APPLICATION-AWARE TRAFFIC ENGINEERING IN SOFTWARE DEFINED NETWORKING

MAY THU ZAR WIN

UNIVERSITY OF COMPUTER STUDIES, YANGON

November, 2019

Application-aware Traffic Engineering in Software Defined Networking

May Thu Zar Win

University of Computer Studies, Yangon

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

Doctor of Philosophy

November, 2019

<u>Statement of Originality</u>

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 May Thu Zar Win

ACKNOWLEDGEMENTS

I would like to thank Ministry of Education for giving me the opportunity to study Ph. D. Course and allowing me to do the research in University of Computer Studies, Yangon, Myanmar.

I would like to express very special thanks to Dr. Mie Mie Thet Thwin, Rector of the University of Computer Studies, Yangon, for allowing me to develop this research and giving me general support during period of my study.

It is a pleasure to thank my supervisor, Dr. Khin Than Mya, Professor, Head of Faculty of Computer Systems and Technologies, University of Computer Studies Yangon for her patience, motivation, constant encouragement and helpful guidance throughout my research.

I would like to thankful my advisor, Dr. Yutaka Ishibashi, Professor, Nagoya Institute of Technologies for his helpful contribution, invaluable guidance, practical suggestions which inspired me to widen my research from various perspectives.

I would like to express my special appreciation and thanks to my external examiner Dr. Aung Htein Maw, Professor, University of Information Technology, for his insightful comments and suggestions which are invaluable to me.

I need to express my gratitude and special thanks to Dr. Khine Moe Nwe, Course-coordinator of Ph. D. 9th Batch, University of Computer Studies, Yangon, for her support and encouragement throughout all stages of the research.

I would like to express special thanks to Daw Aye Aye Khine, Associate Professor, Head of English Department, for her valuable supports from the language point of view and suggestions which help to improve the thesis writing.

I further would like to thank Dr. Zin May Aye, Professor, Head of Cisco Lab, University of Computer Studies, Yangon, and all my PhD 9th Batch colleagues, especially from the Cisco Lab, and my beloved friends for their motivations, knowledge sharing, constant caring, and unforgettable memories we have made in the last five years.

Last but not least, a very special thanks and gratitude go to my parents, U Maung Kyaw and Daw Cho Cho, my elder brother and sisters, my nephew and niece who gave me their love, kindness, patience, physically and mentally supports, and constant encouragement along the way of my life. Without their efforts, nothing can be done. Finally, I would like to thank every person who were directly or indirectly contributed towards the success of this thesis.

ABSTRACT

The integration of control and data planes into the same devices and lack the global centralization control that made the traditional networks may not meet the requirements of the emerging cloud computing, the tactile Internet, and the Internet of Things (IoT) technology. Moreover, the traditional networks cannot provide the complexity of control protocols, complex traffic engineering (TE) tasks, and interconnecting of a huge number of smart devices. Software Defined Networking (SDN) is an architecture that overcomes the above issues of the traditional networks by taking advantage of global centralization control, decouples of the control and data planes, and enabling innovation through the network programmability.

The shortest path-based routing cannot guarantee future traffic demands because the routing only uses the minimum hop counts. The application-aware routing is more efficient than the traditional shortest path-based routing; however, classification of application traffic and estimation of QoS parameters like link utilization and link delay are needed to perform such kind of routing. By taking the advantage of SDN, application-aware traffic engineering can perform more effectively in SDN environments.

This dissertation presents an application-aware traffic engineering (App-TE) in SDN which generally involves three main modules: traffic classification, traffic measurement, and traffic management. Application traffic flows classified into the following two classes: prioritized application traffic and non-prioritized application traffic by using port number and protocol number with the help of traffic analyzer (sFlow-RT). The classified traffic flows are fed to the traffic measurement module to calculate the link utilization, link delay, and Delay Weighted Capacity (DWC) values. Finally, prioritized application traffic flows are routed by using the DWC-aware routing and non-prioritized application traffic flows are routed by using shortest path routing (or) minimum hop-count based routing. The experimental results demonstrated that the average throughput results of the proposed App-TE outperformed the shortest path routing and LU-aware routing.

TABLES OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

CHAPTER 1

INTRODUCTION

The rapid growth of science and technology, computer networks have achieved a great impact and transformed the way of connectivity, entertainment, study, and social networks of human life. Short Message Service (SMS) over mobile networks, smart TVs connected with Internet provider as an on-demand media, e-Learning systems which permit everyone can gain knowledge and study from everywhere, and telemedicine are great examples of technical revolution. Effective network management solutions are required not only to harmonize the dramatic growth of transferring information and requirements of applications but also to provide the best services to users and improve network performance.

The legacy networks mostly implement in dedicated appliances and hardware that lead to limited innovations for both management and configuration aspects. The networks also create multi-vendor equipment. Therefore, network administrators need to have a wide knowledge of all devices from different vendors because different vendors have different syntaxes and commands. The concept of Software Defined Networks (SDN) is to organize and manage networks with software. SDN decouples control and data plane from networks through the control protocols such as OpenFlow [55]. SDN also serves vendor neutrality with the abstraction of network devices such as routers and switches, etc. Network administrators can easily manage the large-scale network with the software when SDN is employed into large-scale, carrier-grade networks. Therefore, SDN, an emerging architecture, has become a popular solution for companies to control networks using a more cost-effective software solution, rather than more expensive hardware methods.

A non-profit organization, Open Networking Foundation (ONF) develops the adoption of SDN through open standards development. In the years since its inception, SDN has obtained a lot of attention not only from academia but also from the Internet industry and SDN has evolved into many networking technologies offered by many vendors such as Cisco, Juniper, VMware, Pluribus, and Big Switch. Now, Google, Amazon, Facebook, Microsoft, and other companies have invested heavily in SDN not only for their data centers but also for their Wide Area Networks (WAN). Therefore, the market size of SDN has been around $6.6 billion in 2017, for 2018 the market size has reached $7.9 billion, and the global SDN market will expect to reach $20 billion by 2022 [92].

Although SDN has presented to enable network innovation, and to program new applications according to the user requirements, there have still problems to encounter in Traffic Engineering (TE), application-aware engineering, Quality of Service (QoS) based routing, and security. This dissertation focuses on the problems, motivations, methods, solutions, and results concerning traffic engineering with application-aware routing in SDN.

1.1 Motivations and Problems definition of the Research

Application-aware engineering finds the optimal route for each application traffic depending on the network resources requirements of each application such as bandwidth, network delay, and jitter. For instance, in cloud data-center networks, the equal treatment of all classes of applications is not the proper way to meet user's application-level requirements because the network resource requirements of applications are varied [16]. Traditional networks that are integrated control and data planes into the same devices and lack of global centralization control. Traditional networks cannot also satisfy the requirements of the tactile Internet and the Internet of things (IoT) technology.

Moreover, traditional networks cannot manage the complex traffic engineering, control protocols, and interconnecting of a huge number of smart devices [80]. In an SDN network, the complex route calculation and security are performed by the controller. SDN is an architecture that overcomes the issues of the traditional networks by taking the advantages of global centralization control, decouples of control and data planes, and enabling innovation through the network programmability [82]. Therefore, this dissertation considered the application-aware traffic engineering in SDN environment.

Traditional shortest path-based routing cannot guarantee future traffic demands because it only considers the minimum hop counts. The application-aware routing is more efficient than the traditional shortest path-based routing, however, the QoS parameters like link utilization, link delay, delay variation, and jitter are needed to perform such kind of routing. There are mainly two techniques in traffic measurement: active and passive techniques [61]. Active techniques estimated QoS parameters by sending probe packets into the network and evaluating how the network traversal affects the network status. This technique can cause a temporary network congestion. Passive techniques estimate QoS parameters by setting multiple measurement points in the network to monitor network statistics and this type of techniques are difficult to deploy in traditional networks.

To deploy passive measurement in SDN architecture, the SDN controllers polled the statistics information of the switches using the statistics request and reply of OpenFlow message. These request/reply processes and querying statistics information from all the switches in the network turns to increase the controller computational time and load. It also increases the latency between the controller and switches.

To perform application-aware traffic engineering in SDN, the classification of application traffic flows and selecting the optimal routes depending on the classified traffic are the necessary tasks like the network QoS parameters estimation. There are many techniques to classify application traffic flows such as port-based, DPI-based, and ML-based techniques. All the techniques have their perspective advantages and disadvantages; however, these techniques made the controller to perform some extra tasks.

Routing plays an important role in traffic engineering. Therefore, managing which application traffic flows route which path is a crucial task. Some application traffic need bandwidth-guaranteed paths but some need delay-guaranteed path. Application traffic flow should route the optimal path depending on their application level requirements.

To address the above issues, this dissertation proposes the application-aware traffic engineering (App-TE) in SDN environment in order to manage the best route for the application traffic flows based on their requirements. App-TE involves three main modules. The first one is to categorize application traffic flows into the following two classes: prioritized and non-prioritized traffic flows based on their requirements. Then, port and protocol number-based traffic classification is performed. As the second one, App-TE estimate the network QoS parameters by using passive technique. The third one is routing application traffic flows by using delay-weighted capacity (DWC) aware routing and minimum hop count-based routing.

1.2 Objectives of the Research

The main objective for this research is the implementation of application-aware traffic engineering in SDN environments and the others of this research area are as follows:

- To meet the network requirements of all classes of applications in SDN environment by considering of applications' QoS requirement such as available bandwidth or link utilization and link delay
- To utilize network resources and to improve overall network performance by steering the best path based on maximized delay weighted capacity values
- To reduce the controller's work load by estimating the link utilization with the help of sFlow-analyzer metrics via REST API, instead of using simultaneous polling of port statistics information for all the switches in the network
- To route the classified application traffic flows through the optimal routes by applying DWC-aware routing and minimum hop-count-based routing.

1.3 Contributions of the Research

The main contributions for this dissertation are as follows. A detailed analysis of application-aware traffic engineering in SDN is described with the previous works of traffic classification, traffic measurement, and traffic management methods.

For the traffic classification, to reduce the computational complexity of the proposed App-TE task, firstly, application traffic flows categorized into two classes: prioritized application traffic and non-prioritized traffic flows. For effective traffic management, the mostly used application traffic in Internet such as video streaming, file transferring, and haptic streaming (for next generation networks, Tactile Internet) are defined as the prioritized application traffic. Others are defined as the non-prioritized application traffic flows. Then, App-TE performed port and protocol number-based traffic classification with the help of traffic analyzer (sFlow-RT).

In the traffic measurement module, App-TE used passive technique like statistics monitoring. App-TE estimated available bandwidth (ABW) and delay-weighted capacity values using port statistics information of OpenFlow messages. To reduce the controller's work load and avoid network congestion due to simultaneously querying statistics information all the switches in the network, App-TE collected port statistics information only from the selected switches according to the sFlow metrics (JSON information via the REST API).

For the effective traffic management, App-TE used two main routing such as DWC-aware routing and minimum hop-count-based routing (MHR). App-TE focused on the prioritized application traffic flows. Since both bandwidth-sensitive and delay-sensitive application traffic flows are included in prioritized application traffic classes, App-TE routed the prioritized application traffic through the optimal path which has maximum ABW and minimum delay. For non-prioritized application traffic, App-TE simply forwarded through the minimum hop-count path by using MHR.

1.4 Organization of the Research

This dissertation organizes with six chapters.

Chapter 1 includes an introduction, the motivations, problem statements, objectives, focuses and contributions of the research work. The remainder of this thesis is organized as follows.

Chapter 2 summarizes some of the most significant previous works relating to traffic engineering in IP-based and MPLS-based networks to highlight the advantages of traffic engineering in SDN. Chapter 2 also surveys the various application-aware TE methods of SDN by describing the main three components: traffic classifications, traffic measurements, and traffic management.

Chapter 3 provides a background theory of SDN including data plane layer with OpenFlow switches, the control plane layer with ONOS controller, northbound APIs with the concept of OpenFlow protocols, southbound APIs, and the application layer with various routing application methods.

Chapter 4 presents the architecture of the proposed application-aware traffic engineering (App-TE) method. The detailed process of traffic classification, traffic measurement, and traffic management modules are described in chapter 4.

Chapter 5 illustrates the design and implementation of the experimental testbed. The chapter also discusses the experimental results of the proposed system compared with other two methods.

Finally, chapter 6 of the thesis provides a conclusion, summary of the dissertation, advantages and limitations. Chapter 6 also describes the directions for the future work.

CHAPTER 2

LITERATURE REVIEW

Traditional shortest path routing or minimum hop count-based routing applied shortest path to route packets from source to destination hosts, even this shortest route did not satisfy the traffic demand or not the optimal one. Traffic Engineering (TE) permits the network providers to omit the shortest path-based routing by using the longer path but a less congested path and satisfy the traffic demand. TE helps network providers to improve network resource utilization and offers more services to the end-user [3]. Typically, the objectives of TE include load balancing, alleviate network congestion, minimize the bandwidth consumption in the network, optimize routing, fault tolerance, network resiliency, and Quality of Service (QoS) guarantees.

2.1 The Issues of Traffic Engineering in Legacy Networks

In general, traditional TE technology is mainly classified into the following two types: IP-based TE and MPLS-based TE [78]. Therefore, this section describes the issues of TE in legacy networks related with IP-based and MPLS-based networks.

2.1.1 IP-based Traffic Engineering

Routing plays an important role in traffic engineering. IP-based TE generally involves optimizing IP routing algorithm, solving the problems of multipath traffic load balancing and alleviating network congestion [35]. Fortz et.al [29] proposed a routing algorithm which used to adjust the link weights of Open Shortest Path First (OSPF) and the proposed routing finally got multiple shortest paths to provide load balancing of the traffic. IP-based TE technology has two clear weaknesses: first, when OSPF link weights are used to control network routing, and traffic cannot be split in an arbitrary proportion that lead to reduce the utilization of network resources.

Second, when connections lost or network topology changes have occurred, the OSPF protocol will take some time to converge a new network topology, which may lead to packet losses, delay, network congestion, and even routing loops. Besides, a variety of new multimedia applications in today networks not only require bandwidth guarantees but also need other QoS guarantees such as packet loss, jitter, end-to-end delay, and energy efficiency. Therefore, QoS guarantees and network resilience schemes also considered as important factors of IP-based TE. Efficient resilience schemes needed to deal with different types of network failures such as network node and link failure [40]. In this situation, the solutions for IP-based TE also need to consider how to minimize the impact of failures on network performance and resources utilization.

Most of IP-based TE solutions [10, 30] proposed a routing method which is based on the shortest path and load balancing schemes with equally split traffic into equal cost multiple paths (ECMP). The fundamental concept of shortest path routing is to set the link weights of interior gateway protocols (IGPs) concerning the network topology and traffic demand to manage intra-domain traffic [22]. Large-scale IP networks usually use IGPs such as Open Shortest Path First (OSPF) or Intermediate System-Intermediate System (IS-IS) which select paths by taking account into static link weights (cost value assigned at each link).

In ECMP, large networks are typically divided into multiple OSPF or IS-IS areas [37]. In some cases, the network may have multiple equal-cost shortest paths between the same pairs of source and destination. The specifications of the OSPF and IS-IS protocols do not dictate how routers manage the presence of multiple shortest paths because the IGP routing algorithm used the static link that did not has the flexibility to divide the traffic between the shortest paths in arbitrary proportions. Therefore, routing based on link weight is not enough to depict all possible solutions to the routing problem. In practice, traffic volumes fluctuate over time due to the dynamic traffic requirements, and unexpected failures can lead to the changes in the network topology. Moreover, obtaining an exact traffic matrix estimation may be hard.

The practical OSPF [70] offers shortest-path-first routing with ECMP to get simple load balancing and ECMP allows traffic uniformly divided between equal-cost paths. Based on the Hash function, ECMP aims to divide the hash space into equal-size partitions that correspond to the outbound paths and then forward packets based on their endpoint information along the path whose boundaries envelop the hash value of the packets. Although the ECMP routing provides better performance with static load balancing, they are inappropriate for dynamic load balancing protocols [51] because the static flow mapping to paths does not consider either the current network utilization or flow size, which may lead to increase network congestion and degrade the overall network utilization [34].

Due to their data plane, control plane, and management plane split and distributed across the different network elements, today's IP networks are far more complex and harder to manage [14, 13]. To encounter the above problems, one of the solutions is to separate the routing decision logic from the protocols between the network elements [14]. This solution includes the decision plane for a network-wide view of the network, the data plane for forwarding traffic, the discovery and dissemination planes for direct control. The other solution [13] introduced Routing Control Platform (RCP), which is a logically centralized platform that separates the IP forwarding plane to bring the scalability to avoid the complexity in the internal Border Gateway Protocol (BGP) architecture. These ideas motivate the SDN researchers and system developers to logically separate the controller network from OpenFlow-enabled switches.

2.1.2 MPLS-based Traffic Engineering

To avoid the issues of IP-based TE, researchers proposed another solution, so-called Multi-Protocol Label Switching (MPLS) which forwarded the network packets by using MPLS labels instead of IP headers [8]. The authors [8, 9] introduced MPLS as an efficient solution to address the constraints of IP networks. In MPLS-based TE, the routing uses the MPLS label switching mechanism where labels assigned and distributed between routers using Label Distribution Protocol (LDP). When a packet enters the network, LDP assigned with the label by the ingress router and then forwarded across the network through a pre-established path called the Label Switched Path (LSP). Finally, the label removed at the egress router and forwarded as the IP packet. MPLS-based TE is used LSP tunnels by a signaling protocol such as the Resource Reservation Protocol (RSVP) which used for Differentiated Services (DiffServ).

Multiple LSP tunnels can be generated between two nodes which allocate the network resources. The traffic between the nodes are spilled among the tunnels by some local policy. The total number of LSPs in an intra-domain network be $O(N^2)$, where N be the number of egress and ingress routers in a single domain [10], which may be considered as non-scalable concerning network protocols [83]. Therefore, scalability and robustness has become problems in MPLS-based TE [1] as aggregate traffic are provided through the dedicated LSPs. If there any link failure occurs in active LSPs, the MPLS-based TE need to consider path protection mechanism such as backup paths, as otherwise traffic cannot be forwarded through the alternative paths. Moreover, network management over MPLS is an important factor in traffic engineering. Eventually, the success of the MPLS-based TE depends on how easy it observes and controls over the network.

However, MPLS have the excessively complex protocols mechanism that can lead to a high-performance overhead and difficult to satisfy the requirements of network traffic demands, network utilization, and energy saving in data centers networks. The simplicity of the SDN can mitigate the complexities of the MPLS control plane with scalability and efficiency at the same time [75]. The OpenFlow extension with MPLS provides much easier and more efficient in network management. The solutions [75,52] simply match and process the MPLS flows with OpenFlow extension and these solutions did not require the MPLS per packet processing operations. Therefore, an emerging architecture, software defined networking (SDN) has become a popular solution for companies to control their networks using a more cost-effective software solution, rather than with more expensive hardware methods.

2.2 Traffic Engineering in SDN

Although traffic engineering methods have widely exploited in the past and current traditional data networks, such as IP and MPLS networks, SDN still needs new traffic engineering methods to adapt its decoupling of control and data plane layer architecture [2]. SDN allows decoupling control functions from data plane devices as well as provides global centralization views including network statistics information. Network programmability can perform at the data plane devices by SDN controller with the help of OpenFlow protocol. Therefore, network operators can easily reprogram and dynamically manage through the network programming interface.

OpenFlow switches can support flow management more effective and efficient because of their multiple flow table pipelines. By taking these advantages of SDN, many researchers pay attention to the following four main sections of TE in SDN. These are flow management, fault tolerance, topology update, and traffic analysis or characterization. Figure 2.1 presents four main sections of TE in SDN environment. When a first new flow enters the OpenFlow-enabled switches and the flow does not match any rules in the switch's flow table, the switch encapsulates and forwards the flow to the controller. The controller computes a forwarding path and installs the appropriate flow entries to the switch. When a high number of new flows enter the switch, there may be substantially overloaded on both control and data planes. Therefore, reducing communication latency and balancing the workload between the controller and the OpenFlow-enabled devices becomes important factors. The authors in [14, 25] considered the flow management solutions for switch level, controller level, and multiple flow tables usage to a trade-off between the load balance and latency.

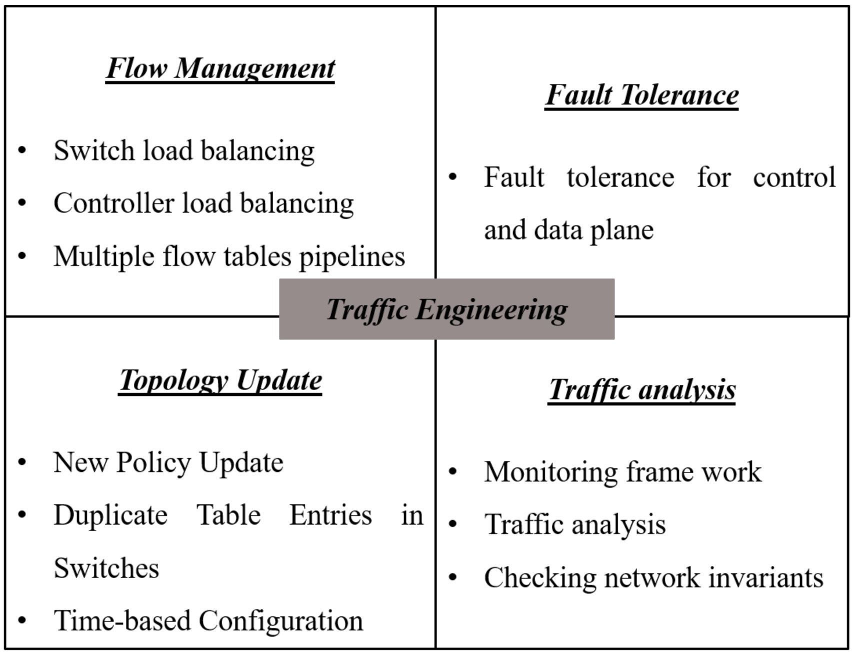

Flow Management
• Switch load balancing
• Controller load balancing
• Multiple flow tables pipelines
Fault Tolerance
• Fault tolerance for control and data plane
Traffic Engineering
Topology Update
• New Policy Update
• Duplicate Table Entries in Switches
• Time-based Configuration
Traffic analysis
• Monitoring frame work
• Traffic analysis
• Checking network invariants

Figure 2.1 The Scope of Traffic Engineering Approaches in Current SDN

Hash table based ECMP solution [37] which separated flows through the available paths by using the flow hashing mechanism. Flow hashing mechanism forwards a path over multiple candidate paths depending on the hash value of selected fields of the packet's headers modulo with the number of paths and then split the load to each path. To address the long live flow collision problems of ECMP, Hedera [25] is one of the solutions that scalable and dynamic flow scheduling schemes to utilize the aggregated network resources efficiently. By taking advantage of the global view of SDN, Hedera gathers flow statistics information from switches, calculates available paths for flows, and steers switches to reroute traffic. Hedera conducted with 8,192 hosts data center and their experiment results outperform the static load balancing methods.

One of the distributed control planes for OpenFlow enabled switches in SDN is a HyperFlow [85]. To address the scalability issue and reduce the flow setup time of control and data planes responds and requests, HyperFlow not physically distributed but logically centralized by applying global network-wide views. HyperFlow is also an application which implemented in Network Operating System (NOS) called NOX with minor modifications.

To gain network reliability, SDN network infrastructure (controllers, switches, and links) should have an ability to support failure recovery [15]. The authors in [49, 74] proposed solutions for fault tolerance in the data plane layer and the authors in [76, 28] discussed solutions for fault tolerance in the control plane layer. When the link or switch failure is detected, CORONET [49] recovered failures during a sub-second time by applying for multipath support. Sgambelluri et.al [74] mentioned the data plane protection path solution which used pre-computed paths as protection paths and installed together with the normal working path into the switch's flow table. When the failure is detected, the switch will be used the protection path. Sharma et al. [76] proposed a control plane restoration mechanism, in which the controller computed alterative paths, after detection of failure. Then, the controller updated and installed packet forwarding rules urgently without caring whether the old rules are expired or not.

For the centralized nature of SDN architecture, the reliability of the controller is important. Configuring backup controllers with the help of OpenFlow protocol but OpenFlow does not have coordination schemes between the backup and primary controllers. CPRecovery [28] provided a primary-backup mechanism that focused on the replication process between the switch component running on the primary and secondary controllers. The switch can check whether the controller is active or not by sending an inactivity probe to the controller. If the controller is down, the switch may not receive a reply within the waiting time. Then, the CPRecovery component started searching for the active backup controller.

The centralized SDN controller needs to manage dynamically global network policies and rules which instruct OpenFlow-enabled data plane devices through the network. While updating policies and rules, the affected flows may be delayed or dropped, which can cause network performance degradations. The general topology update operation implemented as follows: whenever updating to new policies from old policies over multiple switches, each individual packet or flow follows the new or old policies, not both. By considering per-packet consistency (each packet follows through a single network configuration) and per-flow consistency (all packets in the same flow will be followed the same policies), the authors in [67, 68] solved the problems of topology updating in SDNs environment. The authors in [19, 86] described the traffic analysis solutions to acquire timely statistics on network resources at different aggregation levels (such as flow, packet and port). PayLess [19] and OpenTM [86] are query-based monitoring techniques for SDN. PayLess supported a flexible RESTful API for flow statistics collection at different aggregation levels and OpenTM periodically polled the statistics of the switches by tracking all the active flows in the network.

Traffic managing is a crucial task of TE in SDN. Although the traditional Shortest Path First (SPF) algorithm routes the traffic efficiently, however, the congestion may occur. SPF also produces the bottleneck for future traffic demands [48]. SPF only takes account of the minimum hop-count and does not achieve QoS-aware TE and load balancing. Therefore, QoS-aware TE algorithms are still important for the future Internet. There are many TE algorithms which tried to solve the problem of setting up the bandwidth guarantee tunnels in networks [50, 90, 53].

For a wider range of Internet applications, the routing algorithm based on the delay and link utilization has become more important to fulfill the user requirements. A simple solution proposed in [90], where firstly they prune all the links with insufficient bandwidth then this solution chose a path with the smallest delay path. Among the bandwidth and delay constrained routing, the Maximum Delay Weighted Capacity Routing algorithm (MDWCRA) tries to minimize the interference between ingress and egress pairs [53]. It also calculates the shortest disjoint paths, defines critical links for bottleneck traffic, and avoids the links for future demands.

As the rapid growth of cloud computing and Internet of Things (IoT) technology, the traditional network architecture cannot handle the complexity of control protocols and the internetworking of a large number of smart devices [80]. In cloud data-center networks, the equal treatment of all classes of applications is not the proper way to meet user’s application-level requirements because the resource requirements of network applications are varied [16]. By taking the advantages of SDN, application-aware engineering effectively performs traffic engineering mechanism based on the network requirements of applications.

2.3 Application-aware Traffic Engineering

Application-aware engineering is a computation of path or route based on each application requirements such as network delay, link utilization or available bandwidth, and jitter. For instance, FTP traffic may better perform with higher bandwidth path and video traffic needs less delay path to maintain its best quality. Therefore, each application needs their own best path to meet their application level requirements. The application-aware engineering mainly contributes the following steps:

- Identify types of applications based on the QoS requirements and classify application by using different mechanisms.
- Estimate the QoS parameters such as link utilization or available bandwidth, link delay, and jitter by monitoring the switch ports statistics, and sending probe packets to the whole path to measure the end-to-end delay.
- Calculate the optimal routes and reroute the traffic through the optimal routes.

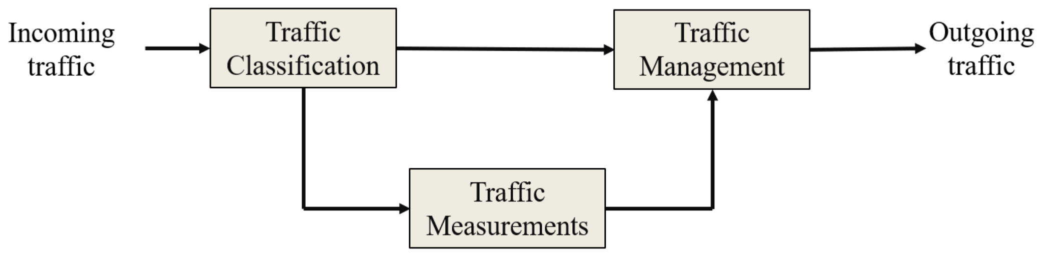


Figure 2. 2 The Overall Architecture of Application-aware Engineering

The following sections describe the above three steps which are related with the application-aware engineering under the name of traffic classification, traffic measurement, and traffic management. These related works include not only in SDN environments but also in SDN-based cloud computing, SDN-based IoT networks and datacenters environments. Figure 2.2 depicts the overall architecture of the application-aware engineering in software defined networking.

2.3.1 Traffic Classification

Traffic classification is an intelligent process that classifies traffic into different categories [96]. Nowadays, in the computer science field, traffic classification is an essential task for Internet Service Providers (ISPs) to recognize which types of application traffic through the network. Network traffic classification is also an important pre-calculation step for network measurement, network monitoring, network management, network security, and network design. By using network traffic classifications techniques, ISPs and network administrators can steer each type of network application traffic to route the optimal path through the network and can control the overall network performance. The following classification techniques are mainly used in traditional networks and even used in SDN environments [23].

2.3.1.1 Port-based traffic classification

In the earliest days of the Internet, most of the Internet protocols are assigned to well-known port numbers by Internet Assigned Number Authority (IANA) [96]. At that times, port-based classifications are an effective one for traffic classification tasks. It classifies the application traffic by extracting port numbers from the packet header and then checks it out with the registered port numbers in IANA. On the other hand, there are increasing numbers of protocols and applications that used dynamic port or random port numbers tried to hide from network security tools. Therefore, port-based techniques give less than 85% accuracy and it cannot recognize over 50% of traffic flows that it is investigated [64].

2.3.1.2 Payload-based traffic classification

To eliminate the limitations of port-based classification techniques, many researchers proposed payload-based classifiers [27]. Payload-based techniques are also called Deep Packet Inspection (DPI). DPI is also a form of packet filtering to identify, classify, reroute, and block packet by checking the content of packet beyond the packet header and matching payloads with the known signature of protocols [32]. DPI inspects the contents of packets at a specified checkpoint and controls the packets according to the rules given by enterprise networks, ISP, or network manager.

The authors [26,103] proposed SDN based DPI scheme set up in the network application layer, controller layer, and infrastructure layer respectively. DPI inspects the packets at the three layers and sends them back to rules and policies. In network application layer, DPI correspondence a network application analogy and it may delay the packet processing time [26]. DPI in SDN controller performs as the network services and sends network statistics information and intelligence to the network application layer via the northbound Application Programming Interface (API). When DPI is running in the network nodes, it simply applied the predefined policies [103].

L.E. Li et.al [56] proposed an idea to extend the OpenFlow protocol with eliminating and adding regular expression rules. In their work, the controller can control the rules tables of DPI and flow tables instead of passing through all packets to DPI. By using pattern matching algorithms and splay tree filtering of enhanced DPI and Intrusion Detection System (IDS), the authors applied network traffic statistics and aimed to optimize the ruled orders. They also demonstrated their algorithm through the string set of Snort [87]. Renukadevi et.al [69] analyzed the application signature data with DPI that placed on the north interface of SDN.

They can dynamically allow or block the provision flows and can enhance the bandwidth. SDN architecture decouples control plane from data plane completely, however, traffic monitoring and control in SDN only depend on network states, not the traffic behaviors. To improve fast packet forwarding and to guarantee QoS demands, they [57] proposed an application-aware traffic control scheme with DPI. The scheme collaborated both network states and traffic behaviors to perform packet classification and behaviors matching respectively. They also designed a publish/subscribe middleware as the exchanger of information between the application layer and network services (topology management and devices monitoring).

2.3.1.3 Machine learning-based traffic classification

Today, most of the internet traffic are encrypted and applications are used dynamically assigned port numbers, therefore, DPI and port-based classification face difficulty to classify the application traffic [77]. To overcome the above difficulties, many researchers have been proposed the Machine Learning (ML) based techniques, which used the statistical features of network traffic flows.

By using the extracted statistical information from network flows, they [4, 5, 31, 65, 23] used ML algorithms to identify the application by applying useful variables. Alshammari et.al [4] used five ML algorithms such as C4.5, RIPPER, Naive Bayes, AdaBoost, and DVM to address the encrypted traffic classification. They used SSH and Skype as the encrypted traffic and they also showed that C4.5 and RIPPER outperform the other three algorithms.

In another work [5], the authors mentioned three different ML algorithms which are C4.5, Genetic Programming, and AdaBoost to identify the encrypted VoIP traffic. Their experiment results proved that C4.5 got better performance results than the others. The authors of [31] identify real-time traffic in virtualized networks according to the

QoS classes by using Naive Bayes and other ML classification algorithms. Among them, Navies Bayes got 95% accuracy and gave the minimum classification time.

The authors in [65] identified application traffic flows in SDN with the help of four Neural Networks methods such as feedforward, Multilayer Perceptron, NARX (Levenberg-Marquardt) and Naïve Bayes. They used the collected data set that contains instant messaging, video streaming, FTP, HTTP and peer to peer protocols. The four ML classification gain accuracy of 95.6%, 97%. 97%, and 97.6%, respectively.

Dias et.al [23] proposed the Naive Bayes algorithm-based video traffic classification module to support QoS requirements. To increase the accuracy of traffic classification, their approach used the relaxation of the hypothesis of independence between the attributes of a class. They tested with three different traffic classes (one file download and two different video services) which assigned QoS priorities. The reason why they classified traffic classes is to get better network performance by giving an effective treatment for requested traffic.

Most of the researchers and network operators have used port-based, payload-based, and machine learning-based techniques not only to identify anomaly traffic for security perspective but also to give effective traffic treatments for different QoS demands. Therefore, traffic classifications techniques used as a preprocessing step for traffic measurement and management.

2.3.2 Traffic Measurement

Traffic measurement is a crucial task for TE. Traffic measurement in SDN covers the following tasks: monitoring networks, defining and measuring networks QoS parameters, analyzing and predicting traffic patterns. This section describes defining network QoS parameters and measuring techniques for those parameters in SDN. Network QoS parameters represent the current situation of networks.

For effective network management, a reasonable network parameter design is required. In SDN network measurement, there are generally three types of network parameters: network topology, traffic, and performance parameters. The network topology parameters are the number of network nodes, link bandwidth, and port statistics. In SDN, the controller detects and maintains the current topology information by using Link Layer Discovery Protocol (LLDP) that specifies in OpenFlow. Figure 2.3 depicts the link discovery process in SDN by using LLDP. The controller sends

Packet_out messages with LLDP to all switches in the network and the switch sends these messages to all switches connected it directly.

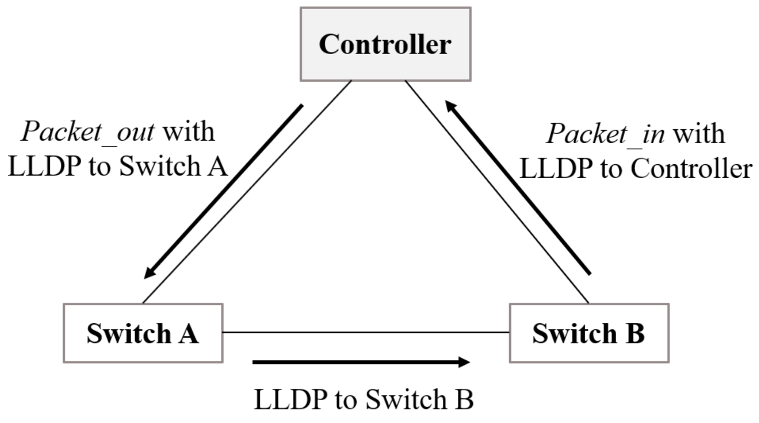

Controller
Packet_out with LLDP to Switch A
Packet_in with LLDP to Controller
Switch A
Switch B
LLDP to Switch B

Figure 2. 3 Link Discovery Process in SDN

When a switch receives an LLDP packet from other switches, it sends *Packet_in* with LLDP to the controller because of no relevant rules in switch flow tables. Then, the SDN controller knows which switches connected to and constructs the current network topology by applying these *Packet_in* message with LLDP. The number of packets that transmitted or received that passed through the network nodes or ports, the total number of packets count, and the total number of bytes count referred to as the network traffic parameters.

Current network status and network users' behaviors are detected and analyzed by applying these network traffic parameters. There are mainly two types of traffic in an SDN network: control traffic and data traffic. A data flow that transmitted between the SDN controller and switches refers to control traffic. A data flow that transmitted between switches called data traffic. The collecting of statistical information on each switch port (number of packets, number of transmitted packets sizes, number of received packet sizes) and traffic matrix are required to characterize flows. To characterized flows and measure the performance of the networks, the following factors are used as the network QoS parameters.

(1) Delay

The end-to-end delay means the requirement of the time when the destination host gets the packet sent by the source host. This network delay can be further separated into the following four parts through the Equation (2.1):

$$Delay_{end-to-end} = delay_{transmission} + delay_{processing} + delay_{propagation} + delay_{queuing} \quad (2.1)$$

Transmission delay

In the wired network, the transmission delay is the time taken that transmits all the packets into the wire [11]. The transmission delay of a link is directly proportional to the data rate of the link.

Processing delay

Processing delay is the time taken to process a packet at a network device. In high-speed routers, processing delays may be a microsecond or less [102].

Propagation delay

Propagation delay is the time taken to transmit the first bit from source to destination through the network and propagation delay depends on the propagation speed and distance.

Queuing delay

Queuing delay is the time taken to wait in a queue before it took for execution. It is also the time difference between when the arrival of the packet at the destination and when the packet was executed.

(2) Delay Variation (or) Jitter

The amount of delay variation in the end-to-end packet transmission is called jitter. Depend on the variation of delays, bits arrive either late or early at the destination. The bits may be overflowed in a buffer when they arrive too early and getting poor quality results when the bits arrive too late. Therefore, jitter is a special problematic factor in a real-time application such as live video streaming, video conferencing, and IP telephony [33].

(3) Packet Loss Ratio

The packet loss ratio represents the percentage of packet loss during the end-to-end packet transmission. Packet loss can cause because the congestion occurs in the network or when the packet buffer overflow occurs at the network devices [6]. Different applications have different levels to tolerate the packet loss ratio. Video traffic is mostly sensitive to packet loss ratio.

(4) Available Bandwidth (or) Link Utilization

Bandwidth is a continuous resource value and it is also the data transfer rate for a fixed period [42]. Available bandwidth (ABW) is an important dynamic characteristic of a network path, being equivalent to the amount of traffic that can be added to the path without affecting the other flows that traverse part of it, and independently from their bandwidth-sharing properties. For instance: a link with XX Mbps maximum capacity, but YY Mbps is already used. The available bandwidth for a link is obtained through the Equation (2.2),

$$Available\ Bandwidth = XX - YY \tag{2.2}$$

In QoS parameter measurement, there are generally classified into two types: active and passive techniques. Active measurements deployed between two points in the network, and the injected traffic attempts to bring to the surface the unidirectional or bidirectional performance properties of end-to-end paths. These techniques usually implemented within an active measurement infrastructure framework and offered the flexibility of running at commodity hardware/software end-hosts at different Internet sites. Active techniques send probe packets into the network and evaluate how the network traversal affects the network status [71]. This technique can cause temporary network congestion.

Passive techniques estimate QoS parameters by setting multiple measurement points in the network to monitor network statistics. These passive techniques are too complex to deploy in traditional networks [42]. By taking the advantages of SDN’s global centralized control, there are some papers to estimate ABW by using passive techniques. Megyesi et al. [60, 61] proposed the ABW estimation in SDN by using OpenFlow messages to track the bandwidth utilization of every link in the network and calculated the ABW on each path in the network based on the statistics information.

In [60], they used the FloodLight controller to estimate end-to-end ABW and they also explained a proper trade-off is required between accuracy, polling rate, and network delay constraints. In [61], they focused on the source of errors for the estimation of ABW measurement and highlighted that these errors are due to the lack of a local timestamping mechanism in OpenFlow. Singh et al. [79] estimated end-to-end ABW on any given path not only by composing link-wise ABW but also validating with a bandwidth measurement tool called Yaz [81]. The ABW measurement is worked well by taking the traffic statistics from the SDN controller [58, 92, 79], but the controller keeps querying statistics from all the switches in the network that may lead to overload network traffic. They [79] explored many algorithms for selecting which OpenFlow-enabled switches to query and they also mentioned there is a trade-off between the querying on every switches and measurement accuracy.

```
Algorithm        : Application-aware routing
Procedure        ABW_Calculation
begin:
1.  Input: srcIP,dstIP,srcPort,dstPort from sFlow-RT
2.  Output        : max_ABW_path
//classify application by validating its port with well-known port (e.g, FTP=20, RTP=5004)
3.    if ( srcPort ==20 OR dstPort==20) then
4.        print(That is FTP traffic and calculate ABW)
//get available paths of given source and destination IP by using ONOS API
5.        available_paths = getAllPaths (srcIp,dstIp)
//get available links along the path
6.        for path in available_paths
7.            available_links = getAllLinks (path)
//collect statistics for each link, calculate ABW by using statistics and select maximum ABW
8.            for link in available_links
9.                collect statistics for each link
10.               calculate ABW for each link
11.               calculate max_ABW_path
12.           end for
13.       end for
14.   else
15.       print(perform default routing)
16.  end if
17. end
```

Figure 2. 4 ABW Calculation Algorithm for Application-Aware Routing

To address these issues, they proposed a solution [88] that used OpenFlow statistics to estimate the end-to-end ABW and reduced excessive network traffic by querying only the selected switches statistics that are provided by the sFlow-RT analyzer. The main tasks of their work are as follows: (i) port-based application traffic classification is performed by using sFlow-RT analyzer, (ii) dynamically collects ports statistics of source and destination switches according to the JSON information that is sent by sFlow-RT, and (iii) reroutes to the best available path based on calculated end-to-end ABW.

Figure 2.4 shows the algorithm for ABW calculation of their work as an example of FTP traffic. This algorithm is written with Java in the application layer of the ONOS controller. After calculating the best path, the new flow entries are added to respective devices along with the path by using *FlowRuleService* which is supported by the ONOS controller. Their experimental results demonstrated that the total throughput of their method outperformed the reactive forwarding (i.e. ONOS's forwarding application) when the traffic volume is larger than the link capacity and their method also reduced packet loss than reactive forwarding.

End-to-end delay is also one of the important networks QoS parameters like ABW. End-to-end delay takes a vital part for sending and receiving of data between the end to end devices. To perform efficient network traffic forwarding, needed to select the minimum end-to-end delay paths. Therefore, many works [41, 36, 20] focused on defining delay models, minimizing delay, and estimating end-to-end delay techniques.

Although the OpenFlow protocol can effectively manage implementation and configuration changes of several networks such as core and data center networks, the control plane instructions must reach data plane elements in a timely manner [41]. The delay may increase according to the increasement of propagation delay between control and data plane, the time increasement for finding matching flow table entries and update flow entries, and the execution speed of the controller.

The authors in [36] proposed querying theory-based delay model by assuming packets are arriving as Poisson distribution but ethernet traffic is not accurately modeled as a Poisson distribution process. Ciucu et.al [20] measured latency concerning with execution and generation of control messages in SDN hardware switches. They also discussed the effect of rule position number in the OpenFlow table and the insertion delay. The authors in [44] proposed a delay model used by network calculus which is also a new alternative approach of querying theory. Their network calculus only supported worst-case bounds on performance metrics analysis and a little hard for practical usage.

Their approaches [41, 12] have some unrealistic assumptions for practical analysis. Iqbal et.al [41] developed end-to-end delay measurement by using a stochastic model and they experimented this delay measurement by using the following three different platforms such as Mininet network simulation environment, GENI, and OF@TEIN testbeds for real traffic scenarios. Their proposed model, a log-normal mixture model for end-to-end delay in SDN fitted to the empirical measurements. They also proved that an M/G/1 model with a log-normal mixture model estimate end-to-end delay in OpenFlow-enabled networks more accurately than the others.

Traditional ‘PING’ or Internet Control Message Protocol (ICMP) is a basic approach to examine the delay within the networks. The authors in [91, 18] estimated end-to-end delay by applying ICMP nature. The authors in [91] presented delay-aware traffic rerouting method in SDN by conducting ONOS controller and Mininet emulator. They estimate the end-to-end delay by sending ICMP probe packets from source switch to the controller through the destination switch.

The authors in [19] proposed a time-stamp based shortest path selection framework for end-to-end applications. This framework measured end-to-end delay by using ICMP nature and probing approach. To overcome the issues of ICMP and packet probing, they applied time-stamp recording which records the arrival and departure time of per-packet flow at Open vSwitch and calculated delay based on the recorded sending and receiving time.

After defining and measuring network QoS parameters, the next important task is the managing traffic depends on the QoS requirements.

2.3.3 Traffic Management

Traffic management is the key player in the TE task. Traffic management is responsible for steering traffic based on the QoS requirements and performs an optimal rerouting scheme to meet the application-level requirements. The following works have applied QoS aware routing schemes and application-aware routing management schemes in SDN environments and SDN-based cloud, IoT and datacenters environments.

Deng et.al [21] mentioned AQRA as one of the Application-aware QoS Routing Algorithms that guaranteed multiple QoS requirements of high-priority IoT applications and selected the better routing paths by adapting the current network status. First, each of the IoT applications needed to send their app_profile to AQRA. App-profile contained IP address of an IoT application server and QoS classifier. When communication is started, IoT application sends *Packet_In* message to the controller’s

AQRA. AQRA started to classify flow into different priorities classes: high-priority application (non-real-time critical mission and delay-sensitive), medium-priority application (real-time services with a stringent delay bound), and low-priority application (no critical mission and no stringent delay bound) according to app-profile that supported from IoT application providers. Then, AQRA searched routed by applying SA (Simulated Annealing) based algorithm and installed flow entries by *Flow_Mod* messages into the network and edge layer. AQRA got better performance results than MINA in terms of delay, jitter, and packet loss rate.

App-RS [16] proposed as one of the solutions for application-aware routing scheme for SDN-based cloud datacenters. First, App-RS classified applications according to the following assumptions: class 1 for real-time application depend on end-to-end delay and link load to alleviate network congestion, class 2 for streaming application based on delay variation and link load to get smooth playback, class 3 application for miscellaneous application took into account link load to reduce packet loss rate. Second, App-RS identified applications by using application ID that store in the options of IPv4 header as a 24-bit label. Third, they determined the routes by using the LARAC (Lagrange Relaxation based Aggregated Cost) routing algorithm as well as considering of minimum link load and delay for class 1 and 2 applications. For the class 3 application, App-RS used the Dijkstra algorithm to find the least congested path. And then, App-RS added a determined flow entry to each switch along the path. By using the FloodLight controller and Estinet emulator, the simulation results of App-RS outperformed CORouting [63] related to the average bandwidth ratio, end-to-end delay, and delay variation. However, App-RS needed to consider the flow aggregation approach to overcome the limited flow tables size of switch’s Ternary Content Accessible Memory (TCAM).

The authors [54] proposed one of the solutions for application-aware bandwidth allocation mechanism in data centers networks. When predefined user requirement is available, *FlowSch*, their first approach can allocate available bandwidth and prioritize the user demands as per flows. However, when the application requires multiple flows to complete their task, *FlowSch* cannot provide such task. To accomplish this work, they then proposed *AppSch* that can allocate available bandwidth to satisfy the application requirements. Their evaluation results showed that when the total demanded bandwidth is close enough, *FlowSch* has been improved the average throughput and increased link utilization. When multiple flows or aggregated flows are required, *AppSh* has been improved link utilization more efficiently and decreased application completion time.

Jarschel et al. [45] proposed a DPI-based application-aware path selection method for YouTube video streaming. Firstly, they defined the threshold for a YouTube streaming application and then monitored buffered playtime and stalling. When currently buffered playtime reaches below the threshold, their method calculated the least load path and reroute the flow through the least congested path. Their method only applied YouTube streaming and required an addition machine to continuously monitor the buffered playtime.

Jeong et al. [46] performed application-aware TE by using the port number and DPI-based traffic classifier to identify application or service flows and distributed the identified flows to multiple queues with different priorities in each switch port. The multiple queues with different priorities forwarded by different treatments according to the maximum bandwidth and assigned priority of the queue. Matching with identified flows and its queue priorities defined by a network admin. This TE searched a routing path for identified flow by considering the current capacity of each queue in the egress ports. Their evaluation results demonstrated that the initial flow treatment delay for DPI of the first flow rule increases but decreases the propagation delays in the congestion scenario. Moreover, the results of their experiments concluded the identified application traffic got increased throughput and reduced packet delay.

Schweissguth et.al [73] mentioned application-aware Industrial Ethernet (IE) based extended TDMA (Time Division Multiple Access) approach which configured both routing and scheduling algorithms that took application requirements into account. They aimed to overcome the drawbacks of the original heuristic algorithm in switched networks by applying a TDMA approach. Cheng et.al [17] proposed application-aware routing big data processing scheme for Hadoop to accelerate its MapReduce data shuffling over a network. They applied the Floodlight controller and their approach outperformed the ECMP-RR and Spanning Tree schemes.

Although the application identification is beyond the knowledge of the combination of port number and protocol type, AMPF [66] used Machine Learning (ML) techniques (C4.5 decision tree) to perform an application-aware multipath packet forwarding for SDN. When the first packet of the flows entered the switch, if the flow rules existed, the switch forwarded packets by the flow rule. If the flow rules did not exist, the switch informed AMPF about the packet, and AMPF sent the received packet of collected feature vectors to ML classifier, then AMPF computed the assigned route to flows according to their priority of class they belong to. AMPF achieved the awareness of application in SDN by using ML techniques instead of using DPI.

Jeong et.al [47] mentioned an integrated DPI with an application-aware traffic management method in the SDN controller. They analyzed application traffic using off-platform DPI instances and sent the classified result to the controller to determine the corresponding flow rules for the incoming application traffic. They further applied Firewall and Bandwidth Manager application on their traffic management application to specify a list of application that may forward or block, and limit the rate of bandwidth. To the performance evaluation, they used the ONOS controller to implement the above traffic management and used FTP as a tested application.

OpenQoS [24] dynamically rerouted the QoS flows such as video streaming application which consisted of a base layer and one or more enhancement layers. OpenQoS was a per-flow based traffic prioritization scheme based on different layers. They categorized QoS flows as two levels: level-I QoS flows are used to send base layer packets, and enhancement layers packet are transmitted as level-II QoS flows. OpenQoS applied only video streaming applications and considered level-I QoS flows are a higher priority than level-II QoS flows. Packets information of each flow needed to periodically collect in OpenQoS. When there are large number of flows needed to collect, OpenQoS may be overtired.

Momin et al. [63] proposed Content Oriented Routing (CORouting). CORouting dealt with all types of application traffic and classified these applications into the following three main classes by taking account of their tolerance of packet loss and delay. Then, different routing methods are applied to handle each class of application. CORouting controlled real time application by Dijkstra's routing algorithm to find the minimum hop count path. CORouting managed streaming and miscellaneous applications by applying weighted Dijkstra’s algorithm to select the least congested path. Network congestion may happen as well as the number of real-time applications may increase because CORouting always selects the shortest path for real-time applications.

Mekky et.al [62] considered application-aware processing in the SDN data plane which aimed to support fast packet handling without restricted to Level 2 to Level 4 information. They kept some application logic at the physical switches instead of limiting application logic at the controller. They installed application-specific packet processing actions at the switches tables that are similar the OpenFlow's flow table. If the new flow that does not match with switch's flow rules in the flow tables, the controller decides the forwarding rules for this new incoming flow. Their results proved that their approach has low overhead and good performance results.

Table 2. 1 Application-Ware Traffic Engineering Schemes in SDN

Approaches	QoS Constraints	Application Classification	Traffic Management Methodology
AQRA [21]	Delay, jitter, packet loss rate.	High-priority, medium-priority and low-priority applications.	SA-based routing with adaptive weights.
App-RS [16].	Link delay, link load, delay variation.	class 1: real-time application, class 2: streaming application, class 3: miscellaneous application	LARAC routing algorithm for class 1 and class 2 applications. Dijkstra algorithm for class 3 application.
Application-aware BW allocation [54]	Link utilization.	Not categorized applications.	*FlowSh* for single flow required application and *AppSh* for multiple flow required application.
Application-aware path selection [45]	Bandwidth	Only applying to YouTube traffic.	Based on buffered playtime.
Application-aware TE in SDN [46]	Link utilization and delay.	Classify application with DPI and assign different priorities and queues.	Frwarded through the predefined priorities queues according to their requirements.
IE based Application-aware [73]	Bandwidth, latency	Not categorized application.	Enhanced TDMA approach with scheduling and routing.

AMPF [66]	Delay, minimum bandwidth.	Class 1: Skype, Class 2: YouTube, Google Docs, Class 3: Gmail, Facebook, Class 4: Dropbox, FileZilla	Prioritized flow and identified application by using MLT. Routed the application based on flows priorities.
Application-aware traffic management with ONOS [47]	Bandwidth	Not categorized application. Only tested with FTP. Classified by DPI.	Feed the classified results into the controller and determines whether it may forward or block with the help of Firewall application.
Open QoS [24]	Bandwidth	video streaming. Level-I QoS for base layer. Level-II QoS for enhancement layer.	LARAC routing algorithm.
CORouting [63]	Packet loss, delay, delay variation.	Real-time applications. Streaming and miscellaneous applications.	Rerouted real-time application with Dijkstra algorithm, streaming and miscellaneous applications with extend-ed Dijkstra algorithm

2.4 Summary of the Chapter

In today's networks, the equal treatment of all classes of applications is not the proper way to meet user's application-level requirements because the resource requirements of network applications are varied. Application-aware routing means such a kind of 'routing' that takes account into the application requirements such as available bandwidth, delay, jitter, and so on. Traditional shortest path routing cannot provide to satisfy the requirements of applications. The traditional IP-based TE and MPLS-based TE also struggle to perform this complex application-aware engineering task. By taking the advantages of SDN's decoupling of control and data plane, global centralized control, and enabling innovation through the network programmability, application-aware engineering can support more efficiently and effectively than the traditional networks.

In SDN, SDN-based cloud, IoT, and data center networks, various methods and approaches for application-aware engineering have been proposed in literature review. Port-based approaches, DPI or payload-based approach, machine learning-based approaches are used to classify application traffic. As the network QoS parameters measurements, estimation techniques for available bandwidth, end-to-end delay, and link weight parameters have been proposed so far. Table 2.1 surveys the various the application-aware TE in SDN. According to the Table 2.1 and literature reviews of this chapter, the efficient application-aware engineering techniques are still required to satisfy the user application level requirement.

CHAPTER 3

THEORETICAL BACKGROUND

Since the main goal of the dissertation is to design the application-aware traffic engineering in Software Defined Networking (SDN), this chapter describes the several related background fields. Firstly, this chapter explains the theoretical background of SDN by describing each layer of SDN architecture. As OpenFlow is one of the main building blocks of SDN, this chapter also describes the structure and functions of one of the OpenFlow switches such as Open vSwitch and OpenFlow protocol. Finally, this chapter presents a short overview of the traffic forwarding and QoS routing methods which have been used in conventional IP and SDN networks.

3.1 Software Defined Networks Architecture

The Open Networking Foundation (ONF), a non-profit organization that is funded by many companies such as Deutsche Telekom, Google, Microsoft, Facebook, Verizon, and Yahoo, focus on the development of SDN and standardizing the OpenFlow protocol aims to promote networking [104]. Software-Defined Networking has concerned a great deal of attention from enterprises, service providers, and industry associations. An emerging architecture, SDN is also an ideal solution for high bandwidth and dynamic nature of today's application because SDN is an adaptive, cost-effective, dynamic, and manageable architecture.

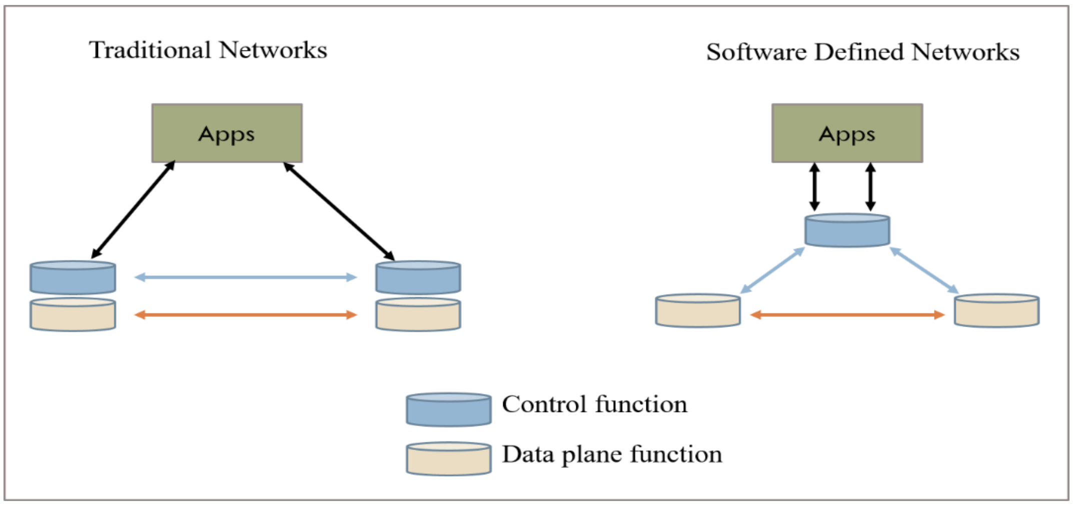


Figure 3. 1 Traditional Network Architecture VS SDN Architecture

SDN architecture decouples control and data function from the network devices such as switches and routers. This architecture also has global centralization control and enabling innovation through the network programmability. In contrast, in most large enterprise networks, network devices are coupled with control and data functions, which may face difficulties for network operators to adjust network infrastructure and configure large numbers of end devices, virtual machines, and virtual networks [58].

The difference between the traditional internets and SDN architecture is presented in Figure 3.1. This figure shows clearly how the data plane layer (network devices) is simplified into simple forwarding elements and the control layer(controller) is logically managed. The data plane layer comprised of network devices (programmable switches) which can either be implemented in hardware or software and these switches also support the OpenFlow protocol for communication and configuration with the controller. The following are some of the advantages of the decoupling of control and data plane function architecture, SDN:

- **Centralized Provision:** In traditional networks, the network administrator needs to manage each device individually and difficult to monitor lots of disparate systems. In SDN networks, network administrators do not need to update or configure each device manually because of the centralized approach of network management. The network administrator can also manage the entire network as a single unit.
- **Reduced operating costs:** SDN reduces operating costs by eliminating the requirements for configuration updates from network administrators and reduces hardware expenses by using virtualized control planes for each unique device.
- **Scalability**: SDN gives the user more scalability because of centralized provisioning but the SDN controller can practically manage a limited number of devices. Therefore, a practically large-scale network may need to deploy multiple SDN controllers.
- **Security**: The movements toward virtualization technology have challenged the network administrators to secure their networks. SDN controller supports a centralized location, therefore, network administrators can manage the entire security of the network.
- **Directly programmable**: Network managers can directly program network

operation with abstracts control of forwarding elements and dynamically adjust network traffic flows when the changing is needed. Therefore, network administrators can manage, configure, secure, and optimize network resources by using their application which was written based on their requirements and the application do not rely on proprietary software.

- **Openness**: There is no depended vendors because every data plane element (i.e., OpenFlow-enabled switches or routers) has a unified data plane programming interface for the OpenFlow controller to collect network status.

The SDN architecture mainly consists of the following three layers: the application layer, control layer, and data plane layer as shown in Figure 3.2.

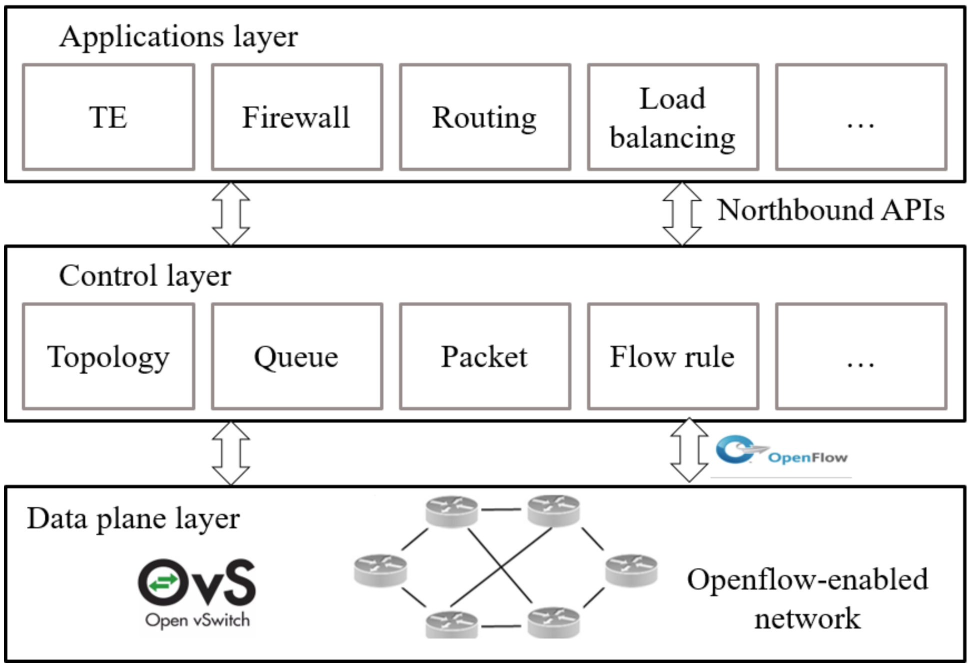


Figure 3. 2 Software Defined Networks Architecture

The SDN applications are programmed to support all kinds of network services such as traffic engineering, load balancing, firewall, routing, and monitoring. The control layer is a core layer of the SDN architecture that extracts the data plane layer information and communicates to the application layer with an abstract view of the network topology, including statistics and events.

The application and control layers communicate by using northbound APIs. The data plane layer consists of network nodes which can forward and processing of the data path. Communications between the data plane and control layers use a standardized protocol called OpenFlow. The SDN Controller defines the data flows that take place in the SDN Data Plane. When the flow is entered to the network, the flow must first take permission from the controller [104]. The controller decides whether the communication is permissible or not according to the network policy. If the flow is permitted, the controller decides an appropriate route for the permitted flow and adds flow entry for the permitted flow in each switch along the path. The SDN controller is responsible for these complex tasks and switches simply manage flow tables and focus on forwarding function.

3.2 Infrastructure Layer (or) Data Plane Layer

The data plane layer would be the physical layer over which network virtualization lay down through the controller. This layer consists of various networking equipment which may be OpenFlow-enabled or OpenFlow-complaint network devices (routers or switches).

Table 3. 1 Example of OpenFlow-Complaint Switches

Vendor	Series
Arista	Arista extensible modular Operating System (EOS), Arista 7124FX application switch.
Cisco	Cisco cat6k, catalyst 3750, 6500 series
Cinea	Cinea Core director running firmware version 6.1.1
HP	HP procurve series-5400 xzl, 8200 zl, 6200yl, 3500yl
Juniper	Juniper MX-240, T-640
NEC	NEC IP8800
Toroki	Toroki Lightswitch 4810
Dell	Dell z9000 and S4810
Quanta	Quanta LB4G4
Open vSwitch	Software switch, Latest version 1.10.0

The OpenFlow enabled switches are either based on the OpenFlow protocol or compatible with it. In the data plane layer, traffic may enter or exit through logical or physical ports by forwarding or processing functions. Management of forwarding functions performed by an SDN controller or other mechanisms that orchestrated in conjunction with the SDN controller. An OpenFlow enabled switch may be a hardware device or software program which are capable of processing and forwarding of the data path. The examples of OpenFlow-complaint switches are shown in Table 3.1.

3.2.1 Open vSwitch

Open vSwitch (OVS) is a multilayer software switch which aims to implement the software switch platform that provides standard management interfaces and opens the forwarding functions to programmatic extension and control. Open vSwitch is well suited to function as a virtual switch in Virtual Machine (VM) environments and exposed standard control and visibility interfaces to the virtual networking layer, it was designed to provide distribution across multiple physical servers. Open vSwitch also supports multiple Linux-based virtualization technologies including virtual box and Xen/ XenServer. It writes by using in platform-independent C and is easily ported to other environments. OVS can also work entirely in user-space without support from a kernel module and the user-space implementation is easier to port than the kernel-based switch. OVS in user space can access Linux or DPDK devices. OVS contains the following distributions:

- **ovsdb-server (database server):** ovsdb-server provides remote procedure called (RPC) interfaces to one or more OVS databases and supports JSON-RPC client connections over Unix domain sockets and TCP/IP. It is a lightweight configuration database server that holds information for bridges, interfaces, tunnel definitions, OVSDB managers, and an OpenFlow controller address. It also allows ovs-vswitchd to query its configuration
- **ovs-vswitchd (daemon):** It is the core part of the OVS and it manages any number of OVS switches on the local machine. The daemon communicates with SDN controllers, ovsdb-server, kernel module, and hosting system by using OpenFlow, OVSDB protocol, netlink, and netdev interface, respectively
- **ovs-dpctl:** ovs-dpctl is a tool for configuring the switch kernel module
- **ovs-vsctl:** ovs-vsctl is a utility for updating and querying the configuration of ovs-vswitchd
- **ovs-appctl:** ovs-appctl is a utility which sends commands to running OVS daemons

3.2.2 OpenFlow Switch Specifications

There are generally two types in OpenFlow-complaint switches: OpenFlow-only and OpenFlow-hybrid. The first one only processed by OpenFlow pipeline, and cannot support otherwise. OpenFlow-hybrid switches provide both OpenFlow and conventional network operations [101]. An OpenFlow switch logically involves one or more flow tables, one or more OpenFlow channels to external controllers, a group table, a meter table, and shown in Figure 3.3.

- ***Ports***: Packets are passed through the network interface called OpenFlow ports between OpenFlow processing and the rest of the network. OpenFlow switches connect each other via OpenFlow ports. There are generally three types of OpenFlow ports: physical, logical, and reserved ports.

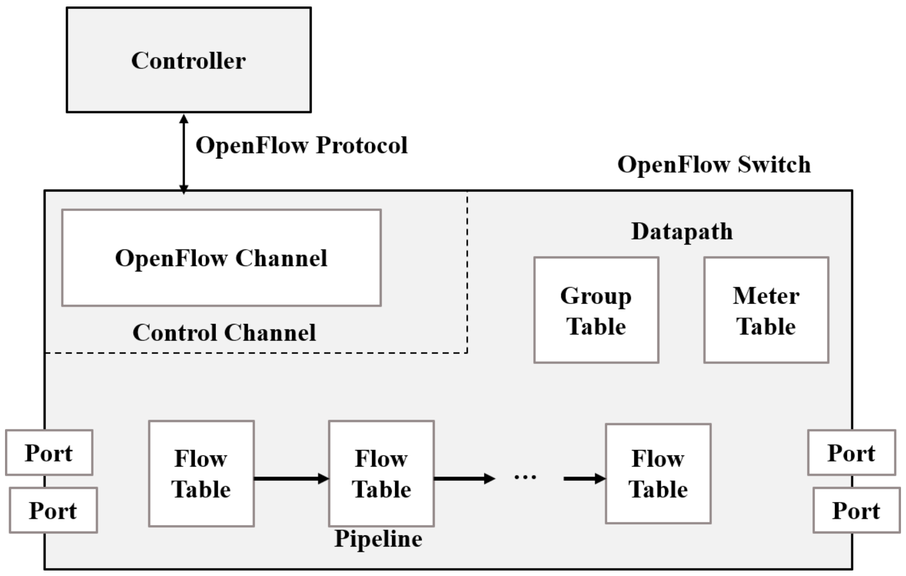


Figure 3. 3 The Main Components of an OpenFlow Switch

- ***Flow Table***: Flow table is a standard table which is used to forward the packet via a single port. A flow table consists of flow entries and each flow entry consists of:
  - *Match fields*: consists of ingress port, packet header, and metadata to match against packets.
  - *Priority*: matching precedence of the flow entry.
  - *Counters*: to update for matching packets.
  - *Instructions*: modify the pipeline processing or action set.
  - *Timeouts*: maximum amount of time before the flow is expired.

    - *Cookie*: Used to provide flow modification, deletion, and statistics by the controller.

- ***Group Table***: Group table consists of group entries and uses for multicast, broadcast, and load balancing purposes. Each group entry contains group identifier, group type, counters, and action buckets.
- ***Meter Table***: Meter table contains meter entries that define per-flow meters which use for various QoS operations such as rate-limiting and DiffServ. A meter measures and controls the rate of packets assigned to it and meter also attaches directly to flow entries. Multiple meters can be used in the same table, but in an exclusive way. Each meter entry is identified by its meter identifier, meter bands, and counters.
- ***OpenFlow Channel***: The interface between OpenFlow switches and controller is called OpenFlow channel. The controller configures the switch, receives the events of the switch, and sends packets to switch via this interface. These OpenFlow channel messages must be configured by the OpenFlow protocol and this OpenFlow channel encrypted using TLS (Transport Layer Security) but may run directly over TCP.

The controller can add, delete, and update the flow tables entries in an OpenFlow switch via OpenFlow protocol.

3.2.3 Pipeline Processing of OpenFlow Switches

In OpenFlow switches, packets are processed in OpenFlow pipeline. OpenFlow packets are received on an ingress port and processed by the OpenFlow pipeline which may forward them to an output port. There are two stages in pipeline processing: ingress and egress processing as shown in Figure 3.4. For the flow tables numbered from 0 to n, the pipeline processing always begins at the ingress processing of flow table 0. The numbers assigned in ingress flow tables must be less than the numbers assigned in egress flow tables.

Firstly, the packet first matches with the first ingress table and other tables may be used depending on the result of the first flow table matched. If the ingress processing outcome is to forward the packet to an output port, OpenFlow switches will start to perform in egress processing in the context of that output port. Egress processing is arbitrary; therefore, a switch may not provide or configure any egress tables to use. If there is no valid configured table at the first egress table, the packet may be executed by the output port or forwarded out of the switch.

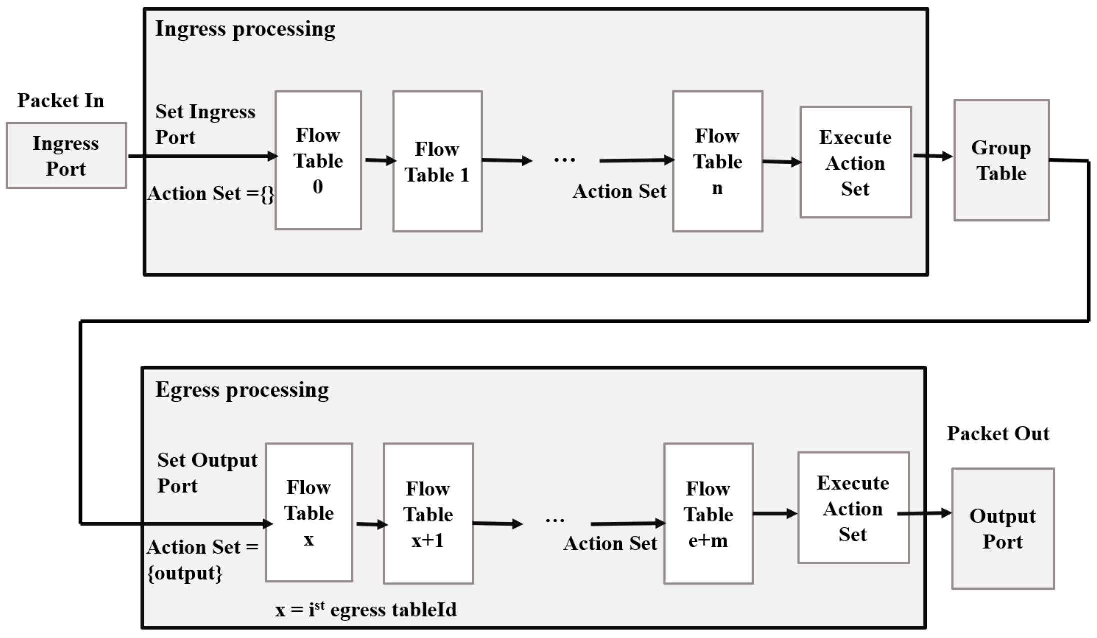


Figure 3.4 An Architecture of OpenFlow Pipeline Process

If there is a valid configured table at the first egress table then the packet must match against the flow entries of that flow table and other tables may be used depending on the result of first flow table matched.

3.2.4 Matching Flow Table in OpenFlow Switches

A flow table entry is identified by its match fields and priority. These match fields and priority is taken together identify a unique flow entry in the flow table. Each flow entries contains match fields (ingress ports + packet header + metadata), counters, and instructions. Figure 3.4 depicts the flow matching structure of OpenFlow.

When handled by a flow table, the packet is matched against with flow entries of a flow table to choose a flow entry. The flow entry instruction set involves actions to be executed at some point of the pipeline. If flow entry is matched, the set of instructions (i.e. Apply-actions, Clear-actions, Write-actions, Write-metadata, and GoTo-table) of this flow entry is operated. When the instruction is GoTo-Table, it may direct the packet to another flow table, where the same process is executed again. When a matched flow entry does not have an instruction that direct to another flow table, the flow table pipeline processing stops at this table then the packet is executed according to this associated action set. If a flow entry is not matched, OpenFlow switch perform table miss function. Table miss performs based on the table configuration

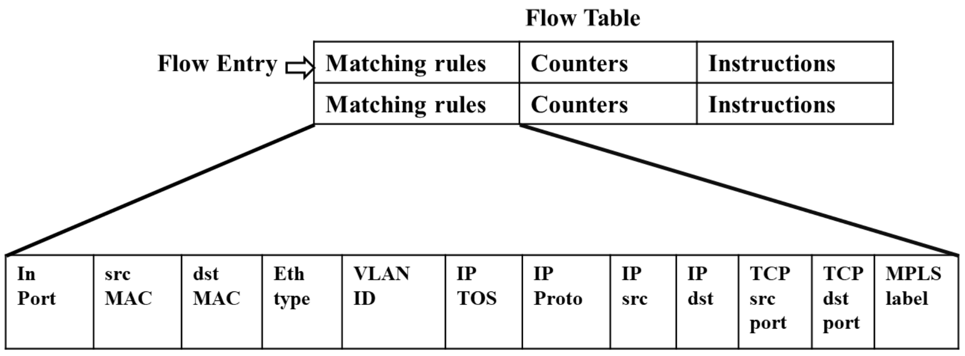


Figure 3.5 Flow Matching Process of OpenFlow

The instruction set for table miss flow table may specify how to execute unmatched packet. The instructions include dropping packets, passing another flow table, and sending back packet-in messages to the OpenFlow Controller via the control channel.

3.3 Protocol Options for Southbound Interface

The control layer communicates the data plane layer by using Southbound APIs (Application Programming Interfaces). The controller uses these APIs to dynamically change forwarding rules that installed in the data plane devices such as switches and routers [72]. There are some examples of southbound APIs that are used for managing network devices in SDN deployment: **NETCONF** (standardized by IETF), **Opflex** (supported by Cisco), **OF-Config** (supported by the Open Network Foundation (ONF)), **OpenFlow** and so on. To support hybrid networks or to utilize traditional networks with software-defined manner, some routing protocols (i.e. OSPF, ISIS, and BGP) have been developed as southbound interfaces in some OpenFlow controller. Currently, the most popular southbound API is OpenFlow.

3.3.1 The Concept of OpenFlow Protocol

OpenFlow is a standardized communication protocol which defines the communication between an OpenFlow switches and OpenFlow controller. It is also a programmable network protocol that supports an open standard-based programming interface for multiple vendors to manage and supports network traffic. For instance, the

SDN controller can configure and manage (i.e. installing packet forwarding rules) data plane devices (i.e. OpenFlow switches) through the OpenFlow protocol. The switches can send notification messages (i.e. different kinds of events) to the controller via OpenFlow.

At initialization, switches configure the IP address and TCP port number of their SDN controller, then switches contact with controller by using these IP and TCP port. Switches establish secure connection by using Transport Layer Security (TLS) session. Afterwards, the controller requests configuration information (for example: port number and mac address) from each switch by sending OpenFlow OFPT FEATURES REQUEST message to know about the existence of the switches in the network. There are mainly three types of OpenFlow messages:

- ***Controller to switch messages***: These types of messages are used to directly control or check the state of the switch and initiated by the controller. These are
  - *Features*: To establish the OpenFlow channel, controller sends a feature request message to switch for requesting the capabilities of a switch and the switch reply a feature reply message.
  - *Configuration*: The switch only responds to the controller's set and query configuration messages.
  - *Modify-State*: These types of messages are also called 'FLOW_MOD' message which are used to modify, add, and delete flow or group entries and sent by the controller.
  - *Read-State*: These messages are used by controller to get numerous information (i.e. current configuration and port statistics) from switches.
  - *Packet-Out*: Packet-out message consists either full or buffers ID sent by the controller.
  - *Barrie*r: Barrier request or reply messages are applied by controller to ensure message dependencies have been met and get notifications for completed operations [101].
  - *Role-Request*: Set the role of its OpenFlow channel used by the controller.
  - *Asynchronous-Configuration*: These messages are used by the controller to define an additional filter on an asynchronous message of OpenFlow channel.
- ***Asynchronous***: These types of messages are applied to change the switch state and update the controller with the network events changes. These messages are

initiated by switches. These messages are:

  - *Packet-in*: Transfer the control of a packet to the controller. It may be table-miss flow entry, TTL checking or packet-in events.
  - *Flow-Removed*: Inform the controller about the flow has been removed because of the controller's flow delete request or the switch's flow expiry process.
  - *Port-Status*: Inform the controller about the status of the port.
  - *Error*: The switch enables to notify the problems to controllers using error messages.

- ***Symmetric***: These types of messages are initiated by either the controller or the switch and sent without solicitation. Five symmetric messages have been represented as a part of the OpenFlow protocol:
  - *Hello*: Hello messages or keep-alive messages exchanged between switch and controller upon connection startup.
  - *Echo*: To verify the liveness of connection, the controller and switch used echo request/reply messages.
  - *Experimenter*: To supports additional functionality within OpenFlow message type space or an area for the features of future OpenFlow versions.

3.4 Control Layer of SDN

The control layer is a core layer of the SDN architecture that extracts the data plane layer information and communicates to the application layer with an abstract view of the network topology, consisting of statistics and event [84].

Table 3. 2 Features Comparison of Popular SDN Controllers

Controller	Implementation	Developers	Application Domain
NOX	Python	Nicira Networks	Campus
POX	Python	Nicira Networks	Campus
Ryu	Python	NTT	Campus
FloodLight	Java	Big Switch Networks	Campus
OpenDayLight	Java	The Linux Foundation	Datacenter
ONOS	Java	Open Networks Foundation	Datacenter, WAN and transport

3.4.1 ONOS Controller

ONOS (Open Network Operating System) supports the control plane for an SDN architecture, manages network components, such as switches and links. ONOS also runs software programs or modules to provide communication services to end hosts and neighboring networks. ONOS designs to help network service providers build carrier-grade software-defined networks architected for high availability, scalability, and performance. Moreover, it can run as a distributed system across multiple servers and its applications and use cases consist of customized communication routing, management, or monitoring services for software defined networks [100].

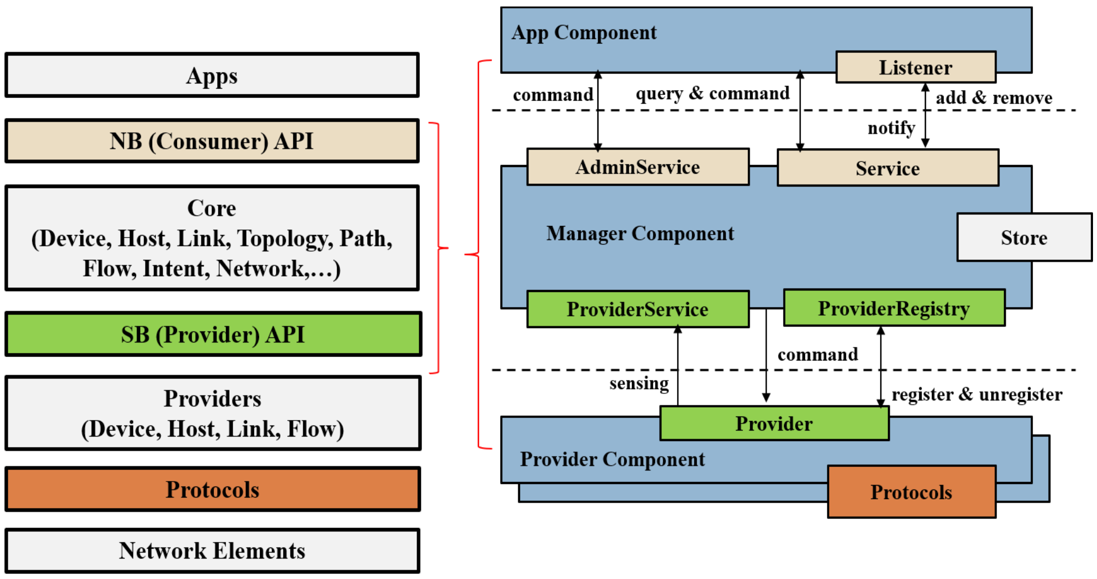


Figure 3.6 ONOS Architecture Tiers and Subsystem Structure

The ONOS kernel, core services, and applications are written in Java as bundles that are loaded into the Karaf OSGi container. OSGi is a component system for Java that permits modules to be installed and run dynamically in a single JVM. Moreover, ONOS can run on several underlying OS platforms because it runs in JVM. Figure 2.3 shows the architecture tiers of ONOS and its subsystem structure. A service is a unit of functionality that consisted of multiple components that produce a vertical slice through the tiers as a software stack. ONOS defines many primary services such as *Device, Link, Host, Topology, PathService, FlowRule, Packet services* and so on. There are three main tiers in ONOS stack: *Apps, Core*, and *Providers*. The *Providers* interconnect with the network elements using several control and configuration protocols and supplying service-specific sensory data to the core.

The *core* tire which consists of *Manager* component and it is managed to accept and transmit information with the *Provider* via southbound APIs (*Provider Service, Provider Registry*) and with the Apps via northbound APIs (*Admin Service, Service*). The *Apps* interconnect with the *core Manager* to obtain the data. For instance, if an *App* needs information from network elements or if an *App* needs to know the current state of network elements, it can request information by using synchronous call or starts listening to asynchronous events. Then, the *Manager* will apply the correct *Provider Services* to retrieve the data from the network elements via protocols and serve it back to synchronous request or trigger an asynchronous event notification. When multiple ONOS instances are applied, the consistency of requested or changed information across ONOS instances is responsible for a Store component.

3.5 Application Layer of SDN

The application layer is an open area for developing as many innovative applications as possible by taking the advantage of the global view of network information, such as all network topology information, network statistic, network status, etc. The SDN applications are programmed to support all kinds of network services such as traffic engineering, load balancing, routing, network monitoring, network setup and management, network troubleshooting, network policy, and security. Such SDN applications can contribute various end-to-end solutions for data center and real-world enterprise networks [38]. SDN applications are directly and programmatically communicated SDN controller via northbound APIs. Besides, the applications can build an abstracted view of the network by gathering information from the SDN controller for decision-making purposes.

3.6 Managements of Flow Entries in OpenFlow Networks

In SDN architecture, the controller must install flow table entries in the forwarding tables of the switches. The wildcard, match fields of flow entries are classically installed in TCAM for fast packet matching and forwarding. TCAMs are relatively small, expensive and limited number of flow entries can be placed in the flow table. The flow entry management system of OpenFlow switches can basically be categorized into two approaches: proactive and reactive.

In proactive flow management, controller pre-calculates and populates flow entries into the flow tables of the switch. This type of installation does not incur additional flow setup time and latency because every flow does not consult with the controller. However, there is no flexibility for real-time network traffic engineering and a large number of entries that hold in flow table might not fit with TCAM. To address the issues of large flow tables management, flow entries can be installed reactively. The basic operations for reactive flow management are depicted in Figure 3.8:

1) Packets arrive at the switch and there are no corresponding flow entries in switch's flow table.
2) Therefore, the switch informs the controller about the packet.
3) The controller determines the path for the packet and puts in suitable rules in each switch along the path.
4) Packets are forwarded to the destination.

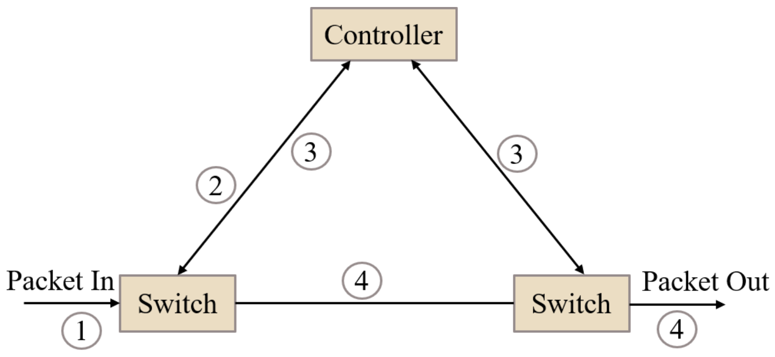


Figure 3.7 Reactive Flow Management

The reactive mechanism is a timeout-base flow management mechanism and the default expiry timer is one second set by the controller. The switch tracks and removes every expiry flow. When more packets of expired flows arrived, the switch requests the flow entries and the controller must calculate for appropriate paths again.

Both reactive and proactive mechanisms have different pros and cons. In a reactive approach, if a new flow arrives or if a switch's flow table has no appropriate flow entry, the controller interaction is needed. This instantiation efficiently uses flow table but every flow suffers additional flow setup time which relies on the control channel and the current load of the controller. Therefore, reactive flow management may diminish the states and number of large flow tables in the switches, but it may rise the delay and reliability requirements of the control channel and control plane software. Especially, the failures of the control plane software and control channel will have a great effect on the overall network performance, if flow entries cannot be settled in a timely manner.

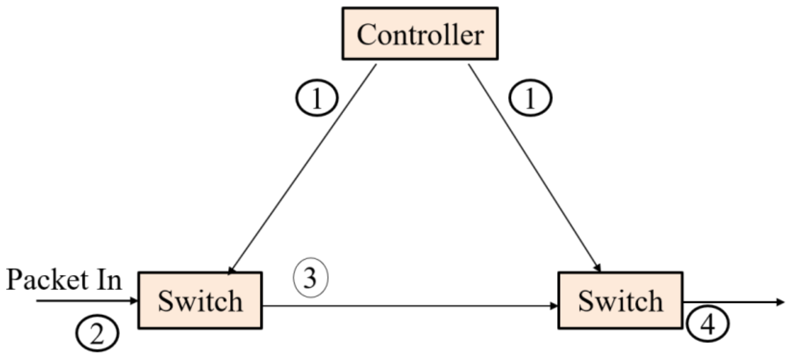


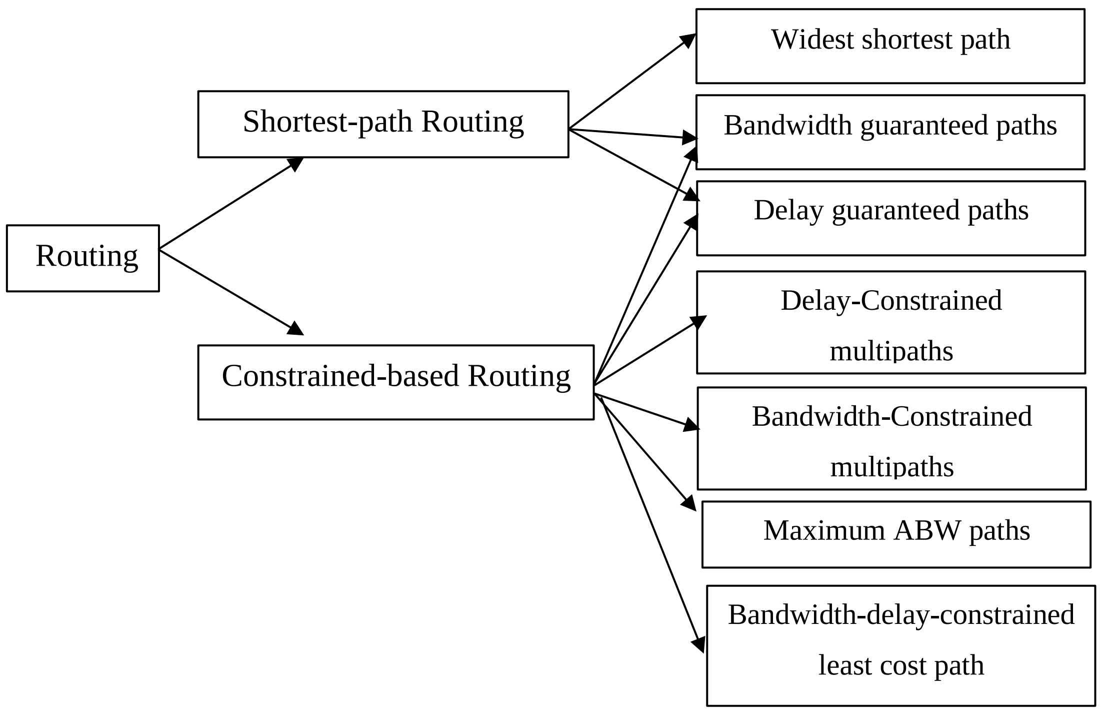
Figure 3.8 Proactive Flow Management

In a proactive approach, all the required flow entries are installed in the flow tables of the switches. Therefore, this approach reduces the controller workload and it is more robust to control layer failure because the required flow entries are already installed into data plane layer switches. However, this approach needs to install a huge amount of flow tables when the network is bigger and this may be caused TCAM limitation problem.

To overcome the limitations of proactive and reactive approaches, the combination of both proactive and reactive mechanism, called a hybrid flow management mechanism becomes popular. Hybrid flow management approach gets more flexibility. Before communication is started, flows rules are installed like proactive, and when communication is started, it treats the flows reactively.

3.7 Innovation Through Routing based SDN Application

In an SDN architecture, Network managers can innovate their application according to their requirements. Therefore, many researchers pay attention to the following applications: traffic engineering application, routing, load balancing, and security applications. In SDN based networks or not, routing generally involves two entities: network state information and routing algorithms. The network state information is the network resources at nodes and links including link utilization, available bandwidth, delay, and packet loss rate.



Routing
Shortest-path Routing
Constrained-based Routing
Widest shortest path
Bandwidth guaranteed paths
Delay guaranteed paths
Delay-Constrained multipaths
Bandwidth-Constrained multipaths
Maximum ABW paths
Bandwidth-delay-constrained least cost path

Figure 3.9 Routing Algorithms in SDN and Traditional Networks

Routing algorithms apply this network state information to find routes with satisfying resources or demand. However, network state information can dynamically change because of links up and down states, fluctuations of load, and connection in and out states. In legacy networks, network state information is gathered by using distributed routing protocols and legacy networks also gain and distributes this network state information from and to routing devices. In SDN, the controller collects and updates the network states information from routing devices via direct connection of OpenFlow.

A routing algorithm in which a router computes the shortest path between each pair of nodes in the network. The Open Shortest Path First (OSPF) Protocol is based on the Shortest Path First (SPF) algorithm. The most critical interests all the time in networks are traffic management or routing which focus how to decide paths depending on required constraints such as network QoS parameters. This type of routing is also called constraint-based routing. Figure 3. 9 depicts the various routing algorithms that are widely used in SDN, SDN based IoT networks, SDN based cloud data centers networks, and conventional networks. According to Figure 3. 9, there are two main types of routing: shortest path routing and constrained based routing.

3.8 Chapter Summary

This chapter briefly explains the background theory of layer taxonomy of software defined networks. This chapter also describes the primary SDN protocol and how it works. Moreover, the architecture and functions of Open vSwitch (popular OpenFlow switches) are presented in this chapter. The traffic management applications such as routing is the most critical task for TE. Therefore, this chapter also presents various routing methods that are widely used in SDN and conventional networks.

CHAPTER 4

THE ARCHITECTURAL DESIGN OF THE PROPOSED SYSTEM

The purposes of the chapter are identifying the problems of traffic engineering without awareness of application, and describing the proposed application-aware engineering architecture with step by step explanation.

4.1 Problem Definitions and Motivations

The traditional Shortest Path First (SPF) algorithm routes the traffic efficiently, but the congestion may occur. SPF also produces bottlenecks for future traffic demands. SPF only takes account of the minimum hop-count and does not achieve QoS-aware TE and load balancing. Different applications need different routing depending on their own application preferences. Therefore, this dissertation has implemented an application-aware traffic engineering in SDN environment.

In this work, ONOS is used controller as the control plane and Mininet network emulator as the data plane network. The detailed requirements and implementations of software and hardware will be explained in section 5.1. In ONOS, there are many applications such as segment routing, SDN-IP, default forwarding (or) reactive forwarding and so on. Reactive forwarding (denoted as onos-app-fwd) is one of the shortest path computation mechanism on discovered topology by using Dijkstra algorithm. This forwarding worked are as follows: when a new packet enters a switch, firstly the switch lookup flow entries in its flow tables. If there have no matched flow entries, the switch asks the controller's decision to manage the packet. The controller processes the packet and defines a flow entry based on the end-to-end paths then installs flow entry into the network switches through the path. Therefore, reactive forwarding is assumed as a shortest path routing and analyzed by multiple testing scenarios.

Figure 4.1 is a sample test topology that has tested to explain the issues of shortest path routing. There are 4 switches and 6 hosts that involved in sample test topology. To easily and simply analyzed, the specified bandwidth of each link in the network is defined as 20 Mbps. Table 4.1 describes the available paths information between source hosts (H1, H2) and destination hosts (H5, H6).

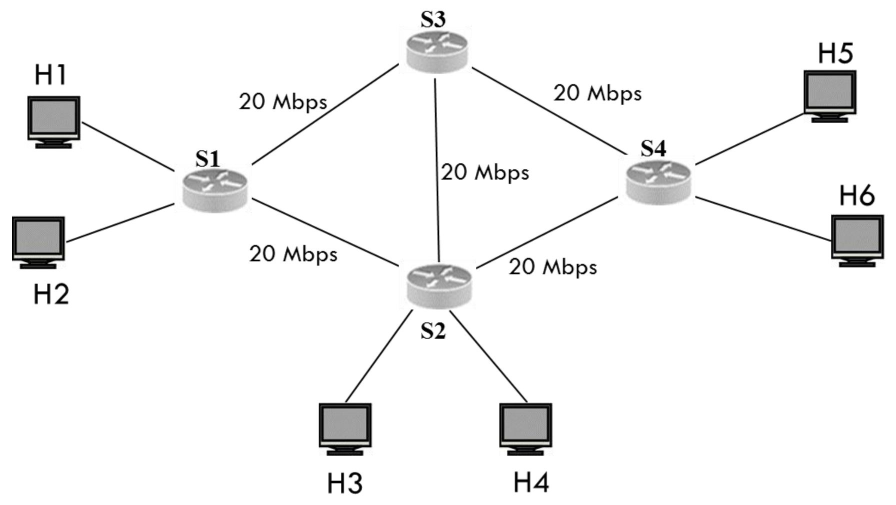


Figure 4.1 Sample Test Topology

This experiment generated different bit rates of UDP traffic flows between switches S_1 and S_4 and carried out the following two tests called test 1 and 2 here.

1. **Test 1**: H1 sends 15 Mbps of UDP traffic to H5 and H2 parallel sending 5 Mbps to H6.
2. **Test 2**: H1 sends 15 Mbps to H5 and H2 parallel sending 10 Mbps to H6.

Table 4.1 Available Paths Between S1 and S2

Source	Destination	Available Paths	Available Bandwidth
H1, H2	H5, H6	P_1= {S_1, S_3, S_4} P_2= {S_1, S_2, S_4} P_3 = {S_1, S_3, S_2, S_4} P_3 = {S_1, S_2, S_3, S_4}	20 Mbps

According to Figure 4.1 and Table 4.1, it is seen that there are two equal cost paths between switches S_1 and S_4 {p_1= {S_1, S_2, S_4}, p2= {S_1, S_3, S_4}}. The idea of the shortest path routing (reactive forwarding or default forwarding) is to select a single shortest path. In test 1, if the traffic volume is equal to the current link capacity, the packet loss rate is nearly % and the average throughput result can be seen in Figure 4.2.

In test 2, if the traffic volume is greater than the link capacity, the average throughput results of 15 Mbps and 10 Mbps for the reactive forwarding (shortest path routing) are shown in Figure 4.3. In this case, the default forwarding selects path p_1 and that path p_1 already consumed 15 Mbps and left 5 Mbps. Parallel to this, when H2 sends 10 Mbps to H6, the default forwarding selects again the path p_1 even the alternative path p_2 provides a better ABW (20 Mbps) because the flow entries for source switch S_1 and S_2 already exists. Therefore, nearly 23% of packet loss occurred in default forwarding.

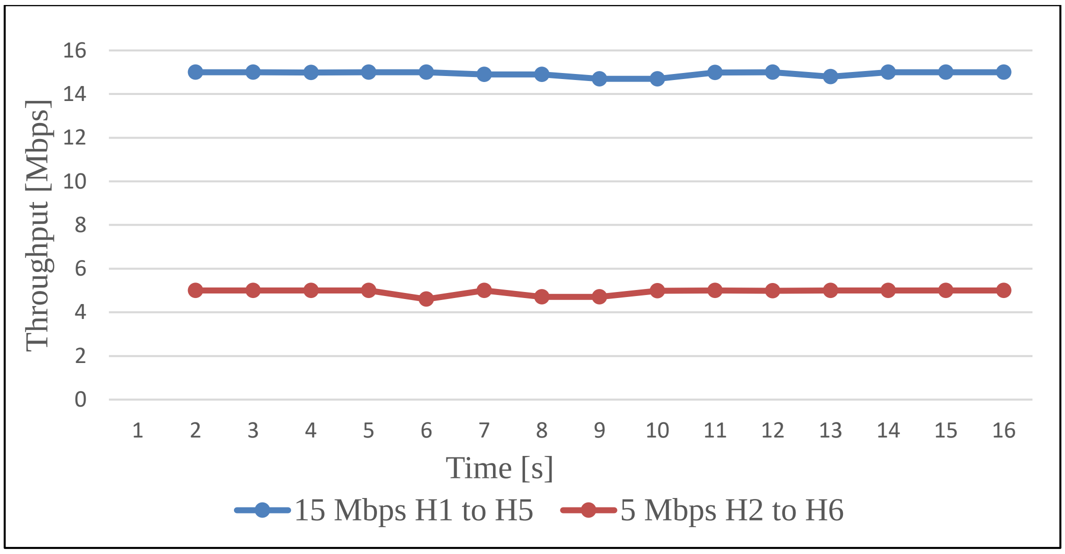


Figure 4.2 Average Throughput Results of Test 1

By analyzing these tests, it can be concluded the shortest path routing (default forwarding) is a simple and fast packet forwarding, however, it always routes every traffic via shortest path, lacks the sense of load balancing and decreases link and path utilization.

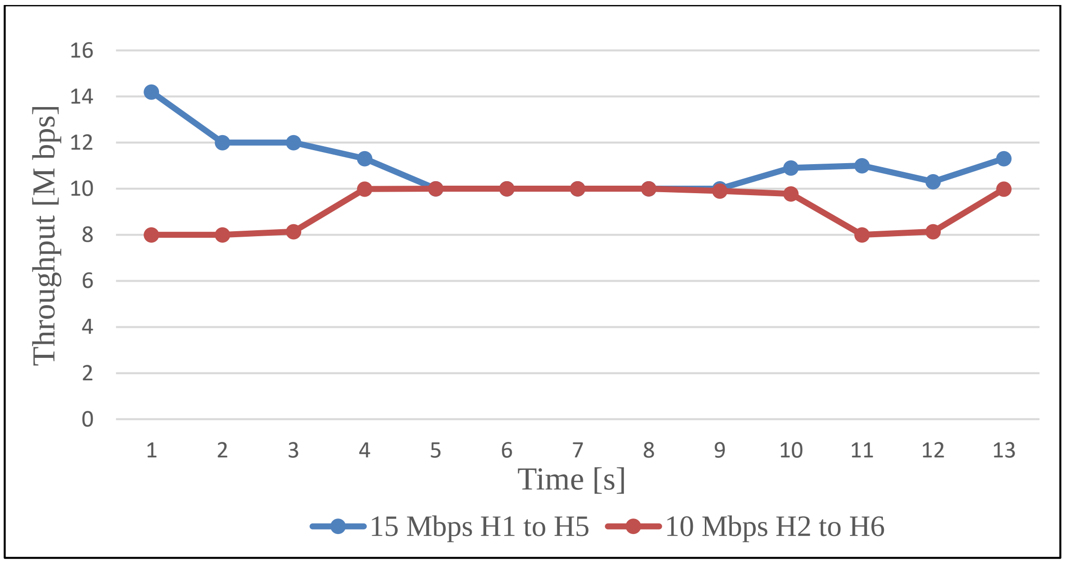


Figure 4.3 Average Throughput Results of Test 2

4.2 Application-aware Traffic Engineering Architecture

The Application-aware TE (App-TE) generally involves three main modules: traffic classification, traffic measurement, and traffic management. When the incoming traffic enters the network, App-TE first classified whether the priorities traffic or not, then performed the other modules (i.e. measurement and management). The overall system design is shown in Figure 4.4.

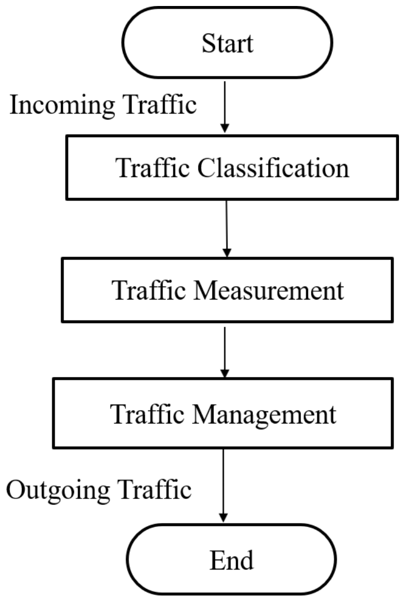


Figure 4.4 The overall system design

The SDN applications are programmed to support all kinds of network services such as traffic engineering, load balancing, routing, and monitoring. As illustrated in Figure 4.5, an App-TE is an application which implemented on the SDN application layer and written in Java. Topology Manager, Device Manager, Port, Link, and Path services are Application Programming Interface (APIs) of the ONOS, SDN controller. The control layer extracts the data plane layer information and communicates to the application layer with an abstract view of the network topology, including statistics and events. Communications between the data plane and control layers use a standardized protocol called OpenFlow.

Moreover, the traffic analyzer (sFlow-RT) is used to help the traffic classification and reduce the controller’s work load. The detailed explanation for each traffic module will be described in the next sections.

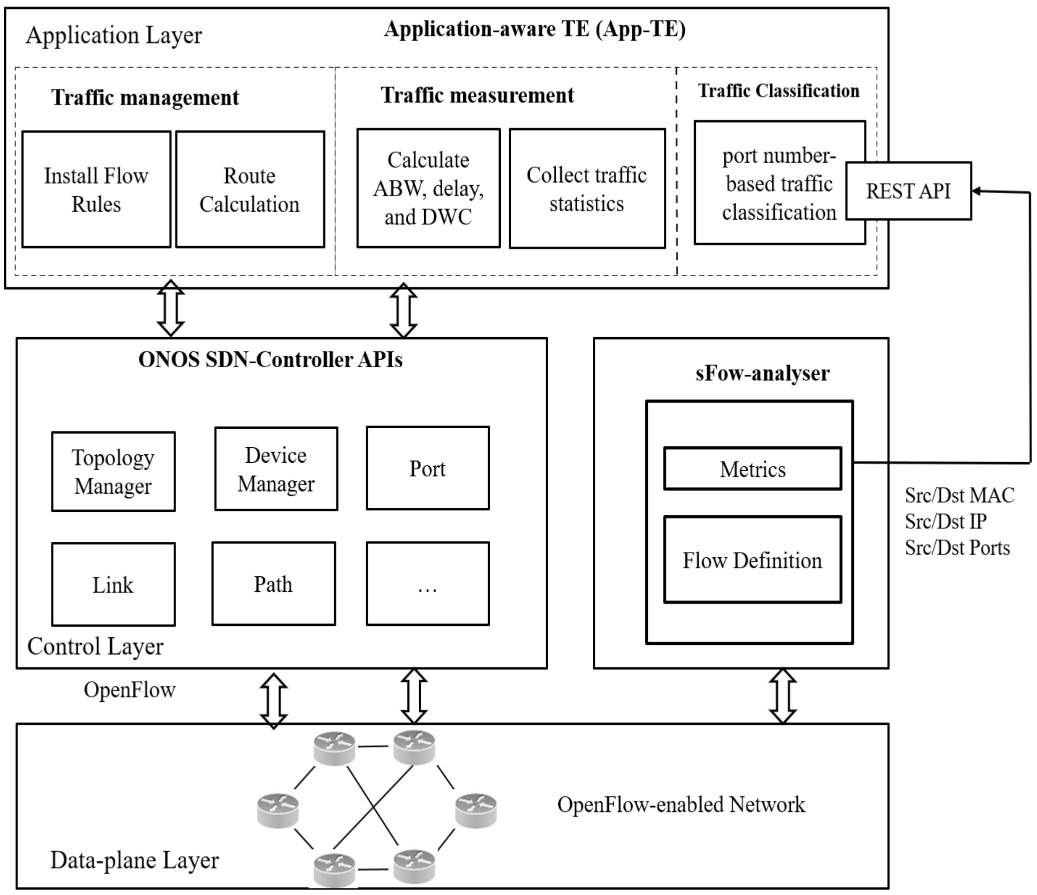


Figure 4.5 The Architecture Design of Application-aware TE (App-TE)

4.3 Traffic Classification in App-TE

In traffic classification module, App-TE used sFlow-RT as a prerequisite for traffic monitoring and classified application traffic by port-number. The sFlow analytics engine obtains a continuous telemetry stream from sFlow Agents embedded in network devices and converts them into actionable metrics, accessible through the REST flow API [89].

Table 4.2 Classified Classes in App-TE

Classes	Traffic
Prioritized	Video streaming, file transferring, and haptic stream
Non-Prioritized	Other traffic

The idea of using sFlow-RT in traffic classification is to reduce the controller work load for monitoring. When the application traffic is entered the network, the sFlow agent accessed this traffic and sent to sFlow collector. sFlow collector analyzed and sent back to controller as the accessible metrics via the REST API.

This section chose application traffic which not only widely used in current networks but also work well in Mininet network emulator. To reduce computational complexity of controller, App-TE simply classified application traffic flows into the following two classes: prioritized and non-prioritized classes. The detailed information of these two classes are shown in Table 4.2. The detailed step by step process for the traffic classification module is written as the Algorithm 4.1 as shown in Figure 4.6.

Algorithm 4.1: Traffic Classification in App-TE

```
1:   #previous_timestamp = 0
2:   #flows = [srcIP, dstIP, srcMac, dstMac, srcPort, dstPort, timestamp]
3:   #priorities_flows = [RTP, FTP, HTTP, Haptics], non-priorities_flows = other flows
4:   function: Flows_Detecting(flows):
5:       if (timestamp > previous_timestamp)
6:           check priorities or non-priorities flows by Port numbers
7:           if  flows = priorities_flows
8:             then DWC-aware Routing(flows)
9:           end if
10:          if flows = non_priorities_flows
11:            then MHR(flows)
12:          end if
13:      end if
14:  end function
```

Figure 4.6 Algorithm for Traffic Classification in App-TE

4.4 Traffic Measurement in App-TE

Collecting traffic statistics, calculating available bandwidth, link delay, and Delay-weighted Capacity (DWC) are the responsibility of traffic measurement module.

4.4.1 Estimating Available Bandwidth using OpenFlow messages

According to the literature reviews of section 2.3.2, many researchers estimated the QoS parameters such as ABW and link delay by using OpenFlow protocol. They estimate these parameters by active technique (sending probe packets to the switches which need to estimate QoS parameters) and passive technique (monitoring at the specified period). For ABW estimation in SDN, passive techniques are more popular and widely used because of the centralized view of SDN.

By taking advantage of SDN's global centralized control, this dissertation also uses OpenFlow messages to calculate the ABW. OpenFlow has many statistics messages such as flow stats, meter stats, aggregate stats, queue stats, port stats, and table stats. OpenFlow permits the controller to query the statistics information of the switches. Therefore, the controller can request the current statistics information from the switches by sending OpenFlow *Statistics_REQUEST* message to the switches. After the time, T_s, the execution time of the switch, the switch reply OpenFlow *Statistics_REPLY* message to the controller with its current statistics information. Figure 4.7 depicts the process of OpenFlow request/reply messages.

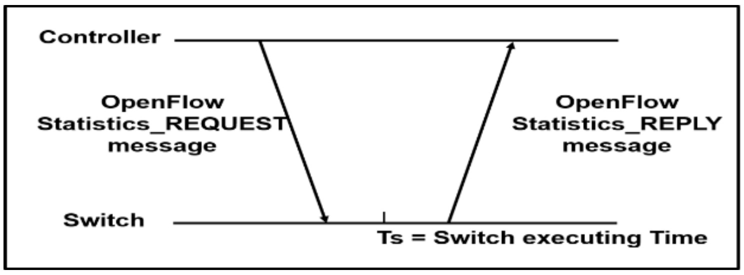

Controller
OpenFlow Statistics_REQUEST message
OpenFlow Statistics_REPLY message
Switch
Ts = Switch executing Time

Figure 4.7 OpenFlow request/reply messages between switch and controller

However, OpenFlow does not implement a way to gather the network QoS parameters, for instance, link utilization or available bandwidth from the switch directly. Therefore, the controller uses to make sense of the raw statistics values to determine the available bandwidth of the switches.

Table 4.3 The port statistics counter values

Type	Metrics	Unit	Counter	Unit
Port Statistics	pktRx	n/s	Received Packets	n
	pktTx	n/s	Transmitted Packets	n
	byteRx	Bytes/s	Received Bytes	Bytes
	byteTx	Bytes/s	Transmitted Bytes	Bytes
	pktRxDrp	n/s	Packet Received Drop Rate	n
	pktTxDrp	n/s	Packet Transmitted Drop Rate	n

There are many types of traffic statistics related with switch or router's port such as packets Received (pktRx), packet Transmitted (pktTx), bytes Received (bytesRx), bytes Transmitted (bytesTx), packet Drop Rate (pktRxDrp) and packet Transmitted Drop (pktTxDrp) as shown in Table 4.3. The received bytes of one switch were more than the bytes transmitted by other switch at the ports through which they are connected.

Therefore, this dissertation only used the transmitted bytes counter values and it also used northbound API (*DeviceService.getDeltaStatisticsForPort*) to extract the port statistics of each switch port. The link capacity is the defined link capacity and the link-load can be obtained by applying the calculated some byte counter values. The link load, L of i^{th} link at time *t* can be obtained through the Equation (4.1):

$$L_i(t) = srcPort\_bytesTx\ (t) + dstPort\_bytesTx\ (t) \tag{4.1}$$

where, $srcPort\_bytesTx\ (t)$ is the source port statistics of transmitted bytes count at time t and $dstPort\_bytesTx\ (t)$ is the destination port statistics of transmitted bytes count at time *t.* After calculating each link load along a given path, the available bandwidth (ABW) of i^{th} link at time *t* can be derived as Equation (4.2):

$$ABW_{link_i}(t) = C_i(t) - L_i(t) \tag{4.2}$$

where, $ABW_{link_i}(t)$ is the available bandwidth for the i^{th} link at time *t*, $C_i(t)$ is the capacity of the i^{th} link and $L_i(t)$ is the link load for i^{th} link at time *t*. We can obtain the ABW of a link by subtracting the link load from the defined link capacity. Then, ABW of a given path is obtained through the following equation Equation (4.3):

$$ABW_{path}(t) = \min_{link_i \epsilon Path} ABW_{link_i}(t) \tag{4.3}$$

where, $ABW_{path}(t)$ is the ABW of a path at time *t*, which is the minimum ABW of links along a given path, and $link_i \epsilon Path$.

However, querying port statistics from all the switches in the network may increase the controller's load and computation time. Therefore, the traffic measurement module in App-TE only queried statistics from the source and destination switches of incoming traffic and then calculated the link utilization.

4.4.2 Estimating Link Delay

There has been numerous research handling estimation of link delay. One of the solutions [100] is to estimate the end-to-end link delay and it is as follows:

$$T_{end-to-end-delay} = T_{total} - (T_{controllertosourceswitch} + T_{controllertodestinationswitch}) \qquad (4.4)$$

where T_{total} is the time duration to send a probe packet from the controller to source switch plus source switch to destination switch plus destination switch to the controller. $T_{end-to-end-delay}$ is the delay time form source switch to destination switch. The solution got one-way delay by subtracting the delay time of the controller to source switch, and controller to destination switch from the total time, T_{total}. This work assumed that the link delays are already known according to the global view of SDN. Therefore, the link delay of each link is specified delay of the link when the network is started. The total end-to-end delay for the path is the sum of each link delay along the path through the network as in Equation (4.5):

$$Delay_{path} = \sum_{link \in Path} link_i \qquad (4.5)$$

where, $Delay_{path}$ is the total end-to-end delay for a path and $link_i$ is the i^{th} link delay and that link include in that path, $link \in Path$.

4.4.3 Estimating Delay-Weighted Capacity (DWC)

In the prioritized application traffic, some are delay-sensitive applications such as haptic and video streaming but the other are non-delay-sensitive applications which required more bandwidth. The non-delay-sensitive applications are file transferring and so on. Therefore, App-TE further considered the weighted value of delay and available bandwidth for prioritized applications. The DWC is obtained dividing the available bandwidth by the total path delay (Delay) as in Equation (4.6).

$$DWC = \frac{Available\ Bandwidth}{Delay} \qquad (4.6)$$

The source switch (*s*) to the destination switch (*d*), the DWC is obtained through the Equation (4.7):

$$DWC_{s,d} = \sum_{p_{s,d}^{i} \epsilon p_{s,d}} \frac{ABW_{s,d}^{i}}{Delay_{s,d}^{i}} \tag{4.7}$$

where, $DWC_{s,d}$ is the DWC value of source (s) and destination (d), $ABW_{s,d}^{i}$ is the available bandwidth of s and d for i^{th} link, and $Delay_{s,d}^{i}$ is the delay value of s and d for i^{th} link. App-TE defined the optimal path with the maximum weighted sum of DWC value between source and destination.

4.5 Traffic Management in App-TE

The traffic management module in App-TE performed the following four main tasks. First, App-TE extracted all possible paths between the source and destination switches. Then, App-TE queried port statistics from the selected switches and calculated ABW, delay, and DWC. After that, App-TE selected maximum DWC path (DWC-aware routing) for prioritized traffic and minimum hop-count path (minimum hop-count based routing) for non-prioritized traffic. Finally, App-TE installed flow rules for calculated paths into the intermediate switches along the path.

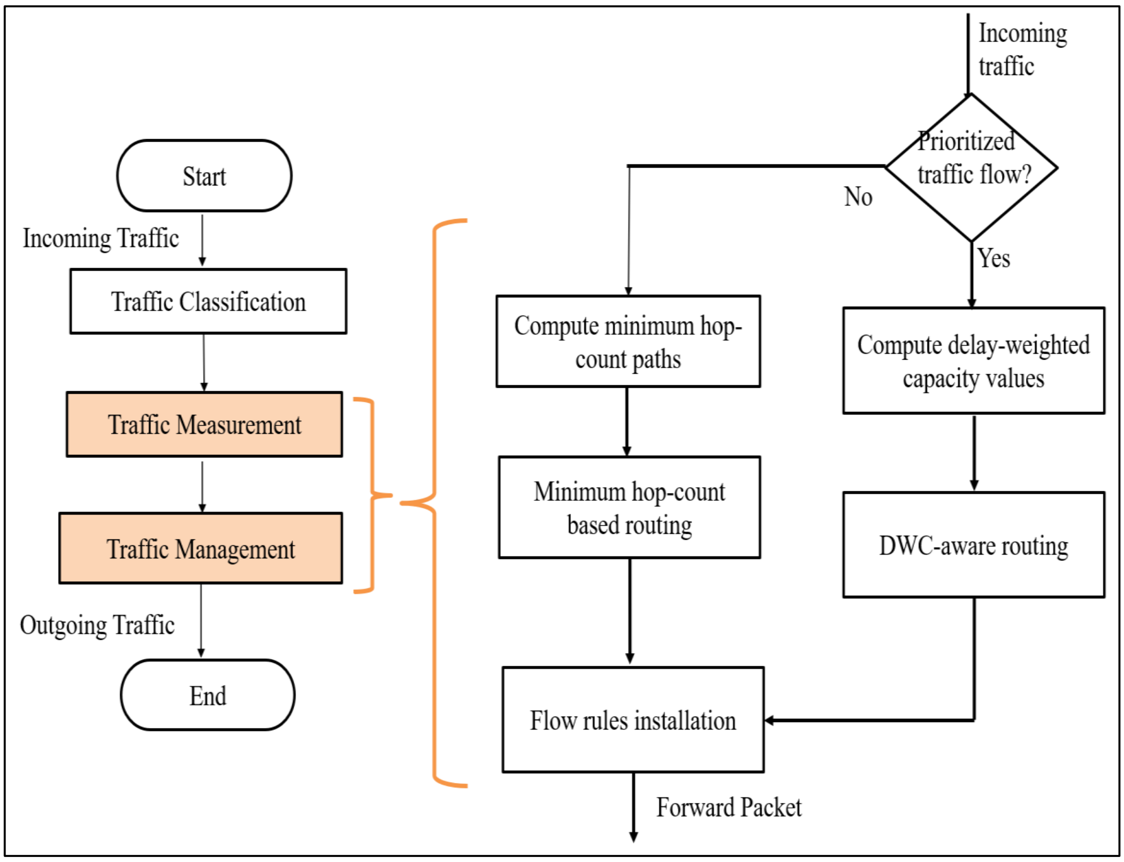


Figure 4.8 The Flow Diagram for Traffic Management Module

As illustrated in Figure 4.8, when the classified traffic is entered the network, the App-TE first checked whether the prioritized traffic or not. If the classified traffic is prioritized traffic, the App-TE calculated the ABW, delay, and DWC. Then, it performed the DWC-aware routing. If the classified traffic is non-prioritized traffic, the App-TE calculated minimum hop-count path and performed the minimum hop-count based routing.

4.5.1 Minimum Hop-count-based Routing (MHR)

When the non-prioritized application traffic is entered the network, App-TE routed this application traffic through the minimum hop-count path. Figure 4.9 shows step by step process of MHR algorithm.

Algorithm 4.2: MHR Algorithm

```
 1:  #srcIP = source host IP address, dstIP = destination host IP address
 2:  #srcMac = source Host MAC Address, dstMac = destination Host MAC
     Address
 3:  #srcPort = source Port number, dstPort = destination Port number
 4:  #srcDeviceId = deviceID of switch that source host connected
 5:  #dstDeviceId = deviceID of switch that destination host connected
 6:  function: Minimum_hop_count_based_Routing (srcIp, dstIp, srcMac,
     dstMac, srcPort, dstPort):
 7:      get srcDeviceId and dstDeviceId from srcMac and dstMac
 8:      all_shortest_paths = getPaths (srcDeviceId, dstDeviceId)
 9:      select one path from all_shortest_paths
10:      install flow rules through the selected path
11:      forward the packets
12:  end function
```

Figure 4.9 Minimum Hop-count-based Routing Algorithm

Firstly, MHR extracts the DeviceIDs of source and destination switches. Then, MHR finds the all possible shortest paths between the source and destination switches by using the ONOS's northbound APIs (PathService). Finally, selects one path from all shortest path and installs flow entries of selected path to the intermediate switches through that path.

4.5.2 Delay Weighted Capacity-aware Routing (DWC-aware Routing)

When the prioritized-application traffic is entered the network, this application traffic is routed by using DWC-aware routing. Figure 4.10 describes how DWC-aware routing is worked.

Algorithm 4.3: DWC-aware Routing Algorithm

```
1:  #srcIP=source host IP address, dstIP=destination host IP address,
2:  # srcMac=source Host Mac, dstMac=destination Host Mac
3:  #srcPort=source Port number, dstPort=destination Port number,
    srcDeviceId=deviceID of switch that source host connected
4:  function: DWC-aware-Routing (srcIp, dstIp, srcMac, dstMac, srcPort, dstPort):
5:      get srcDeviceId and dstDeviceId from srcMac and dstMac
6:      construct Depth first search graph
7:      all_possible_paths = getDFSpaths (srcDeviceId, dstDeviceId)
8:      for path in all_possible_paths do:
9:          for link in path. links () do:
10:             ABW = link. Capacity () - link. load ()
11:             total_delay += link_delay
12:         end for
13:         ABW_path = min (ABW)
14:         DWC = ABW_path / total_delay
16:         optimal_path = max (Delay_weighted_capacity)
17:     end for
18:     install flow rules through the maximum DWC path
19:     forward the packets
20: end function
```

Figure 4.10 DWC-aware Routing Algorithm

DWC-aware routing works as follows:

- extracts the DeviceIDs of source and destination switches.
- constructs the Depth First Search graph for source and destination switches. When ONOS needed to extract the paths between source and destination switches, it used the PathService API. PathService can support shortest paths

and disjoint paths between the source and destination switches. However, one of the objectives of App-TE is to improve network utilization and choose the best path for application traffic. Therefore, App-TE needs all possible paths to perform DWC-aware routing.

- calculates ABW, delay, and DWC for each link and path. Then, selects maximum DWC path.
- Finally, DWC-aware routing installs flow entries of selected path to the intermediate switches through that path.

4.6 Chapter Summary

This chapter describes the architecture of App-TE with its three modules. Firstly, it discusses the issues of shortest path routing and demonstrates with average throughput results. Then, this chapter explains the three main modules of App-TE with system diagram and algorithms.

CHAPTER 5

IMPLEMENTATION AND EVALUATION OF THE PROPOSED SYSTEM

The implementation of experimental testbed and evaluation of the proposed system are discussed in chapter 5. Firstly, the hardware and software requirements of experimental testbed are described. Then, the experiments are carried out by using the following methods: App-TE, link-utilization aware routing, and shortest path routing. Finally, the chapter discussed the experimental results of each method by conducting different scenarios.

5.1 Design and Implementation of Experimental Testbed

In order to assess the performance of the App-TE, the experimental testbed has to be designed. As the App-TE is implemented using ONOS controller, it runs in tested topologies in order to compare and evaluate the results with or without the App-TE. In order to have deterministic and low-cost environments to test, a virtual testbed was created that can run on two machines and do not require additional effort to be maintained and operated. This dissertation applied ONOS controller as the SDN controller and Mininet [94] as the network emulator. Moreover, the analytic engine sFlow-RT analyzer [98] is also used. The Figure 5.1 illustrates the logical testbed design of this dissertation.

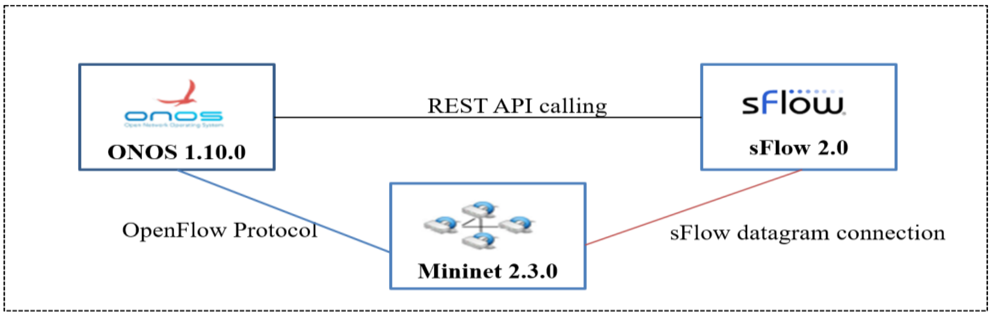


Figure 5.1 Logical Testbed Design

In Figure 5.1, the ONOS controller is connected to Mininet network emulator via OpenFlow protocol. The sFlow agent is run on each switch of Mininet topology by sFlow-RT analyzer. These agents sent sFlow metrics to sFlow collector by using sFlow datagram connection. App-TE got the active flows events from sFlow analyzer by REST API calling.

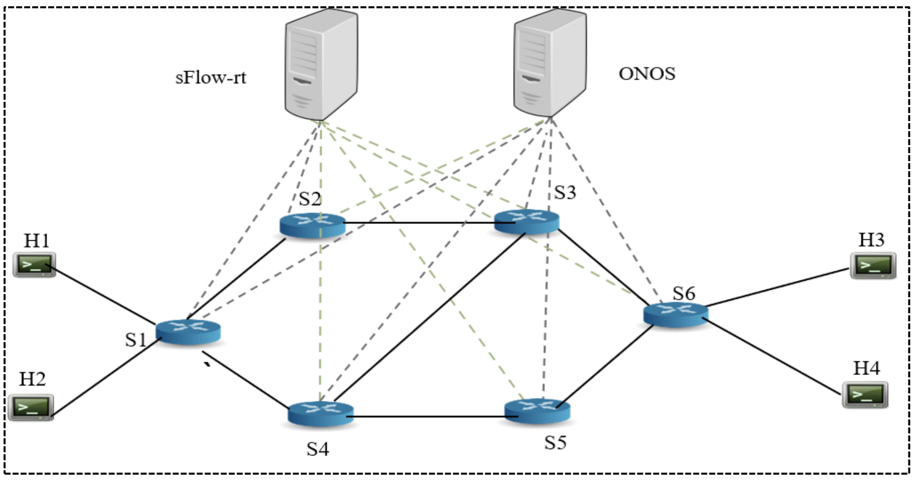


Figure 5.2 The Physical Testbed Design of Classical-Y Topology

According to the literature reviews of application-aware engineering in SDN, the classical-Y topology is widely used and depicted as Figure 5.2. All the switches/routers are connected to SDN controller (ONOS) and traffic analyzer (sFlow-RT). The hardware requirements of the experimental testbed are depicted as Table 5.1. Table 5.2 depicts the software tools that are used in this dissertation.

Table 5.1 Hardware Requirements of Experimental Testbed

Name	Specifications
CPU	Core TM i5-42100U CPU @2.40GHz
RAM & HDD	8.00GB & 500GB
Operating System	Linux 16.04 LTS
Number of PCs	1

Table 5.2 Software and Tools that Used in this Research.

Software	Versions	Used in
ONOS (SDN Controller)	1.10.0 (Kingfisher)	Implementation and Evaluation
Mininet Emulator	2.2.1	Testbed (Evaluation)
sFlow-RT	2.0-r1121	Implementation and Evaluation
Open vSwitch	2.9.2	Testbed (Evaluation)
OpenFlow Protocol	Version 1.3	Testbed (Evaluation)

5.2 Experimental Methods

The experimental results of the proposed App-TE and other traffic management methods which are conducting with multiple scenarios are presented in this section. To highlight the outcome of the proposed App-TE, the following three methods are compared with different scenarios.

5.2.1 Application-aware TE (App-TE)

The proposed App-TE considered two routing methods. When prioritized application traffic is entered the network, App-TE routed the traffic by using Delay weighted Capacity-aware Routing (DWC-aware Routing). If the incoming traffic is non-prioritized application traffic, App-TE routed the application traffic by using shortest path routing (or) minimum hop-count based routing. To perform such kinds of routing mechanism, App-TE needs to estimate Available Bandwidth (ABW), total path delay, and DWC values.

5.2.2 Link Utilization-aware Routing (LU-Routing)

LU-Routing is one of the types of constrained-aware routing. Its interested constraint or weight value is the link utilization. In this dissertation, Link utilization (LU) is defined as the ABW. When the incoming traffic is entered the network, first, LU-Routing estimated the LU, then calculated maximum LU path. Finally, the traffic is routed through the maximum LU path.

5.2.3 Shortest Path Routing

As explained in chapter 2, this dissertation defined one of the applications of ONOS controller, default forwarding, so called reactive forwarding as the shortest path routing. Shortest path routing is fast and simple. It simply forwarded the traffic through the minimum hop-count path.

5.3 Performance Assessment Parameters

The performance assessment parameters that have used in this dissertation are as follows:

(i) Throughput

The throughput refers to the data rate of successful data that delivered over a communication link. Throughput is measured in bits per second (bps) or sometimes in megabyte per second (MB/s). Alternatively, the throughput is the rate at which data is traversing a link and it can be obtained dividing the total payload over the entire session by the total time (or) time taken that transmitted that data [99]. The throughput can be derived as the Equation (5.1):

$$Throughput = \frac{The\ amount\ of data\ transfer}{Time\ taken} \tag{5.1}$$

(ii) Packet Loss Rate

Packet loss rate refers to the ratio of number of loss packets to the total number of sent packets. Packet loss rate can be obtained through the following Equation (5.2):

$$PacketLossRate = \frac{(NumberofLossPackets)}{NumberofReceivedPackets} \tag{5.2}$$

5.4 Traffic Generators

The App-TE used different kinds of application traffic flows such as file transferring, video streaming, haptic stream. To generate these application traffic flows, the following traffic generators are used in this dissertation.

(i) Iperf Traffic Measurement Tool

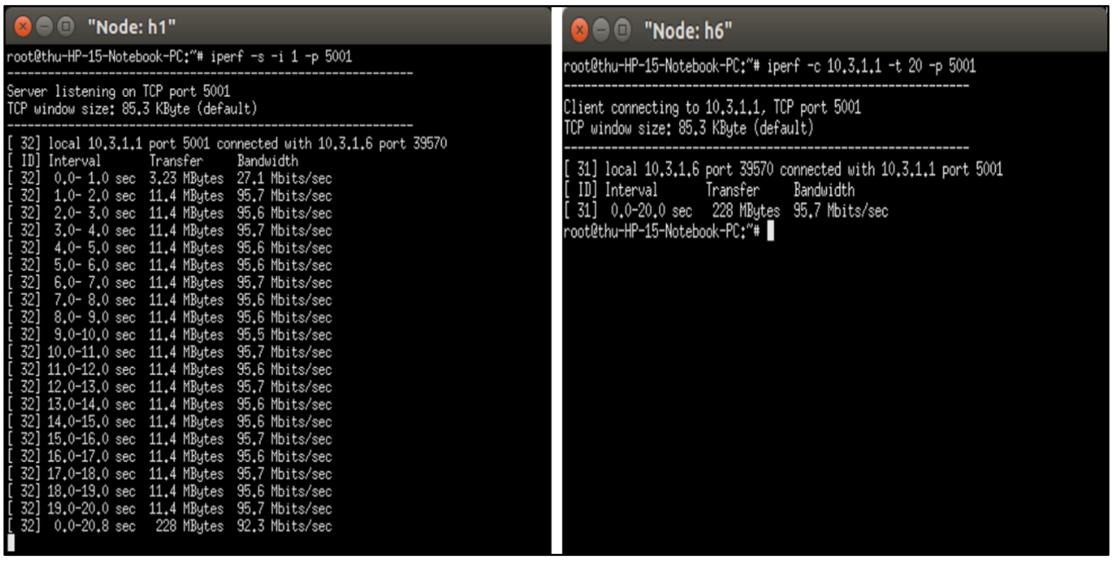
Iperf works as a client-server architecture. By default, Iperf client connects to the Iperf server on the TCP port 5001 and the bandwidth displayed by Iperf is the bandwidth from the client to the server. To generate Iperf TCP traffic, first Iperf server need to run with the following command:

```
$ iperf - s  - i 1 - p 5001
```

where, the arguments for -s denotes the server mode, - i denotes the interval in seconds between periodic bandwidth reports, and - p specifies port number [97]. The iperf client connects to the server by using the following command:

$ iperf - c [server's IP Address] - p 5001 - t 20

where, the arguments for - c indicates client mode and - t specifies the test duration time in seconds. In Figure 5.3, there are two Xterm windows in Mininet emulator. The left one is host H1 and the right one is host H6. Host H1 is used as a server and host H6 is used as a client.



```
root@thu-HP-15-Notebook-PC:~# iperf -s -i 1 -p 5001
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 32] local 10.3.1.1 port 5001 connected with 10.3.1.6 port 39570
[ ID] Interval       Transfer     Bandwidth
[ 32]  0.0- 1.0 sec  3.23 MBytes  27.1 Mbits/sec
[ 32]  1.0- 2.0 sec  11.4 MBytes  95.7 Mbits/sec
[ 32]  2.0- 3.0 sec  11.4 MBytes  95.6 Mbits/sec
[ 32]  3.0- 4.0 sec  11.4 MBytes  95.7 Mbits/sec
[ 32]  4.0- 5.0 sec  11.4 MBytes  95.6 Mbits/sec
[ 32]  5.0- 6.0 sec  11.4 MBytes  95.6 Mbits/sec
[ 32]  6.0- 7.0 sec  11.4 MBytes  95.7 Mbits/sec
[ 32]  7.0- 8.0 sec  11.4 MBytes  95.6 Mbits/sec
[ 32]  8.0- 9.0 sec  11.4 MBytes  95.6 Mbits/sec
[ 32]  9.0-10.0 sec  11.4 MBytes  95.5 Mbits/sec
[ 32] 10.0-11.0 sec  11.4 MBytes  95.7 Mbits/sec
[ 32] 11.0-12.0 sec  11.4 MBytes  95.6 Mbits/sec
[ 32] 12.0-13.0 sec  11.4 MBytes  95.7 Mbits/sec
[ 32] 13.0-14.0 sec  11.4 MBytes  95.6 Mbits/sec
[ 32] 14.0-15.0 sec  11.4 MBytes  95.6 Mbits/sec
[ 32] 15.0-16.0 sec  11.4 MBytes  95.7 Mbits/sec
[ 32] 16.0-17.0 sec  11.4 MBytes  95.6 Mbits/sec
[ 32] 17.0-18.0 sec  11.4 MBytes  95.7 Mbits/sec
[ 32] 18.0-19.0 sec  11.4 MBytes  95.6 Mbits/sec
[ 32] 19.0-20.0 sec  11.4 MBytes  95.7 Mbits/sec
[ 32]  0.0-20.8 sec   228 MBytes  92.3 Mbits/sec
```

```
root@thu-HP-15-Notebook-PC:~# iperf -c 10.3.1.1 -t 20 -p 5001
------------------------------------------------------------
Client connecting to 10.3.1.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 31] local 10.3.1.6 port 39570 connected with 10.3.1.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 31]  0.0-20.0 sec   228 MBytes  95.7 Mbits/sec
root@thu-HP-15-Notebook-PC:~#
```

Figure 5.3 Generating Iperf Traffic between H1 and H6

(ii) Web-based Video Streaming by HTTP

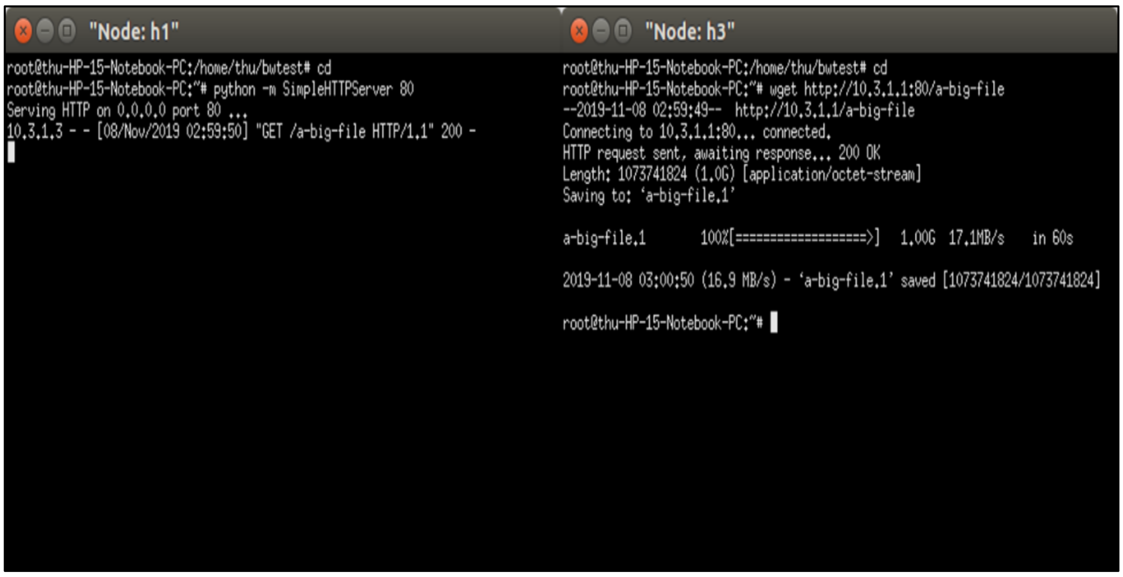
Cisco experts predict 80% of Internet traffic will be video traffic by 2022 [95]. Therefore, App-TE defined video traffic as prioritized-application traffic. In this work, App-TE used web-based video streaming. The Hyper Text Transfer Protocol (HTTP) is a stateless application-level request/response protocol. HTTP follows a classical client-server model. A client opens a connection to make a request, then waits until it receives a response [105]. To perform this, the python SimpleHTTP module is applied as a web server and web clients. The following command is used to implement SimpleHTTP server on port 80:

$ python –m SimpleHTTPServer 80

To get the contents (e.g. videos, images, documents, files, folders) from the server, the client used the following command:

$ wget http://[server IP Address]:80/[file name]

The clients must specify the server's IP address, port number, and file name for this downloading process. Figure 5.4 shows the client (H3) downloading a file from the SimpleHTTP server (H1).



```
root@thu-HP-15-Notebook-PC:/home/thu/bwtest# cd
root@thu-HP-15-Notebook-PC:~# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
10.3.1.3 - - [08/Nov/2019 02:59:50] "GET /a-big-file HTTP/1.1" 200 -
```

```
root@thu-HP-15-Notebook-PC:/home/thu/bwtest# cd
root@thu-HP-15-Notebook-PC:~# wget http://10.3.1.1:80/a-big-file
--2019-11-08 02:59:49--  http://10.3.1.1/a-big-file
Connecting to 10.3.1.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1073741824 (1.0G) [application/octet-stream]
Saving to: 'a-big-file.1'

a-big-file.1        100%[===================>]   1.00G  17.1MB/s    in 60s

2019-11-08 03:00:50 (16.9 MB/s) - 'a-big-file.1' saved [1073741824/1073741824]

root@thu-HP-15-Notebook-PC:~#
```

Figure 5.4 Generating HTTP Traffic Between H1 and H3

(iii) File Transferring with FTP

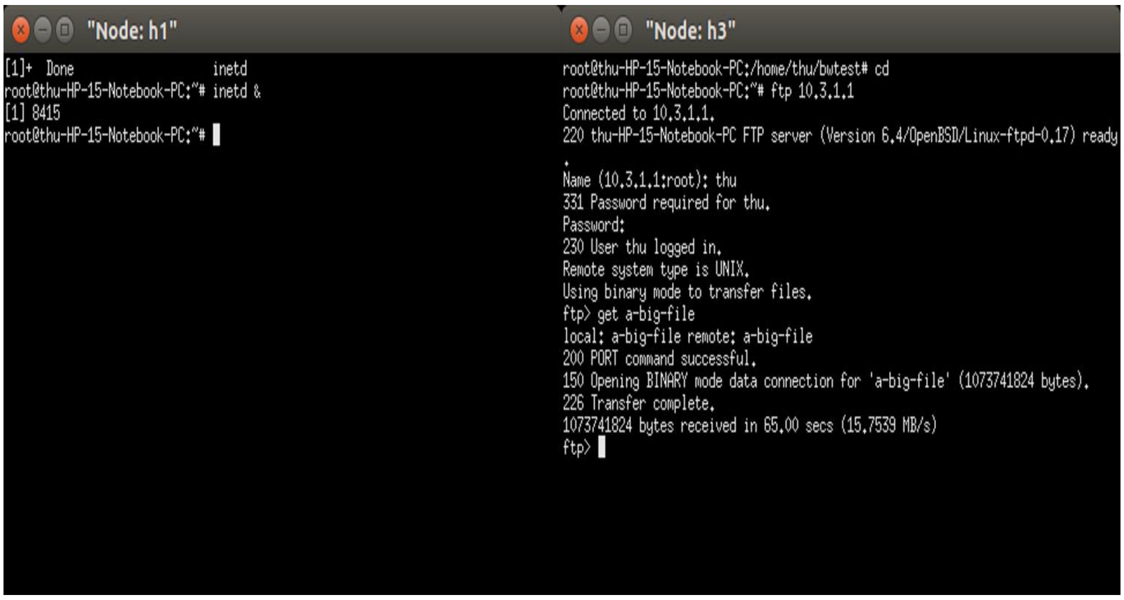
The File Transfer Protocol (FTP) is a standard network protocol used for file transferring between a server and clients on a computer network [93]. The following commands are used to apply FTP server and clients:

$ inetd &

$ ftp [server IP Address]

FTP also protects the username and password, therefore after accessing the correct username and password, **ftp** prompt will appear and can get any file from the server directory. Figure 5.5 shows the file downloading process of FTP client (H1) from the FTP server (H3).

$ ftp> get [filename]



```
[1]+  Done                    inetd
root@thu-HP-15-Notebook-PC:~# inetd &
[1] 8415
root@thu-HP-15-Notebook-PC:~# 
```

```
root@thu-HP-15-Notebook-PC:/home/thu/bwtest# cd
root@thu-HP-15-Notebook-PC:~# ftp 10.3.1.1
Connected to 10.3.1.1.
220 thu-HP-15-Notebook-PC FTP server (Version 6.4/OpenBSD/Linux-ftpd-0.17) ready
.
Name (10.3.1.1:root): thu
331 Password required for thu.
Password:
230 User thu logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> get a-big-file
local: a-big-file remote: a-big-file
200 PORT command successful.
150 Opening BINARY mode data connection for 'a-big-file' (1073741824 bytes).
226 Transfer complete.
1073741824 bytes received in 65.00 secs (15.7539 MB/s)
ftp> 
```

Figure 5.5 Generating FTP Traffic between H1 and H3

(iv) Haptic Stream by D-ITG

There has been a recent concern in the transmission of multi-modal information over the Internet, and especially the transmission of haptic information [39]. The characteristics and QoS requirements of haptic traffic are outlined in Table 5.3.

Table 5.3 The Parameters Settings of Haptic Traffic

Traffic	Characteristics	QoS requirements
Haptic	Constant packet rate. Transmission rate of 1000 packet/sec. Sensitive to delay and jitter. Packet loss <~ 10% and Jitter <~ 2ms.	Delay <~ 50ms. Throughput ~ 500 kbps to 1 Mbps. Packet loss <~ 10% and Jitter <~ 2ms.

Distributed Internet Traffic Generator (D-ITG) is a platform that can produce traffic at packet level accurately replicating appropriate stochastic processes for both inter departure time with random variable packet sizes (exponential, uniform, normal, etc.). Moreover, D-ITG can analyze network by generating network traffic on a packet by packet basis. There are five main modules in D-ITG: ITGSend (Sending Process), ITGRecv (Receiving processes), ITGLog (Storage server), ITGManager (Remote control manager), ITGDec (Analyzing results) [7].

This work generated the traffic which has a constant packet rate and transmission rate of 1000 packet/sec by using the D-ITG. The transmission rate of haptic media is 1000 MU/s and video is 30 MU/s. The maximum allowable delay for haptic media is 30 to 60 ms and the average bit rate of haptic media is 320 kbps [59]. Then, this scenario generated TCP traffic which average bit rate is 320 kbps as the haptic media stream. To generate such kind of traffic, D-ITG used the following command lines:

```
$ ./ITGSend -T TCP -a <Ip-address> -c 100 -C 10 -t 15000

$ ./ITGRecv

$ ./ITGDec sender.log
```

where, parameters -T specifies protocol such as TCP or UDP. -a denotes destination IP address. -c defines the constant payload size which measure by bytes. -C specified constant packet rate (pps) and -t defined the duration (ms).

5.5 Experiment Topology

The experimental test topology is depicted in Figure 5.6. This Figure includes six switches and six hosts that connected to each switch.

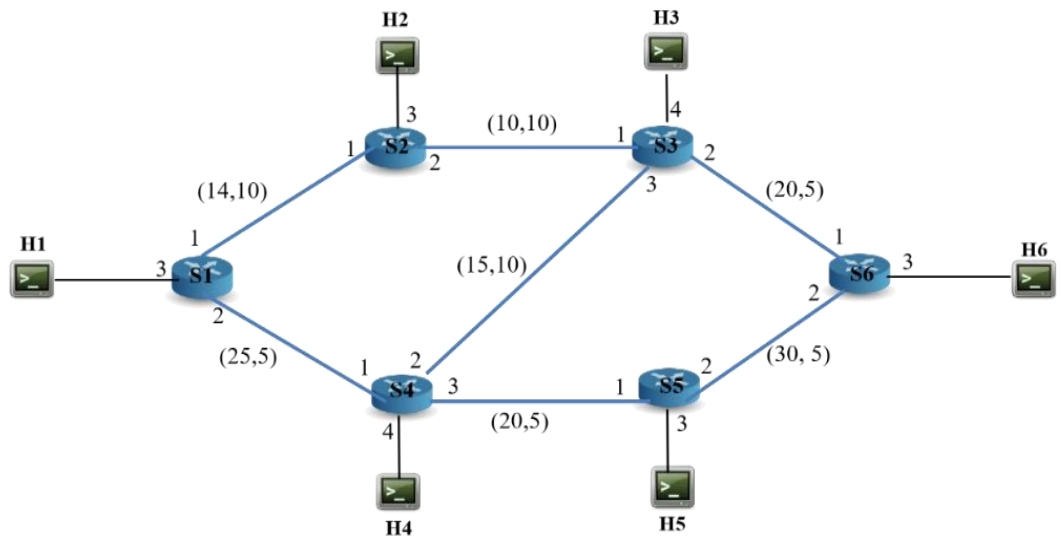


Figure 5.6 Test Topology

The number (1 2, 3, and 4) written at each switch denotes the port numbers for the switch. The two numbers (e.g. (14, 10)) on each link represent the link capacity (Mbps) and the link delay (ms), respectively.

5.6 Experimental Scenarios

Five different experimental scenarios have been carried out in this work. The scenarios are chosen based on the ideas that highlighted the advantage of App-TE. These scenarios are conducted according to Tables 5.1 and 5.2 and tested with Figure 5.6, test topology. The experimental results are calculated based on the average results of five running time.

5.6.1 Scenario I: Analyzing Methods with Iperf

In scenario I, experiments are carried out according to the parameter settings that described in Table 5.4 and testing with test topology, Figure 5.6. This scenario aimed to discuss which methods choose which paths based on its constraints and analyze their performance results.

Table 5.4 The Experimental Parameters Values for Scenario I

Scenario I	Traffic Generator	Iperf Server	Iperf Client	Number of Flows	Run time duration
Test 1	Iperf	H1	H2, H3, H4, H5, H6	5	20 s (one by one)
Test 2	Iperf	H1	H2, H3, H4, H5, H6	5	Parallel

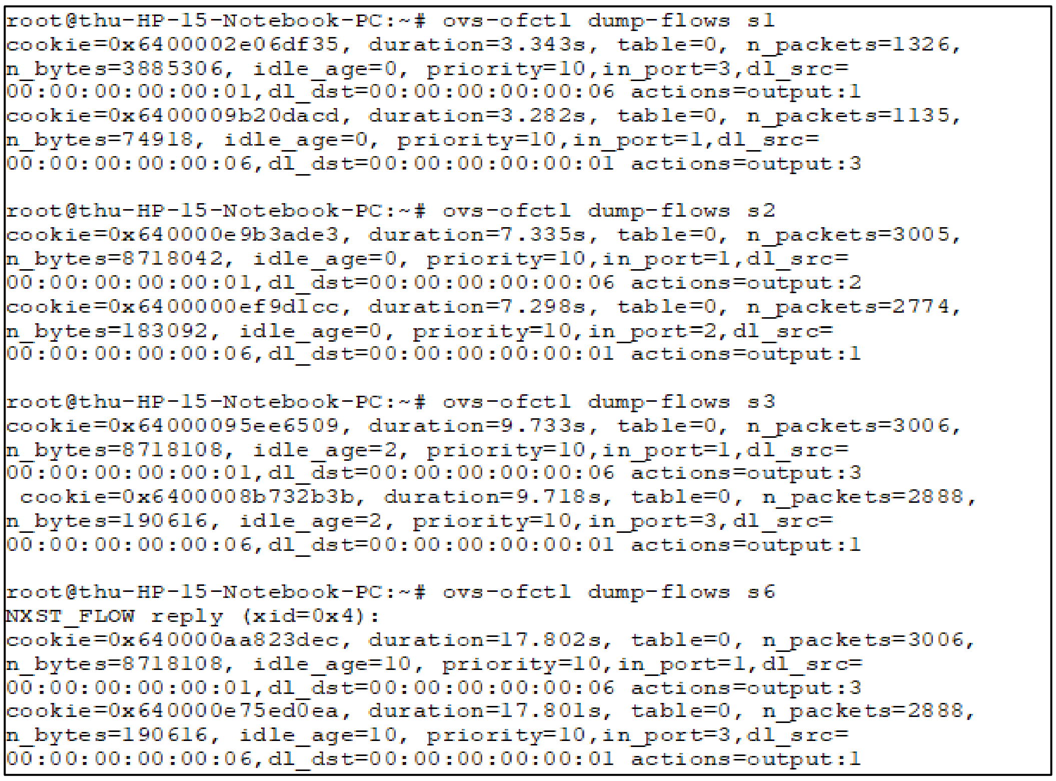
In test 1, host H1 sends Iperf TCP traffic to hosts H2, H3, H4, H5, and H6 for 20s. Then, host H1 parallel sends 5 numbers of Iperf TCP flows to hosts H2, H3, H4, H5, and H6 in test 2. As the sample path calculation, Table 5.5 describes the possible available paths between hosts H1 and H6. There are three shortest available paths for shortest path routing with hop counts constraints.

The LU-aware routing and App-TE extract paths from Depth First Search (DFS) graph. The four possible available paths with LU and DWC constraints for LU-aware and App-TE are also presented in Table 5.5. The constrained values are calculated when generating Iperf traffic between hosts H1 and H6. When host H1 sent Iperf TCP traffic to host H6 by using shortest path routing, H1 forwards the Iperf TCP traffic to H6 through the path p1 = S1–S2–S3-S6. This forwarded path is known by checking flow tables on each switch.

Table 5.5 The Paths and Constraints between Hosts H1 and H6

Source	Destination	Possible Paths	Constraints	Value	Methods
H1	H6	S1-S2-S3-S6	Hop-count	3 hops	Shortest Path Routing
		S1-S4-S3-S6			
		S1-S4-S5-S6			
		S1-S2-S3-S6	Link Utilization	9.9995	LU-aware Routing
		S1-S4-S5-S6		19.995	
		S1-S4-S3-S6		14.9995	
		S1-S2-S3-S4-S5-S6		9.9995	
		S1-S2-S3-S6	Delay Weighted Capacity	0.3887	App-TE
		S1-S4-S5-S6		1.240	
		S1-S4-S3-S6		0.7261	
		S1-S2-S3-S4-S5-S6		0.2429	

The flow table information for the switches are shown in Figure 5.7, where only shows the flow table information of switches S1, S2, S3, and S6 because there is no flow table information in switches S4 and S5.



```
root@thu-HP-15-Notebook-PC:~# ovs-ofctl dump-flows s1
cookie=0x6400002e06df35, duration=3.343s, table=0, n_packets=1326,
n_bytes=3885306, idle_age=0, priority=10,in_port=3,dl_src=
00:00:00:00:00:01,dl_dst=00:00:00:00:00:06 actions=output:1
cookie=0x6400009b20dacd, duration=3.282s, table=0, n_packets=1135,
n_bytes=74918, idle_age=0, priority=10,in_port=1,dl_src=
00:00:00:00:00:06,dl_dst=00:00:00:00:00:01 actions=output:3

root@thu-HP-15-Notebook-PC:~# ovs-ofctl dump-flows s2
cookie=0x640000e9b3ade3, duration=7.335s, table=0, n_packets=3005,
n_bytes=8718042, idle_age=0, priority=10,in_port=1,dl_src=
00:00:00:00:00:01,dl_dst=00:00:00:00:00:06 actions=output:2
cookie=0x6400000ef9d1cc, duration=7.298s, table=0, n_packets=2774,
n_bytes=183092, idle_age=0, priority=10,in_port=2,dl_src=
00:00:00:00:00:06,dl_dst=00:00:00:00:00:01 actions=output:1

root@thu-HP-15-Notebook-PC:~# ovs-ofctl dump-flows s3
cookie=0x64000095ee6509, duration=9.733s, table=0, n_packets=3006,
n_bytes=8718108, idle_age=2, priority=10,in_port=1,dl_src=
00:00:00:00:00:01,dl_dst=00:00:00:00:00:06 actions=output:3
 cookie=0x6400008b732b3b, duration=9.718s, table=0, n_packets=2888,
n_bytes=190616, idle_age=2, priority=10,in_port=3,dl_src=
00:00:00:00:00:06,dl_dst=00:00:00:00:00:01 actions=output:1

root@thu-HP-15-Notebook-PC:~# ovs-ofctl dump-flows s6
NXST_FLOW reply (xid=0x4):
cookie=0x640000aa823dec, duration=17.802s, table=0, n_packets=3006,
n_bytes=8718108, idle_age=10, priority=10,in_port=1,dl_src=
00:00:00:00:00:01,dl_dst=00:00:00:00:00:06 actions=output:3
cookie=0x640000e75ed0ea, duration=17.801s, table=0, n_packets=2888,
n_bytes=190616, idle_age=10, priority=10,in_port=3,dl_src=
00:00:00:00:00:06,dl_dst=00:00:00:00:00:01 actions=output:1
```

Figure 5.7 Flow Table Entries in Switches S1, S2, S3, and S6

Shortest path routing only took account minimum hop counts paths, therefore, when the path p1 does not satisfy the flow demands, the throughput results of shortest path routing may decrease According to the Figure 5.8, when the traffic is sent one by one, the throughput results of shortest path routing may increase and outperform the other two methods. However, when the traffic is sent in parallel, which means all the traffic are parallel using only shortest paths, the throughput results of shortest path has been decreased as shown in Figure 5.9.

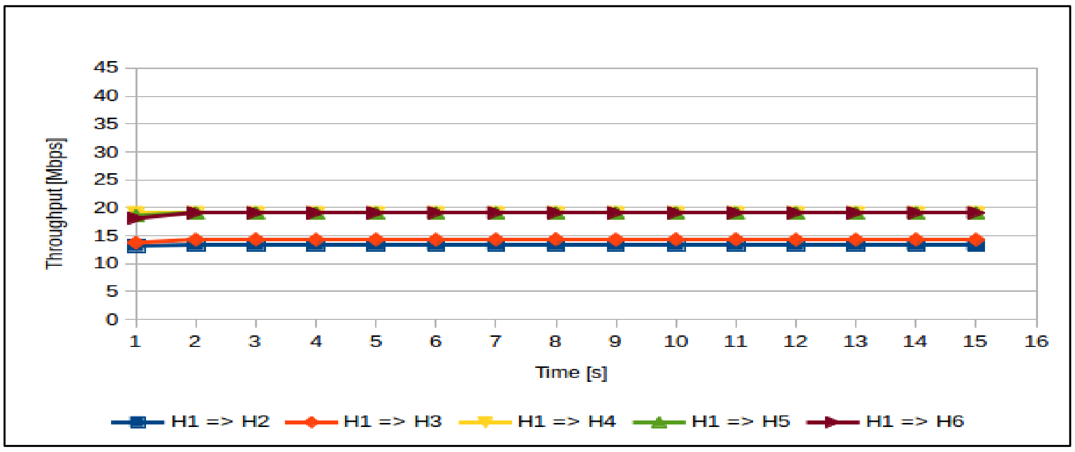


Figure 5.8 Throughput Results for Shortest Path Routing by Test 1 in Scenario I

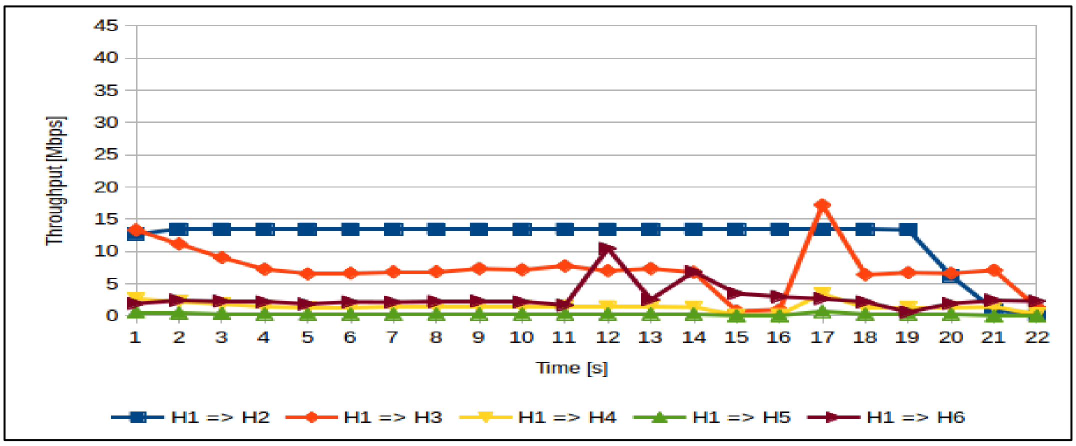


Figure 5.9 Throughput Results for Shortest Path Routing by Test 2 in Scenario I

The throughput results for tests 1 and 2 in scenario I by conducting the LU-aware routing, and App-TE are shown in the Figures 5.10, 5.11, 5.12, and 5.13, respectively. LU-aware routing used the calculated maximum link utilization path and App-TE used the calculated maximum DWC path. The throughput results of LU-aware routing for sending one by one Iperf traffic outperformed the App-TE, however, when the traffic is sent in parallel, the throughput results of App-TE outperformed the other two methods.

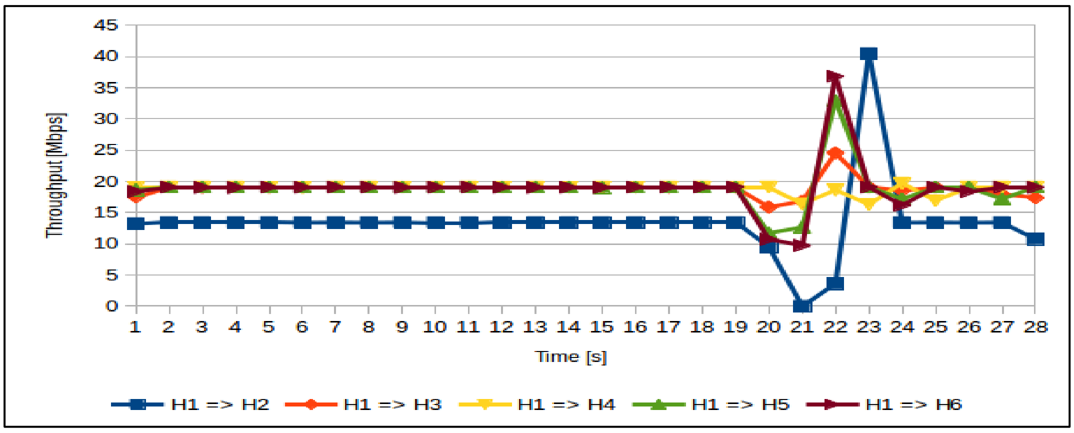


Figure 5.10 Throughput Results for LU Routing by Test 1 in Scenario I

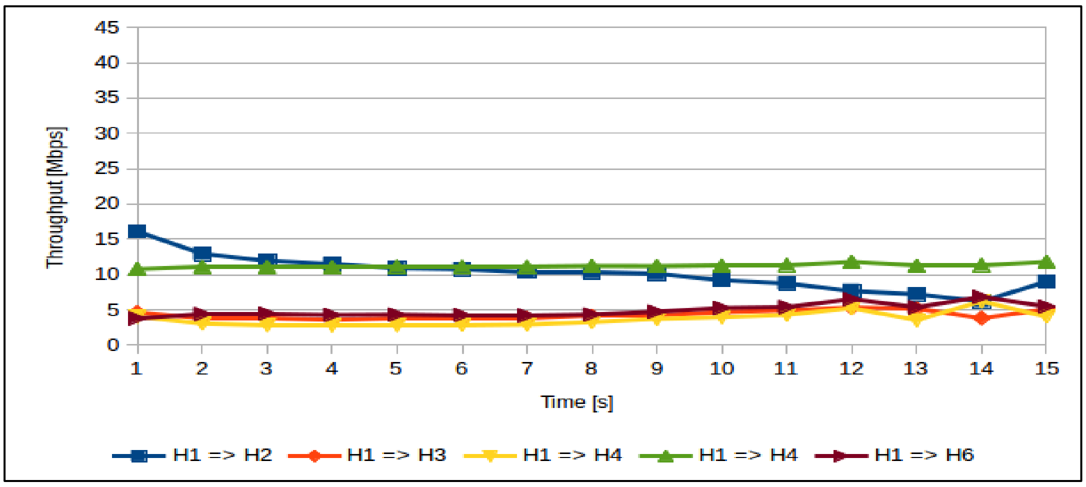


Figure 5.11 Throughput Results for LU Routing by Test 2 in Scenario I

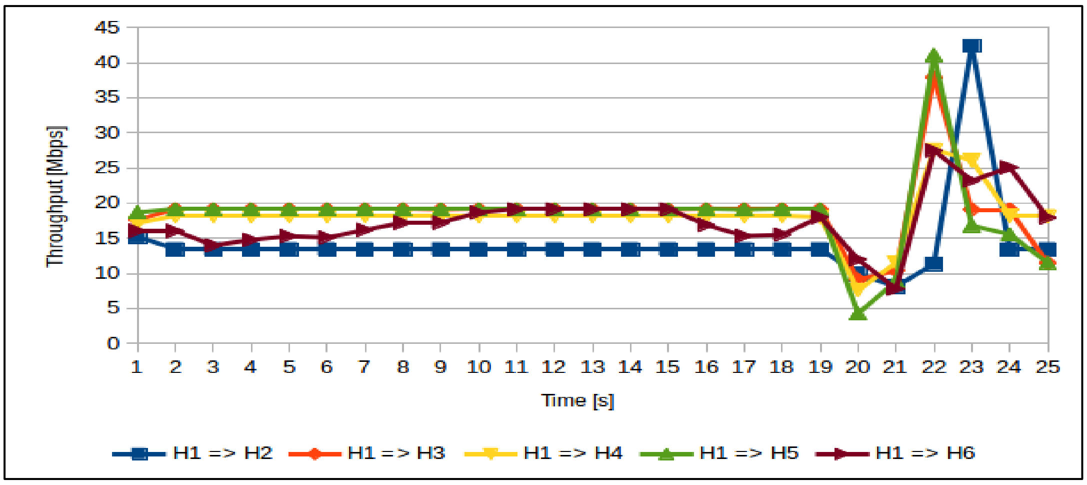


Figure 5.12 Throughput Results for App-TE by Test 1 in Scenario I

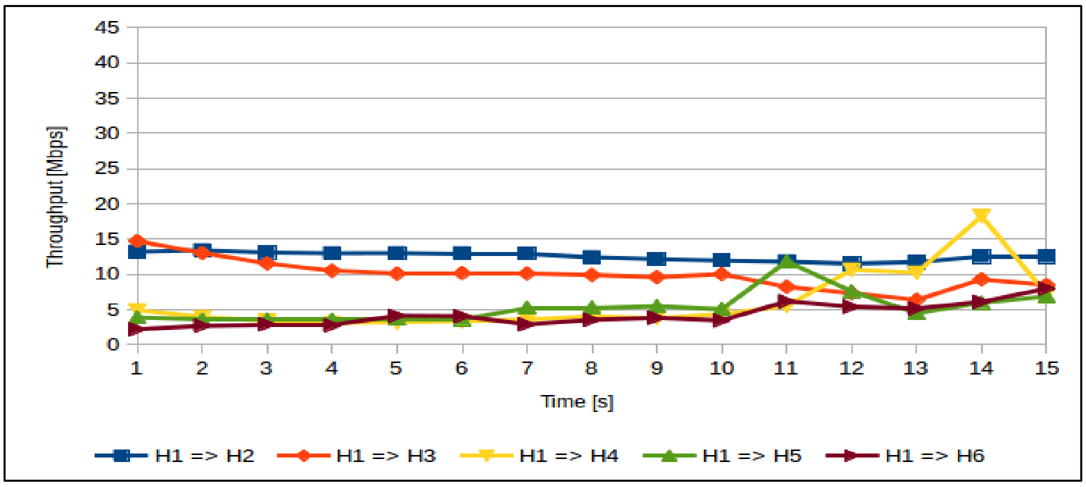


Figure 5.13 Throughput Results for App-TE by Test 2 in Scenario I

5.6.2 Scenarios II: Testing with Larger Link Capacities

In scenario II, experiments are carried out according to the Table 5.6 parameters settings by using test topology, Figure 5.14.

Table 5.6 The Experimental Parameters values for Scenario II

Scenario II	Traffic Generator	FTP Server	FTP Client	Number of Flows	File Size	Run time Duration
Test 1	FTP	H1	H2, H3, H4, H5, H6	25	1G	(one by one) 20s
Test 2	FTP	H1	H6	10	1G	Parallel
		H2	H5			

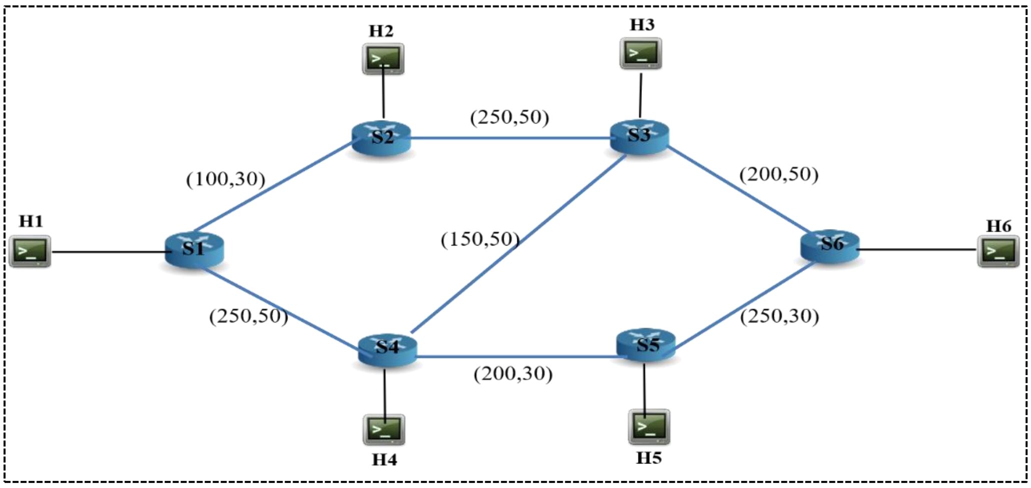


Figure 5.14 Sample Test Topology with Larger Link Capacities

There are 6 hosts and 6 switches in Figure 5.14. The two numbers on each link denote link capacity (Mbps) and link delay (ms). Therefore, the configured link bandwidth for each link ranges from 100 Mbps to 250 Mbps. The configured link delay for each link ranges from 15 ms to 20 ms.

In scenario II, host H1 served as an FTP server and other hosts are served as the clients. Clients have downloaded a file whose size is 1G (1024Bytes). This scenario aims to present how the three methods are handled the FTP traffic, to prove App-ware TE can handle the FTP traffic, and to be conducted with the larger amount of configured link bandwidth.

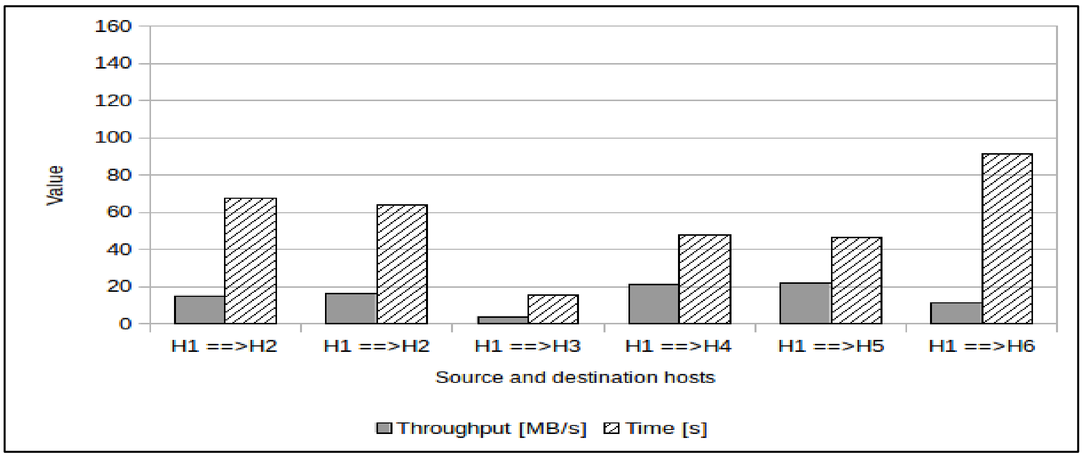


Figure 5.15 Throughput and Time Results of Test 1 in Scenario II by Shortest Path Routing

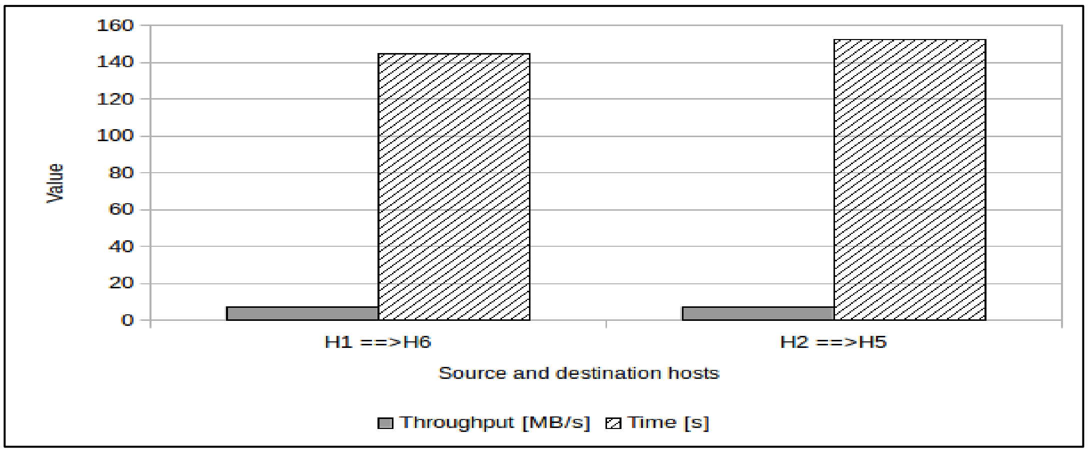


Figure 5.16 Throughput and Time Results of Test 2 in Scenario II by Shortest Path Routing

Figures 5.15 and 5.16 depict the throughput and time results for accessing FTP servers by shortest path routing. These Figures show the analysis results of when FTP clients downloading a file (1G) from an FTP server.

In these experimental results of the figures, the throughput bar shows which Mbps are used to download the file from the server. The time bar represents the duration to accomplish the file downloading process.

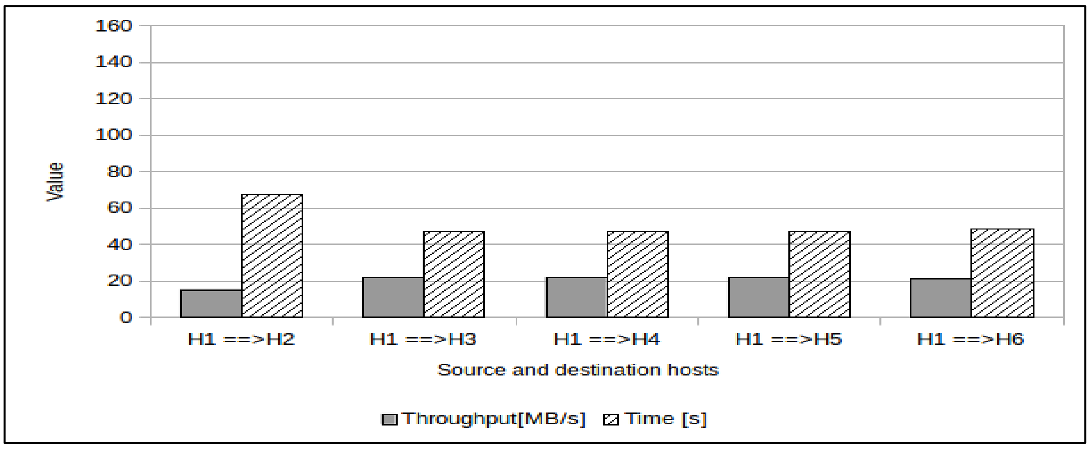


Figure 5.17 Throughput and Time Results of Test 1 in Scenario II by LU-aware Routing

To parallel accessing, this scenario generated cross traffic between hosts H1, H2, H5, and H6. For this test, hosts H1 and H2 served as the FTP servers and hosts H5 and H6 performed as the clients. The Figures 5.17, 5.18, 5.19, and 5.20 depict the throughput and time results for accessing FTP servers in tests 1 and 2 by using LU-aware routing and App-TE.

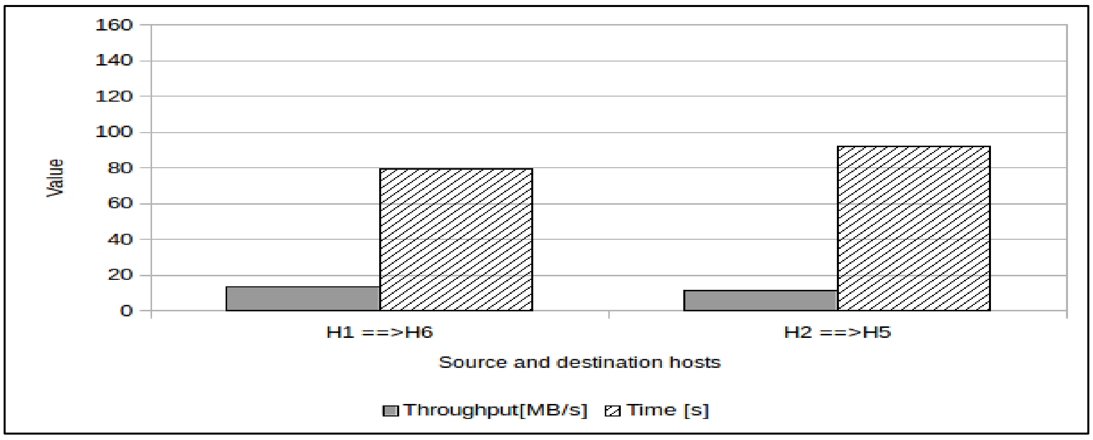


Figure 5.18 Throughput and Time Results of Test 2 in Scenario II by LU-aware Routing

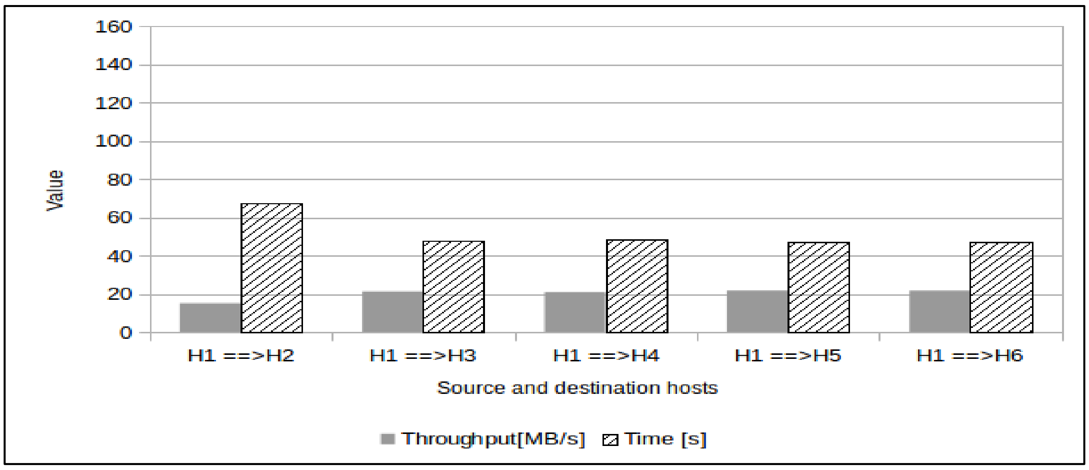


Figure 5.19 Throughput and Time Results of Test 1 in Scenario II by App-TE

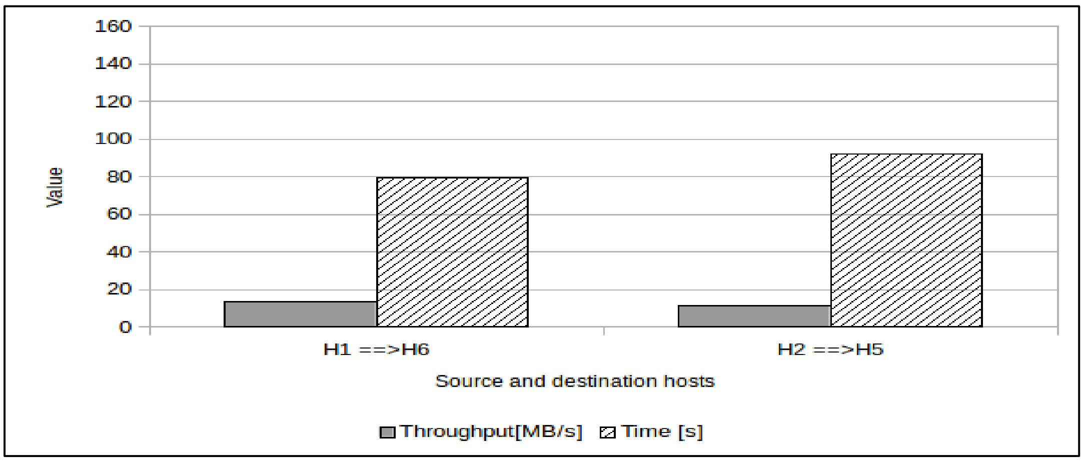


Figure 5.20 Throughput and Time Results of Test 2 in Scenario II by App-TE

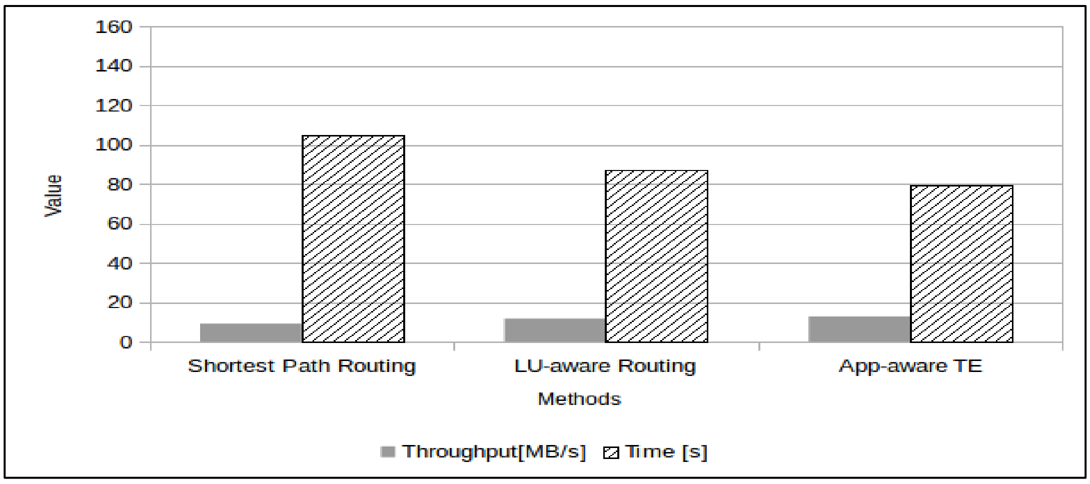


Figure 5.21 Throughput and Time Results of Test 2 in Scenario II by the Three Methods

Figure 5.21 summarized the throughput and time results for parallel accessing FTP servers (H1 and H2) from the clients (H5 and H6). According to this Figure 5.21, shortest path routing has downloaded a file with lower throughput but taken highest amount of time. LU-aware routing has gotten higher throughput than the shortest path routing. The proposed App-TE and LU-aware routing got nearly the same throughput results and their significant difference is 1.146. The throughput results of App-TE is higher than LU-aware routing. Moreover, App-TE took less time to accomplish this FTP file downloading.

5.6.3 Scenario III: Testing with Different Application Traffic Flows

Scenario III conducted different application traffic flows by using three different methods. This scenario is running in sample test topology as shown in Figure 5.14. This scenario also used different application traffic flows such as, video streaming and haptic stream.

Table 5.7 Experimental Parameters for Scenario III

Application traffic flows	Options
Video streaming	VLC Server (H1), VLC Client (H6)
Haptic stream	Generated by using D-ITG

The main goal of this test is analyzing how the three methods perform to conduct these application traffic flows. The experimental parameters are as shown in Table 5.7.

Table 5.8 Parameters Settings

Video File Settings		VLC Streaming Settings	
Parameters	Values	Parameters	Values
Video file type	MPEG-4 (mp4)	Video Codec	H.264+MP3 (MP4)
Size	101.1MB (101,123,423 bytes)	Bit Rate	128Kbps
Duration	11 minutes 25 seconds	Sample Rate	44100KHz
Dimension	1280×720	Encapsulation	MP4/MOV
Codec	H.264	Protocol Type	HTTP

To apply video streaming, this test used VLC server and client. Host H1 served as the VLC server which stream the video to the VLC client host H6. Table 5.8 depicted as the parameter setting of the VLC software and streamed video file.

The streamed video file size is 101.1 MB and the duration for this video is 11 minutes and 25 seconds. VLC server streamed this video by using HTTP protocol. This scenario also analyzed packet loss rate of video streaming by applying the three methods (shortest path routing, LU-aware routing, App-TE). To analyze packet loss rate of video streaming, this scenario checked packet retransmission of TCP connection.

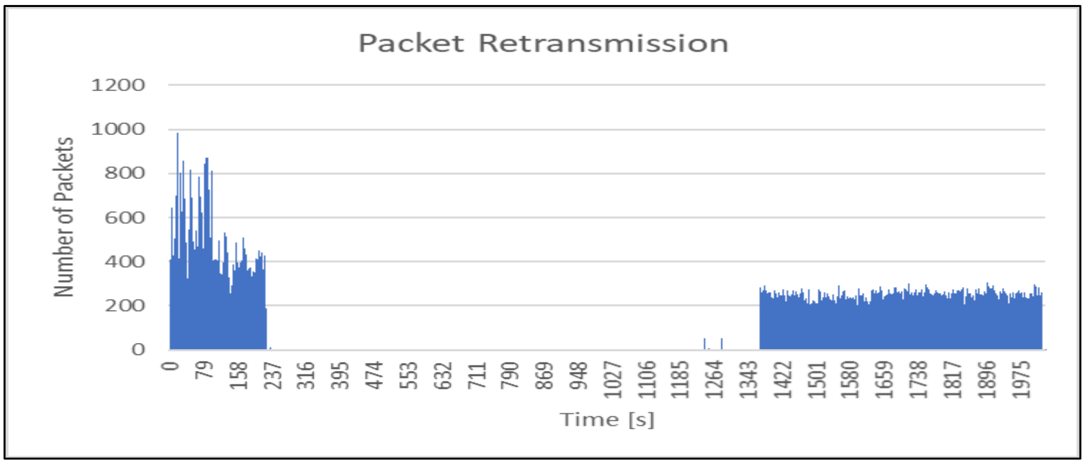


Figure 5.22 Video Streaming with Shortest Path Routing

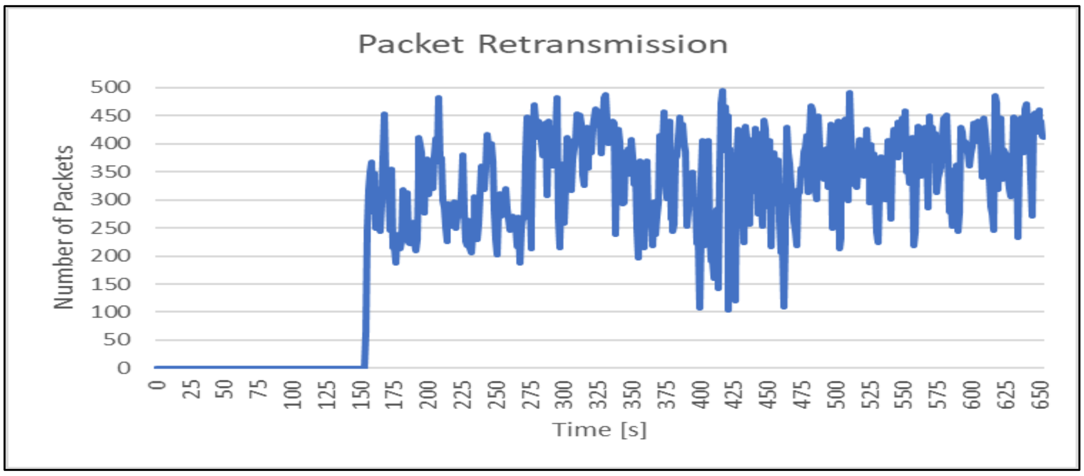


Figure 5.23 Video Streaming with LU-aware Routing

Figures 5.22, 5.23, and 5.24 showed the occurrence of packet retransmission, alternatively, the occurrence of packet loss when the VLC server H1 was streaming video file to VLC client H6. According to the packet retransmission analytical results, the shortest path routing got the highest number of retransmitted packets (nearly 1000 packets) and the packet loss rate is nearly 20% as shown in Figure 5.22.

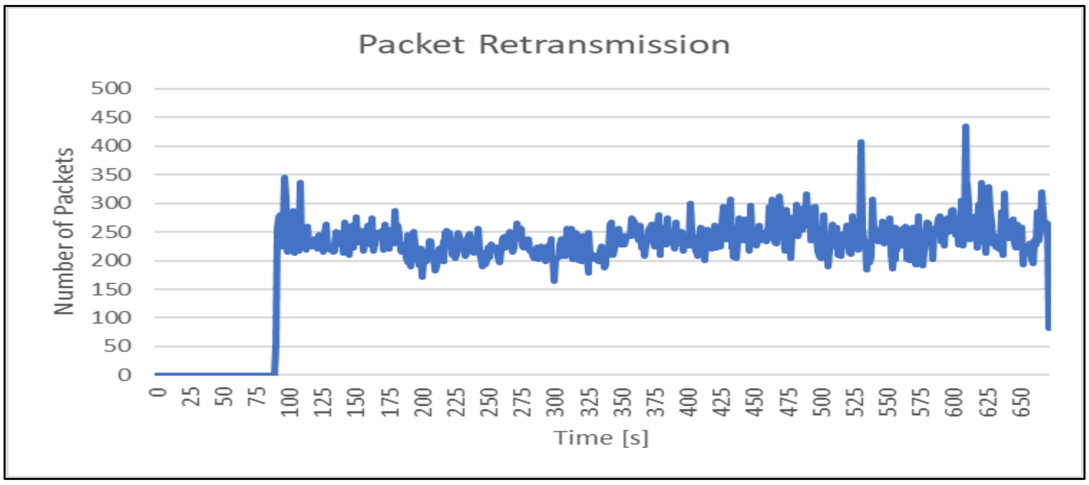


Figure 5.24 Video Streaming with App-TE

When video streaming is conducted by LU-aware and App-TE, the highest number of retransmitted packets are nearly 500 packets and less than 450 packets, respectively. APP-TE not only got a smaller number of retransmitted packets than the others but also got less time duration to transmit the video file.

To conducted the haptic stream, scenario III used D-ITG. Figure 5.25 showed the throughput versus elapsed time of haptic streams by applying the three methods. According to the Figure 5.25, the average throughput result of App-ware TE outperformed the others two methods.

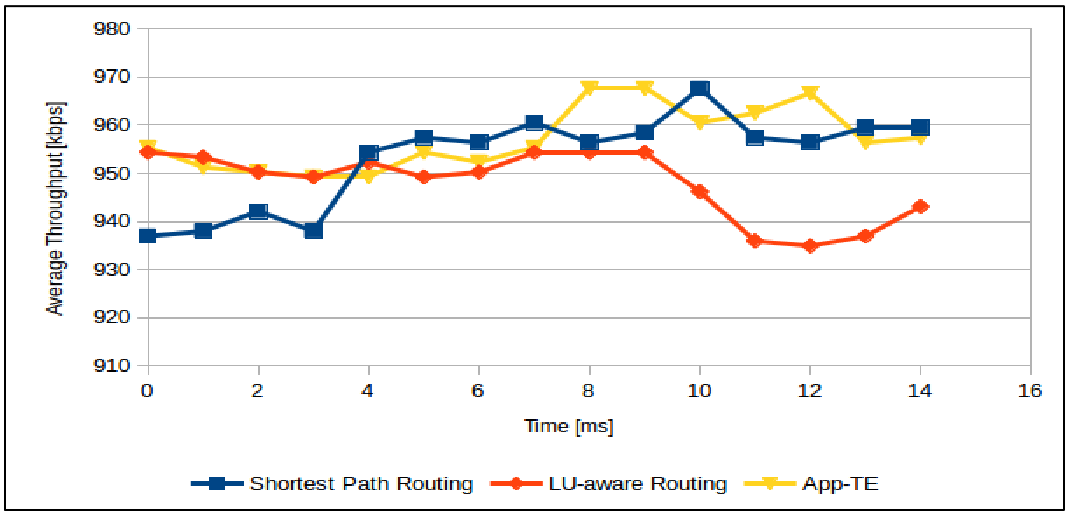


Figure 5.25 Comparative Throughput Results by the Three Methods

5.6.4 Scenarios IV: Testing with and without Application-awareness

In scenario IV, experiments are carried out according to the parameter settings of Table 5.9 by testing with Figure 5.14. This scenario aims to present the effectiveness of with and without application awareness.

Table 5.9 The Experimental Parameters Values for Scenario IV

Application traffic flows	Server	Clients	Number of flows
Prioritized-application traffic: file transferring, video streaming, haptic stream	H1 H2	H6 H5	6
Non prioritized-application traffic: Iperf	H1 H2	H6 H5	6

In scenario IV, 4 types of 12 traffic flows are parallelly generated and conducted by using the following three methods: shortest path routing, LU-aware routing, and App-ware TE. For file transferring, sending a huge file whose size is 1G. For video streaming, a video file whose size is 101.1 MB. Moreover, this scenario also generated haptic stream and Iperf traffic. Whenever the prioritized- or non prioritized-application traffic flows are entered the network, the shortest path routing always forwarded the traffic flow through the shortest path. LU-aware routing forwarded through the maximum link utilization path.

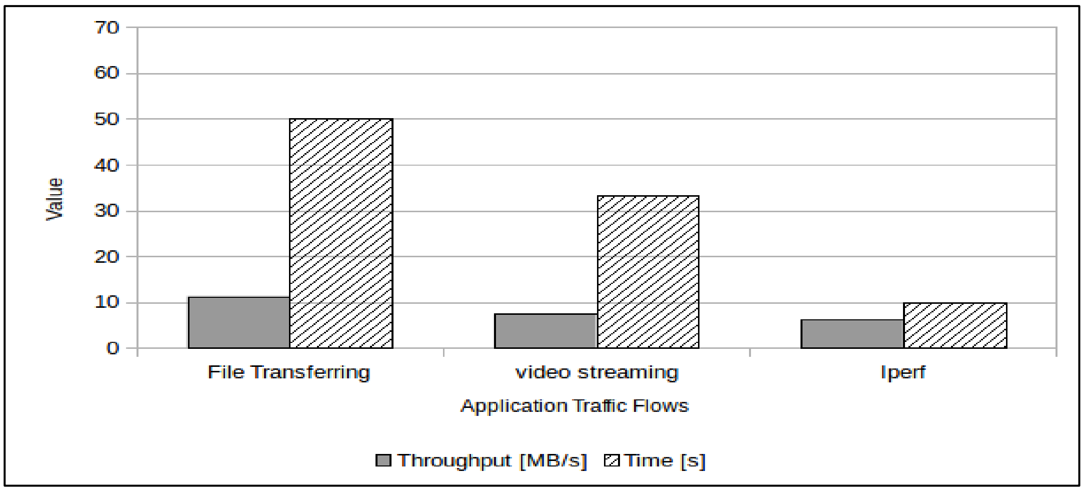


Figure 5. 26 Throughput and Time Results of Scenario IV by Shortest Path Routing

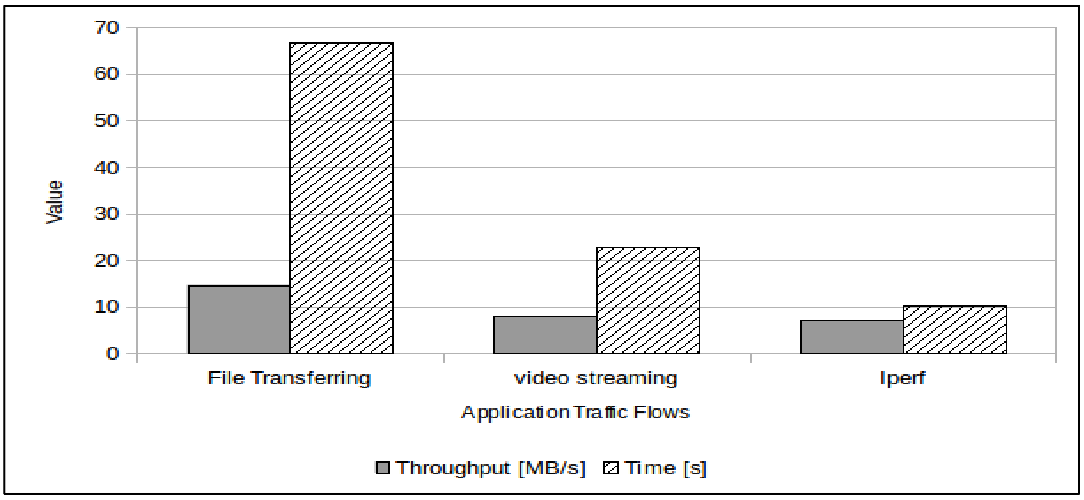


Figure 5. 27 Throughput and Time Results of Scenario IV by LU-aware Routing

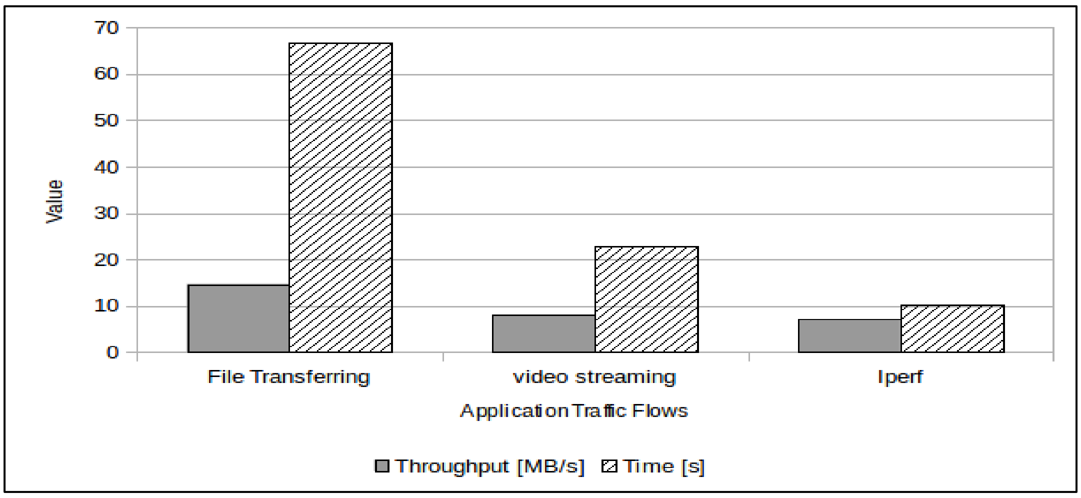


Figure 5. 28 Throughput and Time Results of Scenario IV by App-TE

The goal of App-TE is to find the optimal path for prioritized-traffic to maintain the quality of this traffic. In App-TE, the prioritized-traffic flows are forwarded through the maximum DWC path and non-prioritized traffic flows are simply forwarded through the minimum hop-count paths. Figures 5.26, 5.27, and 5.28 described the throughput and time results of prioritized and non-prioritized application traffic flows conducted by the shortest-path routing, LU-aware routing, and App-TE, respectively.

In Figure 5.29, App-TE has been decreased throughput results for non-priorities traffic (Iperf) than the other two methods. The LU-aware routing got better throughput results for non-priorities traffic (Iperf) than the shortest path routing and App-TE. The shortest path routing got the least throughput results than the other two methods. As the experiment results in Figure 5.29, the throughput results of prioritized-application traffic flows (file transferring, video streaming, haptic streaming) conducted by the App-TE outperformed the other two methods.

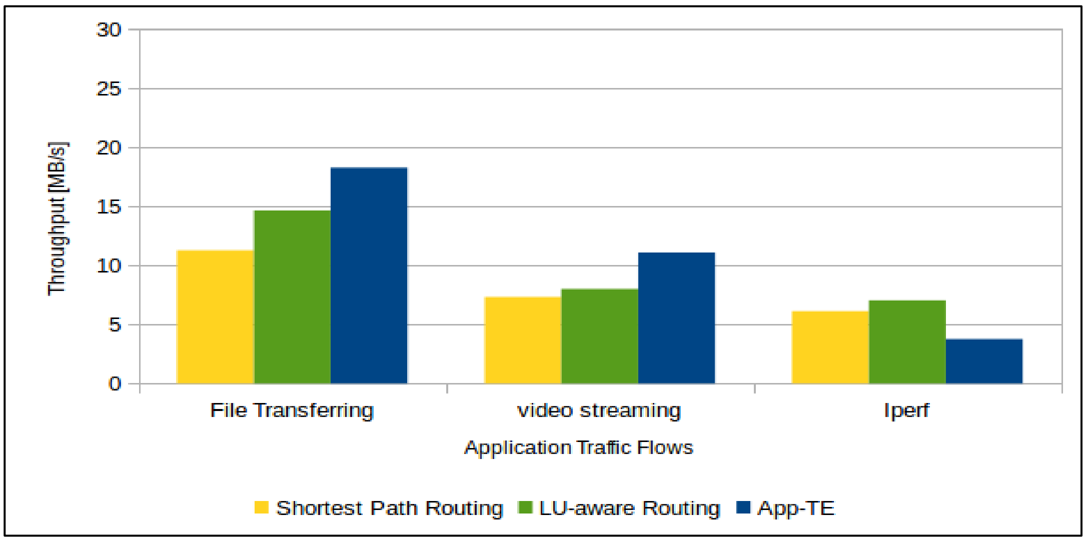


Figure 5.29 Comparative Throughput Results for the Three Methods

5.6.5 Scenarios V: Testing with Different Topologies

In scenario V, experiments are carried out according to the parameter settings of Table 5.10 by testing with different topologies such as leaf-n-spine topology (Figure 5.30) and random topology (Figure 5.31). The main objectives of this scenario are to analyze the performance of the three methods in different topologies and present the effectiveness of with and without application awareness in different topologies.

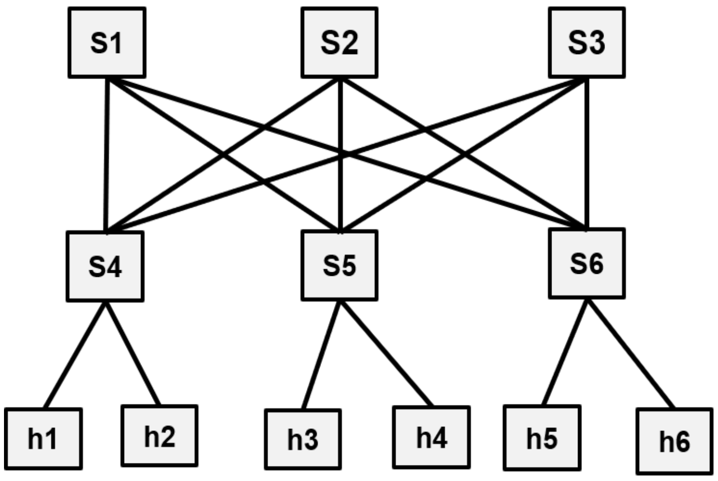


Figure 5. 30 Leaf-n-Spine Topology

Moreover, scenario V compared the topologies that contained multiple equal cost paths and less equal-cost paths. In this scenario, different types and sizes of the application traffic flows are simultaneously generated in random order. These analytical results are calculated by using the average results of six running times.

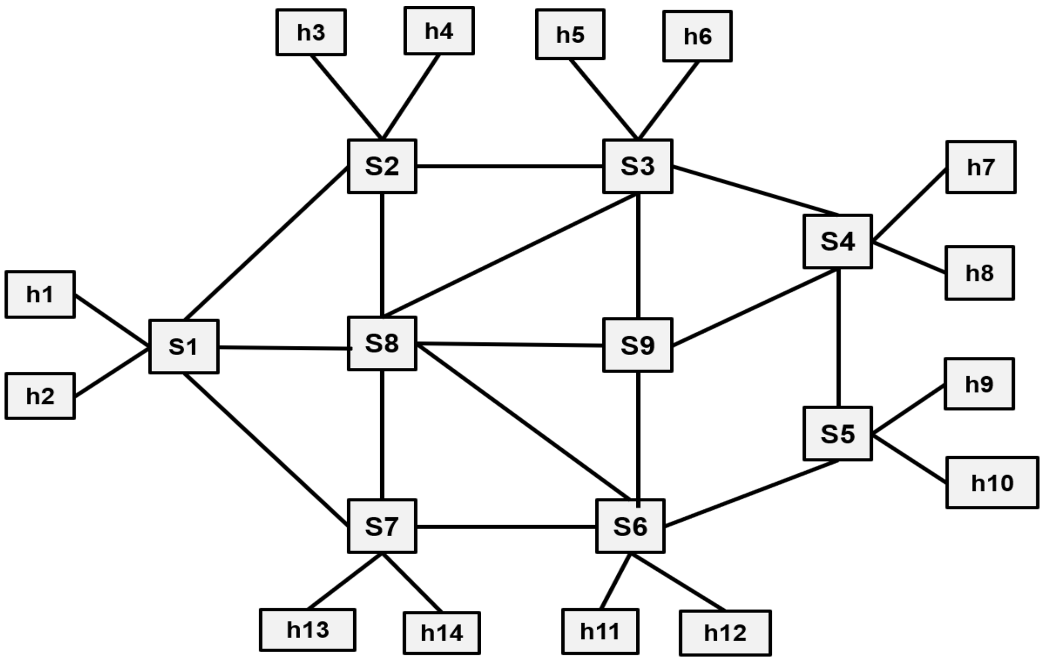


Figure 5. 31 Random Topology

Table 5.10 Topology Settings

Parameters Settings	Leaf-n-Spine Topology	Random Topology
Number of switches	6	9
Number of hosts	6	13
Links bandwidth	150 ~ 350 Mbps	200 ~ 450 Mbps
Links delay	10ms	10ms
Application traffic flows	File transferring, video streaming, haptic stream, iperf	

Figures 5.32 and 5.33 depict the comparative throughputs results by the three methods in leaf -n-spine topology and random topology, respectively. App-TE used DFS to construct the network topology graph.

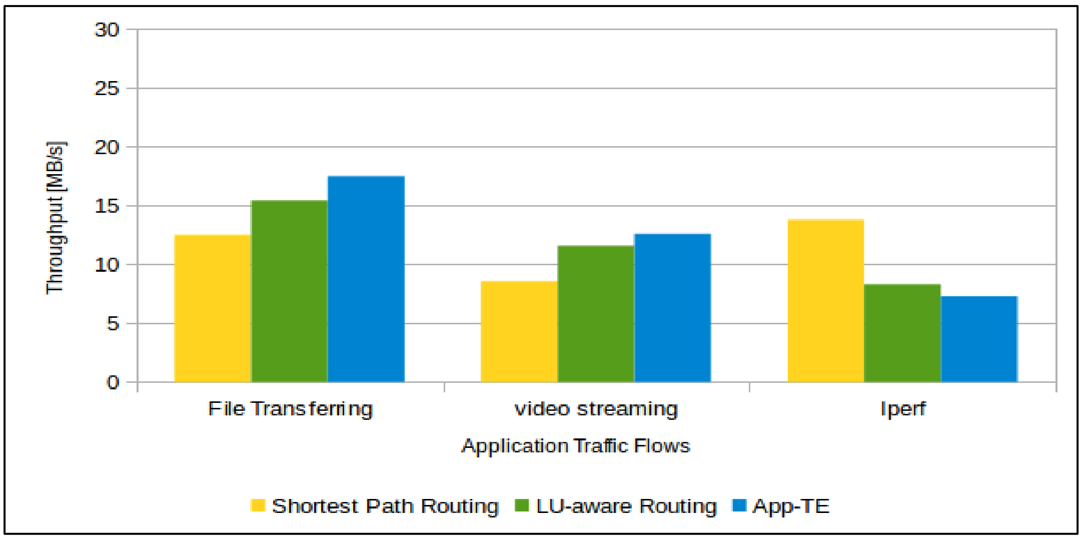


Figure 5.32 Comparative Results of the Three Methods in Leaf-n-Spine Topology

The available paths between source and destination hosts contained all the possible paths rather than disjoint paths and shortest path. That is why, App-TE can perform efficiently in the random topology which involve less equal-cost paths as shown in the experimental results of Figures 5.32 and 5.33.

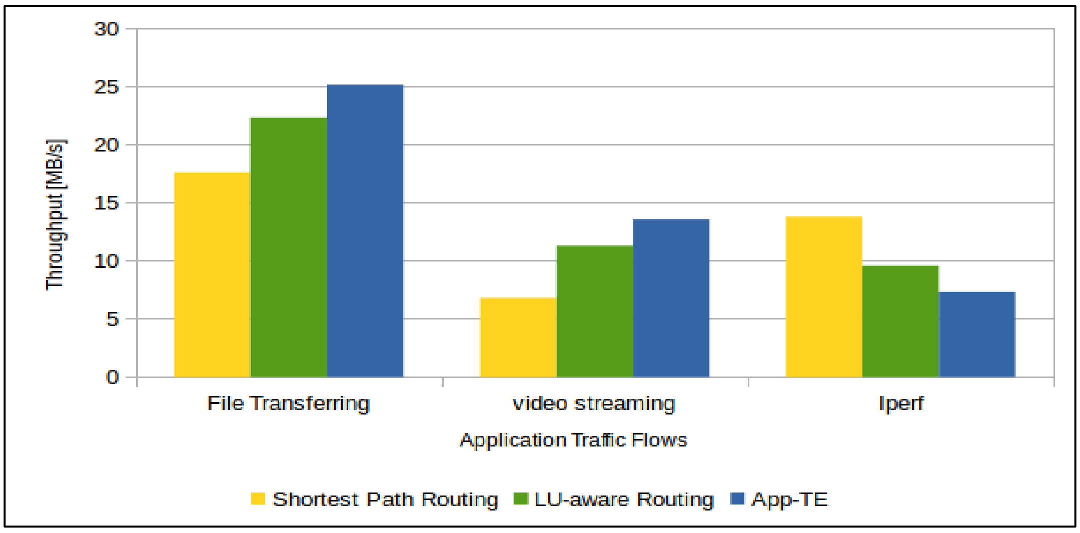


Figure 5.33 Comparative Results of the Three Methods in Random Topology

5.7 Chapter Summary

In this chapter, the experiments are carried out by using different scenarios and analyzed by using different methods. First, to analyze which methods used which paths depending on which constraints, this chapter carried out the testing with scenario I. To analyze how to handle FTP and larger link capacities, the chapter carried out the testing with scenario II. Scenario III tested with different application traffic flows. In scenario IV, two classes of traffic are handled for testing with and without application-awareness of the following three methods: shortest path routing, LU-aware routing, and App-TE. To prove App-TE can apply in different topologies, scenario V tested with App-TE by conducting leaf and spine and random topologies. By analyzing the experimental result of scenarios I, II, III, IV, and V, App-TE got the increased throughput results than the others dealing with prioritized application traffic flows.

CHAPTER 6

CONCLUSION AND FUTURE WORK

The final chapter concluded the proposed work, App-TE by describing the summary of the dissertation, advantages and limitations, and future work.

6.1 Summary of Dissertation

Effective traffic management solution is required not only to harmonize the dramatic growth of networks but also to improve network performance and to support the best services to the user. The equal treatment of all classes of applications is not the proper way to meet user's application-level requirements because the resource requirements of network applications are varied. Different applications have different application requirements. Application-aware engineering is one of the effective traffic management solutions which computed paths based on the application requirements such as bandwidth, network delay, jitter, and so on.

The legacy networks struggle to perform such complex application-aware engineering tasks. It cannot provide complex the complexity of control protocols and interconnecting of a large number of smart devices and it also leads to limited innovations for both management and configuration aspects. Software Defined Networking (SDN) provides an effective traffic management solution by separating control and data planes, global centralization control, and being programmable.

And, the traditional shortest path routing cannot provide effective traffic engineering because it only aware the shortest path. The constraint-aware routing is more efficient than the traditional shortest path routing, however, it needed to estimate constraints such as link capacity, delay, jitter, and so on. This constraint-aware routing cannot guarantee future traffic demands and the prioritized traffic flows.

To overcome such kinds of issues, this dissertation proposed effective application-aware traffic engineering in SDN which contributes the following three main modules: traffic classification, measurement, and management. And, the dissertation described the proposed work with six chapters. In chapter 1, this dissertation was starting to introduce what is application-aware engineering and how important it is. After that, the chapter also presented the motivations and the problem definitions that faced in the three main modules. App-TE aimed to overcome these problems by fulfilling its objectives such as selecting the optimal path for the prioritized application traffic flows. Then, the chapter also mentioned the overall contributions of this work related in three different modules.

To highlight the problems of application-aware traffic engineering in SDN, the dissertation firstly surveyed the issues of traditional IP-based and MPLS-based traffic engineering in chapter 2. Chapter 2 also mentioned some of the surveyed works of traffic engineering in SDN under the scope of flow management, fault tolerance, topology update, and traffic analysis. After that, it presented separately the brief review of previous work for each type of three main modules. In traffic classification, the chapter briefly mentioned the related works of port-based approach, DPI or payload-based approach, and machine learning-based approaches. And, it also mentioned the issues of these approaches. In traffic measurement, the chapter firstly explained the definition of network QoS parameters which represent the current situation of networks, then it also discussed the surveyed of estimation techniques for these parameters especially available bandwidth, end-to-end delay, and link weight. For traffic management, the various methods and approaches of application-aware traffic management in SDN, SDN-based cloud, IoT, and data center networks have been proposed so far. Finally, the chapter concluded by describing the various application-aware traffic engineering in SDN as in Table 2.1 and showed that efficient application-aware engineering techniques are still required to satisfy the user application level requirement.

The background theory for the proposed App-TE was detailed described in chapter 3. Since App-TE implemented in the SDN environment, chapter 3 included the description of the layer taxonomy of SDN architecture which involving infrastructure layer, controller layer, data plane layer, northbound, and southbound protocols. To describe these layers, the chapter also gave examples of each layer for instance; Open vSwitch, its specifications, and processing structure are described in the data plane layer. The ONOS controller for the control layer and the OpenFlow protocol for southbound protocols. Finally, this chapter presented a short overview of flow management approaches (proactive, reactive, and hybrid) and QoS routing methods which have been used in conventional IP and SDN networks.

The overall system architecture of the proposed App-TE is well described in chapter 4. App-TE routed the traffic according to the prioritized classes. Firstly, chapter

4 defined the application traffic as two classes: prioritized and non-prioritized classes. The main objective of App-TE is to route the prioritized traffic through the optimal path. Both bandwidth and delay-sensitive application traffic are included in the prioritized traffic class. Therefore, App-TE considered two different routing methods. When the incoming traffic is entered the network, App-TE classified the traffic by using port and protocol number with the help of a traffic analyzer (sFlow-RT). If the classified traffic is prioritized application traffic, App-TE routed it by using DWC-aware routing. If the non-prioritized traffic is entered the network, App-TE simply routed the traffic through the shortest path. Therefore, step by step process for estimation of available bandwidth, link delay and delay weighted capacity values are also described in chapter 4. Finally, this chapter explained with the detailed process of two different routing methods by using flow diagrams and algorithms.

The experimental design, implementation and evaluation of the App-TE in SDN are presented in chapter 5 with the various types of experimental results. Firstly, the requirements of hardware, software, and tools that are in designing of experimental testbed are described in chapter 5. To highlight the outcome of App-TE, chapter 5 comparatively discussed and analyzed App-TE with other two methods such as Link Utilization aware routing and shortest path routing and it also evaluated and analyzed with five different scenarios. This dissertation used different types of application flows such as video streaming, file transferring, and haptic stream. To conducted this traffic in Mininet network emulator, this chapter also described different traffic generating tools (Iperf, FTP, HTTP, and D-ITG) to generate the traffic as in the real network traffic. The evaluation results are measured by the following parameters: throughput and packet loss. According to the experimental results that conducted with various scenarios in chapter 5, the throughput and packets loss results of App-TE outperformed the other two methods and improved the network utilization.

6.3 Advantages and Limitations

To reroute the non-prioritized applications through the optimal path can occur a lack of network resources for prioritized-applications. The flow management method proposed in this dissertation is so-called App-TE. Consequently, App-TE focused on the prioritized traffic to reroute through the best path. The delay-sensitive and non-delay-sensitive application traffic are involved in the prioritized application traffic class. For this reason, App-TE used the weighted parameters' values of ABW and link delay to choose the best path. For the non-prioritized application traffic, App-TE simply forwarded through the shortest path.

To perform this kind of flow management, first App-TE estimates QoS parameters such as Available Bandwidth (ABW) and link delay. To estimate these QoS parameters, querying statistics information from the switches is needed to perform and this may increase the latency between controller and switches. Moreover, this may increase the controller workload. To overcome this, App-TE queried port statistics from the selected switches that information is provided by the sFlow-RT analyzer via REST API calling instead of querying all the switches in the network.

According to the preliminary experiments of App-TE, the throughput and packet loss rate of App-TE outperformed the other two methods by conducting five different scenarios. To analyze which methods choose which paths based on their constraints, this work used scenario I (section 5.6.1). From this experiment, we observed that all three methods work well and correctly choose their constrained paths. We also know that, when the traffic volume is smaller than the link capacity, the fast and simple shortest path routing got better throughput results than the others because of the other two (App-TE, LU-aware routing) methods incurred the delay time for estimating QoS parameters and installing flow table entries. However, when the traffic is sent in parallel and the traffic volume is larger than the link capacity, the App-TE and LU-aware routing got better throughput results than the shortest path routing.

Not only to present how the three methods handled different application traffic but also to prove App-TE work well with larger link capacity and different application traffic flows (file transferring, video streaming and haptic stream), the performance results of three methods are demonstrated in scenario II and III (5.6.2 and 5.6.3). According to the experimental results of scenario II, the shortest path routing has downloaded a file with lower throughput but taken highest amount of time. LU-aware routing has gotten higher throughput than the shortest path routing. The proposed App-TE and LU-aware routing got nearly the same throughput results and their significant difference is 1.146. The throughput results of App-TE is higher than LU-aware routing. Moreover, App-TE took less time to accomplish the file downloading process.

According to the analytical results of packet retransmission, the shortest path routing got the highest number of retransmitted packets (nearly 1000 packets) and the packet loss rate is nearly 20% when video streaming is conducted in scenario III. When video streaming is conducted by LU-aware and App-TE, the highest number of retransmitted packets are nearly 500 packets and less than 450 packets, respectively. APP-TE not only got a smaller number of retransmitted packets than the others but also got less time duration to transmit the video file.

The performance results of scenario IV analyzed how the three methods are conducted with prioritized and non-prioritized application traffic flows and showed the effectiveness of with and without application awareness. According to these experimental results, App-TE has been decreased throughput results for non-priorities traffic (Iperf) than the other two methods. The LU-aware routing got better throughput results for non-priorities traffic (Iperf) than the shortest path routing and App-TE. The shortest path routing got the least throughput results than the other two methods. When the prioritized application traffic is conducted, the throughput results of App-TE outperformed the other two methods.

To analyze the performance of App-TE with different topologies, scenario V (section 5.6.5) is conducted. This scenario compared the topologies which contained multiple equal-cost paths and less equal-cost paths. According to the experimental results, the average throughput results of App-TE outperformed the others when the random topology (less equal-cost path) is conducted.

As a summarization, App-TE not only got better throughput results but also obtain less packet loss rate. The estimation of QoS parameters and installing flow entries took a significant amount of time. When the traffic volume is smaller than the link capacity, the shortest path routing can get better results than the constraints-aware routing. And, App-TE can apply every topology, however, App-TE got better performance results in topology which has less equal-cost paths.

For the limitations, App-TE mainly focused on two modules: measurement and management in this work. To perform traffic classification and measurement, App-TE does not consider the previous flows which are already classified and it already has assigned flow rules. Therefore, App-TE can avoid unnecessary flow classifications, estimation parameters, and installing flow rules. Moreover, the purpose of using a sFlow-RT analyzer in traffic classification is to reduce controller workload. However, port and protocol number-based application traffic classification cannot exactly categorize application traffic flows; therefore, the effective traffic classification schemes are needed to perform by applying Deep Packet Inspection (DPI) or Machine Learning techniques.

6.3 Recommendations for Future Work

Although App-TE is implemented to fulfill its objectives. There has still left some works for future extension.

App-TE took account of the network QoS constraints such as available bandwidth and delay. Not only to study other network QoS constraints such as jitter and delay variation but also consider application QoS constraints such as buffer playtime for YouTube video.

App-TE aims to guarantee the prioritized application traffic flows, therefore, the traffic management method of App-TE considers two routings. One for prioritized and other for non-prioritized application traffic flows. App-TE does not consider future traffic demands. Therefore, the traffic management solution still need to consider to adapt to future demands.

AUTHOR'S PUBLICATIONS

[p1] M. T. Z. Win, K. T. Mya, and A. H. Maw, "Study on Segment Routing on SDN", In the Proceedings of the 15th International Conference on Computer Applications (ICCA), Yangon, Myanmar, pp. 385-390, February 2017.

[p2] M. T. Z. Win, K. T. Mya, and Y. Ishibashi, "Traffic Engineering with Segment Routing in ONOS Controller", In the Proceedings of the 16th International Conference on Computer Applications (ICCA), Yangon, Myanmar, pp. 380-384, February 2018.

[p3] M. T. Z. Win, Y. Ishibashi, and K. T. Mya, "Available Bandwidth Based Application-aware Engineering in SDN", In the Proceedings of 2019 the 9th International Workshop on Computer Science and Engineering (WCSE), WCSE_2019_SPRING, Yangon, Myanmar, pp. 142-147, February 2019. ISBN 978-981-14-1455-8, [doi:10.18178/wcse.2019.03.024]

[p4] M. T. Z. Win, Y. Ishibashi, and K. T. Mya, "QoS-aware Traffic Engineering in Software Defined Networks", The 25th International Asia-Pacific Conference on Communication (APCC), Ho Chi Minh, Vietnam, pp. 171-176, November 2019.

[p5] M. T. Z. Win and K. T. Mya, "QoS-aware Traffic Management in Software Defined Networking", International Journal of Sciences: Basic and Applied Research Journals (IJSBAR), ISSN:2307-4531 [Online], January 2020. (To be appeared).

BIBLIOGRAPHY

[1] I. F. Akyildiz, T. Anjali, L. Chen, J. C. D. Oliveira, C. Scoglio, A. Sciuto, J. A. Smith, and G. Uhl, “A New Traffic Engineering Manager for Diffserv/MPLS Networks: Design and Implementation on an IP QoS Testbed”, In the Journal of Computers on Communications, Elsevier, vol. 26, no. 4, pp. 388-403, March 2003.

[2] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, “A Roadmap for Traffic Engineering in SDN-OpenFlow Networks”, In the Journal of Computer Networks, Elsevier, vol. 71, pp.1-30, October 2014.

[3] C. Alaettinoglu, “Overcoming Traffic Engineering Challenges with SDN”, APNIC, Tech matters, May 2017, [Online] Available: https://blog.apnic.net/overcoming-traffic-engineering-challenges-sdn.

[4] R. Alshammari and A. N. Z. Heywood, “Machine Learning based Encrypted Traffic Classification: Identifying SSH and Skype”, In the Proceedings of 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA), IEEE, pp. 1-8, July 2009.

[5] R. Alshammari and A. N. Z. Heywood, “Identification of VoIP Encrypted Traffic Using Machine Learning Approach”, In the Journal of King Saud University of Computer and Information Sciences, Elsevier, vol. 27, no. 1, pp. 77-92, January 2015.

[6] C. Arsenault, “Understanding Network Bandwidth vs Latency”, August 2017, [Online] Available: https://www.keycdn.com/blog/network-bandwidth.

[7] S. Avallone, S. Guadagno, D. Emma, A. Pescapè, and G. Ventre, “D-ITG Distributed Internet Traffic Generator”, In the Proceedings of First International Conference on the Quantitative Evaluation of Systems, (QEST), IEEE, pp. 316-317, September 2004.

[8] D. O. Awduche and J. Agogbua, “Requirements for Traffic Engineering over MPLS”, RFC 2702, Tech. Rep., September 1999.

[9] D. O. Awduche, “MPLS and Traffic Engineering in IP Networks”, In the Journal of IEEE Communications Magazine, IEEE, vol. 37, no. 12, pp. 42-47, December 1999.

[10] D. O. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao, "Overview and Principles of Internet Traffic Engineering", RFC 3272, Tech. Rep., May 2002.

[11] M. Azizi, R. Benaini, and M. B. Mamoun, "Delay Measurement in Openflow-Enabled MPLS-TP Network", In the Journal of Modern Applied Science, Canadian Center of Science and Education, vol. 9, no. 3, pp. 90, March 2015.

[12] S. Azodolmolky, R. Nejabati, M. Pazouki, P. Wieder, R. Yahyapour, and D. Simeonidou, "An Analytical Model for Software Defined Networking: A Network Calculus-based Approach", In the Proceedings of 2013 IEEE Global Communications Conference (GLOBECOM), IEEE, pp. 1397-1402, December 2013.

[13] R. Boutaba, W. Szeto, and Y. Iraqi, "DORA: Efficient Routing for MPLS Traffic Engineering", In the Journal of Network and Systems Management, Springer, vol. 10, no. 3, pp. 309-325, September 2002.

[14] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. V. Merwe, "Design and Implementation of a Routing Control Platform", In the Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation, USENIX Association, vol. 2, pp. 15-28, May 2005.

[15] J. Chen, J. Ling, J. Zhou, and W. Zhang, "Link Failure Recovery in SDN: High Efficiency, Strong Scalability and Wide Applicability", In the Journal of Circuits, Systems and Computers, World Scientific, vol. 27, no. 06, pp.1850087-1850117, June 2018.

[16] L. C. Cheng, K. Wang, and Y. H. Hsu, "Application-aware Routing Scheme for SDN-based Cloud Datacenters", In the Proceedings of 2015 7th International Conference on Ubiquitous and Future Networks, IEEE, pp. 820-825, July 2015.

[17] L. W. Cheng and S. Y. Wang, "Application-aware SDN Routing for Big Data Networking", In 2015 IEEE Global Communications Conference (GLOBECOM), IEEE, pp. 1-6, December 2015.

[18] T. Chin, M. Rahouti, and K. Xiong, "Applying Software Defined Networking to Minimize the End-to-End Delay of Network Services", In

the Journal of ACM SIGAPP Applied Computing Review, ACM, vol. 18, no. 1, pp. 30-40, March 2018.

[19] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, “Payless: A Low-Cost Network Monitoring Framework for Software Defined Networks”, In the Proceedings of 2014 IEEE Network Operations and Management Symposium (NOMS), IEEE, pp. 1-9, May 2014.

[20] F. Ciucu and J. Schmitt, “Perspectives on Network Calculus: No Free Lunch, But Still Good Value”, In the Journal of ACM SIGCOMM Computer Communications Review, ACM, vol. 42, no. 4, pp. 311–322, September 2012.

[21] G. C. Deng and K. Wang, “An Application-aware QoS Routing Algorithm for SDN-based IoT Networking”, In the Proceedings of 2018 IEEE Symposium on Computers and Communications (ISCC), IEEE, pp. 186-191, 2018.

[22] N. Deo and C. Y. Pang, “Shortest-path Algorithms: Taxonomy and Annotation”, In the Journal of Networks, Wiley Online Library, vol. 14, no. 2, pp. 275-323, June 1984.

[23] K. L. Dias, M. A. Pongelupe, W. M. Caminhas, and L. Errico, “An Innovative Approach for Real-time Network Traffic Classification”, In the Journal of Computer Networks, Elsevier, vol. 158, pp. 143-157, July 2019.

[24] 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 the Proceedings of 2012 Asia Pacific Signal and Information Processing Association Annual Summit and Conference, IEEE, pp. 1-8, December 2012.

[25] M. A. Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat, “Hedera: Dynamic Flow Scheduling for Data Center Networks”, In the Proceedings of Networked Systems Design and Implementation Symposium (NSDI), vol. 10, no. 2010, April 2010.

[26] G. Finnie, “The Role of DPI in an SDN World”, White paper, Heavy Reading, December 2012.

[27] M. Finsterbusch, C. Richter, E. Rocha, J. A. Muller, and H. Klaus, “A Survey of Payload-based Traffic Classification Approaches”, In the

Journal of IEEE Communications Surveys & Tutorials, IEEE, vol.16, no. 2, pp.1135-1156, October 2013.

[28] P. Fonseca, R. Bennesby, E. Mota, and A. Passito, “A Replication Component for Resilient OpenFlow-based Networking”, In the Proceedings of 2012 IEEE Network Operations and Management Symposium (NOMS), IEEE, pp. 933–939, April 2012.

[29] B. Fortz and M. Thoup, “Internet Traffic Engineering by Optimizing OSPF Weights”, In Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), IEEE, vol. 2, pp. 519–528, March 2000.

[30] B. Fortz, J. Rexford, and M. Thorup, “Traffic Engineering with Traditional IP Routing Protocols”, In the Journal of IEEE Communications Magazine, IEEE, vol. 40, no. 10, pp. 118-124, December 2002.

[31] R. L. Gomes and E. R. M. Madeira, “A Traffic Classification Agent for Virtual Networks based on QoS Classes”, In the Journal of IEEE Latin America Transactions, IEEE, vol. 10, no.3, pp. 1734-1741, June 2012.

[32] Y. Goo, K. Shim, S. Lee and M. Kim, “Payload Signature Structure for Accurate Application Traffic Classification”, In the Proceedings of 2016 18th Asia Pacific Network Operations and Management Symposium (APNOMS), IEEE, pp. 1-4, October 2016.

[33] W. Goralski, “The Illustrated Network: How TCP/IP Works In A Modern Network”, Morgan Kaufmann, April 2017.

[34] A. Greenberg, G. Hjalmtysson, D.A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, “A Clean Slate 4D Approach to Network Control and Management”, In the Journal of ACM SIGCOMM Computer Communication Review, vol. 35, no. 5, pp. 41-45, October 2005.

[35] G. Han, J. Jiang, N. Bao, L. Wan, and M. Guizani, “Routing Protocols for Underwater Wireless Sensor Networks”, In the Journal of IEEE Communications Magazine, IEEE, vol. 53, no. 11, pp. 72-78, November 2015.

[36] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, and M. Thottan, “Measuring Control Plane Latency in SDN-enabled

Switches", In the Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, ACM, pp. 25, June 2015.

[37] C.E. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm", RFC 2992, Tech. Rep., November 2000.

[38] Fei Hu, "Network Innovation Through OpenFlow and SDN: Principles and Design", CRC Press, February 2014.

[39] P. Huang and Y. Ishibashi, "QoE Assessment of Will Transmission Using Vision and Haptics in Networked Virtual Environment", In the Journal of Communications, Network and System Sciences, Citeseer, vol. 7, no. 08, pp. 265-278, July 2014.

[40] G. Iannaccone, C. N. Chuah, R. Mortier, S. Bhattacharyya, and C. Diot, "Analysis of Link Failures in an IP Backbone", In the Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement, ACM, pp. 237-242, November 2002.

[41] A. Iqbal, U. Javed, S. Saleh, J. W. Kim, J. S. Alowibdi, and M. U. Ilyas, "Analytical Modeling of End-to-End Delay in OpenFlow Based Networks", In the Journal of IEEE Access, Special Section on Future Networks: Architectures, Protocols, and Applications, IEEE, vol. 5, pp. 6859-6871, December 2016.

[42] M. Jain and C. Dovrolis, "End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput", In the Journal of IEEE/ACM Transactions on Networking (TON), IEEE Press, vol. 11, no. 4, pp. 537-549, August 2003.

[43] Y. Jarraya, T. Madi, and M. Debbabi, "A Survey and A Layered Taxonomy of Software Defined Networking", In the Journal of IEEE Communications Surveys & Tutorials, IEEE, vol. 16, no. 4, pp. 1955-1980, April 2014.

[44] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and Performance Evaluation of An OpenFlow Architecture", In the Proceedings of 23rd International Teletraffic Congress, International Teletraffic Congress, pp. 1-7, September 2011.

[45] 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 2013 2nd European Workshop on Software Defined Networks (EWSDN), IEEE, pp. 87-92, October 2013.

[46] S. Jeong, D. Lee, J. Hyun, J. Li, and J.W.K. Hong, "Application-aware Traffic Engineering in Software Defined Network", In the Proceedings of 2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS), IEEE, pp. 315-318, September 2017.

[47] S. Jeong, D. Lee, J. Choi, J. Li, and J. W. K. Hong, "Application-aware Traffic Management for OpenFlow Networks", In the Proceedings of 2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS), IEEE, pp. 1-5, October 2016.

[48] R. Jmal and L. C. Fourati, "Implementing Shortest Path Routing Mechanism Using OpenFlow POX Controller", In the Proceedings of 2014 International Symposium on Networks, Computers and Communications, IEEE, pp. 1-6, June 2014.

[49] K. H. Joon, M. Chlansker, J. R. Santos, J. Tourrilhes, Y. Turner, and N. Feamster, "Coronet: Fault Tolerance for Software Defined Networks", In the Proceedings of 2012 20th IEEE International Conference on Network Protocols (ICNP), pp. 1-2, October 2012.

[50] S. Kamat, R. Guerin, A. Orda, and T. Przygienda, "QoS Routing Mechanisms and OSPF Extensions", In the Proceedings of GLOBECOM' 97, Citeseer, 1997.

[51] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic Load Balancing Without Packet Reordering", In the Journal of ACM SIGCOMM Computer Communication Review, vol. 37, no. 2, pp. 51-62, March 2007.

[52] J. Kempf, S. Whyte, J. Ellithorpe, P. Kazemian, M. Haitjema, N. Beheshti, S. Stuart, and H. Green, "OpenFlow MPLS and the Open Source Label Switched Router", In the Proceedings of the 23rd International Tele Traffic Congress, International Tele Traffic Congress, pp. 8-14, September 2011.

[53] M. Kodialam and T. V. Lakshman, "Minimum Interference Routing with Applications to MPLS Traffic Engineering", In the Proceedings of IEEE INFOCOM 2000 and 19th Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE, vol. 2, pp. 884 - 893, 2000.

[54] A. Lester, Y. Tang, and T. Gyires, “Application-aware Bandwidth Scheduling for Data Center Networks”, In the Journal on Advances in Networks and Services, Citeseer, vol. 7, 2014.

[55] Y. Li and M. Chen, “Software Defined Network Function Virtualization: A Survey”, In the Journal of IEEE Access, IEEE, vol. 3, pp. 2542-2553, December 2015.

[56] L. E. Li, Z. M. Mao, and J. Rexford, “CellSDN: Software-Defined Cellular Networks”, Technical Report, Princeton University Computer Science, Citeseer, 2012.

[57] G. Li, M. Dong, K. Ota, J. Wu, J. Li, and T. Ye, “Deep Packet Inspection-based Application-aware Traffic Control for Software Defined Networks”, In the Proceedings of IEEE Global Communications Conference (GLOBECOM), IEEE, pp. 1-6, December 2016.

[58] G. Liang and W. Li, “A Novel Industrial Control Architecture Based on Software Defined Network”, In the Journal of Measurement and Control, SAGE Publications Sage UK: London, England, vol. 51, no. 7-8, pp. 360-367, September 2018.

[59] A. Marshall, K. M. Yap, and W. Yu, “Providing QoS for Networked Peers in Distributed Haptic Virtual Environments”, In the Journal of Advances in Multimedia, Hindawi, vol. 2008, pp. 1-14, June 2008.

[60] P. Megyesi, A. Botta, G. Aceto, A. Pescape, and S. Molnár. “Available Bandwidth Measurement in Software Defined Networks”, In the Proceedings of the 31st Annual ACM Symposium on Applied Computing, ACM, pp. 651-657, April 2016.

[61] P. Megyesi, A. Botta, G. Aceto, A. Pescapé, and S. Molnár, “Challenges and Solution for Measuring Available Bandwidth in Software Defined Networks”, In the Journal of Computer Communications, Elsevier, vol. 99, pp.48-61, February 2017.

[62] H. Mekky, F. Hao, S. Mukherjee, Z. L. Zhang, and T. V. Lakshman, “Application-aware Data Plane Processing in SDN”, In the Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking, ACM, pp. 13-18, August 2014.

[63] M. A. Momin and J. Cosmas, “The Impact of Content Oriented Routing on OpenFlow Burst Switched Optical Networks”, In the Proceedings of 2013 27th International Conference on Advanced Information Networking and Applications Workshops, IEEE, pp. 965-970, March 2013.

[64] A. Moore and K. Papagiannaki, “Toward the Accurate Identification of Network Applications”, In the Proceedings of International Workshop on Passive and Active Network Measurement, Springer, Berlin, Heidelberg, pp. 41-54, March 2005.

[65] M. R. Parsaei, M. J. Sobouti, S. R. Khayamin, and R. Javidan, “Network Traffic Classification Using Machine Learning Techniques Over Software Defined Networks”, In the International Journal of Advanced Science and Applications, vol. 8, no. 7, pp. 220-225, July 2017.

[66] T.V.S. Pasca, S. S. Prasad, and K. Kataoka, “AMPF: Application-aware Multipath Packet Forwarding Using Machine Learning and SDN”, June 2016, [Online] Available: https://arxiv.org/abs/1606.05743.

[67] M. Reitblatt, N. Foster, J. Rexford, and D. Walker, “Consistent Updates for Software Defined Networks: Change You Can Believe In!”, In the Proceedings of the 10th ACM Workshop on Hot Topics in Networks, HOT, ACM, pp. 7, November 2011.

[68] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, “Abstractions for Network Update”, In the Journal of the ACM SIGCOMM Computer Communication Review, ACM, vol. 42, no. 4, pp. 323–334, September 2012.

[69] B. Renukadevi and B. D. M. Raja, “Deep Packet Inspection Management Application in SDN”, In the Proceedings of 2017 2nd International Conference on Computing and Communications Technologies (ICCCT), IEEE, pp. 256-259, February 2017.

[70] G. Retvari and T. Cinkler, “Practical OSPF Traffic Engineering”, In the Journal of IEEE Communications Letters, IEEE, vol. 8, no. 11, pp. 689-691, November 2004.

[71] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell, “Pathchirp: Efficient Available Bandwidth Estimation for Network Paths”, In the Proceedings of Passive and Active Measurements Workshop, April 2003.

[72] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, “SDN Controllers: A Comparative Study”, In the Proceedings of 2016 18th Mediterranean Electrotechnical Conference (MELECON), IEEE, pp. 1-6, April 2016.

[73] E. Schweissguth, P. Danielis, C. Niemann, and D. Timmermann, “Application-aware Industrial Ethernet Based on an SDN-supported TDMA Approach”, In the Proceedings of 2016 IEEE World Conference on Factory Communication Systems (WFCS), IEEE, pp. 1-8, May 2016.

[74] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, “OpenFlow-based Segment Protection in Ethernet Networks”, In the Journal of Optical Communications and Networking, Optical Society of America, vol. 5, no. 9, pp. 1066–1075, September 2013.

[75] A. R. Sharafat, S. Das, G. Parulkar, and N. M. Keown, “MPLS-TE and MPLS VPNS with OpenFlow”, In the Proceedings of ACM SIGCOMM Computer on Communication Review, ACM, vol. 41, no. 4, pp. 452-453, August 2011.

[76] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, “Enabling Fast Failure Recovery in OpenFlow Networks”, In the Proceedings of 2011 8th International Workshop on the Design of Communication Networks (DRCN), IEEE, pp. 164–171, 2011.

[77] H. Shi, H. Li, D. Zhang, C. Cheng, and W. Wu, “Efficient and Robust Feature Extraction and Selection for Traffic Classification”, In the Journal of Computer Networks, Elsevier, vol. 119, pp. 1-16, June 2017.

[78] Z. Shu, J. Wan, J. Lin, S. Wang, D. Li, S. Rho, and C. Yang, “Traffic Engineering in Software Defined Networking: Measurement and Management”, In the Journal of IEEE access, vol. 4, pp. 3246-3256, June 2016.

[79] M. Singh, N. Varyani, J. Singh, and K. Haribabu, “Estimation of End-to-End Available Bandwidth and Link Capacity in SDN”, In the Proceedings of International Conference on Ubiquitous Communications and Network Computing, Springer, pp. 130-141, August 2017.

[80] D. Sinh, L. V. Le, B. S. P. Lin, and L. P. Tung, “SDN/NFV-A New Approach of Deploying Network Infrastructure for IoT”, In the

Proceedings of 2018 27th Wireless and Optical Communication Conference (WOCC), IEEE, pp. 1-5, June 2018.

[81] J. Sommers, P. Barford, and W. Willinger, "A Proposed Framework for Calibration of Available Bandwidth Estimation Tools", In the Proceedings of the 11th IEEE Symposium on Computers and Communications (ISCC), IEEE, pp. 709-718, June 2006.

[82] W. Stalling, "Software-Defined Networks and OpenFlow", In the Journal of Internet Protocol, Citeseer, vol.16, no. 1, pp. 2-14, March 2013.

[83] G. Swallow, "MPLS advantages for Traffic Engineering", In the Journal of IEEE Communications Magazine, IEEE, vol. 37, no. 12, pp. 54-57, December 1999.

[84] R. Thenmozhi and B. Amudha, "Efficient Video Delivery Over a Software Defined Network", In International Conference on Communications and Cyber Physical Engineering, Springer, Singapore, pp. 27-37, January 2019.

[85] A. Tootoonchian and Y. Ganjali, "Hyperflow: A Distributed Control Plane for OpenFlow", In the Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking, vol. 3, April 2010.

[86] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic Matrix Estimator for OpenFlow Networks", In the Proceedings of International Conference on Passive and Active Network Measurement, Springer, Berlin, Heidelberg. pp. 201-210, April 2010.

[87] Z. Trabelsi, S. Zeidan, and M. M. Mohammad, "Network Packet Filtering and Deep Packet Inspection Hybrid Mechanism for IDS Early Packet Matching", In the Proceedings of 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA), IEEE, pp. 808-815, March 2016.

[88] M. T. Z. Win, Y. Ishibashi, and K. T. Mya, "Available Bandwidth Based Application-aware Engineering in SDN", In the Proceedings of 2019 the 9th International Workshop on Computer Science and Engineering (WCSE), WCSE_2019_SPRING, Yangon, Myanmar, pp. 142-147, February 2019, [Online], Available [doi:10.18178/wcse.2019.03.024]

[89] J. Yan and J. Yuan, “A Survey of Traffic Classification in Software Defined Networks”, In the Proceedings of 2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN), IEEE, pp. 200-206, August 2018.

[90] Y. Yang, J. K. Muppala, and S. T. Chanson, “Quality of Service Routing Algorithms for Bandwidth-Delay Constrained Applications”, In the Proceedings of 9th International Conference on Network Protocols (ICNP), IEEE, pp. 62-70, November 2001.

[91] H. T. Zaw and A. H. Maw, " Elephant Flow Detection and Delay-aware Flow Rerouting in Software Defined Network”, In the Proceedings of 2017 9th International Conference on Information Technology and Electrical Engineering (ICITEE), IEEE, pp. 1-6, October 2017.

[92] S. Liu, “Software Defined Networking Market Revenue Worldwide 2013-2021”, Technology & Communication, IT Services, March 2019, [Online] Available: https://www.statista.com/statistics/468636/global-sdn-market-size.

[93] J. Martindale, “What is FTP?”, Digital Trends, August 2019, [Online] Available: https://www.digitaltrends.com/computing/what-is-ftp-and-how-do-i-use-it/

[94] “An Instant Virtual Network on your Laptop (or other PC)”, Mininet, October 2018, [Online], Available: http://mininet.org.

[95] “Cisco Visual Networking Index: Forecast and Trends”, White Paper, pp. 2017–2022, February 2019, [Document ID: 1551296909190103].

[96] “IANA, Internet Assigned Numbers Authority.” [Online], Available: https://www.iana.org/numbers

[97] Iperf, [Online] Available: https://iperf.fr.

[98] “Making the Network Visible”, sFlow, 2019, [Online] Available: http://www.sflow.org.

[99] “Network Throughput -What is It, How to Measure & Optimize!”, PC & Network Download, April 2019, [Online], Available: https://www.pcwdld. com/network-throughput.

[100] ONOS, [Online] Available: https://wiki.onosproject.org /display/ ONOS/ONOS.

[101] "OpenFlow Switch Specification Version 1.5.1 (Protocol version 0x06)", Open Networking Foundation, March 2015, [Online] Available: https://www.opennetworking.org/wp-content/uploads/2014/10/ openflow-switch-v1.5.1.pdf

[102] "Packet Switching and Delays in Computer Network", [Online] Available: https://www.geeksforgeeks.org/packet-switching-and-delays-in-computer-network/

[103] "Service-aware Network Architecture based on SDN, NFV, and Network Intelligence", White paper, Intel Architecture Processors QoSmos DPI Technology, Networking and Communications, 2014.

[104] "Software-Defined Networking: The New Norm for Networks", Open Networking Foundation, ONF White Paper, pp. 1-12, April 2012.

LIST OF ACRONYMS

ABW	Available Bandwidth
APIs	Application Programming Interfaces
App-TE	Application-aware Traffic Engineering
BGPs	Border Gateway Protocols
DFS	Depth First Search
DiffServ	Differentiated Services
D-ITG	Distributed Internet Traffic Generator
DPI	Deep Packet Inspection
DWC	Delay Weighted Capacity
ECMP	Equal-cost Multipath
FTP	File Transfer Protocol
HTTP	Hyper Text Transfer Protocol
IANA	Internet Assigned Number Authority
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IE	Industrial Ethernet
IGP	Interior Gateway Protocol
IoTs	Internet of Things
IP	Internet Protocol
ISPs	Internet Service Providers
IS-IS	Intermediate System – Intermediate System
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LARAC	Lagrange Relaxation based Aggregated Cost

LDP	Label Distribution Protocol
LLDP	Link Layer Distribution Protocol
LSP	Label Switched Path
LU	Link Utilization
MDWCRA	Maximum Delay Weighted Capacity Routing Algorithm
MHR	Minimum Hop-count-based Routing
ML	Machine Learning
MPLS	Multi-Protocol Label Switching
NOS	Network Operating System
ONF	Open Network Foundation
ONOS	Open Network Operating System
OSPF	Open Shortest Path First
OVS	Open vSwitch
QoS	Quality of Service
RCP	Routing Control Platform
REST	Representational State Transfer
RPC	Remote Procedure Called
RSVP	Resource Reservation Protocol
RTT	Round Trip Time
SDN	Software Defined Networking
SMS	Short Message Service
SPF	Shortest Path First
TCAM	Ternary Content Addressable Memory
TCP	Transport Control Protocol
TDMA	Time Division Multiple Access

TLS	Transport Layer Security
VM	Virtual Machine
WAN	Wide Area Network

APPENDIX: SOFTTWARE FOR EXPERIMENTAL TESTBED

(I) Installation and Running of ONOS Controller

The ONOS install process depend on on the environment variable JAVA_HOME being properly set. Both maven and java need to have same Java version. Firstly, it needs to install maven and Karaf as the following:

```
root@thu:~$ cd; mkdir Downloads Applications
root@thu:~$ cd Downloads
root@thu:~$ wget http://archive.apache.org/dist/karaf/3.0.5/apache-karaf-3.0.5.tar.gz
root@thu:~$ wget http://archive.apache.org/dist/maven/maven-3/3.3.9/binaries/apache-maven-3.3.9-bin.tar.gz
root@thu:~$ tar -zxvf apache-karaf-3.0.5.tar.gz -C ../Applications/
root@thu:~$ tar -zxvf apache-maven-3.3.9-bin.tar.gz -C ../Applications/
```

Then, install Oracle Java 8:

```
root@thu:~$ sudo apt-get install software-properties-common -y
root@thu:~$ sudo add-apt-repository ppa:webupd8team/java -y
root@thu:~$ sudo apt-get update
root@thu:~$ sudo apt-get install oracle-java8-installer oracle-java8-set-default
```

Set the ONOS_ROOT environment variable and it need to export in the shell profile *(bash_profile*). After that, enters the ONOS directory then clean and install maven.

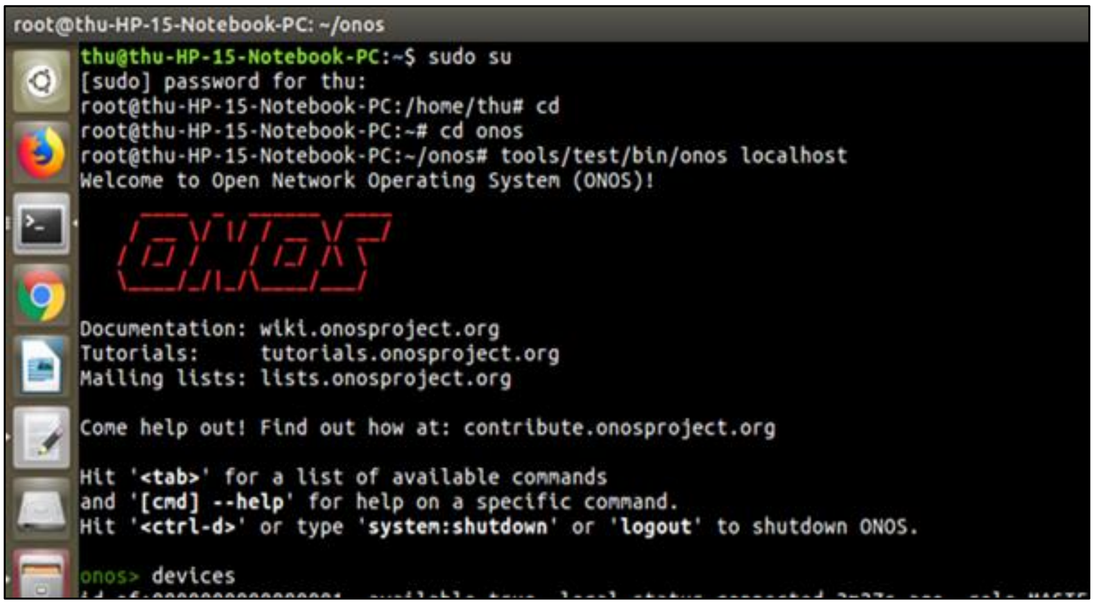
```
root@thu:~$  export JAVA_HOME=/usr/lib/jvm/java-8-oracle
root@thu:~$ export ONOS_ROOT=~/onos
root@thu:~$ source $ONOS_ROOT/tools/dev/bash_profile
root@thu:~$  cd ~/onos
root@thu:~$  mvn clean install
```

Before, maven clean and install, first, need to download ONOS from git and checkout with version (need to install)

```
root@thu:~$  git clone https://gerrit.onosproject.org/onos
root@thu:~$  cd onos
root@thu:~$  git tag
root@thu:~$  git checkout -b 1.8.0
```

After installation required software and ONOS, ONOS controller can be started.

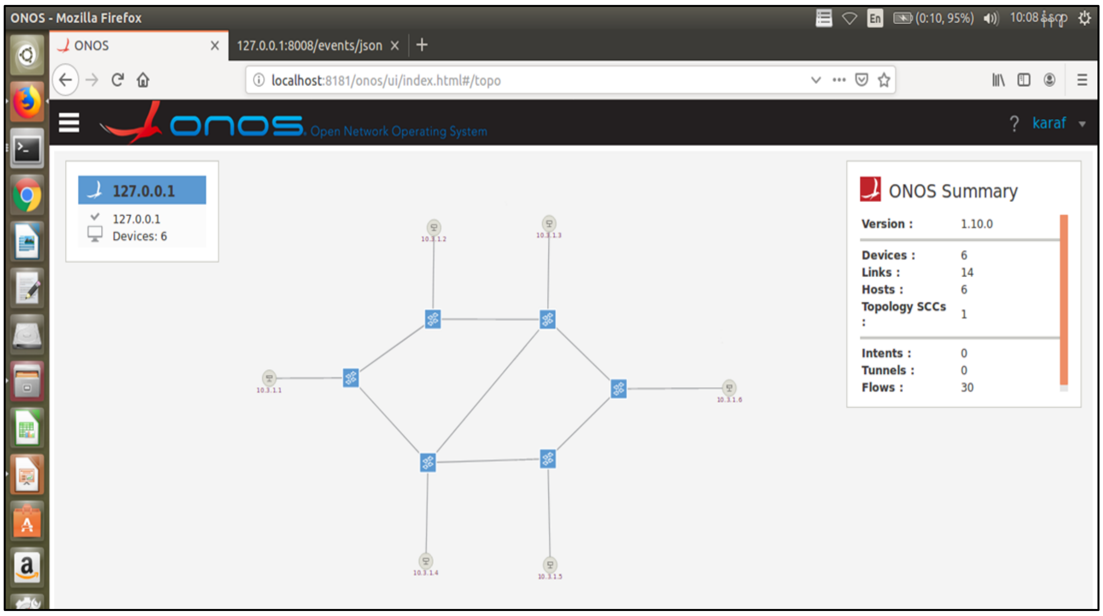
Note: The above installation steps are for the ubuntu user. This dissertation used Ubuntu OS; therefore, the following steps are the required step for Ubuntu OS. The detailed installation steps and installation steps for other OS can find in here (***https://wiki.onosproject.org/display/ONOS14/Installing+and+Running+ONOS***). When the ONOS is loaded correctly, ONOS CLI can be accessed with the commands (***ok clean (or) tools/test/bin/onos localhost)*** as the following.



```
root@thu-HP-15-Notebook-PC: ~/onos
thu@thu-HP-15-Notebook-PC:~$ sudo su
[sudo] password for thu:
root@thu-HP-15-Notebook-PC:/home/thu# cd
root@thu-HP-15-Notebook-PC:~# cd onos
root@thu-HP-15-Notebook-PC:~/onos# tools/test/bin/onos localhost
Welcome to Open Network Operating System (ONOS)!

Documentation: wiki.onosproject.org
Tutorials:     tutorials.onosproject.org
Mailing lists: lists.onosproject.org

Come help out! Find out how at: contribute.onosproject.org

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown ONOS.

onos> devices
```

ONOS's GUI can be accessed from any browser through the IP address of the target machine. The default username and password are onos / rocks.

http://targeted Machine IP:8181/onos/ui/index.html



ONOS Summary
Version : 1.10.0
Devices : 6
Links : 14
Hosts : 6
Topology SCCs : 1
Intents : 0
Tunnels : 0
Flows : 30

(II) Installation and Running of sFlow-RT Analyzer

sFlow-RT analyzer is used to detects the active flows or incoming traffic through the network. The following commands are used to download the sFlow software and unzip the software:

$ wget https://inmon.com/products/sFlow-RT/sflow-rt.tar.gz

$ tar -xvzf sflow-rt.tar.gz

To run and start the sFlow:

$ ./sflow-rt/start.sh

In the script directory and html directory, the JavaScript (.js) extension file is a thread for each of the files after sFlow is started. These are accessed from the following directories:

http://localhost:8008/app/detectFlows/html/

http://localhost:8008/scripts/json

App-TE used Mininet Dashboard which is one of the applications of sFlow RT for real-time dashboard for Mininet and it can provide the dashboard web interface. The followings steps are used to install and run Mininet Dashboard:

$ cd sflow-rt

$ ./get-app.sh sflow-rt mininet-dashboard

$ ./start.sh

The dashboard application not only can add metrics but also can generate events. To gain the sFlow metrics, firstly flows can be defined using JavaScript setFlow() fuction. setFlow() function instructs the controller build a flow cache to track TCP/UDP connections. Alternatively, the defined flows are used to match packets or values.

In the following source code, a defined flow name is "*mn_flow*" and it took the source IP address, destination IP address, source MAC, destination MAC, TCP or UDP source and destination ports, calculated bytes per second, and timestamp of the incoming flow. The attributes in the "*keys*" are called metrics. The setThreshold() function defines "Detect_Flows" as "mn_flow" that exceed 10% of the link's bandwidth (in this case

1Mbit/second) for 1 second or more. The setEventHandler() function processes each "*Detect_Flows*" notification.

```
setFlow('mn_flow',
{'keys':'ipsource,ipdestination,macsource,macdestination,or:tcpsourceport:udpsourcep
ort,or:tcpdestinationport:udpdestinationport','value':'bytes','timestamp':'seconds',
t:2,fs:SEP});

setThreshold('Detect_Flows',
{'metric':'mn_flow','value':1000000/8,'byFlow':true,'timeout':1});

setEventHandler(function(event) {
logInfo(event.flowKey);
},['Detect_Flows']);
```

As the summarization, App-TE can access events of json information (5-turples metrics) of incoming flows (mn_flow) when the flow (mn_flow) is greater than 1 M (Megabit).

(III) Installation and Running of Mininet Network Emulator

The following codes are used to get natively from Mininet source code and to install Mininet:

```
$ git clone git://github.com/mininet/mininet

$ cd mininet/util/

$ mininet/util/install.sh -a
```

where, -a argument means installing everything which contains Open vSwitch, OpenFlow Wireshark dissector, POX and so on. After that, network topologies can be created by a command or scripts that written in python file. The detailed information can be available in here (http://mininet.org/sample-workflow/).