Designing an Openflow Controller for data delivery with end-to-end QoS over Software Defined Networks

Master of Technology

in

Computer Science and Engineering

by

Bandi Sumanth (11CS30006)

advised by

Dr. Sandip Chakraborty

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

May 2016

Declaration

This is to certify that

1. The work contained in this thesis is original and has been done by myself under the general supervision of my supervisor.
2. The work has not been submitted to any other Institute for any degree or diploma.
3. I have followed the guidelines provided by the Institute in writing the thesis.
4. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.
5. Whenever I have used materials (data, theoretical analysis, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references.
6. Whenever I have quoted written materials from other sources, I have put them under quotation marks and given due credit to the sources by citing them in the text of the thesis and giving their details in the references.

Bandi Sumanth

Department of CSE

IIT Kharagpur

Date :

Certificate

This is to certify that the thesis titled **Designing an Openflow Controller for data delivery with end-to-end QoS over Software Defined Networks** submitted by **Bandi Sumanth (11CS3006)** to the Department of Computer Science and Engineering is a bonafide record of work carried out by him under my supervision and guidance. The thesis has fulfilled all the requirements as per the regulations of the Institute and, in my opinion, has reached the standard needed for submission.

Dr. Sandip Chakraborty
Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
May 2016

Abstract

Software Defined Networking (SDN) is a new network paradigm that provides us with a virtualized network infrastructure by segregating the control and data planes of the network forwarders, which results in an automated, flexible, dynamic network architecture. This creates an architecture which is very manageable and easy to operate or control the network. SDN uses the Openflow standard to control the data flow between the forwarders in a network. This flexible network control allows the service providers to define flows in various ways as per the flow requirements and end user requirements. Specific important or special flows can be set with minimum resource requirements based on the path latency, path bandwidth or other parameters. All these can be included in the flow definition, and as such we can have various types of flows. I designed an Openflow Controller to provide end-to-end QoS which programmatically takes the QoS bandwidth requirements from the users in a Software Defined Network architecture. The controller is simulated in an SDN environment via Mininet. In this paper, I detail i) the design and implementation of the controller system ii) how the network QoS is achieved using a Utility Proportional Fairness algorithm for bandwidth allocation iii) The implementation of QoS using the above algorithm in a virtual openflow network using Mininet. iv) The bandwidth sharing algorithm is evaluated with respect to the general scenario where there is no QoS policy. I used the Structural Similarity Index (SSIM) score , throughput and packet delay metrics for the above evaluation and presented the plots for each metric in the results chapter.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Software Defined Networking (SDN) is a paradigm shift in network architecture where the forwarding plane is decoupled from the control plane. This provides an abstraction of the network as a logical and a programmable entity to the services on the upper layers of the network. SDN is a paradigm shift over traditional TCP/IP architecture. To provide support for innovation, programmability and rapid deployment of services SDN uses separation of “data plane" and “control plane". Data plane devices execute control plane prescribed forwarding actions. Tasks of control plane includes gathering of information of data plane, calculation of action set and propagation of action set to the data plane. Unlike traditional TCP/IP architecture, the action set is defined on “flow"; where a flow is defined as a set of packets with matching headers. SDN data plane consists of switches and, control plane devices are generally termed as “controllers".

1.2 Motivation

Performance of the network depends upon quality of services (QoS) provided. Present TCP/IP architecture based QoS policy is insufficient to provide service guarantee as it is hard to implement an “intserv" architecture. On the other hand, “diffserv" architecture is unable to find the end to end flow semantics. Hence, it is hard to optimize a QoS global objective function in absence of global view of the flow.

Control plane can be implemented using single centralized controller. However, to design a scalable and fault tolerant system, controller functionality needs to be deployed in a distributed manner. The challenge in case of distributed controller lies in inter-controller communication overhead. Moreover, in case of large scale distributed system the controllers might belong to different service providers. Distributed controllers requires to maintain consistency of gathered information (e.g. Topology, Flow graph etc.) from the data plane devices. Although, local controllers can gather the information these information needs to be distributed among the controllers. Therefore, the need for inter-controller communication methodology needs to be upgraded, as the current standard implementation (e.g. OpenFlow, OpenDaylight etc.) lacks efficiency.

1.3 Objectives

The basic objective of this research is to enhance the QoS policy for large scale distributed system. To implement framework, software defined network (SDN) is used as the candidate architecture. Based on this framework we would like to design an optimization framework which can provide end to end QoS guarantee in case of multiple QoS flows in a network. We would like to implement this algorithm in a virtual openflow network using mininet and observe its performance.

Chapter 2

Literature Survey

Egilmez et al. [1] proposed an OpenFlow based controller, namely OpenQoS to achieve end-to-end QoS for multimedia applications. They modeled the QoS routing problem as a Constrained Shortest Path (CSP) problem with each route having a dynamically calculated cost and delay measures. The flows are classified into data flows and multimedia flows. The multimedia flows follow the QoS guaranteed routing algorithm, and the shortest path routing algorithm is used for the other data flows Their paper focuses mainly on the multimedia applications whereas in our Openflow Controller we strive to achieve the end-to-end Quality of Service for any time of applications with a proper Utility function which is a measurement of its bandwidth requirements.

Kelly [2] proposed a model called “Optimal Flow Control" (OFC) for efficient congestion control in networks. It provides an efficient way to deal with the network congestion problem and also provides a simpler way to share bandwidth resources among various flows in the network. In [2], the network flow problem is modeled as an optimization problem which is solved using a link pricing policy. An extensive work has been done on these optimization methods to solve network flow control problems. Kelly [2] showed that his approach reaches a proportional fairness in allocation of bandwidth to various competing users at equilibrium.

In [3], the researchers try to find an “optimal" resource allocation to maximize the social welfare in a network, by trying to maximize the aggregate sum of utilities of all the

competing users in the network

In [4], the authors present a best-effort QoS control system in an SDN based cloud infrastructure. Their proposed system deploys five queues with varying minimum and maximum bandwidths at each virtual machine running the servers. The flows have a certain bandwidth requirements. Their system provides best effort Quality of Service, i.e. if the resources are available, it allocates the queues to the respective demanding flows but there is no discussion of any fair sharing of resources among competing Virtual machines for the bandwidth resources.

In [5], the authors talk about a best-effort QoS system, in computer networks, not related to SDN. Their proposed approach, talk about using a best set of paths instead of only a single shortest path as per the traditional shortest path routing algorithms. They identify a set of best paths to forward data, based on parameters like throughput, latency and jitter of a path. The forwarding devices forward data using all the paths in the set of best paths.

Chapter 3

OpenFlow Controller Design

A new distributed flow control algorithm is used to achieve end-to-end QoS where the user's Quality of Service utility function should be continuously non-decreasing over the bandwidth domain. This controller design is extended over the standard OpenFlow controller to be able to set and provide Quality of Service policies over the network.

The main interfaces of the controller design are:

- **Controller - Forwarder Interface**: The controller interacts with the forwarding devices using the OpenFlow protocol to exchange any information. The controller installs the flow tables for the data flows. It gets the network topology information from the forwarding devices and monitors the network.
- **Controller - Controller Interface**: With just a single controller in the architecture, it becomes increasingly difficult to deal with very large networks. Multiple controllers are needed with increase in the number of devices. This controller to controller communication is necessary to distribute the control responsibility over various controllers in the network. This interface has not been developed yet and is left as the future work.
- **Controller - Service Interface**: Using this interface provided by the controller to the service providers, they can define and set various flow definitions. It also

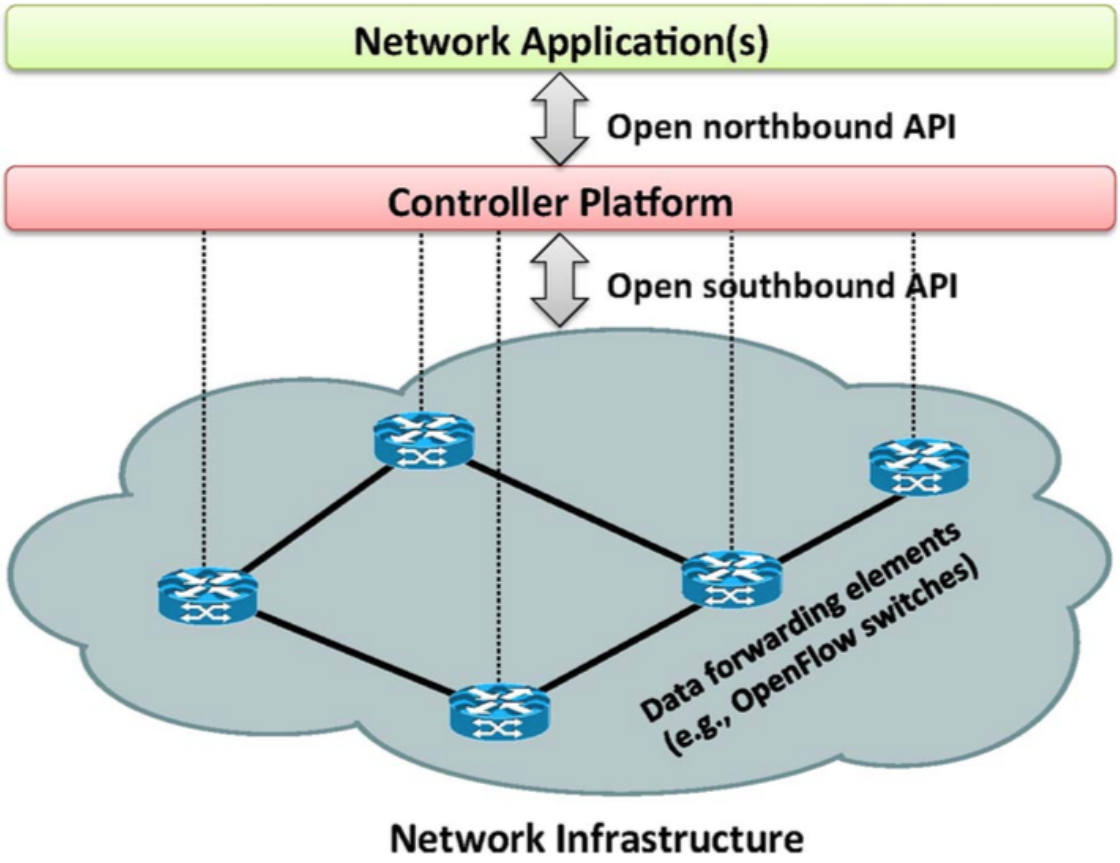


Figure 3.1: Simplified view of an SDN architecture
[6]

provides an interface to signal the controller at the start of a new data flow or when an existing flow is to be removed.

The functions performed by the controller are described below:

- Topology management: The responsibility of this module is to obtain the network graph by requesting and receiving data from the network data forwarding elements.
- Route calculation: This module deals with obtaining the path for each flow in the network. Customized routing algorithm are deployed for various flows based on the requirements set in their flow definitions.
- Flow admission and removal: This function is responsible for collecting the information regarding the current flows set using the service interface mentioned above

by the flow administrator and also also takes a decision if a new flow should be accepted or rejected. A new flow request is rejected if all the existing flows along with the new flow cannot be given a feasible bandwidth allocation as per the minimum and maximum requirements set by the service provider.

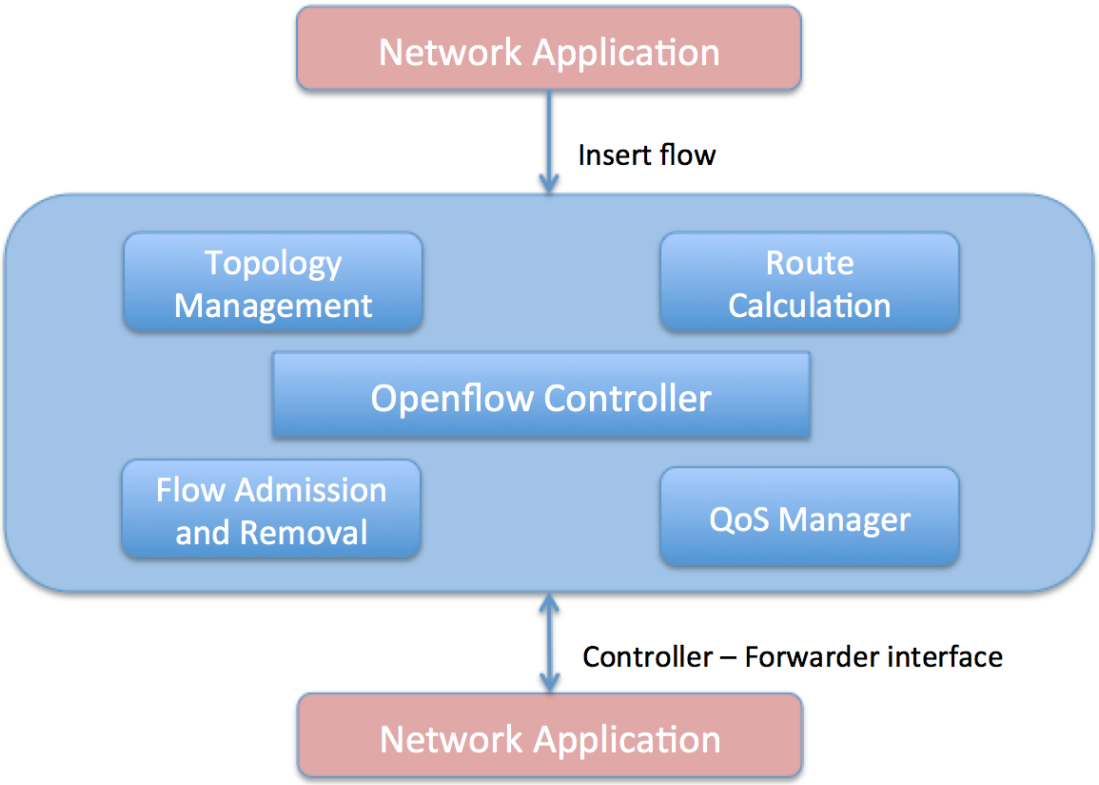


Figure 3.2: SDN controller design

3.1 Routing mechanism

In Computer Networks, routing is done using the defined flow rules in a routing table. The flow rules consist of source and destination IP addresses. When a packet arrives at a router, the router looks at the routing table, and forwards it as per the rules set by the routing protocol used by the network operator.

In Openflow networks, flow routing can be done in many ways, as we can define flows based on a various set of parameters. Openflow provided us with this flexibility of defining

various types of flows in the network. For example, packets with a specific destination port, irrespective of the destination IP address can be set as a flow definition.

Any routing algorithm performance depends on the accuracy of the overall network state information. In large networks, getting the global state of the network is difficult due to the large scale of the network. By having a centralized controller, this problem becomes easier, as the controller collects whatever information the forwarders can provide, and understands the global state of the network, thereby calculating the best possible routes for a data flow in the network.

Chapter 4

Quality of Service QoS algorithm

4.1 Algorithm for Utility based QoS routing

We used the bandwidth allocation algorithm presented in [1] in which bandwidth resources are allocated based on the flow utility requirements. The utility function of the application is non-decreasing over the available bandwidth domain, i.e. its happiness never decreases as the available bandwidth increases. The algorithm converges, and that at convergence, the achieved utility of all the flows are balanced and proportionally fair. Special preference is given to the bandwidth sensitive applications, i.e which demands bandwidth more than others.

In Section 4.2, the network flow control problem is presented. In Section 4.3, we present a resource allocation algorithm which is called “utility proportional fairness" and another alternative algorithm to obtain “utility max-min fairness".

4.2 Network Flow Control Problem

The problem is of resource allocation in computer networks. Consider a network that consists of a set $\mathcal{L} = \{1, 2, ..., L\}$ of unidirectional links with a capacity $c_l, l\epsilon\mathcal{L}$ï£ijï£ij

.The network is shared by a set $\mathcal{S} = \{1, 2, ..., S\}$ of sources. Each source s is characterized by five parameters $\{\mathcal{L}_s, x_s, m_s, M_s, U_s(x_s)\}$. The path $\mathcal{L}_s \subseteq \mathcal{L}$ is a subset of links that connect each source s to its destination, x_s is the transmission rate satisfying $0 \leq x_s \leq m_s \leq M_s \leq \infty$, where m_s and M_s are the minimum and maximum transmission rates required by source s, respectively. $U_s(x_s) : \mathcal{R}^+ \mapsto \mathcal{R}$ is a continuously increasing and bounded utility function which can be used as a QoS performance indicator for source s.

Let $x = [x_1, x_2, ..., x_s]^T$. For each link l, define $S_l = \{s \epsilon \mathcal{S} | l \epsilon \mathcal{L}_s\}$, which is the set of sources that access link l. Note that, $l \epsilon \mathcal{L}_s$ if and only if $s \epsilon \mathcal{S}_l$.
In order to formulate the network flow control problem, we first define the notion of feasible (or attainable) bandwidth allocation.
Definition 1: A bandwidth allocation $x = [x_1, x_2, ..., x_s]^T$ is feasible or attainable if and only if $x_s \epsilon [m_s, M_s]$, and no links in the network are congested, i.e.,

$$\sum_{s \in \mathcal{S}_l} x_s <= c_l \quad \forall l \ \epsilon \ \mathcal{L} \tag{4.1}$$

We assume that a minimum allocation $x = [x_1, x_2, ..., x_s]^T$ is attainable in the network.

When network bandwidth is abundantly available, there is no difficulty in satisfying every source utility, i.e., if $x = [M_1, M_2, ..., M_s]^T$. If the bandwidth resources are scarce, then we face a problem of fair bandwidth allocation across various utility requirements.

4.3 Optimal Flow Control Approach

In [2], [7], [3], the authors try to find an “optimal" resource allocation to maximize the aggregate sum of all utilities obtained, to provide maximum social happiness in a network. This objective can be achieved by solving the following primal optimization problem. I have followed the approach presented in [8] :

$$\mathbf{P} : max_{m_s<=x_s<=M_s} \qquad U(x) = \sum_{s=1}^{S} U_s(x_s) \tag{4.2}$$

$$subject\ to \ \sum_{s\epsilon\mathcal{S}_l} x_s \leq c_l, \quad l = 1, ..., L \tag{4.3}$$

Under a more critical assumption of strict concavity on utility functions, there always exists a unique optimal solution x to the maximization problem $\mathbf{P}$. The optimal solution x ï£ij can be obtained by looking for a $saddle - point$ in the following Lagrangian form.

$$L(x,p) = U(x) - \sum_{l=1}^{L} p_l \left(\sum_{s\epsilon\mathcal{S}_l} x_s - c_l \right) \tag{4.4}$$

$$= \sum_{s=1}^{S} \left[U_s(x_s) - x_s \sum_{l\epsilon\mathcal{L}_s} p_l \right] + \sum_{l=1}^{L} p_l c_l \tag{4.5}$$

$$= \sum_{s=1}^{S} \left[U_s(x_s) - x_s p^s \right] + \sum_{l=1}^{L} p_l c_l \tag{4.6}$$

where the Lagrangian multipliers $p = [p_1, p_2, ..., p_L]^T >= 0$ are the link prices and $p^s = \sum_{l\epsilon\mathcal{L}_s} p_l$ is the path price for source s, which is the sum of the link prices along path $\mathcal{L}_s$. An optimal flow control algorithm can be solved by solving the unconstrained min-max problem in the Lagrangian form $L(x,p)$:

Objective function of the dual problem:

$$D(p) = \max_{x_s\epsilon[m_s M_s]} L(x,p) = \sum_{s\epsilon\mathcal{S}} B_s(p^s) + \sum_{l\epsilon\mathcal{L}} p_l c_l \tag{4.7}$$

where

$$B_s(p^s) = \max_{x_s\epsilon[m_s M_s]} U_s(x_s) - x_s p^s \tag{4.8}$$

$$p^s = \sum_{l \epsilon \mathcal{L}_s} p_l \tag{4.9}$$

and the dual problem is:

$$\mathbf{D} : \min_{p>=0} D(p) \tag{4.10}$$

We use the iterative algorithm using the dual method proposed by Low and Lapsley

$$x_s(t+1) = arg \max_{m_s <= x_s <= M_s} L\left(x(t), p(t)\right) \tag{4.11}$$

$$= \left[{U_s}^{'-1} \left(p^s(t)\right) \right]_{m_s}^{M_s} \tag{4.12}$$

$$p_l(t+1) = \left[p_l(t) - \gamma \frac{\partial L}{\partial p_l} \left(x(t), p(t)\right) \right]^+ \tag{4.13}$$

$$= \left[p_l(t) + \gamma \left(x^l(t) - c_l \right) \right]^+ \tag{4.14}$$

where $\gamma > 0$ is a small step size, $[z]^+ = max\{0, z\}, [z]_a^b = max\{a, min\{b, z\}\}, x^l(t) = \sum_{s \epsilon \mathcal{S}_l} x_s(t)$ is the aggregate source rate at link l. ${U_s}^{'-1}$ is the inverse of ${U_s}^{'}$ and is decreasing over the range $\left[{U_s}^{'}(M_s), {U_s}^{'}(m_s) \right]$

4.4 Utility Fair Flow Control Algorithm

Definition: A bandwidth allocation $x^* = [x_1^*, x_2^*, ..., x_s^*]^T$ is utility proportionally fair, if it is feasible and for any other feasible allocation x,

$$\sum_{s \epsilon \mathcal{S}} \frac{(x_s - x_s^*)}{U_s(x_s^*)} \leq 0 \tag{4.15}$$

We use a distributed utility proportional fairness algorithm for allocation of resources. It

is closely related to the flow control algorithm, but also has significant differences.

This algorithm uses the same flow control structure as the optimal flow control approach does, in which a link algorithm is deployed at each link to update the link price depending on the severity of link congestion, and a source algorithm is implemented at each source edge to adapt the transmission rate based on the feedback path price.

The link algorithm is the same as that of 4.14. At time $t+1$, each link l updates its link price p_l according to

$$p_l(t+1) = \left[p_l(t) + \gamma\left(x^l(t) - c_l\right)\right]^+ \tag{4.16}$$

where $\gamma > 0$ is a small step size and $x^l(t) = \sum_{s \epsilon \mathcal{S}_l} x_s(t)$ is the aggregate source rate at link l. Equation 4.16 says that if the aggregate source rate at link l exceeds the link capacity c_l, the link price will be increased; otherwise it will be decreased. The projection $[z]^+ = max\{0, z\}$ ï£ij ensures that the link price is always non-negative.

Upon receiving the path price (total sum of all prices of links in the path), each source s updates the source rate as follows:

$$x_s(t+1) = {U_s}^{-1}\left(\left[\frac{1}{p_s(t)}\right]_{U_s(m_s)}^{U_s(M_s)}\right) \tag{4.17}$$

where

$$p^s(t) = \sum_{l \epsilon \mathcal{L}_s} p_l(t) \tag{4.18}$$

is the path price of source s, $[z]_a^b = max\{a, min\{b, z\}\}$, and ${U_s}^{-1}$ is the inverse of U_s over the range $[U_s(m_s), U_s(M_s)]$. According to the definition of utility function, it is clear that $x_s(p^s)$ given in 4.17 is decreasing over the path price p^s. When $p^s \geq \frac{1}{U_s(m_s)}$, source s is required to transmit at a minimum rate m_s. When $p^s \leq \frac{1}{U_s(M_s)}$, source s is required to transmit at a maximum rate m_s. In between, source s attains a utility factor of $\frac{1}{p^s}$.

The source algorithms 4.17 and 4.12 both manage the transmission rate as a decreasing function of the path price, but they are significantly different in their mathematical descriptions.

4.5 Optimization and Convergence

The flow control algorithm 4.16 and 4.17 can be viewed as a distributed dual algorithm that solves the following optimization problem. I have followed the approach presented in [8] :

$$max_{m_s<=x_s<=M_s} \qquad \mathcal{U}(x) = \sum_{s=1}^{S} \mathcal{U}_s(x_s) \tag{4.19}$$

$$subject\ to \quad \sum_{s \epsilon \mathcal{S}_l} x_s \leq c_l, \quad l = 1, ..., L \tag{4.20}$$

where

$$\mathcal{U}_s(x_s) = \int_{m_s}^{x_s} \frac{1}{U_s(y)} dy \tag{4.21}$$

is a redefined "utility" function for source s.

The original utility function $U_s(x_s)$ is non-negative, continuous and strictly increasing over the range $x_s \epsilon [m_s, M_s]$. Therefore, $\mathcal{U}_s(x_s)$ must be increasing and strictly concave. If the step size γ in 4.16 is selected to be appropriately small, the sequence ï£ij ï£ij ï£ij ï£ij ï£ij generated by the dual algorithm 4.16 and 4.17 will solve the maximization problem 4.19, 4.20.

4.6 Utility Proportional Fairness

When the flow control algorithm 4.16 and 4.17 converges to the equilibrium (x^*, p^*), the objective function 4.19 is maximized within the feasible solution. For all feasible allocation $x \neq x^*$, the optimality condition is

$$\sum_{s \epsilon \mathcal{S}} \frac{\partial \mathcal{U}_s(x_s^*)}{\partial x_s}(x_s - x_s^*) = \sum_{s \epsilon \mathcal{S}} \frac{(x_s - x_s^*)}{U_s(x_s^*)} < 0 \tag{4.22}$$

where the strict inequality follows from the strict concavity of $\mathcal{U}_s(x_s)$. According to Definition 4.4, it is clear that, at optimality, the resource allocation ï£ij ï£ij is utility proportionally fair.

Chapter 5

Implementation of the Openflow controller

I have used Mininet with the POX openflow controller [9] for the implementation of the controller. Python is used for programming the POX controller. Mininet provides us with a virtual network environment of hosts and switches, where we can get the environment and feel of a realistic network to develop and run experiments using the Openflow protocol. Openflow is a standard communication interface between the control layer and the forwarding layer of a Software Defined Networking architecture. The openflow switch and controller communicate using this protocol. The switches just contain the forwarding tables with rules defining where the incoming packets should be forwarded to. If there is no rule for the incoming packet, the switch simply sends the packet to the controller which then decides what to be done with the incoming packet.

5.0.1 Controller-to-service Interface

Controller-to-service Interface has the $flow - management$ function which takes input from the service provider, the details about the input flow, the parameters which can uniquely identify a flow such as the source and destination IP addresses along with port addresses if necessary to define a flow and also the utility function for the flow which tells

about the flow's bandwidth requirements. These input flows are stored by the controller.

5.0.2 Modules in the POX Controller

- *Topology Management*: The Network state management module collects the network state information such as hosts, switches, and the links between then and the link bandwidths. A convenient way of using specific library and data structures to store the network state information is provided in [10]

- *Route Calculation:* Based on input set of flows, their utility functions and their minimum and maximum bandwidth requirements, for each flow, the route calculation module identifies the path / route of the flow and uses the utility proportional fairness algorithm to determine the bandwidths to be allocated to each flow. The calculated bandwidths along with the QoS route of the flow are input to the QoS Manager which ensures the flows get their share of bandwidth. The route calculation module returns false, if there is no feasible bandwidth allocation as per the minimum bandwidth requirements of all the flows.

- *Flow Admission and Removal:* Whenever a new QoS flow is detected by the controller, the flow admission module decides whether the new flow should be admitted. The flow admission module inputs the current set of flows along with the new flow to the Route Calculation module to make this decision. As mentioned above, the Route Calculation module returns a false, if there is no feasible bandwidth allocation. So, if there is a need to change the call admission policy, the decision making used in the Route calculation module can be varied. The above same procedure is followed, if a flow is removed.

- *QoS Manager:* The QoS Manager takes the responsibility of creation or modification of queues at all the appropriate switches in the QoS paths of the input flows. Multiple queues at a switch port are given id numbers to identify them. Based on the input calculated routes, the flow entries will be installed into the switch flow tables by the QoS Manager. The flow entries contain the queue_id information, to send the matching packets into the respective queues at a switch port.

The state diagram of controller-switch is provided in the figure

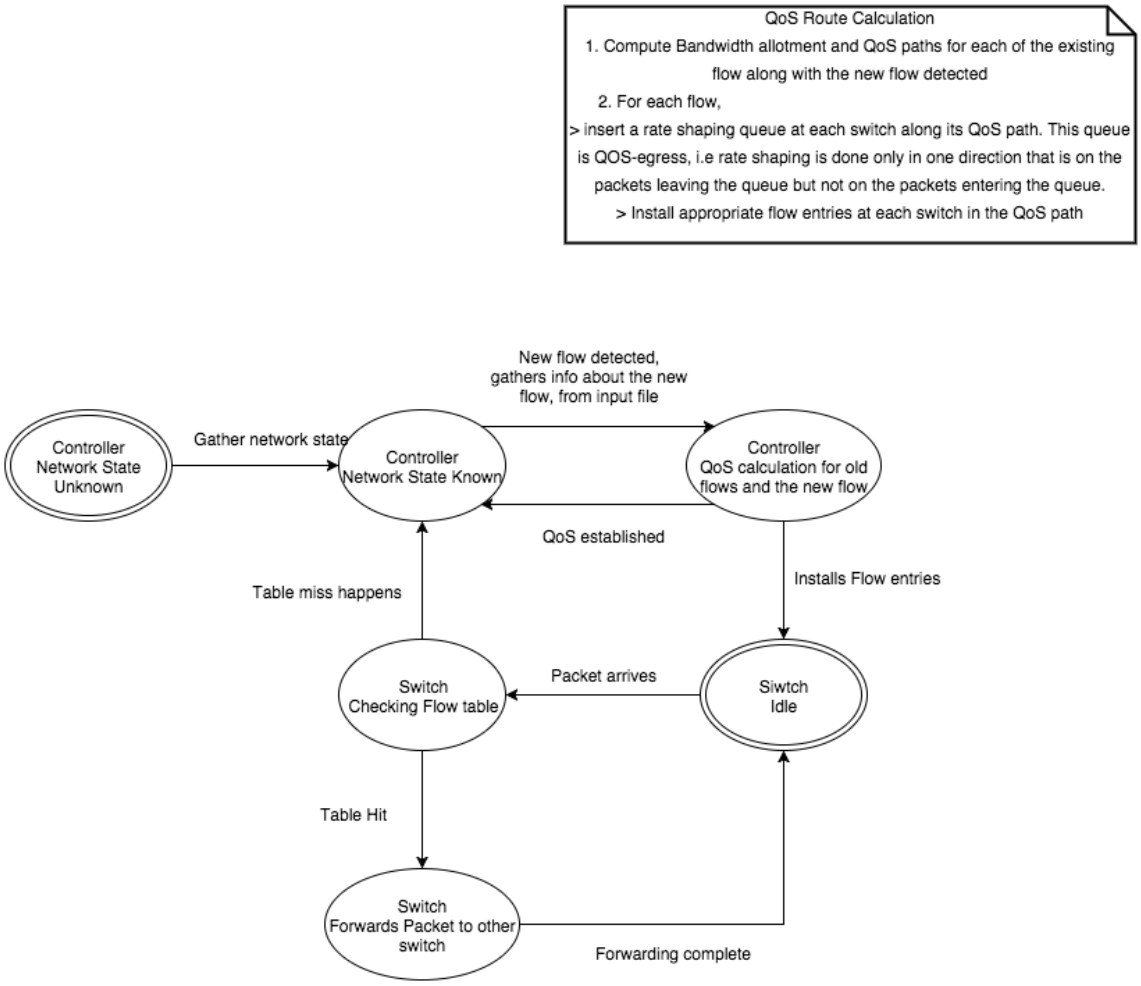


Figure 5.1: State diagram of controller-switch

5.0.3 QOS implementation in Mininet

Bandwidth allotment according to the Utility proportional fairness algorithm is implemented in Mininet with the help of queues at each forwarder. These queues are only QoS-egress, i.e rate shaping is done only in one direction that is on the packets leaving the queue but not on the packets entering the queue.

OpenFlow supports queues for limiting the rate of packets going out of a switch port for implementation of Quality of Service. Queues provide a guarantee on the rate at which the packets leave the queue. So, with this guarantee, multiple queues at various rates can

be used to give preference to priority or special traffic over ordinary or not so important traffic. OpenFlow standard support queues with guaranteed minimum and maximum rates.

Openflow uses the Open Virtual Switch also know as Open vSwitch (OVS). The OVS switch is used for queue configuration specific to each switch. Queues must be set up with OVS commands before using them with Openflow.

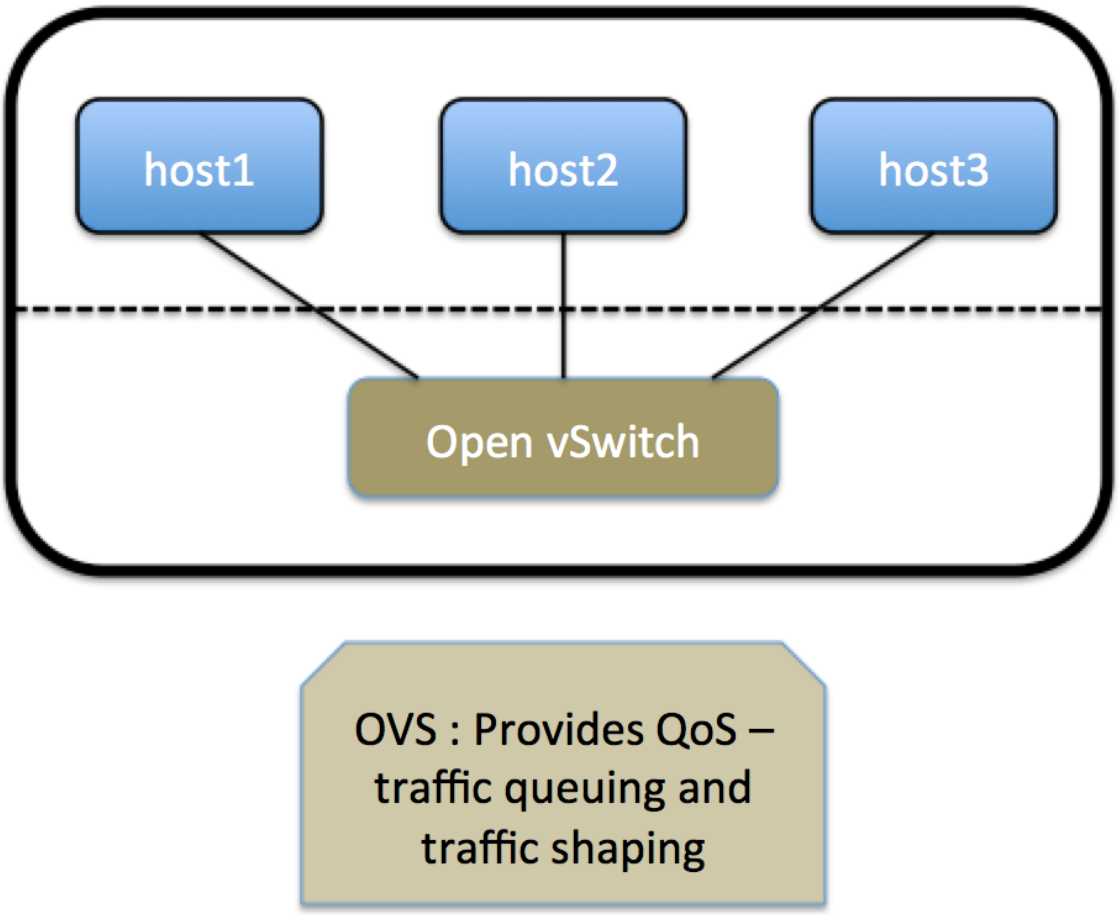


Figure 5.2: Open vSwitch

An example is provided below on how to configure a queue on an Open vSwitch. The mininet installation installs the Open vSwitch package also by default. So once mininet installation is done, the machine is setup to use the Open vSwitch. The example network is provided in the figure 5.3 [11]

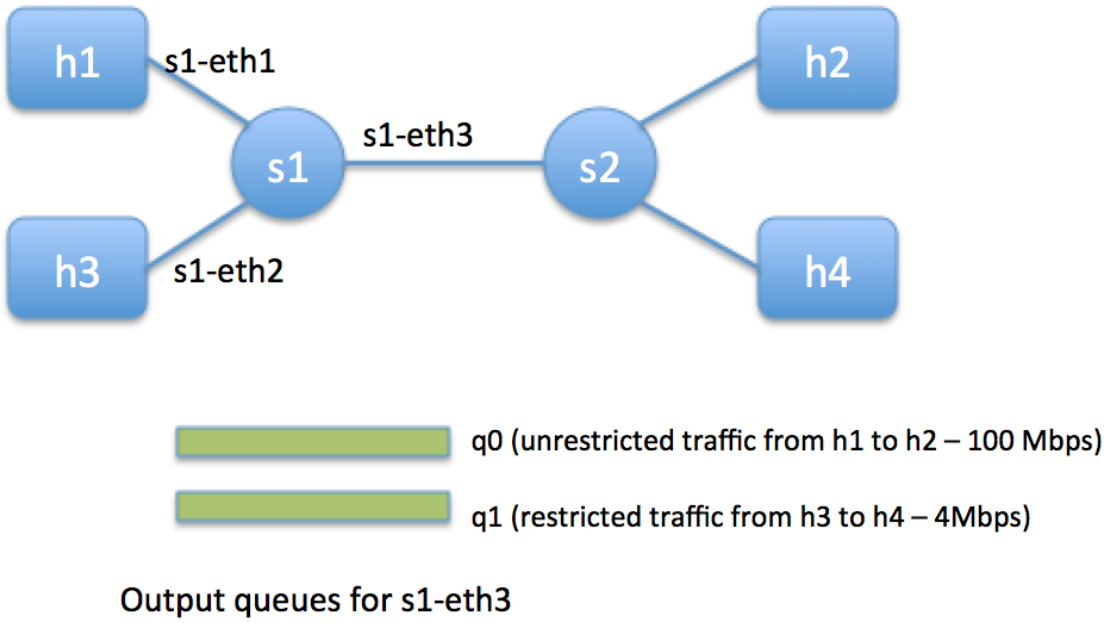


Figure 5.3: Configuration of queues on an OVS switch for QoS

There are 2 switches in the network. 2 hosts are connected to each switch. Lets say, we want to pass unrestricted traffic from h1 to h2 at 100 Mbps, and restricted traffic from h3 to h4 at 4 Mbps.

The below command is run on a linux terminal. The shell interprets the ovs commands.

```
  $ sudo ovs-vsctl -- set Port s1-eth3 qos=@newqos -- \
--id=@newqos create QoS type=linux-htb other-config:max-rate=1000000000 queues=0=@q0,1=@q1 -- \
--id=@q0 create Queue other-config:min-rate=100000000 other-config:max-rate=100000000 -- \
--id=@q1 create Queue other-config:min-rate=4000000 other-config:max-rate=4000000
```

On running the above command, if everything goes well without any errors, exceution of the above command completes by printing the UUID strings for each id in the command as shown in the figure 5.4. We need not be concerned about the UUID strings.

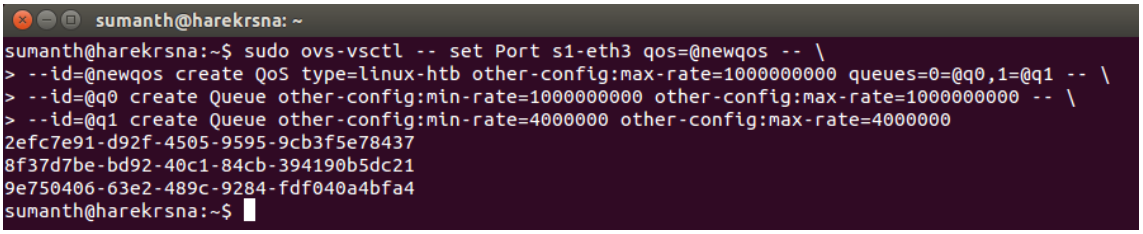


Figure 5.4: OVS command and output

s1-eth3 is the switch port, on which a QoS policy is implemented. A new Quality of

Service instance is created which is a linux-htb type. HTB - Hierarchy Token Bucket shapes traffic using the Token Bucket Filter algorithm. Maximum rate of this QoS policy is set to 1 Gbps. Two queues are created on the port s1-eth3. The first queue q0 is used for enqueuing unrestricted traffic, i.e priority traffic on which we do not want to put any restrictions. The second queue q1 is used to enqueue traffic, to restrict its rate to 4 Mbps. [12]

The flow entries are installed as follows.

```
/* qos_controller.py */
msg = of.ofp_flow_mod()
msg.priority =100
msg.idle_timeout = 0
msg.hard_timeout = 0
msg.match.dl_type = 0x0800
msg.match.nw_src = "10.0.0.1"
msg.match.nw_dst = "10.0.0.2"
msg.actions.append(of.ofp_action_output(port = 3))  # defaults to q0
event.connection.send(msg)

msg = of.ofp_flow_mod()
msg.priority =100
msg.idle_timeout = 0
msg.hard_timeout = 0
msg.match.dl_type = 0x0800
msg.match.nw_src = "10.0.0.3"
msg.match.nw_dst = "10.0.0.4"
msg.actions.append(of.ofp_action_output(port = 3, queue_id=1))
event.connection.send(msg)
```

The measured bandwidths between ($h1 \rightarrow h2$) and ($h3 \rightarrow h4$) using iperf are shown below.

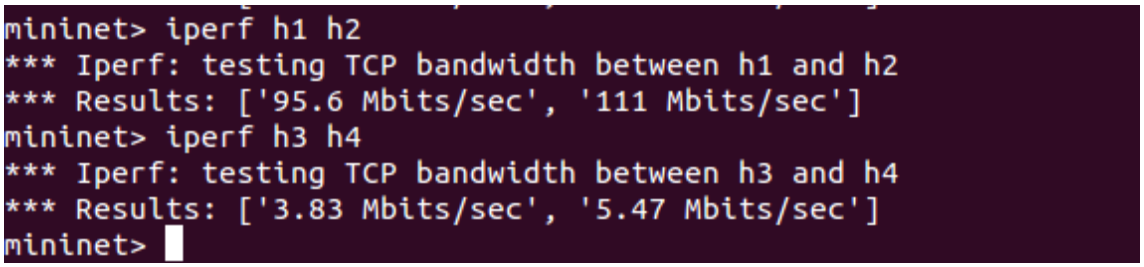


Figure 5.5: Iperf Bandwidth Test

Chapter 6

Experiments and Results

The experiments have been performed to observe how congestion is affecting video QoS. The following experiments help to measure the performance of the Utility Proportional fairness bandwidth allocation algorithm, when applied to the video flows. The two video flows used are in mp4 format and are of the size 720x480p and 360x240p in all the experiments. The duration of 720x480p video is 27 seconds, and that of 360x240p video is 31 seconds. These video files are downloaded from Sample Videos

6.1 Structural similarity (SSIM) index metric calculation

The structural similarity (SSIM) index is used to measure the video quality of the streamed videos on the receiving hosts. SSIM score ranges from 0 to 1. SSIM measures the similarity between the reference video and the output streamed video on the receiving end. An SSIM score of 1 indicates that the output video is identical to the reference video without any loss in quality. The lesser the score, the more is the loss in quality when compared with the reference video.

FFmpeg tool is used to calculate the SSIM similarity score between the reference video file and the saved stream video file. Running the following command outputs the SSIM

score.

```
$ ffmpeg -i saved_stream_video.mp4 -i reference_video.mp4 \
-filter_complex "ssim" "processed.mp4"
```

6.2 Two simultaneous Video streams, with step by step reduction in Link bandwidth

The network used in this experiment consists of 2 switches connected by a link and 4 hosts. 2 hosts are connected to each switch. Hosts h1, h3 are connected to switch s1 and h2, h4 are connected to switch s2. This experiment is repeated multiple times, by reducing the bandwidth of the link between the switches each time from 10 Mbps to 0.5 Mbps.

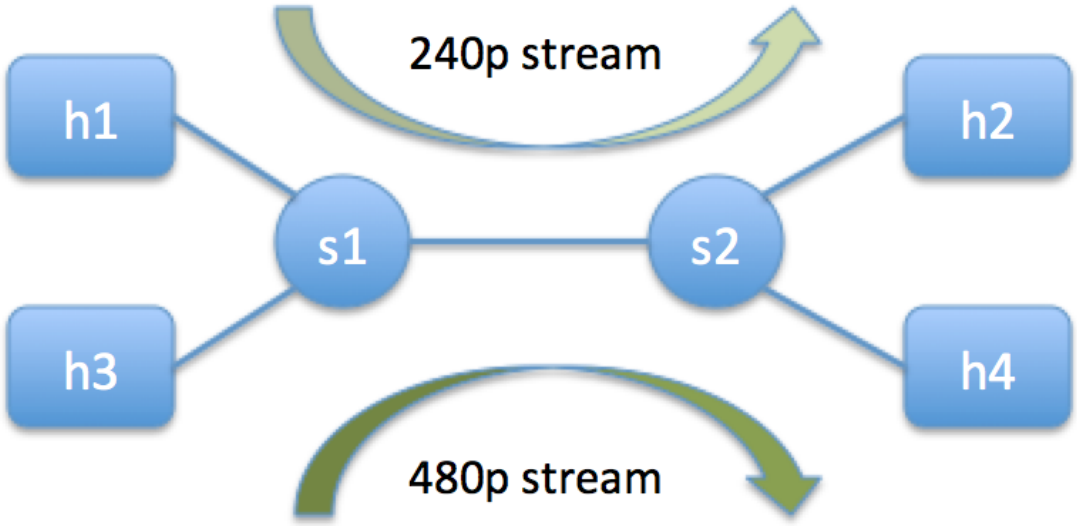


Figure 6.1: Network

This experiment is divided into three parts.

i Only one video is allowed to stream at a time : The smaller video 360x240p is streamed from h1 to h2, with the link bandwidth between the switches set as 10

Mbps and the SSIM score is calculated for the saved stream video on host h2. This is repeated multiple times, each time with a reduction in the link bandwidth. The link bandwidth is reduced from 10 Mbps to 0.5 Mbps over the multple runs of this part of the experiment.

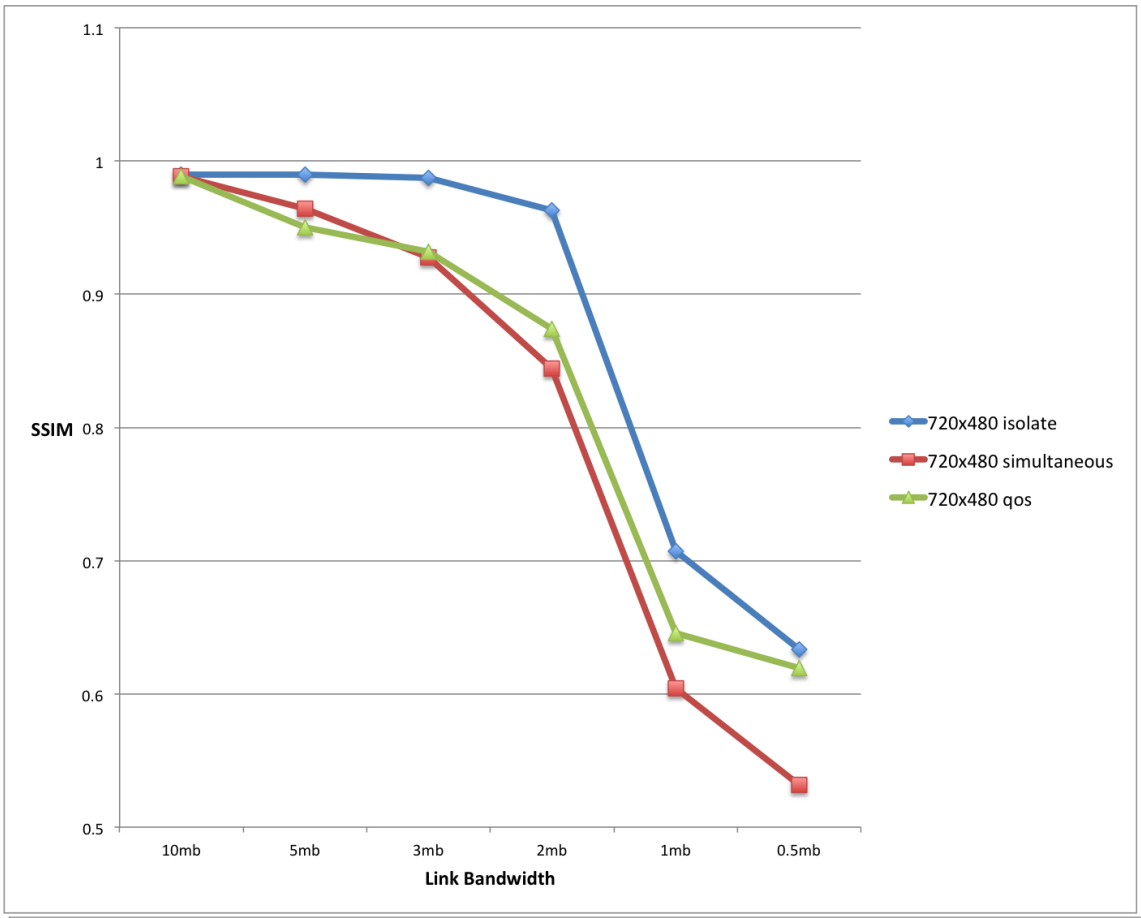
The same is repeated for the bigger 720x480p video too, which is streamed alone from h3 to h4 and the SSIM scores are measured and plotted.

ii Simultaneous streaming without QoS : The above mentioned procedure is repeated but with both the video streams sent at the same time using the link between the switches simultaneously.

iii Simultaneous streaming with QoS : Simultaneous video streaming is done with the QoS policy, obtaining the bandwidth allocation using the Utility proportional fairness algorithm.

Linear utility functions are used for the two video flows. $y = x$ is the utility function for the 480p video and $y = 2x$ is the utility function for the 240p video. This can be understood as follows : the bigger 480p video is not easily satisfied as compared to the satisfaction of the 240p video which gets easily satisfied with an increase in the available bandwidth. With these utility functions, one-third of the bandwidth is allocated to the 240p video flow, and the other two-thirds is allocated to the 480p video flow.

The SSIM plots for each of the video flows are plotted. The three SSIM line curves from the above three parts of this experiment are plotted on a single plot. A plot for each video flow.



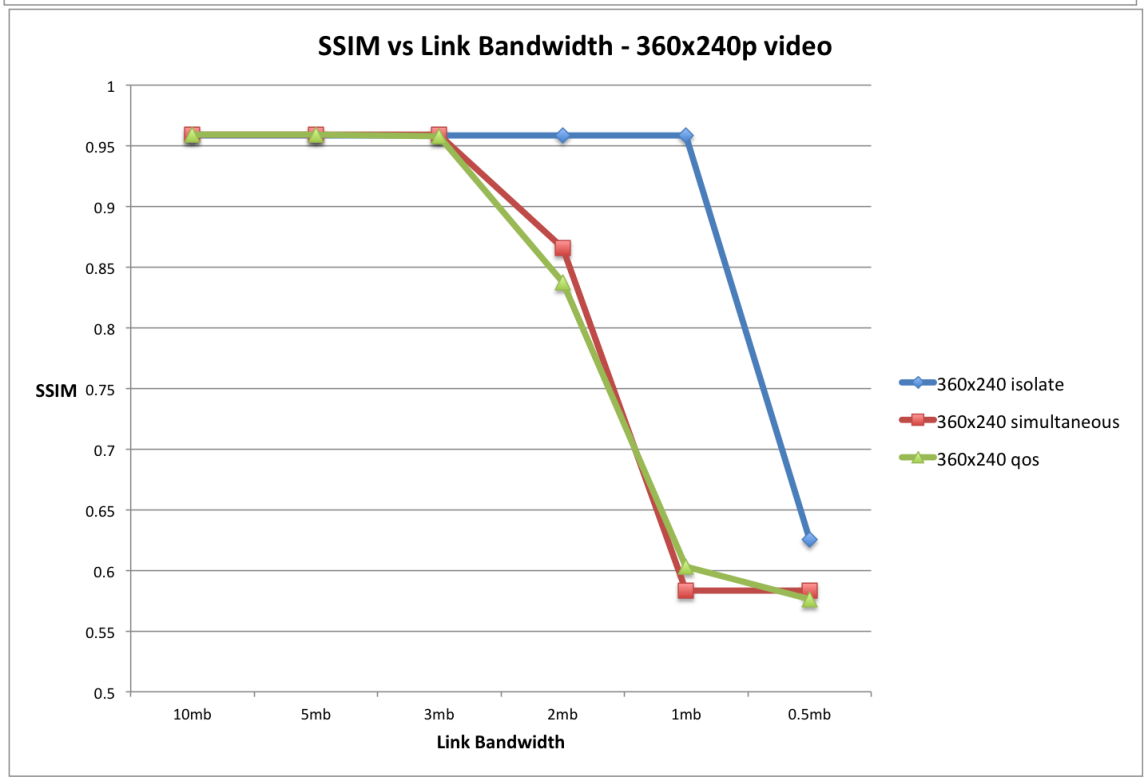


Figure 6.2: SSIM vs Link Bandwidth

It is observed that, with the QoS policy applied, there is an improvement in the video quality of the bigger 480p video, and there is not much difference in the video quality of the smaller 240p video. This can be understood as, the 240p video needs less bandwidth compared to the 480p video. So, allocating more bandwidth to the 480p video improved its streaming performance, at the same time not deteriorating the streaming quality of the smaller 240p video.

6.3 Multiple simultaneous Video streams, with constant Link bandwidth

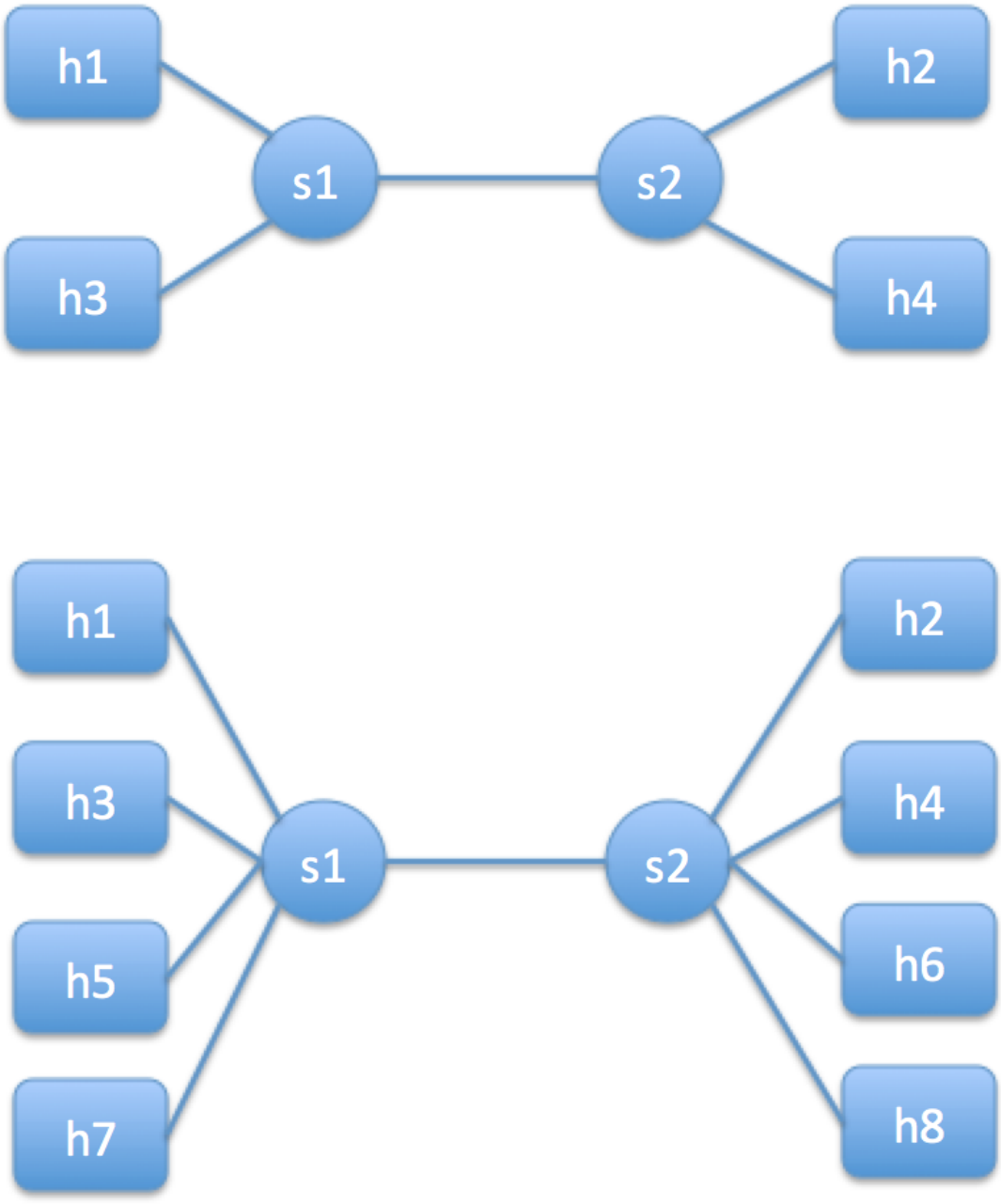


Figure 6.3: Network with hosts increasing on each side

This experiment starts with the same network described as above - 2 switches and 4 hosts with a 10 Mbps link bandwidth between the switches. 2 video flows (240p and

480p) are sent simultaneously over the network. The utility functions are same as used in the previous experiment. Keeping the switches link bandwidth constant at 10 Mbps, the experiment is repeated with increase in the hosts by two on both of the switches to allow for a 1:1 distribution of both the types of video flows. The network starts with 4 hosts, 8 hosts, 12 hosts, ..., till 32 hosts, i.e. 2 flows, 4 flows, 6 flows, ..., 16 simultaneous flows.

6.3.1 Structural Similarity Index - SSIM

For an experiment with a given number of hosts plotted on the x-axis, the average SSIM score of all the multiple video streams is plotted on the y-axis

The plots in figure 6.4 show that, there is an improvement in the SSIM scores when the QoS policy is applied.

6.3.2 Throughput

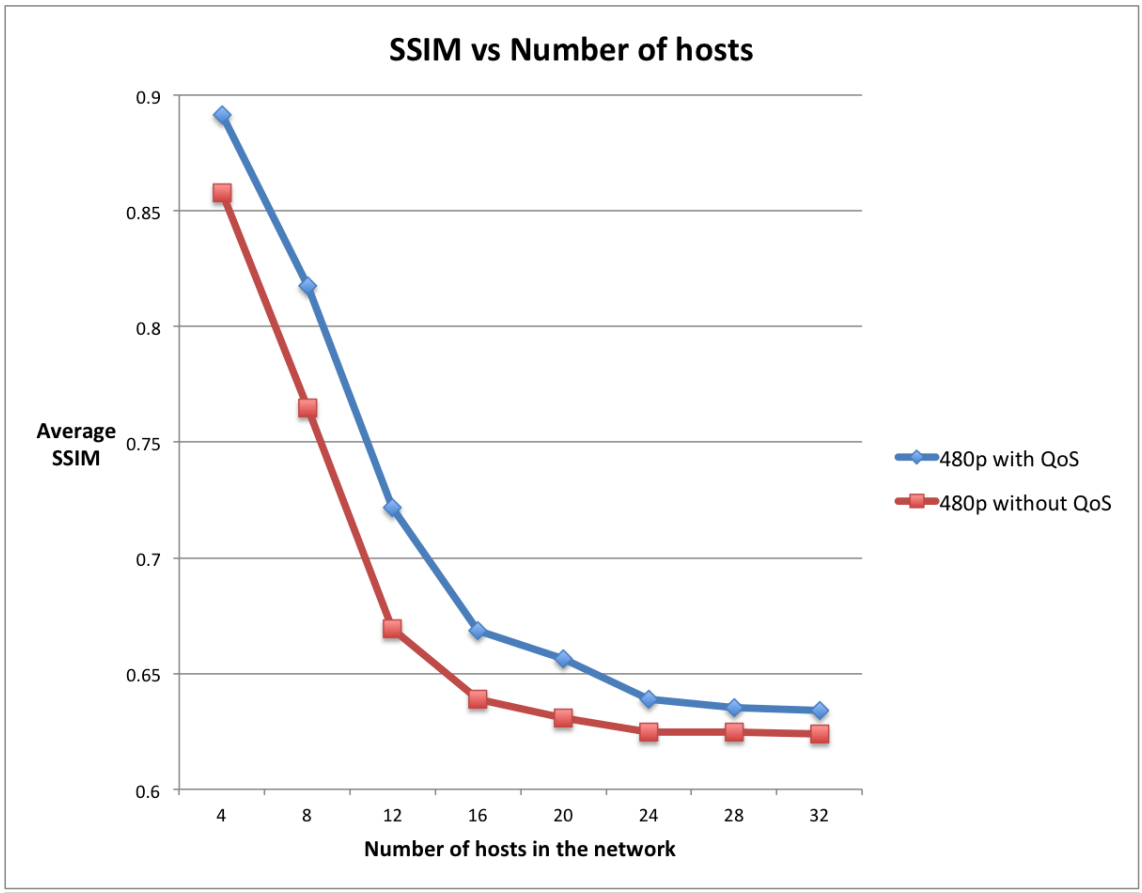
For an experiment with a given number of hosts plotted on the x-axis, the average bit rate in kbps at the receiving hosts is plotted in figure 6.5. Wireshark is used to obtain the statistics.

A significant improvement in the throughput is observed when the QoS policy is applied.

6.3.3 Packet Delay

For an experiment with a given number of hosts plotted on the x-axis, the average per-packet delay in milliseconds at the receiving hosts is plotted in figure 6.6. Wireshark is used to obtain the statistics. The time gap between the arrival times of the consecutive pair of packets at the receiving host is taken as the delay of the second packet.

A significant improvement in the per-packet delay is observed when the QoS policy is applied.



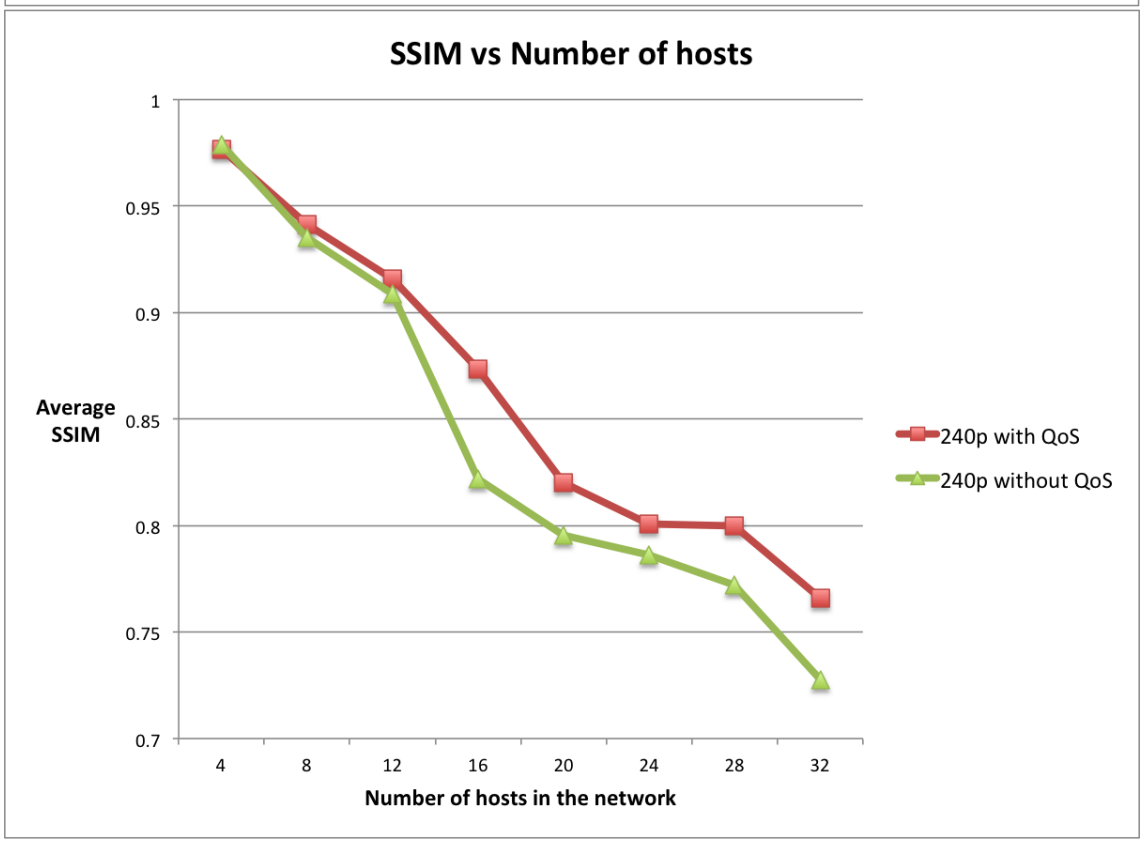
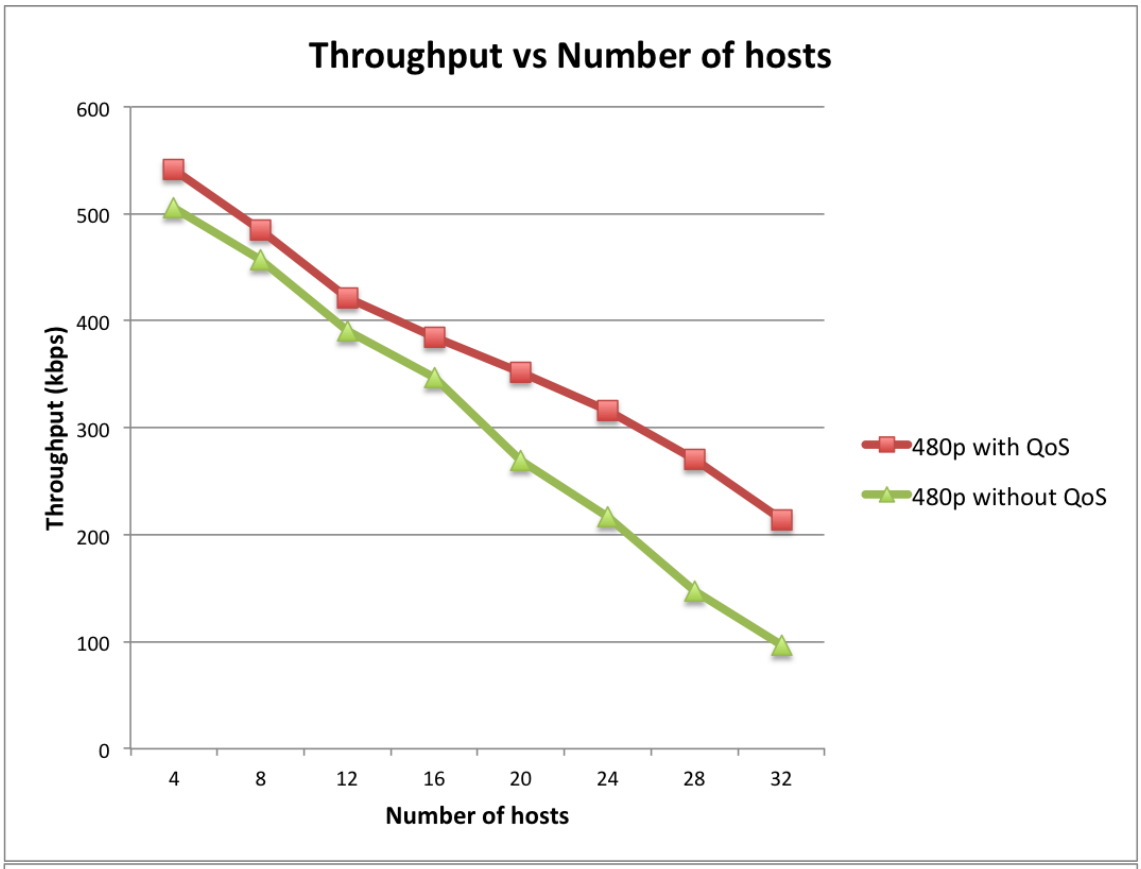


Figure 6.4: Average SSIM of multiple streams



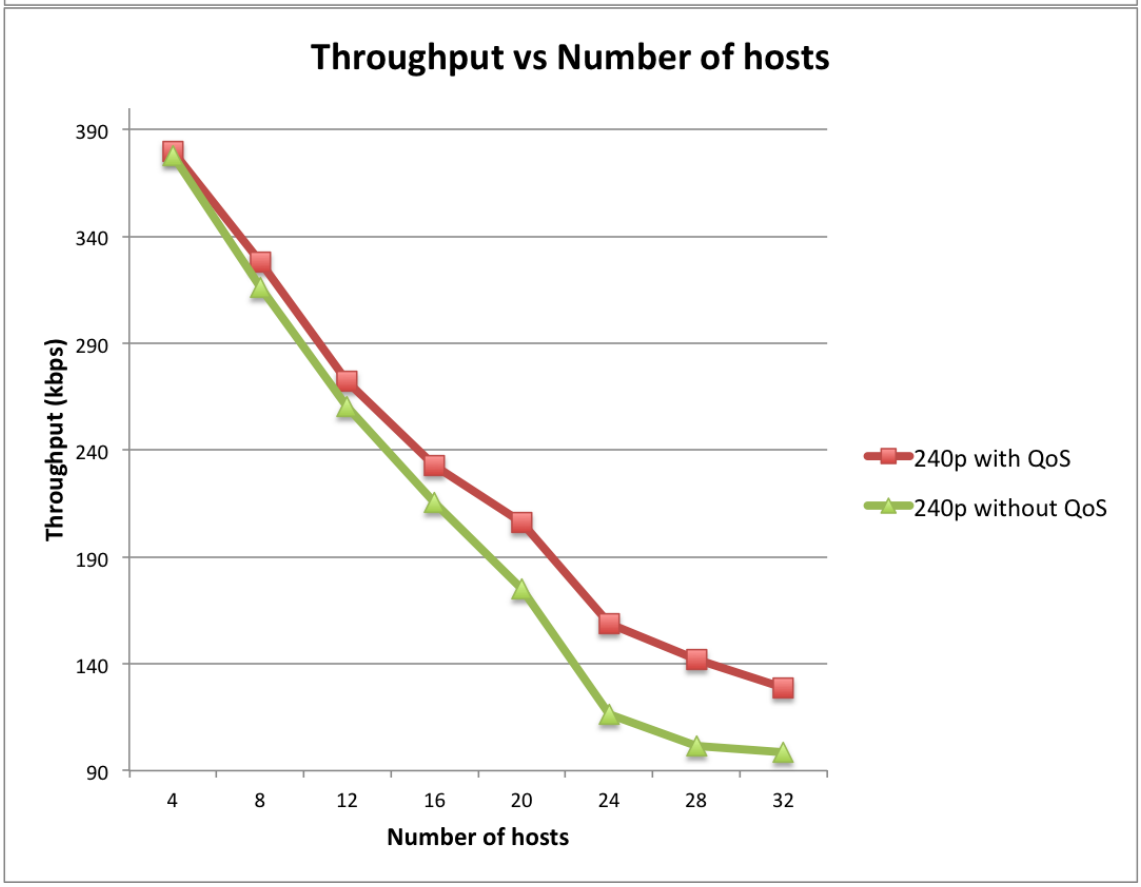


Figure 6.5: Average bitrate of multiple streams

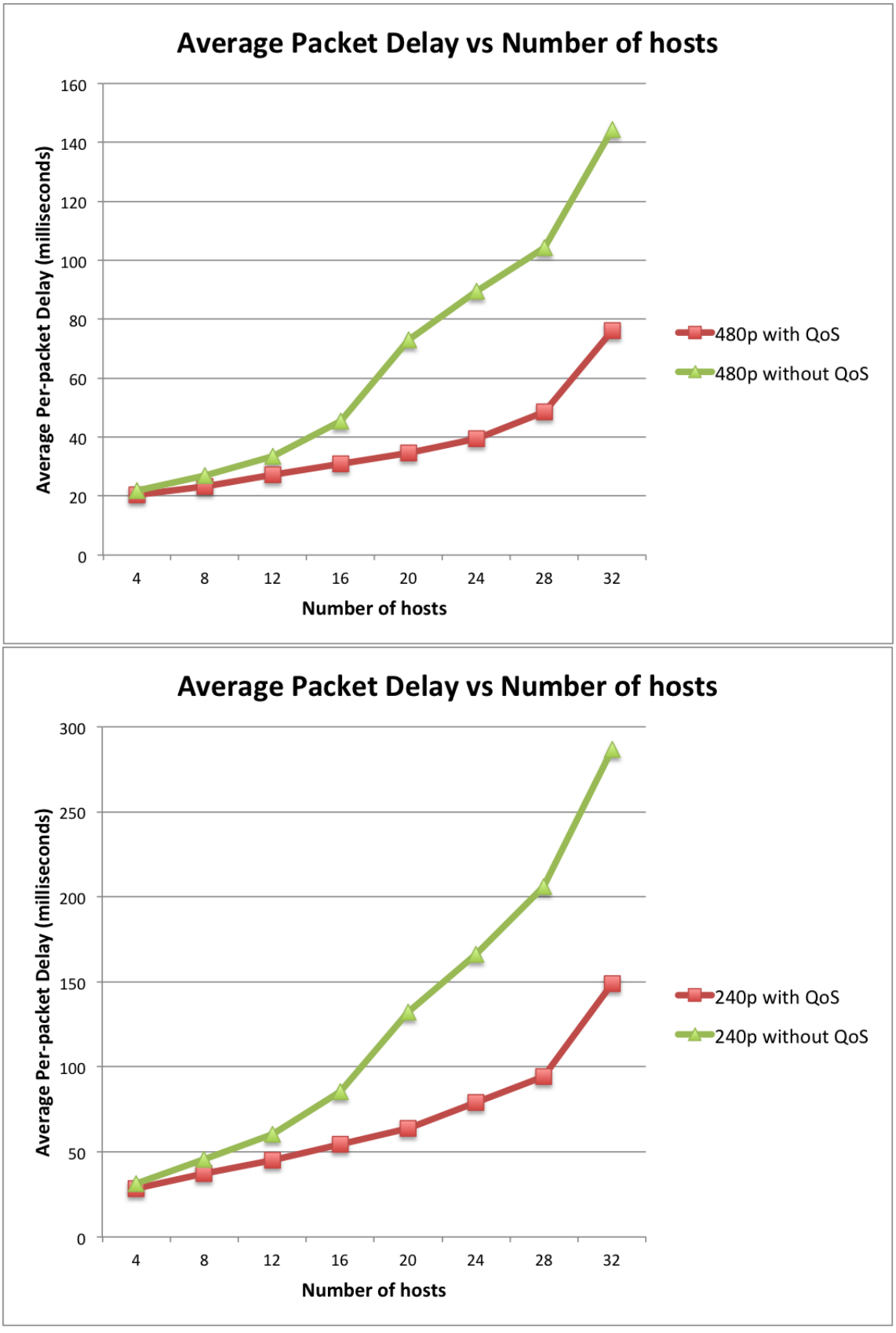


Figure 6.6: Average packet delay of multiple streams

Chapter 7

Conclusions and Future Work

In this thesis, focus is on designing an optimization framework which can provide end to end QoS guarantee in case of large scale multi-service provider network. The video streams had a better quality when the QoS policy is applied.

We have only used a single controller in the current thesis. The algorithm has to be extended to support multiple controllers as in a distributed controller system. The controller-to-controller interface has to be efficiently designed.

Designing and implementation of the Controller-to-Controller interface in a distributed system of controllers is left for the future work. Also, efficient topology gathering and resource monitoring method needs to be developed for a distributed controller system.

Bibliography

[1] Hilmi E Egilmez, S Tahsin Dane, K Tolga Bagci, et al. Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks. In *Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific*, pages 1–8. IEEE, 2012.

[2] Frank Kelly. Charging and rate control for elastic traffic. *European transactions on Telecommunications*, 8(1):33–37, 1997.

[3] Steven H Low and David E Lapsley. Optimization flow controlâĂŤi: basic algorithm and convergence. *IEEE/ACM Transactions on Networking (TON)*, 7(6):861–874, 1999.

[4] Kannan Govindarajan, Kong Chee Meng, Hong Ong, Wong Ming Tat, Sridhar Sivanand, and Low Swee Leong. Realizing the quality of service (qos) in software-defined networking (sdn) based cloud infrastructure. In *Information and Communication Technology (ICoICT), 2014 2nd International Conference on*, pages 505–510. IEEE, 2014.

[5] Bradley R Smith and JJ Garcia-Luna-Aceves. Best effort quality-of-service. In *Computer Communications and Networks, 2008. ICCCN'08. Proceedings of 17th International Conference on*, pages 1–6. IEEE, 2008.

[6] Diego Kreutz, Fernando MV Ramos, P Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *proceedings of the IEEE*, 103(1):14–76, 2015.

[7] Frank P Kelly, Aman K Maulloo, and David KH Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research society*, pages 237–252, 1998.

[8] Wei-Hua Wang, Marimuthu Palaniswami, and Steven H. Low. Application-Oriented Flow Control: Fundamentals, Algorithms and Fairness. *IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 14, NO. 6)*, December 2006.

[9] POX community. POX Openflow Wiki. `https://openflow.stanford.edu/display/ONL/POX+Wiki`, 2015. [Online; accessed 25-April-2016].

[10] G. M. Bernstein. SDN Fun! (Hands on with SDNs). `https://www.grotto-networking.com/SDNfun.html`, 2016. [Online; accessed 25-April-2016].

[11] Chih-Heng Ke. Set traffic to different output queues. `http://csie.nqu.edu.tw/smallko/sdn/mySDN_Lab5.pdf`, 2013. [Online; accessed 25-April-2016].

[12] David R Newman. QoS on OpenFlow 1.0 with OVS 1.4.3 and POX inside Mininet. `http://users.ecs.soton.ac.uk/drn/ofertie/openflow_qos_mininet.pdf`, 2014. [Online; accessed 25-April-2016].