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

However, when we change the size of the MTU on interface `eno2`, the throughput increases.

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

After some discussion we could define following scenarios:

1. Example from lab one basic scenario could be the one in which multiple nodes and applications are trying to communicate with one node.

```
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

- 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 are shown in *Figure* **??** and *Figure* **??**.

```
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

```
root@dublin:~# 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 192.168.0.3 port 48846 connected with 192.168.0.200 port 5000
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec   922 MBytes   773 Mbits/sec
[  3] Sent 657430 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec   726 MBytes   609 Mbits/sec   0.034 ms 139328/657429 (21%)
[  3]  0.0-10.0 sec  1 datagrams received out-of-order
root@dublin:~#
```

Figure 6: Scenario 1 - TCP Test 2

```
------------------------------------------------------------
Server listening on UDP port 5000
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.0.200 port 5000 connected with 192.168.0.69 port 52433
[  4] local 192.168.0.200 port 5000 connected with 192.168.0.3 port 48846
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  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
[  4]  0.0-10.0 sec   726 MBytes   609 Mbits/sec   0.035 ms 139328/657429 (21%)
[  4]  0.0-10.0 sec  1 datagrams received out-of-order
```

Figure 7: Scenario 1 - UDP Test result

```
------------------------------------------------------------
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 44740
[  5] local 192.168.0.200 port 5001 connected with 192.168.0.3 port 58758
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.1 sec  70.4 MBytes  58.6 Mbits/sec
[  4]  0.0-11.6 sec   197 MBytes   142 Mbits/sec
```

Figure 8: Scenario 1 - TCP Test result

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 are listed in *Figure...*

```
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

Execution of the third scenario

In the third scenario will include an `Apache` webserver along with an `iperf` server running on `192.168.0.200`. One client will communicate with the server using `iperf -c 192.168.0.200` and the 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`

```
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

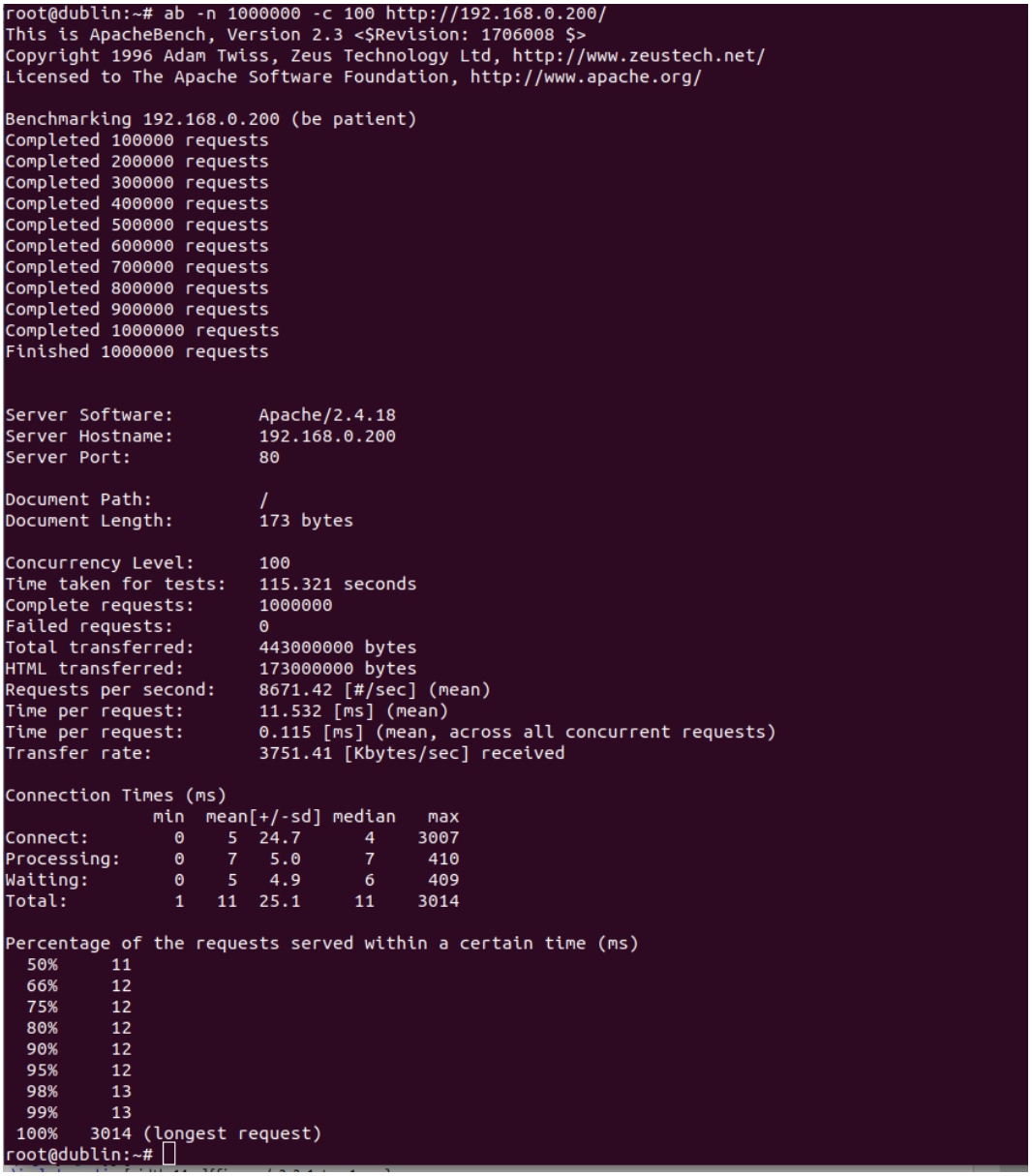

Figure 14: Scenario 3 - Apache Benchmark from Dublin client server to Apache server

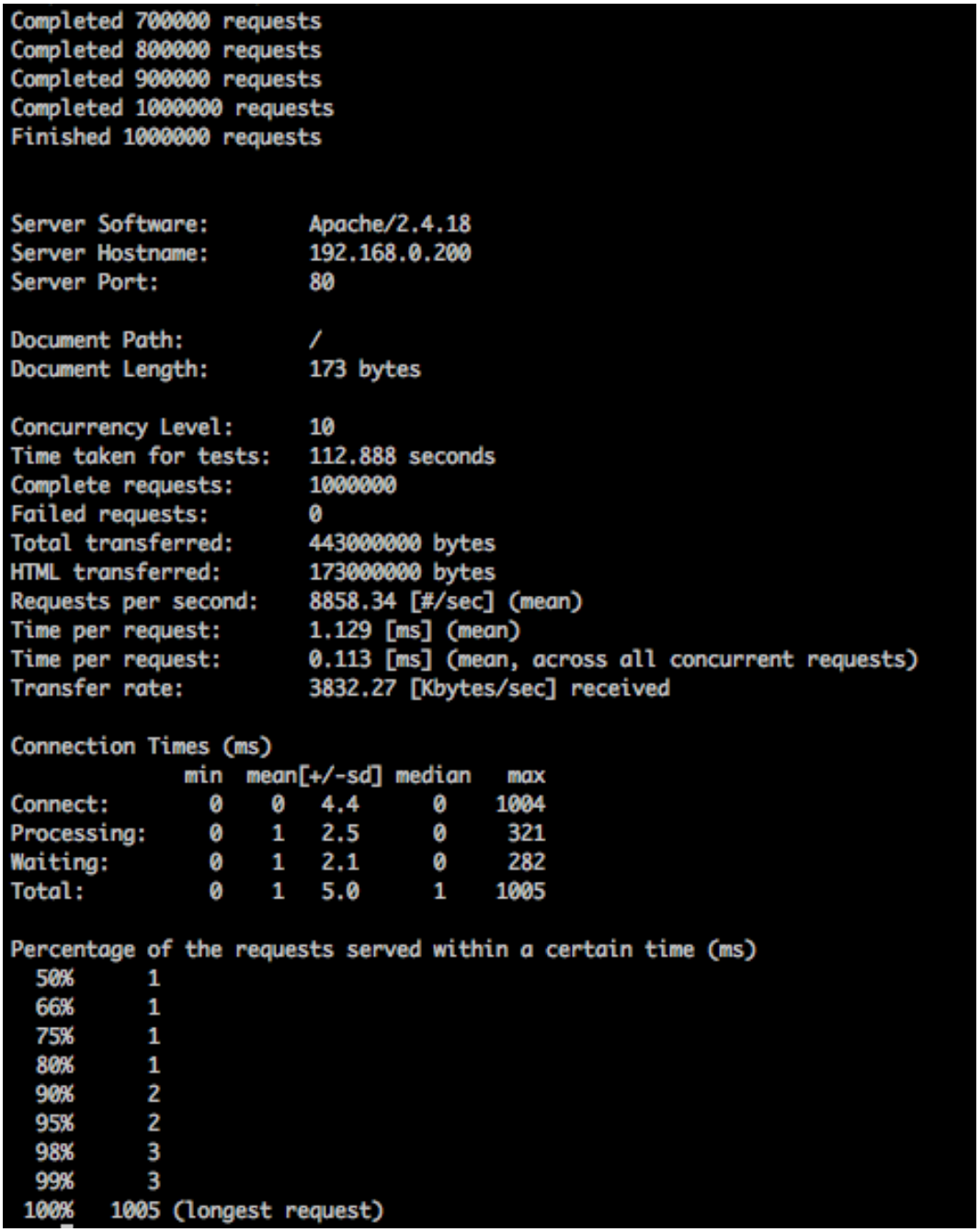

Figure 15: Scenario 3 - Apache Benchmark from client server to Apache server

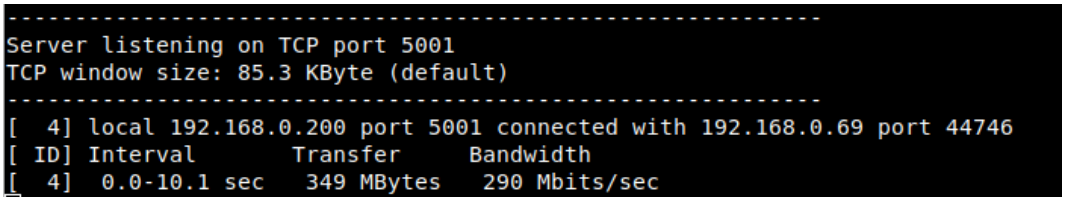

Figure 16: Scenario 2 - Result

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.

```
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 44748 connected with 192.168.0.200 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec   728 MBytes   610 Mbits/sec
```

Figure 17: Cos - TCP Iperf

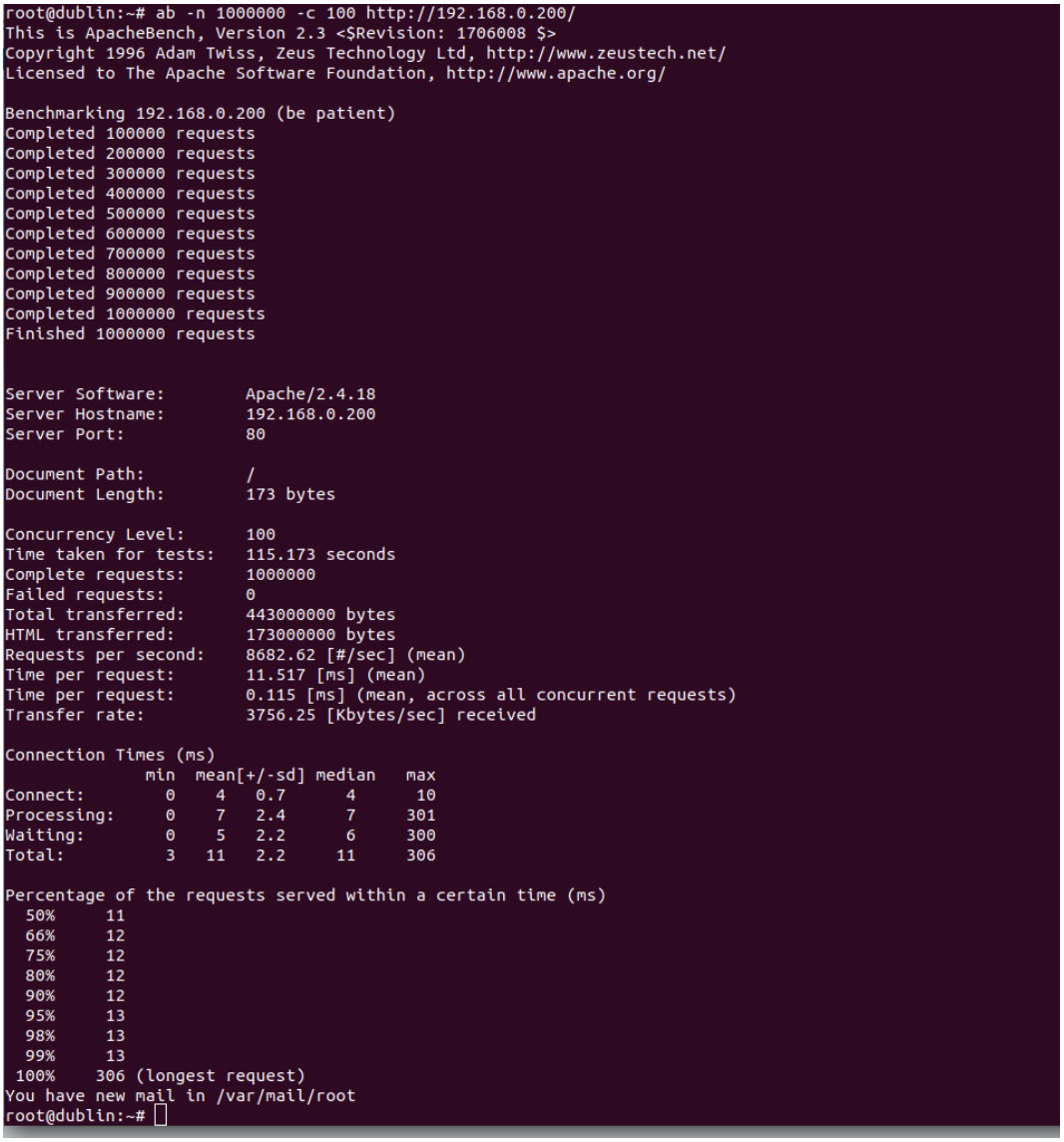

Figure 18: Cos - Apache Benchmark from Dublin client to Apache server with CoS enabled

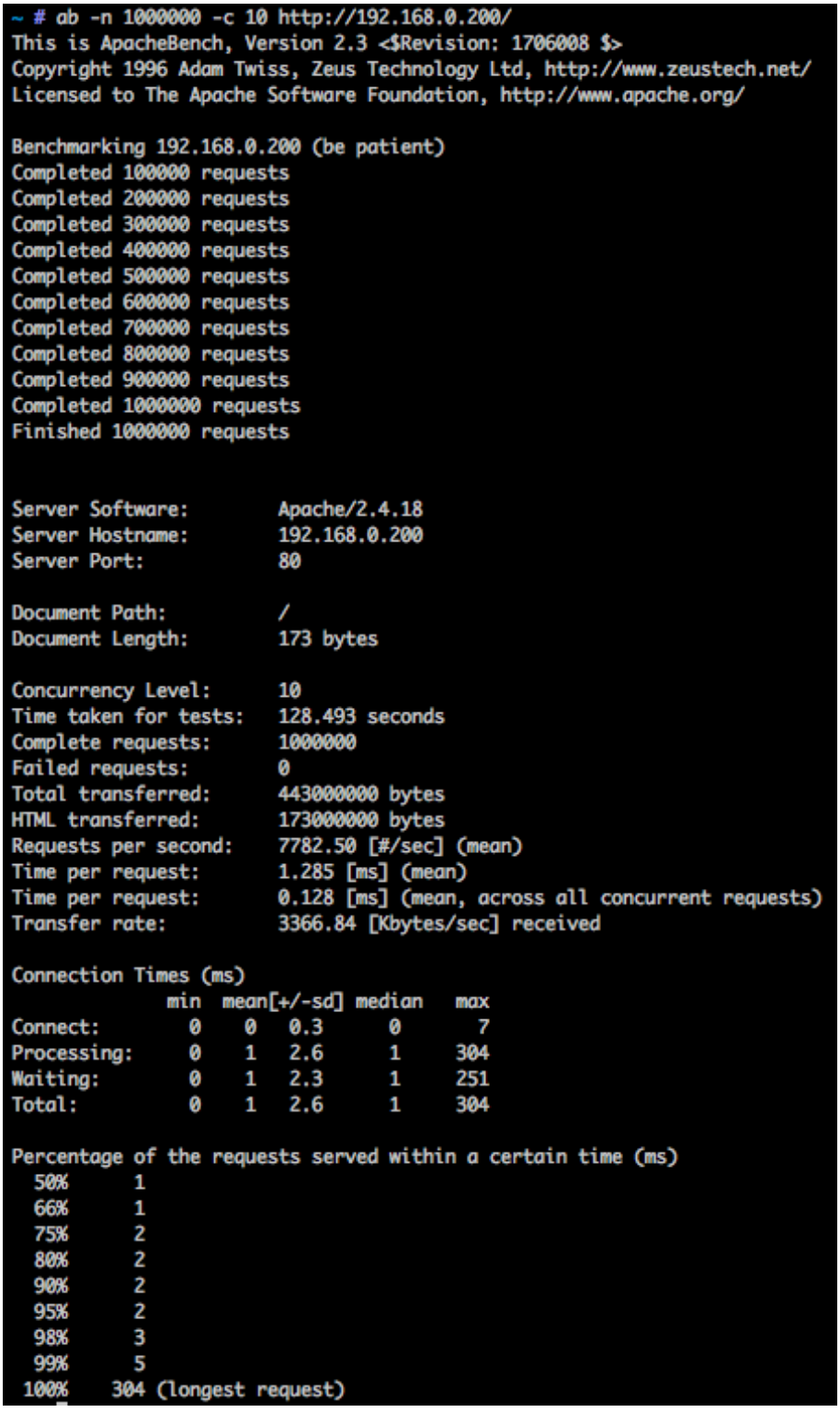

Figure 19: Cos - Apache Benchmark from client to Apache server with CoS enabled

Difference between SRR and other

```
    gr3switch(config)#int G1/0/19
gr3switch(config-if)#no srr-
gr3switch(config-if)#no srr-queue bandwidth share 15 15 15 15
gr3switch(config-if)#int G1/0/17
gr3switch(config-if)#no srr-queue bandwidth share 15 15 15 15
gr3switch(config-if)#int G1/0/23
gr3switch(config-if)#no srr-queue
% Incomplete command.

gr3switch(config-if)#no srr-queue bandwidth
% Incomplete command.

gr3switch(config-if)#no srr-queue bandwidth share
gr3switch(config-if)#srr-queue bandwidth shape ?
  <0-65535>  enter bandwidth weight for queue id 1

gr3switch(config-if)#srr-queue bandwidth shape 45000 45000 45000 45000
gr3switch(config-if)#int G1/0/19
gr3switch(config-if)#srr-queue bandwidth shape 9830 9830 9830 9830
gr3switch(config-if)#int G1/0/17
gr3switch(config-if)#srr-queue bandwidth shape 9830 9830 9830 9830
```

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.

Example rule (has to be fixed by actual rules, but depends on task 2):

```
iptables -t mangle -A OUTPUT -p udp -m udp --sport 5060 -j DSCP --set-dscp-class cs3
```

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 ...

Q4.3 Show how DSCP marking improves traffic performance.

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

By default, syslog is enabled in 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
```

```
Now inside /etc/rsyslog.conf
# for cisco
local7.*        /var/log/cisco-messages.log
```