Advanced Networking 2018

Lab #1: TCP Congestion Control

Report

GROUP: 3

Authors:
Kotaiba Alachkar, Kotaiba.Alachkar@os3.nl
Andrey Afanas'yev, Andrey.Afanasyev@os3.nl
Rick van Gorp, Rick.Vangorp@os3.nl
Henri Trenquier, Henri.Trenquier@os3.nl
University of Amsterdam

Q0.0 Preparation

Configuration of the switch:

We used *Calais* to connect to the *Mgmt* network. From there, we configured *Calais* to connect the Switch *Chico3*:

```
    ## AN lab2 ##
auto eno2
iface eno2 inet static
        address 10.0.1.203 #mgmt
        netmask 255.255.255.0
        broadcast 255.255.255.255
```

After restarting the network service, we can connect to *10.0.1.4* via ssh. From the switch, we can access the second port with

```
    dialout@serialserv4:~ $ minicom -b 9600 -D /dev/ttyUSB1
```

We can now configure the switch:

```
 !
 no service password-encryption
 !
 ip domain-name gr3.com
 !
 hostname gr3switch
 !
 interface Vlan1
 ip address 192.168.0.253 255.255.255.0
 !
 interface Loopback1
 no ip address
 !
 username andrey password 0 cisco
 username kotaiba password 0 cisco
 username rick password 0 cisco
 username henrilarose password 0 cisco
 !
 line con 0
 logging synchronous
 login local
line vty 0 4
 password 7cisco
 login local
 transport input ssh
line vty 5 15
 login
 !
```

Connection to the network:

Later, we can disconnect the UTP connection to the *Mgmt* and connect to *gr3switch*.

The new interface configuration is:

```
    ## AN lab2 ##
auto eno2
iface eno2 inet static
        address 192.168.0.127
        netmask 255.255.255.0
        broadcast 255.255.255.255
```

If we encounter this error:

```
    RTNETLINK answers: File exists
    Failed to bring up eno2
```

The following command has to be entered:

```
    sudo ip addr flush dev eno2
```

Finally, it is possible to connect to the local network. We can ping every team member from Henri's server.: Ping Rick:

```
    ~ \$ ping 192.168.0.200
PING 192.168.0.200 (192.168.0.200) 56(84) bytes of data.
64 bytes from 192.168.0.200: icmp_seq=1 ttl=64 time=0.391 ms
64 bytes from 192.168.0.200: icmp_seq=2 ttl=64 time=0.181 ms
^C
--- 192.168.0.200 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.181/0.286/0.391/0.105 ms
```

Ping Kotaiba:

```
    ~ \$ ping 192.168.0.69
PING 192.168.0.69 (192.168.0.69) 56(84) bytes of data.
64 bytes from 192.168.0.69: icmp_seq=1 ttl=64 time=0.295 ms
64 bytes from 192.168.0.69: icmp_seq=2 ttl=64 time=0.150 ms
^C
--- 192.168.0.69 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.150/0.222/0.295/0.074 ms
```

Ping Andrey:

```
    ~ \$ ping 192.168.0.3
PING 192.168.0.3 (192.168.0.3) 56(84) bytes of data.
64 bytes from 192.168.0.3: icmp_seq=1 ttl=64 time=0.205 ms
64 bytes from 192.168.0.3: icmp_seq=2 ttl=64 time=0.083 ms
^C
--- 192.168.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.083/0.144/0.205/0.061 ms
```

Ping the switch:

```
    ~ \$ ping 192.168.0.253
PING 192.168.0.253 (192.168.0.253) 56(84) bytes of data.
64 bytes from 192.168.0.253: icmp_seq=2 ttl=255 time=1.78 ms
^C
--- 192.168.0.253 ping statistics ---
2 packets transmitted, 1 received, 50% packet loss, time 999ms
rtt min/avg/max/mdev = 1.783/1.783/1.783/0.000 ms
```

Finally, we can connect via ssh with the following command:

```
~ \$ ssh -o KexAlgorithms=diffie-hellman-group1-sha1  henri@192.168.0.253
```

Indeed, the switch's configuration doesn't allow more recent Key exchange algorithm.

To disable HTTP and HTTPS we use the following commands:

```
gr3switch(config)#no ip http server
gr3switch(config)#no ip http secure-server
```

To disable Telnet on other Virtual Terminal lines:

```
gr3switch(config)#line vty 5 15
gr3switch(config-line)#transport input none
```

Q2.1 Data transfer without and with CoS configuration

In this question we have used *iperf* to transfer data between 192.168.0.69 and 192.168.0.200. The server has been set up using `iperf -s` and the client connected to the server using `iperf -c -u 192.168.0.69` and `iperf -c 192.168.0.69`. We have not selected a specific congestion algorithm as the switch is a Gigabit-switch, which matches the speed of the NICs of the servers. The throughput achieved is listed in *Figure 1* and *Figure 2*. We have also tested using different requested window sizes: 100, 100.000 and 100.000.000 bytes. This did not improve the speed, compared to `iperf's` defaults.

```
root@bristol:/home/kotaiba# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 192.168.0.69 port 5001 connected with 192.168.0.200 port 55194
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  1.10 GBytes   941 Mbits/sec
```

Figure 1: TCP Performance

When we change the size of the MTU on interface `eno2`, the throughput decreases. This was done by configuring the MTU-size of the interface:

```
ifconfig eno2 down
ifconfig eno2 mtu 1000 up
```

Then running the server and client with the -M option at 1000. Also when we decrease it to 100, the throughput decreases drastically.

```
root@bristol:/home/kotaiba# iperf -s -u -i 1
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.0.69 port 5001 connected with 192.168.0.200 port 51243
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0- 1.0 sec  96.4 MBytes   809 Mbits/sec   0.015 ms 34747/103525 (34%)
[  3]  1.0- 2.0 sec  96.5 MBytes   809 Mbits/sec   0.009 ms   65/68887 (0.094%)
[  3]  2.0- 3.0 sec  96.6 MBytes   811 Mbits/sec   0.039 ms  169/69102 (0.24%)
[  3]  3.0- 4.0 sec  96.3 MBytes   808 Mbits/sec   0.015 ms  322/69050 (0.47%)
[  3]  4.0- 5.0 sec  96.5 MBytes   809 Mbits/sec   0.019 ms  207/69029 (0.3%)
[  3]  5.0- 6.0 sec  96.4 MBytes   809 Mbits/sec   0.017 ms  211/68980 (0.31%)
[  3]  6.0- 7.0 sec  96.3 MBytes   808 Mbits/sec   0.016 ms  152/68878 (0.22%)
[  3]  7.0- 8.0 sec  96.5 MBytes   809 Mbits/sec   0.035 ms   52/68875 (0.075%)
[  3]  8.0- 9.0 sec  96.3 MBytes   808 Mbits/sec   0.042 ms  176/68902 (0.26%)
[  3]  0.0- 9.5 sec   916 MBytes   809 Mbits/sec   0.014 ms 36167/689435 (5.2%)
```

Figure 2: UDP Performance

Q2.2 Define a number of scenarios where the data transfers interfere with each other

After some discussion we could define following affordable scenarios, which simulate different computer network traffic:

1. Example from lab one basic scenario could be the one in which multiple nodes and applications are trying to communicate with one node.
    - Server node required to run two *iperf* servers accepting the TCP and UDP traffic. Other two nodes generates TCP and UDP traffic respectively.
2. A single node communicates with another node while the rest two try to ping first host at higher time frequency and with large packet sizes (DDOS).
    - Two nodes communicate using iperf running on TCP (server and client).
    - The rest two have a ping a iperf server with the same ping configuration like: `ping -i 0.01 -s 65507 192.168.0.200`
3. One node has a HTTP/1.1 server on board which stress tested by other two nodes another node communicates with first one via `iperf`.
    - One server is configured with Nginx and HTTP/1.1 it has also iperf server on board
    - Two nodes try to stress test Nginx using simple ab test like ab -n 1000000 -c 100 http://192.168.0.200/
    - Last one node is an iperf client.

Execution of the first scenario

We have set up the first scenario by setting up two `iperf` listeners (UDP and TCP) on server `192.168.0.200`, using commands `iperf -s` for TCP and `iperf -s -p 5000 -u` for UDP. The clients then connected to the server's listeners using commands `iperf -c 192.168.0.200` and `iperf -c 192.168.0.200 -p 5000 -u -b 1200M`. The results of this test for the clients are shown in *Figure 3, 4, 5 and 6*. The results of this test for the server are shown in *Figure 7 and 8*.

```
root@bristol:~# iperf -c 192.168.0.200 -p 5000 -u -b 1200M
------------------------------------------------------------
Client connecting to 192.168.0.200, UDP port 5000
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.0.69 port 52433 connected with 192.168.0.200 port 5000
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec   831 MBytes   697 Mbits/sec
[  3] Sent 592530 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec   635 MBytes   533 Mbits/sec   0.046 ms 139422/592529 (
24%)
[  3]  0.0-10.0 sec  1 datagrams received out-of-order
```

Figure 3: Scenario 1 - UDP Test 1

```
root@bristol:~# iperf -c 192.168.0.200
------------------------------------------------------------
Client connecting to 192.168.0.200, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.0.69 port 44740 connected with 192.168.0.200 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-11.3 sec   197 MBytes   146 Mbits/sec
```

Figure 4: Scenario 1 - TCP Test 1

```
root@dublin:~# iperf -c 192.168.0.200
------------------------------------------------------------
Client connecting to 192.168.0.200, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.0.3 port 58758 connected with 192.168.0.200 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  70.4 MBytes  59.0 Mbits/sec
root@dublin:~#
```

Figure 5: Scenario 1 - UDP Test 2

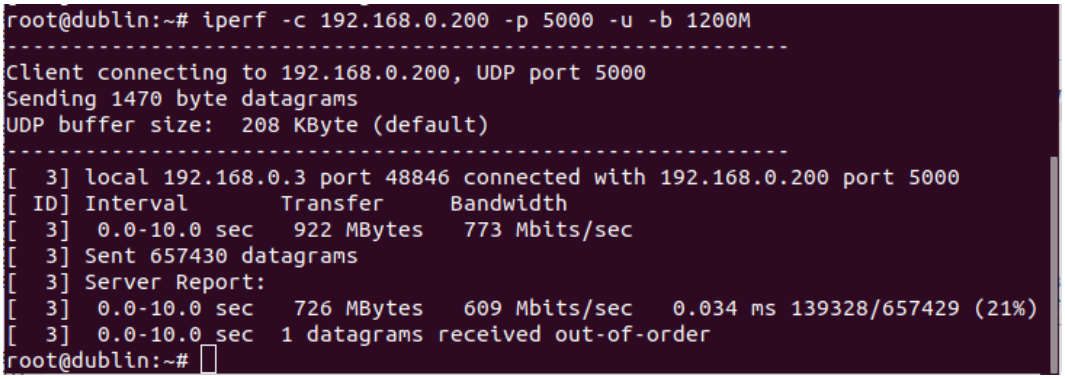

Figure 6: Scenario 1 - TCP Test 2

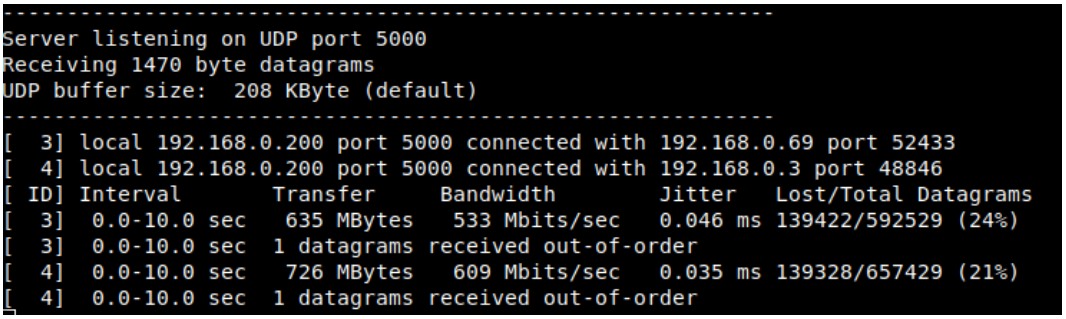

Figure 7: Scenario 1 - UDP Test result

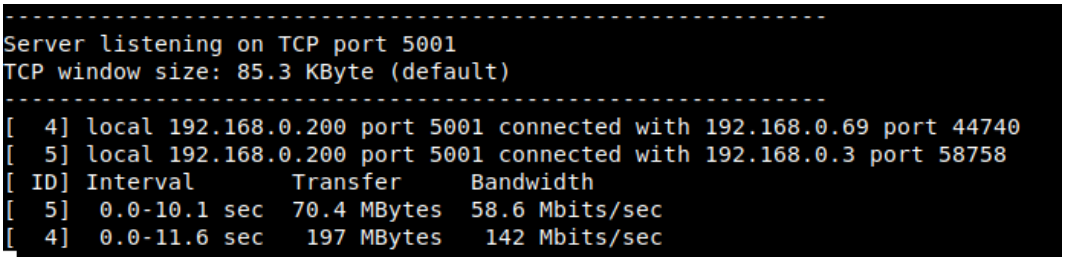

Figure 8: Scenario 1 - TCP Test result

From the results we can derive that there are different results for UDP and TCP traffic. According to *Figure 7 and 8* the UDP traffic claims more bandwidth than the TCP traffic, resulting in the TCP traffic being pushed away. This can be (partially) solved using prioritisation of traffic.

Execution of the second scenario

In the second scenario we have set up an `iperf` server on `192.168.0.200`. We started a transfer on TCP towards the server in `iperf` from one client using `iperf -c 192.168.0.200`. At the same time, another pair of clients start a ping with large packet sizes using command `ping 192.168.0.200 -i 0.01 -s 65507`. The results for the `iperf` client are shown in *Figure 9*. The results of the ping clients are shown in *Figures 10 and 11*. The results of the `iperf` server is shown in *Figure 12*.

```
root@bristol:~# iperf -c 192.168.0.200
------------------------------------------------------------
Client connecting to 192.168.0.200, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.0.69 port 44742 connected with 192.168.0.200 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1013 MBytes   849 Mbits/sec
```

Figure 9: Scenario 2 - TCP Iperf

```
65515 bytes from 192.168.0.200: icmp_seq=3439 ttl=64 time=1.37 ms
65515 bytes from 192.168.0.200: icmp_seq=3440 ttl=64 time=1.34 ms
65515 bytes from 192.168.0.200: icmp_seq=3441 ttl=64 time=1.34 ms
65515 bytes from 192.168.0.200: icmp_seq=3442 ttl=64 time=1.35 ms
^C
--- 192.168.0.200 ping statistics ---
3442 packets transmitted, 3293 received, 4% packet loss, time 35157ms
rtt min/avg/max/mdev = 1.306/1.567/2.585/0.373 ms
root@dublin:~#
```

Figure 10: Scenario 2 - Large packet 1

```
~ # ping -i 0.01 -s 65507 192.168.0.200
PING 192.168.0.200 (192.168.0.200) 65507(65535) bytes of data.
65515 bytes from 192.168.0.200: icmp_seq=1 ttl=64 time=1.36 ms
65515 bytes from 192.168.0.200: icmp_seq=2 ttl=64 time=1.36 ms
65515 bytes from 192.168.0.200: icmp_seq=3 ttl=64 time=1.33 ms
65515 bytes from 192.168.0.200: icmp_seq=4 ttl=64 time=1.32 ms
65515 bytes from 192.168.0.200: icmp_seq=5 ttl=64 time=1.37 ms
65515 bytes from 192.168.0.200: icmp_seq=6 ttl=64 time=1.34 ms
65515 bytes from 192.168.0.200: icmp_seq=7 ttl=64 time=1.35 ms
```

Figure 11: Scenario 2 - Large packet 2

```
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 192.168.0.200 port 5001 connected with 192.168.0.69 port 44742
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.2 sec  1013 MBytes   832 Mbits/sec
```

Figure 12: Scenario 2 - Result

In *Figure 1* we have shown the normal throughput for TCP-traffic between an `iperf` server and client. Because of the ping request sending large sized packets with a high frequency, approximately 110 Mbits/sec was allocated to the ICMP-traffic.

Execution of the third scenario

The third scenario will include an `Apache` webserver along with an `iperf` server both running on `192.168.0.200`. One client will communicate with the server using `iperf -c 192.168.0.200` and other two nodes will use a benchmarking tool like apache benchmark (ab) with the following command `b -n 100000 -c 10 http://192.168.0.200`. The results of the `iperf` client are shown in *Figure 17*. The results of the Apache benchmark clients are shown in *Figures 14 and 15*. The results of the server are shown in *Figure 16*.

```
root@bristol:~# iperf -c 192.168.0.200
------------------------------------------------------------
Client connecting to 192.168.0.200, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.0.69 port 44746 connected with 192.168.0.200 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec   349 MBytes   292 Mbits/sec
```

Figure 13: Scenario 2 - TCP Iperf

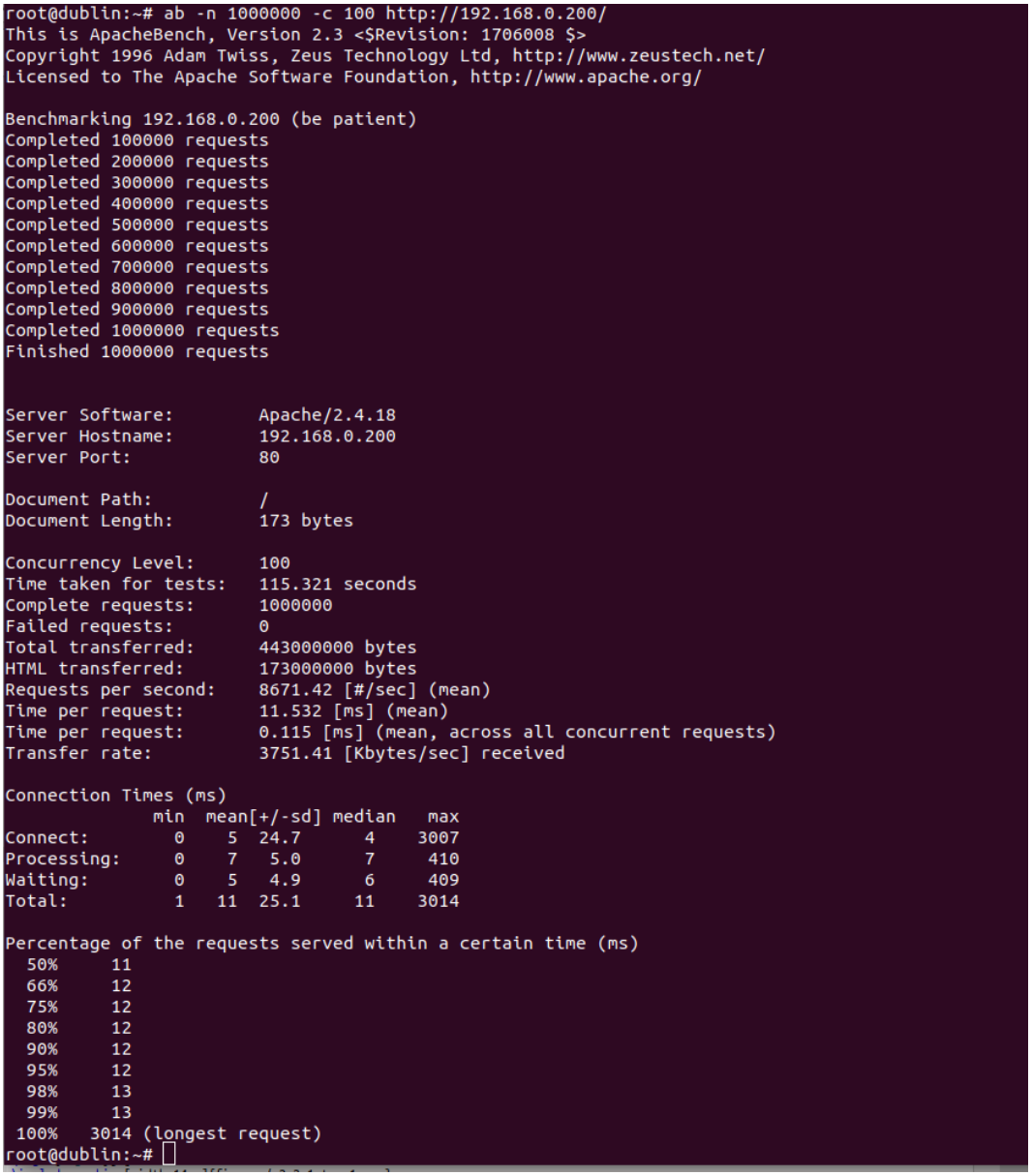

Figure 14: Scenario 3 - Apache Benchmark from Dublin client server to Apache server

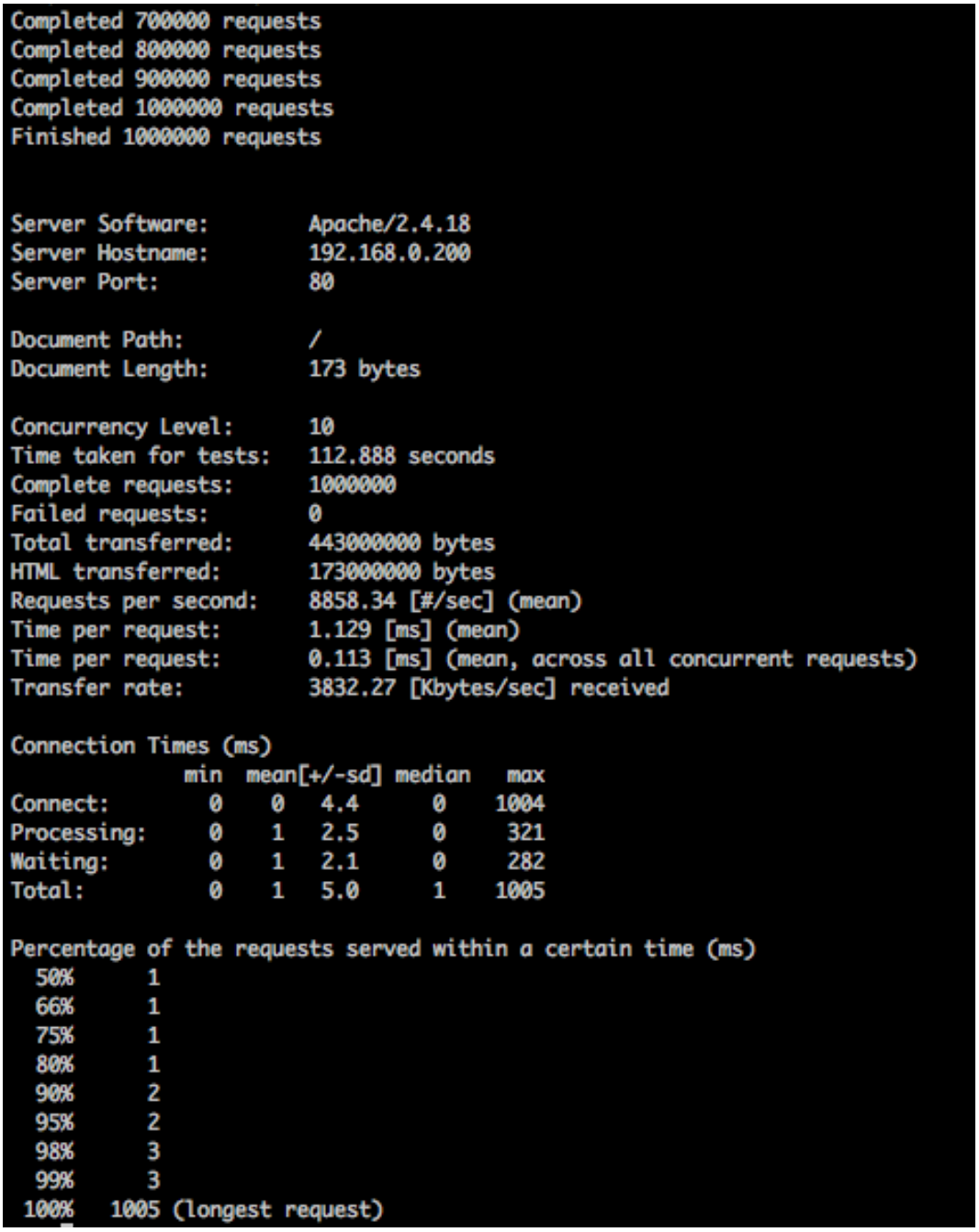

Figure 15: Scenario 3 - Apache Benchmark from client server to Apache server

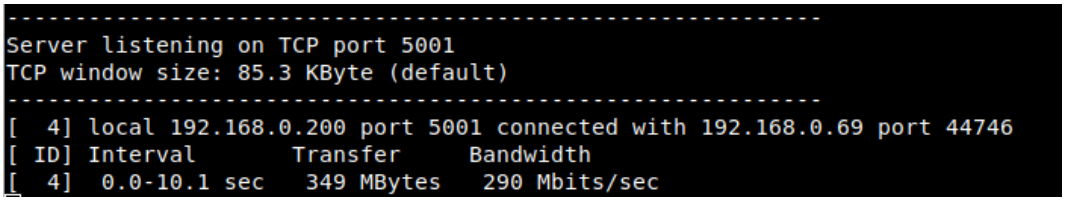

Figure 16: Scenario 2 - Result

From these results we can derive the Apache Benchmark traffic's bandwidth allocation is much higher than the allocation of traffic to the `iperf` transfer. Again, this can be regulated by giving HTTP-traffic a lower priority than `iperf` traffic.

Q3 Configuring CoS

In order to configure CoS, it has to be enabled on the interfaces of the switch where the servers are connected to. First CoS has to be enabled globally on the switch by performing the following configuration steps on the switch:

```
conf t
mls qos
```

We will prioritize two interfaces distributing the interfering traffic (Apache Benchmark) with a lower priority than the other interfaces. This is done by performing the following configuration steps on the switch:

```
conf t
int G1/0/17
mls qos cos 6
mls qos trust cos
int G1/0/19
mls qos cos 6
mls qos trust cos
int G1/0/21
mls qos trust cos
int G1/0/23
mls qos trust cos
```

The impact is shown when we run scenario 3 again in *Figures 17, 19 and 18*.

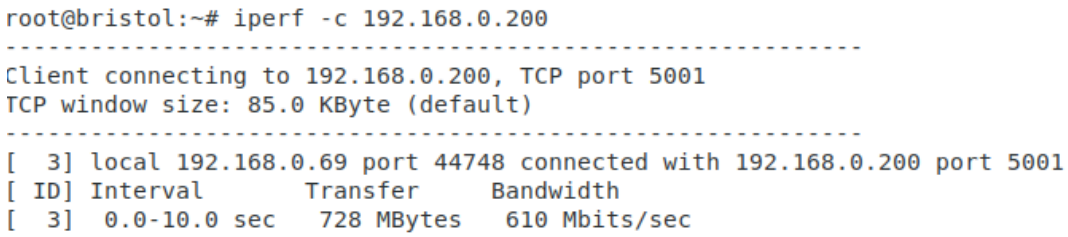

Figure 17: Cos - TCP Iperf

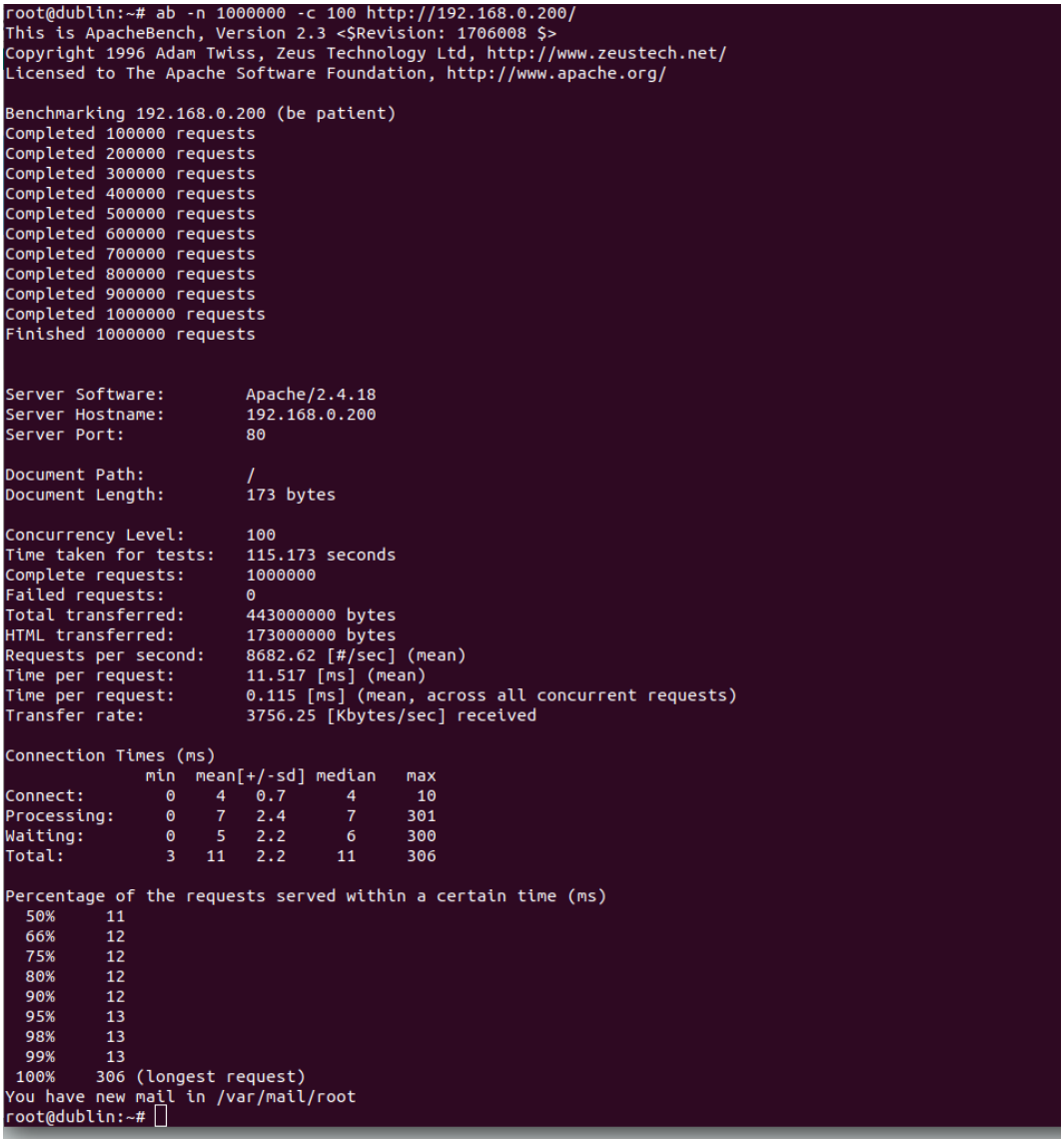

Figure 18: Cos - Apache Benchmark from Dublin client to Apache server with CoS enabled

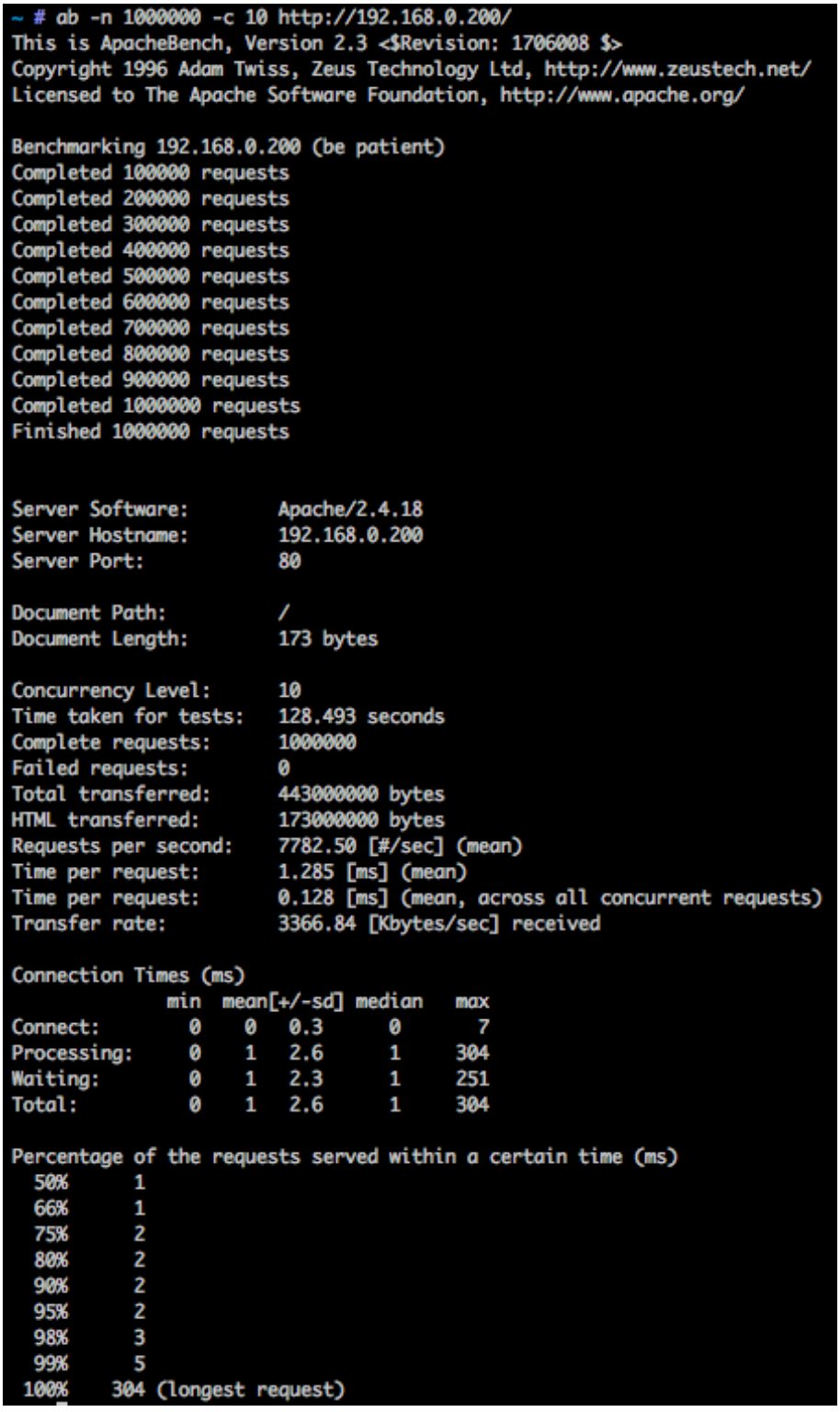

Figure 19: Cos - Apache Benchmark from client to Apache server with CoS enabled

Now the TCP-transfer through `iperf` has much more bandwidth allocated, meaning the prioritisation of the ports appears to work. In order to confirm the results, this test was run twice.

Difference between SRR shape and share

Configuration for shape:

```
conf t
int G1/0/19
srr-queue bandwidth shape 9830 9830 9830 9830
int G1/0/17
srr-queue bandwidth shape 9830 9830 9830 9830
int G1/0/23
srr-queue bandwidth shape 45000 45000 45000 45000
```

Configuration for share:

```
conf t
int G1/0/19
srr-queue bandwidth share 15 15 15 15
int G1/0/17
srr-queue bandwidth share 15 15 15 15
int G1/0/23
srr-queue bandwidth share 70 70 70 70
```

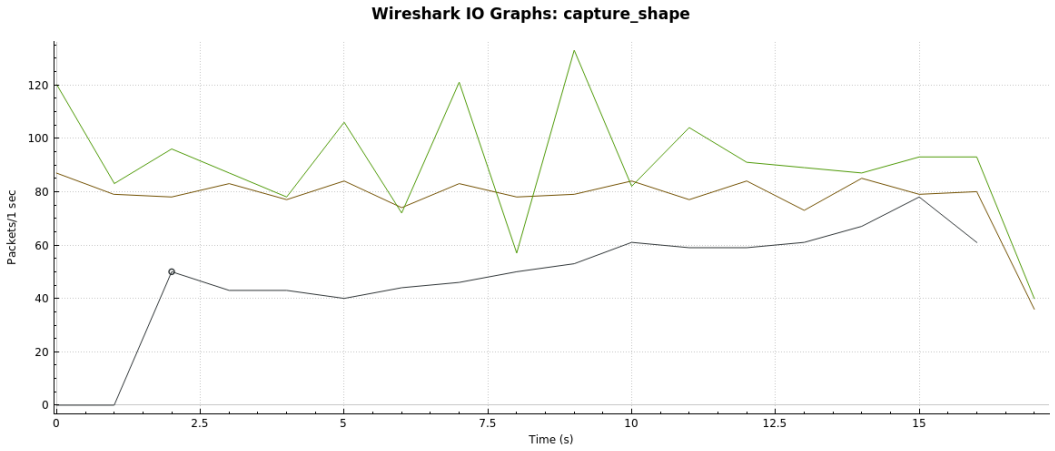

Figure 20: Packets per second sent after shape configuration

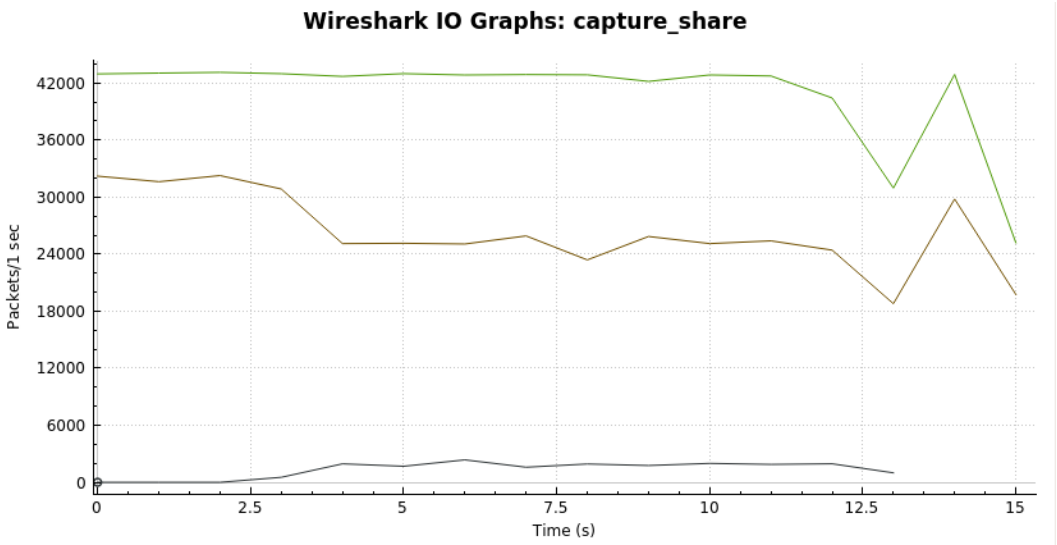

Figure 21: Packets per second sent after share configuration

Given the fact that the framesize of the `iperf` traffic (darkest line) varies from 5858 to 40610 bytes and the framesize of the HTTP-traffic (other two lines) vary from 66 to 147 bytes, the results shown in *Figures 20 and 21* are correct. Both images show the packets sent per second, which should, according to the configured share and shape, result in a higher throughput for the `iperf` traffic. The throughput can be calculated using $Throughput_{traffic} = Packets/s_{traffic} * Framesize$ for every point per traffic type. As the size of the frames for the `iperf` traffic is much higher, the throughput is also higher which displays the queue share and shape values being applied.

Comparing both figures (21 and 20, the difference between sharing and shaping is displayed. With shaping, the throughput is limited to a configured value. With sharing, the traffic gets a guaranteed share of a queue, but is not limited to a value when extra bandwidth is available. We can see this based on the increase in *Figure 21*, when the `iperf` traffic line decreases. In *Figure 20* we see the amount of packets sent per second bumping into a limit and then bouncing back. The value does not become higher than the limit.

Q4.1 Mark your outgoing packets using iptables

One of the characteristics of DSCP is that the classification of the packets can occur at the source (sender of specific traffic). The marking of the packets can be done using iptables, which is simply separating them into classes based on the protocol or ports used.

The rule we have configured is for HTTP traffic (We run it on port 5000):

```
sudo iptables -t mangle -A OUTPUT -p tcp --dport 5000 -j DSCP --set-dscp-class cs3
```

The result of the following command shows the value set in the DSCP field:

```
iptables -t mangle -L -v
```

The result is:

```
Chain OUTPUT (policy ACCEPT 1217 packets, 202K bytes)
```

```
pkts bytes target     prot opt in     out     source               destination
   1    40 DSCP       tcp  --  any    any     anywhere             anywhere        \
   tcp dpt:5000 DSCP set 0x18
```

So, in the next question we would see HTTP traffic in our tcpdump to be marked with 0x18. In Wireshark the ToS-value is shown. According to https://www.tucny.com/Home/dscp-tos, the ToS value for DSCP 0x18 is 0x60.

Q4.2 Show that the packets are marked

In order to show the packets are marked, a `tcpdump` was started on one of the servers. The resulting PCAP-file was analysed using Wireshark. To check whether DSCP was applied to packets, we searched through the packets using filter `ip.dsfield.dscp`. The DSCP-value was retrieved from the `Differentiated Services Field` and is shown in *Figure 27*.

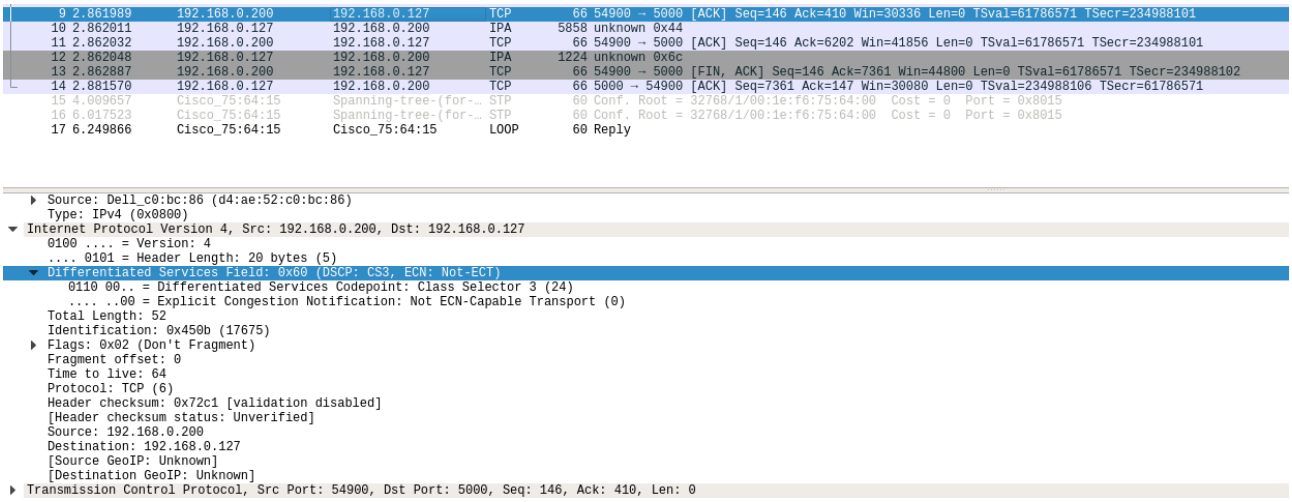

Figure 22: DSCP marker 0x18 (ToS: 0x60)

Q4.3 Show how DSCP marking improves traffic performance.

In order to allow DSCP marking, we configured all ports of the switch to trust DSCP markings on ports. We did this using command:

```
mls qos trust dscp
```

We also configured the default DSCP-CoS mapping:

```
mls qos map cos-dscp 10 15 20 25 30 35 40 45
```

The results of the baseline tests are shown in *Figures 23 and 24*. The figures show the bandwidth share between an `iperf` connection and an HTTP-connection. The `iperf` connection appears to be much faster. However, after configuring DSCP we did not see much improvement (only 2 Mbps in the iperf traffic). This is shown in *Figures 25 and 26*.

```
root@dublin:~# ab -n 1000000 -c 100 http://192.168.0.200/
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.0.200 (be patient)
Completed 100000 requests
Completed 200000 requests
Completed 300000 requests
Completed 400000 requests
Completed 500000 requests
Completed 600000 requests
Completed 700000 requests
Completed 800000 requests
Completed 900000 requests
Completed 1000000 requests
Finished 1000000 requests


Server Software:        Apache/2.4.18
Server Hostname:        192.168.0.200
Server Port:            80

Document Path:          /
Document Length:        173 bytes

Concurrency Level:      100
Time taken for tests:   117.932 seconds
Complete requests:      1000000
Failed requests:        0
Total transferred:      443000000 bytes
HTML transferred:       173000000 bytes
Requests per second:    8479.46 [#/sec] (mean)
Time per request:       11.793 [ms] (mean)
Time per request:       0.118 [ms] (mean, across all concurrent requests)
Transfer rate:          3668.36 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    5   5.4      5    1005
Processing:     0    7   1.5      7     209
Waiting:        0    5   1.7      6     209
Total:          1   12   5.5     12    1012

Percentage of the requests served within a certain time (ms)
  50%     12
  66%     12
  75%     12
  80%     12
  90%     13
  95%     13
  98%     13
  99%     13
 100%   1012 (longest request)
```

Figure 23: AB without DSCP enabled

```
root@dublin:~# iperf -c 192.168.0.200
------------------------------------------------------------
Client connecting to 192.168.0.200, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.0.3 port 49272 connected with 192.168.0.200 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.06 GBytes   907 Mbits/sec
```

Figure 24: iperf without DSCP enabled

```
root@dublin:~# ab -n 1000000 -c 100 http://192.168.0.200/
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.0.200 (be patient)
Completed 100000 requests
Completed 200000 requests
Completed 300000 requests
Completed 400000 requests
Completed 500000 requests
Completed 600000 requests
Completed 700000 requests
Completed 800000 requests
Completed 900000 requests
Completed 1000000 requests
Finished 1000000 requests


Server Software:        Apache/2.4.18
Server Hostname:        192.168.0.200
Server Port:            80

Document Path:          /
Document Length:        173 bytes

Concurrency Level:      100
Time taken for tests:   119.201 seconds
Complete requests:      1000000
Failed requests:        0
Total transferred:      443000000 bytes
HTML transferred:       173000000 bytes
Requests per second:    8389.22 [#/sec] (mean)
Time per request:       11.920 [ms] (mean)
Time per request:       0.119 [ms] (mean, across all concurrent requests)
Transfer rate:          3629.32 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    5   4.4      5    1005
Processing:     0    7   1.4      7     213
Waiting:        0    6   1.5      6     211
Total:          2   12   4.5     12    1012

Percentage of the requests served within a certain time (ms)
  50%     12
  66%     12
  75%     12
  80%     12
  90%     13
  95%     13
  98%     13
  99%     14
 100%   1012 (longest request)
```

Figure 25: AB with DSCP enabled

```
root@dublin:~# iperf -c 192.168.0.200
------------------------------------------------------------
Client connecting to 192.168.0.200, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.0.3 port 43894 connected with 192.168.0.200 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.06 GBytes   909 Mbits/sec
```

Figure 26: iperf with DSCP enabled

Q5.1 Enable logging of events (syslog) to an external server.

By default, syslog is enabled on cisco switches by default. We just need to forward the log to our server. The following command will do that:

```
gr3switch(config)#logging host 192.168.0.69
gr3switch(config)#snmp-server enable traps syslog

Follow the procedure given by this website to \textit{configure remote logging}:
https://du5tin.com/2017/06/10/set-up-syslog-server-in-ubuntu-server-16-04-w-log-rotation/

/var/log # tail -n2 /var/log/remote/192.168.0.253/syslog.log
2018-02-22T18:03:56.342718+01:00 192.168.0.253 183: *Mar  3 05:03:57.338: %SYS-5-CONFIG_I: Configured fi
2018-02-22T18:04:42.496568+01:00 192.168.0.253 184: *Mar  3 05:04:43.550: %SYS-5-CONFIG_I: Configured fi
```

Q5.2 Monitor your switch so that its statistics can be viewed online. This requires to poll via SNMP.

We chose for monitoring NetDisco.

Following the installation process of the Perl/Postgres/Netdisco Stack is fairly simple following Netdisco's instructions.

https://metacpan.org/pod/App::Netdisco

```
    netdisco@calais:~\$ ~/bin/netdisco-backend start
Netdisco Backend                                          [Started]
config watcher: watching /home/netdisco/environments for updates.
```

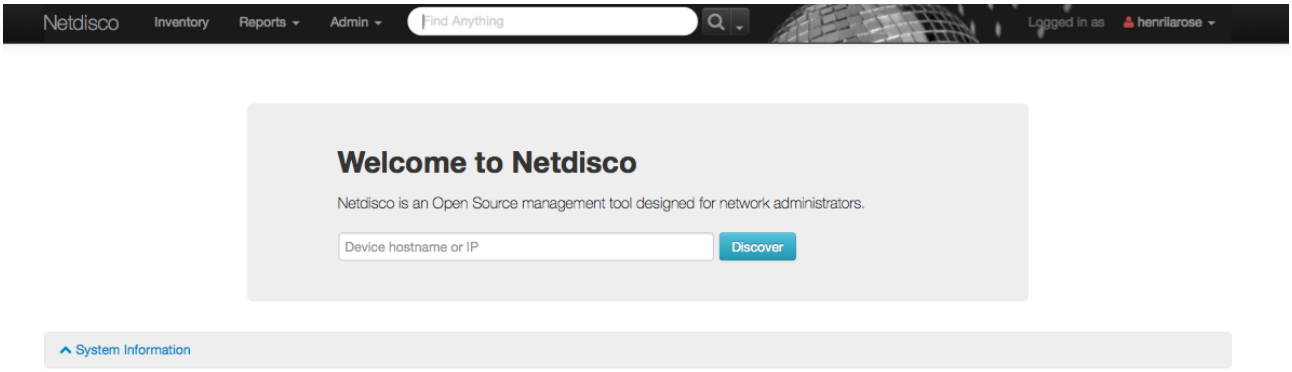

Figure 27: Netdisco network manager Web-Server

However, we can't figure out how to make the web-server receive the SNMP syslog traps from the switch. Thus, Netdisco cannot discover the devices on the server.

Switch SNMP configurations:

```
    snmp-server community an-cos RO
snmp-server community public RO
snmp-server community private RW
snmp-server location os3
snmp-server contact henrilarose
snmp-server enable traps syslog
snmp-server host 192.168.0.127 an-cos  snmp
snmp-server host 192.168.0.127 henrilarose  syslog
```

Switch SNMP status:

```
    gr3switch#sh snmp
Chassis: FOC1151Z7ZQ
Contact: henrilarose
Location: os3
4122 SNMP packets input
    0 Bad SNMP version errors
    0 Unknown community name
    0 Illegal operation for community name supplied
    0 Encoding errors
    4115 Number of requested variables
    0 Number of altered variables
    0 Get-request PDUs
    4115 Get-next PDUs
    0 Set-request PDUs
    0 Input queue packet drops (Maximum queue size 1000)
4122 SNMP packets output
    0 Too big errors (Maximum packet size 1500)
    0 No such name errors
    0 Bad values errors
    0 General errors
```

```
    4115 Response PDUs
    0 Trap PDUs
SNMP global trap: disabled

SNMP logging: enabled
    Logging to 192.168.0.127.162, 0/10, 0 sent, 0 dropped.
SNMP agent enabled
```

After an email exchange with Paola, *snmpwalk*, on the server which runs Netdisco, was able to find all devices:

```
    iso.3.6.1.2.1.1.1.0 = STRING: "Cisco IOS Software, C3750 Software (C3750-IPSERVICESK9-M), Version 1
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2015 by Cisco Systems, Inc.
Compiled Wed 11-Feb-15 11:40 by prod_rel_team"
iso.3.6.1.2.1.1.2.0 = OID: iso.3.6.1.4.1.9.1.516
iso.3.6.1.2.1.1.3.0 = Timeticks: (20331806) 2 days, 8:28:38.06
iso.3.6.1.2.1.1.4.0 = ""
iso.3.6.1.2.1.1.5.0 = STRING: "gr3switch.gr3.com"
iso.3.6.1.2.1.1.6.0 = ""
iso.3.6.1.2.1.1.7.0 = INTEGER: 6
iso.3.6.1.2.1.1.8.0 = Timeticks: (0) 0:00:00.00
...
iso.3.6.1.2.1.4.22.1.3.1.192.168.0.3 = IpAddress: 192.168.0.3
iso.3.6.1.2.1.4.22.1.3.1.192.168.0.69 = IpAddress: 192.168.0.69
iso.3.6.1.2.1.4.22.1.3.1.192.168.0.127 = IpAddress: 192.168.0.127
iso.3.6.1.2.1.4.22.1.3.1.192.168.0.200 = IpAddress: 192.168.0.200
iso.3.6.1.2.1.4.22.1.3.1.192.168.0.253 = IpAddress: 192.168.0.253
...
```

However, the Netdisco back-end server does not seem to be able to do that. We were not able to configure it with Ralph either.