HALMSTAD UNIVERSITY

Bachelor Degree of Computer Science and Engingeering

BACHELOR THESIS

High Quality of Service in SDN

Bandwidth gurantee with QoS

Emma Andersson, Johan Bröhne

Computer Science and Engineering. 15 credits

Halmstad University, September 28, 2017–version 1.2

ABSTRACT

Video streaming through IP networks has risen rapidly over the recent years, and will continue to do so over the coming years. In addition to this, new technologies such as Virtual Reality and robotics will lead to many new applications that will put high pressure on the networks. To combat these challenges, networks need to be application sensitive, and be able to provide Quality of Service (QoS) based on requirement. Network paradigms like Software Defined Networking (SDN) enables the network to be directly programmable, and could thus solve the challenge. In this thesis, the objective is to research if SDN can provide High QoS.

Methods are developed to achieve High QoS with SDN. A combination of Differentiated Services Code Point (DSCP) values and DSCP remarking with Meters are used enable High QoS and together they can give bandwidth guarantee. As a result of the thesis, a solid theoretical method is provided for achieving QoS, tests are performed and show that QoS can be implemented in SDN, but it is unable to implement High QoS due to the lack of implementation for Meters with DSCP remarking.

ACKNOWLEDGEMENTS

We would thank Time Critical Networks for the support and guidance during the thesis.
We also want to thank Professor Magnus Jonsson for academic support and guidance.

CONTENTS

1 INTRODUCTION

1.1 PROBLEM DESCRIPTION

In recent years there have been a growing popularity of video streaming in best effort IP networks, thanks to the increasing computational and display capabilities of user devices. The usage of video streaming will continue to increase rapidly and according to a forecast from CISCO [1], by year 2020 video streaming by IP will be 82% of total IP traffic.

Without adaption from user applications and advanced traffic control, insufficient network resources could cause network congestion. This can lead to video quality degradation that manifests as packet losses or frame freezing during playback. This is particularly critical in unmanaged best-effort networks where there are growing numbers of concurrent video streaming applications from a variety of different user devices.

Furthermore, with technologies like Virtual Reality (VR) and robotics, we can expect many new applications and services. In the not so distant future, a worker might perform his/her task using live video feed from a set of camera eyes of a robot that stream to a pair of VR-glasses. Interaction with objects and the surrounding area might be done from hundreds of miles away, through the hands of said robot. The interactivity of the remote control and the operation applications makes it very sensitive to packet delay and jitter. They require high standards up to the point that they put hard real-time demands on the network.

For these new applications to meet their demands, next generation networks should become application aware and allow applications that are highly sensitive and have the most demanding requirements to negotiate with a network controller to allocate resources and receive a guarantee on the received network resources, for example, a guarantee on a maximum delay of packet delays.

When an application has made a negotiation with the network controller, network resources are allocated to a dedicated channel for that specific application. In practice this is done by automatically configuring flows and port priorities to keep the promised network guarantee. Upcoming technology paradigms like Software Defined Networking (SDN) will enable implementations such as this and other similar ideas.

To take the first step towards a next generation network, the company TCN has defined a project to develop to research SDN. The purpose is to research if SDN is capable of giving high Quality of Service (QoS).

1.2 PROJECT OBJECTIVE

The objective of the thesis is to research if SDN can provide high QoS to selected hosts. The QoS should enable end-to-end bandwidth guarantee between hosts.

To reach high QoS, the following features should be applied

- Provide end-to-end bandwidth guarantee from one host to another
- Correctly prioritizing traffic depending on guarantee, thus protecting the given limits
- Allowing excess traffic when resources are available

1.3 PROJECT QUESTIONS

To achieve the objective of the project, following questions has been developed and must be answered.

- How can SDN provide end-to-end bandwidth guarantees between hosts?
- How to prioritize traffic correctly, thus providing guarantee to selected traffic while pushing aside excess traffic?
- How to allow excess traffic to flow in cases of excess resources?

1.4 PROJECT CONSTRAINTS

The project is limited by the hardware that is used in simulating the network. The hardware used have the following specification.

- Processor: Intel Core i7-4510U CPU @ 2.00GHz x 4
- Graphics card: Intel Haswell Mobile
- RAM: 8GB
- OS: Ubuntu 16.04 LTS

Furthermore, in terms of software, the project was specified to use open source programs and tools.

Software emulated switches will be used, and should also be open source.

Also, the protocol OpenFlow is to be used.

2 BACKGROUND

2.1 SDN

Open Networking Foundation (ONF) can be used to describe SDN. Normally, a networking product has two planes, control and forwarding plane. The forwarding plane is responsible for sending data through the network product, from one port to another. The forwarding plane is sometimes also referred to as the data plane. The control plane manages the intelligence in the network product, it decides where to send the packets that comes through. For example, it can read routing tables to deduct were the packets should be sent.

The architecture of an SDN network breaks down the control and forwarding plane functions so that the network becomes directly programmable and the underlying infrastructure becomes detached from applications and network services. SDN takes over the control from the switch. The traditional static network is transformed to a responsive, intelligent and programmable network that is centrally controlled. [2] [3]

SDN is "The physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices."[3]

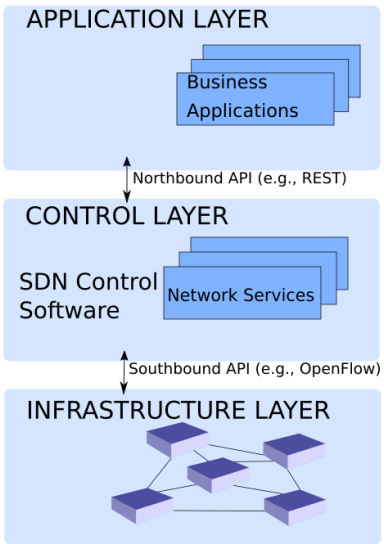

Figure 1: A visual representation of the seperation of the planes

According to SDN: Software Defined Networks: An Authoritative Review of Network Programmability Technologies [4], the control plane is a combination of Layer 2 and Layer 3 of the Open Systems Interconnection (OSI) model [5]. And thus, the data plane is comparable to the Layer 1 of the OSI model.

SDN focuses on a few primary functions. [2][3]

- Separation between control plane and forwarding plane.
- A centralized controller that gives an overview over the network.
- The control over the network is directly programmable, because the forwarding and control functions are detached.
- Because of the detachment between control and forwarding plane it makes it easier for administrators to adjust the traffic flows dynamically to meet changed conditions.
- The logic of the network is inside a centralized software based SDN controller that maintains an overview over the network, which is shown as a logic switch for applications
- SDN allows network administrators to configure, manage, secure and optimize network resources quickly and dynamically with the help of automated SDN programs, that they can program themselves so they don't have to rely on proprietary software.
- When implemented through open standards, SDN simplifies network design and operation because instructions are provided by SDN controllers instead of multiple, vendor-specific devices and protocols.

2.2 SDN CONTROLLERS

The architecture of SDN separates the control and forwarding plane. It manages the control plane separately in a SDN controller. The controller then becomes the centralized brain in the network. A controller works in the same way for a network as a Operating system (OS) works for a computer. The controller is the center of the network and it is responsible for modification of the data and for the communication between applications and network products. Also, with the help of the protocol OpenFlow (OF), it monitors the network.[6]

When an OF switch receives a packet that is completely new, a packet which the switch does not have any instructions or matching data for, the switch sends information about the packet to the controller for instructions. The controller receives the packet and decides how to handle it. It can drop the packet, send it out through a specific port or instruct the switch how to handle similar packets in the future.[7]

There are a heap of different SDN controllers.

- `NOX` [8]: C++ with multithreading.
- `POX` [9]: Based on Python, often used for quick prototyping.
- `Beacon` [10]: Based on Java with multithreading, relies on OSGi and Spring frameworks
- `Floodlight` [11]: Based on Java with multithreading, relies on Netty framework
- `MUL` [12]: C with multithreading
- `Maestro` [13]: Java with multithreading
- `Ryu` [14]: Python, good for quick prototyping.
- `OpenDaylight` [15]: Based on Java.

These mentioned are some of the most popular which also fits into the project constraints of being open source.

2.3 OPENFLOW

OF[16] is a protocol that is a vendor independent standard for communication between the forwarding plane and the controller in the SDN architecture.[3]

OF is a Southbound API for communication between the controller and the switches, thereof it is of great importance that both the controller and all the switches understands the OF protocol. The protocol is used to gain control over the forwarding tables that exists in the switches, with control over the forwarding tables the controller can become the centralized brain in the network. Thus, it is OF that enables the controller to take control from the switches[17]

The architecture of OF consists of the concepts[17]

1. OF compatible switches that makes up the forwarding plane.
2. OF compatible controller that makes up the control plane.
3. A secure channel that binds the forwarding plane and control plane together.

2.4 QOS IN NETWORKING

Accordning to Internet QoS: Architecture and Mechanisms for Quality of Service the defintion of QoS is *"the capability to provide resource assurance and service differentiation in a network."*[18].This can be achieved by different traffic prioritizing or by reserving resources. By giving different priorities to different flows it can provide a much higher quality to those who are in dire need of it, for example, a video stream application can get much higher priority than web browsing.

By giving higher priority to a specific flow, it can enable bandwidth guarantee up to a set limit, since everyone with a lower priority has to wait. Not only can it give bandwidth guarantee to select hosts, it can also be used for congestion avoidance. By giving select traffic low priority in cases of congestion, the network can drop low priority traffic to avoid congestion and make room for those with high priority. [19]

In HiQoS: An SDN-based multipath QoS Solution [20], different tests are performed to show how QoS impacts the network performance. In this article, tests between no QoS and simple priority queuing are established.

The tests are performed in the same way. Two servers are sending data through a switch with the same Host as destination. Server1, which is considered the important one, streams 1Mbit/s. Server2, which is the excess one, streams 3.5 Mbit/s. Together they stream 4.5Mbit/s through a switch with a link limit of 4Mbit/s, thus causing congestion. When congestion occurs, the response time of both Server1 and Server2 is tested to see how reachable they are when streaming data.

The test with no QoS support is called LiQoS, and no priority is given, they have to share the 4mpbs link.

The test with simple QoS is called MiQoS and uses two different queues. Queue 1 has 1.5Mbit/s reserved bandwidth . and is dedicated to the more important Server1, and Queue 2 has 2.5Mbit/s reserved bandwidth and is for Server2.

The following table is derived from the test results of LiQoS and HiQoS, and displays the response time of each server in both testes.

Table 1: Response times

	LiQoS	**MiQoS**
Server1	3000ms	0ms
Server2	3000ms	5000ms

In LiQoS, they have equal response time due to no priority between the servers. In MiQoS, the respons time for the more important Server1 is almost instant, since it stays withing the given queue limit. Only the less important Server2 is getting a huge respons time since it exceeds its given queue rate.

From this, it can be seen how QoS can be used to differentiate between sources of different importancy, and how QoS can affect the perfomance of the network.

2.5 REALATED RESEARCH IN THE FIELD OF QOS IN SDN

With video streaming via IP on the rise [1], network congestion control and Qos is now more important than before. Since SDN and OF enables networks to be more controllable and intelligent thanks to the network being programmable, network administrators no longer have to leave networks unmanaged.

Because of this, researchers have presented different models that achieve QoS in some ways with SDN and OF. Such researches can give guidance since the objective in the thesis is to research ways to achieve high QoS.

In HiQoS: An SDN-based multipath QoS solution [20], QoS is achieved through class specific bandwidth guarantee. HiQoS divides traffic into three different classes. Video streaming, interactive multimedia and best effort stream. These three classes are forwarded into three different queues with preconfigured rates, meaning each queue has a fixed minimum and maximum bandwidth of the classes that passes through.

This way, QoS is achieved between the three classes, but in cases of too much bandwidth from a single class, congestion will occur for that matching queue. Thus, the scalability is limited since too much traffic inside a queue will cause congestion, and the number of queues has to be pre-configured. An adaptive QoS would be a per-flow based bandwidth guarantee, giving guarantee unique to each flow.

In addition to QoS queues, HiQoS also implements a rerouting mechanism that quickly reroutes traffic to other paths if a path is broken.

In another paper, Implementing quality of service for the software defined networking enabled future internet[21], per-flow bandwidth guarantee is used. Here, the type of traffic is dependent on the DSCP/-TOS bits. Traffic is divided into two separate queues. One queue is for traffic with DSCP/TOS bits enabled, and the other queue is when it's not enabled, in other words, for normal traffic. Although it can only divide into two classes with the DSCP/TOS bits, three different queues exist. The third queue is for control traffic, which it cannot automatically divide a flow into.

It simply divides all DSCP/TOS traffic into high priority queue and all regular traffic into best effort queue. When a new flow with high priority traffic is detected, a new rate limiter queue is set up and the flow is directed into it. The rate of the queue is supposed to be negotiated between the flow and the controller. When the rate is set, the flow cannot exceed that given rate, even if the link is mostly empty.

Both papers gives insight on how to perform QoS in SDN. Although, the second paper has an edge for the project objective in this thesis, since it provides a per-flow guarantee. Both papers do have a downside in the way that they do not allow excess traffic in cases of available network resources. The flows rate is limited by the configured rate of the queue, even if the queues use different ways to calculate said rate.

3 METHODS

3.1 CHOICE OF QOS METHOD

Internet Engineering Task Force (IETF) present different QoS architectures. The two most commonly presented are Integrated Services [22] and Differentiated Services [23].

These two architectures are examined and compared to each other, for the purpose of the selection of QoS architecture for this project.

3.1.1 *Integrated Services (IntServ)*

IntServ uses resource reservation to achieve QoS.[22] Every router in the IntServ domain has to follow the same resource reservation policy.

The way IntServ works is that every application that want to receive QoS has to make an reservation in each of every routers in the path between the application and the target destination. Each router with IntServ receives the reservation request and has to respond if they are able to fulfill that request. Every router between the sender and the receiver need to accept the request for the guarantee to come through.

IntServ is divided into two functions. The first function is Flow Spec, which is used by applications to describe their needed reservation for the routers. The second function is the Resource reservation protocol (RSVP) which is used to communicate the requests and answers between routers and applications.

Flow Spec
The Flow Spec is divided into two subparts.

- `TSPEC:` Traffic SPECification. It is used to provide details about the traffic from the application. Details such as frames and packets are provided here so router know how much they need to reserve.
- `RSPEC:` Request SPECification. RSPEC is used to specify the different requirements of the flow. There are three settings, "Best Effort", "Controlled Load" and "Guaranteed".

RSVP

The RSVPP is the protocol in charge of communicating between the applications and the routers for reservations. [24]

RSVP primarily use two different messages to carry out the communication.

- `PATH:` The PATH message is sent by the sender host that wishes to send QoS traffic to a receiver. The PATH message is used to identify the data paths and sends the TSPEC of the sender.
- `resv:` Reservation Messages. The receivers of a PATH message can choose to return a resv message to initiate a flow. The resv message contains the flowspec and is sent through the datapath of the PATH message in reverse, back to the sender. From the resv message, the routers can reserve the correct amount of resources for the flow between the sender and receiver.

IntServ uses a method called Soft State, meaning that the reservations will be cancelled if nothing is heard from the sender after a set period of time. This will also help to dismiss any reservations in cases of link failure like crashes.

3.1.2 *Differentiated services (DiffServ)*

DiffServ is a computer networking architecture that used Differentiated Services Code Point (DSCP) values to classify the packets that goes through its domain. The DiffServ domain is a collection of routers that all use the same defined Diffserv policies [25]. A DiffServ domain is composed of a group of interconnected DiffServ nodes that use the same service policy and PHBs. Each router inside the DiffServ domain is configured to differentiate traffic priority based on what class the incoming packets have.

The DiffServ architecture is based on the principle of traffic classification. The principle is that every packet is placed into a selected number of traffic classes, and thus, is prioritized differently depending on class when entering a DiffServ domain. The routers in the DiffServ domain reads the DSCP value from the IP-header to know how to differentiate the traffic.

Routers that implement DiffServ also implements a term called per-hop behavior (PHB). PHB is used to define the packet-forwarding properties that is associated with each class. PHB is used to apply the policies and priorities to a packet of selected class when performing a hop. A hop can be a transfer from one router to another for example. The PHB behavior is thus determined by both the DSCP value and the Explicit Congestion Notification (ECN) bits.

3.1.3 *Comparison & Selection*

Both QoS architectures described allows for per-flow QoS and thus, per-flow Bandwidth guarantee, which is in line with the project objective.

However, the way that they achieve this is different. DiffServ achieves this by using DSCP to give different priorities to flows, while IntServ does this by making each router in the flowpath reserve resources for each soecific flow. The actual packet handling is also done per flow.

This causes problems for IntServ when used in a large scale network. Each router has stored information and reservation from each and every flow that passes through. In a large scale network, each router will have to store a lot of reservations and it will become hard to keep track of everything. The reservations are different from one router to another since they can have unique flows running through them, making it even harder to keep track of every reservation.

DiffServ avoids this problem. Since it is class based, the routers only need to know what to do when a specific class arrives, and every router has the same class configuration. This makes it a lot easier to keep track of and to make configurations, and it makes the network a lot more scalable, since the class definitions will stay the same even if more flows are created.

Another problem that can arise with IntServ, is when flows are not consistent. As described, IntServ drops any reservations when a period of inactivity has occurred to make space for new reservations. In cases where traffic is not consistent, the applications would have to request reservations each time it wants to send something, making it a very redundant experience.

Again, DiffServ avoids this problem thanks to a class based approach. It does not matter how often a application sends packets, the class configuration is already established and no further setup is needed for transmit.

From this comparison, class based QoS is the best approach and thus, DiffServ is the QoS architecture of choice for this project.

3.2 DIFFERENTIATED SERVICES CODE POINT (DSCP)

To achieve different priority levels in different flows, DSCP values will be used. DSCP is 6-bits that is used to classify different priority levels of IP-packets. The 6-bits are contained within a 8-bit field called Differentiated services field (DS field). The other 2-bits in the DS field is called ECN. ECN changes the way how network signals congestion. With ECN, networks can signal network congestions by marking the 2 ECN bits in the DS field. Usually, networks signal network congestions by dropping packets, thus, ECN allows notification through the network without dropping packets. [23] [26]

It is the DSCP bits and the ECN bits that makes up the DS field. The DS fields itself is contained within the IP header of a packet. Formerly, the IPv4 Type of Service (TOS) field could be used for prioritizing different traffic, the DS field is the modern redefinition of the TOS field [27]

IP Precedence Values
The Precedence Values are contained in a 3-bit field which was used to determine the priority of the packet. The field has a range from 0 to 7, 0 is the lowest priority and 7 the highest. The eight values is the following:

Table 2: Precedence Values

Value (Decimal)	**Description**
000 (0)	Best-Effort
001 (1)	Priority
010 (2)	Immediate
011 (3)	Flash
100 (4)	Flash Override
101 (5)	Critical
110 (6)	Internet
111 (7)	Network

The DSCP values are often compared to the equivalent Precedence Values, since DSCP values and DS field is the replacement for TOS.

3.2.1 *DSCP values & classes*

Since the DSCP value consists of 6-bits, mathematically it can have 64 different values and thus resulting in 64 different traffic classes. Although, not every value is commonly used in networks. [28]

As for PHB, there are no predetermined behaviors encoded. This gives the developer freedom in defining their own behaviors for traffic classes. That said, most networks uses four commonly defined PHB.

1. `Default Forwarding`

   Default forwarding PHB (DF PHB) is the most basic PHB, and is generally the bottom line. Any traffic that does not meet any of the requirements of the defined classes in the network is placed in DF PHB. The traffic that succumbs to DF PHB is usually placed as best-effort traffic, meaning no QoS or guarantee is given. [29]

2. `Expedited Forwarding`

   Expedited Forwarding PHB (EF PHB) is used for traffic that require low latency and low packet loss. This is the highest priority between different traffic classes. Use of EF PHB traffic is usually restricted, since the applications using EF is very sensitive to latency and packet drops. If you would use too much traffic of the same highest priority simultaneously ,it can cause overloads within the priority level, and it will cause delays, thus disabling the purpose of EF PHB. Applications such as VOIP, streaming or any other real-time service benefits greatly from correct usage of EF PHB. [29] [30]

3. `Assured Forwarding`

   Assured Forwarding PHB (AF PHB) is used to assure the delivery of the packets, as long as the traffic is kept within a given constraint. Traffic that exceeds the given constraint risks getting dropped in cases of network congestion, the droprate is dependent on what DSCP value it has. AF is suitable to use if you are trying to implement bandwidths guarantee, since you can assure the delivery withing a given bandwidth.[29] [31]

   AF is divided into four different classes with three different drop rates, thus giving 12 different DSCP values that can be used to tune your DiffServ domain.

The four classes are almost the same in priority, only with higher classes winning out in cases of network congestion when the traffic is divided between different classes. This is usually achieved through Weighted fair queuing (WFQ)

WFQ

WFQ is an algorithm that divides traffic into weight classes. Each packet is placed into a class specific queue which receives a rate depending on its weight compared to the totals sum of weights between all classes. WFQ is extension to normal Fair Queuing (FQ), its purpose is to deliver equal rate divided to all flows. [32]

The rate that each weight class received is calculated with:

$$R \times \frac{w_i}{\sum_{k=1}^{n} w_k} \tag{1}$$

Where R is the maximum link rate and w is the weight of a given flow. The weight is divided by the sum of all weights to give a fraction of the rate for the flow to receive. The weight of each individual flow has to be configured, in AF, higher classes get a higher weight to create priority between classes.

The three drop rates that exist are Low/Medium/High. The drop rate indicates the probability that the packets will be dropped in case of congestion where the traffic all belongs to the same class. If congestion occurs, packets with higher drop probability will be dropped. To avoid issues with Tail drop, Random early detection (RED) is used when packets are dropped.

RED and Tail Drop

Tail drop is a simple queuing management algorithm. It works by simply letting the queue fill up, and then dropping each incoming packet until the queue has space. [33]

The algorithm is unfair and dumb, not giving any care to any priority levels, since it drops every incoming packet. Not only that, it is also unfairly biased for packet-bursts. The bursts will quickly fill up the queue and occupying the place, thus dropping other packets, without losing much packets themselves.

Another major issue that Tail drop can create is TCP Global Synchronization (TGS). TGS can occur when multiple TCP connections has their packets dropped by tail drop. In such cases, every connection will reduce their transmission rate to battle the congestion at the same rate, which leads to unused network resources due to reduced transmission rates from the TCP hosts.

In response to the reduced rate, the TCP hosts will start ramping up their transmission rate at the same time, which then leads to congestions. Once again, they will reduce the rate in order to battle the congestion, and now a cycle of unused resources/network overflow will follow. [34]

Tail drop implements no QoS and cannot be used in conjunction with PHB. Instead RED is used, which gives some forms of QoS to the queue system. RED also helps with avoidance of TGS. This is why RED is used instead of Tail drop for AF. [35]

RED works much differently than Tail drop. RED is based around a calculation of the average queue size with a minimum and maximum threshold for that average value. With each incoming packet it calculates the queue average size and compares it to those two thresholds. If the average is between these thresholds, packets are marked with a drop probability, that increases with the amount of bandwidth the connection has, in other words, the more packets a connection has inside the queue, the higher is the probability of it getting dropped. The packets are then dropped with the calculated probability.

In cases where the average is either lower or higher than the thresholds, no probability is used. If the average is lower than minimum, nothing happens. And if the average is larger than maximum, all packets are dropped, which is similar to Tail drop.

A simple RED algorithm based on Figure 1 in "Random Early Detection (RED) gateways for Congestion Avoidance" . The following algorithm executes on every incoming packet.

```
1
2     Average queue size calculation: avg_size
3     if( max_threshhold ⩾ avg_size > min_threshold)
4     {
5          Calculate drop probability p
6          Drop with probability of p
7     }
8     else if( avg_size > max_threshold )
9          Drop
10
```

Listing 1: RED Algorithm

Where the Drop probability ranges from 0 to 1 and is calculated:

$$P_1 = \frac{avgSize - minThreshold}{maxThreshold - minThreshold} \tag{2}$$

$$p = \frac{P_1}{1 - count \times P_1} \tag{3}$$

Where count is the number of packets received since last packet dropped.

4. `Class selector`

   Since the DS field replaces the old TOS field, the DiffServ still needs backwards compatibility with networks that use the TOS fields instead. That way, networks with DiffServ can work together with networks that still is using TOS. Class selector PHB (CS PHB) is defined to enable that backwards compatibility,

Since TOS only uses 3-bits, the CS only uses the three first bits in the 6-bit DSCP field to match. The CS values goes from 0 to 7, also again to match the priorities of TOS. For the Diffserv to identify that the DSCP value it reads are from an TOS network, the three last bits in the DSCP field are always 0. In short, if the three last bits in the 6-bit fields is 000, then CS is applied. If the three last bits are greater than zero, DF, EF or AF is applied. [36]

Table 3: Class selector values

DSCP Value (Decimal)	**Equivalent TOS priority**
000 000 (0)	Best-Effort (0)
001 000 (8)	Priority (1)
010 000 (16)	Immediate (2)
011 000 (24)	Flash (3)
100 000 (32)	Flash Override (4)
101 000 (40)	Critical (5)
110 000 (48)	Internet (6)
111 000 (56)	Network (7)

As mentioned, not all DSCP values are commonly used. These are the following commonly used DSCP values in networks. [28]

Table 4: Common DSCP values

DSCP Value (Decimal)	**PHB**	**Droprate**	**Precedence value**
101 110 (46)	EF	-	101 (Critical)
000 000 (0)	Best Effort	-	000 (Routine)
001 010 (10)	AF11	Low	001 (Priority)
001 100 (12)	AF12	Medium	001 (Priority)
001 110 (14)	AF13	High	001 (Priority)
010 010 (18)	AF21	Low	010 (Immediate)
010 100 (20)	AF22	Medium	010 (Immediate)
010 110 (22)	AF23	High	010 (Immediate)
011 010 (26)	AF31	Low	011 (Flash)
011 100 (28)	AF32	Medium	011 (Flash)
011 110 (30)	AF33	High	011 (Flash)
100 010 (34)	AF41	Low	100 (Flash Override)
100 100 (36)	AF42	Medium	100 (Flash Override)
100 110 (38)	AF43	High	100 (Flash Override)

3.3 DSCP REMARKING

With the knowledge of how to enable QoS in SDN, the final piece is how to implement it. OF provides support in how to mark packets with different DSCP values of choice.

3.3.1 *Metering*

OF version 1.3 introduces the concept of metering, which allows SDN operators to pass flows into specified meters that perform actions upon those flows. [37] These actions enables the implementation of QoS in SDN. Actions such as DSCP remarking based on flow rate is enabled with metering in OF 1.3

The concept of metering is divided into two parts, Meter Tables and Meter Bands. [37]

Meter Table
A meter table is built up by multiple meter entries, each meter entry is attached to a unique flow. Thus, giving the ability to impose different operations to each flow. With the ability to control each unique flow, QoS can be implemented. With meters, DSCP values can be changed according to flow rate, resulting in that a DiffServ domain can be created.

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

Since the meter entries are attached to each flow, it can distinguish flows from the same ports, meaning, it's not limited by the number of ports to perform QoS operations, but can rather perform these operations based on the number of flows. This results in a more complex system, since it does not group up every flow that belongs to the same port.

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

A meter entry consists of three components

Meter Identifier
The Meter ID is a 32-bit unsigned identifier which is used by the flows to identify which meter entry it belongs to.

Counter
A regular counter that keeps count of the number of packets that has been processed by the meter. The counter is updated for each packet.

Meter Bands
It is the meter that measures the rate of each incoming attached flow, but it is the Meter bands that holds the instructions and executes the operations based on the measured rate of the flows. Each Meter band contains instructions on what to do when a flow reaches a set rate. The meter band applies actions when the flow-rate is greater than the set rate of the meter. [37]

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

A Meter band primarily has two different actions that it can perform when a flow exceeds the set rate of the band. The two actions are the following:

Drop
Order the Meter band to drop every packet if the current flow rate exceeds the Meter bands rate.

DSCP Remark
This action allows the Meter band to increase the droprate by modifying the 6-bit DSCP field. If we are following the AF standard, this can be done by increasing the DSCP value while still in the same class. With DSCP remarking applied based on current flow rate, a DiffServ domain could be created.

To sum it up, a Meter measures the rate and passes that information to multiple Meter bands that apply actions based on that rate. A meter entry is attached to a flow that directs the flow into a defined Meter. Every meter entry makes up the Meter table.

With the ability to shift DSCP values based on flow rates, higher priority can be given to flows within given rates and lower the flows priority if they exceed it. In that way, bandwidth guarantee can be achieved. And since lower priority traffic is only dropped in cases of congestion, excess traffic is allowed when enough resources are available.

3.4 PROOF OF CONCEPT

To prove the methods and theories, real-time tests are constructed. A Proof of Concept (PoC) has been established to provide evidence that SDN can achieve bandwidth guarantees through QoS with the help of DSCP and Meters.

Since virtual switches are used, the only way to achieve a proof of concept is through emulation. An emulation network environment with hosts and SDN-compatible switches fulfills the requirements to perform different tests to prove the theories and methods. The base requirements for the network environment is that it should provide extensive support for SDN-switches and is OF compatible. This is to enable the usage of SDN controllers which is needed to provide DSCP remarking with Meters.

A tool for such purposes is a networking emulation software named Mininet.

3.5 MININET

Mininet is an open-source network emulator with heavy focus on including OF and SDN-controllers with OF support[38] .Thus, Mininet is a tool for emulating SDN networks. Mininet runs on a single Linux kernel and uses Python for its API, it provides a complete networking experience, it provides everything from end-hosts to switches and routers. This results in a tool that can be used to emulate a complete network setup. [39]

Quoting Mininet, *"In short, Mininet's virtual hosts, switches, links, and controllers are the real thing - they are just created using software rather than hardware"* [39]
Thanks to this, an emulated SDN network that behaves similar to a real SDN network can be created, and should be more than enough to create a proof of concept.

Although Mininet allows the creation of a customized network of choice, it does impose some limitations compared to a real physical network. Since everything Mininet emulates shares the same computer resources, thus resulting in a usually slower experience than the physical network can deliver, and it's not suited for experiments ranging around 10gbps. For such cases, physical networks usually give a better experience. [40] As a trade off for performance, emulated software has the advantage of customization, it can create topologies based on the users choice, instead of being limited for example by the number of switches bought or physical space.

Mininet will be used to establish the PoC and provide evidence that DSCP remarking with meters can be used to acheive QoS. A more customizable network can enable more test cases than a high-speed network.

In terms of OS, Mininet requires Linux to run, and Ubuntu is the recommended distribution to use. Using a Virtual Machine (VM) is recommended for the use of Mininet, thus running Mininet inside an OS that is provided from the VM. [41] . With a VM, we can virtualize an OS of choice that provides best compatibility with Mininet.

A VM provides benefits such as changing OS and/or OS version with ease and modifying said OS however you want, without impacting the host computer. These benefits are beneficial features for executing different experiments.

3.6 ORACLE VM VITRUABOX (VBOX)

The VM of choice is VBox. The choice is based on the recommendation for choice of VM from Mininet themselves [41]. Other factors that were taken into consideration were that VBox is completely free. In comparison to VMware, another popular VM that only has a free trial, and later requires a subscription to keep on using its services. [42] .

A VM is an emulator. An emulator can emulate different systems. Emulation is the ability for a computer system to imitate another computer system. With this, the computer can run programs and tools designed for other systems. [43]

What this means is that any OS can be used in order to use Mininet. The VM can emulate any OS for Mininet with no restrictions on what OS the computer originally runs on.

Many VMs such as Vbox creates a separate sandboxed environment for the emulated OS to run on. A sandboxed environment mean that the emulated OS cannot infringe on the Host OS and it enables the user to use multiple OS at the same time, in different windows.

3.6.1 *SDN Hub*

SDN Hub provides a preconfigured OS with Mininet already installed and several other tools and controllers that can be useful. The OS used by SDNHub is Ubuntu which lines up with the recommendations from Mininet. In addition to this, several tools for SDN developing is included which provides ease-of-use to the user that does not have to install these tools themselves.

List of tools taken from Sdnhub [44]

- SDN Controllers: OpenDaylight, ONOS, Ryu, Floodlight, Floodlight-OF1.3, POX, and Trema
- Example code for a hub, L2 learning switch, traffic tap, and other applications
- Open vSwitch 2.3.0 with support for OF 1.2, 1.3 and 1.4
- Mininet
- Pyretic
- Wireshark 1.12.1
- JDK 1.8, Eclipse Luna, and Maven 3.3.3

Although all tools won't be used or needed, a VM with Mininet installed and a couple of useful tools has a huge ease-of-use advantage over creating a fresh VM with Ubuntu and manually installing required tools. All benefits results in the choice of SDN Hubs preconfigured VM over creating an new fresh OS from the ground up.

SDN Hubs VM provides the virtual SDN switch, Open vSwitch for usage with Mininet. Open vSwitch (OVS) is arguably the most used virtual SDN switch. That said, as of now, OVS do support OF 1.3 , but it does not have any support for Meters. Meters is a critical function in how QoS and Bandwidth guarantee will be achieved. Support for Meters are planned in release 2.8, for which no release date is set. [45] Therefore, another virtual SDN switch has to be used.

3.7 OF SOFTWARE SWITCH (OFSOFTSWITCH)

OfSoftswitch is a virtual SDN switch that supports less OF versions than OVS, but is said to have support for the Meters that OVS lack. Although it is less supported, OF 1.3 and Meters is all that is required to achieve QoS. [46]

OFSoftSwitch recommends to use Mininet as the emulated network environment of choice, which suits well since Mininet was already chosen. [47]

3.8 CHOICE OF CONTROLLER

With all tools that is needed for a PoC, a controller should be selected to manage it. There are multiple factors that should be taken into consideration when picking a controller of choice. From the project objective and the PoC, a couple of requirements for the controller can be extracted.

To achieve QoS, DSCP will be used, and in turn, Meters will implement DSCP remarking. For Meters to be enabled, the switches need to support OF 1.3. Thus, the controller also have to support OF 1.3, or else it won't be compatible with each other.

Since it is a PoC that is to be established, a controller suited for quick deployment and prototypes would suit the purpose best. There is no need for a complex, heavy controller targeted for use in large scale deployments.

According to *Feature-based Comparison and Selection of Software Defined Networking (SDN) Controllers* [6], Ryu is the best controller for OF support. Also, according to *Advanced Study of SDN/OF controllers* [7], Ryu is suitable for quick prototyping.

From these two studies, Ryu seems like the ideal choice to set up a PoC to provide evidence.

With every tool needed acquired, a PoC is established and the theories and methods will be tested.

4 POC TESTS

With the PoC established, tests can now be performed in an environment that behaves like a physical SDN network. With this, it can be further researched if SDN can implement QoS.

To perform these tests, a test topology is needed. It is a simple topology with the bare requirements to test QoS in SDN networks. By that, it means that it must be able to simulate congestion to see that different priority levels are working as intended.

4.1 SETUP

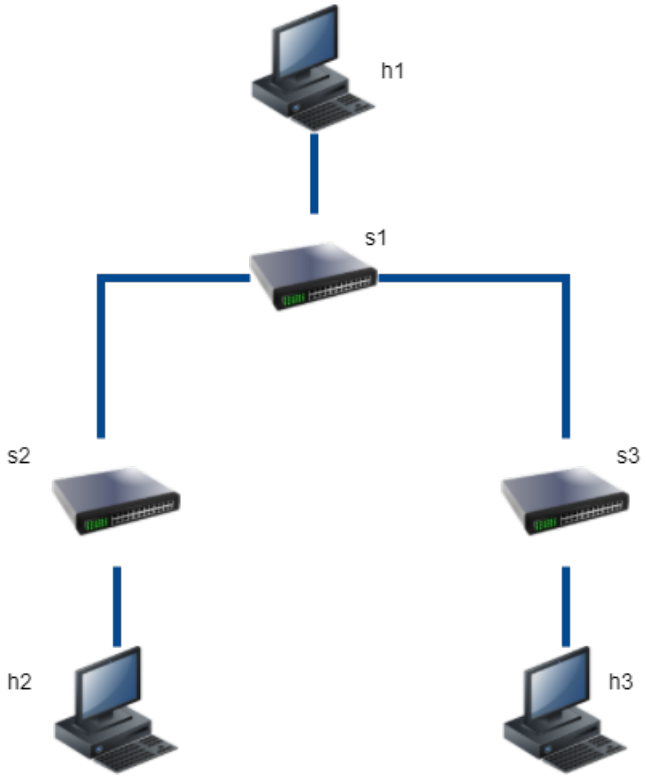

Figure 2: Simple Topology for congestion tests

Figure 2 is a simple topology with the bare minimum that allows congestion when sending traffic to Host1. The topology consists of Three switches and Three hosts. Switch2 and Switch3 will only be used for DSCP remarking, and Switch1 will be used to sort flows into queues depending on their DSCP value.

All hosts will have to pass through a DSCP remarking switch before they can enter a queue sorting switch. The remarking and queue sorting functions needs to be separated into different switches due to OFSoftswitchs poor support of OpenFlows multi-table pipeline processing. [48]

Thus, the tests are limited to sending traffic from Host2 and/or Host3 to Host1. The tests will send traffic from both Host2 and Host3 with goal to cause congestion in Switch1. The traffic sent will have different priorities and use Meters to further impose QoS. When Congestion occurs, the switch should properly prioritize the traffic and results will be shown if QoS is working.

A program, simpleSwitch13, that is provided by Ryu is used to program the switches to behave like a simple regular switch with OpenFlow support.

None of the switches have implemented DiffServs recommended PHB, so normal priority queues has to be used in conjunction with DSCP and Meters.

To perform these tests, a bandwidth testing tool will be used. A tool suited for quick tests in Linux is Iperf, which also came included with Mininet, making it a suitable tool.

4.2 IPERF

Iperf[49] is a tool for measuring bandwidth on IP networks. When testing the system, Iperf will be used to detect the bandwidth of the incoming flows to Host1. Iperf will create UDP streams to the target Host1, from client Host2 and Host3. The UDP streams will be sent with different bandwidth and with different DSCP values so it can be used to provide evidence that the QoS is working as intended. UDP is the protocol of choice since it's more simple than TCP and only sends packets with no follow up.

Iperf will then provide details on the performed UDP stream. Details such as bandwidth and packets transmitted is provided, thus giving information if the expected bandwidth from a client was achieved.

4.3 TEST SETUP

Three different tests with different QoS implementations are performed. In each test, multiple iterations of the same test is necessary to eliminate flukes.

In cases of congestion within the same priority queue, First in - First out (FiFo) queuing mechanism is used. From that, a simulation program (A) has been developed to calculate expected bandwidth. From that program, a mathematical model has been extracted. The purpose of the simulation program and the mathematical model is to gain values to compare the test results against.

The simulation program simulates when traffic of a given rate exceeds the link rate, it simulates congestion for the test cases in this project. The simulator expects that traffic from both hosts arrive at the same time, and therefore, the selection of packets that gets through is random. The simulation is based on a probability that the next packet processed will be from a specific host. By simulating multiple times, an average throughput from each host will be presented, and gives a fair value of what to expect from the real system tests.

For example, Host2 sends 800kbit/s and Host3 sends 500kbit/s into a limit of 1000kbit/s. To simulate the average kbit/s for Host2, the program will take the probability that the next packet processed will be from Host2, which should be larger than Host3. Since 800kbit/s of 1300 kbit/s is from Host2 and is a larger fraction of the total throughput than that of Host3, Host2 should have higher probability to successfully transmit packets. When a packet is put through, it is removed from the pipeline, and the probability for a packet from Host2 is now a little bit lower. For simplicity and to match the system test, all bandwidth outside the accepted 1000kbit/s link is dropped.

The following mathematical model is used for bandwidth calculation in cases of congestion.

$$R = L \times \frac{r_k}{\sum_{k=1}^{n} r_k} \quad ; \quad \sum_{k=1}^{n} r_k > \text{Link} \tag{4}$$

Where R is the calculated Rate, and L is current remaining bandwitdh of the Link that they have to share. r is the bandwidth that the host tries to send.

Each and every test researches how SDN behaves with different degrees of QoS in congestion, and how well QoS can be implemented as of today.

For consistency purposes, the same amount of bandwidth will be transmitted in each test, and will together exceed the link of 1Mbp/s to cause congestion. 800kbit/s from Host 2 and 500kbit/s from Host 3. This way, it should give a better picture on how QoS interacts with the network. Also, to eliminate flukes, each test is performed 100 times,(100 seconds) and given a average kbit/s value from each host.

5 RESULT

5.1 SYSTEM TESTS

Test 1 : No QoS
The first test provides insight in how the traffic behaves with no QoS applied.

Table 5: No QoS

Host	Host2	Host3
Transmitted Bandwidth [kbit/s]	800	500
Calculated Result [kbit/s]	615.4	384.6
Simulated Result [kbit/s]	616	383

Since no QoS is involved, the FiFo simulator and Calculation can be used. The result of the test will show how real world usage compares to the simulated and calculated results. The results are calculated with equation (4)

Test 2 : Priority Queues
The second test will implement QoS through Priority queuing. It uses a simple QoS model which puts AF12 class into a high priority queue, and rest is Best Effort.

Table 6: Priority Queues

Host	Host2	Host3
Transmitted Bandwidth [kbit/s]	800/AF11	500/BE
Calculated Result [kbit/s]	-	-
Simulated Result [kbit/s]	-	-
Expected Result [kbit/s]	800	200

Since Host2 has high priority, it should transmit all 800Kbit/s, and Host3 will have to use what's left, 200kbit/s in this case. The mathematical model and simulation does not take QoS into account.

Test 3 : Priority Queues + DSCP remarking with Meters
In the third test, both priority queues and Meters are used. The test is divided into two sections, where the first section uses DSCP remarking. It tries to achieve the requirements of High QoS based on the methods, and thus achieve bandwidth guarantee. The other section of the test uses Drop to research how it impacts the network.

DSCP Remark
Two tests will be performed, one with a high bandwidth guarantee for Host2, and one with a low bandwidth guarantee for Host2.

Host2 will transmit 800kbit/s with a guarantee of 600kbit/s (AF11). Host3 will transmit 500kbit/s of lower priority traffic (AF12) which will compete for resources with the excess traffic that Host2 sends.

Table 7: High bandwidth guarantee test

Host	Host2	Host3
Transmitted Bandwidth [kbit/s]	800/AF11	500/AF12
Calculated Excess Result [kbit/s]	114.3	285.7
Simulated Excess Result [kbit/s]	114	285
Guarantee [kbit/s]	600	0
Expected Result [kbit/s]	714	285

It is expected that Host1 will keep at least 600kbit/s. Then the excess traffic can be calculated with the model and simulated. The excess traffic is calculated with equation (4). Where L is the remaning link bandwidth = 400 kbit/s, and r is divided by the sum of the excess traffic = 700kbit/s.

The second test for DSCP remarking explores the impact when Host1 gets low bandwidth guarantee, in contrast to high guarantee of the first test. Host2 and Host3 will transmit the same amount of bandwidth, but now Host2 will have a guarantee of 200kbit/s instead of 600kbit/s.

Table 8: Low bandwidth guarantee test

Host	Host2	Host3
Transmitted Bandwidth [kbit/s]	800/AF11	500/AF12
Calculated Excess Result [kbit/s]	436.4	363.6
Simulated Excess Result [kbit/s]	436	363
Guarantee [kbit/s]	200	0
Expected Result [kbit/s]	636	363

Again, it is expected that Host2 can at least manage 200kbit/s which is its guarantee. Excess traffic will be calculated and simulated. Again, equation (4) is used to calculate the excess traffic.

Drop

For the sake of QoS research in SDN, it is interesting to see how drop impacts the network. Two tests will be performed. The first test will test drop on one stream and the second test will test drop on both streams.

Table 9: One Host drop

Host	Host2	Host3
Transmitted Bandwidth [kbit/s]	800	500
Calculated Excess Result [kbit/s]	-	-
Simulated Excess Result [kbit/s]	-	-
Drop level [kbit/s]	200	-
Expected Result [kbit/s]	200	500

Host2 has a Meter with a Meter band that executes drop if it exceeds 200kbit/s. Host3 will then be free to use all of his transmitted bandwidth.

Table 10: Two Host drop

Host	Host2	Host3
Transmitted Bandwidth [kbit/s]	800	500
Calculated Result [kbit/s]	-	-
Simulated Result [kbit/s]	-	-
Drop level [kbit/s]	200	200
Expected Result [kbit/s]	200	200

Both hosts has a Meter band at 200kbit/s and should be kept within that limit.

5.2 IPERF RESULTS

5.2.1 *Test 1: No QoS*

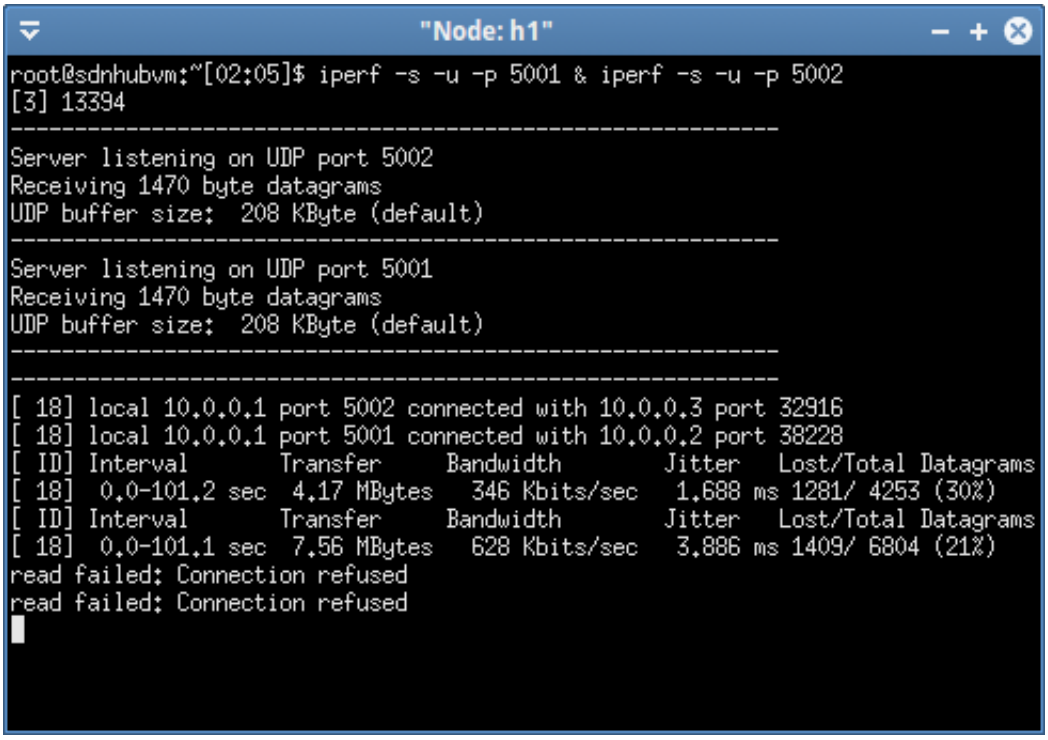

Figure 3: Results from Test1

The test result shows that Host2 transmits 628Kbit/s, which is in line with the expected values. Host 3 transmits 346 Kbits/sec, which is not far from the simulated and calculated values. The outcome of test1 behaves as expected.

5.2.2 *Test 2: Priority Queues*

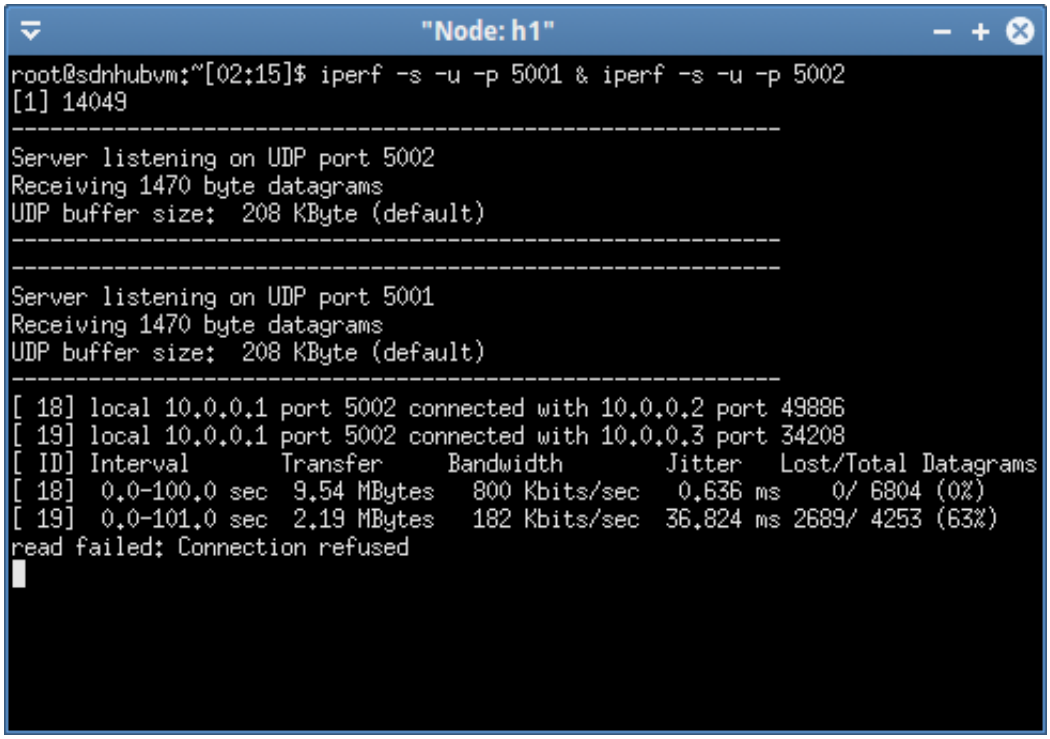

Figure 4: Results from Test2

As expected, Host2 with the high priority queue can transmit all its bandwidth without any interruption from the lesser prioritized Host3. Host3 has to use around 200kbit/s which is left over. The results are in line with the expected results.

5.2.3 *Test 3: Priority Queues with Meters*

DSCP remarking

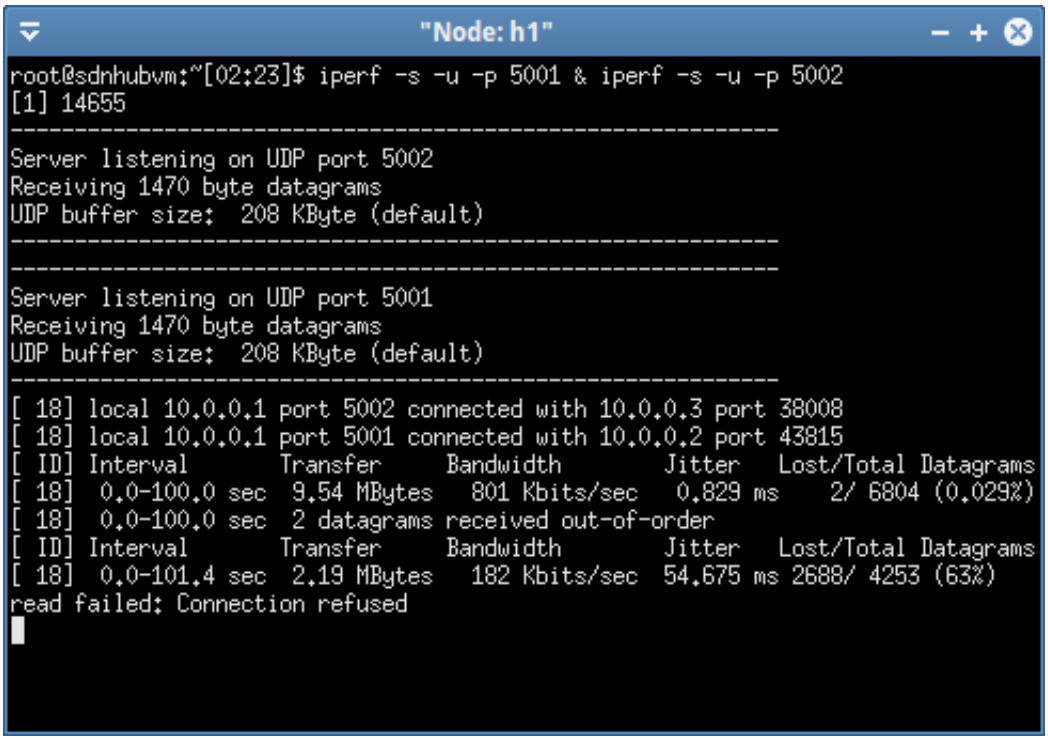

Figure 5: Result from Test3 with guarantee at 600kbit/s

The results of the tests are not in line with the calculated, simulated or theoretical results. From an average of 100 tests (100 seconds), the margin is too large to be in line with the expected results.

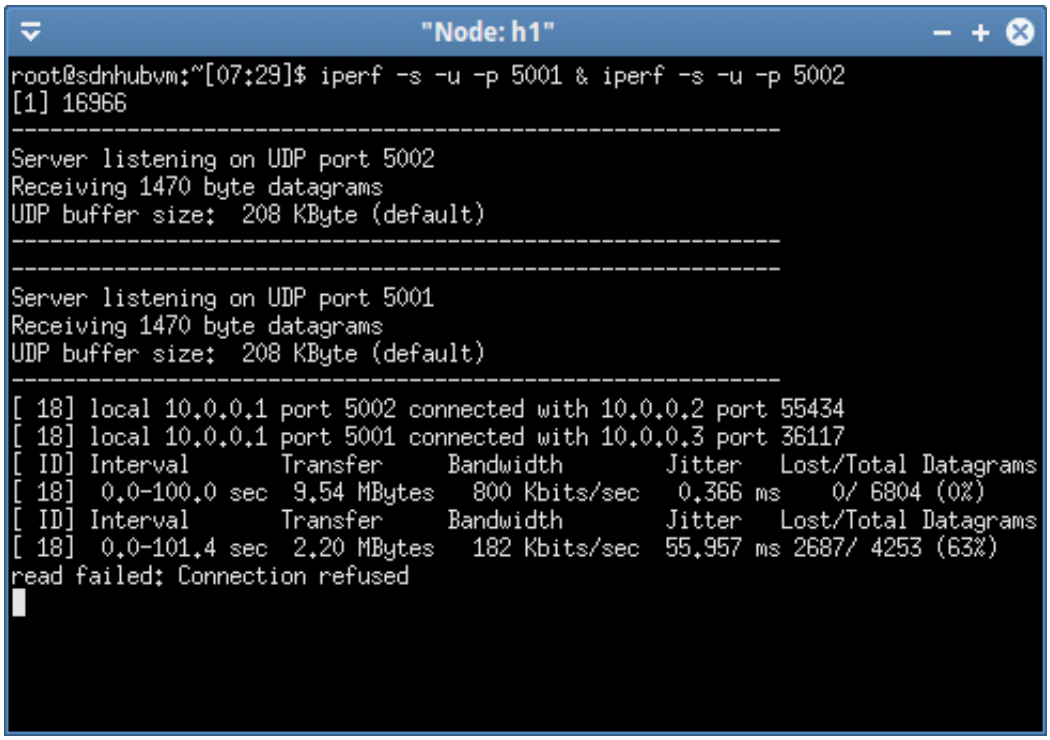

Figure 6: Result from Test3 with guarantee at 200kbit/s

The results of the tests are not in line with the calculated, simulated or theoretical results. From an average of 100 tests (100 seconds), the margin is too large to be in line with the expected results.

Drop

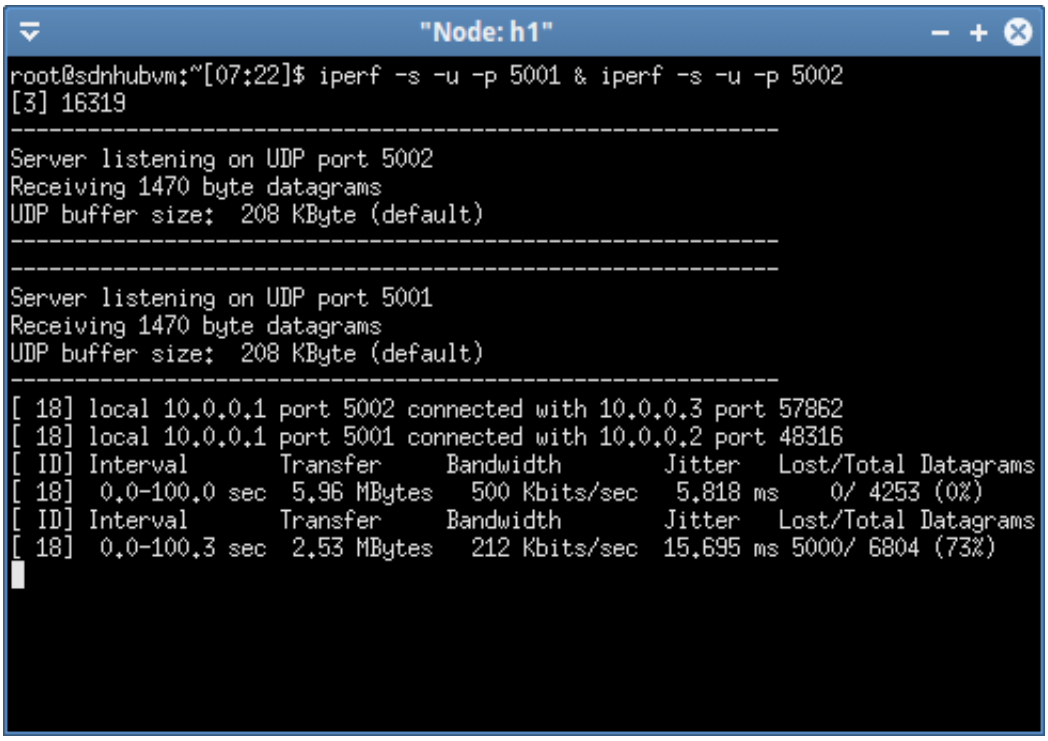

Figure 7: Result from Test3 with one Host drop at 200kbit/s

Host2 that has the Meter band with drop at 200kbit/s is kept around 200kbit/s, which is in line with the expected results. Also, Host3 kan fully utilise the network and transmit his bandwidth without interruption.

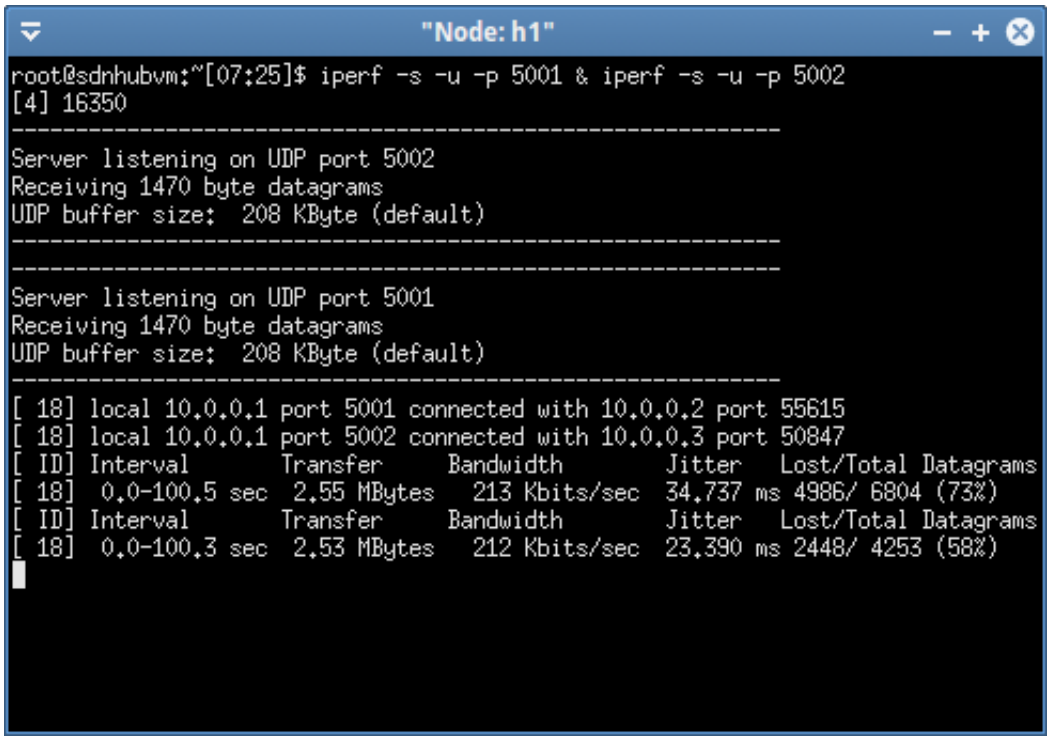

Figure 8: Result from Test3 with both Host drop at 200kbit/s

Both hosts are kept around 200kbit/s, which is in line with the Meter bands and in line with the expected results.

6 DISCUSSION

6.1 RESULTS

Most of the results from our tests were in line with what we expected. The most surprising result were Test3 with DSCP remarking, these test results were very similar to Test2. All three tests gave full 800kbit/s to Host1 even though the traffic exceeded the given limits in Test3.

Since Test3 is basically a Test2 with DSCP remarking and the results were the same anyway, we concluded that DSCP remarking did not work. To further investigate, we tested with Drop function, and we saw that the bandwidth measure feature did indeed work, eliminating Meters as the cause of fault in Test3. With test confirming that Meters and Meter bands with drop are working. The only remaining factor was DSCP remarking as the reason why the tests failed.

Sadly, we needed DSCP remarking for High QoS to work properly and thus, from these tests, we could conclude that we can't implement High QoS in SDN as of today.

Even without DSCP remarking and High QoS, we could still achieve QoS in SDN. Through comparing Test1 and Test2, we could see that QoS was achieved and how it impacts the network.

With the drop function, a maxlimit QoS could be established in a network. This could be useful in networks. If you configure a maxlimit for each flow, it results in a form of bandwidth guarantee, since if nobody can exceed their given maxrate, nobody will impose on your given rate. It can also be useful in Best Effort networks to prevent hosts from flooding the network and causing congestion.

6.1.1 *Comparison to related research*

Even though we could not achieve High QoS in our tests, our test results still provide proof for QoS in SDN. But how do our results compare to other researches in the same field?

Our Test2 is very similar to the approach made in Implementing Quality of Service for the Software Defined Networking Enabled Future Internet.[21] In that research, QoS is achieved by sorting flows into either high priority traffic or regular traffic. This is done by reading if the flows has DSCP enabled or not. In our Test2, we separate flows based on DSCP, and sort them into either a high priority queue, or send traffic as Best Effort.

Our Test3 with drop function proves not only that Meters are working, but also that we can implement a maxlimit QoS. In Enhancing ns-3 with Openflow 1.3 support[48] they also perform tests to prove that Meter with drop enabled can be used for flow ratelimiting. Our result in Test3 with drop is amplified by the similar result in Enhancing ns-3 with Openflow 1.3 support. Meaning that their result strengthens the authenticity of our result.

6.1.2 *A society's perspective*

How can our work affect the surroundings from a society's perspective? Our work is focused around QoS, so it is interesting to discuss how QoS can make an impact. QoS is good from a security point of view. With QoS you can regulate the traffic so that nobody can misbehave.

By using a maxlimiter QoS you can prevent attacks such as Distributed Denial of Service (DDoS) by not allowing any traffic to overflow the network. DDoS works by trying to overflow systems by making a lot of requests to said system. DDos can be used to bring down critical functions in the society. With a maxlimter QoS, certain flows or paths may be restricted in how much data they can use, thus reducing the risk of somebody causing overflow in the network.

Methods such as those used in our Test3 with drop can be used to establish a maxlimiter QoS, flows that try to break their limit simply gets their packets dropped, and thanks to that, attacks such as DDoS are suppressed.

In addition, by limiting the rate of flows, it can give guarantees to critical traffic that attackers might try to disrupt. In conclusion, QoS can be used to increase the security in networks by automatically putting restraints on flows.

6.2 THE FIELD OF SDN

We were given the task of researching QoS in the field of SDN. Since SDN is directly programmable it could be used to solve the problems described in the problem description. SDN is a field that is far from fully developed and thus, lacking a lot of in-depth support. This means that most articles and guides about SDN are very shallow and the in-depth ones are sparse and niche.

All this resulted in that we could not find much documentation that could support our research. In contrast to this, most research papers were too in-depth for our bachelor thesis and were often very specialized and thus could not provide much help for our project objective.

6.3 METERS

To achieve the features in the project objective, we chose Meters to be used in our implementation. A big problem with Meters was that it was very poorly supported by the most used virtual switches. Meters are a part of OF 1.3 which is broadly used in SDN. As mentioned, SDN lacked in many regards, and since version 1.3 of OF is such a small part of whole SDN. The support and documentation of 1.3 with Meters was lacking even more.

This gave us very little choice in using tools that actually claimed they supported Meters, although none of them actually had full implementation of DSCP remark for those meters.

6.4 PHYSICAL SWITCH APPROACH

Another way to provide an PoC with practical evidence of the methods would be to use physical switches with SDN and OF enabled. This approach was examined at the start. Most switches with OF support are expensive enterprise switches designed for datacenters, not suitable for our prototype and experimental work.

We found OF compatible switches by the name of Zodiac FX, which claimed to be the first OF switch for private usage at home, therefor designed for home usage and not for enterprise deployment. A couple of those switches were bought and the company claimed that they had full OF version 1.3 support with Meters fully supported.

After some investigation, we found in their source code that no support for DSCP remarking were implemented at all. We contacted them about this, and they promised support and patches that never came. In addition to this, they claimed that their switches could replace your normal switches in the house. After a fair amount of testing, we discovered that the switch could not perform at high speeds, which was claimed. We also saw that the switch had problems with simply holding the connection to the controller even at low speeds. We have spent a lot of time on these switches that could not deliver what was promised.

So after this approach, we had no way out but to try the virtual switch approach.

6.5 VIRTUAL SWITCH APPROACH

So, what was left to us was the virtual switch approach. As mentioned, the support for version 1.3 and Meters were very sparse and basically the only one that was recommended by other papers was OFsoftswitch, and it seemed to have the best support for our objective. But as we progressed with our tests, even that switch failed to deliver complete support for DSCP remarking.

When configuring OFsoftswitch, we also ran into a lot of problems with dependencies. Every guide used said different things and it turned out that no guide actually worked. We made an own mix of the guides to even get the switch to run, and then we noticed that meters were not fully implemented.

OVS would be much easier to configure since its widely more supported, but it does not even have any Meters implemented as of now, even though they have greater than OF version 1.3 support.

All in all, all these problems made us focus more on support problems than we would have wanted. A lot of time had to be used in other places than in the actual report. It resulted in that it took much more time than was actually needed for the research.

7 CONCLUSION

From Test1 and Test2, it can be seen that QoS is implemented into the SDN network. Both tests are using the same bandwidth parameters but gain different results, in Test2 it provides a lot more bandwidth to Host2 over Host3 in comparison to the result from Test1. This shows that Host2 has a higher priority than Host3 and thus, QoS is achieved.

The results from the DSCP remark + priority queues tests in Test3 were not in line with the expected results. The results are very similar to the results of Test2, which only utilizes priority queues. The results from the Drop tests were in line with the expected results. The drop tests prove that the bandwidth measuring feature of Meters are working and that the Meter bands are using drop correctly. With the drop function, a QoS with a maxlimit for each flow is enabled.

From the very similar results of Test2 and Test3 with DSCP remark, and from the results of the drop tests, it is concluded that DSCP remarking does not work as intended. Proof is given that Meters work, and the DSCP remark + priority queues behave like Test2 with priority queues. Thus, giving proof that the DSCP remarking function is not fully implemented.

To sum it up, SDN is able to implement QoS but not High QoS due to lack of implementation of DSCP remarking.

7.1 FUTURE WORK

From the results and conclusion of this project, some future work comes to mind. Since we lacked the evidence from practical proof to support our methods, future work could include the development of a switch with full support for DSCP remarking with Meters in OF 1.3. Two ways are presented:

- Further development of existing switches to implement full support of ver 1.3. Taking an existing switch and adding support for DSCP remarking is probably the quickest way to achieve a switch with multiple other OF functions.

- Developing a brand new switch designed for the purpose of DSCP remarking with Meters. Developing and implementing the minimum requirements for an OF 1.3 switch with DSCP remarking support, thus making it a very specialized switch with only a few OF functions. It will probably require a lot of time and expertise.

BIBLIOGRAPHY

[1] Cisco. The zettabyte era: Trends and analysis. Technical report, Cisco public, 06 2017.

[2] Sezer S, Scott-Hayward S, Chouhan P, Fraser B, Lake D, Finnegan J, Viljoen N, Miller M, and Rao N. Are we ready for sdn? implementation challenges for software-defined networks. *IEEE Communications Magazine*, 51(7):36–43, 2013.

[3] Open Networking Foundation. Software-defined networking (sdn) definition, 03 2017. URL https://www.opennetworking.org/sdn-definition/.

[4] Nadeau T and Gray K. *SDN: Software Defined Networks: An Authoritative Review of Network Programmability Technologies*. O'Reilly Media, 1 edition, 09 2013.

[5] Networkel. Osi model : 7 layer of the network communications, 09 2017. URL https://networkel.com/osi-model-7-layer-network-communication/.

[6] Khondoker R, Zaalouk A, Marx R, and Bayarou K. Feature-based comparison and selection of software defined networking (sdn) controllers. *Computer Applications and Information Systems (WCCAIS), 2014 World Congress*, 01 2014.

[7] Zimarina D Pashkov V Smeliansky R Shalimov A, Zuikov D. Advanced study of sdn/openflow controllers. *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, (1), 10 2013.

[8] Nox network control platform, 03 2017. URL https://github.com/noxrepo/nox.

[9] Pox controller tutorial, 03 2017. URL http://sdnhub.org/tutorials/pox/.

[10] Beacon, 03 2017. URL https://openflow.stanford.edu/display/Beacon/Home.

[11] Floodlight, 03 2017. URL http://www.projectfloodlight.org/floodlight/.

[12] Openmul controller, 03 2017. URL http://www.openmul.org/openmul-controller.html.

[13] Maestro-platform, 03 2017. URL https://code.google.com/archive/p/maestro-platform/.

[14] Ryu, 03 2017. URL http://osrg.github.io/ryu/.

[15] Opendaylight, 03 2017. URL https://www.opendaylight.org/.

[16] Open Networking Foundation. Openflow - open networking foundation, 03 2017. URL https://www.opennetworking.org/sdn-resources/openflow.

[17] Farhady H, Lee H, and Nakao A. Software-defined networking: A survey. *Computer Networks*, 81:79–95, 2015.

[18] Wang Z. *Internet QoS: Architecture and Mechanisms for Quality of Service*. The Morgan Kaufmann Series in Networking. Morgan Kaufmann, 1 edition, 03 2001.

[19] Inc. Cisco Systems. Quality of service networking, 06 2017. URL http://docwiki.cisco.com/wiki/Quality_of_Service_Networking.

[20] Yan J, Zhang H, Shuai Q, Liu B, and Guo X. Hiqos: An sdn-based multipath qos solution. *China Communications*, 12(5):123–133, 05 2015.

[21] Sharma S, Staessens D, Colle D, Palma D, Goncalves J, Figueiredo R, Morris D, Pickavet M, and Demeester P. Implementing quality of service for the software defined networking enabled future internet. *Software Defined Networks (EWSDN), 2014 Third European Workshop*, 09 2014.

[22] Braden R, Clark D, and Shenker S. *Integrated Services in the Internet Architecture: an Overview*. ISI and MIT and Xerox PARC, 06 1994.

[23] Nichols K, Blake S, Baker F, and Black D. *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*. Cisco Systems and Torrent Networking Technologies and EMC Corporation, 12 1998.

[24] Ed. R Braden, Zhang L, Berson S, Herzog S, and Jamin S. *Resource ReSerVation Protocol (RSVP) –*. ISI and UCLA and IBM Research and Univ. of Michigan, 09 1997.

[25] *Configuration Guide - QoS*. HUAWEI TECHNOLOGIES CO., LTD., 03 2012.

[26] Ramakrishnan K, Floyd S, and Black D. *The Addition of Explicit Congestion Notification (ECN) to IP*. TeraOptic Networks and ACIRI and EMC, 09 2001.

[27] Grossman D. *New Terminology and Clarifications for Diffserv*. Motorola, Inc., 2002.

[28] *Cisco Nexus 1000V Quality of Service Configuration Guide*. Cisco, release 4.2(1)sv1(4) edition, 03 2016.

[29] Baker F, Chan K, and Babiarz J. *Configuration Guidelines for DiffServ Service Classes*. Nortel Networks and Cisco Systems, 2006.

[30] Davie B, Charny A, Benson K, Le Boudec J.Y., Bennett J.C.R, Courtney W, Davari S, Firoiu F, and Stiliadis D. *An Expedited Forwarding PHB (Per-Hop Behavior)*. Cisco Systems, Inc. and Tellabs and EPFL and Motorola and TRW and PMC-Sierra and Nortel Networks and Lucent Technologies, 03 2001.

[31] Heinanen J, Baker F, Weiss W, and Wroclawski J. *Assured Forwarding PHB Group*. Telia Finland and Cisco Systems and Lucent Technologies and MIT LCS, 06 1999.

[32] 6.6:scheduling and policing mechanisms, 08 2017. URL http://netlab.ulusofona.pt/rc/book/6-multimedia/6_06/index.htm.

[33] *Network OS Layer 2 Switching Configuration Guide, 6.0.1a*. Brocade Communications Systems, Inc., 53-1003770-06 edition, 08 2016.

[34] *Cisco IOS Quality of Service Solutions Configuration Guide*. Cisco, release 12.2 edition, 01 2014.

[35] Floyd S and Jacobson V. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, pages 397–413, 08 1993.

[36] Richardson S. The class selector phb and dscp values, 08 2017. URL https://www.ccexpert.us/traffic-shaping-2/the-class-selector-phb-and-dscp-values.html.

[37] *OpenFlow Switch Specification*. Open Networking Foundation, version 1.3.0 edition, 06 2012.

[38] Mininet Team. Openflow controllers, 05 2017. URL https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#controllers.

[39] Mininet Team. What is mininet?, 05 2017. URL https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#what.

[40] Mininet Team. What are mininet's limitations?, 05 2017. URL https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#limits.

[41] Mininet Team. Download/get started with mininet, 05 2017. URL http://mininet.org/download/.

[42] VMware. Vmware learning zone, 08 2017. URL https://www.vmware.com/professional-services/learning-zone.html?src=WWW_US_HP_LearningZone_R2C2_D_NA_StartTrial.

[43] National Library of the Netherlands. What is emulation?, 05 2017. URL https://www.kb.nl/en/organisation/research-expertise/research-on-digitisation-and-digital-preservation/emulation/what-is-emulation.

[44] SDN Hub. All-in-one sdn app development starter vm, 05 2017. URL http://sdnhub.org/tutorials/sdn-tutorial-vm/.

[45] Linux Foundation Collaborative Project. Quality of service (qos), 05 2017. URL http://docs.openvswitch.org/en/latest/faq/qos/.

[46] Openflow 1.3 software switch, 05 2017. URL https://github.com/CPqD/ofsoftswitch13.

[47] Openflow 1.3 tutorial, 05 2017. URL https://github.com/CPqD/ofsoftswitch13/wiki/OpenFlow-1.3-Tutorial.

[48] Jerez Chaves L, Calciolari Garcia I, and Madeira E. Ofswitch13: Enhancing ns-3 with openflow 1.3 support. *WNS3 '16, Proceedings of the Workshop on ns-3*, pages 33–40, 06 2016.

[49] Iperf, 09 2017. URL https://iperf.fr/.

APPENDIX

```
public class Simulation
{
   public static void main(String[] args)
   {
      int kbitavgHost2 = 0;
      int kbitavgHost3 = 0;
      int Host2packetsum = 0;
      int Host3packetsum = 0;
      int simsec = 1000; //Simulated seconds

      for(int sec = 0; sec < simsec ;sec++)
      {
         ArrayList traffic = new ArrayList();
         for(int i = 0; i < 800;i++) //kbit from Host2
         {
            traffic.add("2");//From host2
         }
         for(int i = 0; i < 500;i++) //kbit from Host3
         {
            traffic.add("3"); //From host3
         }

         int Amount = 1000; //1000kbit for link size
         if(traffic.size() < Amount)
         {
            Amount = traffic.size();
         }
         Random randomGen = new Random();
         for(int i = 0; i < Amount; i++)
         {
            int roof = traffic.size()-1;
            int randomInt = 0;

            if((traffic.size()-1) > 0)
            {
               randomInt = randomGen.nextInt(roof);
            }

            if(traffic.get(randomInt).equals("2"))
            {
               Host2packetsum++;
            }
            else
            {
               Host3packetsum++;
            }
            traffic.remove(randomInt);
         }
      }
      kbitavgHost2 = Host2packetsum/simsec;
      kbitavgHost3 = Host3packetsum/simsec;
      System.out.println("Host2 kbit: " + kbitavgHost2);
      System.out.println("Host3 kbit: " + kbitavgHost3);
   }
}
```

Listing 2: Simple Fifo simulation

PO Box 823, SE-301 18 Halmstad
Phone: +35 46 16 71 00
E-mail: registrator@hh.se
www.hh.se