UNIVERSITÀ DI PISA

Advanced Network Architectures and Wireless
Systems
A.A. 2021/2022

SDN-based Traffic Prioritization

Gianluca A. Cometa   Edoardo Ruffoli

Contents

1 Introduction

The objective of this project is to design and develop a Floodlight module that implements a traffic prioritization behaviour inside a network domain. The network should manage traffic flows according to the scenario "A. Traffic Prioritization" of paper [1].

The system shall expose a RESTful interface that allows an user to configure and query the module providing the following functionalities:

- allow an user to register/deregister a new traffic flow, specifying its class and its parameters (e.g., bandwidth);

- allow an user to select the switch (or switches) that will implement the prioritization;

- allow an user to query each switch for the number of packets handled for each class.

This document has the following organization: the second section presents the design of the module, describing the traffic flows and the switches employed, it also show the process of registering a new QoS traffic flow and the management information of the REST interface; the third section shows the software implementation of the module; the fourth reports the testing results in different scenarios of the system; the fifth and final section presents the simulation menu provided for the tests.
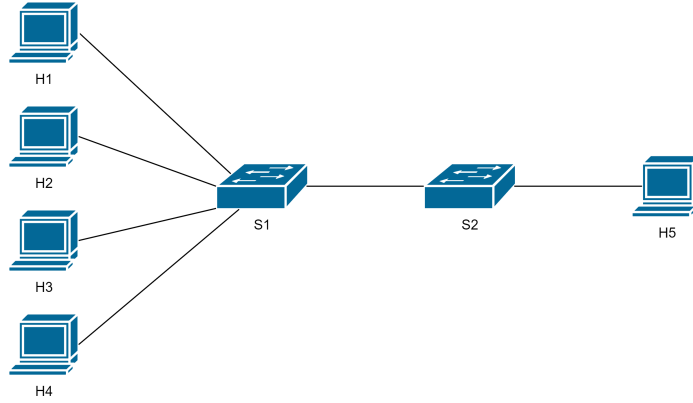

Figure 1: Reference topology

2 Design

2.1 Traffic Flows

In the scenario "A. Traffic Prioritization" of paper [1], the authors presented a traffic prioritization model that allows for two types of traffic flows:

1. **QoS** traffic flows, that have a minimum guaranteed bandwidth;

2. **Best-Effort** traffic flows, that does not have any guarantees in terms of bandwidth.

QoS traffic flows are registered by the user while Best effort traffic flows are all the traffic flows that have not been registered. QoS flows, however, are allowed to send more traffic than their guaranteed rate and, regarding this, they can be categorized in:

- **Conformant** QoS traffic flows, that stay within their guaranteed rates;

- **Non-conformant** QoS traffic flows that exceed their guaranteed rate.

In the model presented, in case of congestion, the switch should drop the non-conformant packets before dropping best-effort packets or, in other words, the non-conformant traffic has a lower priority than the best-effort one. In order to realize this type of traffic prioritization, we need two Open-Flow features: **meters** and **queues**.

2.2 Meters and Queues

Meters guarantee only maximum bandwidth for **ingress** traffic and are applied per flow. If the maximum bandwidth limit is reached, then the switch can either drop the excess traffic (traffic policing) or increase the drop precedence of the DSCP field of the packet's IP header. OpenFlow allows to create, edit an delete meters.

Queues guarantee minimum and maximum bandwidth for **egress** traffic and are applied per port. Apply both traffic policing and traffic shaping, by dropping traffic exceeding the bandwidth limit or buffering to delay packets and eliminate peaks. Although it is specified in the OpenFlow switch specification, queues are not managed (create, delete, edit) by OpenFlow protocol, but from OF-Config, OVSDB or specific tools in the scope of the switch operating system. OpenFlow is only able to query queue statistics from the switch.

The idea is to implement the traffic prioritization behaviour by using meters of type DSCP remark to match ingress traffic, remark the packets that are exceeding the guaranteed bandwidth and send them to a queue or

another based on their DSCP values. By using different queues at different rates we can prioritize "special" traffic over "ordinary" traffic and limit "greedy" traffic as a punishment. Specifically, in each switch port, three different queues are statically created: queue 2 has the highest priority and forwards conformant QoS traffic; queue 1 has lowest priority and forwards non-conformant QoS traffic; best effort traffic uses queue 0, that is the default forwarding queue.
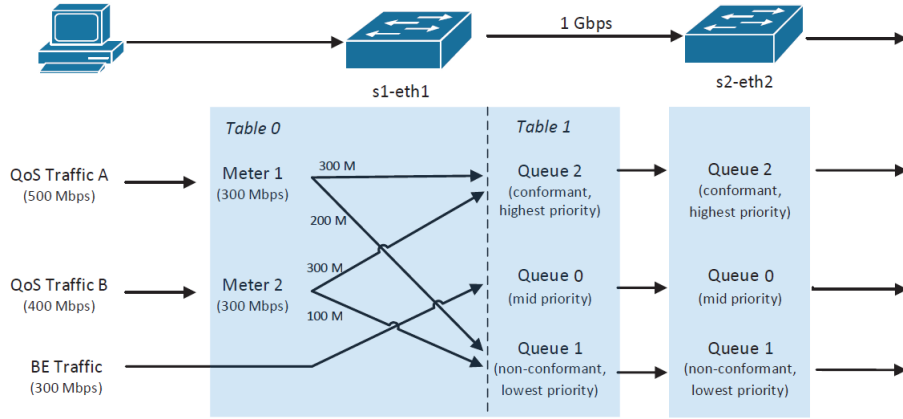

Figure 2: Traffic Prioritization concept [1]

2.3   Switch

Initially we chose *Open vSwitch* as the switch with which test the module functionalities but unfortunately, to this day, the software implementation of *Open vSwitch* does not support meters of type *DSCP remark* but only of type *drop*. To address this issue we employed another software switch, that fully supports OpenFlow 1.3 meters, *BOFUSS*; unluckily BOFUSS does not support OpenFlow 1.3 queues (but only OpenFlow 1.0 queues).

Since we need a switch that supports both meters and queues, we employed both the switches previously mentioned in order to obtain one single "virtual" switch that fully supports OpenFlow 1.3 meters and queues. More in detail, we connected a *BOFUSS* switch and a *OvS* switch in cascade, we implemented the functionalities that require usage of the meters only in the first switch (*BOFUSS*); the first switch then forwards all the traffic, both the one that has been DSCP remarked and the one that remained unchanged, to the second switch (*OvS*) that has the task of carrying out the operations of queuing. An explanatory illustration is shown in figure 3.

The only drawback of this workaround is that whenever the user wants to enable a switch to support the traffic prioritization module, he has no longer to specify a single switch but a pair of switches, making sure that the first is

a *BOFUSS* switch and the second a *OvS* one. It is also required that all the other switches in the topology should support OpenFlow queues otherwise the traffic prioritization behaviour will be lost after the first virtual switch.
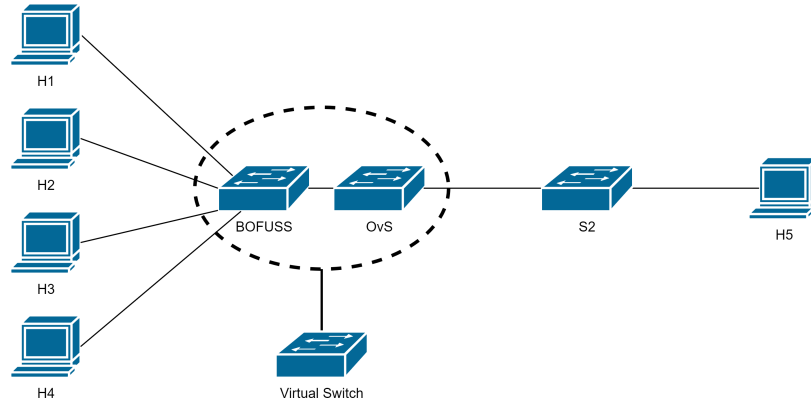

Figure 3: Reference topology with the workaround

2.4 QoS Traffic Flow Registering

When a user registers as QoS a new traffic flow, the controller sends 5 different messages to the specified pair of switches in order to install on them the flows and the meters that are required for the prioritization. The following sections will showcase the details of all the messages sent to each of the two switches.

2.4.1 Meter Switch (BOFUSS)

As said, *BOFUSS* was employed as the switch supporting OpenFlow 1.3 meters, the meters of this software switch perform the DSCP remark action only for certain values of the *Type of Service* field. As you can see from the source code of the switch, that can be viewed here and that is reported below for reference, packets are remarked only if they arrive at the switch with a valid drop precedence (low or medium).

```
1  /* The spec says that we need to increase the drop precedence of the packet. We
        need a valid DSCP out of the process, so we can only modify dscp if the drop
        precedence is low (tos 0x***010**) or medium (tos 0x***100**). Jean II */
2  if (((old_drop == 0x8) && (band_header->prec_level <= 2)) || ((old_drop == 0x10)
        && (band_header->prec_level <= 1))) {
3      uint8_t  new_drop = old_drop + (band_header->prec_level << 3);
4      uint8_t  new_tos = new_drop | (ipv4->ip_tos & 0xE3);
5      uint16_t  old_val  = htons((ipv4->ip_ihl_ver << 8) + ipv4->ip_tos);
6      uint16_t  new_val = htons((ipv4->ip_ihl_ver << 8) + new_tos);
7      ipv4->ip_csum = recalc_csum16(ipv4->ip_csum, old_val, new_val);
8      ipv4->ip_tos = new_tos;
9  }
```

<div align="center">BOFUSS meter dscp remark source code</div>

As a consequence, the controller first installs a flow that set the correct ToS value on all the packets that belong to the registered QoS traffic flow so that they can be DSCP remarked; this flow needs to be the first flow that a packet will go through so it is placed in the first table of the switch (table 0). We chose to set DSCP = 2 which corresponds to ToS 0x08, as you can see in the correspondence table.

TOS (Dec)	TOS (Hex)	TOS (Bin)	TOS Precedence (Bin)	TOS Precedence (Dec)	TOS Precedence Name	TOS Delay flag	TOS Throughput flag	TOS Reliability flag	DSCP (Bin)	DSCP (Hex)	DSCP (Dec)	DSCP/PHB Class
0	0x00	00000000	000	0	Routine	0	0	0	000000	0x00	0	none
4	0x04	00000100	000	0	Routine	0	0	1	000001	0x01	1	none
8	0x08	00001000	000	0	Routine	0	1	0	000010	0x02	2	none
12	0x0C	00001100	000	0	Routine	0	1	1	000011	0x03	3	none
16	0x10	00010000	000	0	Routine	1	0	0	000100	0x04	4	none

<div align="center">Figure 4: ToS Correspondance Table</div>

Next, the packets has to go through the actual DSCP remark. To this scope, the controller sends a message to create a new meter of type DSCP

remark and with the specified guaranteed bandwidth (meters and QoS traffic flows have a 1-to-1 relationship, we have one meter for each QoS traffic flow). The controller also installs a go-to meter flow that sends the packets belonging to the registered QoS traffic flow through the new meter; this flow must necessarily be processed after the flow that set the default ToS value so it is placed in table 1. So, finally, the packets exit the meter switch after being DSCP remarked, if they exceed the guaranteed bandwidth, or with the default value of DSCP if they are not.

Note that it is not possible to create a unique flow that sets the default ToS value and with the go-to meter instruction, since in OpenFlow the actions that involves the meters are evaluated before any other action, regardless their order.

2.4.2 Queue Switch (Open vSwitch)

The queue supporting switch has to perform the operation of setting the queue according to the DSCP value. In detail, two flows are installed by the controller: a flow that matches the DSCP remarked packets and queues them in the less effort queue, the queue with the lowest priority; a flow that matches the packets that are not exceeding the bandwidth rate, and so that have not been DSCP remarked, and queues them in the highest priority queue, the QoS queue.

The following picture shows the processing of the packets belonging to a registered QoS traffic flow.
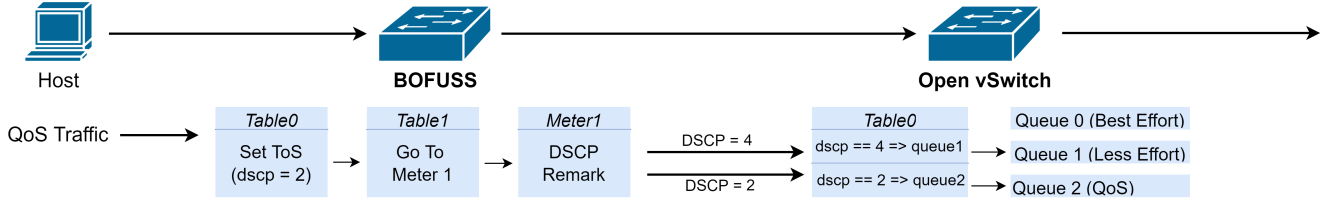

Figure 5: QoS traffic workflow

2.5 REST Interface

A RESTful interface is provided to the user (the network administrator) to manage the network and the traffic prioritization. The controller exposes a set of endpoints that are reachable at the following URLs:

- *http://CONTROLLER_IP:8080/qos/switches/topology*

- *http://CONTROLLER_IP:8080/qos/switches/json*

- *http://CONTROLLER_IP:8080/qos/flows/json*

- *http://CONTROLLER_IP:8080/qos/stats/json*

2.5.1 Endpoint Topology

As analyzed in the second section, to enable the traffic prioritization we need two switches in cascade, the first one (BOFUSS) has to support meters and the second one has to support queues (OpenFlow). This endpoint allows the user to retrieve the information needed to select the switches on which enabling the traffic prioritization module.

- **GET**: retrieves the information of the topology of the switches. In particular it returns, for each switch connected to the controller, the switch type (*OvS* or *BOFUSS*) and a list of switches to which it is connected, so a user can be sure to select, in the correct order, two directly connected switches that support the required functionalities.

```
1  {
2    "00:00:00:00:00:00:00:06": {
3      "switch-type": "OpenFlow 1.3 Reference Userspace Switch",
4      "switch-links": [
5        {
6          "src-switch": "00:00:00:00:00:00:00:06",
7          "src-port": 1,
8          "dst-switch": "00:00:00:00:00:00:00:07",
9          "dst-port": 1,
10         "latency": 29
11       }
12     ]
13   },
14   "00:00:00:00:00:00:00:07": {
15     "switch-type": "Open vSwitch",
16     "switch-links": [
17       {
18         "src-switch": "00:00:00:00:00:00:00:07",
19         "src-port": 1,
20         "dst-switch": "00:00:00:00:00:00:00:08",
21         "dst-port": 1,
22         "latency": 26
23       },
24       {
25         "src-switch": "00:00:00:00:00:00:00:07",
26         "src-port": 5,
27         "dst-switch": "00:00:00:00:00:00:00:06",
28         "dst-port": 5,
29         "latency": 47
30       }
31     ]
32   }
33 }
```

GET response sample

2.5.2 Endpoint Switches

The Switches endpoint permits to the administrator to manage the traffic prioritization features on the switches. A switch is identified by its DPID.

- **GET**: retrieves the list of pair of switches that have been enabled to support the traffic prioritization module.

- **POST**: enables the traffic prioritization on the specified pair of switches. The request is submitted in JSON format specifying the parameters "dpid-meter-switch" and "dpid-queue-switch".

- **DELETE**: disables the traffic prioritization on the specified pair of switches. The request is submitted in JSON format specifying the parameters "dpid-meter-switch" and "dpid-queue-switch".

```
1  [
2    "Meter Switch: 00:00:00:00:00:00:00:06 - Queue Switch: 00:00:00:00:00:00:00:07"
3  ]
```

GET response sample

2.5.3 Endpoint Flows

The Flows endpoint allows the administrator to manage the traffic flows. Note that prior to registering a traffic flow, a user must enable the traffic prioritization module on the selected pair of switches. A QoS traffic flow has the following fields: the source IPv4 address, the destination IPv4 address, the guaranteed bandwidth value and the DPIDs of the pair of switches on which it has been installed.

- **GET**: retrieves the list of QoS traffic flows that have been registered on the specified pair of switches.

- **POST**: registers a QoS traffic flow. The The request is submitted in JSON format specifying the parameters "dpid-meter-switch", "dpid-queue-switch", "src-addr", "dst-addr" and "bandwidth". The module checks if there is already a registered flow between the source and the destination and if the pair of switches has been enabled.

- **DELETE**: de-registers a QoS traffic flow that has the specified fields. The request is submitted in JSON format specifying the parameters "dpid-meter-switch", "dpid-queue-switch", "src-addr" and "dst-addr". The module checks if the flow has previously registered.

```
1  [
2    {
3      "dpid-meter-switch": "00:00:00:00:00:00:00:06",
4      "dpid-queue-switch": "00:00:00:00:00:00:00:07",
5      "src-addr": "10.0.0.2",
6      "dst-addr": "10.0.0.5",
7      "bandwidth": 3000
8    },
9    {
10     "dpid-meter-switch": "00:00:00:00:00:00:00:06",
11     "dpid-queue-switch": "00:00:00:00:00:00:00:07",
12     "src-addr": "10.0.0.3",
13     "dst-addr": "10.0.0.5",
14     "bandwidth": 3000
15   },
16   {
17     "dpid-meter-switch": "00:00:00:00:00:00:00:06",
18     "dpid-queue-switch": "00:00:00:00:00:00:00:07",
19     "src-addr": "10.0.0.4",
20     "dst-addr": "10.0.0.5",
21     "bandwidth": 3000
22   },
23   {
24     "dpid-meter-switch": "00:00:00:00:00:00:00:06",
25     "dpid-queue-switch": "00:00:00:00:00:00:00:07",
26     "src-addr": "10.0.0.1",
27     "dst-addr": "10.0.0.5",
28     "bandwidth": 10000
29   }
30 ]
```

GET response sample

2.5.4 Endpoint Stats

The stats endpoint allows the administrator to retrieves statistics of the module. The QoS classes are: Best Effort, Non-Conformant and Conformant.

- **POST**: retrieves the number of packets handled by each QoS class. The request is submitted in JSON format specifying the parameter "dpid-switch". A switch does not have to be enabled to respond to this request. Note that BOFUSS switches does not support OpenFlow queues so they reply with an empty response.

```
1  {
2    "Best Effort Traffic": 3000,
3    "Conformant QoS Traffic": 1500,
4    "Non-conformant QoS Traffic": 500
5  }
```

POST response sample

3   Implementation

As far as the implementation part is concerned, we mainly implemented and extended two packages within floodlight:
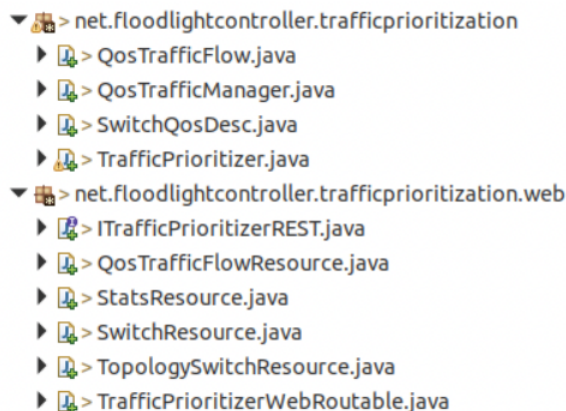
Figure 6: Floodlight Package

- **net.floodlightcontroller.trafficprioritizazion**: contains the main modules responsible for managing traffic prioritization and thus QoS management, flow and DSCP Remark operations;

- **net.floodlightcontroller.trafficprioritizazion.web**: it takes care of the REST API management described in previous chapter.

In order to implement the solution presented in this project, the implementation of the Traffic Prioritization module will now be described, omitting the methods and classes related to the REST interface for simplicity.

3.1   TrafficPrioritizer.java

- `private` `void` installSetTosFlow(`final` IOFSwitch sw, IPv4Address srcAddr, IPv4Address dstAddr)

  Installs a flow that sets the default Type of Service for the traffic flow that has been registered as Quality of Service Traffic flows. Of-SoftSwitch BOFUSS meters apply the DSCP remark only to packets with low value of ToS, for this reason we should first set an appropriate value of ToS to the packets belonging to the registered QoS traffic flow so that the meter applies the DSCP remark.

- `private void` removeSetTosFlow(`final` IOFSwitch sw, IPv4Address srcAddr, IPv4Address dstAddr)

  Removes a flow that sets the default Type of Service for the traffic flow that has been registered as Quality of Service Traffic flows.

- `private void` createMeter(`final` IOFSwitch sw, `final long` meterId, `final long` rate, `final long` burstSize))

  Creates a meter of type dscp_remark in a switch.

- `private void` removeMeter(`final` IOFSwitch sw, `final long` meterId)

  Removes a meter of type dscp_remark from a switch.

- `private void` installGoToMeterFlow(`final` IOFSwitch sw, `int` meterId, IPv4Address srcAddr, IPv4Address dstAddr)

  Installs a flow that makes the packets, belonging to a registered QoS traffic flow, go through the specified meter. The packets exceeding the bandwidth of the meter, will be DSCP remarked.

- `private void` removeGoToMeterFlow(`final` IOFSwitch sw, `int` meterId, IPv4Address srcAddr, IPv4Address dstAddr)

  Remove a flow that makes the packets, belonging to a registered QoS traffic flow, go through the specified meter.

- `private void` installEnqueueBasedOnDscpFlow(`final` IOFSwitch sw, IPv4Address srcAddr, IPv4Address dstAddr, `int` queueId, IpDscp dscp)

  Installs a flow that sends to the specified queue the packets that has the specified DSCP value.

- `private void` removeEnqueueBasedOnDscpFlow(`final` IOFSwitch sw, IPv4Address srcAddr, IPv4Address dstAddr, `int` queueId, IpDscp dscp)

  Removes a flow that sends to the specified queue the packets that has the specified DSCP value.

4 Testing

In the following section will be described four different tests that aim to demonstrate all the functionalities of the Traffic Prioritization module. All the tests have been performed on the reference topology with a 10Mbit/s limited bandwidth link between the virtual switch S1 (BOFUSS + OvS) and the S2 switch, as shown in figure.
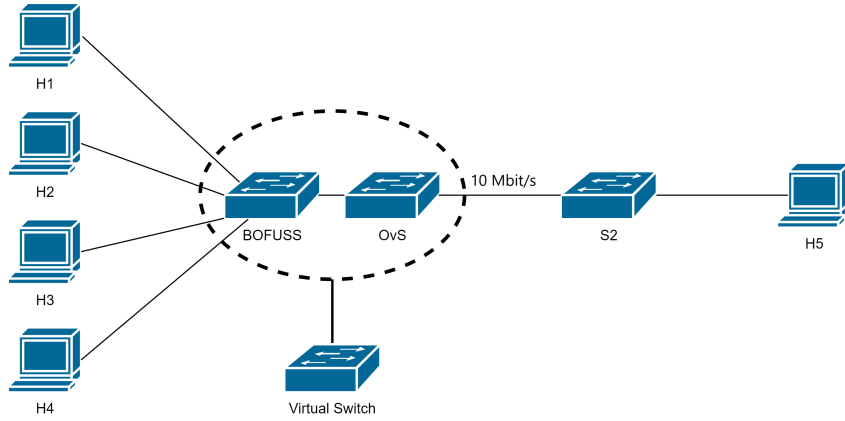
Figure 7: Reference topology for the tests

Mininet provides the *TCLink* class to create links with particular features, e.g. bandwidth limitation, but it does not work properly if used together with OvS queues due to a bug [5]. To overcome this, we simulated the limited bandwidth link using the *max rate* parameter of *linux htb*, when creating the queues.

4.1 Test A: DSCP remark

The aim of the first test is to prove the functionality of lowering the priority of the traffic that exceeds the guaranteed bandwidth rate by performing DSCP remark.
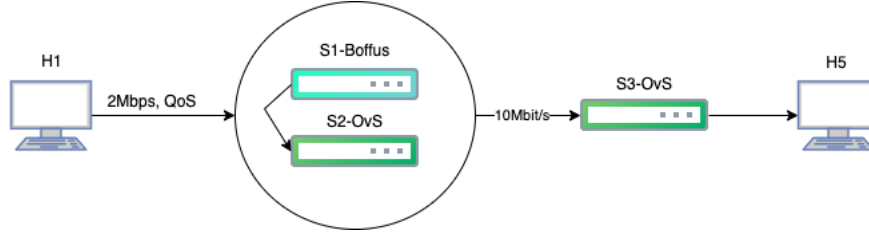
Figure 8: Test A Topology

In particular, host H1 has registered a QoS traffic flow with 2Mbit/s of guaranteed bandwidth, but it sends 5Mbit/s of traffic so about 3Mbit/s of traffic are exceeding the guaranteed rate and hence should be remarked.

It can be checked by inspecting the packets captured using Wireshark: the total number of packets sent by H1 (10.0.0.1) is 17669, of these 11104 were remarked (ToS = 0x10), as shown in the figure 9, and the remaining 6565 packets has the default value of Type of Service (ToS = 0x08) and so they have not been remarked, figure 10. About 37% of the total packets have been actually delivered as QoS that is close to 40%, the value that we expect theoretically.

Figure 9: Packets remarked

Figure 10: Packets not remarked

4.2 Test B: QoS guarantees

This test aims to prove the functionality of traffic rate guarantees by the subscription of a QoS flow.
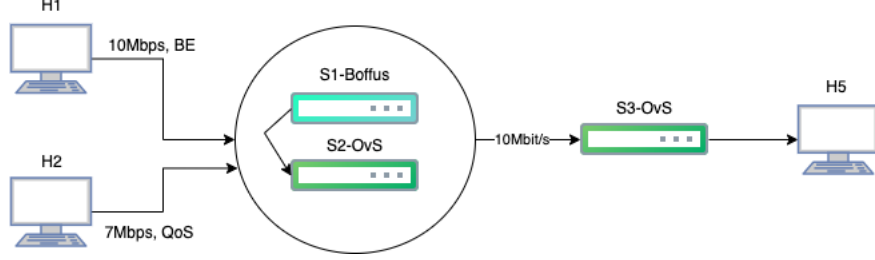

Figure 11: Test B Topology

In particular, host H2 has registered a QoS flow with a guaranteed bandwidth of 7Mbit/s, at the same time host H1 sends on the same network 10Mbit/s, host H1 is assumed to be working in Best Effort mode.
In the simulation, host H1 starts sending 5 seconds before host H2 and H2 stops sending 5 seconds before H1.

What is clear from the graph is that as long as H1 is the only one using the portion of the network, it manages to use the entire available troughput, but when H2 begins to send traffic, H1 suffers a drastic drop of about 7Mbit/s compatible with the respect of the configured QoS of H2 traffic. When H2 stops sending packets, the throughput of H1 returns approximately 10 Mbit/s.

Test 1: Data rate generated at sender hosts

Test 1: Actual throughput at receiving hosts
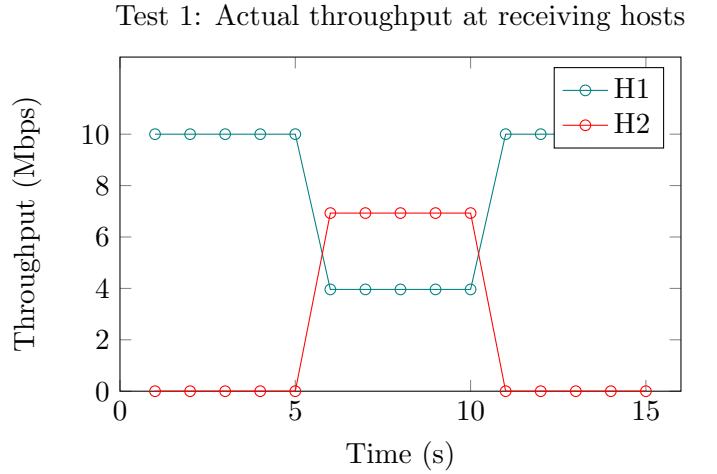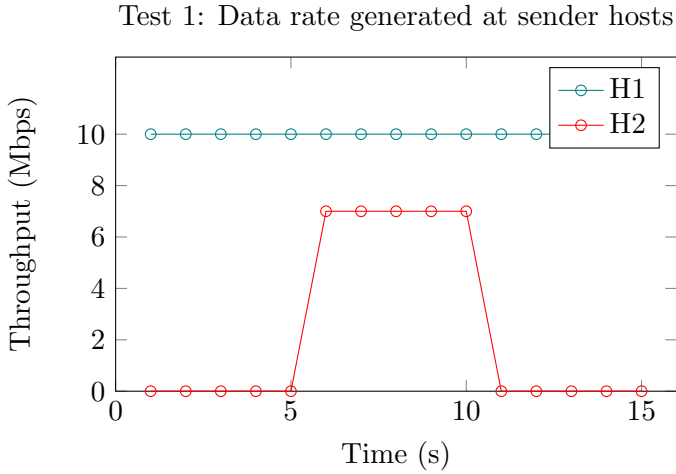
4.3 Test C: Traffic Prioritization

The test C is intended to prove the functionality of penalizing traffic that do not stay within the guaranteed bandwidth rate.
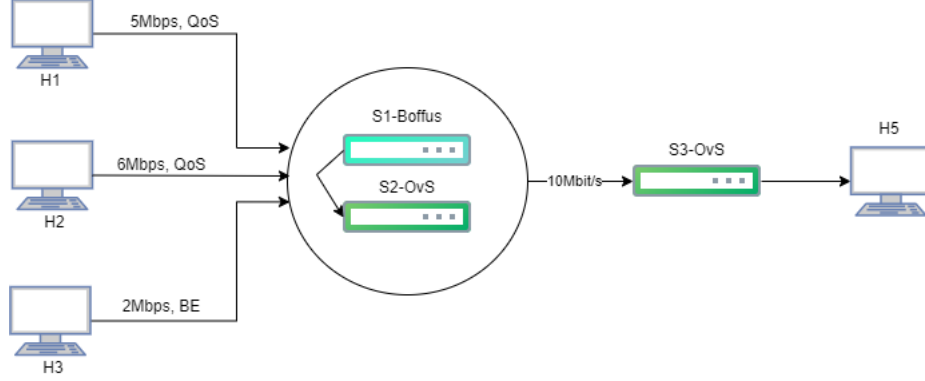

Figure 12: Test C Topology

In particular, host H1 has a guaranteed QoS bandwidth of 5 Mbit/s and sends exactly 5 Mbit/s of traffic, while host H2 has a guaranteed QoS bandwidth of 3 Mbit/s, but sends 6Mbit/s of traffic over the network. H3 sends 2 Mbit/s of Best Effort traffic.

From the graph, it is possible to see how H1's QoS is respected in that it is always guaranteed to send the configured 5Mbit/s, while H2, although it sends 6Mbit/s, is only guaranteed the 3Mbit/s configured in QoS; the remain of the traffic belonging to H2 is dropped; note that H3 traffic, since it has a higher priority than H2 traffic that exceeds the guaranteed rate, it is not dropped.
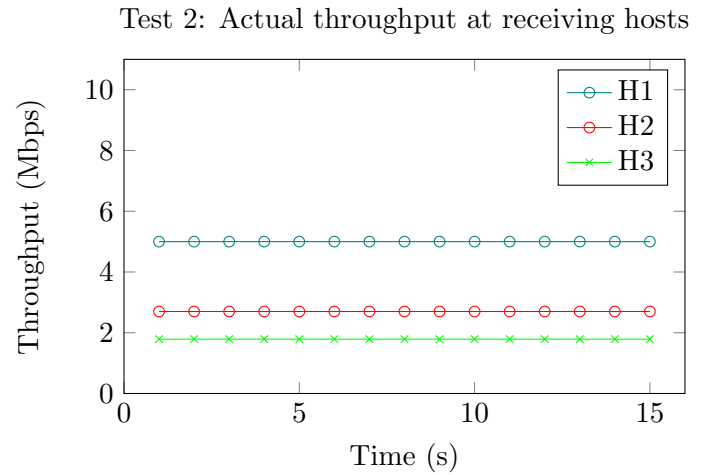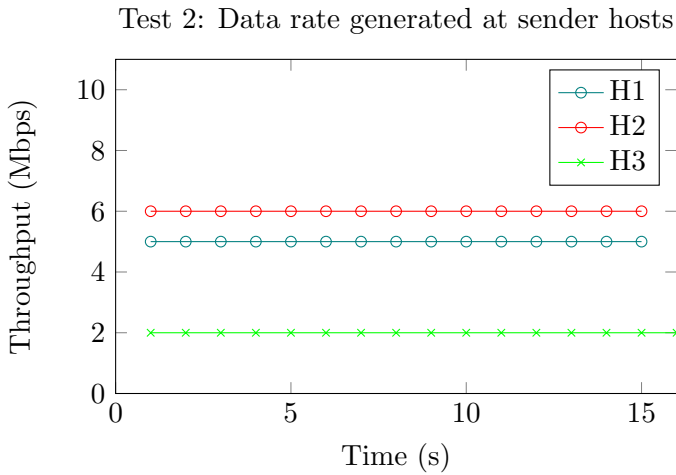
4.4 Test D: Comprehensive test

The last test is similar to the one that the authors of [1] have proposed. H1 generates Best Effort traffic, while H2, H3 and H4 generate QoS traffic, each of them has a guaranteed rate of 3 Mbps.
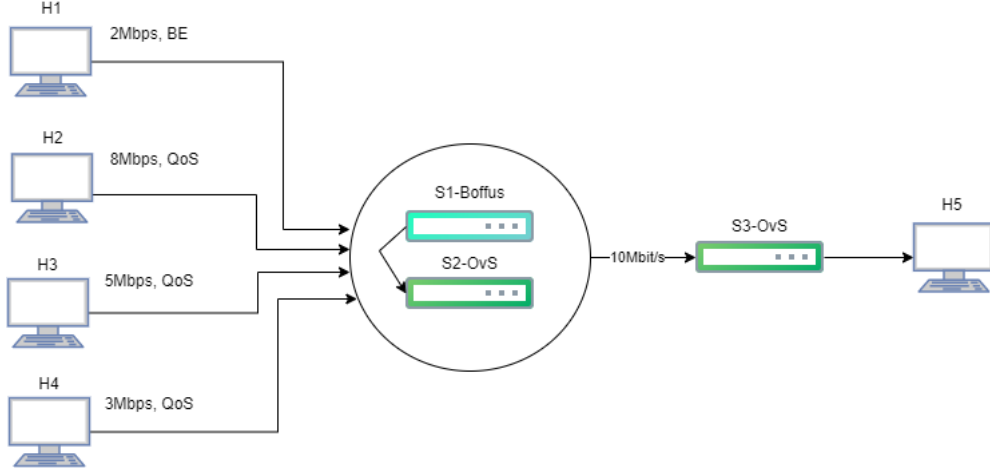

Figure 13: Test D Topology

1. **0s to 5s**. H1 starts sending Best Effort traffic to H5 with a rate of 2 Mbps. At H5, the throughput measures 2 Mbps.

2. **5s to 10s**. H2 starts sending QoS traffic with a throughput of 8 Mbps. H2 has only 3 Mbps of guaranteed rate, however since 8 Mbps of link capacity are not used, all the traffic of H2 is received by H5, with a throughput of 8 Mbps.

3. **10s to 15s**. H3 starts sending 5 Mbps of traffic. Since the bandwidth of the link is not enough to accommodate all the traffic flows at their full rate, the traffic that exceed the guaranteed rate, is forwarded as the one with the lowest priority. So, H5 receives slightly more than 3 Mbps of throughput from H2 and H3, 3 Mbps of guaranteed rate plus some exceeding traffic, and 2 Mbps of throughput from H1, whose flow is Best Effort.

4. **15s to 25s**. H4 starts sending 2 Mbps of traffic. The link will be fully used by the QoS flows of H2, H3, and H4 and by the Best Effort flow of H1, so the switch will drop all the non conformant packets sent by H2 and H3 to accomodate the new QoS flow of H4.

5. **25s to 30s**. H4 increases its data rate from 2 Mbps to 3 Mbps. In order to accomodate all the QoS flows, 1 Mbps of Best Effort traffic from H1 is dropped.
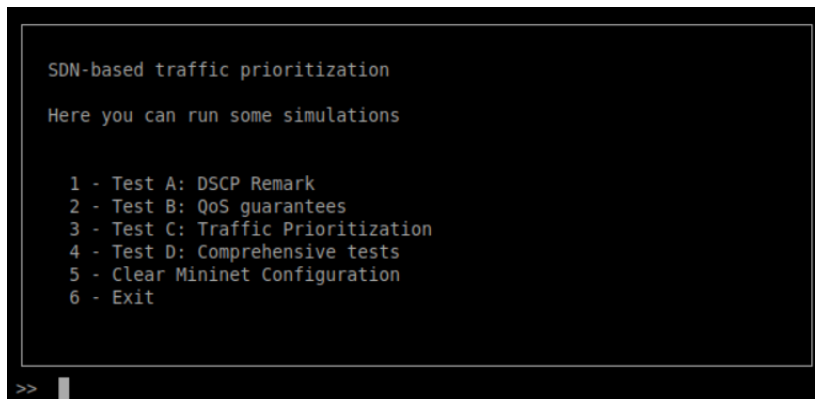
18

Test 3: Data rate generated at sender hosts
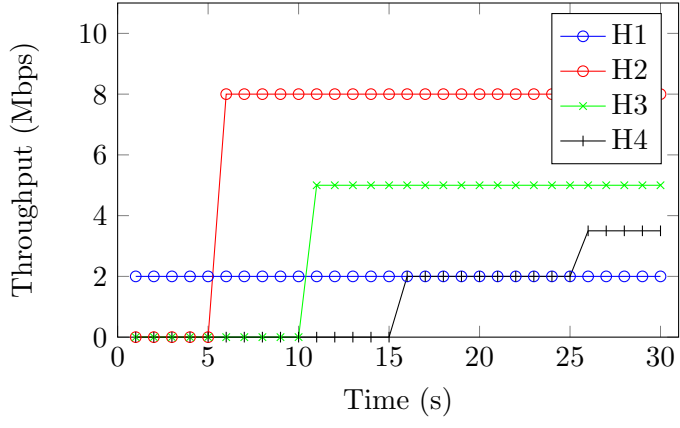
Test 3: Actual throughput at receiving hosts
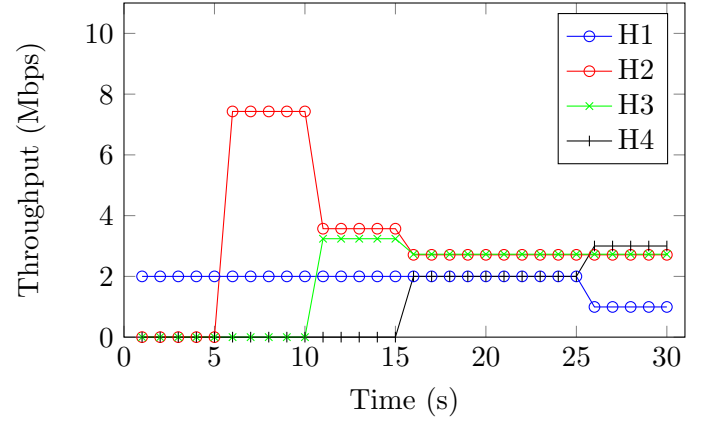
5 Simulation Tool and Prerequisites

The tests that have been presented in the previous section have been compiled into a single Python script that allows you to visualise and choose the simulation you wish to run. This was done by creating an interactive menu where the user can launch the desired test, this will launch the corresponding file containing the topology and Mininet configuration that allows it to be executed. When finished, it is possible to return to the main menu, where a clear function of the Mininet configuration is also contained.

Figure 14: Terminal Input

All remaining prerequisites, including the main steps for installing the environment and bugfixing are contained in the repository Readme file [6].

6 Bibliography

[1] H. Krishna, N. L. M. van Adrichem and F. A. Kuipers, "Providing bandwidth guarantees with OpenFlow," 2016 Symposium on Communications and Vehicular Technologies (SCVT), 2016, pp. 1-6, doi: 10.1109/SCVT.2016.7797664.

[2] `https://github.com/CPqD/ofsoftswitch13`

[3] `https://github.com/openvswitch/ovs`

[4] `https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller`

[5] `https://github.com/mininet/mininet/issues/653`

[6] `https://github.com/edoardoruffoli/SDN-TrafficPrioritization`