, the complexity of DiffServ [12] is significantly lower than IntServ and it works per-hop behavior (PHB) with aggregation for different classes of traffic. However, there are no QoS guarantees in this architecture since bandwidth is shared among flows. Both of these QoS control architectures have not been very successful or implemented on a wide scale since they need some fundamental changes in the current network design.

In order to provide QoS guarantees for a specific application, the traditional internet is facing a challenge to use network resources efficiently in allocating network traffic to limited network bandwidth. In traditional network, network resource allocation is partially implemented in the network components such as routers, switches, and links inside the network and partially in the transport protocol running on the end hosts. The

implementation is not isolated to one single level of a protocol hierarchy. Therefore, resource allocation and congestion control are known as complex issues in traditional network.

SDN has been approached by many researchers in these days to overcome the current Best-Effort limitations explained above. Hence, SDN can provide a global network view of the network resources and their performance indicators such as link utilization and the network congestion level which can help in network resource allocation. Leveraging the advantage of centralized control in SDN, network-wide monitoring, and flow-level scheduling can be used to achieve high QoS for network applications and services such as voice over IP, video conferencing and online gaming. By using the benefits of SDN, the controller makes the routing decision based on the global view of the network resources in the SDN network.

In general, network providers optimize their network performance in order to effectively fulfill as many customer demands as possible with traffic engineering (TE) [1]. An important goal of TE is to use the available network resources more efficiently for different types of load patterns in order to provide a better and more reliable service to customers. The traditional network architectures are not well-suited for developing sophisticated TE systems because they miss a set of desired properties. No entity in the network is able to easily collect statistical information from all network devices and to aggregate them to form a global view of the network which would allow a TE application to understand the current network situation and simplify the computation of routing paths.

Routing is a powerful tool of TE and it allows for controlling network data flows. The aim of TE routing is to route as many demands as possible by reserving the amount of bandwidth resources for each established route. For each traffic flow, the routing scheme needs to select a route between its source and destination along which sufficient resources are reserved to meet its require QoS. Generally, the main function of routing is to find the best path to reduce network congestion and improve the quality of service (QoS).

While QoS in SDN is still an area of research, it would not be wrong to believe that achieving them on SDN would be far easier than it was on the traditional internet considering its programmable nature. In very little time, SDN technology rapidly

developed and improved with the birth of a lot of independent projects working in different areas of SDN.

1.3 Objectives of the Research

This thesis aims to provide high performance of different network traffic by giving network resources to it according to their QoS demands and the current network status. The proposed system uses an OpenFlow network architecture which provides the ability to improve the link utilization while providing the requires bandwidth resources and less packet loss rate as the QoS factor in the overall network.

In order to achieve this, the research work is divided into the following distinct objectives.

1. To provide bandwidth guarantee to QoS flow by applying characteristics and advantages of SDN technologies
2. To propose the resource allocation scheme as a small part of QoS-based traffic engineering for SDN-based Network
3. To improve the link utilization while providing the required bandwidth resources
4. To reduce the packet loss rate of QoS flows and provide better performance

1.4 Contributions of the Research

In this research, the QoS demand approach is considered for end-to-end dynamic network bandwidth resource allocation in the SDN network by taking account into the QoS flow priority and the dynamic characteristics of the network link. The goal is to improve the QoS performance of the high priority flow while providing the required bandwidth resources and less packet loss rate as the QoS factor in the overall network.

There are three research contributions that have been assumed as a hierarchical level to create a QoS capable SDN network in this thesis which are summarized in the following:

- Proposing an end-to-end dynamic bandwidth resource allocation procedure based on TE in SDN to support the QoS requirements of different types of network traffic flows. The QoS guarantee is provided in both controller and data link level. Firstly, an admission process is performed to make sure that the QoS flows get enough bandwidth at the controller level. Then, the proposed system

uses the queue mechanism provided by the OpenFlow in the data link level to improve the performance and to ensure the QoS of the high priority flow.

- Implementing QoS routing for different types of QoS traffic class by taking advantage of SDN technology. The calculation of the feasible path for all traffic can satisfy the user demand bandwidth. Compare the difference routing strategies and showed (by simulation) that the QoS routing gives a substantial gain in all performance metrics, and was better than traditional and multipath routing.
- Implementing Congestion handling to handle the network congestion in case. To mitigate the flow performance degradation, detect the link bandwidth by reserving the required bandwidth for the incoming flow. If the link bandwidth satisfies the predefined threshold value, the proposed system defined it as the bottleneck link and reroute the highest priority flow from the bottleneck link to an alternated link that has enough bandwidth for the rerouting flow.

1.5 Organization of the Research

The structure of the dissertation is described as follows:

- Chapter 1 provides an introduction to the research. It presents a brief background and motivation behind the research and then presents the research questions addressed in this dissertation.
- Chapter 2 briefly introduces the SDN technologies and related research areas and then reviews the relevant literature.
- Chapter 3 presents the State of the Art of SDN and the required technologies background for implementing QoS in the SDN environment.
- Chapter 4 presents end-to-end QoS including the origin, progress, and challenges faced in SDN.
- Chapter 5 explains the architecture and internal details of the various components of the SDN framework that would be used in this dissertation.
- Chapter 6 presents the implementation of the proposed system and essential parts of the developed SDN application to give the reader an understanding of how the platform operates.

- Chapter 7 presents the various experiments conducted in this dissertation. It presents the statement of the experiment followed by its implementation, results, and observations.
- Chapter 8 concludes with the challenges faced and contributions made in this research. It ends with a discussion on future work in this direction.

CHAPTER 2

LITERATURE REVIEW AND RELATED WORK

This chapter presents the traditional network, the SDN network with its application area, and the traffic engineering domain problems exposed in the existing SDN architecture. The first section presents the traditional network's components and its challenges. The second section covers a brief overview of SDN and its advantages over the traditional network. The third section starts with a focus on the application areas of SDN. Finally, the last section reviews the related research studies and its current work.

2.1 Traditional Network

Ethernet switch is one of the most commonly used network elements to serve as the network connection point for hosts in Local Area Networks (LANs). It uses hardware addresses, MAC addresses, to forward the frame at the data link layer of the (Open Systems Interconnection) OSI model. The switch operates at the data link layer of the OSI model to create a separate collision domain of every switch interface. Each network element connected to a switch interface can transmit and receive the data simultaneously.

The switch forwards data frames based on the Media Access Control (MAC) table. When a frame arrives at a switch, the switch will put the source MAC address and correspond incoming interface number in the MAC table as the basis for forwarding new frames. Then the destination MAC address will be inspected. If a switch not having an entry for the destination MAC address in its table, it floods all of its interfaces with a broadcast message requesting the location of the MAC address. Each connected switch relays this broadcast message to all of its neighbors until eventually a switch replies. This reply is traced back to the original switch that initially requested the location of the MAC address and MAC tables are updated by each switch along the way to reflect the newly discovered MAC address.

If the destination MAC address is a multicast address or unknown unicast, it will forward the frame to all the interfaces except the incoming interface. Otherwise, the frame will be forwarded to the specific interface according to the MAC table. When the

switch floods a frame to the network, it may create the traffic loop in the network whose topology consists of loops. To solve this, the legacy switch usually uses a spanning tree protocol that blocks some interfaces so that the resulting logical LAN topology is a tree. Through the spanning tree protocol, traffic loops can be prevented.

2.1.1 Challenges of Traditional Networks

Mobile devices and their contents, cloud computing, and virtualization, have highlighted the need for a new network architecture that the industry is trying to satisfy. This change was necessary because the old network architecture was based on hierarchy, which was built on tiers of Ethernet switches arranged in a tree structure. This kind of architecture was unsuitable for nowadays dynamic computing and storage needs of enterprise data centers, campuses and carrier environments are not satisfied. All this is a challenge for traditional networks.

Traditional networks were static in nature and were manually configured based on service requests. Thus will make it challenging to control the network in both their management and their operation. Traditional networking functions are mainly implemented in dedicated networking devices such as switches, routers, and application delivery controllers. As for network management, networking devices have to be configured on a per-device basis using vendor-specific proprietary interfaces. While network administrators need to define high-level policies and apply them over the whole network, these interfaces only allow low-level configuration of individual devices. And although tools for centralized management exist, they serve rather for monitoring of the network than for its configuration as a whole.

Concerning network operation, typical networking devices use routing protocols to fill their forwarding tables, but may also allow for network administrators to manually configure additional rules. These rules may, for example, provide application port filtering or different treatment for particular quality-of-service classes. Unfortunately, there is no protocol to automatically distribute these more complex policies over the network [29].

With packet forwarding based only on destination addresses or statically defined rules, the network cannot react to the dynamics of the traffic or to the occurring abnormalities. Be it peak loads, applications with high demands for Quality of Service, or applications requiring high bandwidth, with the static setting, the network has no

instruments to appropriately utilize its resources unless equipped with specialized devices like load balancers.

To be fitted for demands of modern deployments, in both campuses and data centers, a network should provide means for automation, so it could react to occurring events on its own and efficiently use available resources while ensuring resilience. Such a network should also be virtualized in order to provide a high-level abstraction for convenient management of the network regardless of the underlying physical layer and its specifics [29].

Software-defined networking, further described in the next session, is assured to bring a solution to the various problems networking is facing today.

2.2 Software-Defined Networking

A new networking architecture called Software-Defined Networking (SDN) emerged in the early 2010s. The essence of SDN is changing the way of the network to change the way of creating and managing the network. The main characteristic of the SDN architecture is that decoupling the control plane and data plan and abstracting from each other. A consequence of this decoupling is lead to greater flexibility in network management. Since networking devices such as switches and routers simply act as the data plane's forwarding devices, the controller at the control plane centrally controls the devices by determining the forwarding rules according to the network condition. Unlike traditional traffic management, instead of managing network traffic at the per-hop level from one host to the next, SDN works with the flow level from source to destination. The controller determines the path and reserves the resources from source to destination when a flow is admitted. This design leads to global views and management of network traffic, which makes it possible to coordinate QoS guarantees and forwarding decisions at larger scales. Overall, SDN makes network management much simpler by providing dynamic reconfiguration of network devices in the data plane and by centralizing flow coordination in a network to minimize resource contention.

2.2.1 Differences between Traditional Networking and SDN

In the traditional network model, the control plane and the data plane are bundled inside the networking devices. The data plane tells the incoming traffic where it needs

to go. The location of the control plane is particularly inconvenient because administrators do not have easy access to dictate the traffic flow.

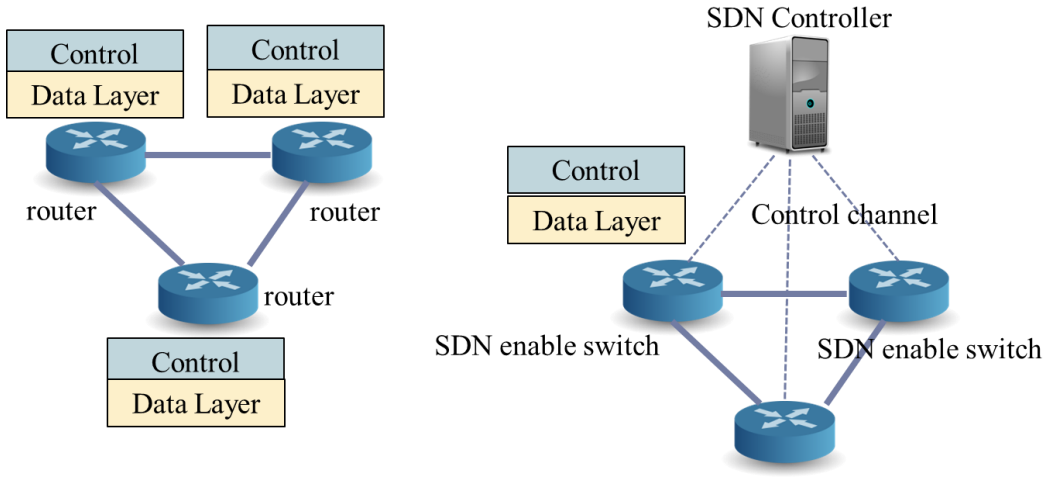


Figure 2.1 Architecture of Traditional Network and SDN Network

In the SDN network model, SDN breaks the vertical integration of the control plane and data plane by separating the network's control logic from the underlying routers and switches that forward the traffic. As a consequence, network switches become simple forwarding devices and the control logic is implemented in a logically centralized controller simplifying policy enforcement and network configuration. Figure 2.1 depicted the comparison of traditional network and SDN network architecture. Three main differences between traditional networking and SDN architecture are as follows:

- SDN controller has a northbound interface to communicate with applications through application programming interfaces (APIs). This allows application developers to program the network directly while traditional networking works through using protocols [46].
- To establish connections and run properly, traditional networking relies on physical infrastructure. Meanwhile, SDN is a software-based network, which allows the network users to control virtual-level network resource allocation via the control plane and to determine network paths and proactively configure network services.
- In traditional networking, the control plane is located in a switch or router, which is particularly inconvenient for the administrators to access it to order the traffic flow. Compared with the traditional networking, SDN has more ability to

communicate with devices throughout the network. SDN offers administrators the right to control traffic flow from a centralized user interface and allows resources provisioning from a centralized location. It virtualizes the entire network and gives users more control over their network capabilities.

2.2.2 Advantages of SDN

The rapid development of new trends in networking, such as server virtualization, cloud services, a vast diversity of mobile data applications, etc., determines the need for new network architectures to manage flexibly with the changing environment. The new emerging technology, SDN has several advantages over the traditional network. The most common specific advantages of SDN are as follows:

- **OPEX reduction:** centralized control helps to eliminate manual interaction with the hardware, improving the uptime of the network.
- **CAPEX reduction:** separating the data plane of the control plane brings to simpler hardware and increases the possibility of more competence between hardware manufacturers since the devices do not depend on the proprietary software.
- **Agility:** since the control layer can interact constantly with the infrastructure layer, the behavior of the network can adapt fast to changes like failures or new traffic patterns.
- **Flexibility:** having a separated abstraction for the control program allows us to express different operator goals, adapting to a specific objective. Operators can implement features in the software they control, rather than having to wait for a vendor to add it in their proprietary products.

The particular advantages of SDN will typically vary from network to network, however, there are benefits from network abstraction and the agility it offers for network administration and automation. The most ideal approach to take advantage of SDN is to assess the network components and infrastructure to decide whether SDN can help address issues, for example, resource availability, virtualization, and network security. The significant advantage of adopting the SDN is the world's largest networks such as Facebook, Yahoo, NTT Communication Deutsche Telekom, Microsoft, Google that have supported SDN based architecture.

2.3 SDN Applications Areas

Currently, SDN has found a great deal of applicability in a wide range of network application areas. SDN provides the opportunities for large scale network like data center network with the help of its real-time programmable framework. Moreover, mobile operators have also shown intense interest in bringing the technology to 5G/ LTE mobile networks to allow simplified yet rapid development and deployment of new services. Also, SDN is widely used by social networking websites (Facebook, Twitter, Google plus etc.) and large database search engines (Google, Yahoo, Ask etc.). The key applications area of SDN and some are highlighted in the following sub-sessions.

2.3.1 Traffic Engineering

Traffic engineering (TE) is a key networking area for measuring and managing the network traffic, designing reasonable routing mechanisms to guide network traffic for improving utilization of network resources, and meeting required quality of service (QoS) of the network. Therefore, the path control process through which the traffic is handled in TE. There are many reasons why network managers need to influence the characteristics of a path, one of them is the use of optimization of network resources. To optimize the network resources, network managers must try to avoid the situation of certain parts of congestion when others are underutilized. Another important reasons are to find the path with certain limitations-constraints that can support the proper performance for some high priority flows. For example, the path for the delay-sensitive flow like VoIP should not be long delay links. Through this process of TE new services are offered with extensive QoS guarantees and investments decline in new network resources such as bandwidth, by optimizing the use of existing ones.

The underlying network architecture is required on today's internet applications to be scalable for a large amount of traffic and to react in real-time. Also, the demand that the user has been also growing and the user now wants to be connected with everything, constantly. Moreover, each application and services generate their own characteristic flow and they shared the overall network bandwidth competitively. Therefore, the network architecture should be able to classify a diversity of network traffic types from different applications and to provide a suitable and particular service for each traffic type in a very short time period. It is not easy to make sure that to

efficiently handle and steer all those varieties of traffic types in order to meet their specific performance for each specific application. Therefore, new networking architectures with more intelligent and efficient TE tools are urgently needed.

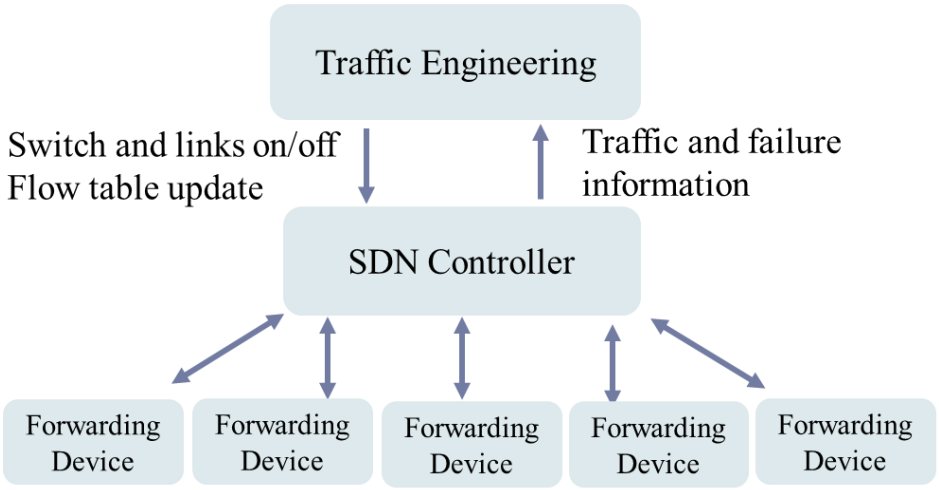


Figure 2.2 Traffic Engineering Architecture in SDN

Figure 2.2 illustrates the abstract view of TE architecture in the SDN network. Compared with the traditional networks, the TE mechanism in SDN can be much more efficient and intelligently implemented due to its distinguishing characteristics. More specifically, SDN provides the concept of decoupling between control and forwarding plane, the programmability of network behavior, and global centralized control.

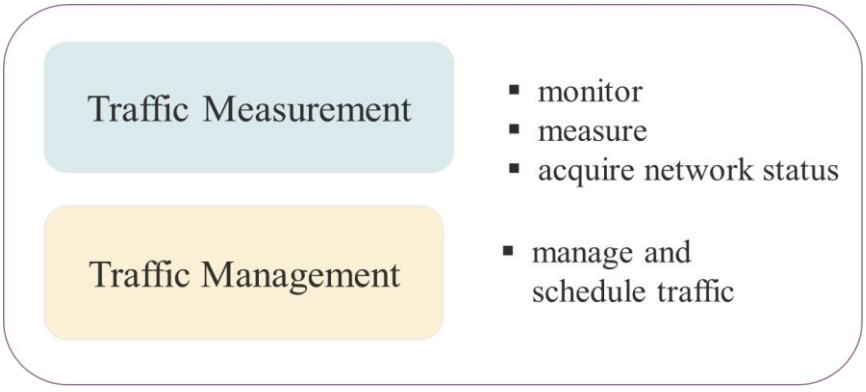


Figure 2.3 The components of Traffic Engineering

Figure 2.3 illustrates the components of TE. The TE technology based on the SDN comprises two portions: traffic measurement and traffic management [1]. The goal

of the traffic measurement is to studies the monitoring, measuring, and acquiring the SDN network's status information in the SDN network. The information includes the status of current topology connection, ports (down or up), various types of packet counters, the counters of the dropped packets, ratios link bandwidths utilization, end-to-end traffic matrices and end-to-end network latency so on. For avoiding network congestions and improving network efficiency, the network status information can be used in validating whether the current network status is current by the administrator and predicting the future traffic trend by analyzing packet counters statistics.

Network management mainly studies how to maintain network availability and how to improve network performance. Network traffic scheduling is an important way to improve QoS performance for the different application traffic. In general, SDN has multiple paths between the source and the destination node, which can be used for traffic scheduling. The controller maintains the global view of the current status of each path in the network using various network measurement technologies mentioned above. Consequently, the network administrator can design a traffic scheduling algorithm to dynamically plan data forwarding paths to meet users' requirements.

2.3.2 Network Resource Optimization

Network resource optimization can involve a number of specific optimization goals, including traffic reduction, blocking reduction, latency reduction, and load balancing. Some of these goals are not exclusive and are closely linked; for example, reducing traffic in the network often results in a reduction in blocking, due to there being additional capacity available within the network for further traffic. However, there are also other ways to reduce blocking within the network, such as spreading the load across the network in a load balancing approach. These close links between optimization goals are important, as they provide insights into multiple solutions to larger optimization problems. These types of problems often require using multi-objective optimization techniques, in order to achieve the best solution possible. This is because multi-objective optimization allows for a far wider search of solution parameters than normal optimization techniques.

With the existing optimization goals in mind, it should be clear that each characteristic being improved could benefit specific scenarios; furthermore, it is important to tailor optimization techniques to the application being utilized on the

network. For example, delay-tolerant content delivery would not benefit significantly from latency reduction, but the reduction in traffic would provide improved scalability due to additional capacity being available for further demands. In addition to this, a reduction in traffic would also reduce transmission costs for any links where cost is linked to usage.

2.3.3 Data Centers and Cloud Environments

Compared with the small scales network, the requirements of TE and policy implementation are really high in the case of large scales network architecture like a datacenter. Generally, increasing network latency and persistent troubleshooting may result not only in undesirable end-user experience but also in substantial effects on the cost penalties for the operators. Due to the significant feature of centralized control framework, the implementation of Datacenter (DC) in SDN can provide the fine-grain network management which makes easier for the network operators to monitor and manage hundreds of network element. For example, Google described four generations of their datacenter networks by using SDN technology in 2015. SDN tied a connection between its geographically distributed data centers from all over the world [42].

The implementation of SDN in the cloud computing environment delivers a solution for powerful TE to increase service scalability and automated network provisioning. Microsoft public cloud [57] and NTT's software-defined edge gateway automation system [53] are the distinguished examples of SDN deployment in a cloud computing environment.

From the perspective of cloud operators, energy consumption becomes the biggest issue for reducing operational costs and expands. In [32], the authors tried to reduce energy consumption by switching off redundant switches from the controller side during low traffic demand.

2.3.4 Campus and High-Speed Networks

The heterogeneous networking technologies integration with a centralized controller and OpenFlow enabled network elements has seen a great deal of applicability in optical high-speed networking. Customer needs for the SDN framework for enterprise networks are urgent due to the diversity of network traffic patterns that require proactive management to adjust network policies and fine-tune performance. Using centralized

real-time programmability of SDN, the network may eliminate middleboxes which provide services such as NAT, firewalls, access control and service differentiation solutions and load balancers [51].

For achieving programmability for the greater network, SDN application, and high speed and campus networking controlled with OpenFlow continue to grow resulting in new as well as hybrid solutions. SDN provides a centralized control plane to effectively monitor the network resources utilization.

2.3.5 Residential Networks

As the number and heterogeneity of network elements connected to the Internet continuously growing, the network in the customer's house has become a critical factor to manage network operations and meet the end-user expectations. SDN provides an ability to manage residential and small office home networks. SDN provides a centralized control plane to effectively monitoring network resource utilization. SDN offers a great opportunity of effective monitoring for network usage visibility to network operators and residential users via the [15], [38].

Using the SDN controller, Dillon and Winters [21] proposed the introduction of virtual residential gateways to allow providers remote management flexibility in delivering service to homes. For fine-tuning and troubleshooting the residential network, an SDN controller controlled and managed remotely the residential router or gateway from the service provider site [21], [25], [16].

Some contrasting schemes propose giving users more control and incorporating SDN based monitoring in the residential environment to change network policies [21] [64] [75]. From a security point of view, it has been argued that an SDN based anomaly detection system in a residential SDN environment provides higher scalability and more accuracy than intrusion detection systems deployed at the side of internet service provider [49]. Feamster in [27], proposed totally outsourcing residential network security utilizing programmable network switches at the customer sites to allow remote management. By employing the outsourced technical expertise, management and running of tasks such as software updates and updating anti-virus utilities may be done more effectively as the external operator also has a wider view of network activity and emerging threat vectors. The privacy of end-users where technical operations related to residential network management are outsourced also requires consideration [52]. SDN

framework for residential networking remains an active area of industry and academic research.

2.3.6 Wireless Communications

The SDN paradigm has been ported to mobile communication networks because of the real-time programmability and potential to introduce new applications and services to consumers. A programmable wireless data plane allowed developers to fine-tune mobile communications performance by offering routing based on flexible MAC and physical address in comparison to the traffic forwarding based on layer 3 logical address[44]. There have been growing efforts to include the SDN layering model in the upcoming 5G mobile communication. There is an opportunity to offer a more modular control and traffic forwarding framework with the use of SDN, user traffic can be separated and routed over different protocols. Similar to information-centric networking, an efficient network resource management scheme is needed to provide maximum utilization network slicing, and guaranteeing fairness among several QoS classes [67].

Using SDN to maximize energy efficiency in 5G networking has, therefore, been the subject of investigation in several studies. SDN has also been test-implemented in 5G to allow rapid application service provisioning while adhering to stringent QoS requirements. At the more local level such as Wi-Fi access networks, SDN could be used to offer a great deal of ubiquity in connecting to different wireless infrastructures belonging to different providers using user device identity management which is in turn coordinated and proactively managed by the SDN controller.

2.4 Research Challenges

This section discusses the major research advances made in several SDN areas in detail. With the growth of network applications in the SDN framework, the highlighted areas of research challenges going from application performance to security in the present SDN architecture. Most of these research provided one or other form of achieving QoS.

2.4.1 Application Performance

The improvement of application performance has been the primary area of focus in a number of SDN related studies ranging from application-aware SDNs, utilizing the framework for optimizing time-critical applications to the development of novel application performance monitoring solutions. The following sub-sections discuss the studies carried out in this regard.

2.4.1.1 Application-awareness in SDN

Application-awareness in SDN infrastructure considers the benefits of the SDN framework to compromise greater performance for particular applications. The southbound APIs like OpenFlow is capable of Layer-2/3/4 based policy enforcement but shortage to provide high-level application awareness. Therefore, network management primitives are employed which alter the traffic forwarding policies for individual applications and the SDN controller translates these into device configuration using a southbound API such as OpenFlow. The concentration in this area has seen several studies on video streaming (YouTube, P2P video, etc.) and voice communications (VoIP), using the SDN architecture to improve the individual application quality of service.

In one such 'application-aware' SDN work, Mekky et. al [50] proposes a per-application flow metering approach using the SDN framework. Applications are identified in the data plane and applied the appropriate policies according to the individual application tables. The proposed scheme minimizes the SDN control channel overhead. The study showed significant improvement in application forwarding performance with low overhead.

2.4.1.2 Application Performance Monitoring

With the recent developments of virtualization technologies, a wide range of applications are hosted on multiple servers in datacenter and cloud environments. To improve the performance of applications in such area requires the development of a tool which can monitor the application traffic in virtual platforms and apply traffic management policies. In this domain, SDN is seen as a key enabling technology due to the decoupling of control logic from forwarding elements.

For large scale network performance analysis, Liu G. and Wood T. [19] presented a platform called NetAlytics which uses network function virtualization (NFV) technology to deploy software-based flow monitors in the network. The system aimed to analyze application performance and application popularity by collecting network traffic statistics.

For cloud-based data centers, Maan et. al [47] developed a network monitoring system to monitor network flows at the edge of the network. The work proposed a scalable network monitoring utility called EMC2 to be used for performance evaluation of switch flow accounting methods. The evaluation recommends NetFlow [105] which can provide network handling ability with minimal use of computing resources to monitor application traffic in virtual environments and cloud-based data centers.

2.4.1.3 Video Streaming and Real-time Communication

Focusing on video streaming applications, Egilmez et. al [22], devised an analytical framework for traffic optimization at the control layer while offering dynamic and enhanced Quality of Service (QoS). The study reported significant improvement for the streaming of encoded videos under several coding configurations and congestion scenarios.

Jarschel et. al [35] instead focused specifically on improving YouTube streaming experience for end-users. The study used Deep Packet Inspection (DPI) and demonstrated how application detection along with application state information can be used to enhance Quality of Experience (QoE) and improve resource management in SDN.

CastFlow [48] was another example of video streaming optimization, which proposed a prototype aimed at decreasing latency for IPTV using a multicast approach, logically centralized and based on OpenFlow networks. During multicast group set all possible routes are calculated in advance to reduce the delay in processing multicast group events (joining and leaving hosts and source changes). Using Mininet based emulation, the reported results showed satisfactory performance gains and the time to process group events appeared to be greatly reduced.

Noghani and Sunay [54], also utilized the SDN framework in allowing the controller not only to forward IP multicast between the video streaming source and destination subscribers but also to manage the distributed set of sources where multiple

description coded (MDC) video is available. For medium to heavy loads, the SDN based streaming multicast framework resulted in enhanced quality of received videos. Some related studies try to verify the importance that the underlying testbeds may have on any evaluations reporting perceived improvements in video streaming quality using SDN.

Panwaree et. al in [56], showed the benchmark of the packet delay and latency performance of videos which were tested on both Mininet environment and actual physical PC clusters using Open vSwitch. It was noted that the packet delay and loss in the PC-cluster testbed were higher than the Mininet emulated testbed suggesting a careful interpretation of performance expectations in realistic environments.

2.4.2 Data Center Solutions and Resource Allocation

Traffic measurements in data centers show signification variation in network workload hosting multiple applications on the same physical or virtual network fabric [83]. The SDN paradigm brings automation and on-demand resource allocation in data center networking [97] [4]. Using SDN, the DC environment can afford faster state changes, a fundamental necessity of modern data centers [42]. Several prior works have discussed the improvement of individual applications and services in the DC network environment.

Application connectivity models were used in [10] and [43] to allocate per-application network bandwidth. However, application delivery constraints are prevalent in data centers where virtual machines from several applications may be simultaneously competing for resources. To address bandwidth allocation, Kumar et. al [42] employed user-space daemons running on application servers to predict anticipated traffic and assigning forwarding paths to applications using operator configured policies.

Jeyakumar et. al [36] proposed a weighted bandwidth sharing model among nested service endpoints allocating resources hierarchically at core fabric, rack, and individual machine level. However, the resulting operator defined per-application bandwidth sharing schemes are highly dependent on the stability of application demands for long enough periods to optimize network traffic. Fang et. al [24] tried to prevent excessive traffic arrival into the network by implementing host congestion controls and proposed multipath selection to achieve optimal network resource utilization.

High-end network vendors propose and recommend the confederation of services approach to improve performance [86]. However, the application

differentiation available at the system and network-level to assign machine limits and create end-to-end network topology per application does not explicitly consider user's application trends. Therefore, resource provisioning on a per-application basis leads operators to pre-set network provisioning models to improve the end-user experience regardless of real-time network conditions. A more user-centric approach where user requirements and activities are captured may present a resource abstraction model, which could offer service providers the ability to fine-tune network resource share on the basis of user traffic classes in view of business and user requirements instead of isolated applications.

2.4.3 QoS Routing and Path Establishment

A variety of network algorithms can be implemented in SDN including shortest-path routing, and more sophisticated ones such as traffic engineering [30]. Recently, different applications have been implemented in SDN with policy-based access control, adaptive traffic monitoring, wide-area traffic engineering, network virtualization, and others. SDN controllers manage the entire network, so they must often change rules on multiple switches.

From the beginning of the networks, communication is decided based on the Shortest Path First (SPF) routing algorithms. In today's network, QoS becomes more and more important for a wide range of communication network settings and applications. However, because of the limitation of the current SPF-based routing algorithm, network link congestion often occurs even when the total load is not particularly high. This is challenging for multimedia applications that require certain QoS [4] levels for appropriate functioning. Generally, the route computation is either carried out in distributed nodes which can Internet Protocol (IP) routers or by a centralized controller. The main goal of the Shortest Path (SP) problems is to minimize a unique end-to-end QoS metric. Based on the category of the network routing problems, there are four types of QoS based routing algorithms [4]:

- Shortest Path (SP)
- Constrained Shortest Path (CSP)
- Multi-Constrained Shortest Path (MCSP)
- Multi-Constrained Path (MCP)

2.4.4 Congestion Control

Most congestion control algorithms in the literature are flow-based in general and the Transmission Control Protocol (TCP) in particular. TCP is a feedback congestion control on flow-level where the transmission rate is based on a sliding window. Packets are acknowledged by the receiving side, and when congestion is detected by the sender not receiving an acknowledgment before a set time elapse, or receiving duplicate acknowledgment, it retransmits the unacknowledged packets [41]. In [68], the authors investigate the predictability of self-similar and how this can be used in congestion control to improve the throughput of TCP by proposing a feedback congestion control, using the throughput as a control variable, and adjusting the bandwidth from the client to maximize the throughput.

Over-demand of network resources can cause congestion which may lead to performance degradation when compared to states with lower demand [31]. It is therefore imperative to assume that network resources are sufficient to cater for the offered traffic most of the time.

Some authors highlight the need for more efficient detection of network congestion. Huang et al. [34] note that TCP is suboptimal for high-speed networks and suggests using free router capacity, ingress aggregate traffic and queue length as decision variables to make TCP converge faster and achieve fairness, and Haas and Winters [31] suggest probing for congestion with test packets. An alternative algorithm to Random Early Detection (RED) using Explicit Congestion Notification (ECN) is presented in [26], where packet loss and link utilization are used, rather than queue length, to detect congestion. The result is a faster detection of congestion and more adequate rate adjustment to mitigate the congestion.

Using the method classification in [28], this research proposed a congestion control scheme that works on hop level and measures flow quality of service, disregarding admission control and transport protocol functions. The effect of the end-to-end performance is studied delay, packet loss and throughput by dynamic traffic aggregation at the nodes and optimal routing with respect to delay. A theoretical investigation of feedback congestion control strategies can be found in [60]. It discussed the differences between feedback from aggregate traffic and individual flows. It is also shown, that the transmission rate resulting from a feedback congestion control can be

expressed as a fixed-point equation, a technique used in this approach to determine optimal routes.

2.5 Literature Review

This section covers an existing literature review of traffic engineering and resource allocation for SDN based networks. A lot of real-time business network traffic such as instant messages, voice data, and so on, is sensitive to packet losses and delays in the transmission process. Thus, an important problem of traffic management is the reasonable scheduling of network resources for providing the QoS for business. SDN has an open control interface supporting flexible network traffic scheduling strategies, which can satisfy different network applications' QoS requirements. Therefore, most of the researchers try to design a traffic scheduling algorithm to dynamically plan data forwarding paths to meet users' requirements.

Some research works studied the SDN control framework and flow rerouting schemes to provide the end-to-end QoS provisioning.

In [6], Jaward et al. proposed a policy-based QoS management framework to achieve end-to-end QoS with rerouting and rate-limiting.

In [73], Chenhui et al. proposed a QoS-enable management framework to guarantee the QoS of specific flow and employ the queue technique and policy to satisfy the requirement of service. Flow rerouting is carried out on the priority flow in order to mitigate the impact on best-effort flow.

Tomovic et al. [65] presented a new QoS based SDN control framework that provides the required QoS level for multimedia applications flexibly and automatically. They aimed to minimize the best-effort traffic performance degradation. They estimated the link utilization by using a threshold value (80% of the link) and rerouted the best-effort traffic before congestion occurs.

The equivalence multipath routing technology (ECMP) [77] was proposed as an effective load balancing solution. ECMP routing based on a hash algorithm. To make hash calculations, the header fields of the packet is extracted whenever the packet arrives at a switch or router. Then, one of the forward paths is selected by the hash value. As the results, the IP packets with the same head are forwarded along the same path. A key limitation of ECMP is that two or more large, long-lived flows can come into collision on their hash and end up on the same output port, creating a bottleneck.

Therefore, there are many works to complete ECMP by implementing the flow detection module and detecting the large flow then schedule these flows along a redundant path with a suitable capacity to improve the network performance.

A dynamic and scalable flow scheduling system, Hedera [5], is for avoiding the limitations of ECMP. By periodically pulling the flow statistics, it detects the elephant flows at the edge switches. Initially, switches send a new flow via the default flow rules with one of its equal-cost paths till the flow size grows and meet the threshold. Then, the flow is identified as elephant-flow. It used 10% of the network interface controller (NIC) as the default threshold. It has functions such as a global view of routing and traffic demands, collection of flow information from switches, computation of non-conflicting paths for flows, and instructing the switches to re-route traffic accordingly.

For improving the network performance and scalability DevoFlow [18] was proposed by maintaining the flows in the data plane without losing the centralized network view. As a result, it decreases the interaction between the data plane and the control plane. It designed wildcards based multipath matching rule. Initially, it forwards the traffic with its multi-path wildcard rules. The controller calculates the path with least congested when an elephant flow is detected and re-routes the traffic to this path.

Mahout [17] modified the end-hosts for detecting elephant-flows to overcome the problem of high resource overhead by the flow detection mechanisms used in Devoflow and Hedera. It used ECMP for routing the normal traffic. The controller calculated the best path when an elephant-flow is detected. The controller collects the link utilization and elephant-flow statistics from the switches to select the least congested path for calculating the best paths. Mahout could faster and lower processing overhead in the detection of the elephant-flows while comparing with other methods. However, it required the end-hosts modification.

On the other hand, some works are trying to place flows based on minimum link utilization and independent of flow size. In [69], F.P.Tso and D.P.Pezaros introduced Baatdaat, measurement-based flow scheduling for reducing congestion in data center networks. It used the lightly-utilized paths and allows flow rerouting to schedule traffic flows.

In [11] Benson et al. presented a traffic engineering mechanism for data center network called MicroTE, which uses an end-host elephant flow detection to detect the elephant flows. MicroTE passively monitors the flow status by flow statistics like the

Mahout. It triggers flow aggregation behavior when the flow status is clearly changed, For judging the current flow is an elephant flow, it can be predicted relying on the difference of the size of the instantaneous flow rate and average flow rate. It starts the routing optimization calculation or deals with it using a heuristic ECMP algorithm when it can predict the flow.

Tootoonchian et al [66] proposed OpenTM which is a traffic matrix estimation system for the SDN. It can detect all active network flows according to the flow forwarding path information and routing information of the controller. It contains the various selective querying methods for routing nodes to receive accurate information of the flow evaluation and packets number.

Furthermore, there are many kinds of solutions that have been proposed to guarantee the QoS requirement in the SDN network. OpenFlow supported queue scheduling is the most common tool to implement QoS control for individual flow in the data plane. It can be used in providing bandwidth guarantees by shaping and prioritizing traffic to share the network bandwidth [8]. In [13], Boley et al. developed a QoS framework to achieve optimal throughput for all QoS flows with the help of meters' function. In [45], Li et al. implemented a queue scheduling technique used on SDN switches to achieve QoS for cloud applications and services.

Yan et al. [40] proposed HiQoS which is a QoS-guarantee solution in the SDN network. To guarantee QoS for the different types of traffic, it identifies multiple paths between source and destination nodes by using the queuing. It can increase throughput and reduce delays according to its experimental result. It reroutes the traffic from failed paths to other available paths for recovering from link failure rapidly.

OpenQoS was proposed to provide a QoS guarantee for multimedia business flows distribution. Since multimedia business flows have different packet heads while comparing with other packet heads, it divides all data traffic into two categories, data flows and multimedia by using OpenFlow configuration matching rules. It observes the forwarding paths performance with packet losses and delays and chooses the best path that can meet with the requirements of QoS. By using the original path, it forwards the remaining data flows. However, it does not consider the business flows with multiple QoS requirements and it only optimizes multimedia flow scheduling.

In order to provide QoS, the appropriate network resource allocation is needed. The knowledge of the current network state is required to make the right decisions with

regard to packet forwarding. Therefore, network monitoring plays an important role in providing QoS. In [76] and [70], the researchers developed a monitoring module for the controller of the SDN, which can analyze dynamic changes in network flows according to messages received by the controller.

User-reservation based end-to-end dynamic bandwidth allocation scheme was proposed in [78] and [80]. Akella et al. [3] studied bandwidth allocation for ensuring the end-to-end QoS guarantee of each cloud user based on SDN. Their work emphasized on bandwidth allocation with queueing techniques.

2.6 Chapter Summary

The majority of studies highlighted in the above discussion included a range of network management models ranging from a more 'application-aware' SDN paradigm to the use of SDN-based QoS routing, including congestion control and SDN based monitoring techniques which allow performance measurement and QoS guarantees for certain services. It is learned that a wide range of techniques can be used to implement QoS-based traffic engineering systems in SDN. To replace the current internet architecture, SDN has to come with solutions to a number of problems. Some of them have been addressed in the literature review. However, the quality of service capabilities of all the different components of SDN will also play a major role in the widespread adoption of SDN in the real internet.

The current work in SDN based traffic optimization focuses on improving the quality of individual applications and services such as VoIP or video streaming in several different network environments. Other studies involve information-centric networking focused on bringing the data sources closer to the network edge, to again improve traffic conditions for the hosted application(s). However, existing studies do not specifically consider that prioritizing specific applications may have on other applications in the SDN architecture.

CHAPTER 3

THEORETICAL BACKGROUND

Software Defined Networking (SDN) is the newly proposed paradigm for drastically changing the way the existing networks are working. The basic principle of SDN lies on the decoupling of the network control and forwarding planes; then an external SDN controller can dynamically inspect rules into SDN capable nodes. Based on these rules, the SDN capable nodes perform packet inspection, manipulation, and forwarding. In addition, the underlying SDN capable nodes (can be Open flow switches, OpenVswitch or Virtual Routers, etc.) can inspect and modify packet headers at different levels of the protocol stack, from layer 2 to application layer [97].

In the SDN architecture, the control layer has network intelligence and the underlying network infrastructure layer and those layers are connected via the APIs. Thus, researchers and developers can be able to focus on each layer of the architecture without considering the other layer complexities. The main feature of SDN architecture is programmability that enables carriers and enterprises to adapt rapidly changing business demands in an automated manner and more flexible [92].

3.1 SDN Reference Architecture

An SDN consists of three layers: application layer, control layer, and data plane layer. A detailed explanation of the key layers is:

- **Data (forwarding) Plane:** In a bottom-up fashion, the data plane is the forwarding device interconnected through wired or wireless means. The data plane's purpose is to forward network traffic as efficiently as possible based on a certain set of forwarding rules which are instructed by the control plane. SDN architecture removes the forwarding intelligence from the networking hardware and moving these functionalities to the control plane. One way traditional OpenFlow switches (i.e., the data plane) provide these forwarding properties is through Ternary Content-Addressable Memory (TCAM) hardware. The forwarding elements and SDN controller communicate using the southbound interface called OpenFlow. At present, the Open Flow protocol [71], serves as a

standard southbound communication protocol supported by several vendors including the ONF [92] [84].

- **Control Plane:** The SDN control plane, often referred to as the controller, is the component that programs and manages forwarding devices over the southbound interface. The control plane is responsible for making decisions on how traffic would be routed through the network from the source node to destination node based on end-user application requirements and communicating the computed network policies to the data plane. The controller becomes the centralized brain in the network and it works as a network operating system (NOS). An SDN controller translates different application requirements such as the need for QoS, traffic prioritizing, bandwidth management, etc. into relevant forwarding rules which are communicated to data plane network forwarding elements. SDN becomes possible to manipulate flow tables in individual elements in real-time based on network performance and service requirements by using network programmability through the control plane. In brief, the controller gives a clear and centralized view of the underlying network giving a powerful network management tool to fine-tune network performance. Furthermore, the control plane provides the network abstraction that can be used by network applications to achieve high-level functionality in the network.
- **Application Plane:** The application plane includes network management applications such as firewalls, routing, and other applications that enforce the policy. An abstract view of the underlying network is presented to applications via a controller northbound API. The level of abstraction may include network parameters such as throughput, delay, and availability. Applications in return request connectivity between end nodes and once the application or network services communicate these requirements to the SDN controller, it correspondingly configures individual network elements in the data plane for efficient traffic forwarding. Centralized management of network elements provides additional leverage to administrators giving them vital network statistics to adapt service quality and customize network topology as needed [51]. For example, during periods of high network utilization certain bandwidth-consuming services such as large file transfers, video streaming, etc. can be load-balanced over dedicated channels. In other scenarios, such as during an

emergency like fire alarms service such as VoIP can take control of the network i.e. telephony taking precedence over everything else. Figure 3.1 illustrates a simplified scheme for SDN.

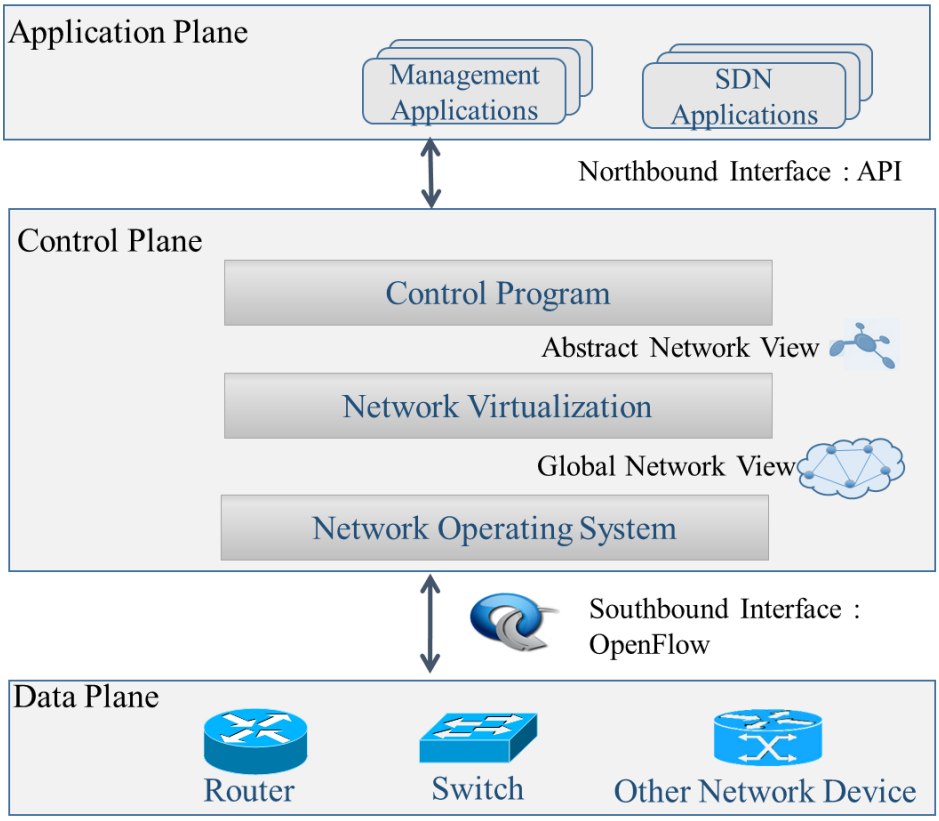


Figure 3.1 Simplified SDN Scheme

3.2 Southbound Interfaces and Protocols

In order to configure the forwarding in the data plane, an SDN controller needs to have a communicating with the forwarding devices. The family of protocols used for communicating is called southbound interfaces. There are several well-known southbound interfaces, e.g., OpFlex [102], POF [46], ForCES [16], and OpenFlow [88]. The leading southbound protocol is OpenFlow, supported by Cisco, HP, Juniper, and IBM. Complementing the southbound interfaces, there are southbound protocols such as Open vSwitch Database (OVSDB) and OpenFlow Management and Configuration Protocol (OF-CONFIG) to control the operations of the forwarding devices (e.g., tunneling, shutting down a network port, and queue management) [97]. In this section, we review OpenFlow and OVSDB, the southbound interface and protocol used in our research.

3.2.1 OpenFlow

The most popular southbound protocol, OpenFlow, was founded at Stanford University in 2008, and is currently managed by the Open Networking Foundation (ONF) [83] and become. OpenFlow is the communication protocol that allows the SDN controller to directly manage the data plane and install forwarding decisions on the network devices. The controller can create, remove, and also modify flow table entries in the switch using this protocol. The controller deals with the OpenFlow switch via the secure channel. The secure channel is one of the interfaces to make the connection from each OpenFlow switch to the Controller. This interface is used in managing and regulating the switch by the Controller, informing the events via the switch and sending packets through it. The interface may vary relying on OpenFlow switch implementation. However, standardized OpenFlow protocol is needed in sending each message through the secure channel need.

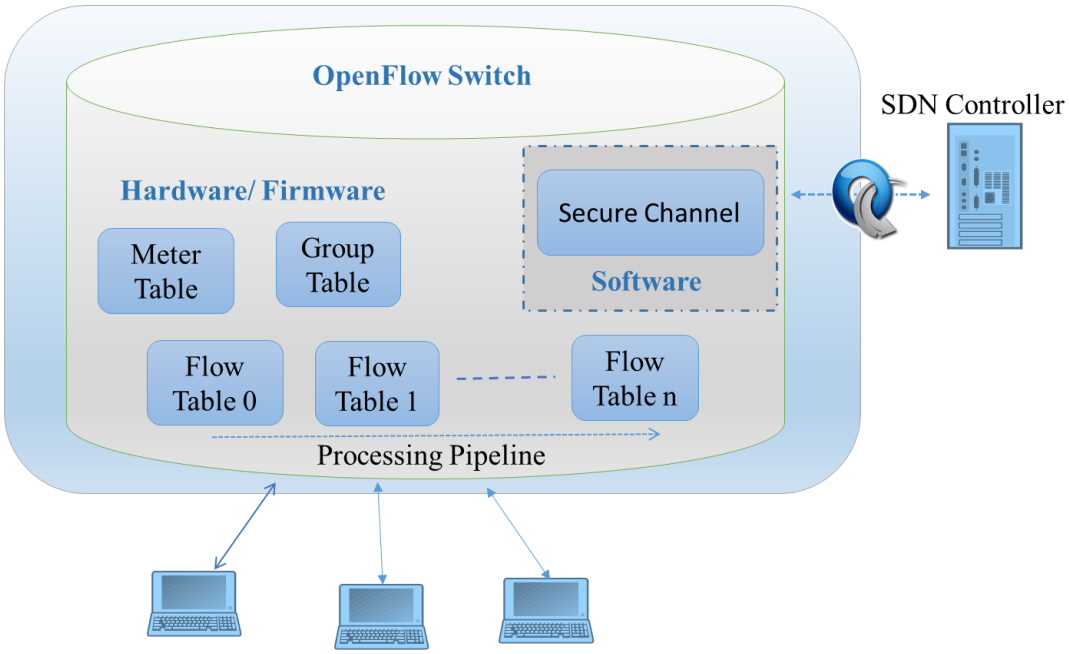


Figure 3.2 OpenFlow Architecture

As illustrated in Figure 3.2, an OpenFlow switch is a software switch that consists of one or more pipelined flow tables, a group table, which performs packet lookups and forwarding, and an OpenFlow channel to an external controller [71]. The flow table is essentially a lookup table with match fields and actions and is processed

like a pipeline. Pipelined flow tables contain traffic flows, as defined later. A flow will match the first flow table, and potentially be forwarded to a port or another flow table. This is what we mean by pipelined; the flow match rules happen iteratively, like water flowing through a pipe.

A traffic flow is a "sequence of packets sent from a particular source to a particular unicast, anycast, or multicast destination that the source desires to label as a flow" [83]. Flow classifiers are typically based on the 5-tuple consisting of a destination address, source address, protocol, destination port, source port. The primary benefit of flow-based routing is that it eliminates the need to do lookups to the routing table on a per-packet basis. The route lookup can be done for the first packet in a flow, and then the same transform applied to each packet in the sequence. Flow tables can easily be implemented in hardware, and most vendors support some form of flow matching in either software or hardware ternary content-addressable memory (TCAMs) [88].

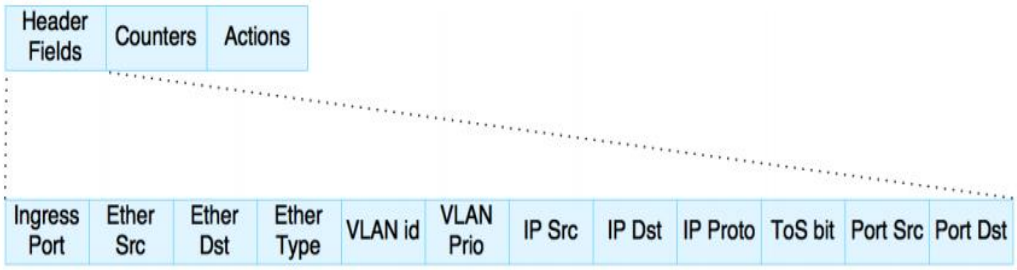


Figure 3.3 Flow Entry Scheme

In the SDN model, OpenFlow serves as the data plane handling packet forwarding operations for the OpenFlow controller [71]. The flow tables handle packet lookups and forwarding. As shown in Figure 3.3, a flow entry consists of header fields (e.g., source and destination IP addresses and ports) to uniquely identify each flow, counters for collecting the stats of how many times a flow entry is used successfully, cookies used for annotation by SDN controller, timeouts that control how long to keep a flow entry in the flow table, and priority that helps the switch to choose amongst multiple matches (if there are). Lastly, there are actions that determine the policy for successfully matched packets. To clarify, when a packet arrives at the switch, the switch starts looking for a match and the matched flow entry will determine the action, for instance forwarding the packet on a specific port. Upon receiving a packet, a forwarding

device scans the flow tables, starting from table 0 (which is the mandatory flow table), for matching flow entries. If there is no match in flow table 0, it will start looking for the match in flow table 1 (if table 1 exists, the number of flow tables is configured by the user). The process will continue until a successful match is found. In the case of multiple matches in a single flow table, the entry with the higher priority will be picked. The devices perform the action (i.e., forwarding the packet to a specific port) defined in the flow entry. There is also a special flow entry called table-miss flow entry with the priority of zero that matches all the packets. This entry catches all mismatched flows. This entry may direct the device to drop unknown packets or send them to the controller. The controller can install a new flow entry on the switches for the flow or can drop the packet.

Think of this as an if-then rule in an L3 (layer 3) router. If the frame matches this 5-tuple, then we apply this action set. An L3 router is a router which performs forwarding decisions based on the L3 Internet Protocol (IP) payload. An L3 packet comes in and is sent to the ingress flow table, which is matched by the table-miss flow entry. This flow entry will then forward the packet to the controller for a route lookup. The controller finds the appropriate next-hop and the proper network interface, and pushes a new flow entry to the OpenFlow switch for this packet and forwards it out the appropriate interface. The next packet in that flow will match the flow entry that was just pushed down into the OpenFlow switch, which will then apply the action to the packet forwarding it out the same egress interface the previous packet was sent to and applying the same action. Only the first packet in a flow would cause a route lookup, speeding up packet processing. [88]

3.2.2 OpenFlow Flow Table

The flow table is essentially a lookup table with match fields and actions and is processed like a pipeline. When the frame ingresses the port it is processed by Table 0 by the highest-priority matching flow entry. This flow entry will contain an action set which can either output the frame to a specific port, apply actions, or send the frame to another table. In the event of a table miss the frame is dropped by the switch. A table miss happens when there is no matching rule in the table to match the frame. Each Flow Table contains the following fields [71]. Recall that a flow router consists of a lookup

table and an action set; this is the lookup table that matches on various fields in the packet header.

Table 3.1 OpenFlow Flow Table Fields

Type	Description
Match Fields	The match criteria for frames. Consists of header data and metadata information. Match fields are placed on the flow table in order to define the packet to which an action is to be performed. This contains the 5-tuple information and some additional criteria that can also be used.
Priority	The match priority. Matches occur in priority order. Useful for defining exception entries and default entries in the table pipeline.
Counters	Counts the number of matches.
Instructions	Defines what is to be done to the frame after a match; there are one or more of these.
Timeouts	Defines how long a flow can exist in the switch. A soft timeout defines how long the flow lives if a matching frame has not been seen. A hard timeout defines how long a flow lives no matter the match count.
Cookie	Controller defined field. This is not used in packet processing but is useful for filtering flow statistics.

3.2.3 OpenFlow Messages Types

There are three general types of OpenFlow messages: controller-to-switch, asynchronous, and symmetric. The controller uses the controller-to-switch messages to query information from, transmit packets to, or configure the switch. Normally, the controller initiates the controller-to-switch message with or without being required to send a response from the switch. On the other hand, asynchronous messages are sent without solicitation from the controller. Examples of these include packet-in messages, flow-removed, port-status, and packet-out messages. Symmetric messages require a response from the receiving party. Examples of these are hello, error, echo, and experimenter messages. [71] OpenFlow defines a specification that one can use to talk

to OpenFlow switches, and this technology will be utilized throughout this thesis to provide the mechanism to insert flows into these switches.

3.3 SDN Controller for the Control Plane

In the SDN architecture, the controller works as a brain of the network and it is where the control plane resides as depicted in Figure 3.1. A controller is a software that serves as a central control point that overlooks the network and through which applications can access and manage the network. When the controller is said to be a central point of the network, it is only meant to be logically centralized. The controller software is typically deployed on a high-performance server machine, but to distribute the load or to ensure high availability and resilience, more servers may be involved and connected in various topologies [29].

The controller is responsible for the following tasks:

- **Device discovery:** the controller takes care of the discovery of switches and end-user devices, and their management.
- **Network topology tracking:** the controller investigates the links interconnecting devices in the network and keeps a view of the underlying resources.
- **Flow management:** the controller maintains a database mirroring the flow entries configured in the switches it manages.
- **Statistics tracking:** the controller gathers and keeps per-flow statistics from the switches.

It is important to emphasize that the controller does neither control the network in any way nor does it replace any networking devices. Even the basic switching or routing functionality has to be provided by specific applications that approach the network through the controller. Communication with networking devices is realized through a southbound interface, for which Open SDN promotes the OpenFlow protocol. These interfaces are used to configure and manage the switches and to receive messages from them. The connection is realized via a secure channel and depending on the setting is either encrypted or unencrypted.

Applications communicate with the controller using a northbound interface. Through this interface, they retrieve information about the network and send their requests, while the controller uses it to share information about occurring events.

Depending on the implementation, the interface may be low-level, providing unified access to individual devices, or high-level, abstracting much of the underlying layer and rather presenting the network as a whole. There is no standard for the northbound interface and every controller implements its own APIs – be it Java API, Python API, REST API [27] or else. This current lack of a standard northbound interface makes it difficult to create controller independent applications [29].

3.4 Software Switch for the Data Plane

A summary of command software switches which are also used for experimentation and new service development were given in Table 3.2. Detailed expression of two well-known software switches presently available is present in the below sub-session.

Table 3.2 Common OpenFlow Software Switches

Switch	Implementation	Description
Open vSwitch	C/Python	OpenFlow stack that is used both as a virtual switch and ported to multiple hardware platforms [93].
Ofsoftswitch13	C/C++	User-space software switches implementation [89].

3.4.1 Open vSwitch

OpenFlow may be deployed either at the software level or hardware level onto forwarding devices in the data plane. More specifically, many well-known networking vendors like Cisco, Juniper, IBM, and HP, support OpenFlow, either with a dedicated product or running an OpenFlow software switch on top of their switches. Open vSwitch is one of the software switches implemented to support OpenFlow which can be installed to enable OpenFlow [71]. Open vSwitch is a multilayer software switch that is intended to function as a virtual switch. Open vSwitch supports all versions of OpenFlow from 1.0 to 1.5 as well as GRE tunneling, queues, and so forth. The core of Open vSwitch is the switch daemon (ovs-vswitchd). This daemon tracks statistical queries and flow management internally on the switch, and also handles communication with external devices and services [93]. For management and configuration, in parallel with

connectivity to the OpenFlow controller, it is possible to configure and control the Open vSwitch via ovs-vsctl and ovs-ofctl. ovs-vsctl is a command line tool to configure ovs-vswitchd by providing an interface to its configuration database, while ovs-ofctl is a command line tool for monitoring and administering Open vSwitch. Moreover, ovsdb-monitor is a tool to view flow tables and databases of Open vSwitch. As illustrated in Figure 3.4, ovsdb-server relies on the OVS Management protocol to communicate with Remote Open vSwitch db (a database maintained by Open vSwitch to store its information). Unlike flow entries in the switch, the OVSDB configuration is preserved even after the switch restarts.

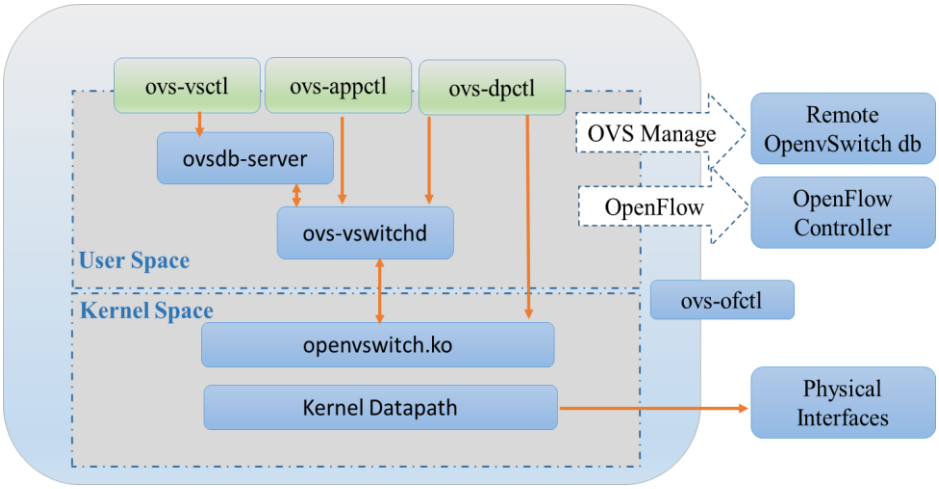


Figure 3.4 Simple Open vSwitch Architecture

3.4.2 OfSoftSwitch (CPqD)

The OfSoftSwitch13 (CPqD) [89] is another switch that is widely used in the research community. It is an experimental switch forked from the Ericsson Traffic Lab 1.1 SoftSwitch implementation with changes in the forwarding plane to support OF1.3 [37]. The Ofsoftswitch13 is running in the user space and it also supports multiple OpenFlow versions [89]. Ofsoftswitch13 supports a variety of OpenFlow features but it has recently run into some compatibility issues with the latest versions of Linux (Ubuntu 14.0 and beyond) and developer support has also stagnated. It comes packaged with the following tools to control and manage the data plane:

- **OfDatapath:** The switch implementation.
- **OfLib:** A library for converting to/from OF1.3 wire formats.
- **DPCTL:** Console tool to configure the switch.

- **OfProtocol:** A secure communication channel with the controller.

All this makes it a complete alternative to the OVS. However, the authors of the switch state the following, “Despite the fact the switch is still popular for adventurers trying to implement own changes to OpenFlow, support now is on a best-effort basis. Currently, there are lots of complaints about performance degradation, broken features, and installation problems.”[97]. The switch still has one of the best support for OF1.3 features among the available soft-switches, specifically the optional features like meter tables, etc., which makes it an attractive candidate to get hands-on with. Additionally, the soft switch supports a management utility called Data Path Control (Dpctl), to directly control the Open Flow switch including the flow addition and deletion, query switch statistics and modify flow table configurations.

3.5 Flow Removal and Eviction

Flows can be removed from the controller in three ways: at the request of the controller, by expiration, or via the switch eviction mechanism. [71] The Flow switch expiration mechanism defines a hard and an idle timeout. The idle-timeout causes eviction if and only if no frames have been seen for the duration. The hard-timeout causes eviction no matter if matches have been seen. The controller can also send a DELETE message, causing flow removal. The flow switch eviction mechanism lets the switch evict flows in order to reclaim resources. Upon removal, a FLOW REM message may be optionally sent if the SEND FLOW REM flag is set in the flow entry. This message is used to inform the controller that flow has been evicted so it can either keep statistics or make decisions based on this information.

3.6 Chapter Summary

This chapter considered the software-defined networking technologies in detail. The northbound and southbound communication interfaces allow for several key protocols to be used in the SDN framework. Protocols such as OpenFlow on the southbound and RESTful API on the northbound controller interfaces have seen significant adoption in both academic and industry research. In addition to communication protocols, recent years have also seen the development of several key controller platforms aimed at furthering the SDN paradigm and bringing substantial technical variety for researchers and operators to experiment and explore.

CHAPTER 4

END-TO-END QUALITY OF SERVICE

Quality of Service (QoS) in SDN is an area of ongoing research and has been increasingly becoming interested in the research community. There is no standard or formal definition of QoS. But, there are a number of definitions at the communication level where the notion originated to describe certain technical characteristics of data transmission. In this chapter, the traditional concept of QoS at the network level is introduced, and some of the interesting research works are summarized in the area of QoS in SDN. Moreover, a taxonomy of the applications and their QoS requirements is presented in this chapter.

4.1 The Quality of Service

Quality of Service (QoS) is the ability to satisfy the requirement of specific traffic. Lots of today's network applications require certain QoS guarantees. For example, an application like video streaming requires a small delay but the data communication requires less packet loss rate since it is less sensitive to delay. A failure to meet this standard might lower the Quality of Experience.

Standard Internet Protocol (IP)-based networks provide network services based on the “best-effort” delivery model. There are no bandwidth or latency guarantees in the “best-effort” delivery model since the model offers a point-to-point delivery service to deliver data to their destinations as soon as possible. The highest guarantee the network provides is reliable data delivery by using protocols, such as TCP. Although this is acceptable for traditional applications such as Telnet and FTP, this is inadequate for applications requiring timeliness guarantees.

Increasing bandwidth is a common solution to adequately accommodate real-time applications like VoIP, but it is still not adequate to sidestep jitter during traffic bursts. IP services must be supplemented to provide some appropriate level of quality for real-time applications. Typically, this requires extending the network software to provide a certain level of quality in potential packet loss, jitter and delay. That is exactly what Quality of Service (QoS) protocols are designed to do. Although QoS provides the ability to manage bandwidth, it can not create bandwidth. The network administrator

can configure the QoS properly and used more effectively to meet the wide range of application requirements. The main goal of QoS is to provide some level of predictability and control beyond the current IP "best-effort" service.

4.2 Applications QoS

At the present time, the key concept of QoS extends from the communication level up to the application level, in order to map QoS application requirements into low-level QoS parameters. So far, QoS has been specified in terms of system resources (CPU, memory utilization) or network resources (bandwidth, delay) and the network infrastructures have been deployed to support real-time QoS and controlled end-to-end delays.

4.3 QoS Provisioning in Traditional Network

QoS provisioning over the Internet is essential to ensure high-quality performance for different applications. There are two major approaches for supporting QoS into IP based network:

- **Resource reservation:** Resource reservation is one of the approaches to provide per-flow end-to-end QoS guarantee by allocating network bandwidth resources to guarantee QoS for a specific flow (e.g., a video streaming session). According to an application's QoS request, the network resources are allocated and subject to bandwidth management policy.
- **Prioritization:** Prioritization is one of the approaches to classify the network traffics and allocate network resources according to bandwidth management policy criteria. To enable QoS, network elements give preferential treatment to classifications identified as having more demanding requirements.

Even the different applications running on the same distributed system may have different QoS requirements with different parameter values. Moreover, some of these QoS parameters may be time depending but may not be mutually independent. There are a number of different QoS protocols and algorithms to accommodate the need for these different types of QoS:

- **ReSerVation Protocol (RSVP)** [79]**:** RSVP is a transport layer protocol which can provide the signaling to enable network resource reservation. By using RSVP, the receiver initiates network resource reservations by sending a

message, and the amount of reservation guarantees the satisfaction of bandwidth, timing, and buffer size constraints. Along the chosen path to the sender, reservation is established in a soft state and set aside the required resources. The soft state is refreshed periodically by the receiver or else it times out canceling the reservation. RSVP typically used on a per-flow basis and also used to reserve resources for aggregates.

- **Differentiated Services (DiffServ)** [12]: DiffServ was developed to provide a coarse and simple way to categorize and prioritize network traffic flow aggregates. As a first step, DiffServ classifies all flows into a limited number of classes and define a different "per-hop behavior" for each class. To define per-hop behaviors, it takes 6 bits from the Type-of-Service (TOS) field in the IP header. Then, a certain profile of traffic is created by the clients and pay it to the network provider. Client packets are marked by the edge routers. When the packets arrive at the core router, the core router will know what to do by seeing the 6-bit per-hop behavior code from the IP header.
- **Multi-Protocol Labeling Switching (MPLS)** [58]: MPLS is data forwarding technology that provides bandwidth management for flow aggregates via network routing control according to labels in packet headers.
- **Subnet Bandwidth Management (SBM)** [74]: SBM is a signaling scheme that enables categorization and prioritization at the data link layer (Layer 2) on shared and switched IEEE 802 networks.

4.4 QoS Provisioning in Software-Defined Networking

Despite the effectiveness of QoS-guarantee provided by IntServ and DiffServ, the QoS guarantee remains a challenge on a large scale. This challenge fundamentally conceptualizes to resource management and traffic directing. The current Internet architecture is based on distributed networking protocols running on network elements (e.g., routers and switches). The use of distributed protocols and coordination of changes in conventional networks remains incredibly complex to configure and policy on the underlying network hardware to enable multiple services from traffic routing and switching. Keeping the state of several network devices and updating policies becomes extra challenging when increasingly sophisticated policies are implemented through a constrained set of low-level configuration commands on commodity networking

hardware. As a result, frequently misconfigurations such as changing traffic conditions require repeated manual interventions to reconfigure the network, however, the tools available might not be sophisticated enough to provide enough granularity and automation to achieve optimal configurations.

4.5 QoS Support in Different Versions of OpenFlow

OpenFlow has supported the notion of QoS since the beginning. However, support has been limited. As new versions of OpenFlow arrived over the years, each new release of OpenFlow brought new features or updated existing ones. In this subsection, we summarize what changes each OpenFlow specification made regarding the QoS features. The earliest versions of OpenFlow OF1.0 - OF1.1 supported queues with minimum rates. OF1.2+ started supporting queues with both minimum and maximum rates. OpenFlow queues have broad support across the board. Most of the popular software switch implementations (e.g., OVS and CPqD OfSoftSwitch) and many hardware vendors (e.g., HP 2920 and Pica8 P-3290) support OpenFlow queues.

Table 4.1 QoS Related Features in Different OpenFlow Versions

OpenFlow	Specific Features
1.0	Enqueue action, minimum rate property for queues and new header fields
1.1	More control over VLA and MPLS
1.2	Maximum rate property for queues and controller query queues from switches
1.3	Introducing the meter table, rate-limiting and rate monitoring feature
1.4	Introducing several monitoring features
1.5	Replacing meter action to meter instruction

OF1.3 introduced the concept of meter tables to achieve more fine-grained QoS in OpenFlow networks. While queues control the egress rate of the traffic, meter tables can be used for rate-monitoring of the traffic prior to output. In other words, queues control the egress rate and meter tables can be used to control the ingress rate of traffic. This makes queues and meter tables complementary to each other. OpenFlow switches also have the ability to read and write the Type of Service (ToS) bits in the IP header. It is a field that can be used to match a packet in a flow entry. All these features collectively

enable the network administrator to implement QoS in their networks. The following Table 4.1 summarizes the QoS related features in OpenFlow versions.

4.5.1 Queues

As mentioned earlier, the OpenFlow protocol described a minimum rate-limiting queue in OF1.0 and a minimum and maximum rate-limiting queue in OF1.2. According to the OpenFlow specification, OpenFlow uses queues on switches but does not handle the queue management on switches. The management of queues on the switches happens outside of the OF protocol, and the OF protocol itself can only query the queue statistics from the switch. There are two protocols to provide the queue management task (creation, deletion, and alterations) in OpenFlow-enabled switches: OF-Config and OVSDB. Besides, there is standard queue management provided by any OpenFlow controller [93].

4.5.2 OVSDB

OF-Config and OVSDB are two southbound protocols to control the operations of the forwarding devices other than the forwarding decisions. In particular, OVSDB manages switch operations like tunneling, switch port status, queue configuration, and QoS management [93]. OVSDB uses many tables to manage the Open vSwitch. These tables include the flow tables, port tables, NetFlow tables, and others. Similarly, it maintains tables for QoS and Queues. While most of the other tables are root-set tables of the OVSDB schema, i.e., the table and its entries are not automatically deleted if it cannot be reached. Thus the QoS and the queue tables exist and can be altered independently, whether or not they are referenced by a port. The port table is related to a QoS table and an interface table. The relation with the interface table is mandatory, meaning that each port has to be associated with an interface. The relationship with QoS, however, is optional. A port may exist without a QoS setting attached to it. A port can have a QoS table which may have multiple queues assigned to it.

Once the QoS and queues have been set up on a switch, flows can be directed to a particular queue using the OpenFlow set queue action. This action will forward the flows that match the matching criteria to the mentioned queue. If more than one flow goes through the switch at the same time, the aggregate rate of the flows will be controlled at the egress according to the defined min rate and max rate by the queue. Let

us dive a little deeper into how the queues are implemented in the OpenFlow protocol and OVS.

The OpenFlow specification [71] states the following properties about queues:

- **min rate:** The guaranteed minimum data rate for a queue. The capacity is shared proportionally based on each queue min rate. Once the min rate is set, the switch will prioritize the queue to achieve the stated minimum rate. If there is more than one queue in one port, with a total min rate higher than the capacity of the link, the rates of all those queues are penalized.
- **max rate:** The possible maximum data rate allowed for a queue. If the actual rate of flows is more than the stated queue's max rate, the switch will delay packets or drop them to satisfy the max rate.

While OpenFlow specifications mention these guidelines for the OpenFlow compatible switches, it is left to the switch implementation to realize these features. The Open vSwitch, which is based on Linux, uses the Linux Kernel's Traffic Control (TC) program to implement queues.

4.5.3 Linux Traffic Control

Linux Traffic Control (TC) is a Linux utility used to configure traffic control in the Linux Kernel. TC can be used to achieve the following in the Linux kernel:

- **Traffic Shaping:** It can be used to shape the transmission rate of the traffic going through a Linux server or any other device. It can also smooth out any bursts of traffic for better network behavior.
- **Scheduling:** By scheduling the packets, it is possible to achieve better network behavior during bulk transfers. Reordering and scheduling of packets can also be called prioritizing, which is a widely accepted phenomenon in QoS.
- **Policing:** Several network policies can be implemented in TC. This policing occurs at ingress.
- **Dropping:** Traffic exceeding the defined bandwidth can also be dropped either at ingress or egress, based on usage.

TC uses three types of objects to achieve this: Queuing Discipline (Qdisc), classes and filters. Whenever the kernel needs to send any traffic to an interface, it enqueues the traffic into a qdisc. Which is then sent to the interface by the qdisc later. A simple qdisc is a simple FIFO queue. Classes and filters are used to implement more

sophisticated queuing disciplines like the classful and the classless queueing disciplines. In OVS, there are two classful queuing disciplines, they are Hierarchical Token Bucket (HTB) [9] and Hierarchical Fair Service Curve (HFSC) [63]. Both of these queuing disciplines allow Hierarchical Queuing Disciplines and bandwidth borrowing. Therefore, HTB will be used in this thesis for queue management.

4.5.4 Hierarchical Token Bucket (HTB)

Generally, the hierarchical token bucket (HTB) is a queuing discipline which is intended to be a replacement for Class-Based Queuing (CBQ), a standard in older TC implementations. To allow granular control over the outbound bandwidth on a given link, HTB uses the concepts of multilevel token buckets. In the HTB algorithm, tokens are generated at a fixed rate and stored in a fixed capacity bucket. If there is an available token in the bucket, packets can be dequeued or sent to an output port. Within an HTB instance, multiple classes may exist.

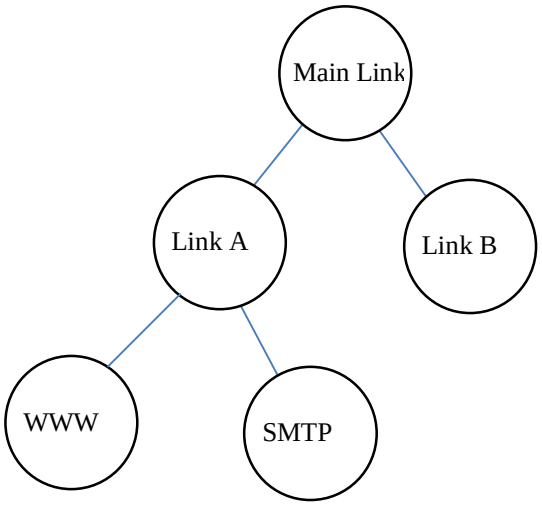


Figure 4.1 Sample HTB Class Hierarchy

Figure 4.1 demonstrates a simple HTB hierarchy for solving the following problem:

"Two customers A and B are connected to the internet via the same connection. We need to allocate 40Kbps and 60 Kbps to A and B respectively. As bandwidth needs to be subdivided into 30Kbps for WWW and 10Kbps for other applications. Any unused bandwidth should be shared among the two customers." [91]

In this example, 40Kbps is assigned to A. If A's bandwidth usage for WWW is less than the allocated bandwidth, the unused bandwidth will be used for other traffic if

demanded. The sum of A's WWW and other traffic will not exceed 40Kbps. If A were to request less than 40 kbps in total, then the excess would be given to B. However, only two levels of hierarchy can employment in OpenFlow Queue implementation because a child class of a root cannot have any children. The common important properties of HTB classes are as follow:

- **rate:** It is the maximum guaranteed rate for this class and its children. It is equivalent to a Committed Information Rate (CIR).
- **ceil rate:** It is the maximum rate at which this class is allowed to send.
- **priority:** It defines the priority of the class where the class with higher priority (priority 0 has the highest priority) is offered idle bandwidth first. This prioritization should not affect other classes' guaranteed rate.

In OVS, the ovs-vsctl command is used to create queues. This command creates an entry in OVSDB and then implements it in the switch using Linux TC. An example of creating QoS and queues in an OVS port is shown below:

```
ovs-vsctl -- set port s1-eth1 qos = @newqos -- --id = @newqos create qos type = linux-
htb other-config:max-rate = 100000000 queues = 0 = @ q0,1= @q1,2 = @q2 -- \
        --id = @q0 create queue other-config: max-rate = 10000000 --
        --id = @q1 create queue other-config:max-rate = 30000000 --
        --id = @q2 create queue other-config:max-rate = 60000000
```

Figure 4.2 Queue Implementation Example

Figure 4.2, shows how to create the example QoS and queues in port eth1 of switch s1. It is shown as new entries in the Queue table and QoS table in OVSDB. Then, OVSDB puts a relation between the newly created QoS entry and entry eth1 in Port table. Consequently, eth1 should behave according to rules stated in this QoS. The switch invokes the TC application during this process to create qdisc and classes in the background.

4.5.5 Meter Tables

The meter table is a new feature that was introduced in the OpenFlow protocol in OF1.3. Unlike queues which are used to control the egress rate, meter tables are used

to monitor the ingress rate of the flows [85]. A meter table contains meter entries, defining per-flow meters. Meters are associated with flows rather than ports. Flow entries can specify meters in its instruction; the meter controls the aggregate rate of all the flow entries associated with it. Per-flow meters enable OpenFlow to implement different QoS techniques such as rate-limiting. It can be combined with per-port queues to implement complex QoS frameworks like DiffServ.

Each flow that is attached to a meter is required to pass through the meter and meter bands before it gets forwarded. The meter measures the rate of each flow that passes through, giving options to impose operations based on rates with the help of Meters Bands.

A flow is not required to attach to a meter entry, it is up to the developer to specify which flows, or type of flows that should be attached to a meter entry and passed through the meters. A flow can also go through multiple meters. It cannot be attached to multiple meters at the same time, but it can be used in succession. This is done through different meter entries in different flow tables.

A meter table entry consists of the following components:

- **Meter Identifier:** It is a 32-bit unsigned integer identifier which is used by the flows to uniquely identify which meter entry it belongs to.
- **Meter Band:** It is the meter that measures the rate of each incoming attached flow, but it is the Meter band that hold the instructions and executes the operations based on the measured rate of the flows. Each Meter band contains the instructions to process the associated packets on what to do when a flow reaches a set rate. The meter band applies actions when the flow-rate is greater than the set rate of the meter. [85]
- **Counters:** It is a simple counter that is updated every time a packet is processed by a meter. It is mainly for statistical purposes.

A meter can define multiple meter bands, although the only one-meter band may be applied each time the packet passes through the meter. In cases where a meter has multiple Meter bands defined, only the Meter band with the highest set rate still being below the current measured flow rate is applied. In cases where the flow-rate is lower than any Meter band rates configured, no actions will be applied.

There are two band types to define how a packet would be processed; these are drop and dscp remark. Bands effect on the traffic that exceeds the defined rate. The drop

band drops the packets that exceed the rate specified in the band's rate. It can be used to define a rate-limiting band. The DSCP remark band, on the other hand, is used to increase the drop precedence of the DSCP field in the IP header. It can be used to implement DiffServ.

4.6 QoS in SDN Controllers

Several OpenFlow controllers provide additional tools and platforms to help users with queue configuration. Here, four well-known open-source OpenFlow controllers are described with a review of their QoS support [39].

- Floodlight [94], maintained by Big Switch Networks, is a well-known open-source SDN controller in the Java programming language. For queue configuration and management, a QoS module [72] is built on top of Floodlight as a community module which takes advantage of OpenFlow 1.0 queue support. Another module developed is QueuePusher [55]. It controls queue configuration and management using the Floodlight API [39].
- Ryu [96] is a component-based SDN controller implemented in the Python programming language. One of the main advantages of Ryu, unlike many other SDN controllers, is that it supports all OpenFlow versions from 1.0 to 1.5. Moreover, because of its component-based design and full OpenFlow version support, it is often used for the fast prototyping of an SDN module. Ryu provides an API to configure and manage queues in Open vSwitch using OVSDB.
- OpenDaylight (ODL) [95] is a Linux Foundation collaborative open source project with the goal of promoting SDN. Many of the proprietary SDN controllers like the Brocade SDN controller are based on ODL. Similar to Ryu, ODL also has an OVSDB plugin that helps users with queue configuration and management [39].
- Open Network Operating System (ONOS) [90] is a community open-source project hosted by Linux Foundation with the goal of creating a highly scalable, highly available, and high-performance SDN operating system. Currently, ONOS supports metering in OpenFlow. However, unlike the aforementioned SDN controllers, ONOS QoS support lacks many features and is not fully complete [39].

The following Table 4.2 summarizes the QoS related features in OpenFlow controllers.

Table 4.2 QoS Related Features in Different OpenFlow Controllers

Controller	Features and Modules
Floodlight	QoS module, QueuePusher module
Ryu	OVSDB API
OpenDayLight	OVSDB API
ONOS	OpenFlow Metering (limited QoS support)

In the experiment, the Ryu controller is chosen, mainly due to its component-based architecture, the powerful northbound API, and excellent documentation for QoS. This choice is further justified in Chapter 6.

4.7 Chapter Summary

This section presented the notion of QoS including origin and progress. It is learned that the various type of techniques can be used to implement QoS systems in SDN, various challenges faced by the SDN community and suggested possible solutions to some of the most pressing issues.

The emerging of various network services carried by the Internet, competing for the network resource and most of which require QoS performance guarantees. It is difficult to deliver the newly emerging network services in a flexible way and to fulfill the huge amount of demand with better performance in a current network. To solve these problems, traffic engineering schemes need to consider the mix of user applications, and the performance requirements the end-users may experience as a consequence of individual service improvement.

SDN aims to address the problem of flexibility in the present-day internet architecture and provides a software-driven approach for new techniques and protocols to thrive in its ecosystem. The biggest advantage of SDN is that it is easier to adapt to it and move away from the existing “rigid” internet setup. But, for SDN to replace the current architecture of the real-world network, it needs to provide very fine-grained control to the network administrators to control the quality of services along with several other enhancements.

CHAPTER 5

END-TO-END DYNAMIC BANDWIDTH RESOURCE ALLOCATION BASED ON QOS DEMAND IN SDN

SDN and QoS themselves are just concepts. A number of technologies, techniques, and applications have to work together to realize them. All the different parts of the puzzle have to fit together to create an SDN network with QoS capabilities. In this chapter, the resource allocation scheme is proposed to support Quality of Service (QoS) for various types of traffic while maintaining the network link utilization as much as high. The architecture of a resource allocation scheme based on Software Defined Networking (SDN) is presented that integrates the proposed scheme to provide better performance.

5.1 Architecture of Proposed Resource Allocation Scheme

In this section, the overall architecture of the proposed resource allocation scheme based on SDN is described in Figure 5.1.

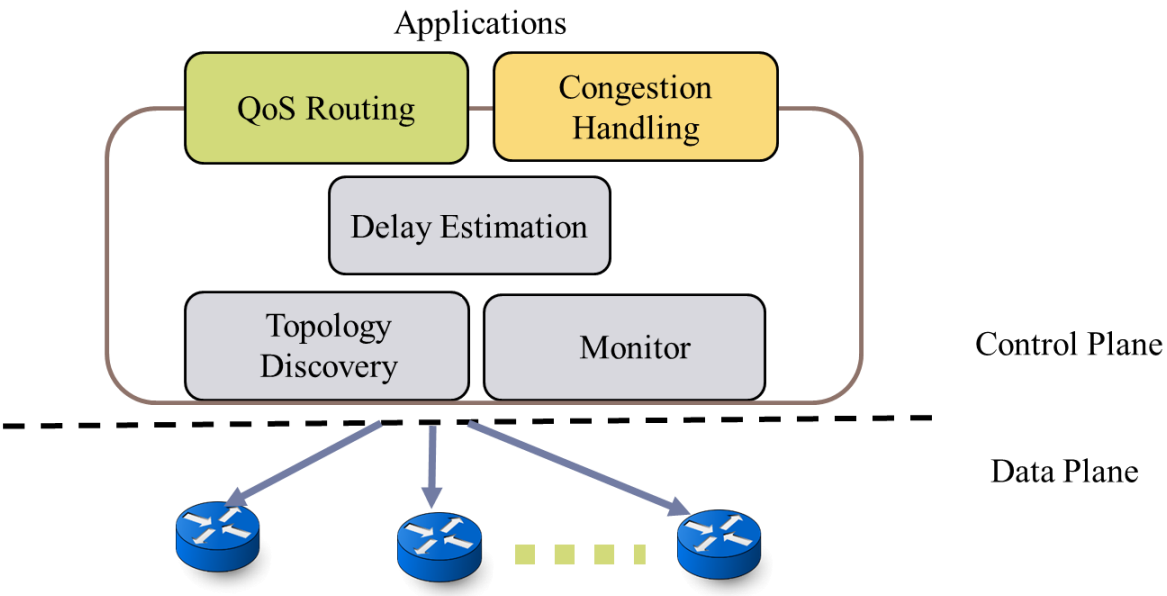


Figure 5.1 Architecture of Proposed Resource Allocation System

The system includes five main modules: topology discovery module, network monitoring module, Delay Estimation module, QoS routing module, and Congestion

Handling module. Each module has its own functions and they are linked with each other in the proposed scheme. Below we will explain the workflows of the proposed system modules individually.

5.1.1 Topology Discovery Module

This module is used to discover the SDN switches connected to the controller and have knowledge of the links between them to calculate a route for the network connection. The route cannot contract without discovering the information about the network links, hosts and switches in the network. Furthermore, keeping up-to-date visibility of the topology is a critical function. The network topology changes whenever the switches leave and join the network. Consequently, it may affect routing decisions that the controller has to make continuously.

In OpenFlow-based SDN, after an OpenFlow switch joins to the network, it establishes a TCP connection with the SDN controller. Afterward, the SDN controller requests the switch for its active ports and their respective MAC addresses using the OFTP_FEATURE_REQUEST message. The switch replies with an OFTP_FEATURE_REPLY message containing the requested information which is needed for topology discovery. Although there is no specific standard for discovering the topology of an OpenFlow-based SDN, most SDN controllers' implementations follow the OFDP protocol relying on LLDP packets [62]. Therefore, the topology discovery module firstly sent out the Link Layer Discovery Protocol (LLDP) packets to all the connected switches through packet_out messages to acquire topology and connection information. After that, the messages instruct the connected switches to send LLDP packet_out messages overall its ports to other connected devices. Then this message would be delivered to the controller as packet_in messages since the switch does not have a flow entry for this LLDP message. These packet_in messages contain information about the switch's port that the specific host connects to. SDN controller creates a connection based on these packet_in messages. In this way, global topology information can be gained. LLDP messages are periodically exchanged to check whether the connection links go up or down. The collected information of switches and links, including MAC and IP address of all the connected hosts in a database called topology database. Figure 5.2 shows the detailed steps of how the network detection module works with the SDN controller to discover the network topology.

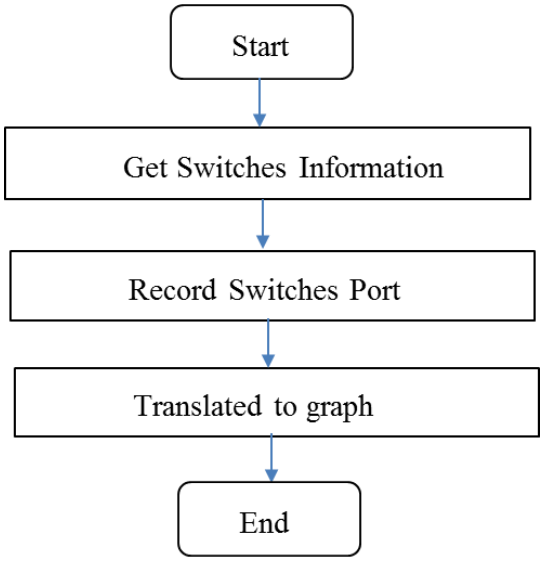


Figure 5.2 Topology Discovery Module

5.1.2 Network Monitoring Module

This module is used to do real-time measurements of the network. It is very hard to calculate the route without knowing link load information for all the relevant links. However, SDN solves the problem and makes it much easier for both users and operators because the controller already has the global network view and access to all the information of topology. The controller keeps track of how much bandwidth is allocated in the network.

In order to achieve some information like link utilization and the network topology updates, the controller listens to asynchronous messages such as OFPT_PACKET_IN message, OFPT_FLOW_REMOVED message and OFPT_PORT_STATUS message from each switch. The monitoring module tracks the amount of traffics by periodically polling flow statistics such as received and transmitted bytes or packets from all connected switches and takes a snapshot of the current network status. The module calculates link utilization and available bandwidth for bandwidth allocation. To calculate the link utilization of each link i for every time unit can be computed by using the number of transmitted bytes from the port statistic as follows:

$$LU_i = [\ B(i, t_{j+1}) - B(i, t_j)] / [t_{j+1}) - (t_j)\ ] \qquad \text{Equation 5. 1}$$

Where t_j, t_{j+1} indicate the two consecutive responses time and the number of transmitted bytes reported at time t_j for link i is denoted as $B(i, t_j)$. $B\ (i, t_{j+1})$ indicate the

number of transmitted bytes reported at time t_{j+1}. Then, the available bandwidth (ABW_i) of each link can be computed simply by subtracting the link utilization (LU_i) from the network bandwidth capacity (BW_i) as follows:

$$ABW_i = BW_i - LU_i \qquad \text{Equation 5. 2}$$

After calculating available bandwidth, the monitoring module sends this information to the QoS routing module to compute routes to deliver the traffic form the source node to the specific destination node. Figure 5.3 depicts the work flow of network monitoring module in detail.

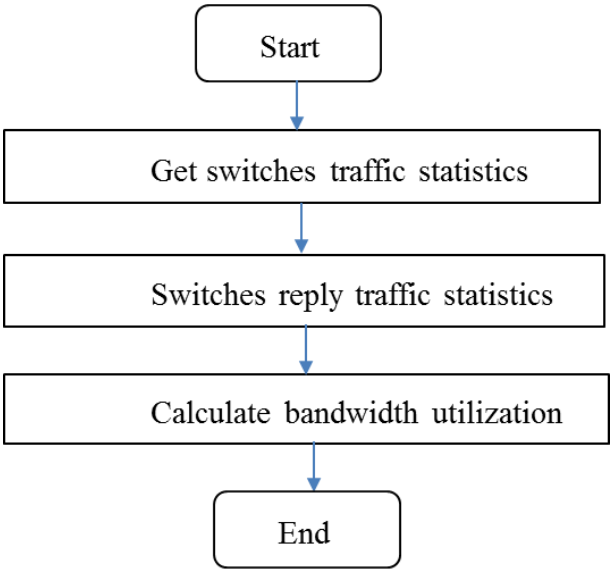


Figure 5.3 Network Monitoring Module

5.1.3 Delay Estimation Module

This module is responsible to estimate a real-time delay of the network. This module adopts the LLDP protocol which is described in the topology discovery module since the SDN controller discovers all the links, in both directions, to updates its view of the network periodically. Figure 5.4 shows the workflow of the delay estimation module.

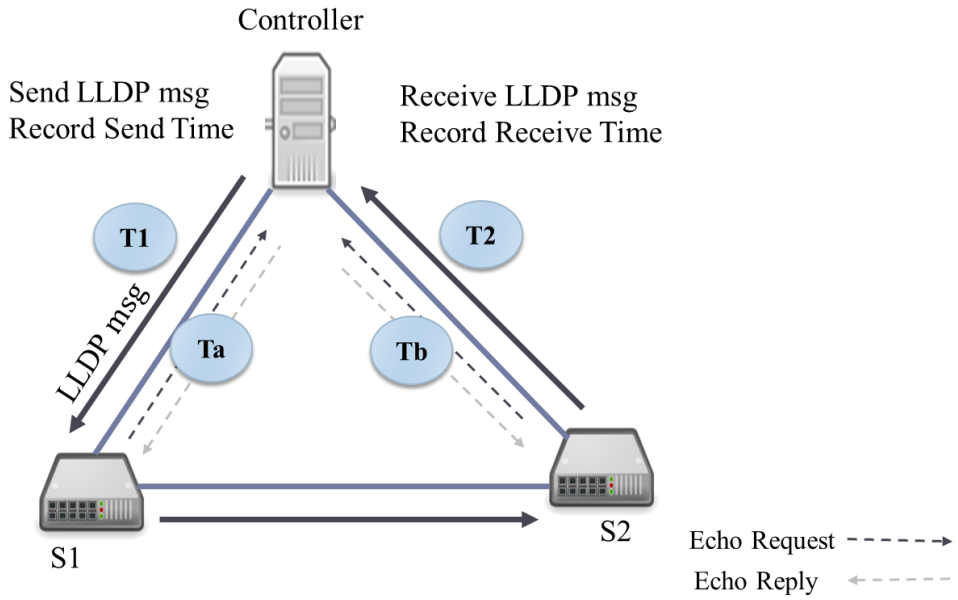


Figure 5.4 Delay Estimation Work Flow

At first, the controller sets the timestamp at the beginning of the LLDP data transmission and then subtracts the received timestamp to estimate the delay from the controller to the switch S1. Then, from switch S1 to switch S2, and then report the delay T1 to the controller, an example is shown in the thick black arrow of Figure 5.4. The same inverse delay T2 consists of grey arrows. In addition, the controller-to-switch round-trip delay consists of a light black arrow and a grey arrow. This part of the delay is tested by the echo message, Ta, Tb.

To get the logic of T1 and T2, the measurement method is as follows: the data from the Switches module are necessary. First, the LLDP packet is parsed from Packet_in to obtain the source DPID and source port. Then, according to the data of the sending port, the sending timestamp data in the port data is obtained, and the sending timestamp is subtracted from the current system time to obtain a delay, and finally saved to the graph data.

After that, this module needs to test the echo round-trip delay between the controller and the switch. The measurement method is as follows: the controller sends a time-stamped echo_request message to the switch, and then parses the echo_reply returned by the switch, and subtracts the sending time of the data part from the current time to obtain the round-trip time difference. So the implementation of the timing and parsing of echo_request is necessary.

After the calculation of the echo delay is completed, it is saved in the echo_latency dictionary and is ready for subsequent calculations. After the delay data is obtained, it is also necessary to calculate the delay of the link based on the data, and the formula is:

$$T=(T1+T2-Ta-Tb)/2 \qquad \text{Equation 5.3}$$

5.1.4 QoS Routing Module

The main function of this module is to find the best path to alleviate network congestion and improve the QoS of network applications such as media streaming and online games which require strict QoS guarantees. In general, network providers optimize their network performance in order to effectively fulfill customer demands with traffic engineering (TE). Routing is a powerful tool of TE, and it can be used for controlling network data flows. The aim of TE routing is to route network data flows as much as possible by reserving the required bandwidth resource for each established route. A routing engine needs to select a route between a source and destination for each traffic flow.

This module uses the network topology information from the topology discovery module and the traffic statistics from the monitoring module to compute multiple paths and pushes the resulting computation as flow rules to the SDN switches. Route calculation module calculates the shortest path tree from each source node to all the destinations by applying the shortest path finding algorithm, Dijkstra.

This module uses Dijkstra's shortest path algorithm [59] to find a set of candidate paths between a pair of source and destination. Dijkstra's algorithm calculates the shortest path between two nodes on a network using a network topology graph. It can assign a cost value to every node. Set it to zero for the initial node (source node) and infinity for all other nodes. Firstly, the algorithm divides the nodes into two sets: tentative and permanent. Then, it chooses nodes, makes them tentative, examines them, and if they pass the criteria, makes them permanent. The outline of the Dijkstra's algorithm can be expressed as shown in algorithm 5.1, [2], [59]:

Algorithm 5.1: Dijkstra's Algorithm

Step 1: Start with the source node: the root of the tree.

Step 2: Assign a cost of 0 to this node and make it the first permanent node.

Step 3: Examine each neighbor node of the node that was the last permanent node.

Step 4: Assign a cumulative cost to each node and make it tentative.

Step 5: Among the list of tentative nodes

i. Find the node with the smallest cumulative cost and mark it as permanent. A permanent node will not be checked ever again; its cost recorded now is final.

ii. If a node can be reached from more than one direction, select the direction with the shortest cumulative cost.

Step 6: Repeat steps 3 to 5 until every node becomes permanent.

All the paths are stored in HashMap <key, value> form is used to store all the paths, and later the controller will use to determine the routes for different types of traffic with their QoS constraint. When a new flow arrives to the OF switch, it will send to the controller if the OF switch does not have flow entry for it. According to the flow information including in the packet header fields, the controller will select a suitable path with a sufficient amount of bandwidth available for it and send back to the OF switch as flow entries for packet forwarding.

Whenever a new flow with bandwidth request arrives, the controller allocates the demand flow based on the current link utilization. After calculation possible path lists, check the available bandwidth of the path which can be implemented by using the statistic of the network monitoring module. If it is enough for the bandwidth guarantee rate, the controller selects the path as the optimal candidate path for routing. If it is not enough, the link is simply removed to avoid link performance degradation. After selecting the routing path, the controller updates the flow table of the switches along the path. Then, QoS mapping is implemented for QoS flow with a priority queue to provide a bandwidth guarantee.

5.1.5 Congestion Handling Module

Over-demand of network resources can cause network congestion which may lead to performance degradation. It is therefore imperative to assume that network resources are sufficient to cater for the offered traffic most of the time. One way to deal with congestion on the hardware side is to increase the bandwidth on links in the networks. However, providing an oversupply in bandwidth is expensive and packet loss in a large network like data centers is primarily happening on links that are not heavily utilized on average. Hence, increasing the bandwidth would not solve the problem. The reason for the losses can be found in the bursty nature of the network traffic which causes congestion when multiple traffics flows transmitted on the same link produce high peaks simultaneously.

The aim of the congestion control module is to provide network services to its users that meet certain performance criteria, often represented by a set of QoS parameters. One of the challenges in meeting QoS requirements is avoiding bandwidth starvation of certain traffic types by others. Flow rerouting is one of the ways to deal with the QoS degradation of the flows in the heavy network. A natural method for congestion control is using the low load path. Such routing also achieves load balancing of the network resources. The controller needed to reroute some of the current flows on the bottleneck link is detected to mitigate the flow congestion. Figure 5.5 shows the workflow procedure of the flow rerouting and the proposed flow rerouting algorithm will introduce in section 5.2.4.

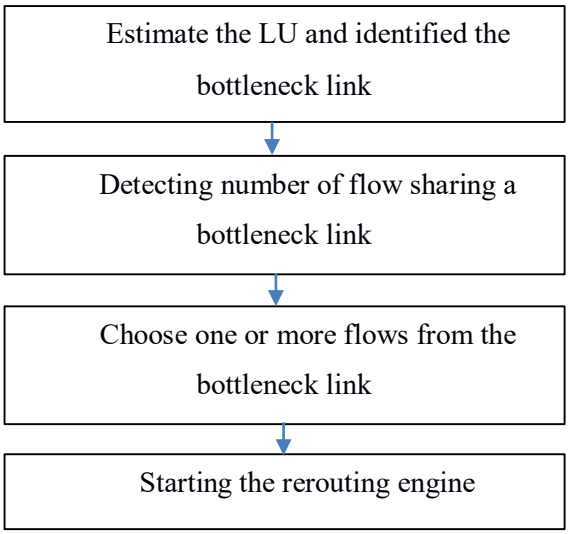


Figure 5.5 Workflow to Start the Flow Rerouting

To accomplish the best accommodation of resources possible, the rerouting algorithm is implementing with a non-dominated sorting genetic algorithm (GA) while focusing on rerouting the flows with the highest priority traffic. NSGA is an extension of the Genetic Algorithm for multiple objective function optimization. NSGA-II [20] is one of the most popular multi-objective optimization algorithms with three special characteristics, fast non-dominated sorting approach, fast crowded distance estimation procedure, and simple crowded comparison operator. NSGA-II can be roughly expressed as the following steps. Algorithm 5.2 describes the outline of the NSGA-II procedure in detail.

Algorithm 5.2: NSGA-II Algorithm

Step 1: Population initialization: Initialize the population based on the problem range and constraint.

Step 2: Non dominated sort: Sorting process based on non-domination criteria of the population that has been initialized.

Step 3: Crowding distance: Once the sorting is complete, the crowding distance value is assigned front wise. The individuals in a population are selected based on rank and crowding distance.

Step 4: Selection: The selection of individuals is carried out using a binary tournament selection with a crowded-comparison operator.

Step 5: Genetic Operators Real coded: GA using simulated binary crossover and polynomial mutation.

Step 6: Recombination and selection: Offspring population and current generation population are combined and the individuals of the next generation are set by selection. The new generation is filled by each front subsequently until the population size exceeds the current population size.

Subsequently, the NSGA-II algorithm selects the flow with the highest priority traffic and checks out if there is any other parallel route for this flow with enough free capacity to carry its traffic. In the case that there is another possible path with sufficient capacity, the flow will be routed through that route, sending the corresponding flow entries to each of the OpenFlow switches. Once the flow with highest priority traffic has

been routed across another path, the process starts again, and, in the case that the congestion still exists, the same procedure will be followed, moving the highest priority traffic flows along another route.

5.2 The Proposed End-To-End Dynamic Bandwidth Allocation Scheme

A network is modeled as a graph $G = (N, E)$, where N is a set of nodes and E is the set of (directed) edges. Every edge $(i, j) \in E$ has two associated properties: link capacity c_{ij} reflecting the bandwidth available to the corresponding link and the user required bandwidth r_{ij}. The link capacity is usually fixed, while the residual bandwidth is varied based on traffic on the link. The main notations of the proposed scheme are presented in Table 5.1.

Table 5. 1 Main Notations

Symbol	Definition
$G = (N,E)$	the network graph
N	the set of nodes
E	the set of (directed) edges
(i, j) ∈ E	the link between switch i and switch j
Lbw	the link bandwidth
Mbw	the maximum bandwidth usage
$D(s, d, r)$	the flow demand matric
s	source
d	destination
r	the user demand bandwidth
P_{ij}	the path from switch i and switch j
Lu	the link utilization
p	priority
B	the number of transmitted bytes
t_j , t_{j+1}	two consecutive responses time

A demand matrix $D = D (s, d, r)$ expresses the traffic demand from node s to node d in the network where all the links have required bandwidths equal or greater than

r. If the user demand can be accepted, the controller reserves bandwidth of *r* (Mbps) along path P_{ij}.

The outline of the proposed end-to-end dynamic bandwidth allocation scheme based on user QoS demands are presented in Table 5. 2.

Table 5.2 Outline of the Proposed Scheme.

Input	A network *G (N, E)* with necessary information e.g. link bandwidth Flow demand *D (s, d, r)*
Output	A feasible path p_i or no route satisfying the demand
Steps	For each *D (s, d, r):* 1. Find a feasible path satisfying QoS requirements of the flow. 2. Allocate bandwidth along the path. 3. Estimate the link utilization and identified the bottleneck link. 4. Reroute the highest priority flow.

The implementation of the proposed QoS routing scheme can be divided into two levels; the controller and switch levels. The controller calculates the feasible path based on the user's demand QoS in Module 1 that is flow-based routing to provide the QoS for the individual flow. Moreover, the flow rerouting algorithm is proposed in Module 2 that is responsible for congestion management in flow-based routing at the controller level of the proposed scheme.

5.2.1 Bandwidth Allocation at Controller Level

When a user wants a desire QoS such as bandwidth, the user can request the controller by sending a request packet which includes the flow information such as the source, destination and the required QoS factors such as the amount of bandwidth they need and a delay tolerate value. When the controller receives the request packet, the controller starts the routing engine and calculates the route for bandwidth allocation according to the user QoS demand by using topology and monitoring engine.

Finally, the routing decision is issued by the per-flow routing policy. According to the demand QoS factors of bandwidth and network conditions, path selection is carried out for each flow which is advantageous to network resource orchestration and QoS guarantee. The SDN controller seeks the feasible paths that satisfy QoS

requirements of flow based on user demand. Then, the SDN controller enforces the QoS policy in the data plane.

5.2.2 Bandwidth Allocation at Switch Level

After calculating a feasible path for the request flow at the controller level, the proposed system tries to provide network resources to the flow at the switch level by taking advantage of the queue mechanism supported by OpenFlow protocol. Queuing allows us to ensure that important traffic, applications, and users have precedence. Each output interface can configure eight queues as the maximum number of queues per interface and flow entries mapped to a particular queue is treated according to the configuration of the queue. The controller maps the incoming flow according to its flow demand into the pre-create queues, and it installs the forwarding rules on each SDN switch over the determined path to support the QoS guarantee.

5.2.3 Module 1: Flow-based Routing

The hierarchy of the proposed QoS routing work flow has described in Figure 5.6.

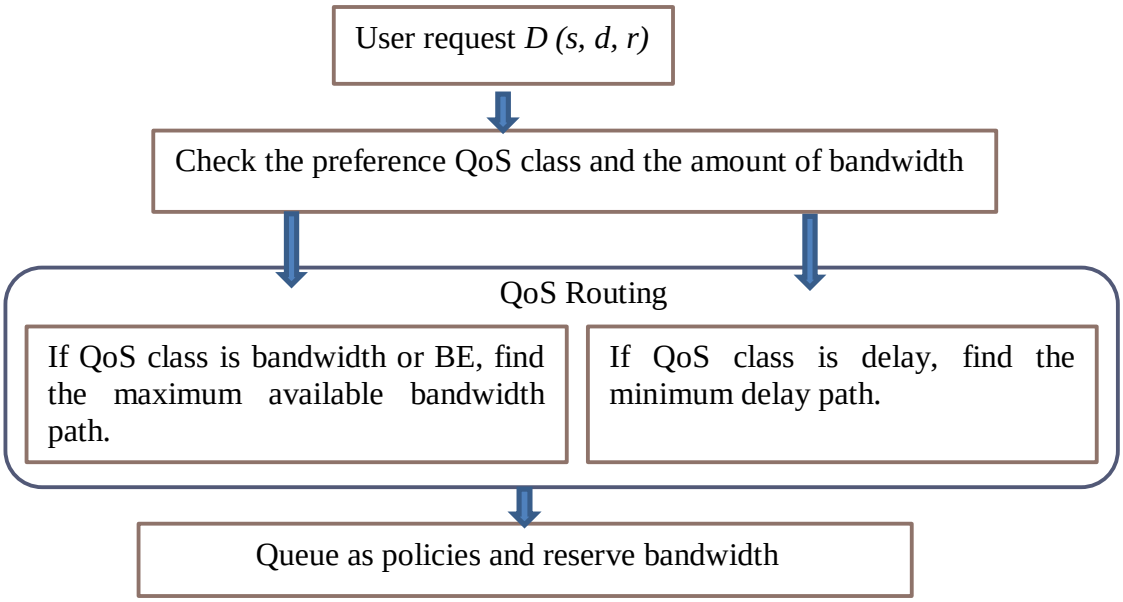


Figure 5.6 Hierarchy of QoS Routing

Let us consider the network traffic as in three classes: two QoS classes and the best-effort class (non-QoS). Whenever a new flow has arrived, the controller extracts

flow information such as source node, destination node and request bandwidth. The controller checks the flow priority information which can show the incoming flow is QoS class types or best-effort class.

In the proposed scheme, the routing engine firstly finds the most feasible path for required QoS. A feasible path can provide sufficient resource *r* to satisfy all the QoS requirements of the flow. For example, if the incoming flow type is the bandwidth demand QoS class or the Best-effort class, the QoS routing module chooses the maximum available bandwidth path by calculating link utilization. For the minimum delay demand QoS class flow, the QoS routing module calculates the link delay between the source and the destination nodes. Then, choose the minimum delay paths in order to meet the QoS requirement of the flow. After the path is calculated, the SDN controller installs flow entries to each switch along the path and updates the flow database of the switches along the path. Then, QoS mapping is implemented for QoS flow with a priority queue to provide a bandwidth guarantee. Ovs Switch can forward the packet by using the flow rule.

5.2.4 Module 2: Flow Rerouting

The proposed scheme attempts to avoid traffic congestion when a new flow is added to the link. The accepted flows bandwidth is investigated by reserving the required bandwidths (*r*) for incoming flows to know the maximum bandwidth usage (*Mbw*) by Equation (5.4):

$$Mbw\,(\text{new}) = Mbw\,(\text{old}) + r \qquad \text{Equation 5.4}$$

After calculating the maximum bandwidth usage, the controller checks whether the usage bandwidth exceeds a predefined threshold to identify the bottleneck link. Identifying the network link bottleneck is very useful for both end-users and service providers. By identifying the bottleneck link, the proposed system can eliminate paths that have lower bandwidth and reroute the traffic over the bottleneck link to an alternative path with the highest bandwidth by using QoS routing. If the link bandwidth is greater than the predefined threshold value, we define the link as the bottleneck link and reroute the highest priority flow from the bottleneck link to an alternative link that has enough bandwidth for the rerouting flow.

If the link utilization exceeds a predefined threshold by allocating the new flow requested bandwidth, the controller reallocates the network resources by using the alternative path that has enough bandwidth to allocate the reroute flow. A typical routing algorithm routes just one flow in each step while the proposed rerouting algorithm reroutes one or more flows according to their priorities and reserved bandwidth to reduce the packet loss rate and provides higher QoS performance to the users. The proposed flow rerouting algorithm is presented in algorithm 5.3.

Algorithm 5.3: Rerouting Algorithm

Input: *G=(N, E), Lbw, Lu* of bottleneck links, flows on the bottleneck link, number of paths
for flow in flows do:
extract the flow information *(s, d, r, p)*
list the flow ascending order according to *p*
end for
String FlowChoose ()
    for flow in listed flows:
        Chosen Flow = []
        Mbw -= r
        If *Mbw* > predefined value, then
            Chosen Flow += flow
        else return Chosen Flow
    end for
Invoke NSGA-II (Chosen Flow, number of paths)
    return optimal candidate path, max-Mbw
Reroute the flow to optimal candidate path
Update *p* by adding 1 //to prevent repeatedly rerouting
Update the flow table along the path
Queue in priority queue interface of each switch
Reserve request bandwidth along the path

5.3 Chapter Summary

It is difficult to deliver the newly emerging network services in a flexible way and to fulfill the huge amount of demand with better performance in a current network. Traditional routing cannot provide accessible performance for all traffic types, and it can cause excessive link utilization in the network. Another common problem that arises in networks that have to deal with large amounts of traffic is congestion. When a network device is receiving more data packets than it can process, the packets are delayed or lost which in turn reduces the overall throughput of the network. This certainly lowers the performance of the services and leads to a dissatisfying end-user experience.

This chapter presents the proposed end-to-end dynamic bandwidth resource allocation scheme design in detail. The proposed scheme work is based on the QoS demand of the network users in the SDN network. We will demonstrate the effectiveness of the proposed scheme will be shown on the emulated SDN network in chapter 7. The details of individual experiments will be provided in-depth in the next chapter on implementation.

CHAPTER 6

DESIGN AND IMPLEMENTATION OF THE PROPOSED SYSTEM

This chapter covers the design for the implementation of the experiments. Furthermore, essential modules of the developed SDN application are adequately described to provide the reader an understanding of how the platform operates. This chapter instantly begins with an introduction of the proposed end-to-end QoS implementation, explains the applied SDN controller, and offers the mandatory information relating to the technology and options used in the implementation.

6.1 The Proposed End-To-End QoS Implementation

Fundamentally, QoS can offer better service to certain flows. This is often done by either raising the priority of a flow or carefully limiting the priority of another flow. Once using congestion-management tools, the network administrator tries to lift the priority of flow by queuing and servicing queues in different ways. Generally, the queue management tool used for congestion avoidance raises priority by intentionally dropping lower-priority flows before higher-priority flows. Policing and shaping give priority to flow by limiting the throughput of different flows.

Generally, the type of flow must be identified to provide preferential service to an individual flow. Common ways of distinguishing flows embody access control lists (ACLs), policy-based routing, committed access rate (CAR), and network-based application recognition (NBAR). The proposed approach has been implemented based on policy-based routing.

6.1.1 Class of QoS

Each year the network service usages are growing and 2019 will be no different. The services differ in their level of QoS strictness, that the service can be bound by specific bandwidth, delay, jitter, and loss characteristics. On the other hand, the current IP-based network faces significant challenges in providing some types of service guarantees for various types of traffic. This has been a specific challenge for streaming

video applications, which regularly need a significant quantity of reserved bandwidth to be useful.

According to a new report from Cisco [101], by 2019, online video is accountable for four-fifths of worldwide Internet traffic. In order to provide a good viewing experience, video streaming services have strict requirements on bandwidth and delay. Since the Internet is designed to offer best-effort services (i.e., no guarantee on bandwidth and delay) for cost efficiency as well as better reliability and robustness, it is essential and difficult to provide a Quality-of-Service (QoS) guarantee for video streaming services.

Another highly demanded service in the current network is VoIP. Compared to video streaming, VoIP traffic does not consume a large amount of bandwidth but have different and stricter QoS requirements. Using a voice service implies that users interact with each other, since the service is rather sensitive to delays and jitter in the communication, due to the bi-directional nature of a teleconference or voice call. For instance, if the user must wait too long for the other user to respond, then the conversation can end up, thus the experience is affected. The category includes teleconferences and calls with and without video, and will simply be referred to as Voice. Moreover, a category that covers the fundamental service which may include HTTP, FTP, SNMP, PoP3 and Telnet can be specified as the best-effort service type. Hence, these types of services are robust regarding network traffic conditions and not nearly as sensitive to varying network conditions as voice and video are.

Table 6.1 Example QoS Types and QoS Requirement

Types	**Characteristics**	**QoS requirement**
Voice	Alternative talk-spurts and silence intervals Talk-spurts produce constant packet-rate traffic	Delay <~ 150 ms Jitter <~ 30 ms Packet loss <~ 1%
Video	Highly bursty traffic Long range dependencies	Delay <~ 400 ms Jitter <~ 30 ms Packet loss <~ 1%
Best-effort	Poisson type Sometimes bursty, or sometimes on-off	Zero or near zero packet loss Delay may be important

This leads to three categories: Voice, Video and Best Effort. They will be the basis for the traffic to study when determining the requirements for the network. Table 6.1 describes the example of QoS types and their requirements for network communication [103].

Since the QoS is at the forefront of the present networking, the future internet brings us the notion of user-based QoS in which QoS policies are based on a user as well as application. Therefore, the user demand QoS approach is considered to draw the QoS policies on our work. In the proposed approach, we differentiate the QoS traffic classes based on these baseline QoS requirements as shown in Table 6.1. In order to allocate an appropriate route for each traffic, we assume that network users register their preferences of QoS demands to the controller. The controller maintains the registered information and finds the most feasible path for each QoS demand. The possible options for preferences QoS demands are minimum delay, bandwidth and default (best-effort). Table 6.2 shows the available QoS classes in the proposed approach.

Table 6.2 Available QoS Classes

QoS class	**Applications**
Minimum-delay demand	Gaming , remote control application (haptic application), VoIP
Bandwidth demand	Multimedia streaming (voice, video), data storage
Best effort	Telnet, FTP, HTTP

6.1.2 Flow Requirements

Individual flows may have certain network performance requirements, such as bandwidth, delay, minimum error rate and so on. If a flow carries a significant video stream, then it is of interest to forward the traffic down a path that supports the capacity requirements. To make sure that the flow maintains its requirements in the future, reservations would be necessary to prohibit other traffic streams to occupy resources on the same route (QoS).

Since the SDN controller is the Policy Decision Point, it can be developed to own full management of the incoming flows to the network. By programming the

controller to get the network topology information, it is possible to make forwarding decisions based on its information regarding the flow and map this to the topology. Information about particular flows must be predefined, for example, the minimum bandwidth requirements of the flows. Additionally, the controller would also need to store flow reservations in a database and maintain state.

6.1.4 Flow Priority

Among the network, the transmitted traffic may have totally different priorities based on the user's QoS-demand. An SDN controller is programmed to handle traffic differently and assign priority to the flows. This can be realized either by using the priority field within the flow rules or by maintaining state regarding the policy priority. The prioritized traffic should be able to meet the desired QoS factor once it arrives to the destination at the other end. Due to possible network capacity limitations, there should be trade-offs once the requirement for capacity is higher than what the network can offer. Therefore, prioritizing of traffic could be a mitigation strategy to confirm that the highest priority traffic is distributed and received across the network without service degradation too much then the lower priority traffic. The priority ranges from 1 to 16, where 1 is the highest priority. Configuring the flow priority is the important factor for the application, and it will become the primary factor when the network faces the congestion, the top priority flow will reroute first.

6.1.5 Queue Implementation

This section is responsible for configuring queues on the output interfaces of the switches and maintains the queue configuration information. As a maximum number of queues per interface, each output interface can configure eight queues. For our study, only three queues are created for each output interface of the switches. Flows are classified into different levels and allocate network resources dynamically to provide high QoS for each traffic. Different types of traffic will be transmitted through different queues. For example, the QoS-flows will queue into the high priority queue to acquire sufficient bandwidth resources since the cross-traffic queue has the lowest priority.

Rate guarantees can be classified into Soft QoS and Hard QoS. From the point of implementation view, Soft QoS is more flexible but it does not provide very strong guarantees. On the other hand, Hard QoS guarantees are rigid that reserves a portion of

the bandwidth to be used only by a specific flow. It has stringent policies on the admission of the flow. If the required bandwidth is not available, the flow is rejected at the ingress itself.

In the study, three different queues are set at a port and assign different priorities to them. Therefore, one of the three queue priorities can be assigned for the different QoS flows. The incoming flows are categorized into the three QoS priority classes (high, medium, and low) and map with the priority queue according to their flow properties. For example, services like voice and video applications which are particularly sensitive to latency but less sensitive to packet loss can be mapped to the high priority queue. A QoS policy rule is assigned to the QoS priority flow associated with the rule. However, the QoS priority has differed from the queue priority. From the perspective of the flow priority, bandwidth demand flows is set as the highest priority flow in the proposed. When the controller finds the bottleneck link, the controller reroutes the flow based on the QoS flow priority.

6.1.6 Policy Setting

Each application or client has its own set of requirements, typically defined in their Service Level Agreements (SLAs). Quality-of-Service (QoS) requirements include end-to-end bandwidth and latency among other attributes, as we discussed in the previous session. This section will present the policy setting and policy lists. It will use to calculate the route that can meet the QoS requirement of the network application as the user demand. Before the connection setup, the user needs to register the required bandwidth and favorable QoS demand factor. Based on this information, a policy will draw to meet the user QoS-demand. The policy-setting will load at the start-up phase of the Ryu controller. In the run-time phase, all of the incoming flows will be checked against this list.

Each policy contains a pair of match conditions and actions. Match conditions are defined to map a policy with a particular flow. The policy needs a minimum of one match condition to figure for the QoS_policy. The policy list is applied to at least one direction of the flow which means the different policies can fetch for each direction of the flow. For instance, a host (h1) initiates a connection to host (h2), the policy-check will run for h1-h2. After matching, the actions will follow. A policy can be designed to

have several actions. Each policy is configured to own priority and enqueue-id in actions.

Table 6.3 Possible Policy List

QoS_Policy	Match	Action
1	user_ip or server_ip	bandwidth_request = *r* Mbps, set_priority = 1, set_queue = 2
2	user_ip or server_ip	bandwidth_request = *r* Mbps, set_priority = 8, set_queue = 1
3	user_ip	bandwidth_request = 0 Mbps, set_priority = 16, set_queue = 0

When a new incoming flow arrives, the controller will find a matching policy. Table 6.3 lists the possible policy list. The example of the policy structure is as follow:

- If the incoming flow is a min_delay demand flow, it will against the QoS-Policy1 by reserving the user request bandwidth *r*.
- If the incoming flow is a bandwidth demand flow, it will against the QoS-Policy2 by reserving the user request bandwidth *r*.
- If the user does not make register for the QoS demand, the controller will draw the default policy without reserving bandwidth.

When a flow is coming to the network, the network administrators define policies according to the pre-register information of the users. The register information of the users is stored in the policy list. The controller will make a forwarding decision based on the specific policy. Whenever the incoming flows reach the controller, the application will browse the policy list for it and check packet parameters against match conditions. Once the policy is accepted, the application will then compile the policy into flow rules, where it will configure the MAC source and destination addresses of the communication entities as match conditions for the flow rules. After a policy is applied and used in the network, it is keeping to a different list however further parameters are added, such as chosen path and flow information. Figure 6.1 illustrates the logical design of the policy storage with the forwarding decision process.

This list saves the enforced policies, that the controller keeps the state of every running policy on each path within the network. The forwarding decision process will

examine this list if it is necessary to reroute a flow in the congestion handling process. Despite this, it is necessary to notice that the figure only displays the policy process; the controllers monitored the view of the topology and traffic influences the forwarding decision process.

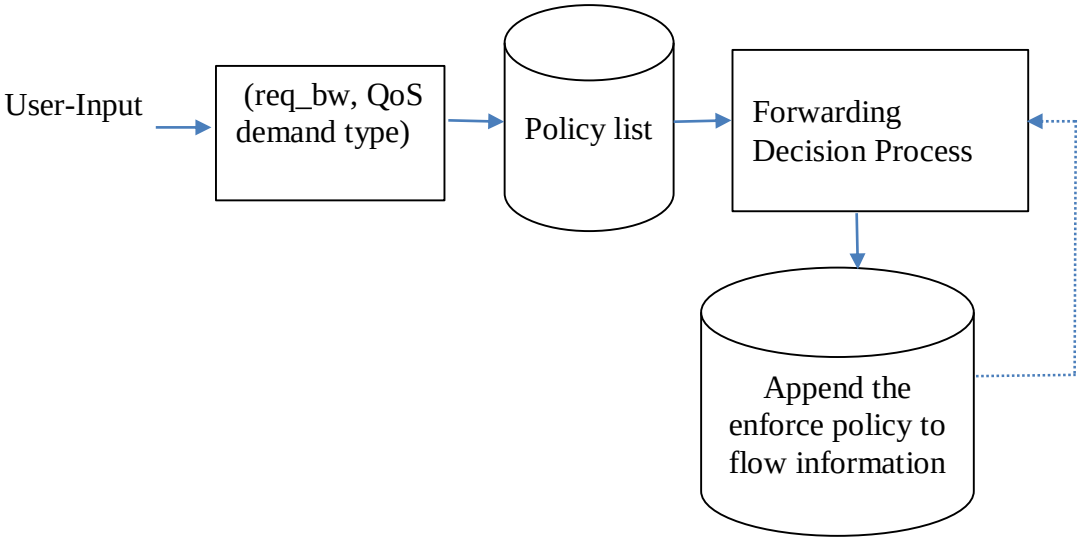


Figure 6.1 The Logical Policy Storage

6.1.7 Forwarding Decision

Traffic engineering mechanisms which can police the traffic to mitigate the network or parts of the network fail because of the network bottleneck link (too heavy loaded link). The traffic should be spread across the network to prevent congestions on individual links. When connections break down or fail, it is congested in the network link, it is important to undertake to seek out new paths to the destination. Re-routing is an important ability that the network must perform in a prompt manner. Another interesting component of routing is to enforce randomness to avoid predictable flowing paths, which could be advantageous in a security perspective. The controller is programmed to find alternative ways by obtaining topology information, once the network congestion happens, as well as use random generation algorithms to choose paths when incoming flows arrive at the network. The controller can maintain the paths which were previously chosen.

6.2 Ryu Controller

In the SDN environment, it is essential to select the controller to conduct the proposed application. The common open-source SDN controllers were already introduced in Chapter 4. In this section, an SDN controller, Ryu [96] will be discussed through that operations flow. The logo of the Ryu controller is dragon and Ryū in Japanese stands for a dragon. Ryu is usually mentioned as component-based, open-source software-defined by a networking framework. It is supported by NTT's labs and executed entirely in Python. Figure 6.2 below depicts the Ryu framework and its main components [96].

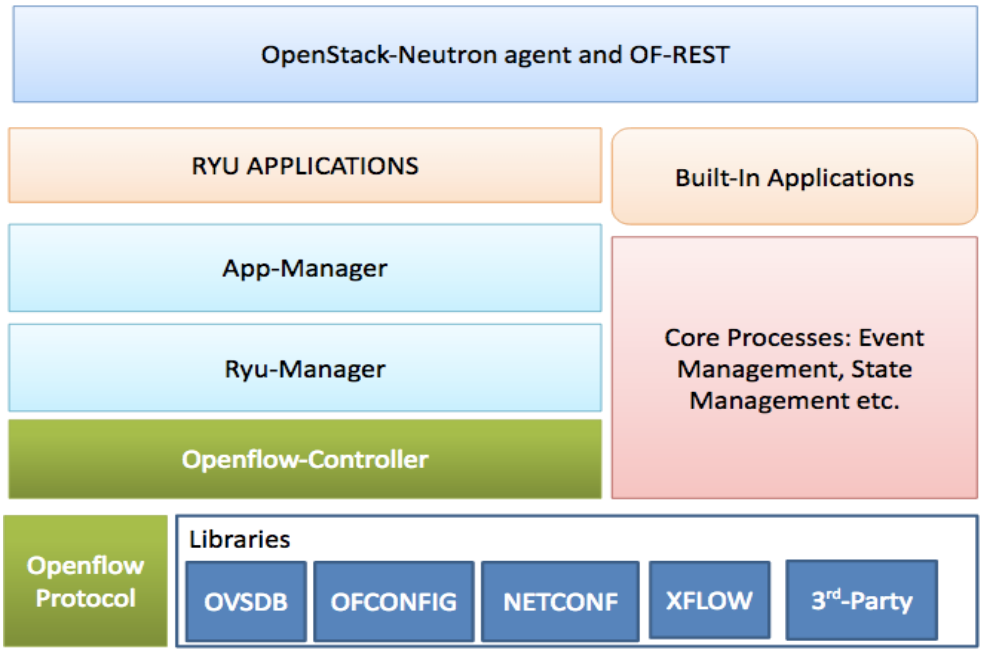


Figure 6.2 Ryu Framework

In the framework, Ryu provides software components with well-defined APIs. By using these APIs, the network developers can make new network management and control applications. Additionally, Ryu supports multiple southbound protocols for managing devices, like Network Configuration Protocol (NETCONF), OpenFlow, OpenFlow Management and Configuration Protocol (OF-Config), and others. The vital components of the architecture are explained in the sub-session bellow.

6.2.1 Ryu Libraries

Ryu supports several libraries and multiple southbound protocols. Relating to southbound protocols, Ryu also supports Open vSwitch Database Management Protocol (OVSDB), OF-Config, NETCONF, Sflow [99], Netflow [105], and other third-party protocols. Sflow and Netflow protocols can be used for network traffic measurement by using various methods such as packet sampling and aggregation. The third-party libraries embody Open vSwitch Python binding, the Oslo configuration library and a Python library for the NETCONF client. With the help of Ryu's packet library, the network developer can analyze and build several protocol packets, like VLAN, MPLS, etc.

6.2.2 OpenFlow Protocol and Controller

The Ryu framework includes an internal controller and the OF protocol which is one of the supported southbound protocols. Ryu supports the OpenFlow protocol starting from version 1.0 to the latest version 1.4. Table 6.4 summarizes the OpenFlow protocol messages and corresponding API of the Ryu controller.

Table 6.4 OpenFlow Protocol Messages and Corresponding API of Ryu

Controller to switch message	**Asynchronous message**	**Symmetric message**	**Structures**
• Handshake • switch-config • flow-table-config modify/read state • queue-config • packet-out, barrier • role-request	• Packet-in • flow-removed port-status • Error.	• Hello • Echo-Request & Reply • Error • experimenter	• Flow-match
• send_msg API and packet builder APIs	• set_ev_cls API and packet parser APIs	• Both Send and Event APIs	

In the Ryu architecture, the OpenFlow controller is one of the internal event sources and which can manage the switches and events. In addition, Ryu includes an OpenFlow protocol encoder and decoder library.

6.2.3 Managers and Core-processes

The main executable component in the Ryu architecture is the Ryu manager. In the run time, the Ryu manager creates a listener that can connect to the OpenFlow switches. Once it is run, it listens to the specified IP address and the specified port (6633 by default). Then, any OpenFlow switch can connect to the Ryu manager. The App-manager is one of the main components for all Ryu applications since they need to inherit functionality from the App-manager's RyuApp class. The core-process component in the architecture includes messaging, event management, in-memory state management, etc. In the architecture, the northbound Application Programming Interface (API) is illustrated in the uppermost layer, where supported plug-ins can communicate with Ryu's OF operations.

6.2.4 RYU Northbound API

At the API layer, Ryu generously supports a REST interface to its OpenFlow operations. Ryu also includes an Openstack Neutron plug-in that supports both typical VLAN and GRE-based overlay configurations. In a worthy addition, the researcher can easily create REST APIs by using a framework for connecting web servers and applications in Python called WSGI.

6.2.5 RYU Applications

Ryu application is one of the essential elements since the control logic and behavior is defined in it. Multiple applications are already included in the Ryu framework such as topology, simple_switch, firewall, router, etc. Although Ryu applications are implemented and provided various functionalities, they work as the single-threaded entities. Formerly, Ryu applications send asynchronous events to each other.

Each Ryu application ordinarily has its own receive queue for possible events, that is especially FIFO to properly preserve the executive order of events. Furthermore, each application typically includes a thread for properly processing events from the queue. The thread's main loop pops out events from the receive queue and calls the suitable event handler. Therefore, the event handler is naturally called within the context of the event-processing thread, that works in a blocking fashion, i.e., once an event

handler is given management, no additional events for the Ryu application are going to be processed till management is returned. The functional architecture of a Ryu application is introduced in Figure 6.3.

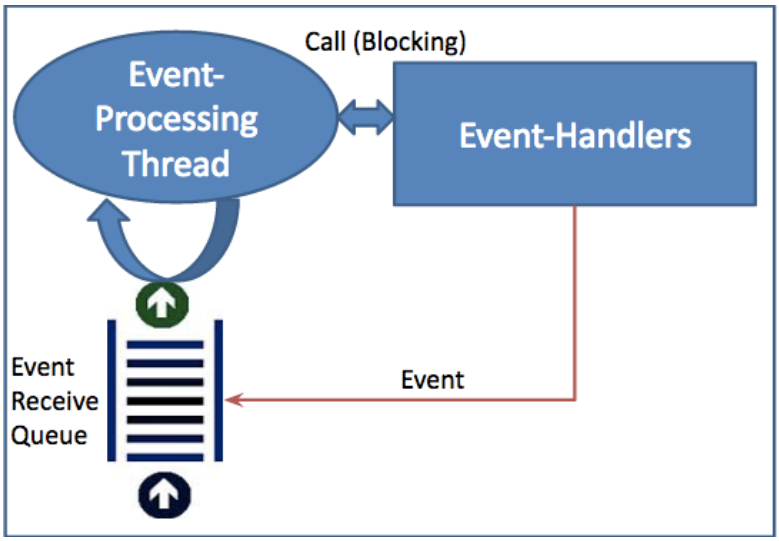


Figure 6.3 Functional Architecture of Ryu Application

6.3 Mininet Network Emulator

To emulate an entire network infrastructure in the SDN environment, Mininet [87] network emulator is used. It is the main tool for Software-Defined Networking testbed environments to design, undergo and verify OpenFlow projects. Mininet provides a high level of flexibility since topologies and new functionalities are programmed using python language. It also provides a scalable prototyping environment, able to manage up to 4000 switches on a regular computer. This is possible due to an OS-level virtualization feature, as well as including processes and network namespaces, that permits to produce completely different and separate instances of network interfaces and routing tables that typically operate the independents of every different.

Mininet mainly allows the hosts, switches, and controllers and it creates a huge network simulation in a single PC. It creates the virtual open flow network which includes SDN controller, OpenFlow software switches, multiple hosts and links in a real or virtual machine. The Mininet tool will work in the different operating systems such as Mac OS, Windows, and Linux. Mostly for the Research part, Linux is preferable.

Using Mininet can create the small data center consists of hosts and open flow switches. By implementing this experiment, the output can be achieved.

6.3.1 Topology Elements

There are four topology elements that Mininet can create:

- **Link:** emulates a wired connection between two virtual interfaces which act as a fully functional Ethernet port. Packets are delivered through one interface to another. It is possible to configure Traffic Control for the links importing the TCLink library via the Python API.
- **Host:** emulates a Linux computer which is simply a shell process moved into its own namespace, from where commands can be called. Each host has its own virtual Ethernet interface.
- **Switch:** software OpenFlow switches typically provide the identical packet delivery semantics that will be provided by a hardware switch.
- **Controller:** Mininet allows to create controller within the same emulation or to connect the emulated network to an external controller running anywhere there is IP connectivity with the machine where Mininet is running.

6.3.2 Command Used to Create Topology in Mininet

To execute Mininet passing a file containing a particular topology, the command **mn** has to be accompanied to the parameter **--topo mytopo** and the parameter **--custom mytopo**, and the name of the file is **simple_topo.py**.

The command used to launch a custom network topology is this:

```
root@ntz: sudo mn --custom ~/mininet/custom/ simple_topo.py --topo
mytopo --controller=remote, ip=127.0.0.1, port=6633
```

The option **--mac** is used to set automatically the host MACs addresses. With **--custom** we indicate the path in which is present the file from which you will take the topology, in this case **mytopo.py**, followed by the command **--topo mytopo** which is the specified name given to the **topo** variable inside the file **simple_topo.py**.

6.4 Traffic Generator and Measurement Tools

For this academic research, accuracy in the measurement is not a necessary objective. However, it should be precise enough to sufficiently reveal general behavior and distinct tendencies of the network traffic. Subsequently, the following tools were properly selected for the objective measurement of the achieved bandwidth and analysis of traffic under different QoS settings: iPerf [82], Wireshark [98], and DITG [7].

6.4.1 Iperf

Iperf is a very useful performance measurement tool for measuring the maximum bandwidth available between two nodes. It is utilized in TCP (Transport Control Protocol) and UDP (User Datagram Protocol) connections, through the modulation of various parameters. Mainly Iperf is used for the bandwidth and datagram loss. It mainly uses to calculate the network flow between the two nodes. It must be installed on both nodes, then it must be started as a server on one node, and as a client on the other one. The transmission procedure will take only a few seconds and then you will see the bandwidth.

The following work can be done by using Iperf:

- measure bandwidth
- in a client-server network, the client generates a UDP flux, with
- a specific bandwidth (BW)
- measure packet loss
- measure jitter
- work in a multicast environment.

6.4.2 Wireshark

Wireshark in common is a common network packet analyzer which can be popularly used on the controller or Mininet host to properly look at OpenFlow exchange message between the controller and individual switches. It adequately captures a packet within the network to instantly show its TCP/IP layer information as detailed as possible, letting to look at its explicit content for purposes like network troubleshooting, security examinations, protocol debugging and network protocol learning.

This tool permits to properly capture live packets from a network interface (physical or virtual), displaying correctly the packet information with elaborate protocol information, and saving all packet captures for additional studies and reliable statistics. The captured information can be analyzed under different criteria, in which the necessary information can be altered by timestamps, TCP/UDP ports, protocols, TCP sessions, and more.

Since Wireshark remain an effective tool that works at user-space, it uses the pcap library to permit Wireshark to capture packets at a lower level. This typically allows the capture of packets with a more precise timestamp since there is no additional delay caused by the internal communication process between the user-space and kernel-space levels. This research has used the Wireshark dissector provided by a Mininet package, that enables the OpenFlow filter to typically capture OpenFlow messages and carefully observe their message format in detail, as well as flow entries and group entries.

6.4.3 Distributed Internet Traffic Generator (DITG)

One important component of the experiment framework is the traffic generator which is used to generate network traffic flows in the emulated network. In the network experiments, single flow traces collected in real networks should be replayed as they occurred in the real network which means that the packet timing and packet sizes should replicate the real scenario as exactly as possible.

There are several traffic generators that support such a trace-based traffic generation like TCPreplay [100] or TCPopera [33]. However, because of the possibility to schedule multiple flows and the more comprehensive and convenient way of altering properties of single flows, the distributed internet traffic generator (D-ITG) was selected to be integrated into the experiment framework. Furthermore, D-ITG offers more advanced features for calculating and logging network metrics and provides also an analytical model-based traffic generation mode. In this mode, D-ITG is capable of producing realistic network workloads that replicate stochastic processes [7]. During the experiments, this functionality was not utilized but might be useful in some other scenarios which are the reason why the experiment framework becomes more flexible by integrating the D-ITG traffic generator rather than a trace-based only generator.

6.5 The Test-bed Implementation with Mininet

The SDN controller, Ryu framework is designed to run on the host computer with a TCP connection to the emulated Mininet network topology. The detailed software versions are shown in Table 6.5.

Table 6.5 Testbed Requirements

No	Name	Specification
1	Operating System	Ubuntu 16.04 LTS (64 bits)
2	Ryu controller [Ryu]	Version 4.30
3	Mininet Emulator [Mininet]	Version 2.2.1
4	OpenFlow Protocol [OpenFlow]	Version 1.3

The test-bed implementation is structured as illustrated in Figure 6.3

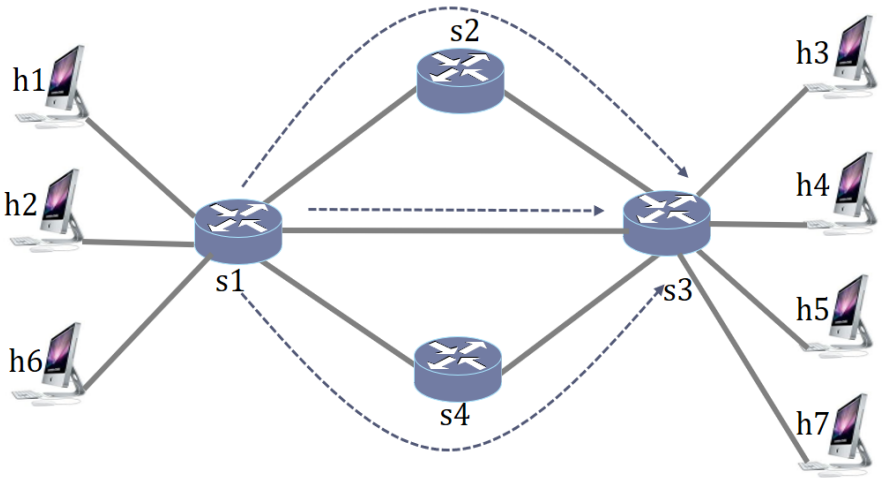


Figure 6.4 Simple Network Topology

Firstly, run a simple network topology as shown in Figure 6.4, composed by three clients, h1, h2, and h6 and four servers h3, h4, h5, h7. There are three paths to communicate from the clients to the servers. Hence, path (s1, s3) is the direct communication link and it is the minimum hop count path (shortest path) leaving the other two equal-cost paths (s1, s2, s3) and (s1, s4, s3) as the second shortest. The Mininet script file is shown in the script file below.

simple_topo.py

```
# !/usr/bin/env python
from mininet.net import Mininet, CLI
from mininet.node import RemoteController, OVSKernelSwitch, UserSwitch, Host
from mininet.link import TCLink,Link
from mininet.term import makeTerms, makeTerm, runX11
import argparse
import subprocess
net = Mininet(controller=RemoteController, switch=OVSKernelSwitch, link=TCLink)
h1 = net.addHost ( 'h1', mac = '00:00:00:00:00:01', ip = '10.0.0.10' )
h2 = net.addHost ( 'h2', mac = '00:00:00:00:00:02', ip = '10.0.0.20' )
h3 = net.addHost ( 'h3', mac = '00:00:00:00:00:03', ip = '10.0.0.30' )
h4 = net.addHost ( 'h4', mac = '00:00:00:00:00:04', ip = '10.0.0.40' )
h5 = net.addHost ( 'h5', mac = '00:00:00:00:00:05', ip = '10.0.0.50' )
h6 = net.addHost ( 'h6', mac = '00:00:00:00:00:06', ip = '10.0.0.60' )
h7 = net.addHost ( 'h7', mac = '00:00:00:00:00:07', ip = '10.0.0.70' )
s1 = net.addSwitch ( 's1', cls = OVSKernelSwitch, protocols = 'OpenFlow13' )
s2 = net.addSwitch ( 's2', cls = OVSKernelSwitch, protocols = 'OpenFlow13' )
s3 = net.addSwitch ( 's3', cls = OVSKernelSwitch, protocols = 'OpenFlow13' )
s4 = net.addSwitch ( 's4', cls = OVSKernelSwitch, protocols = 'OpenFlow13' )
net.addLink( s1, s2, port1=1, port2=1)
net.addLink( s2, s3, port1=2, port2=2)
net.addLink( s1, s4, port1=2, port2=1)
net.addLink( s3, s4, port1=3, port2=2)
net.addLink( s1, s3, port1=3, port2=1)
net.addLink( s3, h6)
net.addLink( s3, h4)
net.addLink( s3, h5)
net.addLink( s3, h7)
net.addLink( h1, s1)
net.addLink( h2, s1)
net.addLink( h3, s1)
net.addController('c0')
net.start()
CLI(net)
net.stop()
```

The topology in mininet is created with this command:

root@ntz: sudo python simple_topo.py

After this command is launched, the CLI devolves the output shown in Figure 6.5.

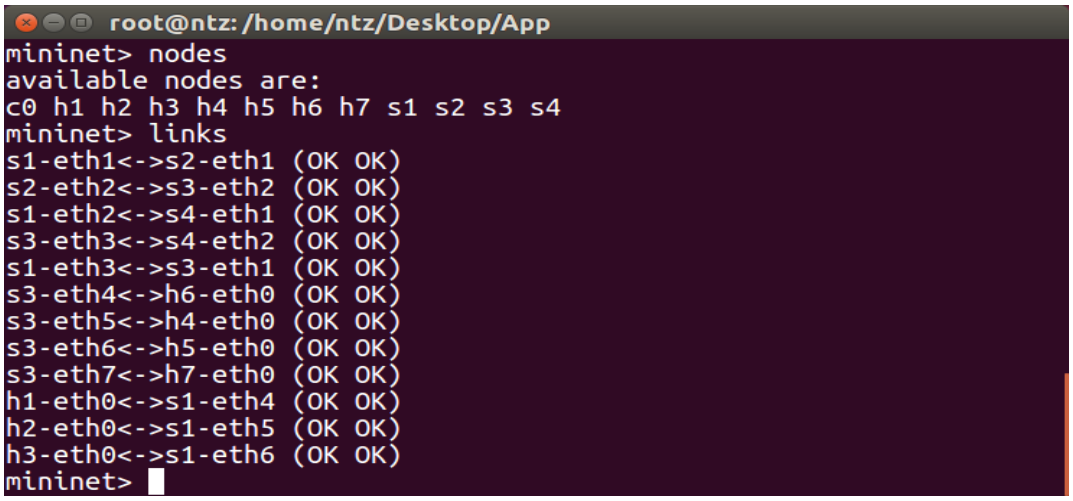


Figure 6.5 Network Topology Created

The Ping command is used to verify connectivity among devices. In Mininet it is possible to use the pingall command, that does an all-pairs ping. This command verifies that the created links function. The figure below shows that when running the first time the pingall command, the first host does not have connectivity with other hosts, while all the other links are good.

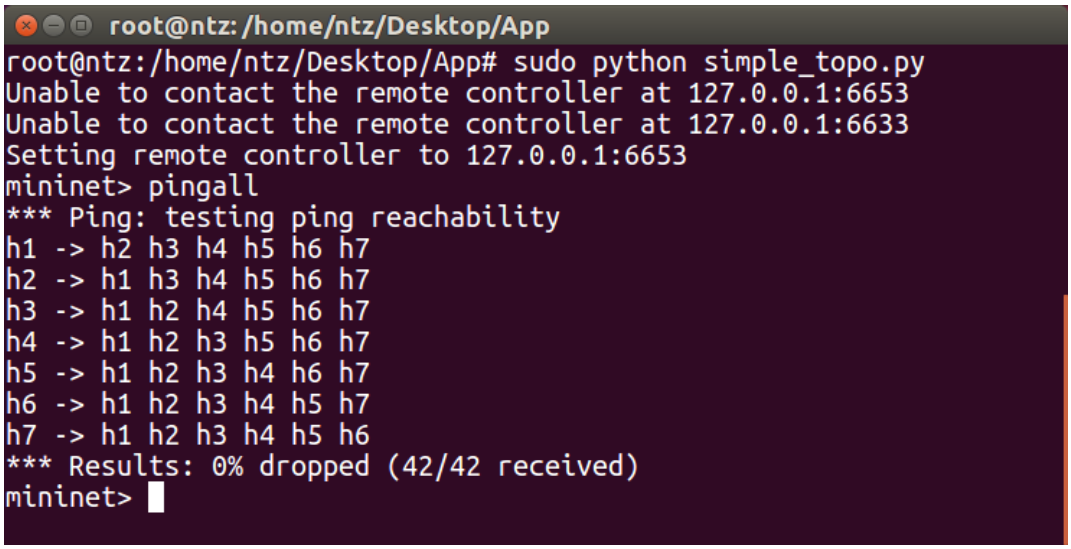


Figure 6.6 Pingall Command Executed

6.6 QoS Measurement Parameters

The proposed QT approach is compared with three methods: (i) conventional shortest path routing, (ii) multipath routing and (iii) Hedera in fat tree topology in terms of performance. The performance of the proposed QT approach is measured by QoS parameters such as:

1. **Throughput:** It is the rate of successful message delivery over a network. Throughput is measured in Bps (bytes per second) or bps (bits per second). If more data transferred, higher throughput result.

2. **Delay:** It's the amount of time it takes to send information from one point to the next. Delay is usually measured in milliseconds or ms. The ITU-T recommends that in general network planning, a maximum of 400 ms for the one-way delay should not be exceeded. However, they note that many interactive applications (e.g, voice calls, video conferencing, interactive data applications) are affected by a much lower delay. The experiences of most applications are generally considered acceptable if the delay is kept below 150 ms. As the traffic latency increases, the impact on applications' experiences becomes noticeable. When the delay exceeds 400 ms, most applications will encounter unsatisfactory performance. Several factors affect the end-to-end delay of transmitted data packets. They include processing delay, queueing delay, transmission delay, and propagation delay. It impacts the user experience and can change based on several factors. For simplicity, only transmission delay will consider in the experiment and assume the other type of delays is negligible.

3. **Jitter:** It is based on the delay - specifically, delay variations. Jitter is the difference between the delay of two packets. It often results in packet loss and network congestion.

4. **Packet Loss:** It occurs when one or more packets of data traveling across the network fail to reach their destination. One of the major causes of packet loss is link congestion. It is either caused by errors in data transmission.

Results are expected to be better for the QT approach in terms of throughput and packet loss because in this approach the end-to-end bandwidth resource allocation is presented to provide better QoS performance for different types of traffic. Moreover,

the QT approach can be reduced the amount of delay for the minimum delay demand flow due to its delay estimation module which is used in the QoS routing module.

6.7 Chapter Summary

In this approach, it is assumed that network users register their preferences of QoS demands to the controller in order to allocate an appropriate route for each traffic. The controller maintains the registered information and finds the most feasible path for each QoS demand. The minimum delay demand flow has the highest priority flow; the bandwidth demand flow has the medium priority flow and leaving the default flow as the lowest priority flow. For the bandwidth demand flow, the controller finds the maximum bandwidth path to improve the performance of the flow throughput. For the delay demand flow, the controller chooses the minimum delay paths.

This chapter presents the QoS classes and the related policies that will apply in the proposed system. By using these technologies and features, the proposed system will demonstrate and test the performance of the proposed approach in the next chapter.

CHAPTER 7

EXPERIMENTAL RESULTS AND ANALYSIS

This chapter investigates the performance of the proposed approach (QT) and tests the validity of the proposed resource allocation scheme. This chapter shows and explains the results obtained with the scenario proposed in the previous chapter. The analysis consists of end-to-end measurements of throughput, latency, jitter, and losses for UDP and TCP connections with different traffic patterns. The first point with analysis of the default behavior of the OpenFlow's supportive QoS features are also included. All the measurements have been done by generating the network traffics with the use of DITG. In order to evaluate the performance of the proposed approach (QT), the experimental testbed has to be designed. As the proposed system is implemented using the Ryu controller, it runs different topologies in order to compare and evaluate the results with or without the proposed method. In order to have deterministic and low-cost environments to test, a virtual testbed was created on a PC with Intel Core i7-6500U processor, 8 GB of RAM, and 1TB of hard disk space running Ubuntu 16.04 LTS operating system. The Python implementation of the experiment used Python version 2.7.

7.1 Preliminary Experiments with User-space Switches (CPqD)

This section serves as an initial exploration of the bandwidth guaranteeing system with the OpenFlow protocol. Currently, OpenFlow is supported for QoS in the SDN environment by two options, specifically the queue and meter. A queue is an egress packet queuing mechanism in the OpenFlow switch port. Although OpenFlow supports the queue features, it does not handle queue management; it is just able to query queue statistics from the switch. Therefore, the queuing feature of OpenFlow is a property of a switch port. Meters have been introduced in OpenFlow protocol in version 1.3 to measure and control the ingress rate of packets in switches. OpenFlow metering enables the ingress rate monitoring of flow and performs operations based on the rate of the flow. Unlike a queue, a meter is attached to flow entries. A meter has a component called meter band which specifies the rate at which the meter is applied. A meter can have one or more meter bands but only a single band is applied for each flow at a time based on

the rate of packets. A flow which is mapped to a meter directs packets to the meter which measures the rate of the packets and activates appropriate meter band if the measured rate of packet goes beyond the rate defined in the meter band. This experiment focuses on the studying of providing bandwidth guarantee with OpenFlow supportive QoS features, queue and meter.

7.1.1 Experimental Setup

All network topologies are emulated on an Ubuntu 16.4LTS virtual machine. The emulation is conducted using a network topology including one controller, four switches, and four hosts as shown in Figure 7.1. Mininet is used for the network emulation in all the experiments. Mininet is a network emulator that can run multiple end-hosts on a single Linux kernel. Various network topologies with different switches, routers, and links can be created. Once a topology is set up, each element of it runs on the same kernel. Links can be set up at arbitrary bandwidth, network delay and packet loss. Furthermore, each host in Mininet behaves just like a real machine. “Ryu” controller is used to control the topology which is supported by the emulator. All the switches are User-space switches (CPqD) for the purpose of testing since OpenvSwitch (v2.8) does not fully support meter features with DSCP_remark. Only the CPqD switch can support the ‘dscp_remark’ meter band setup.

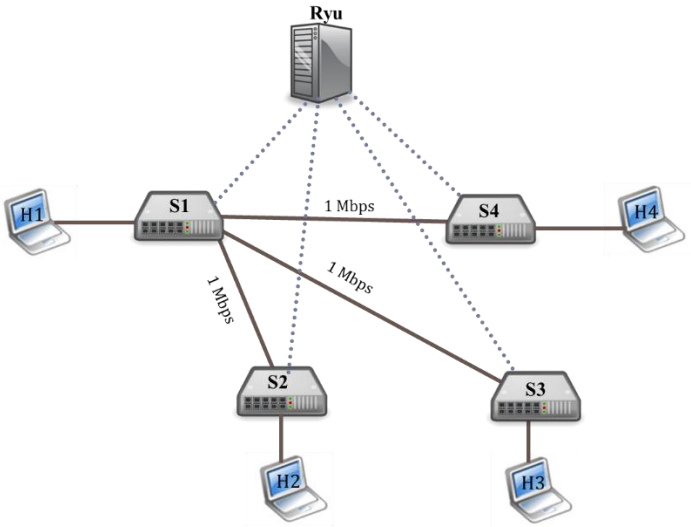


Figure 7.1 Test-bed Environment

A network topology has been developed via a script in Mininet. After running the topology script, the controller takes action to load topology information using Ryu's library and set QoS configuration to a link between switches. Then, we apply the configuration of Iperf client and server to each virtual host to verify the throughput of the network. For all the tests, the proposed system used the same topology in order to have the basis for comparison.

7.1.2 Evaluation Results

This section investigates the use of OpenFlow's meter function in QoS control with different scenarios and measures the performance of the network. Iperf utility is used to generate traffic in all of our experiments. Iperf is a generally-used network testing tool that can create Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) data streams and measure the throughput of a network that is carrying them. Iperf allows the user to set various parameters to test the performance of the network. For example, the user can measure the throughput between the two ends since Iperf has a client and server functionality. Instead of TCP traffic flow, UDP traffic flows are adopted, which provide the most efficient means of congesting the bottleneck link.

7.1.2.1 Scenario 1: Without QoS Setting

In the first scenario, testing starts all the flows at the same time in the best effort fashion with bandwidth limitation. Traffic flow is set as shown in Table 7.1.

Table 7.1 Network Experiment Flows for Scenario 1

Flow ID	Flow Type	Source- Destination	Destination Port	Protocol	Traffic (kbps)
Flow 1	BE	H2 - H1	5001	UDP	800
Flow 2	BE	H3 - H1	5002	UDP	400
Flow 3	BE	H4 - H1	5003	UDP	600

A flow may congest the network with other flows if all traffic is handled in a best-effort fashion and it is possible to see that all traffic competes for the total bandwidth. Flow congestion will increase whenever a new flow arrives. According to the common best-effort manner, packets are simply dropped if congestion happens.

Figure 7.2 shows that there is no bandwidth guarantee for all the flows without QoS implementation.

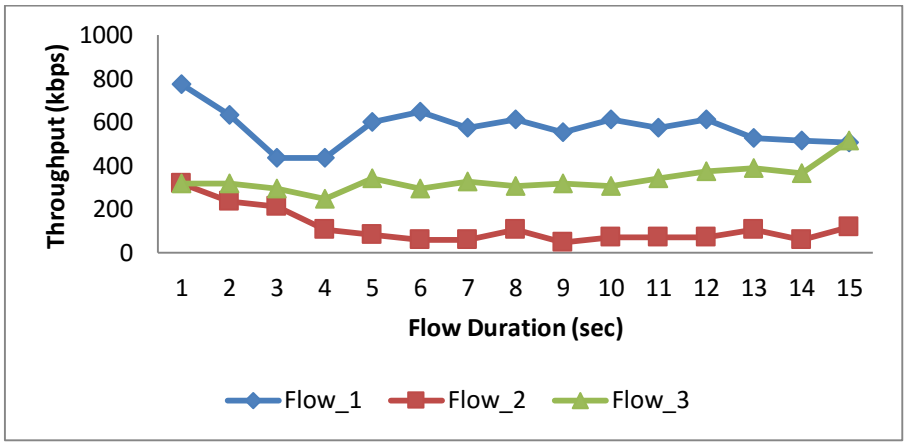


Figure 7.2 Flow Bandwidth Distribution Between All Data Flows

7.1.2.2 Scenario 2: With QoS Setting

The second scenario is to run the same topology as scenario 1 by adding queue and meter functions to reserve network bandwidth for QoS flows. In order to abstract simulations, the queue was created with predefined bandwidth allocation in S1 and configure the static meter setting in other switches. In S1, every port has three different level queues and different QoS parameters (minimum bandwidth) are configured for those queues. All the incoming packets to S1 are assigned to one of these queues before forwarding to the destination. Table 7.2 shows the bandwidth allocation with the queue setting.

Table 7.2 Queue Configurations for Experiment

Queue ID	Minimum Rates (kbps)	QoS ID	DSCP value
0	80	1	0
1	120	2	10
2	800	3	12

This scenario classifies the traffic flows into two types: QoS-flow and best-effort flow. QoS is implemented using DiffServ and uses different differentiated services code point (DSCP) numbers to classify network traffic for quality of service (QoS) levels. DSCP = 0 is used for the best-effort flow in this experiment. DSCP = 10 (AF11) and DSCP = 12 (AF12) are used for QoS-flow 1 and QoS-flow 2 respectively. Each DSCP

value is matched with a meter instruction in the meter table and looks up the corresponding queue on the switch's flow table. Then, packets are sent out to the neighbor switches from their corresponding output ports and queue based on the DSCP number in the packet header. Bandwidth guaranteed for AF11 class traffic is set (QoS-flow 1) with 400 kbps. Table 7.3 shows the meter band, and Table 7.4 shows flow entry information with meter and queue.

Table 7.3 Meter Band Setting

Meter ID	Flags	Bands
1	kpbs	type:dscp_remark, rate:400 kbps

Table 7.4 Flow Entry Information with Meter

QoS ID	Meter ID	Source- Destination	Destination Port	Protocol	Traffic (kbps)
1	-	H2 - H1	5001	UDP	800
2	1	H3 - H1	5002	UDP	400
3	-	H4 - H1	5003	UDP	600

A comparison of throughput fluctuations is made between the best-effort data flow and QoS-flow. Figure 7.3 shows the time-varying throughput for the best-effort flow and QoS-flow 1. If AF11 class traffic exceeds 400 kbps, re-marked the traffic as AF12 class and treated as excess traffic. Figure 7.4 shows the flow bandwidth distribution between the best-effort flow and QoS-flow 2 (AF12), and according to these results, AF12 is more preferentially guaranteed bandwidth than the traffic of the best effort.

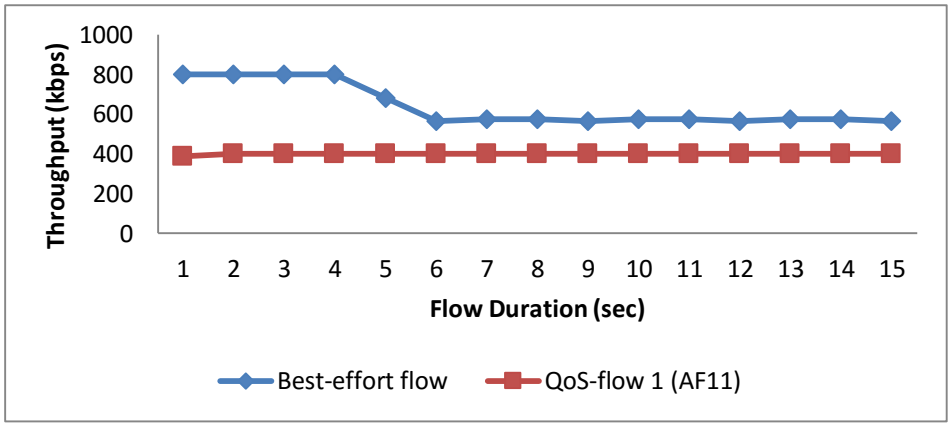


Figure 7.3 Flow Bandwidth Distribution Between Best-effort Flow and QoS-flow 1 (AF11)

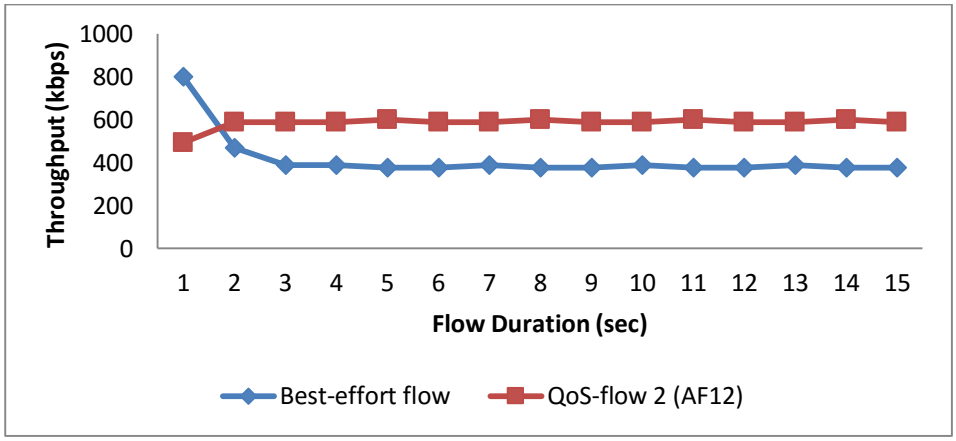


Figure 7.4 Flow Bandwidth Distribution Between Best-effort Flow and QoS-flow 2 (AF12)

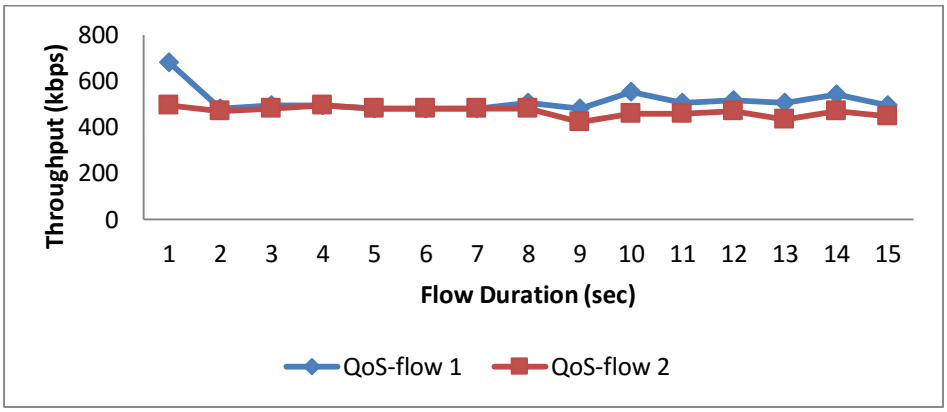


Figure 7.5 Flow Bandwidth Distribution Between QoS-flow 2 (AF12) and QoS-flow 2 (AF12)

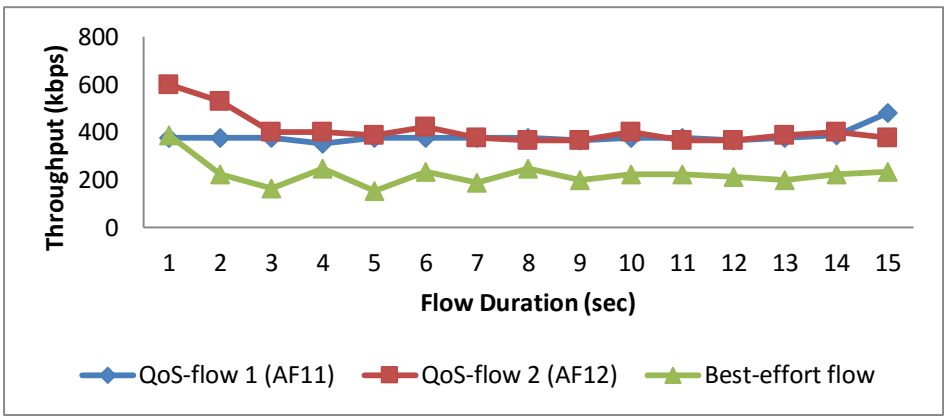


Figure 7.6 Flow Bandwidth Distribution Between QoS-flows and Best-effort Flow

Figure 7.5 shows that when the two QoS-flow 2 (AF12) is faced with network congestion both of them drop at the same rate. According to the results in Figure 7.6, meter bands are limiting bandwidth per-flow and the queue provides bandwidth guarantees for each specific application as expected. For this round, best-effort network traffic is generated with 400 kbps instead of using 800 kbps like all the above experiments. For the sake of demonstration, the network congestion rate has been reduced slightly because the test-bed's link capacity is set to 1 Mbps.

According to the QoS configuration setting, best-effort flows are passed through q1 and QoS-flow 1 with DSCP=10 will be passed through to q3. Lastly, QoS-flow 2 (remark packets/flows) is passed through to q2. Figure 7.7 describes the statistical information of all the queues related to port 1 in S1, and Figure 7.8 notifies the statistics of meter used in the experiment.

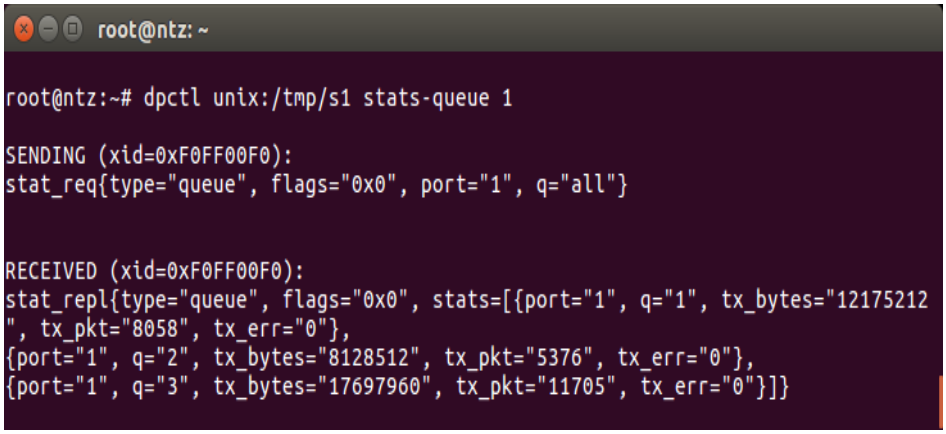


Figure 7.7 Statistical Information of Queues in S1

Figure 7.8 Statistic of Meter in S3

A well-designed QoS system should give access to the right amount of network resources needed by the various data flows using the network. In this experiment, implementation and verification of QoS control in SDN with the help of OpenFlow's QoS functionality were presented. Also, the researcher has demonstrated how to provide bandwidth guarantees with OpenFlow's meter function by carrying out experiments. The results of the experiments confirmed that the meter function of OpenFlow can provide bandwidth guarantee effectively in high QoS network and we can adapt the DSCP values for traffic classification to make QoS control easier.

However, the QT program is configured with OVS. Since OVS does not support the full capability of the OpenFlow meter function, DSCP remark, it works just like a traffic shaper at the ingress port. Therefore, the use of the OpenFlow meter function is left as future research. In the later experiment, the queue mechanism will just be used to provide a bandwidth guarantee.

7.2 Preliminary Experiments with Open Vswitch (OVS)

Before the evaluation of the proposed approach (QT), this section exploits a bandwidth guaranteeing system with OpenFlow protocol in the SDN network with OpenVswitch. Since the bandwidth is the key component for offering QoS, the focus is only on bandwidth guarantees in this experiment. Since the OVS cannot fully support the metering feature, it will only explore the egress queues defined in the OF1.3 specifications. This experiment implements and verifies QoS control with OpenFlow's queuing techniques HTB over SDN. It describes the results of the experiments in the SDN emulation network environment.

In this study, HTB queuing is used to provide the bandwidth guarantee for the QoS flows. The HTB qdisc allows arranging traffic classes in a multi-layered hierarchical tree. In the proposed approach, two layers hierarchical tree is used where the root node (in the first layer) represents the parent class for all kinds of traffic. The root node is configured as soon as a switch connects to the controller. The maximum rate and minimum rate for the root class are both equal to the link speed. Typically, the maximum rate for the class is equal to the root (link speed) unless obviously stated otherwise in the request.

7.2.1 Experimental Design

The network topology is implemented by using python script in the Mininet emulator. Figure 7.9 shows the simulated network topology for the experiment. The implementation of the queue mechanism is demonstrated with simple linear network topology which consists of two switches and four hosts. The Open Virtual Switches (OVS) are used in the network topology. All the network links between OVS switches have been set to 100 Mbps, and the capacity of the output ports in all the OVS switches are also100 Mbps. Hosts h1,h2, and h3 are traffic senders via OVS switch S1 and h4 is the traffic receivers via OVS switch S2. OVS switches S2 and S4 are the intermediate switches in the prototype.

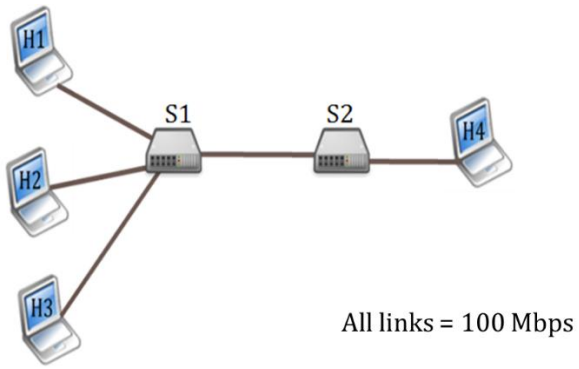


Figure 7.9 Linear Network Topology

In the first scenario, testing starts all the flows at the same time in the best effort fashion with bandwidth limitation. Traffic flow is set as shown in Table 7.5. In order to abstract simulations, the queue are created with predefined bandwidth allocation in S1. In S1, every port has three different level queues and different QoS parameters (minimum bandwidth) are configured for those queues. All the incoming packets to S1 are assigned to one of these queues before forwarding to the destination. Table 7.6 shows the bandwidth allocation with queue setting.

Table 7.5 Network Experiment Flows Information

Flow ID	Source- Destination	Destination Port	Protocol	Traffic (Mbps)
Flow 1	H2 - H1	1111	UDP	100
Flow 2	H3 - H1	2222	UDP	100
Flow 3	H4 - H1	3333	UDP	100

Table 7.6 Queue Configurations Setting

Queue ID	Max-Min Rates (Mbps)	QoS ID	DSCP value
0	10 (Max)	1	0
1	30 (Min)	2	10
2	60 (Max)	3	12

7.2.2 Evaluation Results

Figure 7.10 shows the created network topology information in mininet.

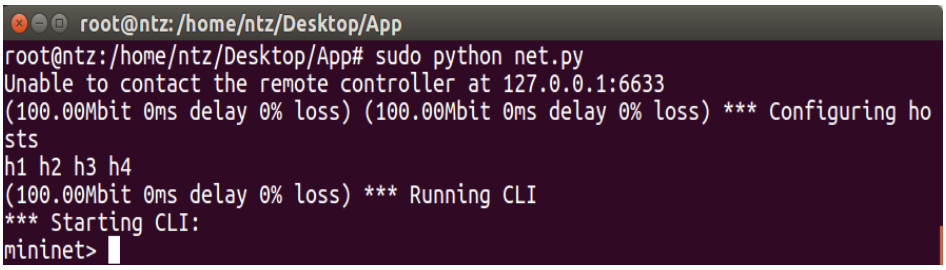


Figure 7.10 Created Network Topology

Figure 7.11 shows the throughput results according to the predefined bandwidth allocation for all flows.

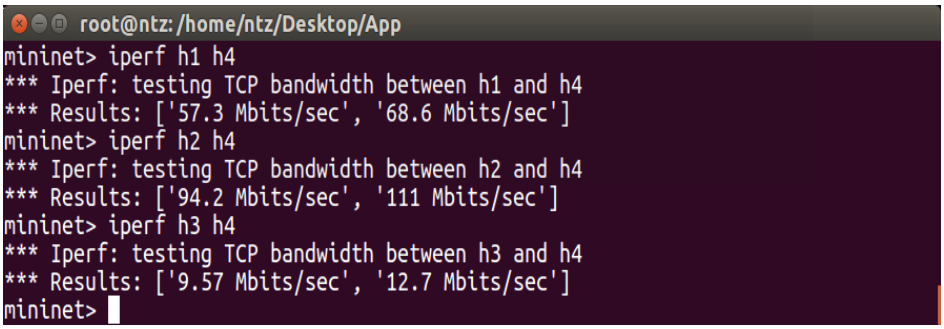


Figure 7.11 Throughput Testing with Iperf Utility

In the beginning, the H1 is sending to H4. The throughput is around 58 Mbps since the flow H1-H4 enqueued to q2 which has 60 Mbps as the maximum queue size. After 10 seconds later, we generate H2 flow for 10 seconds. When H2 starts to send traffic to H4, the throughput of flow H1-H4 drops immediately to 600 Kbps as shown in Figure 7.12. This is because the flow H2-H4 enqueued to q1 which gave 20 Mbps as the minimum bandwidth upmost to the link capacity as the maximum bandwidth. Therefore, the throughput of the H2-H4 is around 96.6 Mbps. After 10 seconds later, H2

flow is finished and then start H3 to H4. The throughput of H1-H4 will not be affected by H3-H4.

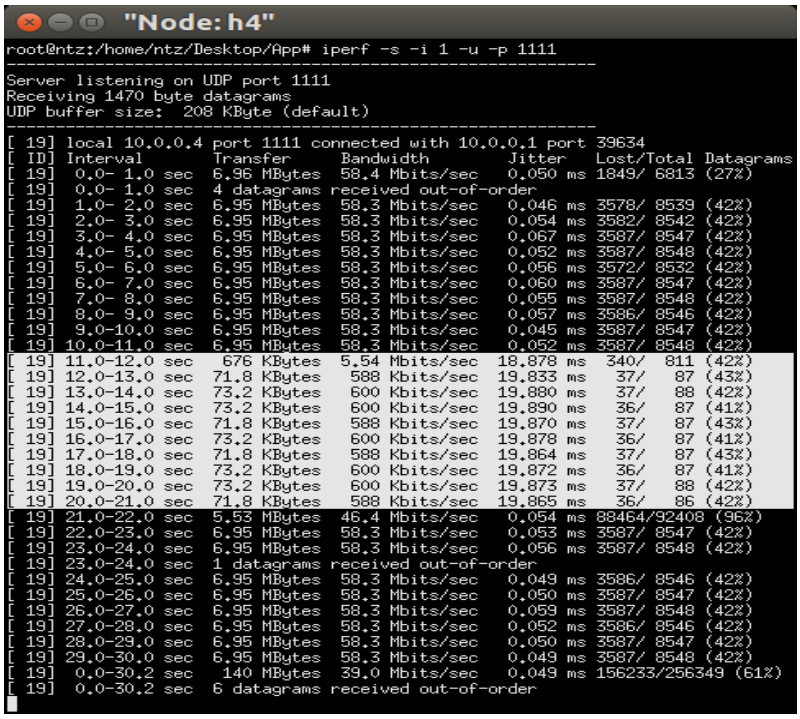


Figure 7.12 Iperf Throughput Results Shown in the Server Site H4

From these two experiments, the OpenFlow-Queueing mechanism improves QoS by providing a bandwidth guarantee for the high priority traffic is confirmed. But Specification of OpenFlow is not finished to fully support QoS implementation in the current state. For example, OpenFlow Queues are not a mandatory part of the specification and managing OpenFlow Queues is not handled by OpenFlow Protocol. As a future work, investigating and evaluating further QoS mechanisms in SDN by using queue statistic and OpenFlow meter is still is needed. Moreover, the numbers and priority of the Queues are vendors dependent on the current network. Therefore, for the proposed approach (QT), (1:3:6) ratio of the link bandwidth will be set as a reasonable amount of bandwidth for the testing purposed in the below experiments.

7.3 Experimental Design for the Proposed Approach (QT)

This section exploits the prototype implementation and demonstration of the proposed resource allocation scheme in the different network topology. In the test environment, the SDN controller obtains the global network information and status by

periodically monitoring the network condition. The controller estimates the link bandwidth utilization and identifies whether the links are bottlenecks or not. If a bottleneck link is detected, the controller chooses one or more highest priority flows to reroute to the alternative path. The data traffic is divided into three QoS classes. One is the traffic that does not have QoS requirements (called best-effort flows) and the other two are the traffic that has one or more QoS requirements such as bandwidth, delay, jitter or packet loss ratio (called critical flows or QoS flows). An emulated OpenFlow environment is configured and used to validate the proposed solution by using a Mininet emulator. The Ryu controller is used as an SDN remote controller in the control plane of the OVS for the proposed system.

In order to allocate an appropriate route for each traffic, it is assumed that network users register their preferences of QoS demands to the controller. The possible options for preferences QoS demands are bandwidth, delay, and default (best-effort). The controller maintains the registered information and finds the most feasible path for each QoS demand. In the experiment, the delay demand flow has the highest priority flow; the bandwidth demand flow has the medium priority flow and leaving the default flow as the lowest priority flow. For the bandwidth demand flow, the controller finds the maximum bandwidth path to improve the performance of the flow throughput. In order to minimize the flow delay, the controller estimate and choose the minimum delay paths for the minimum delay demand flow if the link bandwidth is enough for it.

To limit the maximum, minimum traffic rates for different flows, three different queues are set with different rates for all the interfaces of the switches. Queue configuration setting with specific flow type which is used throughout the whole experiments is shown in Table 7.7.

Table 7.7 Queue Configurations for Experiment

QoS Class	QoS priority	Queue	Max-rate	Min-rate
Delay	High	q1	-	3Mbps
Bandwidth	Medium	q2	6 Mbps	-
BE	Low	q0	1 Mbps	-

Queue configuration setting and queue mapping with specific flow types are shown in Table 7.8. The example command to create the queues is shown below.

```
ovs-vsctl -- set port s2-eth2 qos=@newqos -- --id=@newqos create qos type=linux-htb
other-config:max-rate=100000000 queues=0=@q0,1=@q1,2=@q2 -- \
 --id=@q0 create queue other-config:max-rate=10000000 --
--id=@q1 create queue other-config:min-rate=30000000 --
--id=@q2 create queue other-config:max-rate=60000000
```

For all the experiments, that the bandwidth proportion of the three priority queues is assumed 1:3:6, for simplicity, follow the authors used in [61]. If the flow request exceeds the available bandwidth which is defined by the queue scheduling module to serve for this specific type of flow, the service will not be guaranteed. The flow priority for the bandwidth demand flow is set as a higher priority than minimum delay demand flow. Since the queue priority of the minimum delay demand flow is the highest, it will affect all other queues and flows. Therefore, the routing module will set the highest flow priority to bandwidth demand flow in order to prevent service degradation too much because of the priority queue. Cross-traffic flow is a simple best-effort flow and it has the lowest priority. The cross-traffic is generated to change the congestion level for the demonstration purpose. The routing module will calculate the new path to reroute the high priority flow when the network is highly congested.

7.3.1 Experiment 1: Simple Network Topology

To test the fundamental work of delay estimation module and QoS routing approach, a simple network topology is used. An emulated OpenFlow environment is configured and used to validate the proposed solution by using a Mininet emulator. The Ryu controller is used as an SDN remote controller in the control plane of the OVS for the proposed system.

7.3.1.1 Experimental Setup

To check the effectiveness of the proposed approach (QT), an environment is built as shown in Figure 7.13. The emulation was conducted by using the network topology including one controller, four switches, and seven hosts as shown in Figure 7.13. Bandwidths of all the links were limited to 100Mbps for testing purposes. In

accordance with the topology in Figure 7.13, we have three clients, h1, h2, and h3 with four servers h4, h5, h6, and h7.

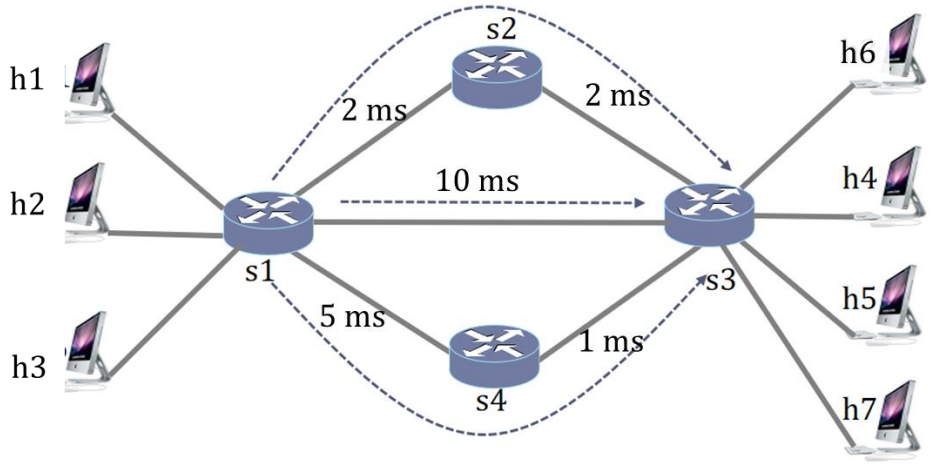


Figure 7.13 Simple Network Topology with Link Delay Parameter

In order to simulate the UDP traffic, the DITG network test tool is customized and seven flows are generated. To observe the flow performance for the proposed scheme, two minimum delay demand flows, two bandwidth demand flows and three best_effort traffic flows are used with various packet sizes and rates. The minimum bandwidth request ratios for both QoS flows and cross-traffic are shown in Table 7.8.

Table 7.8 Network Experiment Flows for Simple Network Topology

Flow id	Source- Destination	Packet rate (pps)	Packet size (Bps)	Time (sec.)	Type
1	h1-h4	VoIP	-	60	UDP
2	h1-h6	3500 (Video)	1024	60	UDP
3	h2-h5	1000 (Haptic)	128	60	UDP
4	h2-h7	3000 (BE)	1024	60	UDP
5	h3-h6	3000 (Video)	1024	60	UDP
6	h3-h7	2500 (BE)	1024	60	UDP
7	h3-h8	Telnet	-	60	TCP

The host pair (h1-h4) means the traffic from h1 to h4 and it is denoted as minimum delay demand flow (VoIP). The host pair (h1-h6) means the traffic from h1 to h6 and it is denoted as bandwidth demand flow. Then the host pair (h2-h5) means the traffic from h2 to h5 and it denoted as haptic traffic.

To limit the maximum and minimum traffic rates for different flows, we set three different queues with different rates for all the egress interfaces of s1, s2, and s4. For this experiment, q0 has with the maximum bandwidth (1 Mbps) for the default demand flow and q1 has the minimum bandwidth (3 Mbps) for the minimum delay demand flow. Then, q2 is configured with a large maximum bandwidth (6 Mbps) for the bandwidth demand flow.

After setting policies in all the intermediate switches, the two flows from client h1 is sent simultaneously at time zero for 60 seconds. After five seconds later, the other two flows from h2 is sent for 60 seconds while the lest traffic flows from the h3 is start at another 5 seconds later for 60 seconds.

According to the prototype network topology, there are three possible traveling paths between traffic senders and traffic receivers.

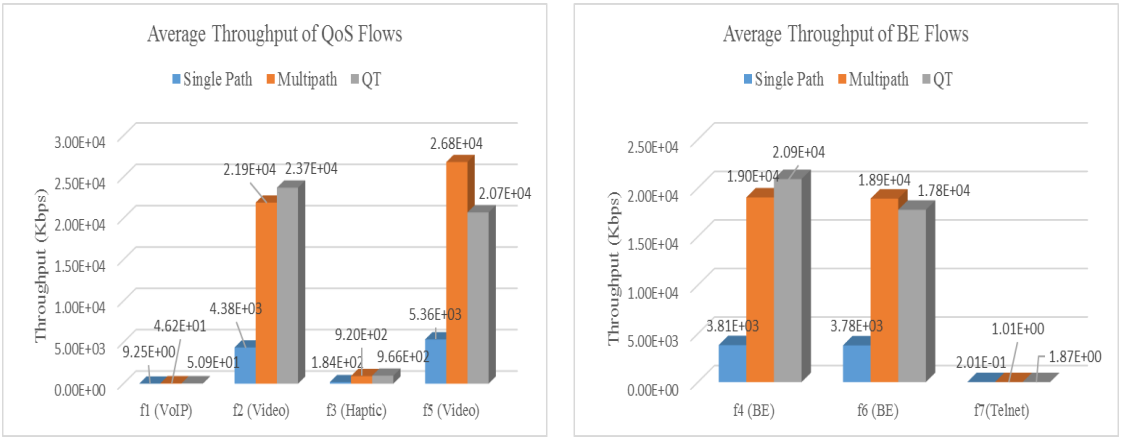
First Path: s1 - s3
Second Path: s1 - s2 - s3
Third Path: s1 - s4 - s3

According to the link delay setting, the first flow from client h1 to h4 will select the second path (s1 - s2 - s3) which has the minimum delay and reserved the user's requested bandwidth for it. In the initial stage, there is no other flow is allocated on the network link, the second flow from client h1 to h5 will select the first path (s1-s3) since the first path has the shortest length. After 5 seconds, h2 requests for the two different flows. The first one from client h2 to h5 will select the minimum delay path among these three possible paths since it is the minimum delay demand flow type. Then, the left flow from client h2 selected the path with to have maximum available bandwidth for it. After 10 seconds later from the most first generated flow, the client h3 request three types of traffic flows and paths are chosen for them based on their demand QoS types. The link will be defined as a bottleneck when the total reserved bandwidth over a link is exceeded then the predefined threshold (80%) [65]. When the bottleneck link is detected, the

highest priority flow will reroute to other best path in order to present bandwidth starvation and packet loss rate for the bandwidth demand flow.

7.3.1.2 Evaluation Results

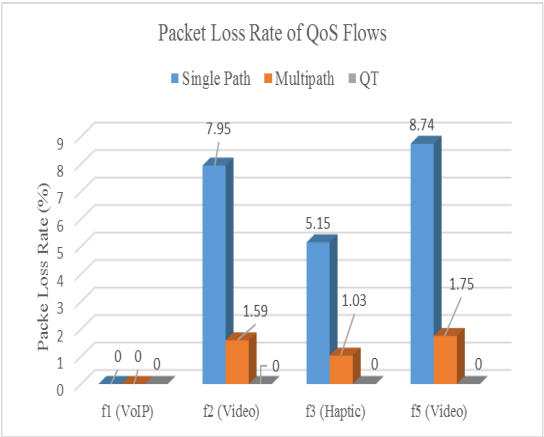
The goal of this experiment is to test the fundamental work of the delay estimation module and the QoS routing module. The performance of the proposed approach (QT) will measures with all of the QoS parameters as the researcher already mentions in section 6.6. Then, the comparison of the experimental results is made by using the proposed scheme with the traditional shortest-path routing scheme and multipath routing schemes. The throughput and packet loss is measured for all traffic flows. The average throughput for both QoS flows types and Best-effort flows types are shown in Figures 7.14 (a), 7.14 (b), respectively.



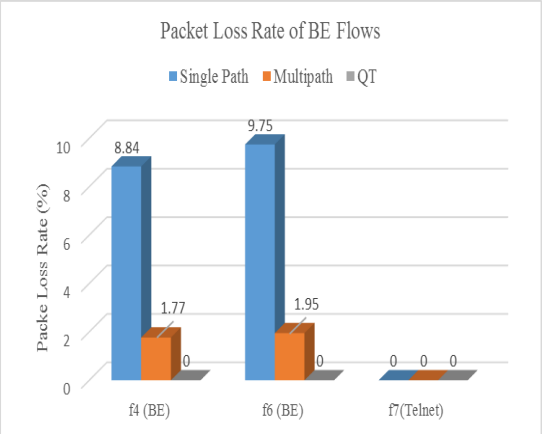
(a) Average Throughput of the QoS flows (b) Average Throughput of BE flows

Figure 7.14 The Average Throughput of the Experiment on Simple Network Topology

According to Figure 7.14, the traditional single path scheme has minimum throughput performance than the other two schemes. This is the consequence of using always the same shortest path (s1, s3) for routing. In Figure 7.14(a), it can be seen that the throughput of network flows in both multipath and the proposed approach (QT) have a high throughput rate than single path routing. This is because all the flows were routed to all the available paths instead of sharing a single path to route flows from the sources to the destinations in both approaches.



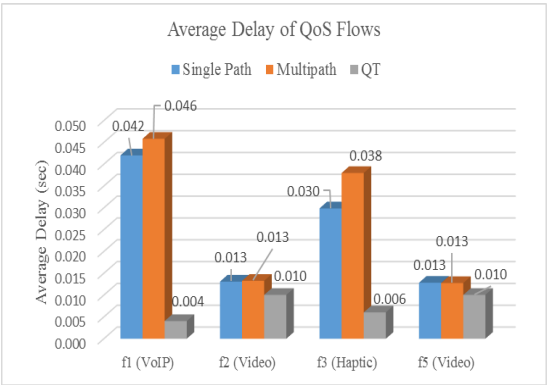
(a) Packet Loss Rate of the QoS flows



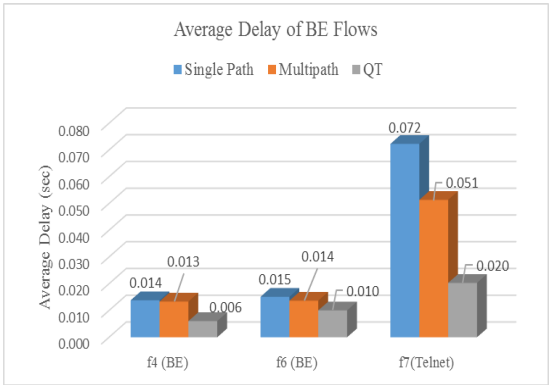
(b) Packet Loss Rate of BE flows

Figure 7.15 Packet Loss Rate of the Experiment on Simple Network Topology

The packet loss rate of the three approaches for each flow is shown in Figures 7.15. According to the experiment results, the proposed approach (QT) has a zero or nearly zero packet loss rate for all traffic. Figure 7.15, the packet loss rate is high in a traditional single-path routing scheme. It is observed that the packet loss rate of our proposed scheme is less than those of the other two schemes. Although multipath have some packet loss rate of approximately 1 % in each flow, it can be acceptable and neglectable.



(a) Average Delay of the QoS flows



(b) Average Delay of BE flows

Figure 7.16 Delay of the Experiment on Simple Network Topology

From Figure 7.16, it can be seen that the delay performance of the proposed approach (QT) for all the flows is less than both single path and multipath routing schemes. From the experiment results, it is observed that the proposed approach (QT) works well and provides better performance in terms of packet loss rate for the QoS

traffics. Later, the performance of the QoS routing with large network topology will evaluate.

This experiment focused on the fundamental work of the proposed approach (QT) in simple network topology by using the queuing technique. The goal is to improve the link utilization while reducing the packets loss rate as the QoS factor in the overall network. To realize this goal, the proposed approach (QT) tries to allocate the network traffic dynamically by using the available bandwidth which is provided from the network monitoring module. The results of the experiments showed that the proposed approach (QT) achieves better performance in terms of throughput, end-to-end delay and packet loss rate than that of traditional shortest-path and multipath routing.

7.3.2 Experiment 2: Abilene Network Topology

For testing the QoS routing module, Mininet is used to create the network topology. Open vSwitch is chosen due to its flexibility and good support for OpenFlow switch specifications. The topology was set up based on Abilene core topology in Mininet OpenFlow network with 1 controller and 11 switches as shown in Figure 7.17. DITG was used as a testing tool to generate UDP data streams in their simulation.

7.3.2.1 Experimental setup

To test the proposed QoS routing module, a network topology with 11 switches (Open vSwitch) and 8 hosts is created in Mininet. The topology is shown in Figure 7.17. The bandwidth of the links between all switches is set to 100 Mbps.

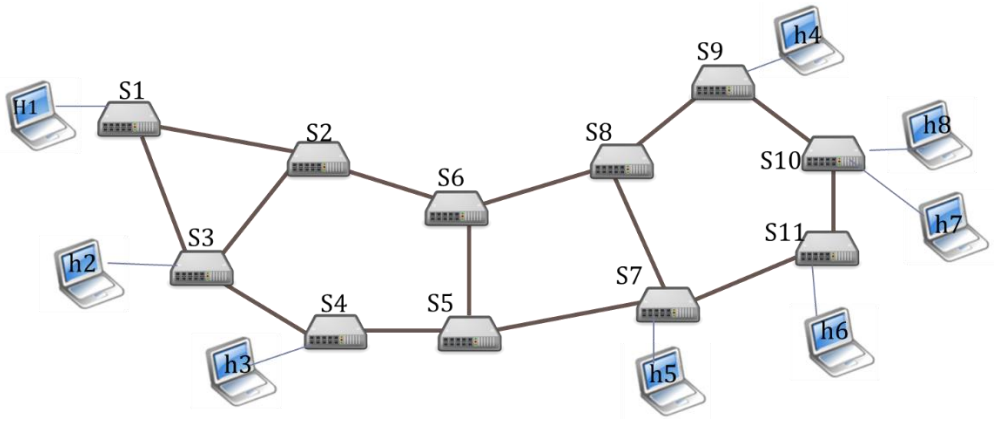


Figure 7.17 Abilene Network Topology

After that, 9 flows are generated using DITG. Five of these flows were critical flows which include three minimum delay demand flows and two bandwidth flows. Table 7.9 shows the list of flows in this experiment. Using Python script, we started the Mininet topology and then generated the flows in the order shown in the below table.

Table 7.9 Network Experiment Flows for Abilene Network Topology

Flow id	Source-Destination	Type of Demand	Packet rate (pps)	Packet size (Bps)	Time (sec.)	Type
1	h1-h4	Min-delay	VoIP	-	60	UDP
2	h1-h6	Bandwidth demand	3500	1024	60	UDP
3	h1-h8	BE	1000	1024	60	UDP
4	h2-h5	Min-delay	1000	128	60	UDP
5	h2-h6	Bandwidth demand	1000	1024	60	UDP
6	h2-h7	BE	3000	1024	60	UDP
7	h3-h4	Min-delay	VoIP	-	60	UDP
8	h3-h7	BE	3000	1024	60	UDP
9	h3-h8	BE	1000	1024	60	UDP

At the beginning of the experiment, multiple (eight) flows were generated for 60 seconds. Firstly, three types of different traffic flow from client h1 to different servers were generated and followed three other types of flows from client h2. Lastly, three flows from client h6 were generated. The average results were calculated based on 5 running times.

Firstly, the network controller needs to choose the monitoring time interval before starting the experiment. The proposed system needs to regularly query the switches to retrieve flow statistics using the equations described in section 5.2.1. Hence, the proposed system used the fixed polling method which may poll all the active flows after the fixed timeout expires. The available bandwidth is calculated by the network controller when the monitoring module receives the number of bytes sent and the

duration of each flow. However, frequently updating the flow information may increase the monitoring overhead.

To examine the network polling interval, the network delay is extrapolated in this experiment. According to Figure 7.18, extrapolation gives fewer network delay results in 3 seconds for all traffic. But accuracy may largely depend based on how frequently the controller is polling the switches to get the network statistics and how dynamically the network traffic is changing.

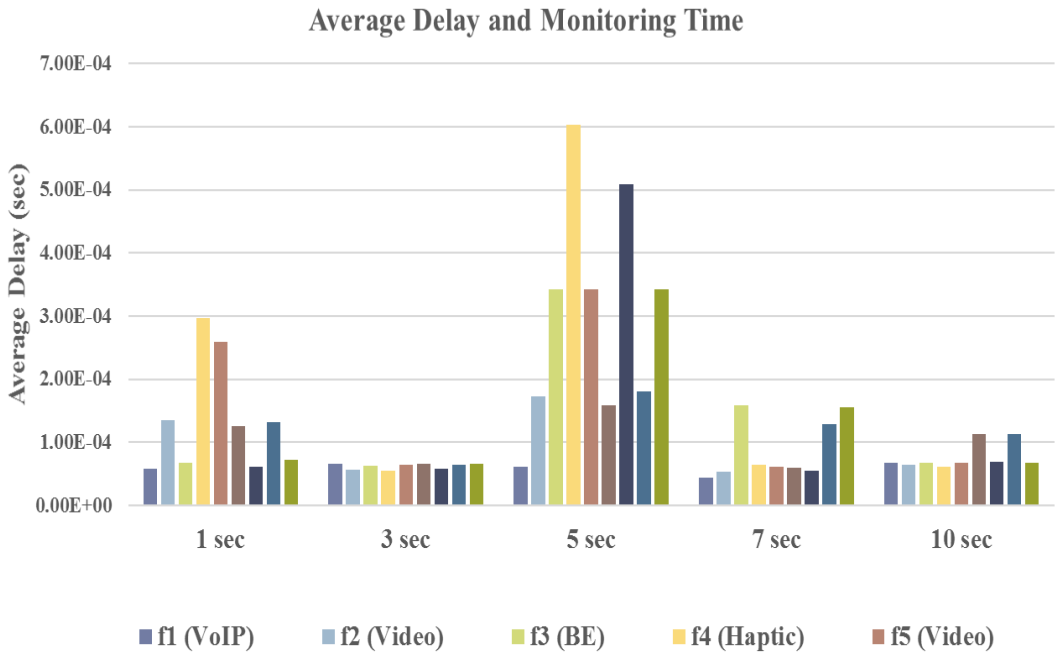
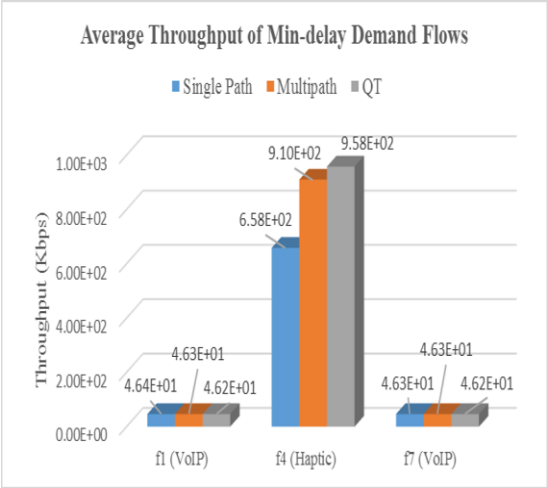
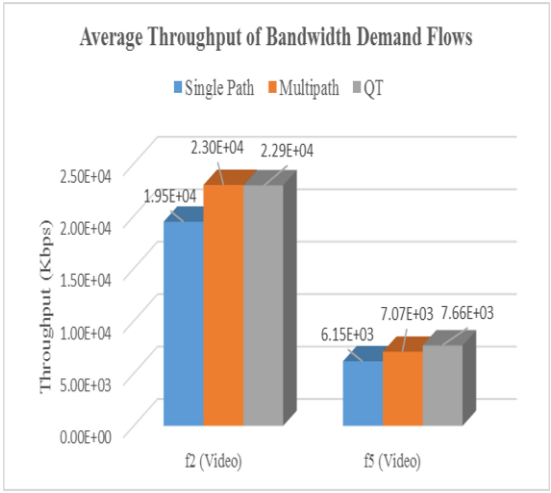


Figure 7.18 The Delay-based Extrapolation

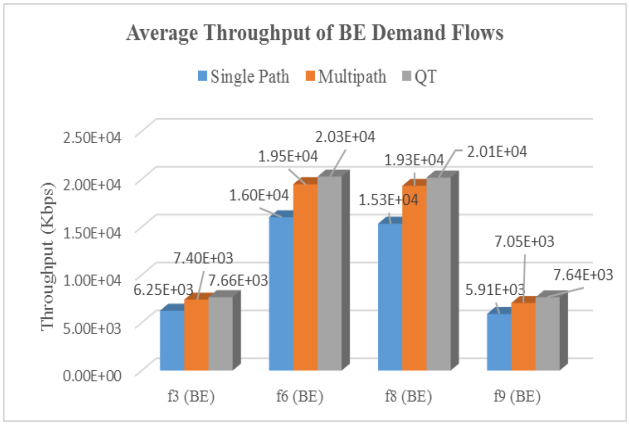
7.3.2.2 Evaluation results

The performance of the proposed scheme was analyzed in terms of throughput and packet loss rate. A comparison between the proposed scheme, the traditional single-path routing scheme, and the flow-based multipath routing scheme was made. The proposed scheme installed the priority flow rule reactively according to user demand. The throughput of the experiment is shown in Figure 7.19.





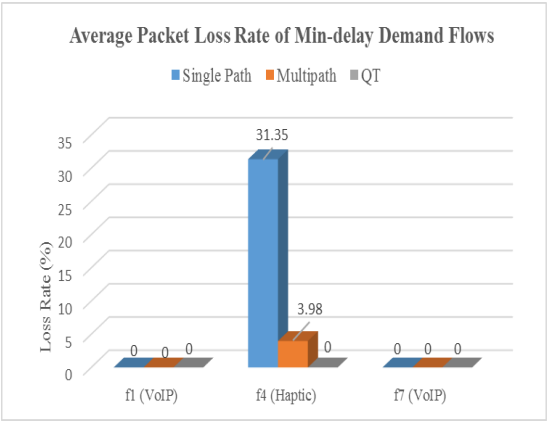
(a) Throughput of Minimum delay demand flows (b) Throughput of Bandwidth demand flows

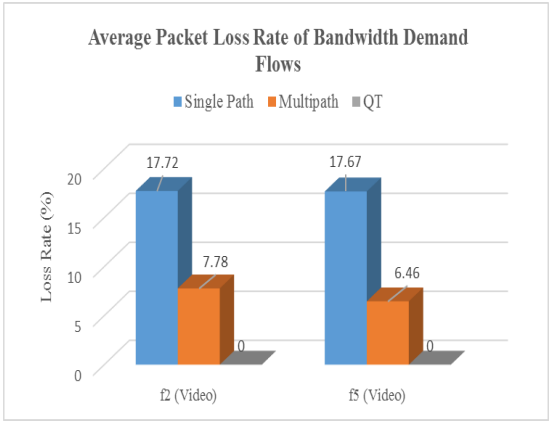


(c) Throughput of BE flows

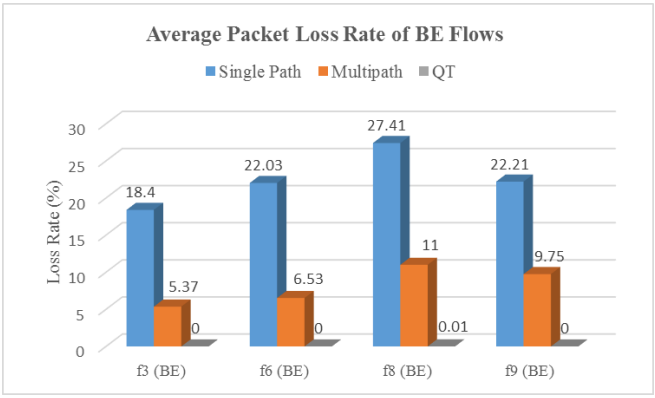
Figure 7.19 The Average Throughput of the Experiment on Abilene Network Topology

According to the experiment results, as shown in Figure 7.19, the traditional single path scheme has worse throughput performance than the other two schemes in almost every flow. This is the consequence of using always the same shortest path for routing due to its sharing one single path. The throughput of the minimum delay demand flows is shown in Figure 7.19(a). Compared with Figure 7.19(b) and 7.19(c), the amount of throughput in the proposed approach (QT) is larger than the single path routing and multipath routing approach. Therefore, the results of the average throughput for all traffic flow is better in the proposed resource allocation scheme.





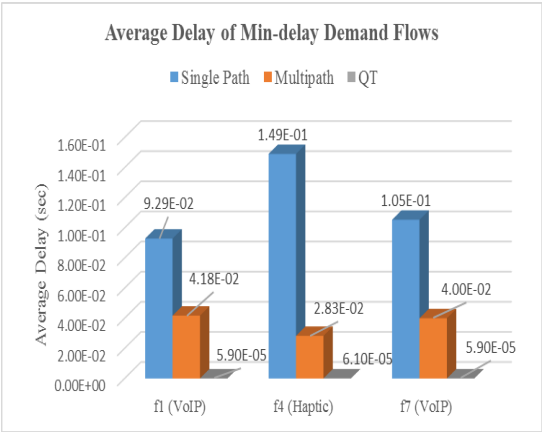
(a) Packet loss rate of Minimum delay demand flows (b) Packet loss rate of Bandwidth demand flows

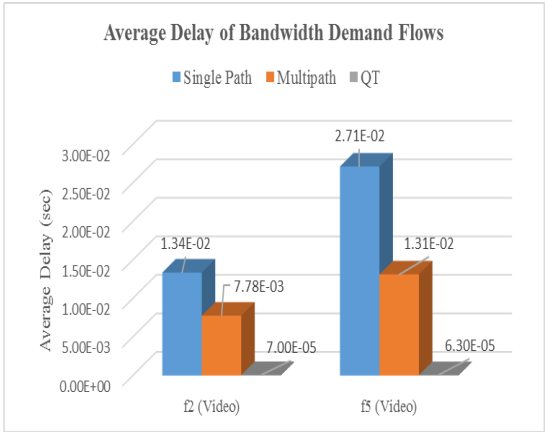


(c) Packet loss rate of BE flows

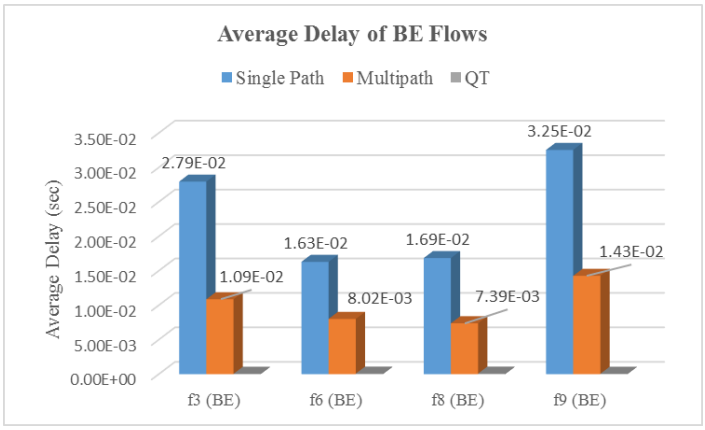
Figure 7.20 The Packet Loss Rate of the Experiment on Abilene Network Topology

Hence, the packet loss rate is also a QoS parameter which is widely used in the network area, we demonstrated the comparison of the packet loss rate in the two aforementioned schemes. The packet loss is unavoidable since the total requested bandwidth of the three types of flows is higher than the maximum available bandwidth of the links (100 Mbps). According to the results of Figure 7.20(a) (b), the total packet loss rates for the critical flows which included the minimum delay demand flows and bandwidth demand flows are clearly zero in the proposed (QT). Therefore, the proposed allocation scheme can provide better performance in terms of packet loss rate for the QoS traffics. Moreover, the packet loss rate of the BE traffic is also less than both of Single Path and Multipath routing.





(a) Average Delay of Minimum delay demand flows (b) Average Delay rate of Bandwidth demand flows

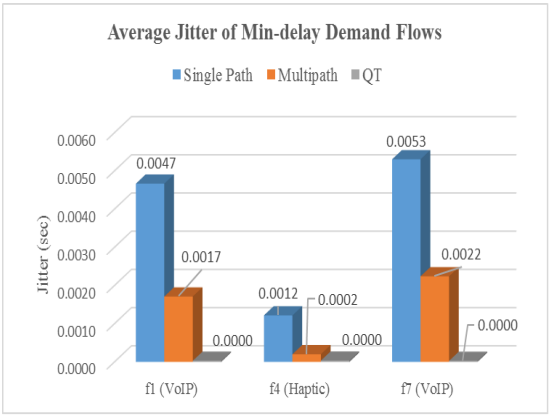


(c) Average Delay rate of BE flows

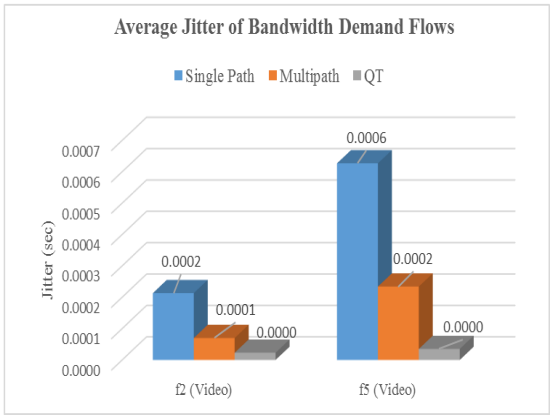
Figure 7.21 Average Delay Rate of the Experiment on Abilene Network Topology

There are many measures to specify different aspects of QoS requirements. Delay is one of the most important parameters and used to measure the performance of the network. From the point of application perspective, the delay is very sensitive in some applications like VoIP and haptic, etc. Since the proposed approach (QT) tries to meet the delay factor for these types of traffic flows, the delay value will also measure.

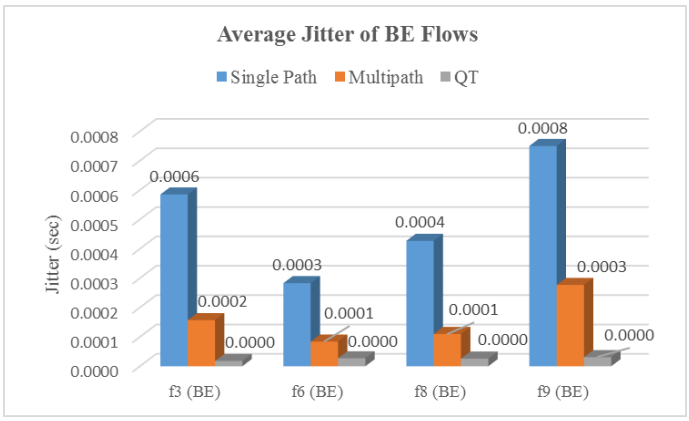
Figures 7.21 and 7.22 show the average delays and jitter of the three schemes for each flow, respectively. Figure 7.21 shows that the delay performance of our proposed scheme for all the flows is obviously less than both single path and multipath routing schemes. The comparison of the jitter value is described in Figure 7.22. According to the results of Figure 7.21 and 7.22, the researcher can conclude that the proposed approach (QT) provides the low delay variation as the QoS factors for the minimum delay demand flows.



(a) Jitter of Minimum delay demand flows



(b) Jitter of Bandwidth demand flows



(c) Average Delay rate of BE flows

Figure 7.22 Jitter of the Experiment on Abilene Network Topology

The focus of this experiment is on providing better service for QoS flows based on the user demand, by dynamically setting up forwarding paths in the data plane. To that end, the control program will monitor the status of the network and direct critical flows over a better path by installing OpenFlow rules on the switches. A QoS routing module is developed and implemented on the controller.

The performance evaluation shows that the proposed approach (QT) can significantly improve the throughput and reduce the delay value obtained by the QoS flows, compared with the shortest path routing and multipath routing used in current networks. Moreover, the proposed approach (QT) can provide better performance in terms of packet loss rate for the QoS traffics.

7.3.3 Experiment 3: Fat-tree Network Topology

A “fat-tree” network is a tree with hosts at the leaves and increasing capacity between switches forming the trunk. These trees are useful because they allow an impressive number of hosts to be connected. It is common to add some diversity of connections to add robustness. This is the norm of datacenter and campus design, with parts of the tree often named Core, Aggregation and Access layers. The escalating bandwidth towards the core makes this design unaffordable for networks that carry a lot of traffic. Testbed setup is introduced in session 7.3.3.1 and results discussion is described in the next subsection.

7.3.3.1 Experimental Setup

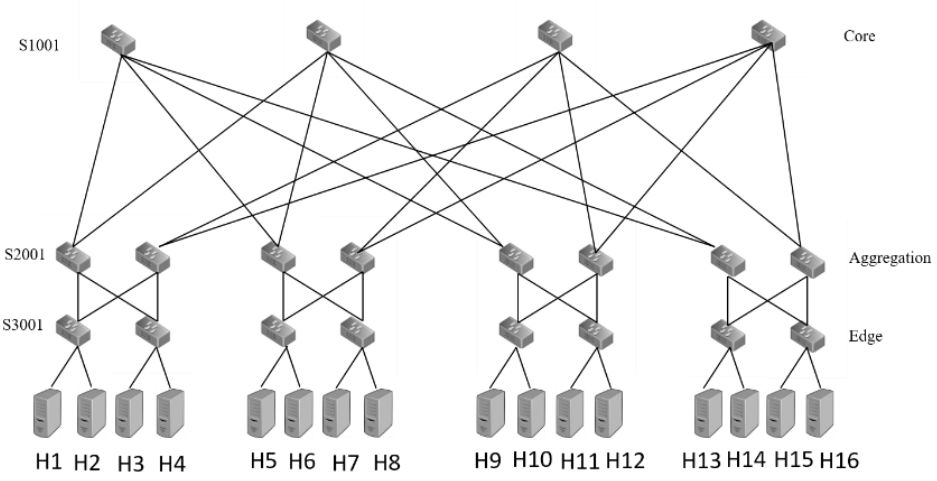


Figure 7.23 Fat-tree Network Topology

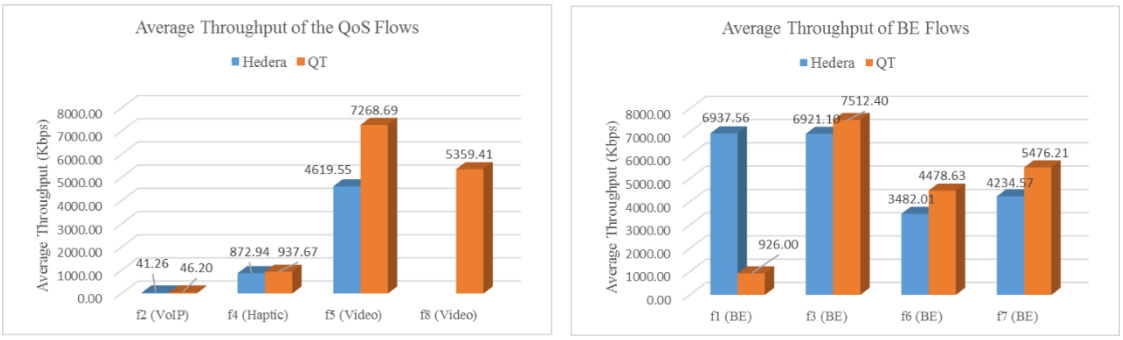
To check the effectiveness of the proposed approach (QT) in the data center network, fat-tree network topology was built as shown in Figure 7.23 . and made a performance testing by comparing it with the famous Hedera, the traffic scheduling approach in data center network. Hence, Hedera can only be used in data center network topology, the proposed approach (QT) try to support not only datacenter but also various network topology. The following network traffic flows are generated as shown in Table 7.10 to measure and examine if the proposed approach (QT) can be also used in the datacenter network.

Table 7.10 Network Experiment Flows for Fat-tree Network Topology

Flow id	Source-Destination	Type of Demand	Packet rate (pps)	Packet size (Bps)	Time (sec.)	Type
1	h1-h9	BE	1000	1024	60	UDP
2	h2-h10	Min-delay	VoIP	-	60	UDP
3	h3-h11	BE	1000	1024	60	UDP
4	h4-h12	Min-delay	1000 (Haptic)	128	60	UDP
5	h5-h13	Bandwidth demand	1500	1024	60	UDP
6	h6-h14	BE	1000	1024	60	UDP
7	h7-h15	BE	1000	1024	60	UDP
8	h8-h16	Bandwidth demand	1500	1024	60	UDP

At the beginning of the experiments, all the above eight flows, as expressed in Table 7.10. are generated simultaneously one after another.

7.3.3.2 Evaluation Results

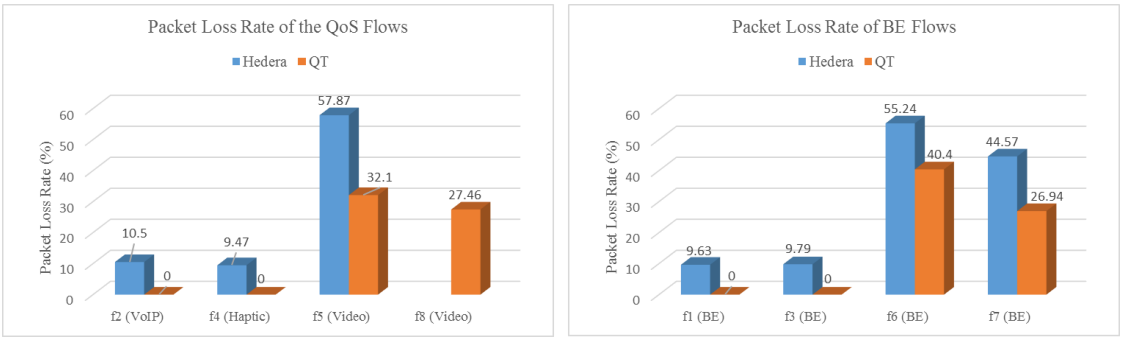


(a) Average Throughput of the QoS flows (b) Average Throughput of BE flows

Figure 7.24 Throughput of the Experiment on Fattree Network Topology

The throughput comparison of the experiment is depicted in Figure 7.24. Figure 7.24 (a) shows that 12% for the flow-id 2 (VoIP), 7% for the flow-id 4 (Haptic), and 57 % for the flow-id 5(video) throughput improvement as compared with Hedera for this experiment. According to the experiment results, the proposed approach (QT) can handle all of the generated flows (eight flows) which are come in the simultaneous form. Hence, Hedera can accept only seven flows in these conditions. This means that the

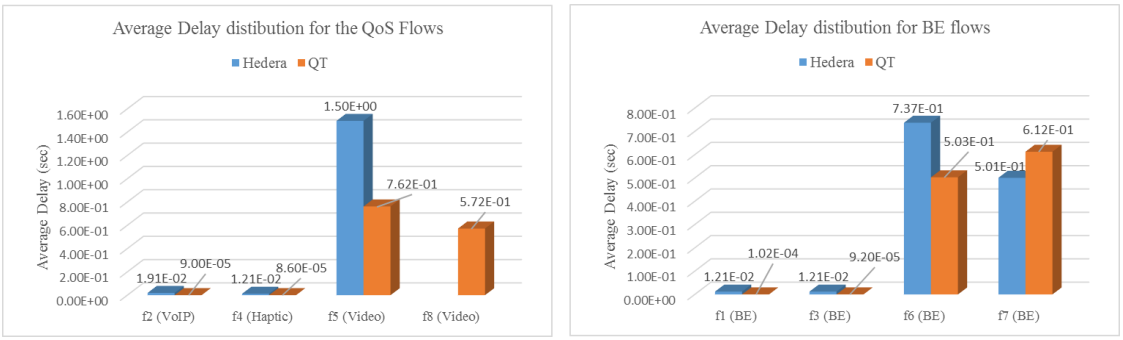
proposed approach (QT) has better link utilization and effective allocation, QT considers the flow priority and link utilization in flow reroute while Hedera only depends on link utilization. On the other hand, Figure 7.25 depicts the packet loss rate for this experiment. The number of packet loss rates in the proposed approach (QT) is obviously smaller than the Hedera approach. Therefore, the result of the average packet loss rate of the overall traffic flows is better in the proposed resource allocation scheme.



(a) Packets loss rate of QoS flows (b) Packet loss rate of BE flows

Figure 7.25 Packet Loss Rate of the Experiment on Fattree Network Topology

The average delay for the QoS flows and the BE flows can be seen that in Figure 7.26. According to the graph of Figure 7.26 (a), the proposed approach (QT) has the low delay value compared with the Hedera for all the QoS traffic flows. However, it can be seen that there is one of the flow showing the high delay value than the Hedera in BE flows in Figure 7.26 (b) and this may happen when the traffic are come simultaneously enter the network, the controller may allocate one or more flows in the same route before updating the link utilization information.



(a) Average Delay of QoS flows (b) Average Delay rate of BE flows

Figure 7.26 Delay of the Experiment on Fattree Network Topology

In recent years there has been an increasingly growing interest in the data center and cloud computing environment. This is motivated by the need for efficient utilization of computing resources and reducing costs. Such infrastructure usually hosts various kinds of applications for different clients. Each application/client has its own set of requirements, typically defined in their Service Level Agreements (SLAs). Quality-of-Service (QoS) requirements include end-to-end bandwidth and latency among other attributes, as we discussed in previous chapters. Several efforts have been made to address the challenges of providing QoS to various types of network applications in different environments using various protocols and techniques. QoS provisioning and monitoring in the cloud-based data center network are even more difficult due to the complexity of its shared infrastructure environment.

The proposed approach (QT) try to meet the user demand QoS factors of the flows and confirmed that the required QoS is guaranteed for high priority flows in the data center network. The experimental results showed that the proposed approach (QT) has better results than the existing flow scheduling approach, Hedera in terms of throughput, packet loss rate, and average end-to-end delay.

7.4 Chapter Summary

In this experiment, the end-to-end dynamic bandwidth allocation scheme based on QoS demand in SDN is presented and confirmed that the QoS is guaranteed for high priority flows. When multiple flows arrive simultaneously, allocating network flows over the same link happen due to the periodic flow monitoring. Consequently, the flow rerouting to the alternative path is essential to provide the QoS performance guarantee. The congestion handling module in QT used the flow priority to reroute started from the high priority flows in order to provide high throughput and QoS guarantee. The experimental results showed that the proposed scheme outperforms the existing single-path routing, multipath routing and Hedera in terms of throughput, packet loss rate, and average end-to-end delay.

CHAPTER 8

CONCLUSION AND FUTURE WORK

In this final chapter, all the work in this thesis is concluded in brief and the future work is explained that may improve the methods of the system in the future study.

8.1 Thesis Summary

This section summarized the research work in the previous chapters in order to better understanding the proposed system.

Chapter 1 is the preamble of the research work. In Chapter 1, a brief background theory related to the SDN is introduced and listed the research problems firstly. Then, research motivations of the thesis that lead to the necessity of the research work are explained. Finally, the objectives and contributions of the proposed system which are a significant part of the thesis are presented.

Chapter 2 summarizes some fundamental research works in the major areas relating to the resourced allocation as well as QoS and the traffic engineering solutions in SDN environments.

In the next chapter, Chapter 3 is about the theoretical background of the proposed system. It initially outlines the SDN reference architecture and its backend theory and technology. This chapter helps the reader to a better understanding of the SDN environment and its workflows.

Chapter 4 presents the end-to-end QoS from the origin of the IP-network start to a current SDN environment. The goal of this chapter is to integrate the theory findings across SND, TE, QoS and resource allocation approaches.

Chapter 5 covers the proposed end-to-end dynamic bandwidth resource allocation and a detailed explanation of its components. The chapter begins with the comprehensive descriptions of the system architecture for the proposed resource allocation scheme and its components are discussed.

In Chapter 6, the conceptual design of the proposed system and its implementation are presented. Then, the explanation of detail design requirements are discussed.

Chapter 7 presents the various experiments with the examples of network topologies to give the reader an understanding of how the platform operates. In this chapter, the bandwidth guaranteeing system with OpenFlow protocol is initially explored by using software switches namely CPqD and OVS. After that, the prototype implementation and the demonstration of the proposed system in different network topology is presented. Then, the analysis of the proposed system and evaluation results are discussed.

8.2 Conclusion

With global Internet traffic growing by an estimated 22% per year, the demand for bandwidth is fast outstripping providers' best efforts to supply it. Providing higher bandwidth is just not enough because that involves a higher cost which both the providers and the consumers cannot afford. Therefore, to handle the issue with limited costs, the best we can do is control the bandwidth with which the data are being sent from the source to the desired destination. The network administrator can eliminate the paths that have lower bandwidth (bottleneck) and select a path with the highest bottleneck bandwidth using an existing algorithm. Identifying network bottlenecks is very useful for end-users and service providers. Unfortunately, it is very hard to identify the location of bottlenecks unless one has access to link load information for all the relevant links.

Moreover, a number of today's network applications such as media streaming and cloud services require steady network resources with strict Quality of Service (QoS) requirements. In general, network administrators are able to manage their resources more efficiently without provisioning the network successively by using the QoS control mechanism and the related notion of traffic engineering. It is not easy to make sure that efficient bandwidth allocation is done in order to provide high QoS for each data flow. If congestion occurs in a network, packets are simply dropped instead of being buffered or sent out after idle periods. There is no bandwidth guarantee about flows and their rates without QoS control. A network administrator can implement the traffic policing where flows can only influence each other based on predetermined parameters with the help of a QoS control mechanism. Moreover, the QoS requirement and importance may vary according to service type, price and user's requirement. Also, the QoS provisioning mechanism of a network depends on the user's requirement, availability of resources,

price, service types etc. Providing high QoS in existing network architectures is a long-standing and still open issue in the networking area.

The emerging networking technology, SDN is introduced to address this issue efficiently for modern network architectures like 5G. In SDN, OpenFlow provides flow level programmability to program the network according to QoS requirements of the applications. Since SDN and OpenFlow enable networks to be more controllable and intelligent with the help of programmability, network administrators no longer have to leave networks unmanaged. Currently, OpenFlow is supported for QoS in the SDN environment by two features, namely the queue and meter. A queue is an egress packet queuing mechanism in the OpenFlow switch port. Although OpenFlow supports the queue features, it does not handle queue management; it is just able to query queue statistics from the switch. Therefore, the queuing feature of OpenFlow is a property of a switch port [71].

In any given network, usually, the overall bandwidth is competitively shared among various application traffic. According to the traditional single-path routing scheme, all of the traffic share the same link and compete over the network link bandwidth. Congestion happens when the traffic load exceeds the network link bandwidth. If congestion occurs in the network, the network will face the packet loss. When the packet loss exists on the network, the users will experience large delays and service degradation. However, there may be more than one single path to reach a particular destination, and some paths may be underutilized. Suitable path selection from among the multiple paths to optimize the overall network performance is one of the critical issues in the network area. Another major challenge is the dynamic link bandwidth allocation with congestion management that can support the QoS requirements for each traffic and alleviate the service degradation for high priority flows.

To solve these problems, an end-to-end dynamic bandwidth resource allocation scheme based on QoS demand is proposed in SDN to support the QoS requirements for an individual flow. SDN is an emerging architecture that may play a critical role in future network architectures. SDN can provide a global network view of the network resources and their performance indicators such as link utilization and the network congestion level which can be used in network resource allocation. By using the benefits of SDN,

the controller makes the routing decision based on the global view of the network resources in an SDN network.

In the proposed QT, the flow priority and the dynamic characteristics of the network link are considered in order to provide the high QoS performance for high priority flows. In addition, we calculate feasible paths for all the traffics that can satisfy the user bandwidth demands. In order to mitigate the flow performance degradation and congestion, the controller checks the link bandwidths by reserving the required bandwidths for incoming flows. If the link bandwidth is smaller than the predefined threshold value, the bottleneck link will be defined and the highest priority flow from the bottleneck link will be rerouted to an alternative one that has enough bandwidth for the rerouting flow. Furthermore, to improve the performance and to ensure the QoS of the high priority flows, a queue mechanism provided by the OpenFlow at the data link level is used. The goal is to improve the QoS performance of the high priority flows while providing the required bandwidth resources and less packet loss rate as QoS factors in the overall network.

According to our preliminary experiments, the OpenFlow Queueing mechanism improves the QoS by providing a bandwidth guarantee for the high priority traffic is confirmed. Therfore, the fundamental work of the proposed approach (QT) in simple network topology is evaluated by using the queuing mechanism in our experiment 1 (section 7.3.1). From this experiment, we can observe that the proposed approach (QT) works well and provides better performance in terms of packet loss rate for the QoS flows. Later, the performance of the QoS routing with large network topology is evaluated in our experiment 2 (section 7.3.2). In this experiment 2, we analyse the network monitoring interval to query the statistics of the network from the switches. Accoding to our experiment, we suggested that the three seconds interval is the suitable choice for our testing with Abilene Network Topology. We found that the accuracy may largely vary based on how frequently the controller is polling the switches to get the network statistics and how dynamically the network traffic is changing. In experiment 3 (section 7.3.3), we compare with Hedera approach, the most popular flow allocation approach in the data center network. According to the experiment 3 (section 7.3.3) , the proposed QT outperforms 12% for the flow-id 2 (VoIP), 7% for the flow-id 4 (Haptic), and 57 % for the flow-id 5(video) in the throughput performance than Hedera. The proposed QT considers the flow priority and link utilization in flow rerouting whereas

Hedera only depends on link utilization. Due to the evaluation reults, the proposed QT has better link utilization and effective allocation compare with Hedera approach and provide better performance for all QoS flows.

8.3 Limitations and Future Work

The resource allocation proposed in this thesis has used traffic engineering and predicted network states for routing decisions. Currently, the proposed system doesn't provide capabilities for queue management due to the controller can only query some queue statistic and limited configuration parameters through the OpenFlow protocol. The trade-off between measurement overhead and real-time statistics should be carefully considered since the network is measured actively in every 'n' seconds to get the real-time update measurement result. Moreover, the LLDP protocol that the proposed system used to estimate the network dealy is not suitable for a large network. There we need to find a more suitable way to estimate the link delay for a large network.

For the future work, more realistic techniques such as effective queue scheduling in the data plane and apply a metering feature of the OpenFlow protocol in the control plane should be implemented and investigated. The application-aware approach should be studied to allocate the bandwidth automatically by estimating the amount of the bandwidth resources that the flow requires in real-time. It should, therefore, be combined with admission control to protect the network from severe overload and end-to-end flow control to achieve fairness. Furthermore, since different services are sensitive with respect to different QoS measures, a combined metric for route optimization should be investigated. There will be a plan to explore additional traffic engineering (TE) methods to ensure the QoS guarantee as well as larger platforms for the approach.

AUTHOR'S PUBLICATIONS

[1] Nwe Thazin, Khine Moe Nwe, "Efficient Resource Allocation Framework for Network Function Virtualization" , in Proceedings of the 15th International Conference on Computer Applications (ICCA 2017), Yangon, MYANMAR, February 2017. Page [112-116]

[2] Nwe Thazin, Khine Moe Nwe, Yutaka Ishibashi, "Resource Allocation Scheme for SDN-Based Cloud Data Center Network" , in Proceedings of the 17th International Conference on Computer Applications (ICCA 2019), Yangon, MYANMAR, February 2019. Page [15-22]

[3] Nwe Thazin, Khine Moe Nwe, Yutaka Ishibashi, "End-to-End Dynamic Bandwidth Resource Allocation Based on QoS Demand in SDN", in Proceedings of the 25th Asia-Pacific Conference on Communications (APCC), Ho Chi Min, Vietnam, November 2019. Page [244-249]

[4] Nwe Thazin, Khine Moe Nwe, "Quality of Service (QoS)-Based Network Resource Allocation in Software Defined Networking (SDN)", International Journal of Sciences: Basic and Applied Research Journals (IJSBAR), ISSN: 2307-4531 [Online], January 2020. (To be appeared).

BIBLIOGRAPHY

[1] S. Agarwal, M. Kodialam, and T. Lakshman, "Traffic engineering in software defined networks," in Proceedings of the 32nd IEEE International Conference on Computer Communications (INFOCOM). IEEE, pp. 2211-2219, 2013.

[2] U. Agarwal and V. Gupta, "Network Routing Algorithm using Genetic Algorithm and Compare with Route Guidance Algorithm," Int. J. Sci. Res. Eng. Technol., pp. 3-4, 2014.

[3] A. V. Akella and K. Xiong, "Quality of service (QoS)-guaranteed network resource allocation via software defined networking (SDN)," in Proc. IEEE 12th Int. Conf. Dependable, Autonomic Secure Comput. (DASC), pp. 7-13, Aug. 2014.

[4] I. Akyilidiz, A. Lee, P.Wang, M. Luo and W. Chou,"A roadmap for traffic engineering in SDN -OpenFlow networks", Computer Networks 71,pp. 1-30, Elsevier Publications, 2014

[5] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in Proc. Netw. Syst. Design Implement. Symp. (NSDI), Vol. 10., pp. 19, 2010.

[6] A. Alijawad, P. Shah, O. Gemikonakli, and R. Trestian. "Policy-based QoS management framework for software-defined networks." In 2018 International Symposium on Networks, Computers and Communications (ISNCC), pp. 1-6, IEEE, 2018.

[7] S. Avallone, S. Guadagno, D. Emma, A. Pescapè, and G. Ventre, "D-ITG distributed internet traffic generator," In Proceeding of 1st International Conference on Quantitative Evaluation of System (QEST). IEEE, pp. 316-317, Jan. 2004.

[8] S. U. Baek, C. H. Park, E. Kim and D. Shin, "Implementation and verification of QoS priority over software-defined networking," In Proceeding of the International Conference on Internet Computing (ICOMP). The Steering Committee of the World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2016.

[9] D. G. Balan and D. A. Potorac, "Linux HTB queuing discipline implementations,"IEEE First International Conference on Networked Digital Technologies, Ostrava, Czech Republic, pp. 122-126, Jul. 2009.

[10] H.Ballani, P.Costa, T.Karagiannis, and A.Rowstron, “Towards predictable datacenter networks”. SIGCOMM Comput. Commun. Rev. Volume 41, Issue 4, pp. 242-253, Aug. 2011.

[11] T. Benson, A. Anand, A. Akella, and M. Zhang, “MicroTE: Fine grained traffic engineering for data centers,” in Proc. 7th Conf. Emerg. Netw. Experim. Technol. (CoNEXT), pp. 1-12, Dec. 2011.

[12] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. “An Architecture for Differentiated Services. RFC 2475 (Informational).” Updated by RFC 3260. Internet Engineering Task Force, 1998. url: http://www.ietf.org/rfc/rfc2475.txt.

[13] J.M.Boley, E.S. Jung, and R.Kettimuthu, “Adaptive QoS for data transfers using software-defined networking.” 2016 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), IEEE, pp. 1-6, 2016.

[14] R. Braden, D. Clark, and S. Shenker. “Integrated Services in the Internet Architecture: an Overview. RFC 1633 (Informational).” Internet Engineering Task Force, 1994. url:http://www.ietf.org/rfc/rfc1633.txt.

[15] K.L. Calvert, W.K. Edwards, N. Feamster, R.E. Grinter, Y. Deng, and X. Zhou, “Instrumenting home networks.” ACM SIGCOMM Computer Communication Review, Vol. 41, Issue. 1, pp. 84-89, 2011.

[16] D. D. Chowdhury, “Forces: an Elastic Routing Architecture for,” no. August, 2016.

[17] A. R. Curtis, W. Kim, and P. Yalagandula, “Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection,” in Proc. IEEE INFOCOMM, pp. 1629-1637, Apr. 2011.

[18] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, “DevoFlow: Scaling flow management for high-performance networks,” ACM SIGCOMM Comput. Commun. Rev., Vol. 41, no. 4, pp. 254-265, 2011.

[19] A. Dainotti, A. Pescap´e, and C. Sansone, “Early Classification of Network Traffic through Multi classification”. In: Domingo-Pascual, J., Shavitt, Y., Uhlig, S. (eds.) TMA 2011. LNCS, vol. 6613, pp. 122-135. Springer, Heidelberg 2011.

[20] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” IEEE Trans. Evol. Comput., vol. 6, no. 2, pp. 182-197, 2002.

[21] M. Dillon, T. Winters, “Virtualization of Home Network Gateways,” in Computer, vol.47, no.11, pp.62-65, Nov. 2014

[22] H.E. Egilmez, “Distributed QoS Architectures for Multimedia Streaming over Software Defined Networks,” Multimedia, IEEE Transactions on, vol.16, no.6, pp.1597-1609, Oct. 2014.

[23] 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 Proc. Signal Inf. Process. Assoc. Summit Conf., pp. 1-8, Dec. 2012.

[24] S. Fang, Y. Yu, C. H. Foh, and K. M. M. Aung, “A loss-free multipathing solution for data center network using software-defined networking approach,” in APMRC, 2012 Digest , pp.1-8, Oct. 31 2012-Nov. 2 2012.

[25] N. Feamster, “ Outsourcing home network security,” in Proceedings of the 2010 ACM SIGCOMM workshop on Home networks, pp. 37-42. ACM, 2010.

[26] W. C. Feng, K. G. Shin, D. D. Kandlur, and D. Saha, “The Blue active queue management algorithms,” IEEE/ACM Trans. Netw., vol. 10, no. 4, pp. 513-528, 2002

[27] Roy T. Fielding (2000). “Chapter 5: Representational State Transfer (RE ST)”. Architectural Styles and the Design of Network-based Software Architectures (Ph.D.). University of California, Irvine.

[28] M. Gerla and L. Kleinrock, “Flow Control: A Comparative Survey,” IEEE Trans. Commun., vol. 28, no. 4, 1980.

[29] P. Goransson, and B. Chuck. “Software-Defined Networks A Comprehensive Approach.” In IEEE Communication Surveys & Tutorials, pp. 7-17, 2014.

[30] K. Greene, (2009), “TR10: Software-defined networking. MIT Technology Review, March/April 2009” http://www2.technologyreview.com/article/ 412194 /tr10-software-defined-networking/

[31] Z. J. Haas and J. H. Winters, “Congestion Control By Adaptive Admission,” Proc. IEEE Int. Conf. Comput. Commun., pp. 560-569, 1991.

[32] B. Heller, R. Sherwood, N. Mckeown, The controller placement problem, 420 Acm Sigcomm Computer Communication Review, vol. 42, issue 4, pp. 7-12, 2012.

[33] S. S. Hong and S. F. Wu, "On interactive Internet traffic replay," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 3858 LNCS, pp. 247-264, 2006.

[34] X. Huang, C. Lin, F. Ren, G. Yang, P. D. Ungsunan, and Y. Wang, "Improving the convergence and stability of congestion control algorithm," in Proceedings -International Conference on Network Protocols, ICNP, pp. 206-215, 2007.

[35] 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 the Proceedings of the Second European Workshop on Software Defined Networks (EWSDN), pp. 87-92, Berlin, Germany, Oct. 2013.

[36] V. Jeyakumar, A. Kabbani, J. C. Mogul, and A. Vahdat, "Flexible Network Bandwidth and Latency Provisioning in the Datacenter," 2014. Available Online: http:// http://arxiv.org/abs/1405.0631

[37] P. Jha, "End-to-end Quality-of-Service in Software Defined Networking" by University of Dublin, Trinity College," no. September, thesis, 2017.

[38] J. Jo, S. Lee and J. W. Kim, "Software-defined home networking devices for multi-home visual sharing," in IEEE Transactions on Consumer Electronics, vol. 60, no. 3, pp. 534-539, Aug. 2014. doi: 10.1109/TCE.2014.6937340

[39] M. Karakus and A. Durresi, "Quality of Service (QoS) in Software Defined Networking (SDN): A survey," J. Netw. Comput. Appl., vol. 80, pp. 200-218, 2017.

[40] H. Krishna, N. L. M van Adrichem, and F. A. Kuipers, " Providing bandwidth guarantees with OpenFlow," in Proc. of IEEE 2016 Symposium on Communications and Vehicular Technologies (SCVT), pp. 1-6, 2016

[41] J. Kristoff, "TCP Congestion Control," 2002.

[42] A.Kumar, S.Jain, et al., "BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing," in proceedings of the 2015 ACM

Conference on Special Interest Group on Data Communication (SIGCOMM '15). ACM, New York, NY, USA, 1-14.

[43] Y. Lee et.al, "ALTO Extension for collecting data center resource in real - time", http://datatraceker.ietf-org/doc/draft-lee-alto-ext-dc-resource/

[44] L.E. Li, Z.M. Mao, and J. Rexford. Toward software-defined cellular networks. In Software Defined Networking (EWSDN), 2012 European Workshop on, pp. 7-12, 2012.

[45] F. Li, J. Cao, X. Wang, Y. Sun and Y. Sahni, "Enabling Software Defined Networking with QoS Guarantee for Cloud Applications," 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), Honolulu, CA, 2017, pp. 130-137.doi: 10.1109/CLOUD.2017.25

[46] S. Li et al., "Protocol Oblivious Forwarding (POF): Software-Defined Networking with Enhanced Programmability," in IEEE Network, vol. 31, no. 2, pp. 58-66, March/April 2017.doi: 10.1109/MNET.2017.1600030NM

[47] V. Mann, A. Vishnoi and S. Bidkar, "Living on the edge: Monitoring network flows at the edge in cloud data centers," 2013 Fifth International Conference on Communication Systems and Networks (COMSNETS), Bangalore, pp. 1-9, 2013.

[48] C. A. C. Marcondes, T. P. C. Santos, A. P. Godoy, C. C. Viel and C. A. C. Teixeira, " CastFlow: Clean-slate multicast approach using in-advance path processing in programmable networks," Computers and Communications (ISCC), 2012 IEEE Symposium on, Cappadocia, pp. 000094-000101, 2012.

[49] S. Mehdi, J. Khalid, and S. Khayam. Revisiting traffic anomaly detection using software defined networking. In Recent Advances in Intrusion Detection, pp. 161-180. Springer, 2011.

[50] H. Mekky, F.Hao, S.Mukherjee, Z.Zhang, and T.V. Lakshman. 2014. Application-aware data plane processing in SDN. In Proceedings of the third workshop on Hot topics in software defined networking (HotSDN '14). ACM, New York, NY, USA, pp.13-18, 2014.

[51] J. Metzler, "Understanding Software-Defined Networks," Information Week Reports, pp.1-25, http://reports.informationweek.com/abstract/6/9044/Data-Center/research-understanding-softwaredefined-networks.html,October 2012.

[52] R. Mortier, T. Rodden, T. Lodge, D. McAuley, C. Rotsos, AW Moore, A. Koliousis, and J. Sventek. Control and understanding: Owning your home network. In Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on, pp. 1-10. IEEE, 2012.

[53] S. Natarajan, A. Ramaiah, and M. Mathen, "A software defined cloud gateway automation system using OpenFlow," in Proc. IEEE 2nd Int. Conf. CloudNet, pp. 219-226, Nov. 2013.

[54] K. A. Noghani and M. O. Sunay, "Streaming Multicast Video over Software -Defined Networks," 2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems, Philadelphia, PA, pp. 551 -556, 2014.

[55] D. Palma et al., "The QueuePusher: Enabling Queue Management in OpenFlow," 2014 Third European Workshop on Software Defined Networks, London, pp. 125-126, 2014. doi: 10.1109/EWSDN.2014.34

[56] P. Panwaree, K. Jongwon and C. Aswakul, "Packet Delay and Loss Performance of Streaming Video over Emulated and Real OpenFlow Networks," Conference: Proceedings of the 29th International Technical Conference on Circuit/Systems Computers and Communications (ITC-CSCC), 2014, At Phuket, Thailand

[57] P. Patel et al., "Ananta: Cloud scale load balancing," in Proc. ACM SIGCOMM Conf., pp. 207-218, 2013.

[58] E. Rosen, A. Viswanathan, R. Callon, RFC 3031 : Multiprotocol Label Switching Architecture (2001). URL www.ietf.org/rfc/rfc3031.txt

[59] Y. Sharma, S. C. Saini, and M. Bhandhari, "Comparison of Dijkstra ' s Shortest Path Algorithm with Genetic Algorithm for Static and Dynamic Routing Network," Int. J. Electron. Comput. Sci. Eng., vol. 1, no. 2, pp. 416-425, 2012.

[60] S. Shenker, "A Theoretical Analysis of Feedback Flow Control," in The Conference on Communications Architecture and Protocols (SIGCOMM), 1990, pp. 156-165.

[61] Y. Shi, Y. Zhang et al. "Using Machine Learning to Provide Reliable Differentiated Services for IoT in SDN-Like Publish/Subscribe Middleware." Sensors (Basel, Switzerland) vol. 19,6 1449. 25 Mar. 2019, pp. 1-25.

[62] Z. Shu et al., “Traffic Engineering in Software-Defined Networking: Measurement and Management,” IEEE Access, vol. 4, no. August 2018, pp. 3246-3256, 2016.

[63] I. Stoica, H. Zhang and T. S. E. Ng, “A hierarchical fair service curve algorithm for link-sharing, real-time, and priority services,” in IEEE/ACM Transactions on Networking, vol. 8, no. 2, pp. 185-199, Apr. 2000.

[64] A.Takacs, E. Bellagamba and J. Wilke, “ Software-defined networking: The service provider perspective,” in Ericsson Review , Feb. 2013.

[65] S. Tomovic, N. Prasad, and I. Radusinovic, “SDN control framework for QoS provisioning,” in Proc. Telecommun. Forum Telfor (TELFOR), pp. 111-114, Nov. 2014,

[66] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, “ OpenTM: Traffic matrix estimator for OpenFlow networks,” in Passive and Active Measurement. Berlin, Germany: Springer, pp. 201-210, Apr. 2010.

[67] R. Trivisonno, R. Guerzoni, I. Vaishnavi, and A. Frimpong, “ Network Resource Management and QoS in SDN-Enabled 5G Systems,” 2015 IEEE Global Communications Conference (GLOBECOM), San Diego, CA, pp. 1-7, 2015.

[68] J.T. Tsai, J.C. Fang, and J.H. Chou, “Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm,” Comput. Oper. Res., vol. 40, no. 12, pp. 3045-3055, 2013.

[69] F. P. Tso and D. Pezaros, “Baatdaat: Measurement-Based Flow Scheduling for Cloud Data Centers,” in Proceedings of the 2013 IEEE Symposium on Computers and Communications (ISCC), pp. 765-770 , July 2013.

[70] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, ``OpenNetMon: Network monitoring in OpenFlow software-defined networks,”in Proc. IEEE/IFIP Netw. Oper. Manage. Symp., pp. 1-8, May 2014.

[71] S. J. Vaughan-Nichols, “ OpenFlow: The next generation of the network ?,” IEEE Computer 44 (8) (2011) 13-15. URL http://dblp.unitrier.de/db/jo-urnals/ computer /computer44.html#Vaughan-Nichols11

[72] R. Wallner and R. Cannistra, “An SDN Approach: Quality of Service using Big Switch’s Floodlight Open-source Controller,” Proc. Asia-Pacific Adv. Netw., vol. 35, no. 0, pp. 14, 2013.

[73] C. Xu, B. Chen, H. Qian, Quality of service guaranteed resource management dynamically in software defined network, in: Journal of Communications, vol. 10, pp. 843-850, 2015. doi:10.12720/jcm.10.11.843-850.

[74] R. Yavatkar, D. Hoffman, Y. Bernet, F. Baker and M. Speer, “SBM (Subnet Bandwidth Manager): A Protocol for RSVP-based Admission Control over IEEE 802-style networks.” RFC 2814 (2000): 1-60.

[75] Y. Yiakoumis, K.K.Yap, S. Katti , G. Parulkar, and N. McKeown, “Slicing home networks,” in Proceedings of the 2nd ACM SIGCOMM workshop on Home networks (HomeNets '11). ACM, New York, NY, USA, pp. 1-6, 2011.

[76] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, “FlowSense: Monitoring network utilization with zero measurement cost,” in Proc. Int. Conf. Passive Active Meas., vol. 1. pp. 31-41, 2013.

[77] H. Zhang, X. Guo, J. Yan, B. Liu and Q. Shuai, “SDN-based ECMP algorithm for data center networks,” 2014 IEEE Computers, Communications and IT Applications Conference, Beijing, pp. 13-18, 2014.

[78] G. Zhang, D. Zhang, L. Zhou, and X. Liu. “End-to-end dynamic bandwidth allocation based on user in software-defined networks.” International Journal of Future Generation Communication and Networking 9, no. 9, pp. 67-76, 2016.

[79] L. Zhang, S. Berson, S. Herzog, S. Jamin, RFC 2205, “Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification (1997)”. URL www.ietf.org/rfc/rfc2205.txt

[80] C. Zhang, X. Huang, G. Ma and X. Han, “A dynamic scheduling algorithm for bandwidth reservation requests in software-defined networks,” 2015 10th International Conference on Information, Communications and Signal Processing (ICICS), Singapore, pp. 1-5, 2015.

[81] “Internet Engineering Task Force.” [Online]. Available: http://www.ietf.org/.

[82] “Iperf”, Abailable: http://iperf.sourceforge.net.

[83] “Open Networking Foundation,” accessed: 2016-05-25. [Online]. Available: https://www.opennetworking.org/

[84] “Open Networking Foundation,” accessed: 2016-05-25. [Online]. Available: https://www.opennetworking.org/

[85] “OpenFlow Switch Specification”. [Online]. Available: https://www.open-networking.org/wp-content/uploads/.../ OpenFlow-spec-v1.3.1.pdf. (accessed August 1, 2015).

[86] “CiscoVirtualized Multiservice Data Center Framework, 2016.” [Online]. Available: http://www.cisco.com/enterprise/data-center-designs-cloud-com-puting/white_paper_c11-714729.html

[87] “Mininet: An Instant Virtual Network on your Laptop”. [Online]. Available: http://mininet.org/.

[88] O. N. Foundation, “OpenFlow-open networking foundation” [Online]. Available:https://www.opennetworking.org/sdn-resources/openflow (accessed August 23, 2016).

[89] “ofsoftswitch13 – cpqd”. https://github.com/CPqD/ofsoftswitch13

[90] “ONOS project”. http://onosproject.org/

[91] Onos thesis(2017) - End-to-End Quality of Service in Software Defined Networking.pdf

[92] Open Networking Foundation, “Software-Defined Networking: The new norm for networks,” Tech. Rep., 2012, white paper

[93] “Open vSwitch”. [Online]. Available: http://openvswitch.org/support/

[94] “Project FloodLight”. [Online]. Available: http://www.projectfloodlight.org /floodlight/

[95] “Project OpenDayLight”. [Online]. Available: http://www.opendaylight.org/ project/

[96] “Ryu”. [Online]. Available: http://osrg.github.com/ryu/

[97] SDN Architecture (2014), Issue 1, “Open Networking Foundation”. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technicalreports/TR_SDN_ARCH_1.0_06062014.pdf

[98] “Wireshark”. [Online]. Available: https://www.wireshark.org/

[99] https://blog.sflow.com/

[100] https://tcpreplay.appneta.com/

[101] https://tubularinsights.com/2019-internet-video-traffic/

[102] https://wiki.opendaylight.org/view/OpFlex:Opflex_Architecture

[103] https://www.airtel.in/opennetwork/reportIssues

[104] https://www.cisco.com/c/en/us/products/ios-nx-os-software/quality-of-service-qos/index.html

[105] https://www.pcwdld.com/what-is-netflow