UNIVERSITY OF OTTAWA

MASTER THESIS

Dynamic Routing with Online Traffic Estimation for Video Streaming over Software Defined Networks (SDN)

Riqiang LIU

A thesis submitted for the degree of Master of Applied Science

in the

School of Electrical Engineering and Computer Science (EECS)

March 19, 2020

Abstract

The traffic generated by video streaming applications constitutes a large portion of the Internet traffic over today's networks. Video streaming demands for low latency and high bandwidth. In particular, transmission of high-quality (high-resolution) streaming video may put the network under pressure. Therefore, high-quality video traffic requires network managers to make routing timely and intelligently. SDN provides a global view and centralized control for the whole network which gives opportunities to dynamically manage networks. Meanwhile, machine learning techniques are widely applied in traffic estimation. In this thesis, we use an OpenFlow-based SDN environment and propose a dynamic routing scheme with online traffic estimation to increase the quality of high-quality video streaming and the throughput of the network. The traffic is clustered using an unsupervised machine learning algorithm, and then, the high-quality video traffic flows are rerouted to disjoint paths to relieve the network congestion and have better video quality. The whole design is tested in the Mininet simulator. Simulation results show that the proposed scheme improves the link utilization and reduces the dropped frames caused by delay.

Acknowledgements

Firstly, I would like to thank the Faculty of Graduate and Postdoctoral Studies, the University of Ottawa for giving me this platform and opportunity.

I would like to express my special thanks of gratitude to my supervisor, Dr. Melike Erol-Kantarci for providing me guidance and motivation in each stage of my thesis. She inspired my interest in the development of innovative technologies. I consider myself lucky to have this great project. She taught me how to carry out research and present the work. I would also like to thank her for her friendship, kindness and patience.

I would like to thank my colleagues here in NETCORE lab, Medhat Elsayed, Mohammad Sadeghi and Kevin Shimotakahara for their valuable suggestions and guidance throughout my thesis.

In the end, I would like to thank my parents for supporting me to pursue higher education.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

AI	**A**rtificial **I**ntelligence
ARP	**A**ddress **R**esolution **P**rotocol
CBR	**C**onstant **B**it**R**ate
DBSCAN	**D**ensity **B**ased **S**patial **C**lustering of **A**pplications with **N**oise
DPI	**D**eep **P**acket **I**nspection
EGB	**E**xtreme **G**radien **B**oosting
LLDP	**L**ink **L**ayer **D**iscovery **P**rotocol
NNTP	**N**etwork **N**ews **T**ransport **P**rotocol
NSI	**N**etwork **S**tate **I**nformation
PCA	**P**rincipal **C**omponent **A**nalysis
POP3	**P**ost **O**ffice **P**rotocol - version 3
QUIC	**Q**uick **U**DP **I**ternet **C**onnections
QoS	**Q**uality of **S**ervice
RF	**R**andom **F**orests
SDN	**S**oftware-**D**efined **N**etworking
SGB	**S**tocastic **G**radien **B**oosting
SMTP	**S**imple **M**ail **T**ransfer **P**rotocol
SVM	**S**upport **V**ector **M**achine
TCP	**T**ransmission **C**ontrol **P**rotocol
UDP	**U**ser **D**atagram **P**rotocol

Chapter 1

Introduction

1.1 Motivation

According to a recent white paper from Cisco [1], IP video traffic will share 82 percent of the total global IP traffic by the year 2022. From 2017 till now, the share of video traffic has been between 80 and 90 percent of the total traffic. This includes several types of video traffic such as video conferencing, video-streamed online gaming, video file transfer, live video and so on. In addition, video content is expected to cover over 75 percent of mobile traffic by the end of 2020 [2].

Internet traffic can be classified as small flows (e.g. web page loading, small files transferring, etc) and big flows (e.g. video streaming) [3]. Big flows represent 10 percent of the total number of flows, but they cover 80 percent of the whole traffic volume [4]. With the increase of video applications, Internet traffic is changing to a more dynamic pattern and in the meanwhile video streaming has strict latency requirements, making it a challenge to serve these flows effectively.

Recently, Software-defined networks (SDN) has emerged to ease network management by separating control and data planes. SDN provides the flexibility that enables network managers to optimize video streaming [5]. In the conventional IP network, it is difficult to change network management policies according to the fast-changing

network conditions. SDN came out as there was a need to transfer pieces of specialized hardware (routers, switches, etc.) into a powerful single platform [6].

SDN is a promising way to provide flexibility in the modern network [7]. SDN separates the control plane from the data plane. This separation brings both centralization and programmability into the network. SDN can respond to faults or load changes dynamically by breaking the integration of those two planes. As the CPU processing power and storage ability keep growing, all network management functions can be moved to the central controller. In that case, switches and/or routers are only responsible for packet forwarding.

Flexibility and programmability are introduced by SDN such that network managers can implement new control policies much easier and guarantee the end-to-end quality of service (QoS) of the Internet [8]. Video streaming, as a killer network application, consumes many infrastructure resources. With SDN, networks can be dynamically configured on-demand for better management of big video flows.

Software-defined networking also plays an important role in 5G core networks. Previous generations of communications technologies primarily satisfied human communications such as voice, data and Internet. 5G on the other hand aims for vertical industries such as automotive, energy, city management, healthcare and so on. In 5G networks, SDN is paving the way for allocating network resources to satisfy the different QoS needs of vertical industries [9].

As the brain of the network, an SDN controller can significantly benefit from artificial intelligence (AI) techniques as well [10]. An increasing amount of effort is made to achieve the AI-enabled networks [11]. SDN combined with AI shows great potential in routing strategy, network security and resource allocation [12]. As a main application of AI, machine learning algorithms can be used by the SDN controller to make decisions in many fields such as flow clustering, dynamic routing and load balancing.

1.2 Thesis Objectives

The goal of this thesis is to develop a dynamic routing algorithm with online traffic estimation over SDN. The routing approach includes the following key characteristics:

- Real-time Traffic Monitoring: We monitor the network in real time for both network conditions and traffic features.
- Online Traffic Estimation: We use machine learning to identify high-quality video traffic flows.
- Dynamic Routing: We reroute high-quality video flows to less congested paths to improve user experience and relieve traffic congestion.

1.3 Thesis Contribution

In this thesis, we propose an SDN based dynamic routing approach that uses online traffic estimation and reroutes high-quality videos over less congested paths. We use DBSCAN to perform online traffic estimation and distinguish high-quality video flows, from low-quality video traffic and background traffic. We use SDN reports to discover less congested paths and dynamically reroute the big flows to those in order to achieve better performance. The following issues are especially addressed with our design and implementation:

- Efficiency and Timeliness in Network Management: We apply unsupervised learning and a sliding window with a small number of packets. The short sliding window guarantees timeliness in decision making.
- Video Quality Improvement: To show the advantages of applying machine learning and test the effectiveness of the routing strategy, we implement both clustering and dynamic routing in the SDN controller.

Simulation results show that the SDN controller with online clustering and dynamic routing can increase video streaming quality causing less frame drops and efficiently balancing the load between link.

1.4 Thesis Organization

The rest of the thesis is organized into five main chapters.

Chapter 2 presents recent research work related to our research. Many efforts are made to improve the performance of SDN by including different kinds of machine learning techniques. This chapter focuses on research that classifies or clusters network traffic with machine learning which are the body of works that are closest to ours.

Chapter 3 introduces the traffic classification technique used in our research. We use DBSCAN which is a widely used clustering technique. We identify the appropriate features to feed into the unsupervised algorithm. We also illustrate the details of the messaging system defined in OpenFlow protocol, which provides many features at both flow-level and packet-level.

The dynamic rerouting scheme is introduced in Chapter 4. This chapter describes the whole system design. The dynamic routing system contains four parts: topology discovery, real-time network monitoring, timely clustering and rerouting.

Chapter 5, shows the simulation results. Mininet is used for simulations. Real video traffic with different levels of quality and background traffic are used for performance evaluation. Five scenarios are set to show different network behaviours during dynamic rerouting. Each scenario runs with five different random seeds to show generality.

Chapter 6 presents the conclusion of the proposed dynamic routing strategy with the emphasis on the link load enhancement and video quality improvement.

Chapter 2

Background and Related Work

2.1 SDN Architecture

In software-defined networking, the control plane is decoupled from the data plane. The data plane refers to all the functions of forwarding packets from input to output. The control plane determines the routing path. SDN puts these two planes into two different layers of devices: control plane runs on a server and data plane runs on generic forwarding devices such as switches and routers. SDN controllers are running on powerful devices [13].

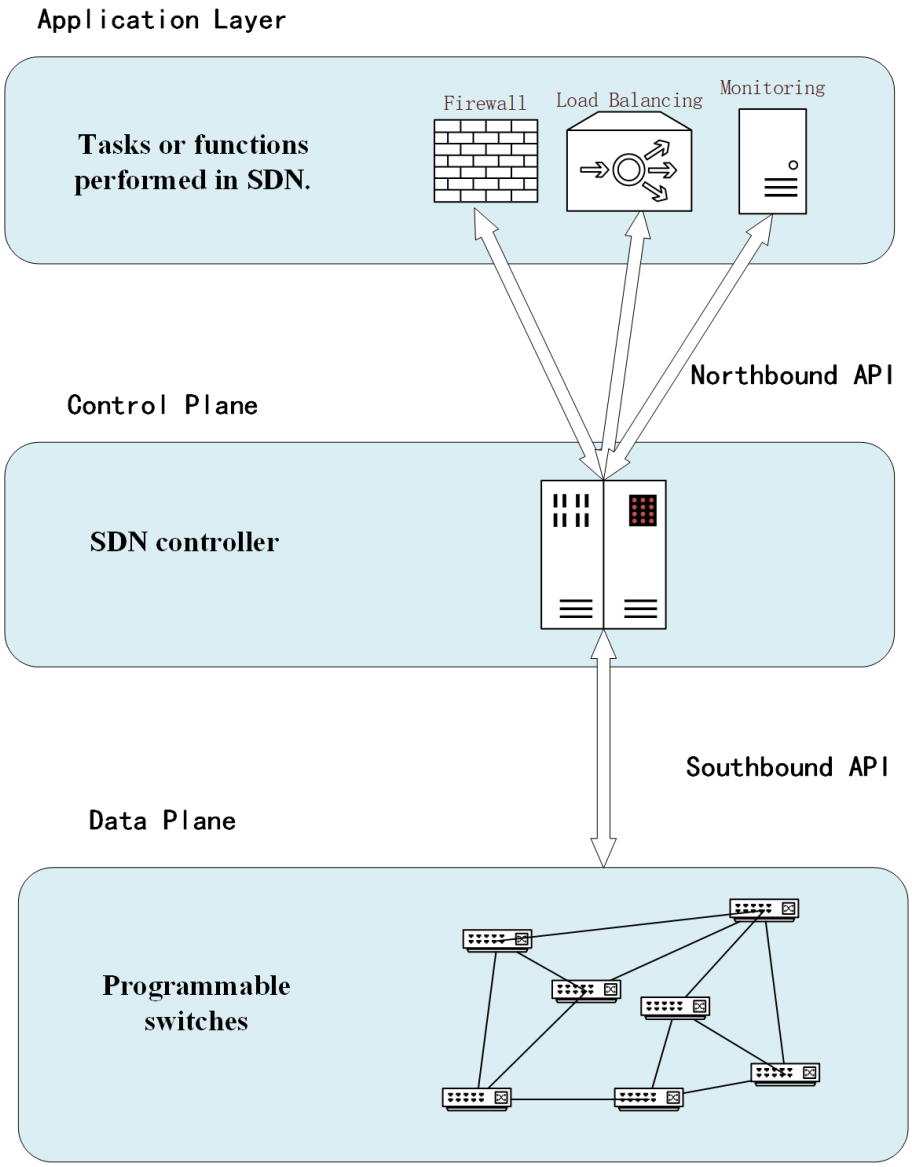


FIGURE 2.1: SDN Architecture

SDN architecture is shown in Figure 2.1. There are three layers: application layer, control plane and data plane. The application layer is also called the management plane and the data plane can be called the infrastructure plane. In a conventional network, networking tasks or functions such as network monitoring, load balancing and intrusion detection(IDS) are implemented through firmware in the hardware devices [6]. In software-defined networks, applications are moved to the application layer and the application layer communicates with the controller through the northbound API. The control plane communicates with the data plane through a southbound interface which is well defined by SDN protocols such as OpenFlow [14].

SDN provides network managers network virtualization and abstraction which is a popular trend nowadays [15]. SDN is not a new concept but thanks to the development of chips, controllers with much more powerful processing abilities, they are able to handle the global control of the whole network. The fast upgrades in computer hardware such as graphics processing unit (GPU) enable the controller to use resource-consuming machine learning techniques such as deep learning [16]. The controller sends the control logic to the switches and the switches only have forwarding elements. Thus much cheaper switches with programmability can be deployed in the network.

SDN architecture brings other benefits such as consistency and scalability. All the switches can communicate with the same controller so they can receive forwarding rules from the controller based on a global view. This centralized control also enables SDN to handle many devices with easier management. A centralized controller is able to collect a big amount of data from applications to be analyzed by machine learning [16].

2.2 OpenFlow Protocol

OpenFlow is the protocol defining how controllers send or receive forwarding rules to switches [17]. OpenFlow is the most widely deployed SDN protocol. OpenFlow-based applications can run on most of the SDN controllers (e.g., Nox [18], Pox [19], Floodlight [20], etc.) [21]. Pox is the improved version of Nox and it is written in Python. OpenFlow is well implemented in Pox so this thesis will use Pox as the SDN controller.

2.2.1 Definitions

Some important concepts are defined in OpenFlow protocol.

- Packet: a packet is an Ethernet frame containing packet headers and data payloads.
- Port: packets go in/out the pipeline via ingress/output ports on switches.
- Pipeline: pipeline is a set of flow tables.
- Flow Table: a flow table includes flow entries.
- Flow Entry: match fields and forwarding rules are included in flow entries.
- Priority: it decides the order of checking flow entries.
- Match Field: a match field could contain ingress port, Ethernet source/destination address, IPv4 or IPv6 source/destination address, TCP or UDP source/destination address and IPv4 or IPv6 protocol number.
- Action: a set of operations when forwarding a packet.
- Instruction: when a packet matches the match field switches will process the packet according to the instructions installed.

2.2.2 Packet Processing

Figure 2.2 shows how packets are processed in OpenFlow protocol. When an OpenFlow switch receives a packet, it starts a flow table lookup in the first flow table of the pipeline. The switch checks if the packet header fields can match any field in a flow entry. A packet's current state is represented by the match fields.

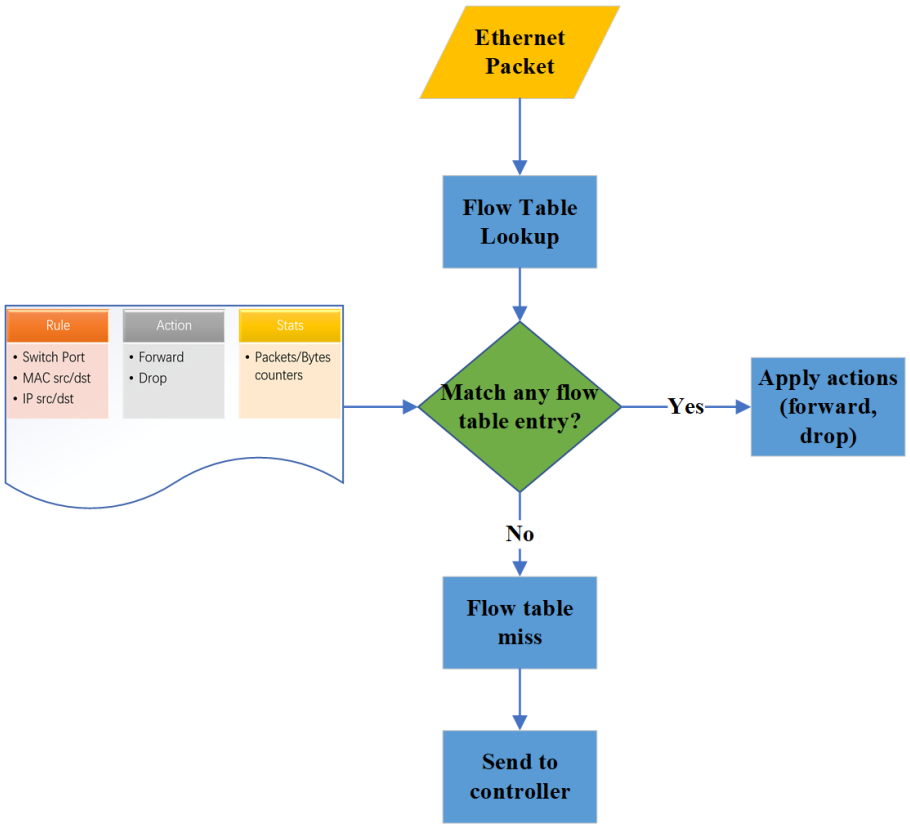


FIGURE 2.2: Packets Processing in OpenFlow Protocol

If the values of a packet's 'match field' match with the 'match field' of a flow entry then the packet is matched to this flow entry. 5-tuple packet matching or more header fields might be used if more fine-grained classification is required. Implementation requires less packet header checking for good performance [22]. If we want to omit a match field in flow entry, it is set to ANY. As long as the packet is found to be matched to a flow entry, the required action stored in the flow entry will be executed. The most

common action is Output. The Output action forwards the matched packet to the output port. Packets who do not have an Output action would be dropped.

If a packet cannot match to any flow entries, a table miss flow entry is used to process the unmatched packet. Then the switch will send asynchronous messages to the controller [23]. The decision of how to control a packet is transferred to the controller via a packet-in message. A network manager implements the function to handle packet-in messages in a controller. Then the controller sends back the message to modify flow entry. Machine learning techniques or optimization methods are usually applied to the messaging process. A controller also sends stats request messages to switches to collect the network state information (NSI). The performance of routing algorithms depends on the accuracy of NSI [24].

2.3 Video Streaming over SDN

As high-quality video streaming demands more and more network resources, video applications would compete for limited resources. SDN can assist network managers to distribute video content to relieve the potential congestion [25]. SDN shows its ability to monitor the whole network while detecting changes with timeliness which is important for video streaming [26]. However, the optimization of responding to all the requests over the whole network while maximizing network resources utilization is NP-complete [27]. If the optimization is NP-complete, there is no known way to find a solution quickly. Jian Yang et al. applied the Q-learning algorithm to make intelligent routing decisions but the convergence of Q-learning found to be slow [28].

2.4 SDN Simulation Platforms

There is always a need for testing the design of SDN with fault tolerance and scalability at low-cost [29]. The most common way is using simulation software. Another way is to build a testbed that provides real results while at a higher cost even for a small scale testing. A testbed is usually built to test the practicality of a proposed architecture. Simulation is more suitable for large scale testing especially when machine learning is applied in SDN where big data could also be involved.

Mininet is the most popular simulator and emulator of SDN [30]. Besides Mininet, some other platforms are designed to simulate SDN. Fs-sdn [31] is a flow record generator integrated with the POX controller proposed in 2013. Fs-sdn is an extension of a discrete event simulator written in Python relying on TCP throughput models. The use cases are limited and the maintenance has been stopped for at least four years.

Hyunmin Kim et al. [32] aimed to implement an SDN testbed with Raspberry-Pi on a small scale. A Raspberry-Pi device runs on an arm-based system with a Linux kernel. The tested topology had only two switches. The controller is implemented in one Raspberry-Pi device. Floodlight controller is used which is written in Java language. Two switches are implemented in two devices to form a linear topology. Scalability is sacrificed to lower the cost. The performance of the testbed is usually limited by the insufficient power of the hardware.

Robert Barrett et al. [33] implement their dynamic traffic diversion algorithm for SDN in both Mininet and a testbed using Cisco devices. Mininet can have slight differences in the measuring of delay if no link delay is set. Mininet is proved to be suitable for testing the network design and scalability.

2.4.1 Mininet

Mininet is written in Python and runs on a Linux system. To simulate software-defined networks, three main components need to be simulated: hosts, switches and links.

Mininet hosts and switches behave like isolated machines. The hosts share the computing resources of one computer. Hosts can have different IP and MAC addresses. Programs or applications can run on each host by running commands. Packets are processed by virtual switches. Users can create any network topology where hosts and switches are connected by configured links. When adding a link, besides both ends, several parameters can be configured: bandwidth, delay, queue size, and packet loss.

Mininet implements a simple controller with limited functions supporting up to 16 switches. Network managers can develop any customized controller with required functions such as machine learning techniques. Mininet also provides some basic tools to measure network performance but researchers usually design and implement their monitors.

A favourable way for simulating SDN in Mininet is connecting Mininet to an already running remote controller. Both Mininet and the specific controller can exist on the same PC. In this case, two Python scripts need to be created and executed: one for creating network topology, one for implementing a central controller.

2.5 Machine Learning for Traffic Estimations and Classification

It is simple to extract network conditions and traffic information with the global view of software-defined networks. SDN also provides packet-level feature extraction. SDN can also include artificial intelligence in routing decisions [34]. Network managers can

customize different traffic analyzing functions in an SDN controller to generate optimized forwarding rules.

Machine learning algorithms can be divided into three basic categories: supervised learning, unsupervised learning and reinforcement learning. Reinforcement learning concerns how agents should take actions in an environment to maximize reward. The biggest drawback of reinforcement learning is that it usually takes a large amount of time to converge. Timeliness is a critical requirement in traffic routing so reinforcement learning is barely studied in SDN traffic classification.

2.5.1 Supervised Learning

Supervised learning takes a labelled set of input dataset and builds a model to classify new instances into a known classes. Supervised learning consists of two phases: the training phase and the testing phase. During training the algorithm, the training data is the input paired with the correct labels. Supervised learning algorithms analyze the training dataset with different methods such as support vector machines (SVM) [35], linear regression, naive Bayes, decision trees, random forest, k-nearest neighbour algorithm and so on. After the training phase, supervised learning will select which label the testing dataset belongs to based on the training results and the chosen method. Then testing phase will evaluate the accuracy of the algorithm. Supervised learning performs well for labelled traffic but it is difficult to classify unknown applications [36].

2.5.2 Unsupervised Learning

Unsupervised learning helps finding patterns in a dataset without labels. The algorithms cluster the unlabeled data in a way that the objects share the most similarity. In

terms of clustering, two well-known algorithms are summarized below:k-means and DBSCAN.

K-Means

K-Means [37] clustering is a popular method in data mining. K-Means aims to cluster n data points into k clusters with the nearest centroids. The standard algorithm can be roughly divided into two steps.

The first step is: assigning all the data points to the nearest centroids. In the first step, two things need to be done:

- The number of k has to be decided which depends on the knowledge of the given data set.
- Randomly initializing the positions of the k centroids.

The second step is assigning the data points to the centroids and calculating the new centroids of each cluster. The second step will be iterated until reaching the convergence of the algorithm. The K-Means algorithm converges when the centroids no longer change and the clusters stabilize.

There are several obvious disadvantages of K-Means:

- K-Means tries to create the same sized clusters no matter how the data points are scattered
- K-Means does not care about the density the data is present
- K-Means does not perform well with non-globular clusters
- K-Means usually find a local optimum rather than a global one

In the case of streaming videos at constant bitrate (CBR), traffic features could have a high density. We do not know the number of classes beforehand, so a density-based approach is needed for video streaming.

DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) [38] treat clusters as dense areas of data points that are separated by sparse areas. Unlike K-Means or other partition-based algorithms, a density-based algorithm is not limited to finding spherical or globular shaped clusters only. DBSCAN performs well in finding arbitrary shapes of dense clusters.

In DBSCAN, a cluster can be described as a neighbourhood centred on a data point or a union of connected neighbourhoods centred on several data points. Two critical parameters have to be set in DBSCAN:

- epsilon (eps): the radius of an eps-neighbourhood centred on one object
- minPts: the minimum number of points to form an eps-neighbourhood

For a point that is not assigned, if its eps-neighbourhood has at least minPts of points, this point is defined as a core point. A core point is directly-reachable from all the points in its neighbourhood. A point is said to be density-reachable to a core point if it can be linked to that core point by other directly-reachable core points. All the density-reachable points form a cluster. If a point is not reachable from any other core points then it is considered as a noise point.

The reachability is an important concept of DBSCAN to define a cluster. In a cluster, all the points are mutually reachable to each other. If a point can be density-reachable from any point of a cluster, it belongs to this cluster. At the beginning of the DBSCAN algorithm, an unassigned point is picked and its neighbour points with the epsilon

are found. If the amount of its neighbour points is less than the minPts, this point is considered as a noise point temporarily. If the neighbour contains enough points, the point is a core point and the neighbour becomes an eps-neighbour. Then the cluster extends according to the reachability. A temporary noise point could join other eps-neighbours when other core points are looking for neighbours. The above steps repeat until there is no unassigned point.

2.6 Classification in SDN

As mentioned in the previous chapter, data features need to be determined before any analysis. The early approach of traffic classification in a conventional network is to group the traffic of the same port defined by INNA [39]. However, dynamic port assignment and application tunnelling leads to inaccuracy. For this reason, payload inspection is used for traffic identification [40]. Payload inspection consumes large computing resources and privacy requirements may prevent this method so machine learning is applied in recent years. Many efforts are put in learning both flow-level features and packet-level features.

2.6.1 Classification Using Flow-Level Features

A flow is usually defined by five-tuple information: the source IP address, the destination IP address, the source port number, the destination port and the transport layer protocol.

Parsaer et al. [41] classifies traffic from different applications with different Neural Network estimators. Their method needs to extract traffic information of full flows. During the data collection phase, massive data of the full flows from all the applications are collected. The obtained features include five tuples and other statistical features

(average size of the packets, packets rate on the flow and average amount of bytes rate on each flow).

Hayes et al. proposed a scalable architecture for network-wide traffic classification in SDN in [42]. Typically, traffic information collection and process are done at the controller, which could consume much resources of the controller. The data plane only contains forwarding devices such as routers ans switches. The main contribution of their work is that they design a device for the data plane that collects and processes the traffic features. The proposed device relieves the burden of the controller since it classifies the traffic for the controller. The prototype of the device shows a good performance in increasing the scalability. In SDN, the role of the data plane is to handle packets processing and forwarding only so all the network management functions can stay in the controller. The device itself needs high computing power to guarantee the forwarding performance. Traffic classification would still be an important part of an SDN controller as the processing power of a controller is much stronger than that in the data plane. The paper [42] also mentions the disadvantages of using five tuples for classification. Because of port reuse, a set of five-tuple information might no longer belong to only one flow.

Erman et al. [43] in 2007 applied K-Means in traffic classification with the knowledge of only unidirectional flow statistics. The traffic classification is at the flow level. The following flow features are used for clustering: flow duration, mean packet size, the total number of bytes, total number of packets transferred, and average inter-arrival time between packets. After clustering flows based on the flow-level features, the next step is to map new flows to the clusters. Before mapping, expert knowledge is needed to identify a cluster. Manual classification could be complicated and time-consuming.

One of their main contributions is proving the effectiveness of clustering based on

unidirectional traffic data sets. They compared testing results from the data sets including only server-to-client flows, both directions, and only client-to-server flows. Unlike DBSCAN, K-Means requires the input of k. Increasing k usually improves the clustering accuracy and it could cause over-fitting at the same time. In this paper, k is increased from 25 to 400. The server-to-client data sets always provide the highest accuracy (95 percent about flows and 97 percent about bytes). The accuracy in this paper is the percentage of correctly classified flows or bytes among the total flows or bytes.

2.6.2 Supervised Learning in Traffic Classification

Tang et al. [44] proposed an efficient approach to detect big flows. Their work is based on the fact that detecting and rerouting the big flows only can improve network management effectively. The approach has two phases: one is for reducing sampling time and the other one is for classification. They estimate the inter-arrival time between big flows to improve efficiency.

During the classification phase, improved C4.5 [45] is used to distinguish the big flows from the small flows. C4.5 builds decision trees which could be used for classification, The training features include five-tuple information and packet size. They use only one packet from a flow to be classified. Their assumption could be the packet size is stable during network communications. However, our simulation shows the packet size of all the packets from the same flow can vary.

Da Silva et al. [46] proposed a framework to extend flow features. They put traffic features into three main categories: scalar features, statistical features and Discrete Fourier Transform of the features. Scalar is defined by minimum and maximum. The statistical feature is the traffic feature's mean or variance. They use principal component analysis (PCA) [47][48] to find the principal components among all the transformation of the features. Then they use a support vector machine to classify the reduced

dataset.

Qazi et al. [49]presented a framework called Atlas to provide fine-grained and accurate application classification in SDN with machine learning. Atlas can collect active network sockets. They use decision tree C5.0 for classification and achieve 94% accuracy among forty known Android applications.

Li et al. [50] combines different classification methods in the environment of a campus software-defined network. Machine learning and deep packet inspection are applied together to achieve both speed and accuracy. Bryan Ng et al. [51] investigated the SDN classification platforms for an enterprise network. They developed a platform called Nmeta combining multiple classification methods for general or specific use. They also point out the importance of security and hardware requirements in an enterprise network.

2.6.3 Unsupervised Learning in Traffic Classification

Ibrahim et al. claimed that an application running in different networks would have different traffic features. Same applications run in different networks generate different traffic dataset. Hence, the training dataset and testing dataset might be inconsistent [52].

Bernaille et al. used K-Means to identify which application a flow belongs as early as possible [53]. There is a trend that the TCP port number becomes more and more dynamic. Inspecting the packet payload faces challenges such as encryption and processing cost. Their work applies unsupervised learning to learn an important statistical feature of packets, the packet size. The testing results show that the size of the first transmitted packets can be a very useful feature to distinguish different applications. An assumption is made so that their approach collects flow information in both directions. A simple mail transfer protocol (SMTP) server establishes a connection with the

clients during the initial stage so more packets are from the server to the clients. E-Donkey clients send the file requests during the initial negotiation so more packets are from the clients to the server. This illustrates the behaviour of the packets of different flows could be different.

Their method has two phases: offline phase and then the online phase. The offline data set is the one hour trace of a university network. K-Means algorithm is applied to cluster the packets based on the packet size only. All the packets would fall in one dimension. The clusters are mapped to the applications. After this step, flows are labelled as samples for future online clustering. During the online clustering phase, a flow's ID and the size of its first incoming packets are collected. The online classifier will find the closest labelled cluster from the offline training data for the new online clusters.

The simulation results show an inspiring performance of accuracy but it also shows an obvious drawback. Because only one feature is used for clustering an application could be fully covered by other applications. If an application is covered by other applications the accuracy would be zero. For example, Post office protocol - version 3 (POP3) flows are hidden by network news transport protocol (NNTP) and SMTP flows. Since this method has to collect the first few packets, it could fail when the start of a flow is not captured. For the same reason, it is not applicable for timely clustering.

In 2006, Erman et al. [54] compared the performance among three clustering techniques: AutoClass, K-Means and DBSCAN. One data-trace of only connection samples in three days and one data set of the entire traffic monitored in their university during one hour are used. Only collecting one hour of payloads would consume 60GB of storage. Only the flows generated by TCP-based applications are studied. Then the start and the end of a flow can be simply identified. Flow-level statistical characteristics are used so the clustering is not real-time. Commonly used data features such as mean

packet size and bytes transferred are collected. Before comparing the clustering techniques, they need to identify the flows with the help of either a port-based approach or a payload-based approach. After that, traffic flows are mapped to different categories of applications such as P2p and SMTP.

K-Means and DBSCAN are over ten times faster than AutoClass as shown in [54]. AutoClass takes hours to cluster thousands of flows which is not acceptable in timely clustering. In their work, the number of clusters, the input parameter of K-Means is set incrementally. The accuracy of K-Means is low when K is too small and it keeps increasing as K reaches 150. However, setting K too large can cause K-Means to be over-fitting. Their results show that DBSCAN can achieve good accuracy quickly and it has a huge potential in producing good clusters. Producing good clusters means larger clusters should contain a larger percentage of traffic, in other words, DBSCAN could emphasize different clusters by reasonable weights.

2.6.4 Timely Classification

Automated network management requires timely classification even at the very beginning of a flow [55]. Real-time monitoring and clustering are also needed in video streaming which belongs to long-lived big flow.

Network management needs to adapt to the dynamic network conditions such as link load and delay [56]. Nguyen and Armitage in [57] considered two main points in traffic classification: decreasing the training time and saving memory storage. Their contribution includes applying a sliding window to balance the needs of resources and precision. The sliding window here is not for real-time clustering. A sliding window of size N is considered as a sub-flow. In their work, a full flow is represented by multiple sub-flows selected at different times. Some other research, for example, [58] only uses the start of a flow for classification but the connection establishment of a traffic flow

could be missed because of its small portion compared to a full flow. Sometimes it is not possible to guarantee the integrity of a flow [57]. In [57], a supervised learning algorithm, Naive Bayes is trained and tested to distinguish the packets amongst an online game from other traffic. They manually select the sub-flows to contain the key periods of a flow, for example, the initial handshake. Then the Naive Bayes learning is trained on the combination of all the sub-flows from the original flow.

The paper [57] also proves that a small piece of a traffic flow (25 packets) can contain enough information for classification. However, the selection of sub-flows needs human inspection and specific knowledge of that particular online game. Their approach can distinguish the traffic of an online game from the other traffic. However, it lacks the ability to group all kinds of traffic. SDN network management needs to have a global view of the whole network.

Amaral et al. proposed an architecture not only for traffic classification but also for traffic feature collection [59]. Their work collects the data features of the first five packets of a flow. Their work only studies the traffic over transmission control protocol (TCP). The main reason is that they need the TCP flags which are created during the TCP initial handshake. Synchronization (SYN), one of the TCP flags, is used in the 3-way handshake or the connection establishment. By receiving the TCP flags, they can catch the start of a flow. There are two disadvantages when TCP flags have to be caught. On one hand, as mentioned in [57], it is not possible to receive the start of a flow all the time. On the other, there is a big amount of traffic carried on the user datagram protocol (UDP) too. UDP has its advantages in real-time traffic transmitting. For example, Quick UDP Internet Connections (QUIC) developed by Google aims to achieve zero handshakes to provide much lower latency [60]. Only monitoring TCP flows misses information of such applications.

The following features are collected from each of the first five packets: packet size,

timestamp and inter-arrival time. In the data collection phase, traffic features belonging to a full flow need to be collected as well: bytes count, flow duration and the number of packets. The timeliness can not be provided when a full flow needs to be obtained as mentioned in [36]. The architecture is deployed in a switch which mirrors all the traffic data obtained from the network to the SDN controller. Traffic data is generated by real Internet applications such as Youtube and Facebook. Three supervised machine learning methods are tested: Random Forests (RF) [61], Extreme Gradient Boosting (EGB) and Stochastic Gradient Boosting (SGB). Their work shows only five packets can still provide valuable information for traffic classification.

Chapter 3

Traffic Estimation using Online Clustering

SDN makes it easier to have an intelligent and dynamic network management with the opportunity to estimate the traffic. Typically, port numbers and Deep Packet Inspection (DPI) are widely used. However, port numbers may be not accurate and DPI causes high overhead and privacy issues. To guarantee high availability and efficiency in traffic estimation, we use DBSCAN to cluster traffic features while monitoring the traffic in SDN. The clustering results will be used by the controller to identify high-quality video traffic and make proper routing decisions.

The first section of this chapter shows how the messaging system works in the OpenFlow protocol and what measurements we have considered for clustering. The second section shows traffic feature reduction before clustering. The third section explains how we select the right parameters for the DBSCAN clustering algorithm. The last section of this chapter will show our clustering results.

3.1 Openflow Messaging System

An SDN controller communicates to a switch via messages. The controller sends out a port states request to the switch and then the switch sends the port states reply which include the port's statistics.

According to the OpenFlow switch specification, OpenFlow protocol defines the information of port's statistics as:

- port_no: Port number
- rx_packets: Amount of received packets
- tx_packets: Amount of transmitted packets
- rx_bytes: Amount of received bytes
- tx_bytes: Amount of transmitted bytes
- rx_dropped: Amount of packets dropped by RX
- tx_dropped: Amount of packets dropped by TX
- tx_errors: Amount of transmit errors
- rx_frame_err: Amount of frame alignment errors
- rx_over_err: Amount of packets with RX overrun
- rx_crc_err: Amount of CRC errors
- collisions: Amount of collisions
- duration_sec: Time port has been alive in seconds

The above list shows the SDN provided list of measurements. All the provided information in port statistics are accumulated values. Therefore, current measurements minus the previous measurements gives us the traffic conditions for a period of time. Besides the features directly defined by OpenFlow, mean packet size is an important derived statistical feature needs to be considered[45][53][54]. The total amount of received bytes during a time interval is divided by the total number of received packets to obtain the mean received packet size during the same interval:

$$MeanPacketSize = \frac{TotalReceivedBytes}{AmountofReceivedPackets}$$

Using the above features clustering can be done. However, some features do not contribute to the output significantly. Feature selection is the vital part of a clustering algorithm [62]. In order to identify the important factors for clustering, we first perform the well-known PCA algorithm. After that, the principal measurements are used for clustering the incoming traffic.

3.2 Feature Reduction for Traffic Estimation

The port statistics obtained from the SDN controller provides many features. To select important features, we apply PCA on the selected seven features: received bytes, transmitted bytes, mean packet size, transmitted packets, transmit errors, packets dropped by RX and collisions.

To demonstrate the selection of features, we use 10 high-quality video flows, 20 low-quality video flows and 10 CBR flows that are generated in our simulation environment. There will be a warm-up period before simulation gets steady. As we start each video streams at different times, the traffic at the cool down phase is not stable either. These two periods should be excluded from the results. The video file lasts 80

seconds and we collect 70 seconds of traffic data excluding the simulation warm up and cool down periods.

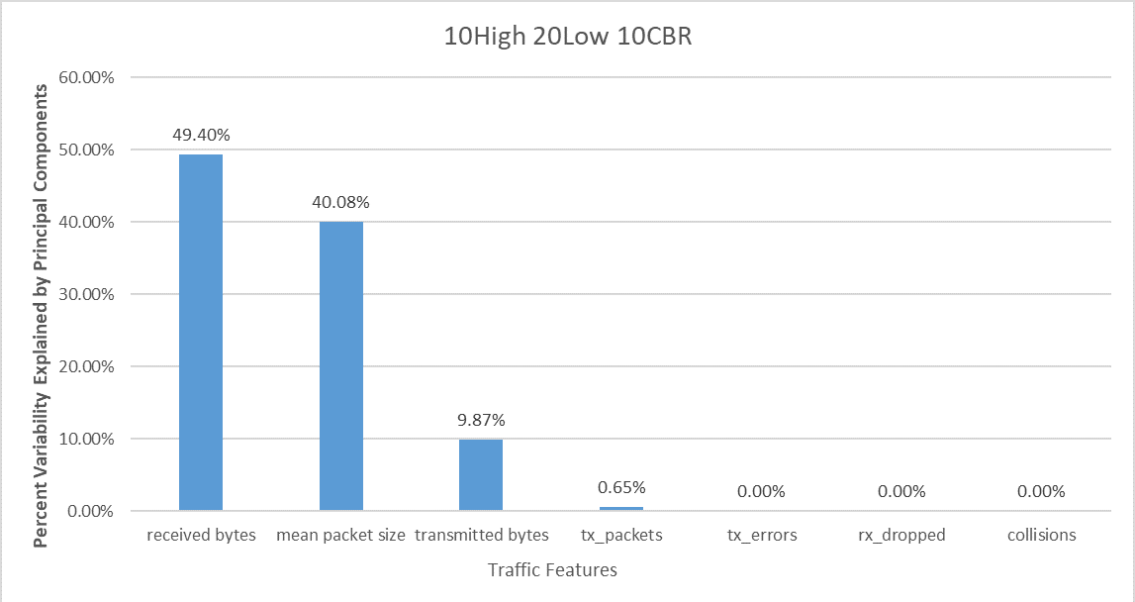


FIGURE 3.1: PCA Explained Variability of Features

Figure 3.1 shows the percent variability explained by different traffic features in PCA after mean subtraction and normalization. Received bytes explain (in other words, contain information about) 49.40% of variability and mean packet size explains 40.08%.

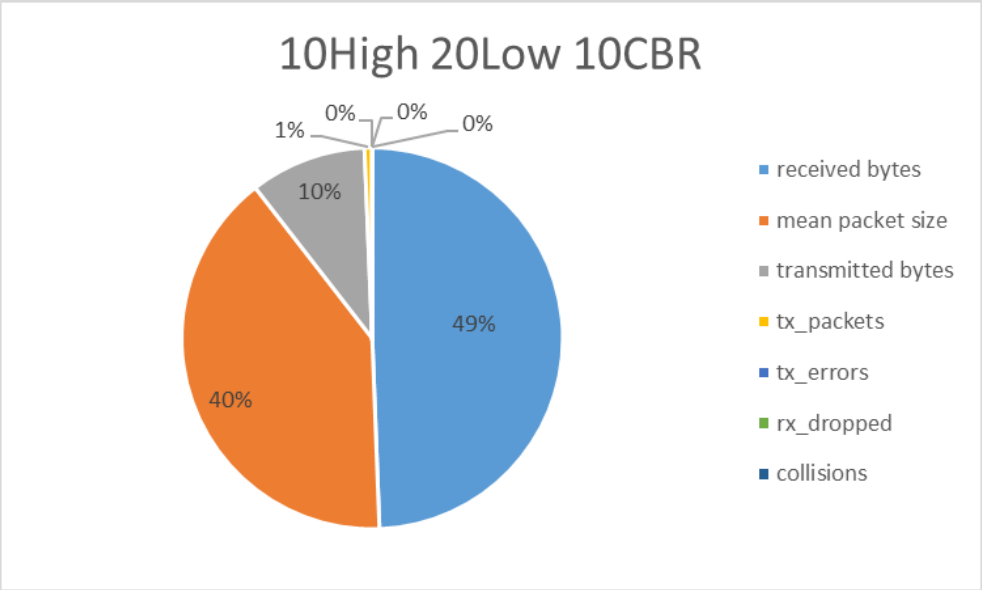


FIGURE 3.2: PCA Variances Portion

The pie chart 3.2 shows portions of each feature on variances. Received bytes and mean packet size explain 89.48% of variability. This means, using these two features the incoming traffic can be represented well. We use these two features for traffic clustering. In the next section, we further study the parameter setting in DBSCAN with these two features.

3.3 Traffic Estimation with DBSCAN

As illustrated in Chapter 2, DBSCAN needs the user to set two parameters: neighbourhood search radius(epsilon) and the minimum amount of neighbours to form a cluster(minpts).

In our study, the SDN controller collects the network statistics every second. We perform clustering every 7 seconds in order to adjust our dynamic routing scheme based on changes in the traffic. In this chapter, we demonstrate our clustering results with a simple scenario. In this scenario, 20 hosts would generate high-quality video traffic

and the other 20 hosts would generate low-quality video traffic. We collect the traffic features from the 40 hosts every second for 80 seconds so there are 3200 data points.

There are 1600 data points of high-quality video traffic and 1600 data points of low-quality video traffic. we set minpts to 100 to avoid too many clusters of noise. According to the original DBSCAN paper[38], epsilon value can be estimated by generating a k-distance graph as shown in Figure 3.3.

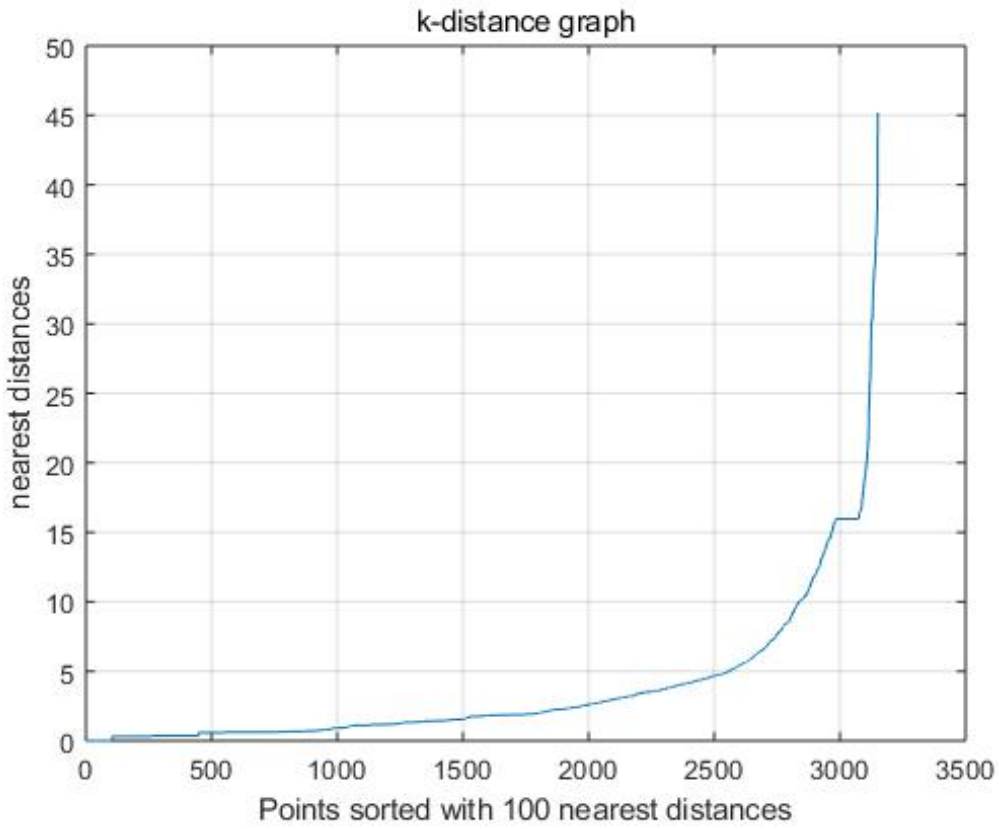


FIGURE 3.3: K-Distance Graph when HQ=20;LQ=20

There is a knee in the figure where exceeding the knee means noise would be included in the clusters. We set epsilon to 16 which is around the knee of the observed curve.

After setting minpts as 100 and epsilon as 16 we plot the Figure 3.4 of DBSCAN clustering.

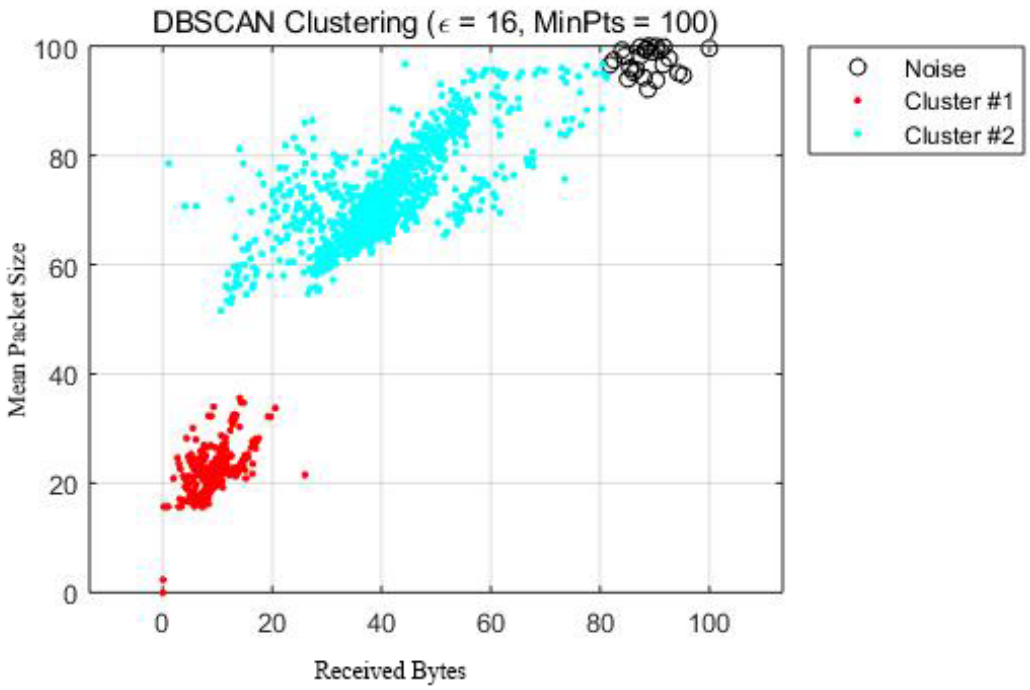


FIGURE 3.4: DBSCAN Results when HQ=20;LQ=20

In Figure 3.4, X-axis represents received bytes and Y-axis represents mean packet size. X-axis and Y-axis are both normalized to 0-100 to have a good quality of clustering. There are two clear clusters and a small amount of noise points.

3.4 Online Traffic Estimation Using DBSCAN

This section shows the clustering results from the 5 scenarios in the simulation with two dimensions of data features. There are 30 video flows and 10 CBR flows. 5 different scenarios are created by assigning different number of high-quality video flows among the total 30 video flows: 6, 8, 10, 12 and 14 high-quality video flows with 24, 22, 20, 18 and 16 low-quality video flows. Each figure has 6 plots for each scenario showing the

cluster snapshots in one time window. The time windows are consecutive and allow for online traffic estimation. One of the scenarios where 10 high-quality video flows, 20 low-quality video flows and 10 CBR flows are being generated is shown in Figure 3.5.

As described in the system design, the Mininet and POX controller starts at different times. In the simulation, we start Mininet and POX controller manually so the start time of the 5 figures could be different. We do clustering and save the results into plots every 7 seconds. The X-axis shows the number of received bytes, while the Y-axis shows the mean packet size in bytes.

In the simulation, three kinds of traffic are generated which are high-quality video traffic, low-quality video traffic and CBR traffic at 0.15Mbps. When clustering, data is normalized then the cluster results are mapped to the original data and plotted.

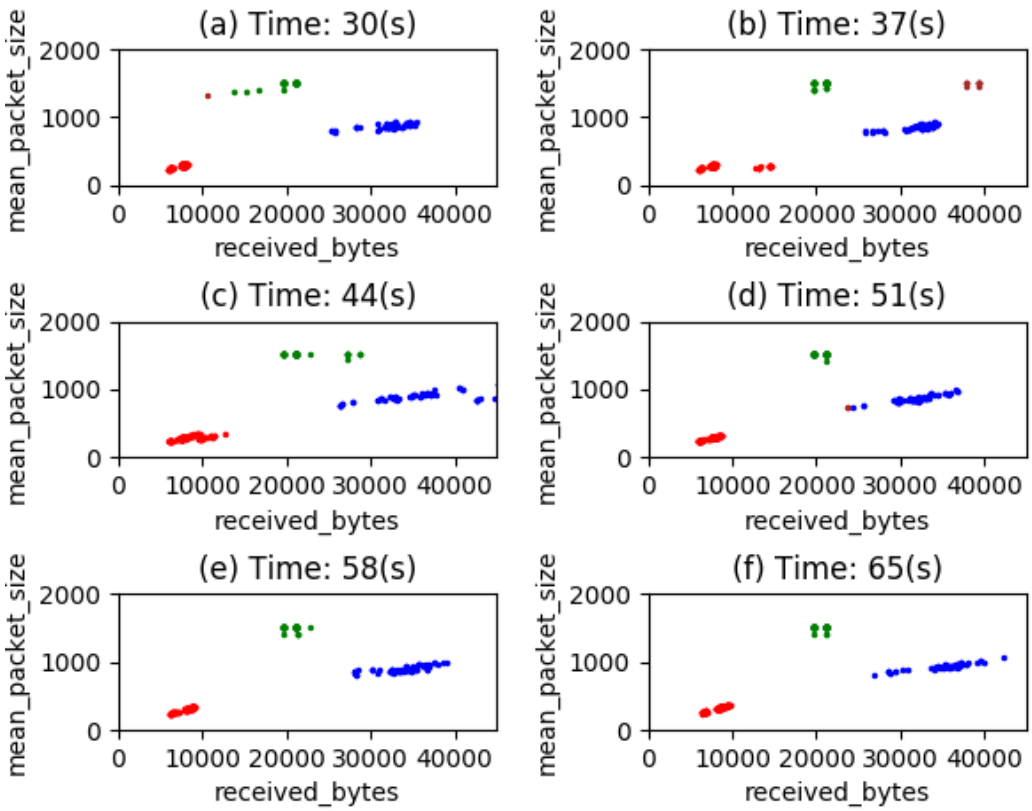


FIGURE 3.5: DBSCAN Results when HQ=10;LQ=20;CBR=10

Figure 3.5 shows the clustering results in one scenario where 10 high-quality video flows, 20 low-quality video flows and 10 CBR traffic flows are generated. Each subfigure has a timestamp showing the time when the clustering begins. The first time of clustering begins at POX controller system time 30s and the second plot (Figure 3.5 (b))starts at 37s. Each clustering has an interval of 7 seconds. Blue, red and green points represent high-quality, low-quality and CBR traffic, respectively. Noise is shown as brown points. The throughput of CBR traffic is a bit less than 2000bytes/s which matches the speed 0.15Mbps.

Note that, clustering results are different regarding the CBR traffic at time 37s. This reflects the difficulty of choosing an appropriate epsilon. Epsilon is set as 0.18 based on the k-distance after normalizing the traffic features. A smaller epsilon can exclude the noise at time 37s but may also exclude some of the data points from the other clusters. The goal of our study is to reroute high-quality video traffic to improve video quality and relieve network congestion. Therefore, clustering high-quality traffic data with higher accuracy is important.

The following four figures 3.6, 3.7, 3.8, 3.9 show the clustering results from the other 4 different scenarios in the simulation clearly. The clustering results show that DBSCAN can efficiently cluster incoming traffic. The blue cluster contains the high-quality video traffic which has the highest received bytes. The red cluster contains the low-quality video traffic which has the lowest received bytes. The green cluster contains the CBR traffic.

The CBR traffic is concentrated with little fluctuation in both throughput and packet size. Low-quality traffic data points are more concentrated than the high-quality which means the high-quality traffic disperses much more especially when it comes to the received bytes. This also shows that only the throughput would not be enough to distinguish the CBR traffic from the high-quality video traffic.

Clustering results show that with two dimensions of data, DBSCAN can cluster the incoming traffic into three clear clusters. In chapter 4, our dynamic routing scheme will make use of this online clustering technique.

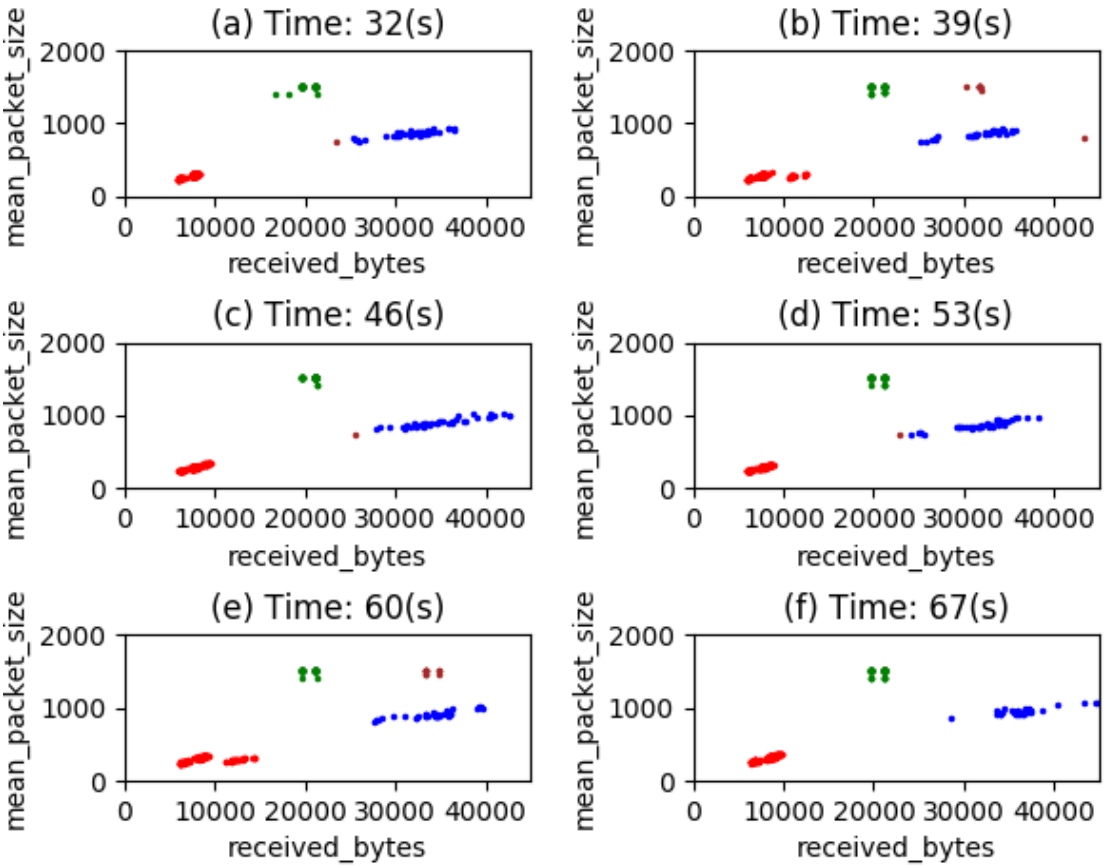


FIGURE 3.6: DBSCAN Results when HQ=6;LQ=24;CBR=10

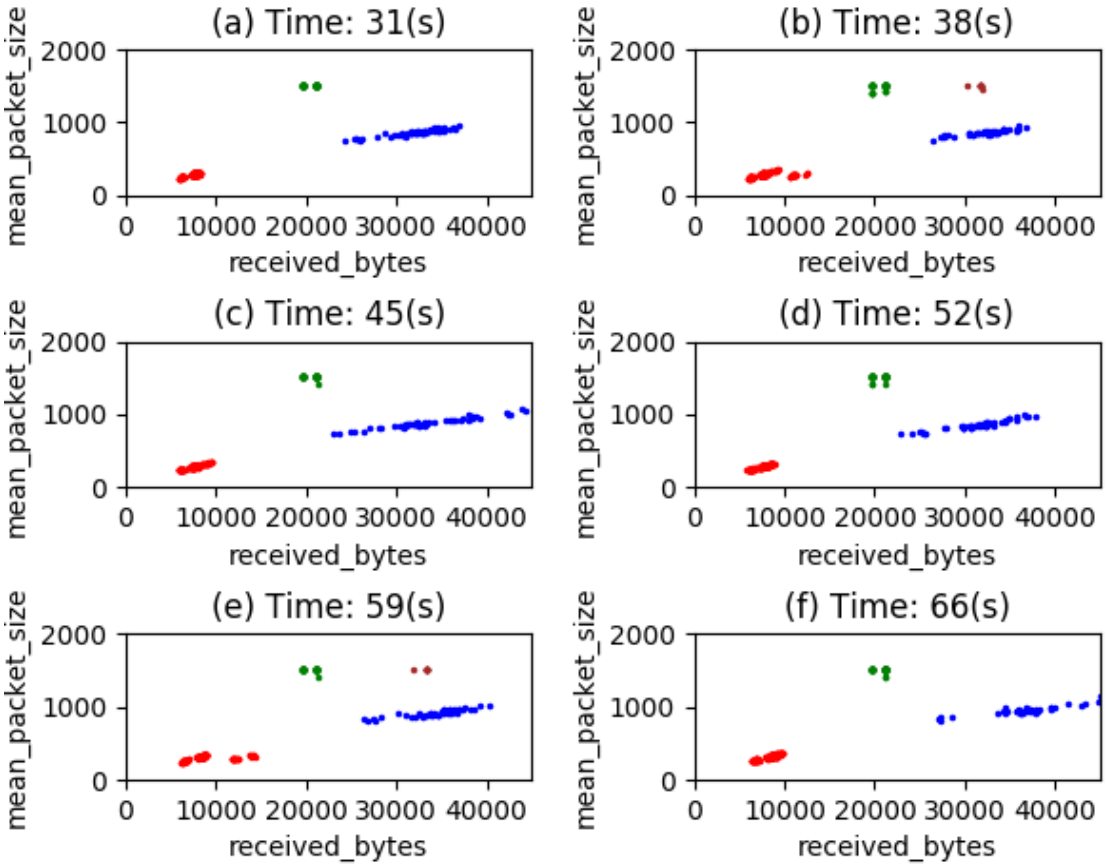


FIGURE 3.7: DBSCAN Results when HQ=8;LQ=22;CBR=10

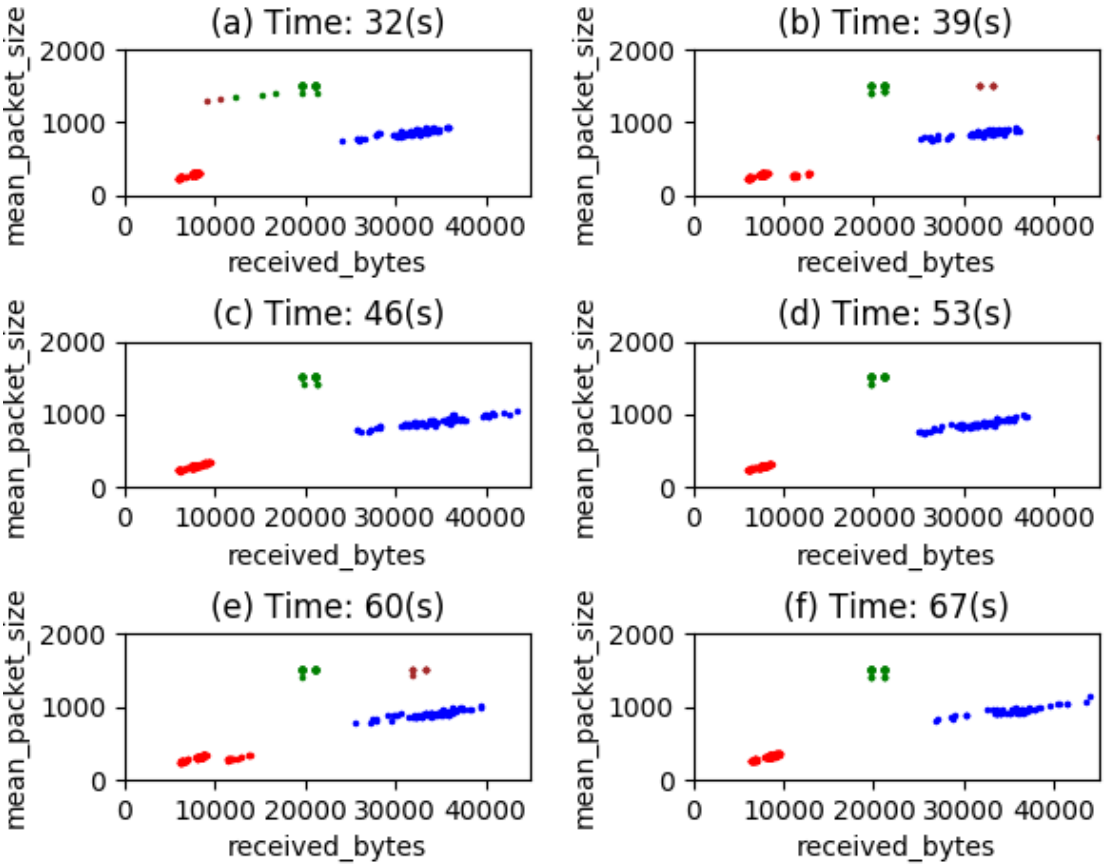


FIGURE 3.8: DBSCAN Results when HQ=12;LQ=18;CBR=10

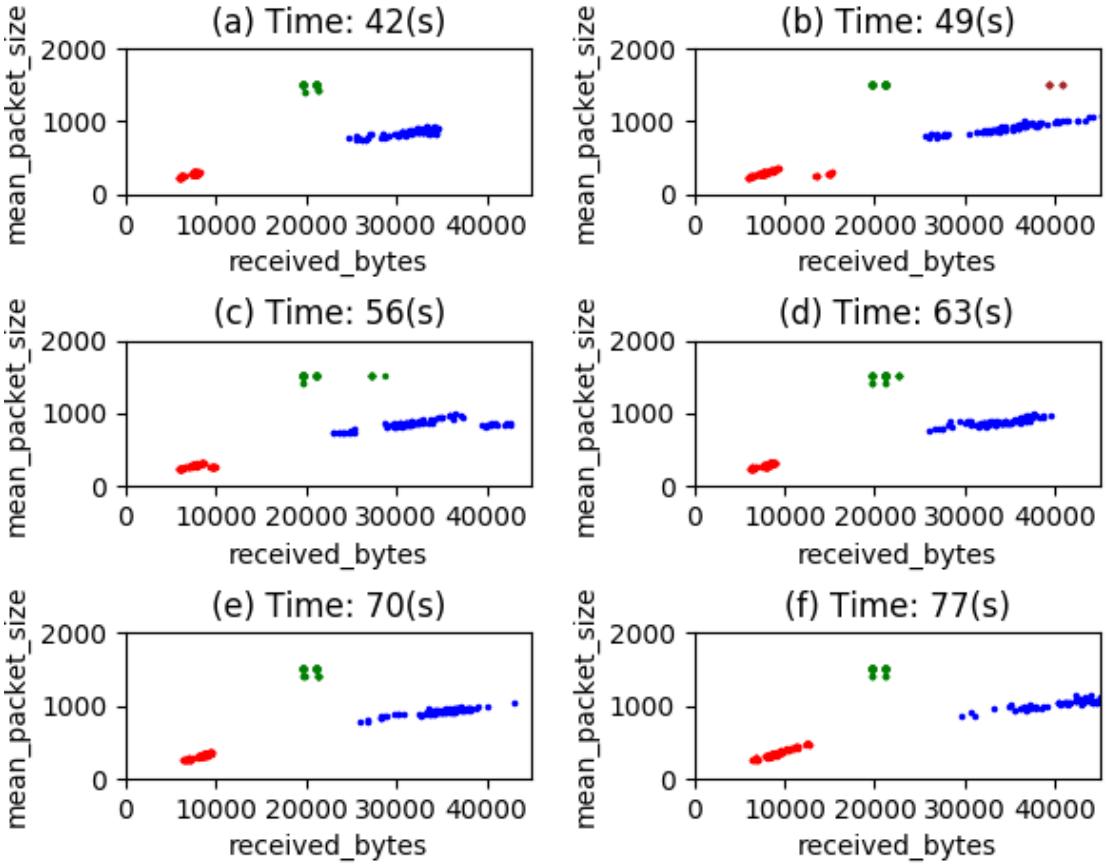


FIGURE 3.9: DBSCAN Results when HQ=14;LQ=16;CBR=10

Chapter 4

Dynamic Routing for Streaming Video over SDNs

In this chapter, we present our dynamic routing scheme over SDN. The proposed scheme makes use of the traffic estimation that is introduced in Chapter 3.

4.1 Network Topology and Control with SDN

Figure 4.1 shows the network topology and SDN controller. The POX controller is connected to all the switches in the network.

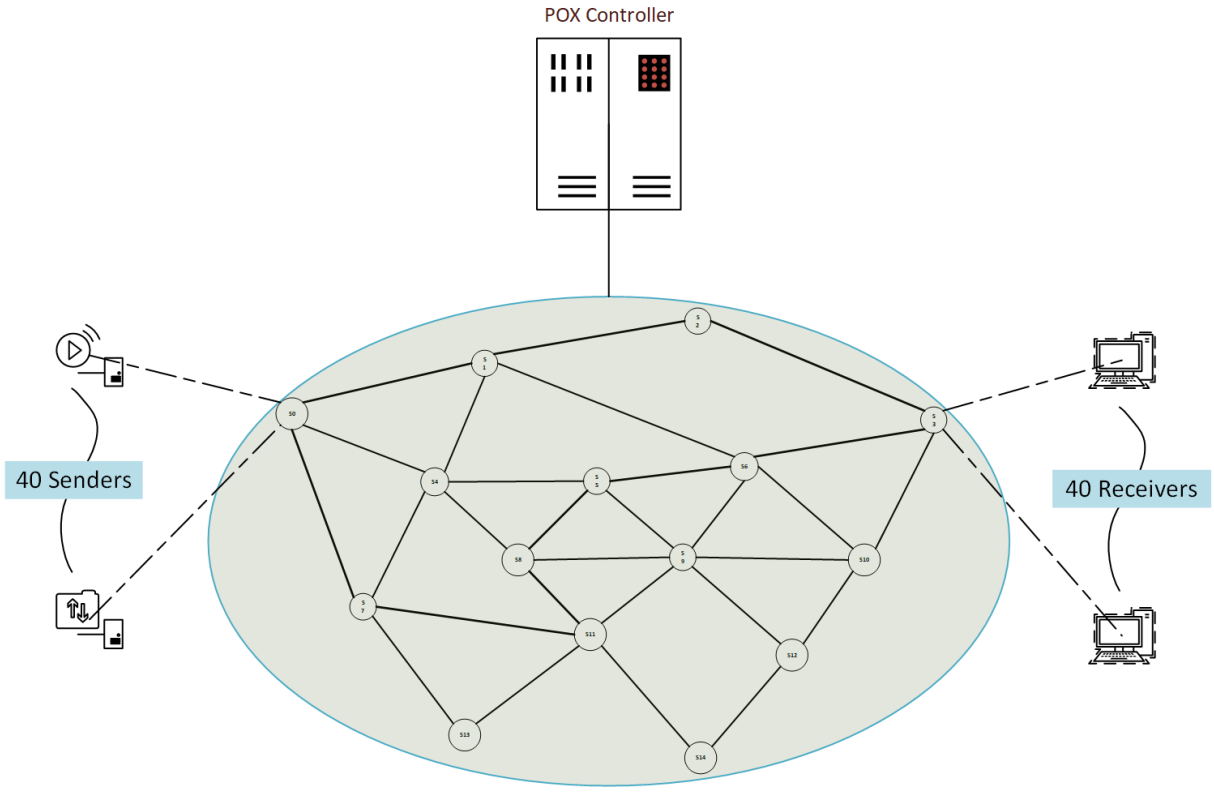


FIGURE 4.1: SDN Architecture

For demonstration purposes, we consider that all servers are connected to one ingress switch and clients are connected to one egress switch. All the incoming traffic enters the network through the ingress switch on the left as shown in the figure. Through the egress switch on the right, all the traffic is delivered to the clients or hereafter called as receivers. Receivers use vlc players to receive video streams.

To classify and route the incoming traffic, three functions need to be installed in the POX controller: network monitor, incoming traffic classifier and dynamic rerouting scheme. Details will be explained in the next section.

4.2 System Design

The flow chart in Figure 4.2 shows the whole system design. Our simulation environment is a combination of mininet and POX controller. Mininet is the network data plane that contains switches transmitting all kinds of traffic and the POX controller sends routing rules to the switches.

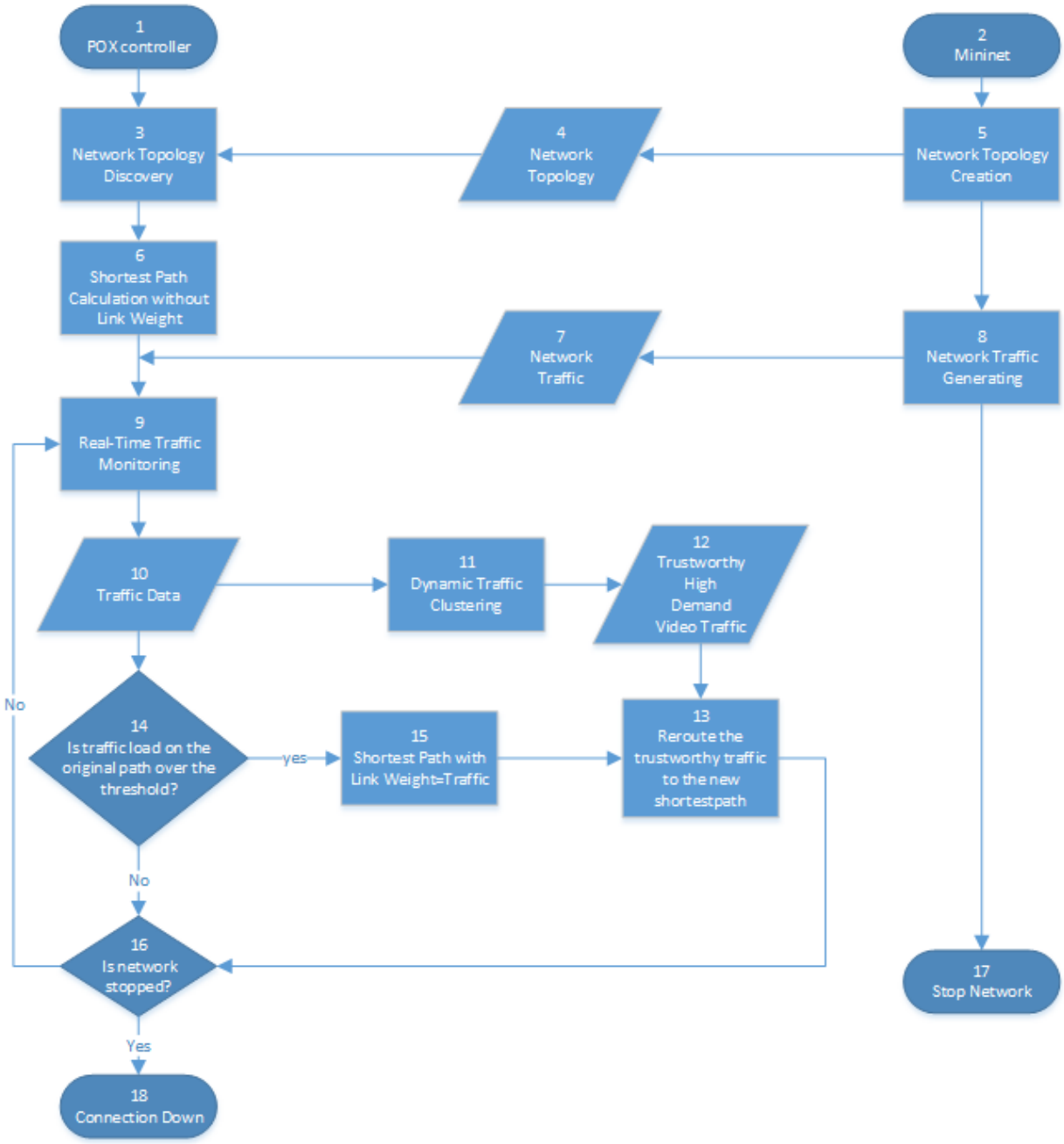


FIGURE 4.2: System Design

In our design, Mininet provides two kinds of network data to the controller: network topology and network traffic. As shown in blocks 1, 2, 5 and 8, POX controller starts running and waits for the Mininet network to connect. Mininet creates network based on the topology settings and then generates traffic. Then controller discovers the topology and keeps monitoring the traffic.

4.2.1 Network Topology Discovery

Mininet creates the network topology with hosts sending or receiving different types of traffic. The POX controller discovers the topology which is created in Mininet and it monitors the traffic generated in Mininet.

Network topology is discovered by a component in the POX controller called openflow.discovery. This component of POX controller listens to link events including the link attributes by sending and receiving link layer discovery protocol (LLDP) messages from switches. Then, the topology is saved as a graph. A link event is sent to the controller when a link is up or down. Then the link is added or removed. Link attributes include the data path IDs of the connected switches and the ports used on the switches. During the establishment of the network, edges are added with weight 1 by default.

4.2.2 Shortest Path with Default Weight

Without the use of our proposed dynamic routing technique, the controller will first calculate the shortest path with all edges weighted 1 by default. Dijkstra's algorithm is used and a shortest path is discovered. The green path in Figure 4.3 is one of the original shortest paths.

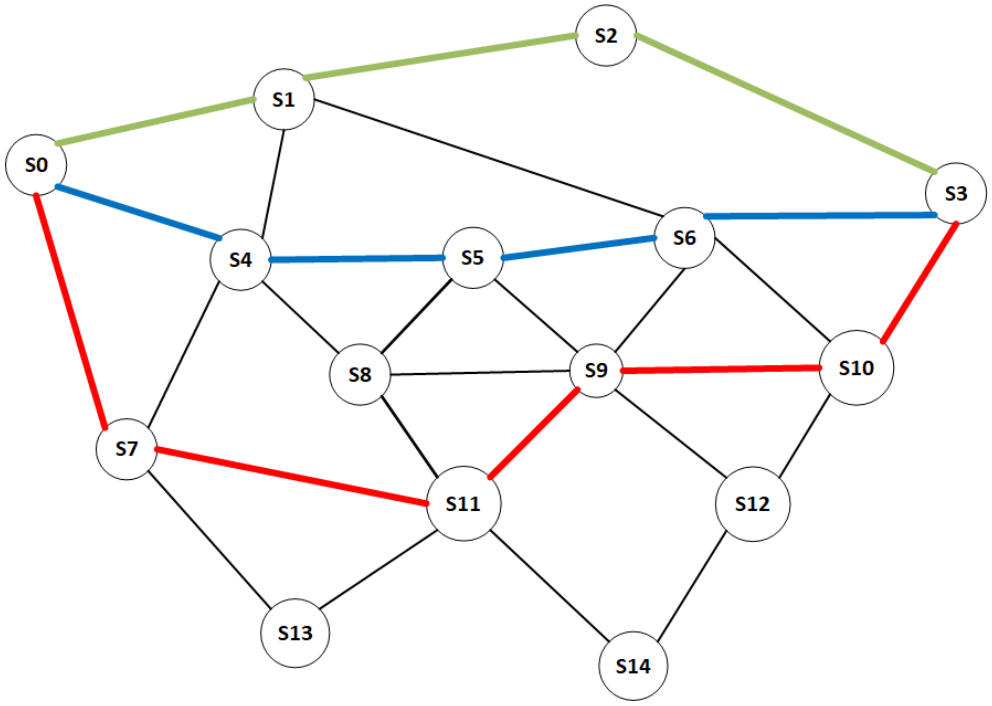


FIGURE 4.3: Network Topology with Path Selection.

Then the controller would install flow tables in switches s0, s1, s2 and s3 along the path with the calculated shortest path to make the ingress switch connected to the egress switch. A flow table should contain match fields such as ingress port and instructions for the packets. For example, s1 connects to s0, s2, s4 and s6 via port 1, port 2, port 3 and port 4 respectively. The controller would install a flow table in s1 to forward incoming packets from port 1 to port 2. If no dynamic rerouting scheme is implemented, all the packets arriving at s0 will be forwarded to the same path according to the original flow table.

4.3 Dynamic routing with Online Traffic Estimation

To cluster and route the incoming traffic, two kinds of data need to be collected:

- Real-time global network condition

- Latest features of the incoming traffic at the ingress switch for clustering

Real-time traffic load on all the links of the whole network should be collected and updated frequently so the controller can choose the optimal routing path in a short responding time. The default port states update rate defined in OpenFlow protocol is 1 second. We want the controller to react to the network changes timely so the time interval between each monitoring reading is kept as 1 second.

Besides, the real-time link load information, dynamic routing decisions needs traffic estimation. For this purpose we use the technique introduced in Chapter 3. As discussed before a sliding window is applied to store the latest historical features of incoming traffic captured at the ingress switch. Then, these features are fed into DBSCAN algorithm.

4.3.1 Data Collection

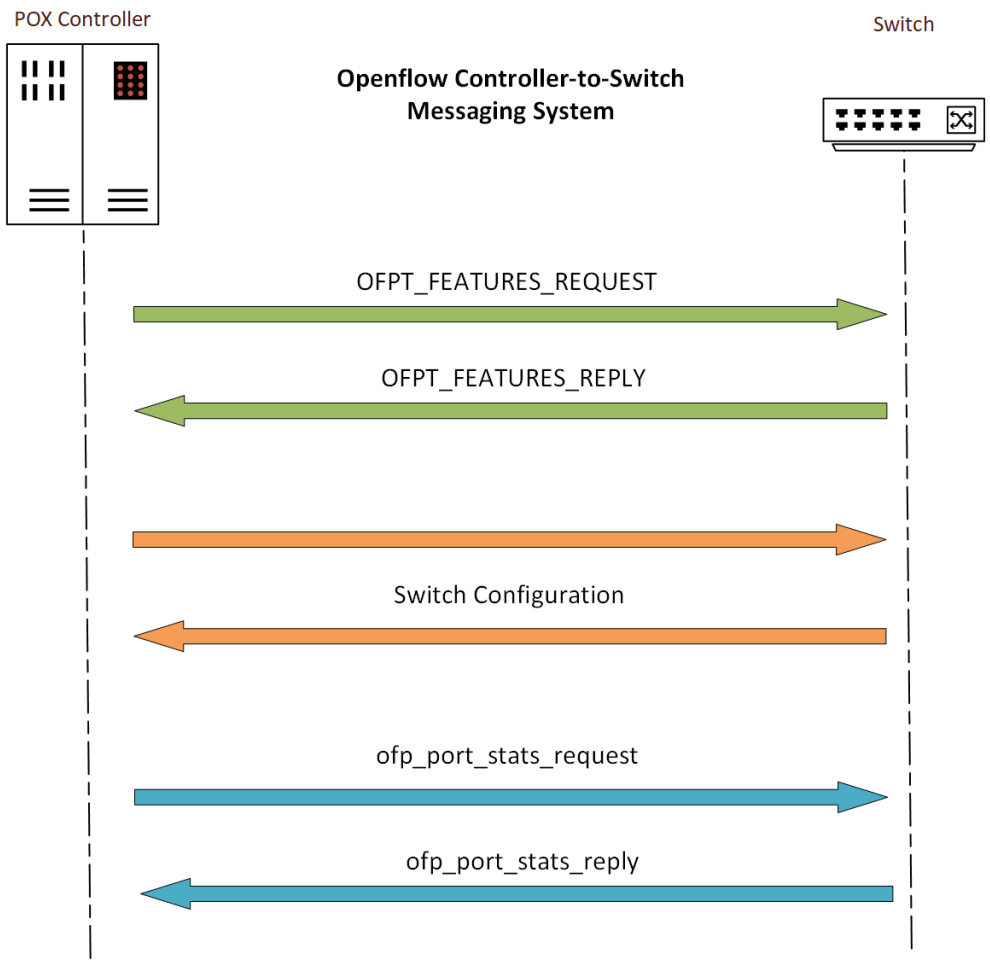


FIGURE 4.4: OpenFlow Controller-to-Switch Messaging System

The controller-to-switch messaging system of the OpenFlow protocol is briefly mentioned in chapter 3. As shown in Figure 4.4, a controller can request features of a switch or configure a switch by sending messages to switches. In our implementation, the controller sends port states request every one second to all of the switches to collect the traffic transmitted through all of their ports. Link load equals to the number of transmitted bytes via one of the connected ports' divided by time interval:

$$LinkLoad = \frac{TransmittedBytes}{TimeInterval}$$

4.3.2 Data Pre-processing

Link load in bytes is assigned to each link as its weight and the link weight is updated every second. Each link has same link capacity so link load can reflect the network condition. The collected traffic data need to be pre-processed for two reasons:

(1) Besides video traffic or CBR traffic, ARP packets are also monitored at the ports. ARP packets are sent before the first time real traffic is sent. The amount of ARP packets is small and they should not be included in clustering. As mentioned in the previous chapter, the clustering function starts late enough to avoid clustering the ARP packets.

(2) Another reason for pre-processing is that the SDN controller sends extra port state requests to switches. In our design, there are two kinds of port state requests: one is triggered by time and one is triggered by event. We make the controller send extra port states requests periodically so we expect that each measurement has the same time interval. By default in OpenFlow protocol, the controller sends port states messages to check or configure the switches which are triggered by packet events or link events. We want the messages sent and received on a fixed 1 second time interval then we can calculate traffic features. However, extra messages triggered by events lead to more messages, in other words, shorter and unpredictable time intervals between each time of monitoring. For this reason, we remove the record of a port states reply if it arrives less than 400ms later than the previous one.

After pre-processing, we use DBSCAN to cluster incoming traffic. DBSCAN clusters traffic packets with similarities. DBSCAN should give us three clusters: a cluster of high-quality video traffic, a cluster of low-quality video traffic and a cluster of background traffic. The traffic features are collected at the ingress switch so we know which flow a packet belongs to. If a flow contains the packets which are all clustered as high-quality video traffic then this flow can be safely considered as a high-quality video traffic flow. Then we can reroute these flows based on our rerouting strategy.

4.4 Rerouting High-Quality Video Traffic

With the knowledge of clustering and network condition, the controller will calculate the weighted shortest path and reroute the high-quality traffic flows if needed. We assume that there are more than 1 disjoint paths in the network and all links are idle at the initial stage of the network. As mentioned in the previous section, the link's weight equals to the link load in bytes. The sum of the weights could be tens of thousands of bytes. Let us consider the graph in Figure 4.3 as an example. At first, all the links are assigned weight 1. The original shortest path is the green path. When traffic goes through the green path, its total weights go beyond tens of thousands since our algorithm assigns link load as weight. Then the blue path will be selected to carry some of the high-quality traffic, again with link weights exceeding 1. In that case, red path can be selected as the shortest path.

A congesting happens when a link is fully loaded. We set a maximum link load threshold to 3900kbps which is 98% of the 4Mbps link capacity, to indicate if a link is fully loaded. If a link's load is over 3900kbps we can safely consider it congested. The rerouting process will not stop until the traffic load on the original path is under the maximum link load threshold.

In our algorithm as shown below, the controller reroutes several high-quality video flows to the uncongested path each time a rerouting decision is made. In our study, each time we select a bundle of 3 flows to reroute. Rerouting a small number of flows may cause the controller take too much time to relieve the network congestion. On the other hand, rerouting a large number of flows at once may cause congestion on the candidate path. The number of flows rerouted each time is a trade-off between efficiency and precision. The POX controller will stop rerouting if the congestion is relieved or simulation ends.

After rerouting, all the three output links (s0-s1, s0-s4 and s0-s7) connected to the

Algorithm 1: Rerouting High-Quality Video Traffic Flows

```
Input: Incoming Traffic Flows, Network Topology, Network Conditions
Output: Output Path
if Network Topology is Discovered then
    Set each link's weight to 1;
    Calculate the original shortest path with Dijkstra's algorithm;
else
    Send LLDP messages;
end
while Link Load on Original Path > 3900kbps do
    Change each link's weight to its current link load;
    Calculate a new shortest path with Dijkstra's algorithm;
    if High-Quality Video Flows on Original Path >= 3 then
        Reroute a high-quality video flow bundle to the new shortest path;
    else
        Reroute the rest of high-quality video flows to the new shortest path;
    end
end
```

ingress switch in our topology would be utilized. This technique can be generalized to other topologies as long as there are several disjoint paths between ingress and egress switch.

Chapter 5

Performance Evaluation

5.1 Performance Metrics

In this section we define the metrics that are used to evaluate the performance of the proposed scheme.

5.1.1 Link Load

In the literature, link load, delay and dropped packets have been used as performance indicators for streaming video [63].

Link load is the transmitted bytes divided by time interval as defined in the previous chapter. In our simulation, all the links have the same capacity so higher link load means higher link utilization. As mentioned in Chapter 4, three output links (s0-s1, s0-s4 and s0-s7) are connected to the ingress switch and they are the most congested link on the whole path. We show their link load to indicate the path performance. We put them together to show the network throughput of the whole network.

5.1.2 Dropped Frames

Video streaming is sensitive to network delay. Less delay usually means higher video streaming quality [64]. In our simulations we use vlc player which compares the stream clock with the system clock and marks packets as dropped if they have delay higher than a certain threshold. The stream clock is included in the stream to predict when a frame should be displayed. The system clock is the local clock which is set and continuously updated as receiving a stream. As soon as a receiver receives a frame, it will decode it and compare the local system clock to the stream clock. Delayed packets will be discarded. The threshold is set to its default value in our simulations which is equal to 20ms.

The rest of this chapter includes our results gathered in four subsections: link load without any rerouting, link load with rerouting, comparison of the amount of dropped frames with and without rerouting and comparison of the average delay of dropped frames.

5.2 Simulation Settings

We use the network topology mentioned in Chapter 4. Each link has 4Mbps capacity. Forty senders represent different kinds of servers in SDN with different traffic characters. The total number of video traffic flows is 30. Different combinations of high quality and low-quality video traffic creates different scenarios. Network traffic also contains 10 CBR traffic flows as background traffic.

There is a trade-off between timeliness and accuracy in choosing the right window size. The data would lose timeliness if the window size is too large. It would also be difficult to distinguish the data points from the noise if the window size is too small.As mentioned in the related work [57], 25 can be considered as the least amount of packets

to form a cluster. In our simulation setting, there are at least 6 hosts sending high-quality video traffic. If we store traffic data of the past 7 seconds, we can have at least 42 data points in one cluster. Then we can have enough data and guarantee timeliness at the same time. The rest of the simulation parameters are summarized in Table 5.1:

Parameters	Values
Number of Switches	15
Number of Links	28
Link Capacity	4Mbps
Max. Network Throughput	12Mbps
Vlc Delay Threshold	20ms
Monitoring Frequency	1s
Window Size	7s
Number of Runs for Each Scenario	5
Size of a Flow Bundle	3
Link Load Threshold	3900kbps

TABLE 5.1: Simulation Settings

Five scenarios are created by assigning different numbers of high-quality traffic flows among the total 30 flows: 6, 8, 10, 12 and 14 high-quality video flows with 24, 22, 20, 18 and 16 low-quality traffic flows. The rate of each CBR flow is 0.15M bits/s and it is the same for the five scenarios.

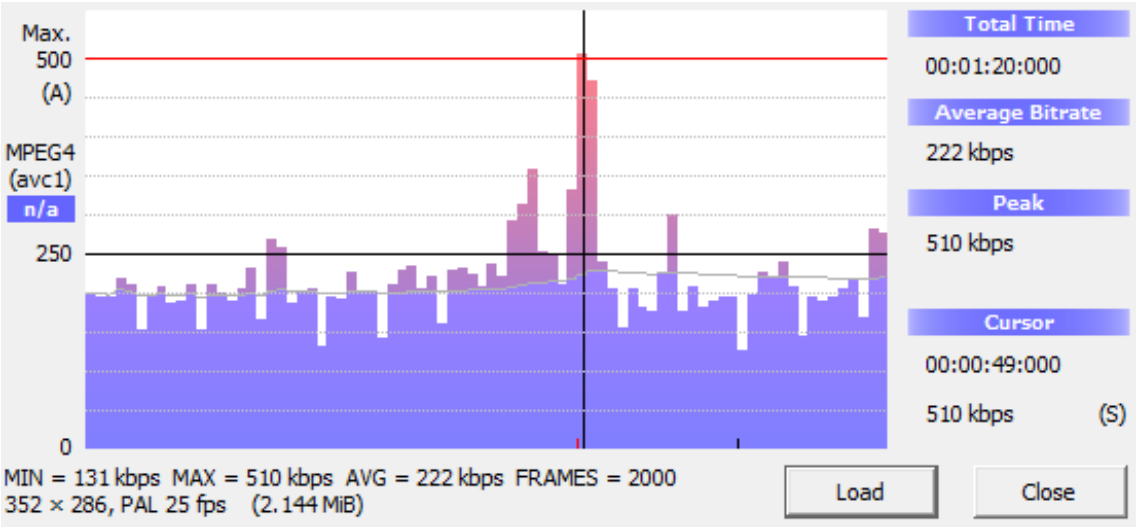


FIGURE 5.1: Bit Rate Analysis of the Source Video

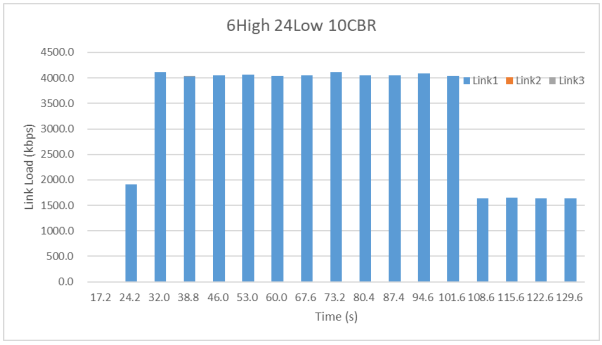
The high-quality video is coded in cif and low-quality video is coded in qcif formats with the same source video file. Basic information and bitrate analysis of the source video file is shown in Figure 5.1. The video has 2000 frames and the total length of the video is 1 minute and 20 seconds. The same video is sent by the servers and replayed at the clients.

The average bit rate of the source video is 222 kbps and at time 49s the bit rate reaches its peak of 510 kbps. Each traffic flow carries one source video. The bit rate from time 45s to 50s is much higher than the average and the link load would increase accordingly, which will be shown in the link load results. There is also a period of bit rate drop after the peak.

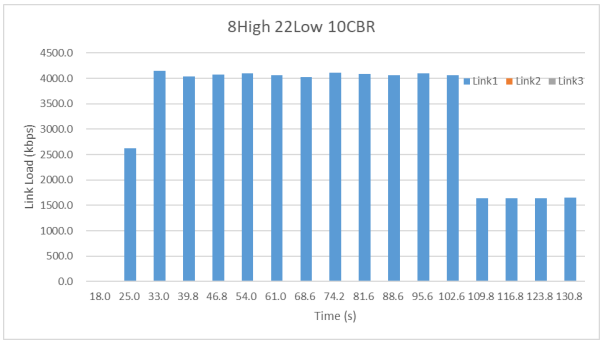
5.3 Link Load without Dynamic Routing

This section includes the link load on each output link connected to the ingress switch when our dynamic routing scheme is not implemented. This case study serves as baseline. When there is no dynamic routing, all the traffic will go through the original

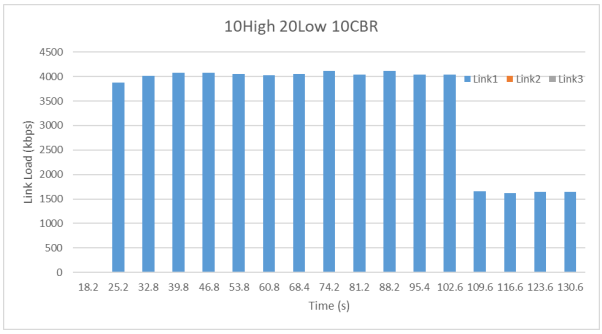
shortest path calculated with the unweighted graph.



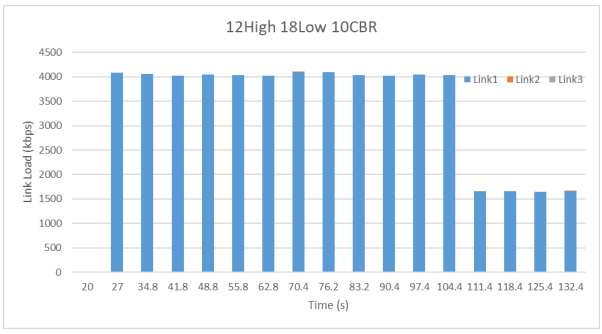
(A) HQ=6;LQ=24;CBR=10



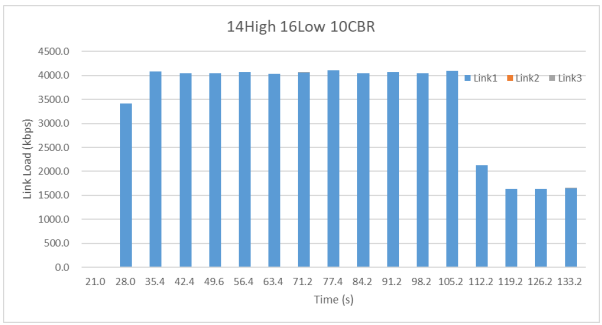
(B) HQ=8;LQ=22;CBR=10



(C) HQ=10;LQ=20;CBR=10



(D) HQ=12;LQ=18;CBR=10



(E) HQ=14;LQ=16;CBR=10

FIGURE 5.2: Link Load without Dynamic Routing

Note that, in Figure 5.2 the X-axis is the system time of the POX controller. The controller starts before the mininet so there is no traffic monitored at the beginning.

As mentioned in chapter 3, mininet is started manually after the POX controller starts running. Traffic flows starts at a different random time but within a narrow time interval to avoid traffic burst and introduce randomness. 5 different seeds are used to generate different random start time.

For this reason, the first bars of all the sub-figures are at different time. The first columns of all the sub-figures are different because in some scenarios not all the traffic starts at the first measurement. In sub-figures 5.2a and 5.2b, only part of the hosts starts sending at time 25s. As two more high-quality video flows are transmitted in sub-figure 5.2b than 5.2a, at time 25s, more traffic starts being transmitted. Same pattern can be found when finishing video transmission. In Figure 5.2e, at time 112.2s, there are still some video flows being transmitting shown in the forth last column.

The link capacity is 4Mbps. Starting from the second column in all the five scenarios, link1 is fully loaded and the other two links are empty. It is obvious in the figure that link load reaches its threshold and congestion would happen.

Note that, the length of the source video is 1 minute and 20 seconds. After a bit less than 80 seconds, video streaming finishes so there is a drop after that. After finishing video streaming, only CBR traffic is being transmitted. The CBR traffic is also transmitted through the whole simulation as background traffic.

5.4 Link Load with Dynamic Routing

In this section, we present the results of the dynamic routing scheme proposed in Chapter 4. The simulations settings are the same as the previous section.

5.4.1 6 High-Quality Video Flows and 24 Low-Quality Video Flows

Figure 5.3 shows the link load when 6 high-quality video traffic flows, 24 low-quality video flows and 10 CBR flows are generated.

At time 24.2s traffic starts being generated and transmitted and the link load on link 1 is only around 1400kbps which is much lower than threshold. However, at time 32.0s, link load reaches the capacity threshold. According to our algorithm, three flows from

the clustered high-quality flows will be rerouted. At time 38.4s, there are three flows on link 2 but link 1 is still congested. In the next interval three more flows are rerouted to link 3. After this period the traffic on link 1 is lower than the threshold as seen from Figure 5.3.

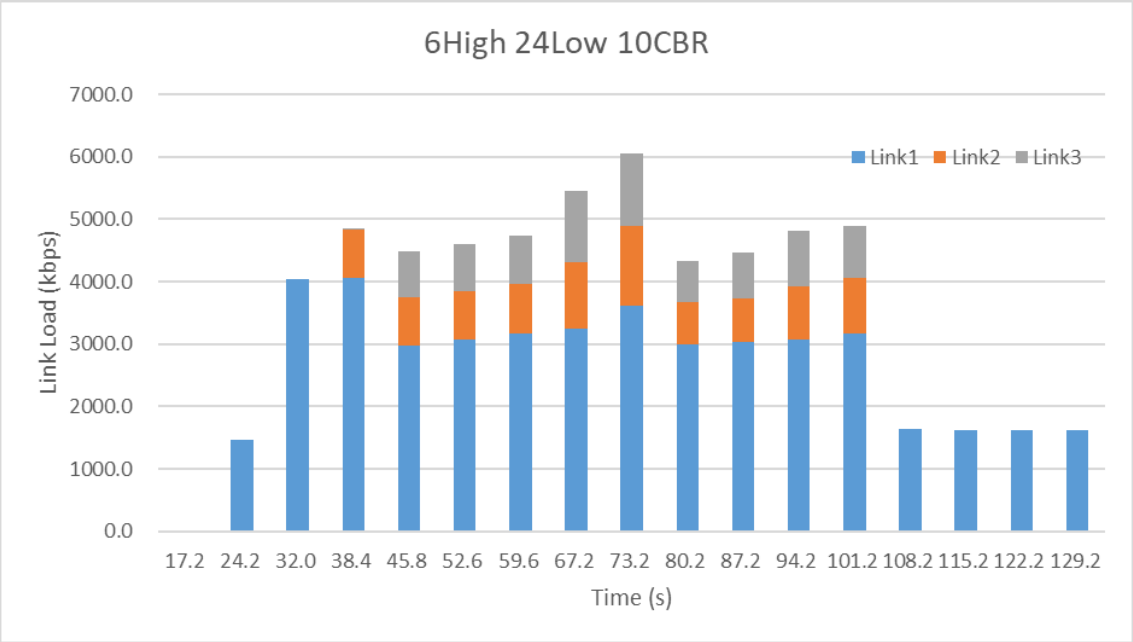


FIGURE 5.3: Link Load when HQ=6;LQ=24;CBR=10 with Dynamic Routing

Note that, link load follows the pattern of the original video content. The link load increases and reaches its peak at time 73.2s. At time 73.2s, the video streaming is at its playing time 50s so the bit rate is also the highest which causes the link load to peak in the figure. This fluctuation is more obvious on link 1 and link 2 because they have high-quality video traffic and link 3 only has low quality video traffic and CBR traffic.

5.4.2 8 High-Quality Video Flows and 22 Low-Quality Video Flows

Figure 5.4 shows the link load when 8 high-quality video traffic flows, 22 low-quality video flows and 10 CBR flows are generated.

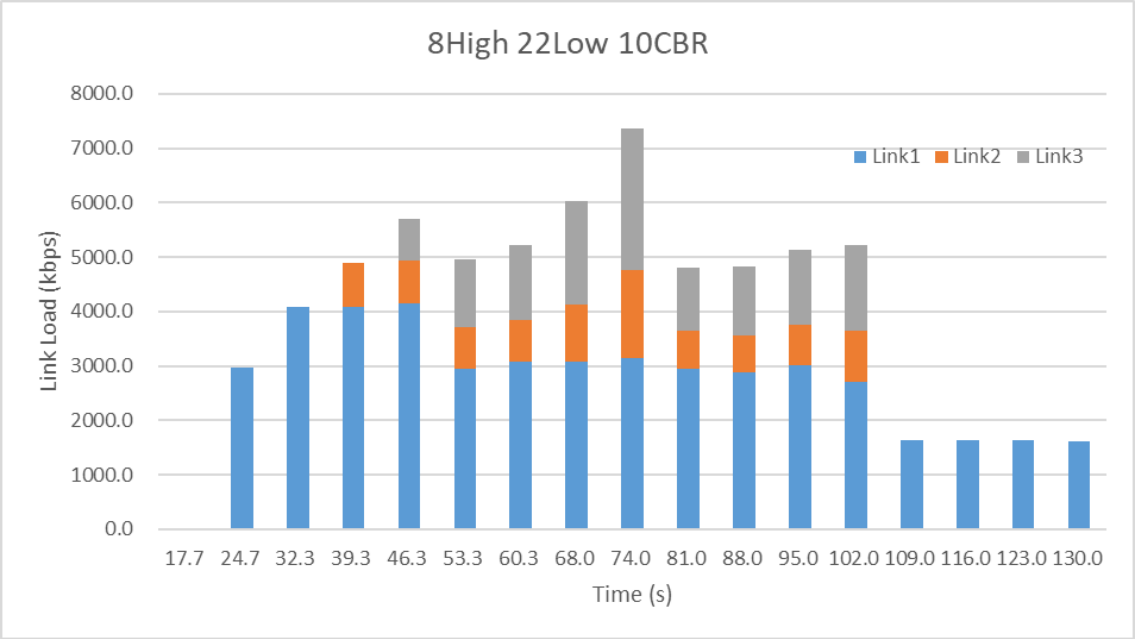


FIGURE 5.4: Link Load when HQ=8;LQ=22;CBR=10 with Dynamic Routing

The controller detects the link load exceeds the threshold at time 32.3s. Three flows are rerouted to link 2 and 3 more flows are being rerouted to link 3. At time 46.3s, 6 high-quality video flows are rerouted but link 1 is still congested. Two high-quality flows are left and they are rerouted to link 3. Then the congestion on link 1 is relieved. Link load reaches its highest point at time 74.0s as the video has been streamed for 50 seconds.

In this scenario, the controller takes three rounds of monitoring and rerouting to reroute enough high-quality traffic.

5.4.3 10 High-Quality Video Flows and 20 Low-Quality Video Flows

Figure 5.5 shows the link load when 10 high-quality video traffic flows, 20 low-quality video flows and 10 CBR flows are generated.

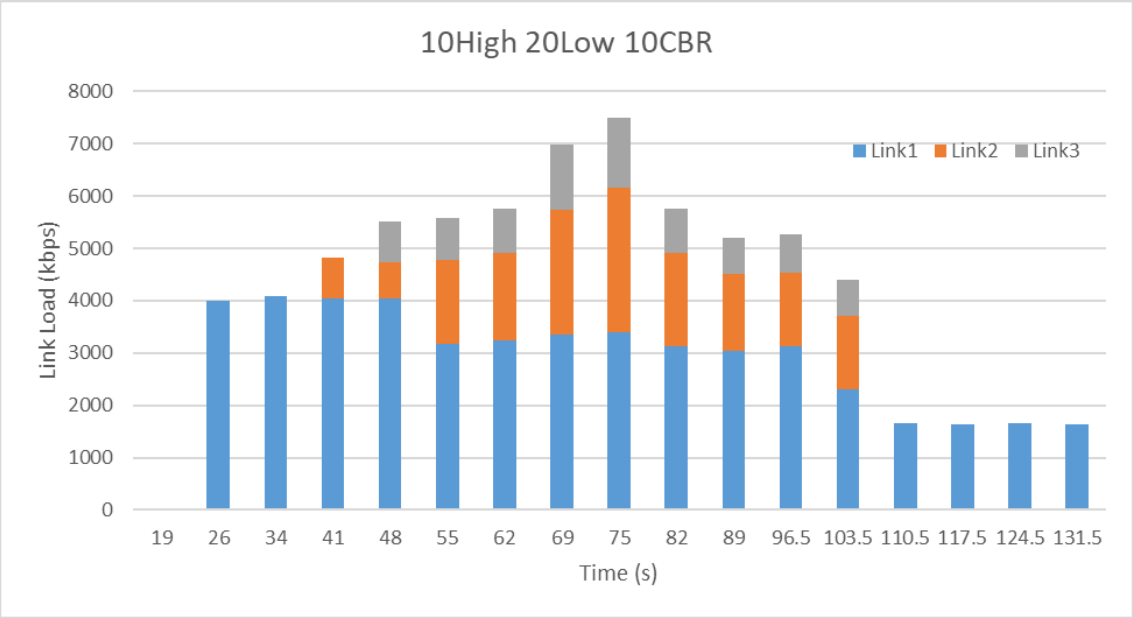


FIGURE 5.5: Link Load when HQ=10;LQ=20;CBR=10 with Dynamic Routing

At time 26s, the link load on link 1 is over the threshold but there is no traffic rerouted as we can see at time 34s. The reason is that the controller has not collected enough traffic data for clustering. As mentioned, the incoming flows does not start at the same time so at time 26s some flows has not started transmitting yet. If the controller cannot collect enough data for clustering then it cannot distinguish and reroute the high-quality video flows. After rerouting, 3 high-quality video flows are on link3 and 6 high-quality video flows are on link2. From time 55s, one high-quality video traffic flow remains at link 1 so we can see the link load slightly increases from time 55s to 75s.

As described in the system design, the intelligent controller uses the traffic data in the past 7 seconds which means the controller starts to reroute at the second round.

5.4.4 12 High-Quality Video Flows and 18 Low-Quality Video Flows

Figure 5.6 shows the link load when 12 high-quality video traffic flows, 18 low-quality video flows and 10 CBR flows are generated. The overall trend is similar as shown in the previous scenarios.

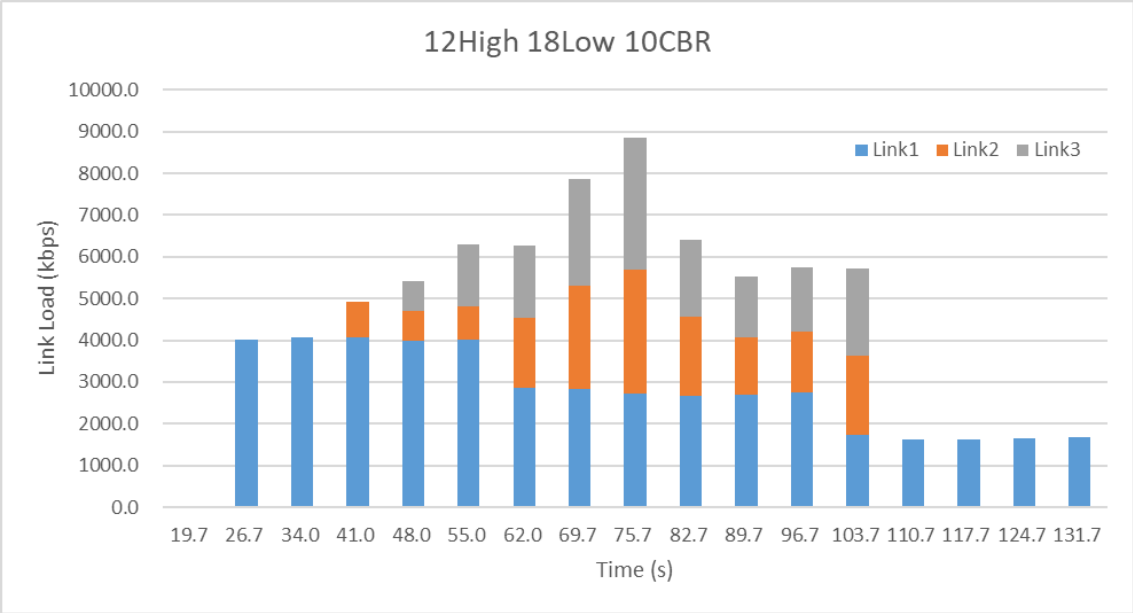


FIGURE 5.6: Link Load when HQ=12;LQ=18;CBR=10 with Dynamic Routing

From time 26.7s to 55.0s, link 1 is congested for 28.3s which is much longer than the previous. This scenario many high-quality video flows and we reroute the flows 3-by-3 so it would take much longer time to complete rerouting. Longer duration of congestion would cause more dropped frames and delay, which would be illustrated later in the subsection of dropped frames. At time 26.7s, there is no rerouting because of the same reason mentioned in the previous sub-section. From time 34.0s to 55.0s high-quality video flows are rerouted to link2 and link3 step by step.

At time 62.0s, 6 high-quality flows are on link 1 and another 6 flows are on link 2. Link 3 does not carry any high-quality video traffic so the link load on link 3 does not increase from time 62.0s to 75.7s.

5.4.5 14 High-Quality Video Flows and 16 Low-Quality Video Flows

Figure 5.7 shows the link load when 14 high-quality video traffic flows, 16 low-quality video flows and 10 CBR flows are generated.

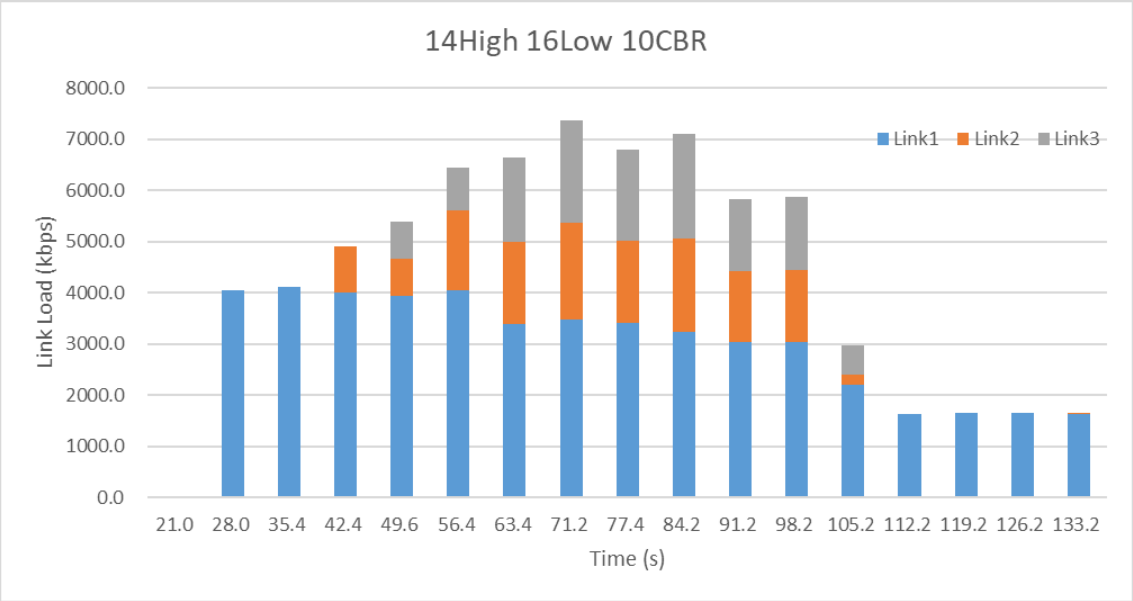


FIGURE 5.7: Link Load when HQ=14;LQ=16;CBR=10 with Dynamic Routing

The controller detects the congestion at time 30.5s and solves it at time 58.5s. At time 58.5s, dynamic routing results in 6 high-quality flows to be on link 2 and another 6 to be on link 3. Two high-quality video flows are left on link 1 so a slight peak can be seen in the figure from time 58.5s to time 71.8s. This scenario contains highest portion of high-quality traffic flows so the total link load is higher than others.

5.5 Amount of Dropped Frames

Video streaming is sensitive to network delay. During video streaming, if a frame arrives later than a threshold the application on the receiver side would drop the frame since delayed frames will be obsolete for the video. Dropped frames may cause video

freezing and distortion and negatively impact the Quality of Experience (QoE) of users. This section compares the amount of dropped frames when our proposed scheme is used and when it is not used.

5.5.1 6 High-Quality Video Flows and 24 Low-Quality Video Flows

Figure 5.8 shows the amount of dropped frames when 6 high-quality video traffic flows, 24 low-quality video flows and 10 CBR flows are generated with or without rerouting. The figure shows only the drop rate for high-quality flows since our approach aims to improve the performance of high-quality video streaming.

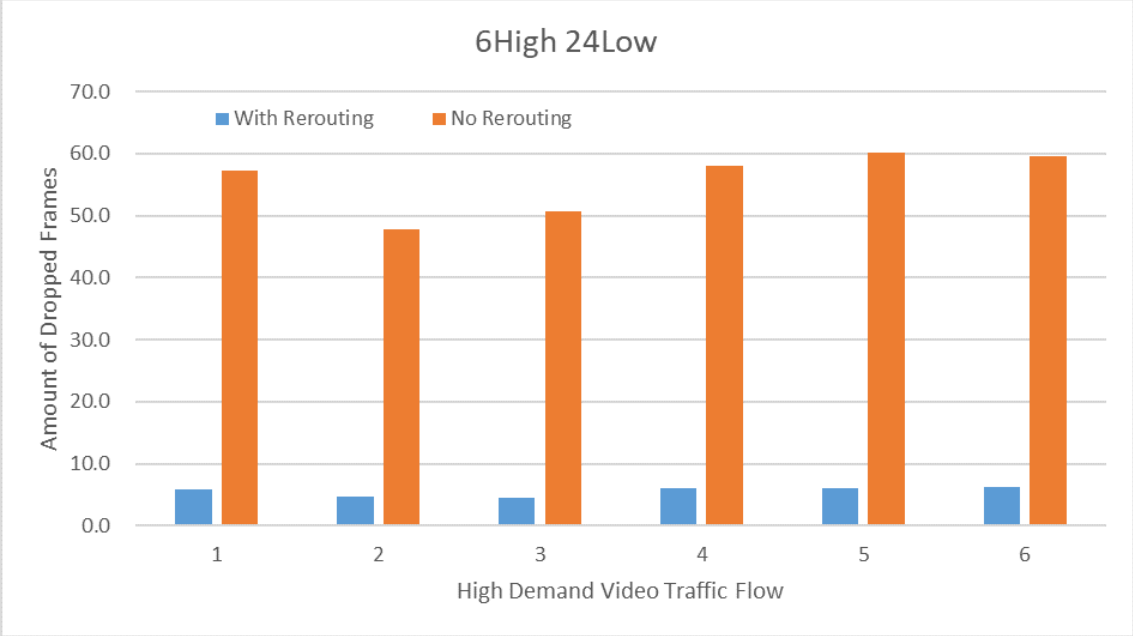


FIGURE 5.8: Dropped Frames when HQ=6;LQ=24;CBR=10

As seen from Figure 5.8, with our dynamic routing approach, for each received video file only 10 frames are dropped. However, without rerouting, up to 60 frames are lost for each flow. This shows the success of the proposed approach in terms of dropped packets.

5.5.2 8 High-Quality Video Flows and 22 Low-Quality Video Flows

Figure 5.9 shows the amount of dropped frames when 8 high-quality video traffic flows, 22 low-quality video flows and 10 CBR flows are generated with or without rerouting.

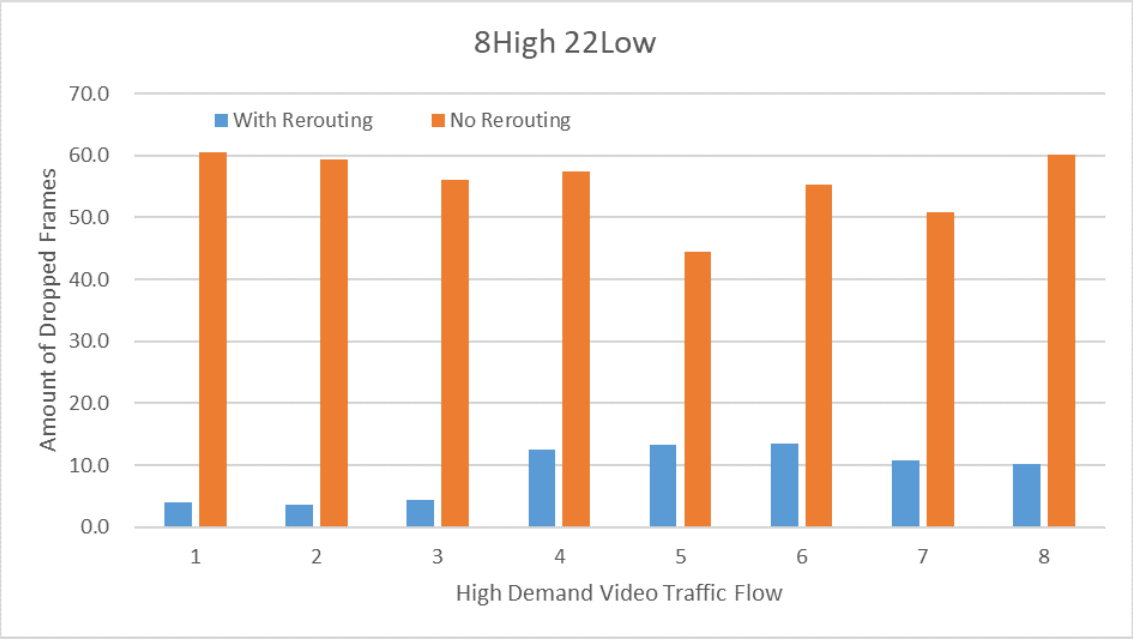


FIGURE 5.9: Dropped Frames when HQ=8;LQ=22;CBR=10

As observed from Figure 5.9, the number of dropped frames is significantly less when our proposed approach is used. With our approach the number of dropped frames are less than 15 for all flows while the number of dropped frames when our approach is not used, reaches 60 frames. Note that, the first three flows have less dropped frames than the rest of the flows. The reason is that our rerouting algorithm reroutes these three flows first.

5.5.3 10 High-Quality Video Flows and 20 Low-Quality Video Flows

Figure 5.10 shows the amount of dropped frames when 10 high-quality video traffic flows, 20 low-quality video flows and 10 CBR flows are generated with or without rerouting.

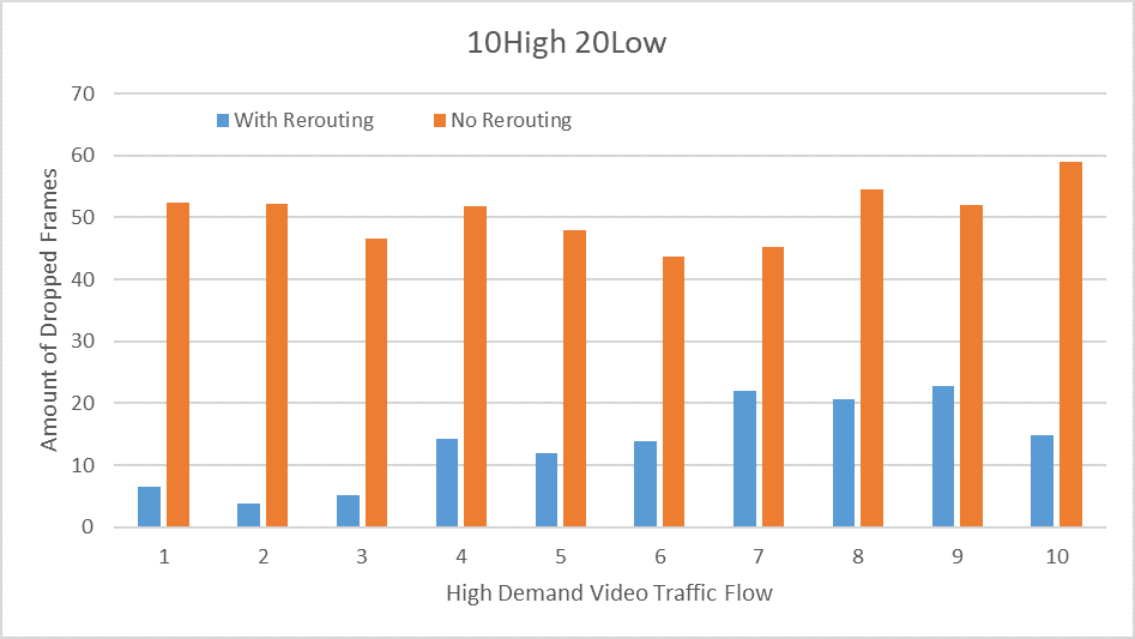


FIGURE 5.10: Dropped Frames when HQ=10;LQ=20;CBR=10

As observed from the figure, our proposed approach still has less number of dropped packets. An important observation is in our approach the number of dropped frames show a step function pattern. This is due to the fact that we reroute flows in a bundle of three starting from flow 1.

5.5.4 12 High-Quality Video Flows and 18 Low-Quality Video Flows

Figure 5.11 shows the amount of dropped frames when 12 high-quality video traffic flows, 18 low-quality video flows and 10 CBR flows are generated with or without rerouting.

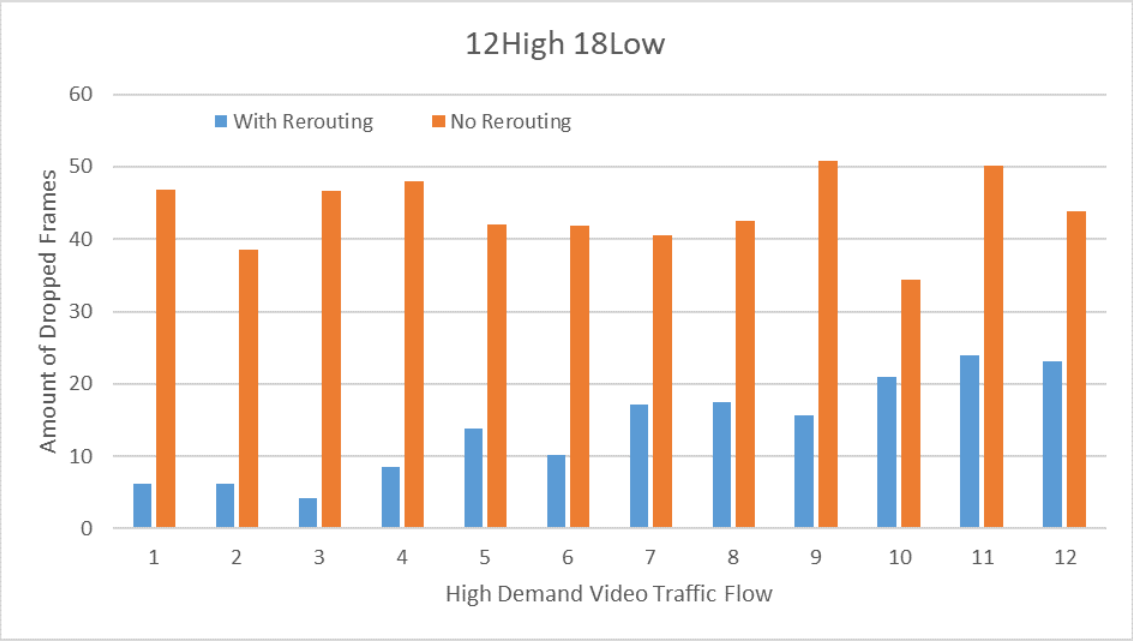


FIGURE 5.11: Dropped Frames when HQ=12;LQ=18;CBR=10

When there are 12 high-quality video flows, the controller needs four rounds to execute our dynamic routing approach, taking traffic measurements, clustering the traffic and then selecting the flows to reroute, three flows a time. The figure shows the four rounds as four clear steps. The last three flows have about 20 dropped frames each. In any case, the dropped frames of high-quality video stream is less when our proposed aproach is used.

5.5.5 14 High-Quality Video Flows and 16 Low-Quality Video Flows

Figure 5.12 shows the amount of dropped frames when 14 high-quality video traffic flows, 16 low-quality video flows and 10 CBR flows are generated with or without rerouting.

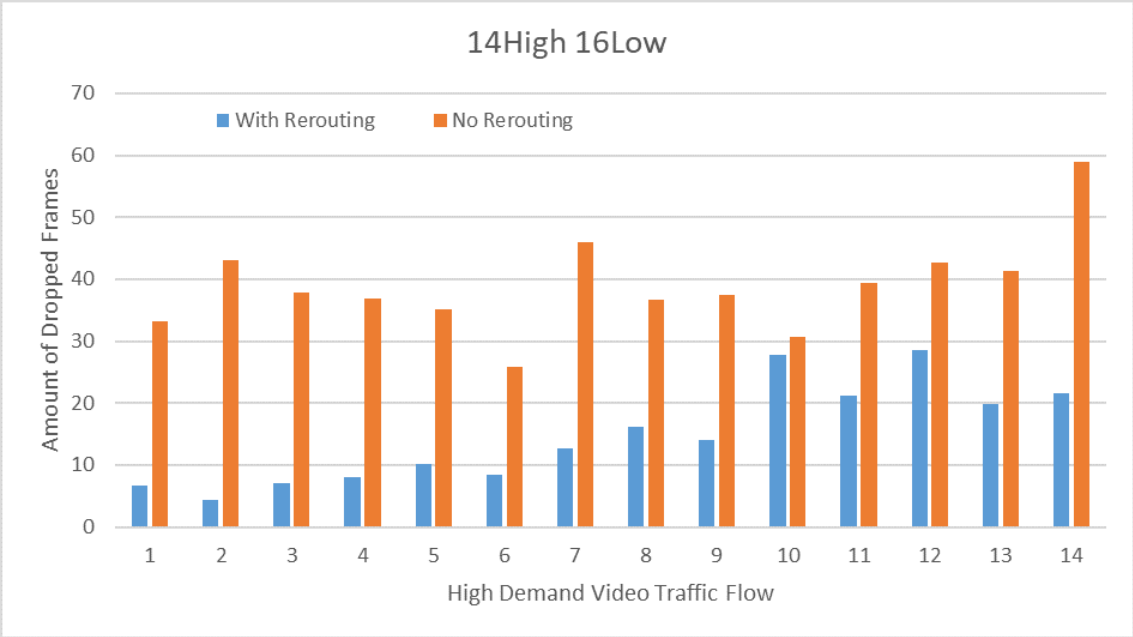


FIGURE 5.12: Dropped Frames when HQ=14;LQ=16;CBR=10

Figure 5.12 shows similar trends with the previous figures where the proposed approach offers less dropped frames. Flows between 10 and 14 receive higher drops than the other flows. Overall, flows has less dropped frames if it is rerouted earlier. However, when the disjoint paths are congested dropped frames are observed on alternative paths. To illustrate more details of the delay improvements brought by rerouting, average delay of dropped frames will be shown in the next section.

5.6 Average Delay of Dropped Frames

This section shows average delay of dropped frames in five different scenarios. Average delay of dropped frames indicates the overall delay performance which is important for replaying videos on the client side without freezing.

5.6.1 6 High-Quality Video Flows and 24 Low-Quality Video Flows

Figure 5.13 shows the average delay of dropped frames when 6 high-quality video traffic flows, 24 low-quality video flows and 10 CBR flows are generated with and without rerouting.

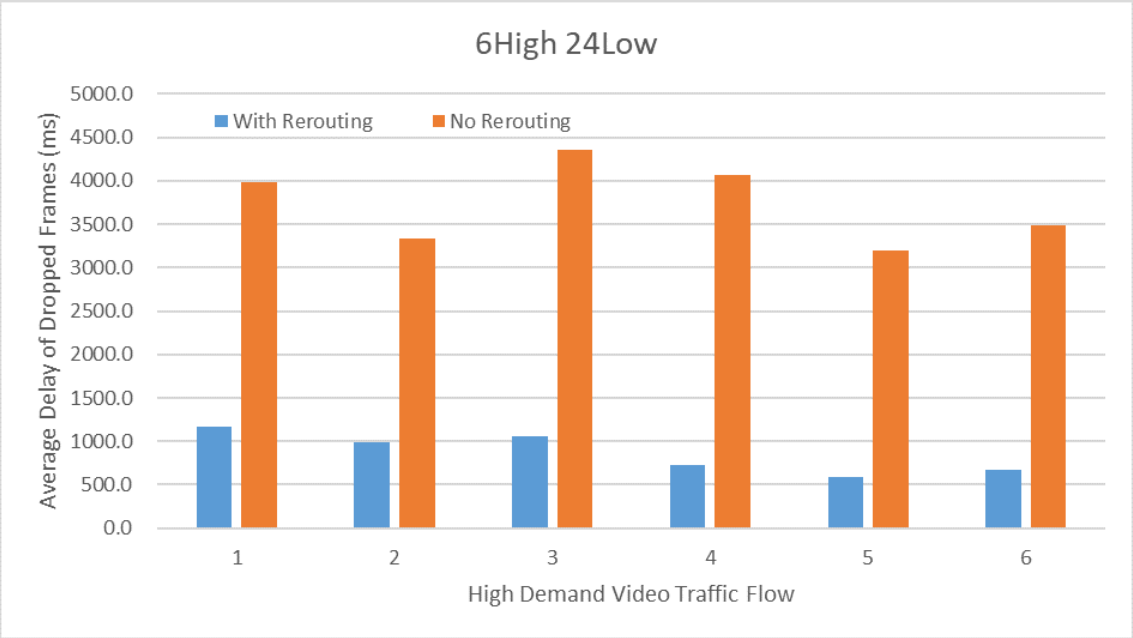


FIGURE 5.13: Average Delay of Dropped Frames when HQ=6;LQ=24;CBR=10

As seen from the figure, using our proposed scheme the average delay of dropped packets is close to 1000ms. On the other hand, without rerouting flows, the average delay of drooped packets is close to 3500ms which is over three times than that of our scheme.

5.6.2 8 High-Quality Video Flows and 22 Low-Quality Video Flows

Figure 5.14 shows the average delay of dropped frames when 8 high-quality video traffic flows, 22 low-quality video flows and 10 CBR flows are generated with and without rerouting.

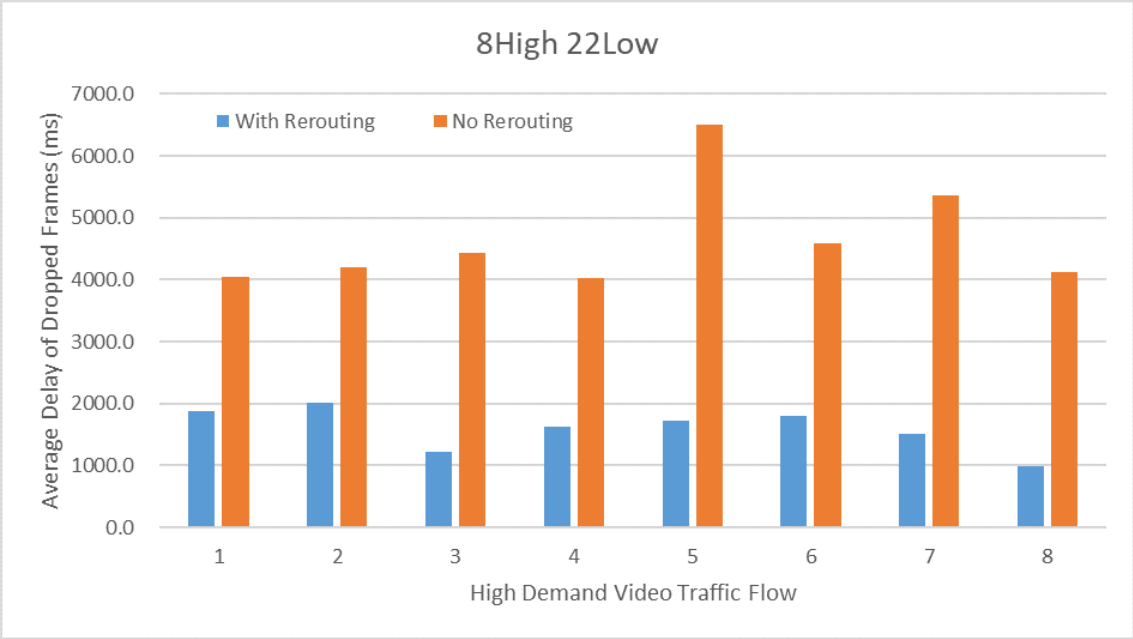


FIGURE 5.14: Average Delay of Dropped Frames when HQ=8;LQ=22;CBR=10

As seen from the figure, as more high-quality traffic flows are generated, the average delay increases for both routing approaches. However, our proposed approach attains less delay.

5.6.3 10 High-Quality Video Flows and 20 Low-Quality Video Flows

Figure 5.15 shows the average delay of dropped frames when 10 high-quality video traffic flows, 20 low-quality video flows and 10 CBR flows are generated with and without rerouting.

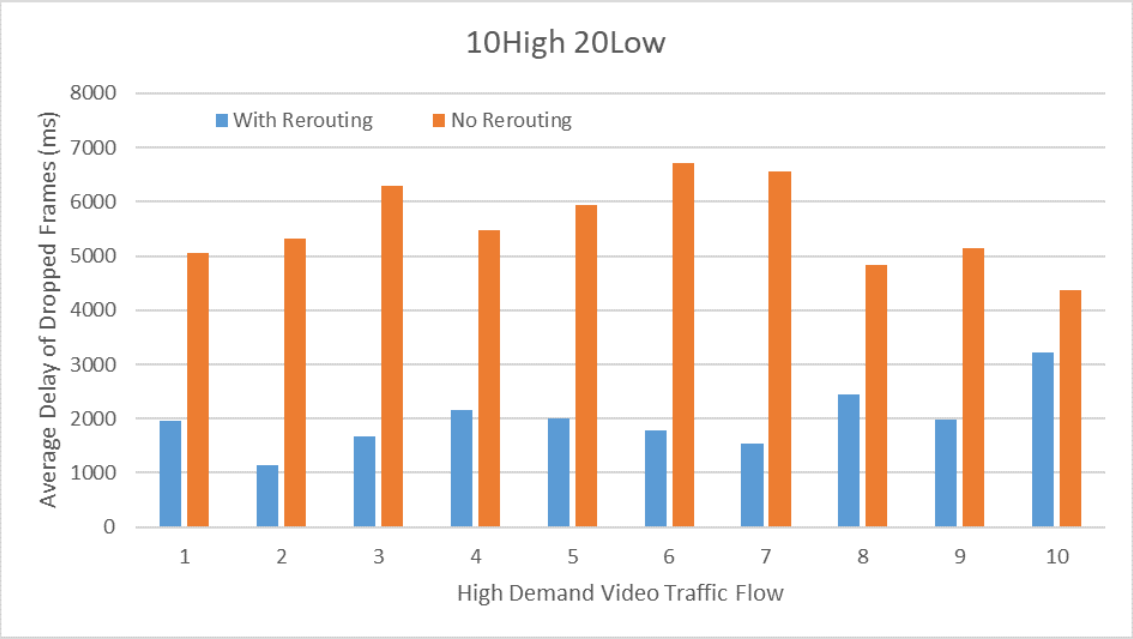


FIGURE 5.15: Average Delay of Dropped Frames when HQ=10;LQ=20;CBR=10

As seen from the figure, our proposed scheme can attain the delay values close to 2000ms while the delay can exceed 6000ms for the case when no rerouting is applied.

5.6.4 12 High-Quality Video Flows and 18 Low-Quality Video Flows

Figure 5.16 shows the average delay of dropped frames when 12 high-quality video traffic flows, 18 low-quality video flows and 10 CBR flows are generated with and without rerouting.

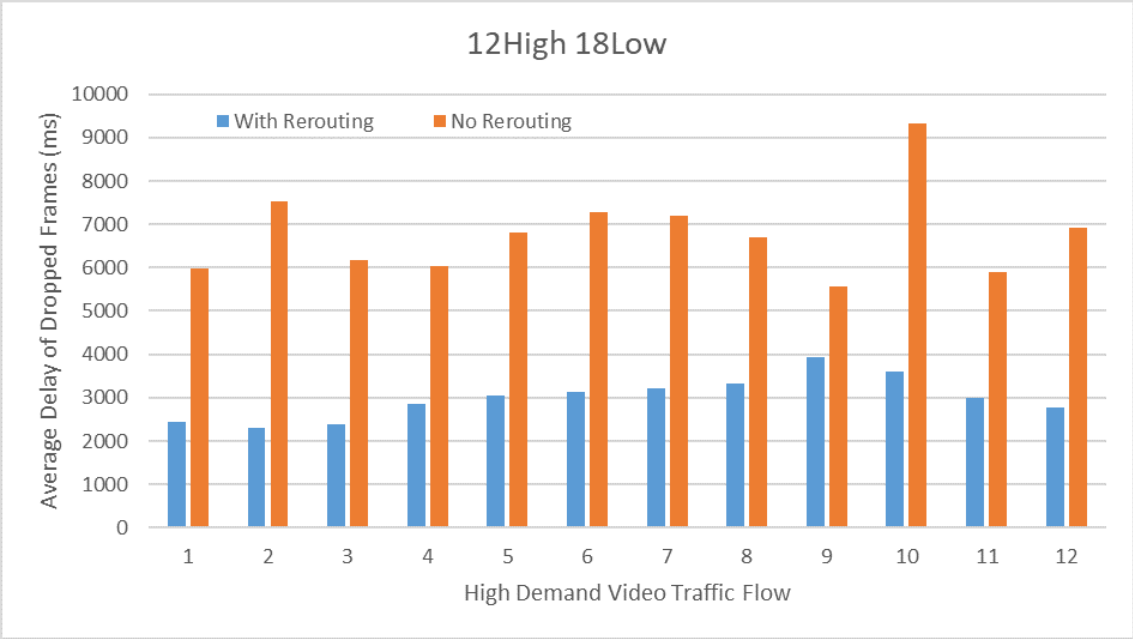


FIGURE 5.16: Average Delay of Dropped Frames when HQ=12;LQ=18;CBR=10

The results observed in Figure 5.16 follow a similar trend with the previous figures. Our approach incurs less delay for dropped packets.

5.6.5 14 High-Quality Video Flows and 16 Low-Quality Video Flows

Figure 5.17 shows the average delay of dropped frames when 14 high-quality video traffic flows, 16 low-quality video flows and 10 CBR flows are generated with and without rerouting.

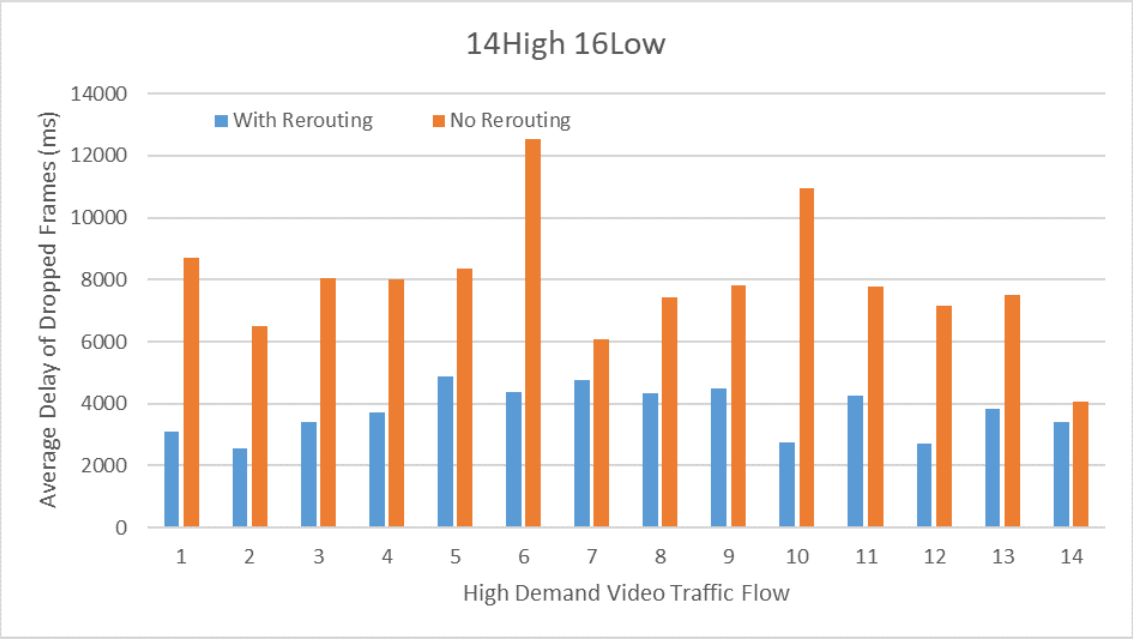


FIGURE 5.17: Average Delay of Dropped Frames when HQ=14;LQ=16;CBR=10

As more number of flows are added and the paths become congested the average delay increases as expected. However, our proposed approach has less delay than the case with no rerouting.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

High-quality video traffic contributes to a large portion of today's Internet traffic. The conventional routing algorithms are not suitable for fast-changing network conditions. SDN enables network managers to implement network functions based on needs by separating the control plane from the data plane. However, there is still a need to identify high-quality video traffic flows and prioritize those flows by offering less congested paths. Machine learning techniques can be applied in the powerful SDN controller for this purpose. The decoupled architecture of SDN, switch-to-controller messaging system, OpenFlow and mininet enable researchers to collect traffic data and manage the network intelligently. Recent work on traffic classification using supervised/unsupervised learning at flow level or packet level is presented in this thesis. Timeliness of traffic estimation is an important issue but it has not been well studied. Since traffic flows vary over time, there is a need for online traffic estimation coupled with dynamic routing approaches.

In this thesis, we addressed the online traffic estimation problem for a limited traffic type and its application in the dynamic routing of video streaming over SDN. In Chapter 3, we presented the online traffic estimation technique. We first applied PCA

to select principal traffic features. Then DBSCAN is used to cluster incoming traffic within a small time window. The time window slides throughout the estimation period amount of data to achieve online clustering. In Chapter 4, we presented our dynamic weighted shortest path routing algorithm which uses the clustering information from our online traffic estimator as well as network related information from SDN monitor, and reroutes traffic to uncongested disjoint paths.

In Chapter 5, we presented our simulations results for five scenarios containing different number of video streaming flows. We showed that our proposed dynamic routing approach enhances network throughput and reduces the number of dropped packets.

6.2 Future Work

The major focus of this work is to distinguish high-quality video traffic from the other flows. More tests need to be conducted with varying simulation parameters. Centralized control in SDN could lead to concerns about scalability so a larger and more complicated topology should be tested in the future. Unsupervised learning with appropriate features can cluster the high-quality video traffic but it only treats the other traffic as background traffic. Therefore, the accuracy of traffic estimation needs to be improved if further classification is needed. Deep learning can be used to enhance the quality of traffic estimation.

Additionally, our dynamic routing algorithm only uses link load when assigning weights. For future studies, delay, energy and other path properties can be added. Furthermore, our algorithm simply selects the three flows, each time, based on their start time. Flow statistics can also be monitored and flows who are suffering from poor performance can be given priority. In addition, reinforcement learning, with the

properly addressed environment, states, actions and reward, could be an option to optimize network resource utilization while meeting good QoS.

Bibliography

[1] Cisco, *Cisco visual networking index: Forecast and trends, 2017–2022*, 2019. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.pdf, (accessed: 18.December.2019).

[2] M. U. Rehman and G. A. Safdar, Eds., *Lte communications and networks: Femtocells and antenna design challenges*, Apr. 2018. DOI: 10.1002/9781119385271. [Online]. Available: http://eprints.gla.ac.uk/200805/.

[3] R. Trestian, K. Katrinis, and G. Muntean, "Ofload: An openflow-based dynamic load balancing strategy for datacenter networks", *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 792–803, Dec. 2017, ISSN: 2373-7379. DOI: 10.1109/TNSM.2017.2758402.

[4] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis", in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, ACM, 2009, pp. 202–208.

[5] S. Shen, "Efficient svc multicast streaming for video conferencing with sdn control", *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 403–416, Jun. 2019, ISSN: 2373-7379. DOI: 10.1109/TNSM.2019.2911086.

[6] S. Ortiz, "Software-defined networking: On the verge of a breakthrough?", *Computer*, no. 7, pp. 10–12, 2013.

[7] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey", *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015, ISSN: 1558-2256. DOI: 10.1109/JPROC.2014.2371999.

[8] A. Binsahaq, T. R. Sheltami, and K. Salah, "A survey on autonomic provisioning and management of qos in sdn networks", *IEEE Access*, vol. 7, pp. 73 384–73 435, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2919957.

[9] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira, "Network slicing for 5g with sdn/nfv: Concepts, architectures, and challenges", *IEEE Communications Magazine*, vol. 55, no. 5, pp. 80–87, 2017.

[10] M. Latah and L. Toker, "Artificial intelligence enabled software-defined networking: A comprehensive overview", *IET Networks*, vol. 8, no. 2, pp. 79–99, 2018.

[11] M. Elsayed and M. Erol-Kantarci, "Ai-enabled future wireless networks: Challenges, opportunities, and open issues", *IEEE Vehicular Technology Magazine*, vol. 14, no. 3, pp. 70–77, Sep. 2019, ISSN: 1556-6080. DOI: 10.1109/MVT.2019.2919236.

[12] Y. Zhao, Y. Li, X. Zhang, G. Geng, W. Zhang, and Y. Sun, "A survey of networking applications applying the software defined networking concept based on machine learning", *IEEE Access*, vol. 7, pp. 95 397–95 417, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2928564.

[13] S. Tomovic and I. Radusinovic, "Toward a scalable, robust, and qos-aware virtual-link provisioning in sdn-based isp networks", *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 1032–1045, Sep. 2019, ISSN: 2373-7379. DOI: 10.1109/TNSM.2019.2929161.

[14] N. Jawad, M. Salih, K. Ali, B. Meunier, Y. Zhang, X. Zhang, R. Zetik, C. Zarakovitis, H. Koumaras, M. Kourtis, L. Shi, W. Mazurczyk, and J. Cosmas, "Smart television services using nfv/sdn network management", *IEEE Transactions on Broadcasting*, vol. 65, no. 2, pp. 404–413, Jun. 2019, ISSN: 1557-9611. DOI: 10.1109/TBC.2019.2898159.

[15] E. Keller and J. Rexford, "The" platform as a service" model for networking.", *INM/WREN*, vol. 10, pp. 95–108, 2010.

[16] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu, "A survey of machine learning techniques applied to software defined networking (sdn): Research issues and challenges", *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 393–430, Jan. 2019, ISSN: 2373-745X. DOI: 10.1109/COMST.2018.2866942.

[17] O. N. Foundation, *Openflow-spec-v1.3.1*, 2012. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.3.1.pdf, (accessed: 17.December.2019).

[18] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: Towards an operating system for networks", *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[19] M. et al., *Pox manual*, 2015. [Online]. Available: https://noxrepo.github.io/pox-doc/html/, (accessed: 16.December.2019).

[20] *Floodlight openflow controller*, May 2018. [Online]. Available: http://www.projectfloodlight.org/floodlight/, (accessed: 16.December.2019).

[21] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using openflow: A survey", *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 493–512, Jan. 2014, ISSN: 2373-745X. DOI: 10.1109/SURV.2013.081313.00105.

[22] C. Hsieh, N. Weng, and W. Wei, "Scalable many-field packet classification for traffic steering in sdn switches", *IEEE Transactions on Network and Service Management*, vol. 16, no. 1, pp. 348–361, Mar. 2019, ISSN: 2373-7379. DOI: 10.1109/TNSM.2018.2869403.

[23] E. Kim, Y. Choi, S. Lee, and H. J. Kim, "Enhanced flow table management scheme with an lru-based caching algorithm for sdn", *IEEE Access*, vol. 5, pp. 25 555–25 564, 2017, ISSN: 2169-3536.

[24] E. Akin and T. Korkmaz, "Comparison of routing algorithms with static and dynamic link cost in software defined networking (sdn)", *IEEE Access*, vol. 7, pp. 148 629–148 644, 2019, ISSN: 2169-3536.

[25] M. Mu, M. Broadbent, A. Farshad, N. Hart, D. Hutchison, Q. Ni, and N. Race, "A scalable user fairness model for adaptive video streaming over sdn-assisted future networks", *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 8, pp. 2168–2184, Aug. 2016, ISSN: 1558-0008. DOI: 10.1109/JSAC.2016.2577318.

[26] G. S. Park and H. Song, "Video quality-aware traffic offloading system for video streaming services over 5g networks with dual connectivity", *IEEE Transactions on Vehicular Technology*, vol. 68, no. 6, pp. 5928–5943, Jun. 2019, ISSN: 1939-9359. DOI: 10.1109/TVT.2019.2909547.

[27] K. T. Bagci and A. M. Tekalp, "Dynamic resource allocation by batch optimization for value-added video services over sdn", *IEEE Transactions on Multimedia*, vol. 20, no. 11, pp. 3084–3096, Nov. 2018, ISSN: 1941-0077. DOI: 10.1109/TMM.2018.2823907.

[28] J. Yang, K. Zhu, Y. Ran, W. Cai, and E. Yang, "Joint admission control and routing via approximate dynamic programming for streaming video over software-defined networking", *IEEE Transactions on Multimedia*, vol. 19, no. 3, pp. 619–631, Mar. 2017, ISSN: 1941-0077. DOI: 10.1109/TMM.2016.2629280.

[29] B. Lantz and B. O'Connor, "A mininet-based virtual testbed for distributed sdn development", in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 45, 2015, pp. 365–366.

[30] L. L. Zulu, K. A. Ogudo, and P. O. Umenne, "Simulating software defined networking using mininet to optimize host communication in a realistic programmable network", in *2018 International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD)*, Aug. 2018, pp. 1–6. DOI: 10.1109/ICABCD.2018.8465433.

[31] M. Gupta, J. Sommers, and P. Barford, "Fast, accurate simulation for sdn prototyping", in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13, New York, NY, USA: Association for Computing Machinery, 2013, 31–36, ISBN: 9781450321785. DOI: 10.1145/2491185.2491202. [Online]. Available: https://doi.org/10.1145/2491185.2491202.

[32] H. Kim, J. Kim, and Y. Ko, "Developing a cost-effective openflow testbed for small-scale software defined networking", in *16th International Conference on Advanced Communication Technology*, Feb. 2014, pp. 758–761. DOI: 10.1109/ICACT.2014.6779064.

[33] R. Barrett, A. Facey, W. Nxumalo, J. Rogers, P. Vatcher, and M. St-Hilaire, "Dynamic traffic diversion in sdn: Testbed vs mininet", in *2017 International Conference on Computing, Networking and Communications (ICNC)*, Jan. 2017, pp. 167–171. DOI: 10.1109/ICCNC.2017.7876121.

[34] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and openflow: From concept to implementation", *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 2181–2206, Oct. 2014, ISSN: 2373-745X. DOI: 10.1109/COMST.2014.2326417.

[35] C. Cortes and V. Vapnik, "Support-vector networks", *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[36] J. Yan and J. Yuan, "A survey of traffic classification in software defined networks", in *2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN)*, IEEE, 2018, pp. 200–206.

[37] S. Lloyd, "Least squares quantization in pcm", *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[38] H.-P. K. J. S. Ester M. and X. Xiaowei, "A density-based algorithm for discovering clusters in large spatial databases with noise", *Proceedings of the Second International Conference on Knowledge Discovery in Databases and Data Mining*, pp. 226–231, 1996.

[39] Y. Zhai and X. Zheng, "Random forest based traffic classification method in sdn", in *2018 International Conference on Cloud Computing, Big Data and Blockchain (ICCBB)*, Nov. 2018, pp. 1–5. DOI: 10.1109/ICCBB.2018.8756496.

[40] S. A. Baset and H. Schulzrinne, "An analysis of the skype peer-to-peer internet telephony protocol", *arXiv preprint cs/0412017*, 2004.

[41] M. Reza, M. J. Sobouti, S. Raouf, and R. Javidan, "Network traffic classification using machine learning techniques over software defined networks", *International Journal of Advanced Computer Science and Applications*, vol. 8, Jan. 2017. DOI: 10.14569/IJACSA.2017.080729.

[42] M. Hayes, B. Ng, A. Pekar, and W. K. Seah, "Scalable architecture for sdn traffic classification", *IEEE Systems Journal*, no. 99, pp. 1–12, 2017.

[43] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson, "Identifying and discriminating between web and peer-to-peer traffic in the network core", in *Proceedings of the 16th International Conference on World Wide Web*, ser. WWW '07, ACM, 2007, pp. 883–892, ISBN: 978-1-59593-654-7. DOI: 10.1145/1242572.1242692. [Online]. Available: http://doi.acm.org/10.1145/1242572.1242692.

[44] F. Tang, L. Li, L. Barolli, and C. Tang, "An efficient sampling and classification approach for flow detection in sdn-based big data centers", in *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, IEEE, 2017, pp. 1106–1115.

[45] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[46] A. S. Da Silva, C. C. Machado, R. V. Bisol, L. Z. Granville, and A. Schaeffer-Filho, "Identification and selection of flow features for accurate traffic classification in sdn", in *2015 IEEE 14th International Symposium on Network Computing and Applications*, IEEE, 2015, pp. 134–141.

[47] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space", *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[48] H. Hotelling, "Analysis of a complex of statistical variables into principal components.", *Journal of educational psychology*, vol. 24, no. 6, p. 417, 1933.

[49] Z. A. Qazi, J. Lee, T. Jin, G. Bellala, M. Arndt, and G. Noubir, "Application-awareness in sdn", in *ACM SIGCOMM computer communication review*, ACM, vol. 43, 2013, pp. 487–488.

[50] Y. Li and J. Li, "Multiclassifier: A combination of dpi and ml for application-layer classification in sdn", in *The 2014 2nd International Conference on Systems and Informatics (ICSAI 2014)*, Nov. 2014, pp. 682–686. DOI: 10.1109/ICSAI.2014.7009372.

[51] B. Ng, M. Hayes, and W. K. G. Seah, "Developing a traffic classification platform for enterprise networks with sdn: Experiences lessons learned", in *2015 IFIP Networking Conference (IFIP Networking)*, May 2015, pp. 1–9.

[52] H. A. H. Ibrahim, O. R. Aqeel Al Zuobi, M. A. Al-Namari, G. MohamedAli, and A. A. A. Abdalla, "Internet traffic classification using machine learning approach: Datasets validation issues", in *2016 Conference of Basic Sciences and Engineering Studies (SGCAC)*, Feb. 2016, pp. 158–166. DOI: 10.1109/SGCAC.2016.7458022.

[53] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly", *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 2, pp. 23–26, Apr. 2006, ISSN: 0146-4833. DOI: 10.1145/1129582.1129589. [Online]. Available: http://doi.acm.org/10.1145/1129582.1129589.

[54] J. Erman, M. Arlitt, and A. Mahanti, "Traffic classification using clustering algorithms", in *Proceedings of the 2006 SIGCOMM workshop on Mining network data*, ACM, 2006, pp. 281–286.

[55] T. T. T. Nguyen, G. Armitage, P. Branch, and S. Zander, "Timely and continuous machine-learning-based classification for interactive ip traffic", *IEEE/ACM Transactions on Networking*, vol. 20, no. 6, pp. 1880–1894, Dec. 2012, ISSN: 1558-2566. DOI: 10.1109/TNET.2012.2187305.

[56] M. Alsaeedi, M. M. Mohamad, and A. A. Al-Roubaiey, "Toward adaptive and scalable openflow-sdn flow control: A survey", *IEEE Access*, vol. 7, pp. 107 346–107 379, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2932422.

[57] T. Nguyen and G. Armitage, "Training on multiple sub-flows to optimise the use of machine learning classifiers in real-world ip networks", *Proc. IEEE 31st Conference on Local Computer Networks*, Nov. 2006.

[58] O. S. P. Haffner S. Sen and D. Wang, "Acas: Automated construction of application signatures", *MineNet '05: Proceeding of the 2005 ACM SIGCOMM workshop on Mining network data*, pp. 197–202, Aug. 2005.

[59] P. Amaral, J. Dinis, P. Pinto, L. Bernardo, J. Tavares, and H. Mamede, "Machine learning in software defined networks: Data collection and traffic classification", Nov. 2016, pp. 1–5. DOI: 10.1109/ICNP.2016.7785327.

[60] Y. Cui, T. Li, C. Liu, X. Wang, and M. Kühlewind, "Innovating transport with quic: Design approaches and research challenges", *IEEE Internet Computing*, vol. 21, no. 2, pp. 72–76, 2017.

[61] L. Breiman, "Random forests", *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[62] M. Halkidi, Y. Batistakis, and M. Vazirgiannis, "Cluster validity methods: Part i", *ACM Sigmod Record*, vol. 31, no. 2, pp. 40–45, 2002.

[63] S. Zhu, Z. Sun, Y. Lu, L. Zhang, Y. Wei, and G. Min, "Centralized qos routing using network calculus for sdn-based streaming media networks", *IEEE Access*, vol. 7, pp. 146 566–146 576, 2019, ISSN: 2169-3536.

[64] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance", in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 377–392.