ABSTRACTION LAYER FOR IMPLEMENTATION OF EXTENSIONS IN PROGRAMMABLE NETWORKS

Collaborative project co-funded by the European Commission within the Seventh Framework Programme

Grant agreement no: 317880
Project acronym: ALIEN
Project full title: "Abstraction Layer for Implementation of Extensions in programmable Networks"
Project start date: 01/10/12
Project duration: 24 months

Deliverable D5.3
Experimental-driven research - Appendix II (OpenFlow QoS extensions experiment results)

Version 7.9

Due date: 30/10/2014
Submission date: 12/11/2014
Editor: Maider Huarte (EHU)
Author list: Eduardo Jacob, Victor Fuentes, Maider Huarte (EHU).

Dissemination Level		
☒	**PU:**	Public
☐	**PP:**	Restricted to other programme participants (including the Commission Services)
☐	**RE:**	Restricted to a group specified by the consortium (including the Commission Services)
☐	**CO**	Confidential, only for members of the consortium (including the Commission Services)

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

Table of Contents

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

Figure summary

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

Executive Summary

This document complements deliverable D5.3 “Experimental-driven research” with details about testing scenario steps and all materials like logs, screenshots, packet dumps collected during performing of experimentation tests. We highly encourage to read first D5.3 which contains overview of the experiment, goals to be achieved, experiment environment and also discussion about experiment validation.

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

1 OpenFlow QoS extensions

1.1 Testing scenario

The testing carried out for the demonstration of the OpenFlow QoS extensions developed is described in the next workflow. It describes how the proxy works and the interaction between OpenFlow and DOCSIS to provide the abstraction of the QoS mechanisms.

1.1.1 Step 0: Default QoS profiles

ALIEN DOCSIS default configuration related to QoS is defined in a new configuration file. This configuration file is parsed by ALHINP (ALien HAL-based Integration Network Proxy). Proxy reads this file and generates a default profile of service flows to be loaded into cablemodem. Furthermore, it is also possible to set an ad-hoc configuration of QoS for a certain cablemodem specifying its corresponding mac address and defining the custom queue set to be generated for it.

The configuration file is organized as the next example:

```
DEFAULT_QUEUELIST = ({ queueID  = "default";
                       max_rate = 500000;
                       min_rate = 100000;
                       direction = "bidirectional";
                       source_port = 0xffffffff
                       destination_port = 12 },

                     { queueID = "openflow";
                       min_rate = 2000000;
                       max_rate = 2000000;
                       direction = "bidirectional";
                       source_port = 0
                       destination_port = 0 }
);
```

QueueID represents the name of the queue to be created. Both OpenFlow and default labelled queues are mandatory to be defined. Otherwise the proxy alerts you to create them before running the proxy. Max_rate is the maximum allowed bandwidth in bps. Min_rate represent the guaranteed bandwidth in bps. Direction can be set as one-direction or bidirectional. Source_port and destination_port: specifies between which ports the queue is going to be created. OFPP_ALL is used to represent all client ports, whereas a 0 value means not to care, and it is only used for OpenFlow queue only.

Default configured queues are two. The first one, corresponding to OpenFlow control link, is a 2 megabit symmetric connection, whereas default queue is a 500kb/100kb connection.

The section parsed to load a custom profile for a cablemodem is described as follows:

```
CMLIST = ( { mac = A4A24A441F29;
                    queuelist = (

                      { queueID  = "openflow";
                        max_rate = 1000000;
                        min_rate = 1000000;
                        direction = "bidirectional";
                        source_port = 0xffffffff
                        destination_port = 0 },

                      { queueID  = "default";
                        max_rate = 1000000;
                        min_rate = 1000000;
                        direction = "bidirectional";
                        source_port = 0xffffffff
                        destination_port = 12 },

                      { queueID  = "custom1";
                        max_rate = 20000000;
                        min_rate =  3000000;
                        direction = "bidirectional";
                        source_port = 0xffffffff
                        destination_port = 12 } );
            }
);
```

The configuration structure is almost the same as the default profile, but a field where the mac address of the device has to be added.

1.1.2 Step 1: Network boot up process

When a new cablemodem is connected and a RF interface is configured and synchronized, a DHCP request is sent by the cablemodem. This request packet matches a specific flow rule in the aggregation switch and the action set for this matching is to send the packet to controller, (ALHINP proxy). Once the packet arrives the ALHINP, the cablemodem mac address is gathered. Then a VLAN is assigned to the cablemodem and this configuration is pushed to the CMTS. ALHINP checks if there is any specific QoS string for it. If not, default queue schema is compiled for it.

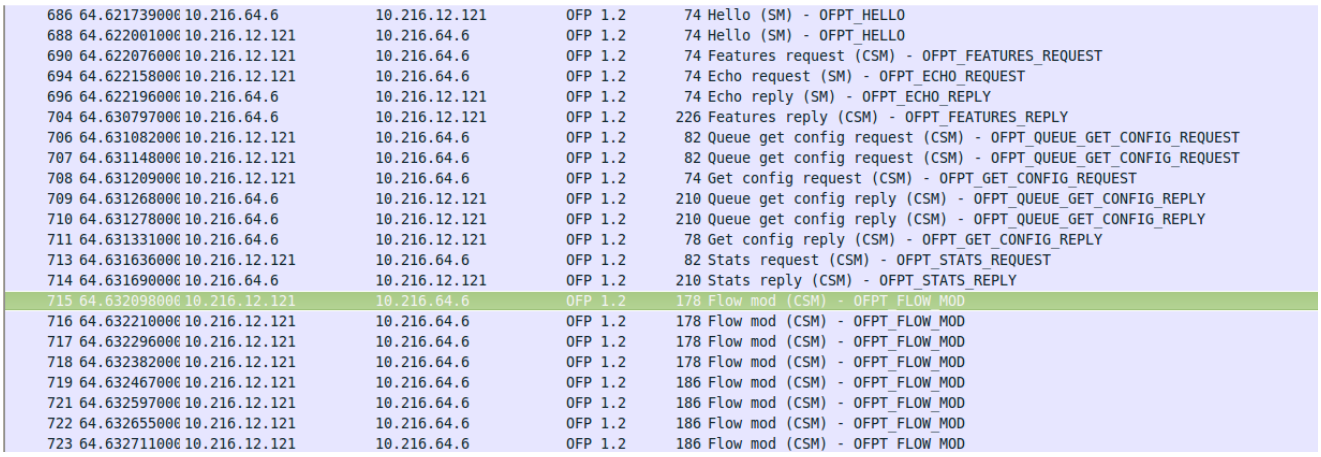
Based on the cablemodem binary file template, as many service-flows as queues requested are configured, keeping the control of some of the parameters:

- A service-flow reference number has to be unique in the cablemodem configuration file, so the proxy controls the order of service-flow entities created.
- When a bidirectional queue is requested, two service-flows are created, one for each direction.
- Assignation of queue identifiers is done based on the VLAN assigned to the cablemodem in the L2VPN configuration and appending the auto assigned queue number. There is a limit of 16 configurable queues per cablemodem. Queue identifier is used as VLAN ID for the classifier associated with the service flow.

1.1.3 Step 2: Controller, ask for queues

Once the cablemodem is configured and the service-flows and classifiers are provisioned, the OpenFlow User Instance, the helper located at the user side, will try to contact ALHINP. After this connection is established, the ports will be reported to the controller as available by using a PORT_STATUS message. However, in this message, information about queues available for this device cannot be reported. Therefore, the controller should ask for the queues available for the new ports sending a QUEUE_GET_CONFIG_REQUEST.

After receiving QUEUE_GET_CONFIG_REQUEST, the corresponding reply message is built where details are given such as *src_port*, maximum allowed bandwidth and maximum sustained bandwidth. The message exchange is depicted in Figure 1-1 OpenFlow message exchange between controller and proxy.



686	64.621739000	10.216.64.6	10.216.12.121	OFP 1.2	74 Hello (SM) - OFPT_HELLO
688	64.622001000	10.216.12.121	10.216.64.6	OFP 1.2	74 Hello (SM) - OFPT_HELLO
690	64.622076000	10.216.12.121	10.216.64.6	OFP 1.2	74 Features request (CSM) - OFPT_FEATURES_REQUEST
694	64.622158000	10.216.12.121	10.216.64.6	OFP 1.2	74 Echo request (SM) - OFPT_ECHO_REQUEST
696	64.622196000	10.216.64.6	10.216.12.121	OFP 1.2	74 Echo reply (SM) - OFPT_ECHO_REPLY
704	64.630797000	10.216.64.6	10.216.12.121	OFP 1.2	226 Features reply (CSM) - OFPT_FEATURES_REPLY
706	64.631082000	10.216.12.121	10.216.64.6	OFP 1.2	82 Queue get config request (CSM) - OFPT_QUEUE_GET_CONFIG_REQUEST
707	64.631148000	10.216.12.121	10.216.64.6	OFP 1.2	82 Queue get config request (CSM) - OFPT_QUEUE_GET_CONFIG_REQUEST
708	64.631209000	10.216.12.121	10.216.64.6	OFP 1.2	74 Get config request (CSM) - OFPT_GET_CONFIG_REQUEST
709	64.631268000	10.216.64.6	10.216.12.121	OFP 1.2	210 Queue get config reply (CSM) - OFPT_QUEUE_GET_CONFIG_REPLY
710	64.631278000	10.216.64.6	10.216.12.121	OFP 1.2	210 Queue get config reply (CSM) - OFPT_QUEUE_GET_CONFIG_REPLY
711	64.631331000	10.216.64.6	10.216.12.121	OFP 1.2	78 Get config reply (CSM) - OFPT_GET_CONFIG_REPLY
713	64.631636000	10.216.12.121	10.216.64.6	OFP 1.2	82 Stats request (CSM) - OFPT_STATS_REQUEST
714	64.631690000	10.216.64.6	10.216.12.121	OFP 1.2	210 Stats reply (CSM) - OFPT_STATS_REPLY
715	64.632098000	10.216.12.121	10.216.64.6	OFP 1.2	178 Flow mod (CSM) - OFPT_FLOW_MOD
716	64.632210000	10.216.12.121	10.216.64.6	OFP 1.2	178 Flow mod (CSM) - OFPT_FLOW_MOD
717	64.632296000	10.216.12.121	10.216.64.6	OFP 1.2	178 Flow mod (CSM) - OFPT_FLOW_MOD
718	64.632382000	10.216.12.121	10.216.64.6	OFP 1.2	178 Flow mod (CSM) - OFPT_FLOW_MOD
719	64.632467000	10.216.12.121	10.216.64.6	OFP 1.2	186 Flow mod (CSM) - OFPT_FLOW_MOD
721	64.632597000	10.216.12.121	10.216.64.6	OFP 1.2	186 Flow mod (CSM) - OFPT_FLOW_MOD
722	64.632655000	10.216.12.121	10.216.64.6	OFP 1.2	186 Flow mod (CSM) - OFPT_FLOW_MOD
723	64.632711000	10.216.12.121	10.216.64.6	OFP 1.2	186 Flow mod (CSM) - OFPT_FLOW_MOD

Figure 1-1 OpenFlow message exchange between controller and proxy

The controller must be extended to understand correctly the custom properties. Any case, *min_guaranteed_bw* and *max_allowed_bandwidth* can be understood by any of controller as they are in the OpenFlow 1.2 specification.

In case of using a controller based on ROFL libraries, it is only needed to add and use the new message class: *cofqueue_prop_src_port*. Otherwise, it would be required to implement the corresponding property handler according to the information of the structure provided above.

If the controller does not set SET_QUEUE action in a flow-rule, ALHINP proxy installs the flow rule using the default queue identifier for it.

1.1.4 Step 3: Default QoS usage

The controller installs flow mods to enable 2 different traffics, based on IP fields. The rules installed are the next ones:

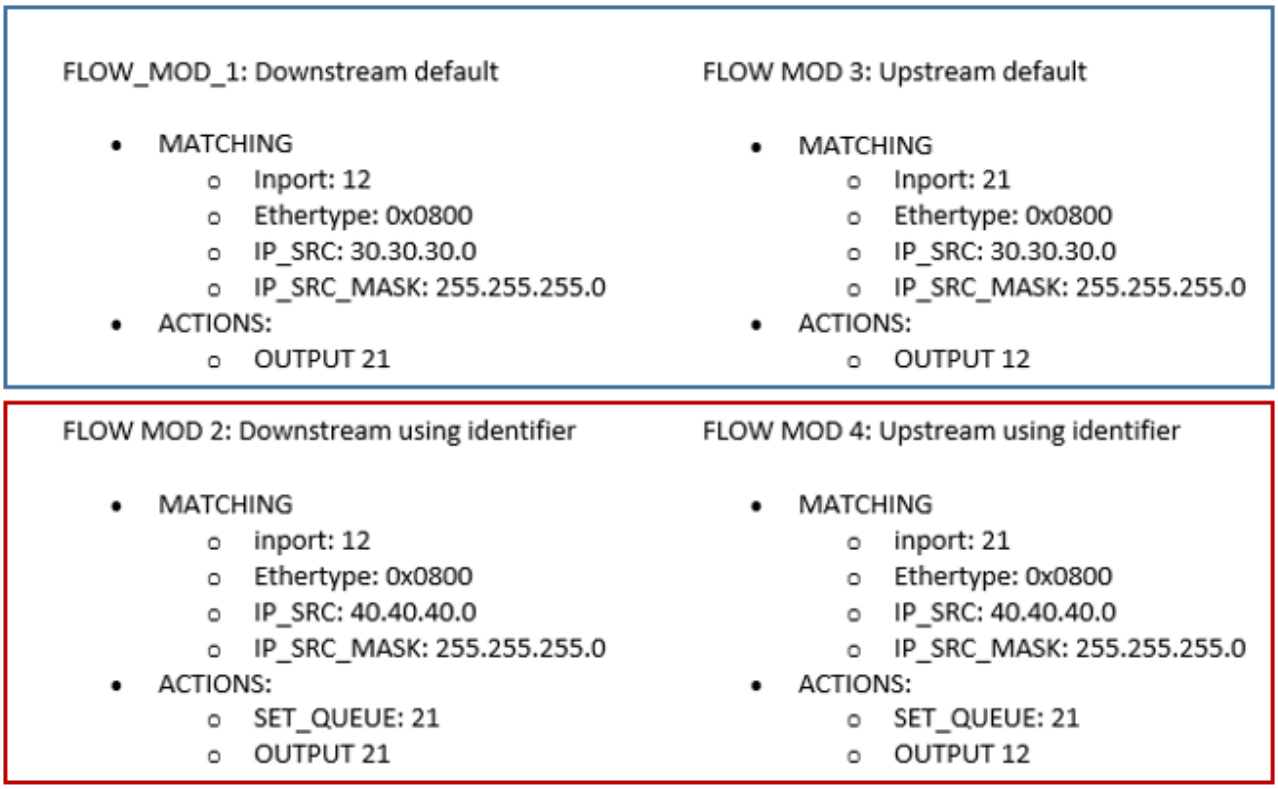


Figure 1-2 Default installed Flow Mods

In the first case, concerning to the IPs 30.30.30.0/24 no SET_QUEUE action is defined, so the proxy automatically will use the default queue for it. In case of flow mod matching 40.40.40.40/24 IPs, SET_QUEUE is used but as the only queue defined is default one (Figure 1-2 Default installed Flow Mods), both traffic will have the same behaviour and the bandwidth usage is the same.

1.1.5 Step 4: Enforcing new queues

The creation of a new queue is requested, using an experimenter message for it (Figure 1-3 Experimenter message for new queue creation). The properties of the new queue are 20Mb/3Mb from the port 12 (aggregation port) and port 21 (a client port) where the greater bandwidth is related to the network to user direction.

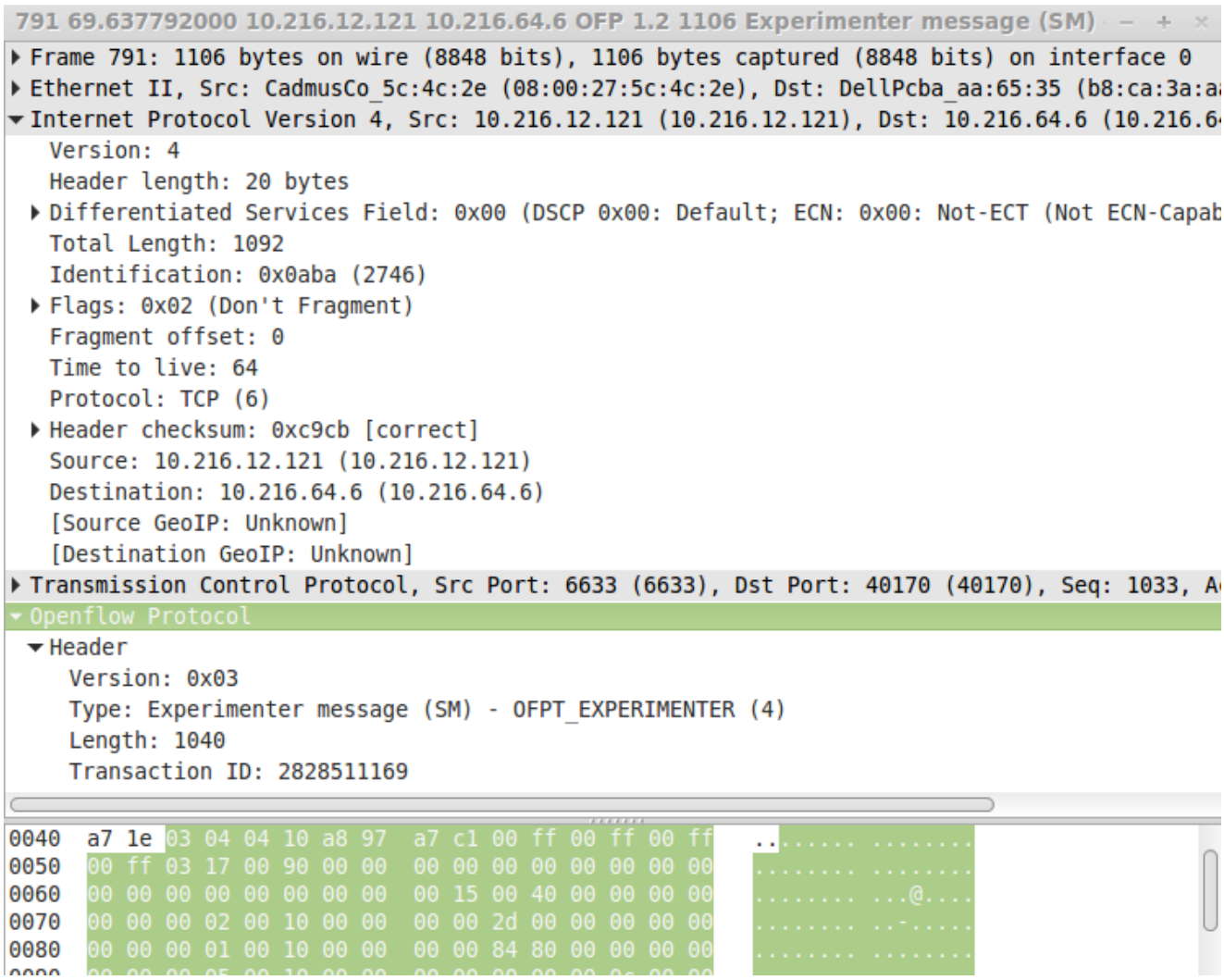


Figure 1-3 Experimenter message for new queue creation

1.1.6 Step 5: Enforcing new queues

A new profile is pushed to the provisioning and then applied to the cablemodem. Once the service flows are provisioned, a new identifier is assigned to this queue and reported to the controller via OFP_GET_QUEUE_CONFIG_REPLY message where the new properties and the identifier are detailed. From this moment, this QoS is enabled for being used by the controller.

1.1.7 Step 6: Using new queues

Now, the new queue is enabled and the controller overrides the rule for traffic from 40.40.40.40 by setting the new queue for it, this is, adding SET_QUEUE action with the new identifier. (Figure 1-4 Flow mods installed using 2 different queues).

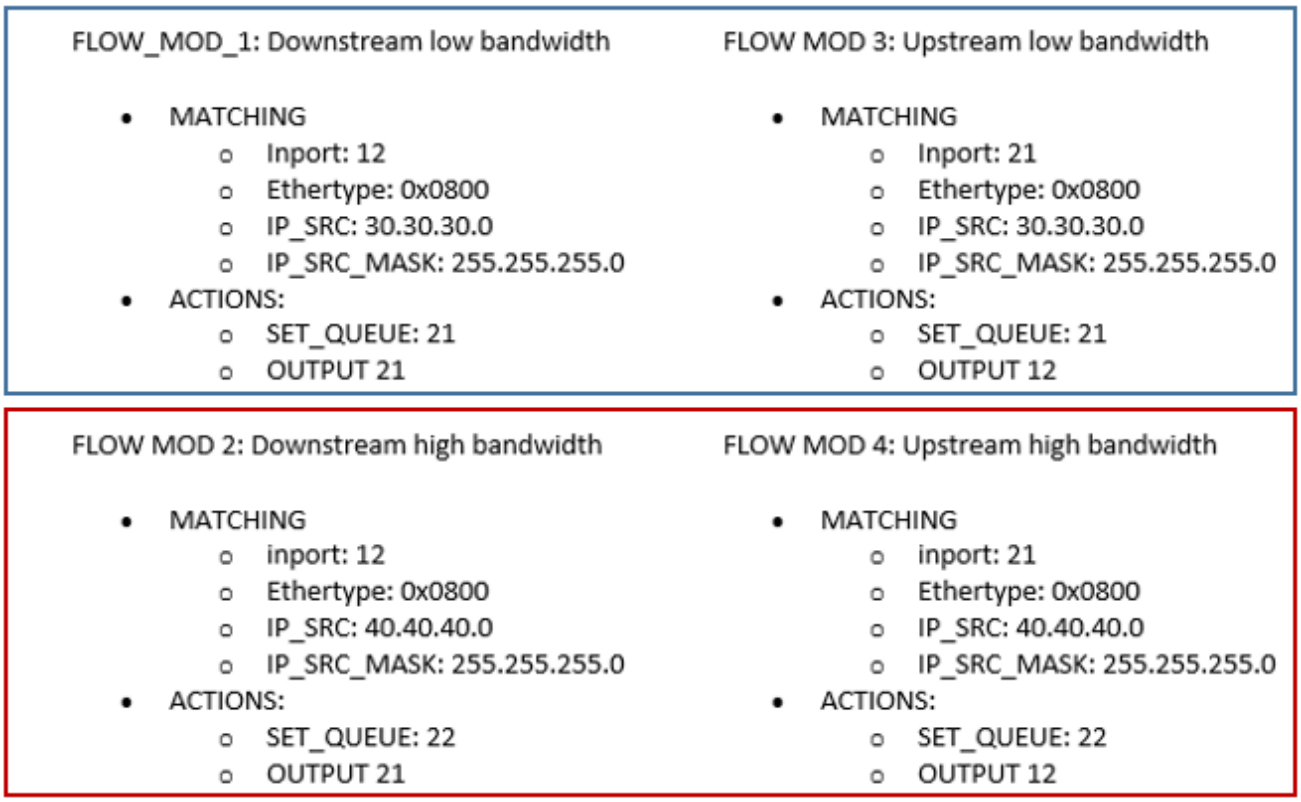


Figure 1-4 Flow mods installed using 2 different queues

1.1.8 Step 7: checking the bandwidths

In order to test the application of new queues, an Iperf test is performed between IPs from the range of 30.30.30.0/24 and then for IPs from 40.40.40.0/24. According to the rules installed and queue numbering default queue is used for 30.30.30.30/24 whereas the high bandwidth one is used for 40.40.40.0/24 IPs. The bandwidth obtained is limited by DOCSIS access network.

1.2 Experiment results

When the configuration of the ALIEN DOCSIS is finished, the controller can interact with the proxy, as shown in Figure 3.4. Then IPs from the different ranges are given to the client and server and the Iperf test is performed. The expected results are that even using the identifier the queue to be used is the same, so for both cases the bandwidth used should be almost the same, as can be seen in the captures.

```
i2t@i2tVPN ~ $ iperf -c 30.30.30.31 -r
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
------------------------------------------------------------
Client connecting to 30.30.30.31, TCP port 5001
TCP window size: 21.9 KByte (default)
------------------------------------------------------------
[  4] local 30.30.30.32 port 43901 connected with 30.30.30.31 port 5001
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-38.7 sec   384 KBytes  81.3 Kbits/sec
[  5] local 30.30.30.32 port 5001 connected with 30.30.30.31 port 45186
[  5]  0.0-15.4 sec   896 KBytes   478 Kbits/sec
i2t@i2tVPN ~ $ sudo ifconfig eth1 40.40.40.42/24
```

Figure 1-5 Bandwidth usage when using 30.30.30.0/24 IPs

```
^C^Ci2t@i2tVPN ~ $ iperf -c 40.40.40.41 -r
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
------------------------------------------------------------
Client connecting to 40.40.40.41, TCP port 5001
TCP window size: 21.9 KByte (default)
------------------------------------------------------------
[  4] local 40.40.40.42 port 45790 connected with 40.40.40.41 port 5001
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-40.8 sec   384 KBytes  77.1 Kbits/sec
[  5] local 40.40.40.42 port 5001 connected with 40.40.40.41 port 40451
[  5]  0.0-15.4 sec   896 KBytes   478 Kbits/sec
```

Figure 1-6 Bandwidth usage when using 40.40.40.0/24 IPs

Then, after new QoS profile is loaded, and the new 20Mb queue is available. These results were obtained using the network range 40.40.40.0/24, which has been configured by the controller to use the 20Mb/3Mb services (Queue ID 22).

```
i2t@i2tVPN ~ $ iperf -c 40.40.40.41 -r
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
------------------------------------------------------------
Client connecting to 40.40.40.41, TCP port 5001
TCP window size: 21.9 KByte (default)
------------------------------------------------------------
[  5] local 40.40.40.42 port 39489 connected with 40.40.40.41 port 5001
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.2 sec  23.9 MBytes  19.7 Mbits/sec
[  4] local 40.40.40.42 port 5001 connected with 40.40.40.41 port 37896
[  4]  0.0-15.3 sec  4.38 MBytes  2.40 Mbits/sec
```

Figure 1-7 Benchmarks obtained using the 20Mb/3Mb queue

These other results were obtained using the network range 30.30.30.00.0/24, which has been configured by the controller to use the 500Kb/100Kb services.

```
i2t@i2tVPN ~ $ iperf -c 30.30.30.31 -r
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
------------------------------------------------------------
Client connecting to 30.30.30.31, TCP port 5001
TCP window size: 21.9 KByte (default)
------------------------------------------------------------
[  5] local 30.30.30.32 port 37647 connected with 30.30.30.31 port 5001
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-38.7 sec   384 KBytes  81.2 Kbits/sec
[  4] local 30.30.30.32 port 5001 connected with 30.30.30.31 port 42644
[  4]  0.0-17.6 sec  1.00 MBytes   477 Kbits/sec
```

Figure 1-8 Benchmarks obtained using the low bandwidth (default) queue