Andre Hei Wang Law

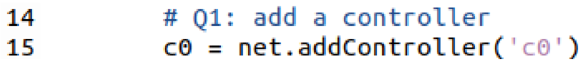
4017 5600

COEN 3166 – FL-X

Mininet Assignment 2 + Wireshark Assignment 3

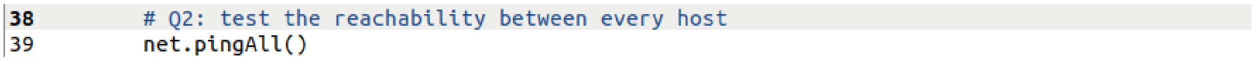
1. Mininet Assignment 2

1. Add a controller ‘c0’ in your topology.

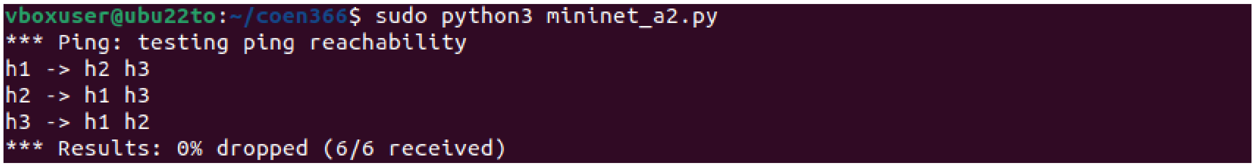


```
# Q1: add a controller
c0 = net.addController('c0')
```

2. Test the reachability between every host using pingall (take screenshot)

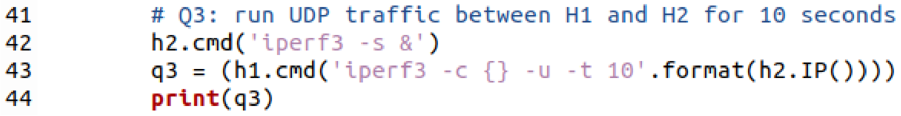


```
# Q2: test the reachability between every host
net.pingAll()
```

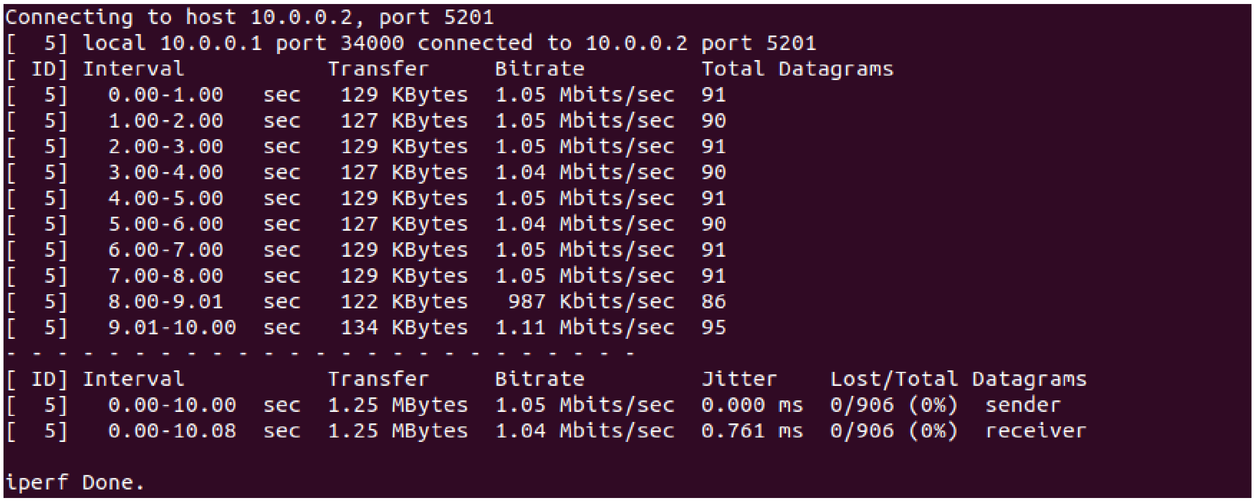


```
vboxuser@ubu22to:~/coen366$ sudo python3 mininet_a2.py
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
```

3. Run UDP traffic for 10 seconds between H1 (client) and H2 (server). Provide the iperf commands and the result (take screenshot).

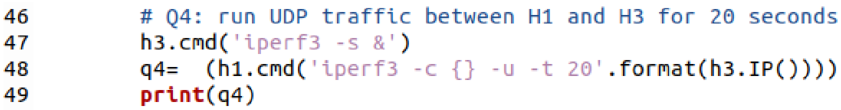


```
# Q3: run UDP traffic between H1 and H2 for 10 seconds
h2.cmd('iperf3 -s &')
q3 = (h1.cmd('iperf3 -c {} -u -t 10'.format(h2.IP())))
print(q3)
```

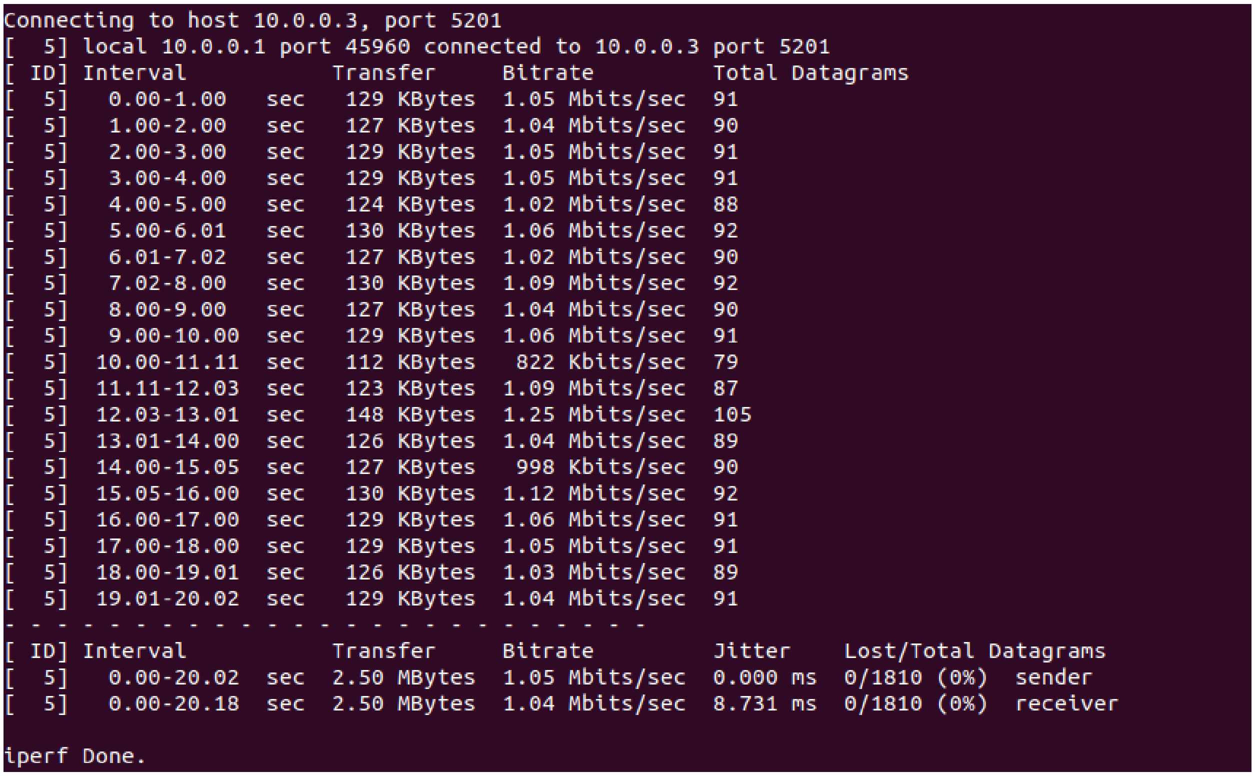


```
Connecting to host 10.0.0.2, port 5201
[  5] local 10.0.0.1 port 34000 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[  5]   0.00-1.00   sec   129 KBytes  1.05 Mbits/sec  91
[  5]   1.00-2.00   sec   127 KBytes  1.05 Mbits/sec  90
[  5]   2.00-3.00   sec   129 KBytes  1.05 Mbits/sec  91
[  5]   3.00-4.00   sec   127 KBytes  1.04 Mbits/sec  90
[  5]   4.00-5.00   sec   129 KBytes  1.05 Mbits/sec  91
[  5]   5.00-6.00   sec   127 KBytes  1.04 Mbits/sec  90
[  5]   6.00-7.00   sec   129 KBytes  1.05 Mbits/sec  91
[  5]   7.00-8.00   sec   129 KBytes  1.05 Mbits/sec  91
[  5]   8.00-9.01   sec   122 KBytes   987 Kbits/sec  86
[  5]   9.01-10.00  sec   134 KBytes  1.11 Mbits/sec  95
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  5]   0.00-10.00  sec  1.25 MBytes  1.05 Mbits/sec  0.000 ms  0/906 (0%)  sender
[  5]   0.00-10.08  sec  1.25 MBytes  1.04 Mbits/sec  0.761 ms  0/906 (0%)  receiver

iperf Done.
```

4. Run UDP traffic for 20 seconds between H1 (client) and H3 (server). Provide the iperf commands and the result (take screenshot).

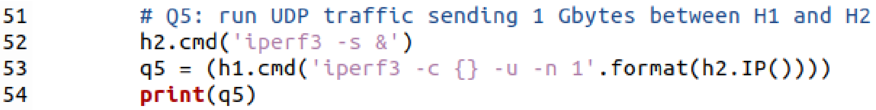


```
        # Q4: run UDP traffic between H1 and H3 for 20 seconds
        h3.cmd('iperf3 -s &')
        q4=  (h1.cmd('iperf3 -c {} -u -t 20'.format(h3.IP())))
        print(q4)
```

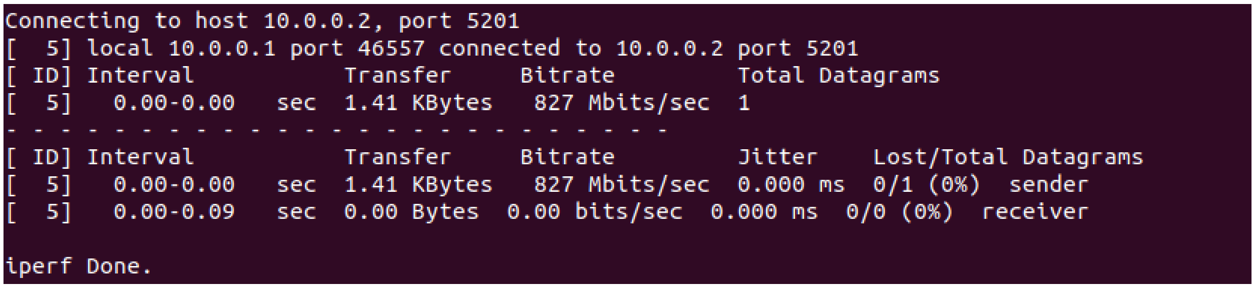


```
Connecting to host 10.0.0.3, port 5201
[  5] local 10.0.0.1 port 45960 connected to 10.0.0.3 port 5201
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[  5]   0.00-1.00   sec   129 KBytes  1.05 Mbits/sec  91
[  5]   1.00-2.00   sec   127 KBytes  1.04 Mbits/sec  90
[  5]   2.00-3.00   sec   129 KBytes  1.05 Mbits/sec  91
[  5]   3.00-4.00   sec   129 KBytes  1.05 Mbits/sec  91
[  5]   4.00-5.00   sec   124 KBytes  1.02 Mbits/sec  88
[  5]   5.00-6.01   sec   130 KBytes  1.06 Mbits/sec  92
[  5]   6.01-7.02   sec   127 KBytes  1.02 Mbits/sec  90
[  5]   7.02-8.00   sec   130 KBytes  1.09 Mbits/sec  92
[  5]   8.00-9.00   sec   127 KBytes  1.04 Mbits/sec  90
[  5]   9.00-10.00  sec   129 KBytes  1.06 Mbits/sec  91
[  5]  10.00-11.11  sec   112 KBytes   822 Kbits/sec  79
[  5]  11.11-12.03  sec   123 KBytes  1.09 Mbits/sec  87
[  5]  12.03-13.01  sec   148 KBytes  1.25 Mbits/sec  105
[  5]  13.01-14.00  sec   126 KBytes  1.04 Mbits/sec  89
[  5]  14.00-15.05  sec   127 KBytes   998 Kbits/sec  90
[  5]  15.05-16.00  sec   130 KBytes  1.12 Mbits/sec  92
[  5]  16.00-17.00  sec   129 KBytes  1.06 Mbits/sec  91
[  5]  17.00-18.00  sec   129 KBytes  1.05 Mbits/sec  91
[  5]  18.00-19.01  sec   126 KBytes  1.03 Mbits/sec  89
[  5]  19.01-20.02  sec   129 KBytes  1.04 Mbits/sec  91
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  5]   0.00-20.02  sec  2.50 MBytes  1.05 Mbits/sec  0.000 ms  0/1810 (0%)  sender
[  5]   0.00-20.18  sec  2.50 MBytes  1.04 Mbits/sec  8.731 ms  0/1810 (0%)  receiver

iperf Done.
```

5. Run UDP traffic that sends 1 Gbytes between H1(client) and H2 (server). Provide the iperf commands and the result (take screenshot).

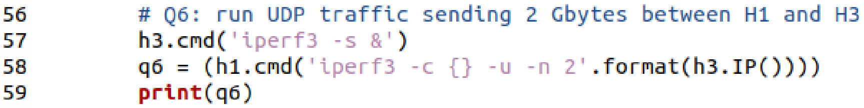


```
        # Q5: run UDP traffic sending 1 Gbytes between H1 and H2
        h2.cmd('iperf3 -s &')
        q5 = (h1.cmd('iperf3 -c {} -u -n 1'.format(h2.IP())))
        print(q5)
```

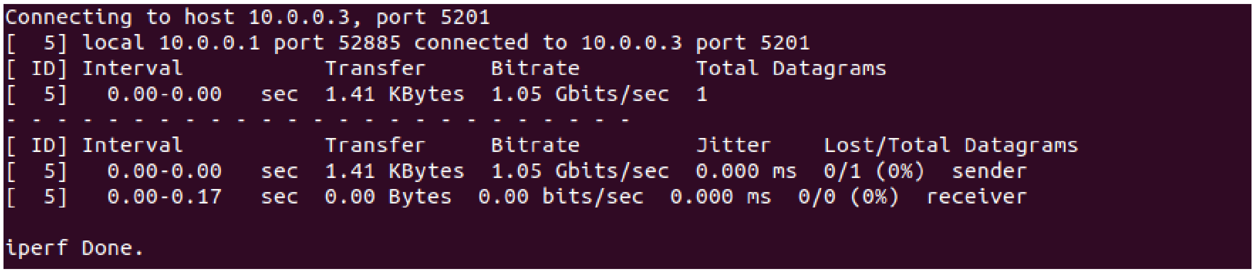


```
Connecting to host 10.0.0.2, port 5201
[  5] local 10.0.0.1 port 46557 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[  5]   0.00-0.00   sec  1.41 KBytes   827 Mbits/sec  1
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  5]   0.00-0.00   sec  1.41 KBytes   827 Mbits/sec  0.000 ms  0/1 (0%)  sender
[  5]   0.00-0.09   sec  0.00 Bytes  0.00 bits/sec  0.000 ms  0/0 (0%)  receiver

iperf Done.
```

6. Run UDP traffic that sends 2 Gbytes between H1 (client) and H3 (server). Provide the iperf commands and the result (take screenshot).

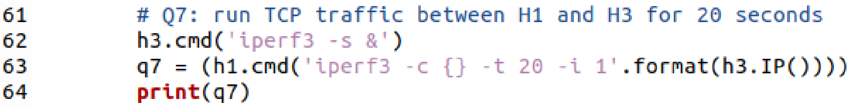


```
        # Q6: run UDP traffic sending 2 Gbytes between H1 and H3
        h3.cmd('iperf3 -s &')
        q6 = (h1.cmd('iperf3 -c {} -u -n 2'.format(h3.IP())))
        print(q6)
```

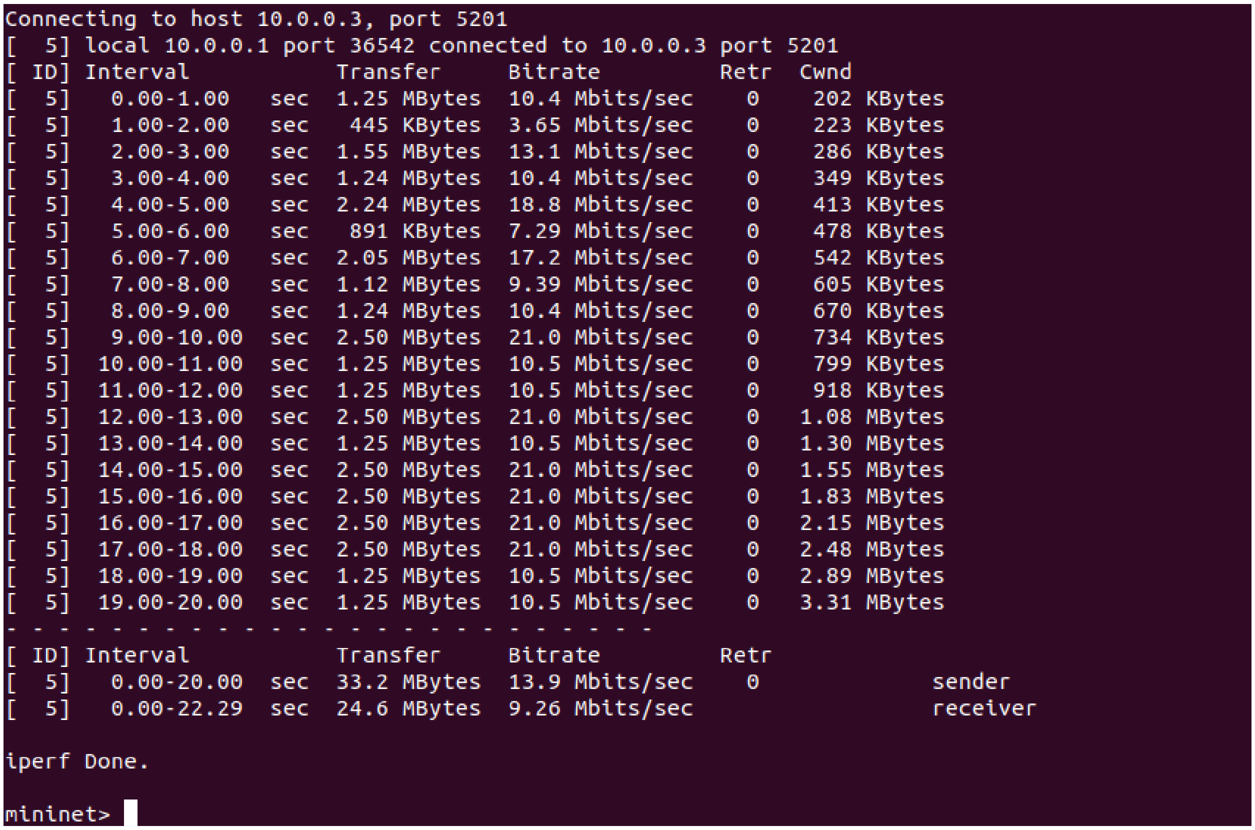


```
Connecting to host 10.0.0.3, port 5201
[  5] local 10.0.0.1 port 52885 connected to 10.0.0.3 port 5201
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[  5]   0.00-0.00   sec  1.41 KBytes  1.05 Gbits/sec  1
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  5]   0.00-0.00   sec  1.41 KBytes  1.05 Gbits/sec  0.000 ms  0/1 (0%)  sender
[  5]   0.00-0.17   sec  0.00 Bytes  0.00 bits/sec  0.000 ms  0/0 (0%)  receiver

iperf Done.
```

7. Run TCP traffic for 20 seconds between H1 (client) and H3(server), monitor the result on the server each 1 second. Provide the iperf commands and the result (take screenshot).

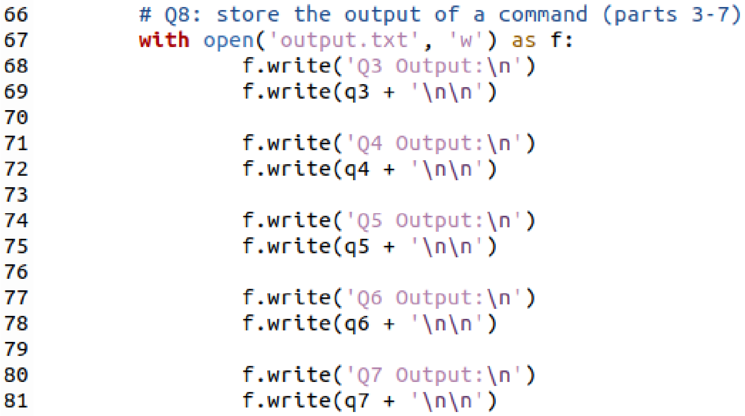


```
        # Q7: run TCP traffic between H1 and H3 for 20 seconds
        h3.cmd('iperf3 -s &')
        q7 = (h1.cmd('iperf3 -c {} -t 20 -i 1'.format(h3.IP())))
        print(q7)
```



```
Connecting to host 10.0.0.3, port 5201
[  5] local 10.0.0.1 port 36542 connected to 10.0.0.3 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  1.25 MBytes  10.4 Mbits/sec    0    202 KBytes
[  5]   1.00-2.00   sec   445 KBytes  3.65 Mbits/sec    0    223 KBytes
[  5]   2.00-3.00   sec  1.55 MBytes  13.1 Mbits/sec    0    286 KBytes
[  5]   3.00-4.00   sec  1.24 MBytes  10.4 Mbits/sec    0    349 KBytes
[  5]   4.00-5.00   sec  2.24 MBytes  18.8 Mbits/sec    0    413 KBytes
[  5]   5.00-6.00   sec   891 KBytes  7.29 Mbits/sec    0    478 KBytes
[  5]   6.00-7.00   sec  2.05 MBytes  17.2 Mbits/sec    0    542 KBytes
[  5]   7.00-8.00   sec  1.12 MBytes  9.39 Mbits/sec    0    605 KBytes
[  5]   8.00-9.00   sec  1.24 MBytes  10.4 Mbits/sec    0    670 KBytes
[  5]   9.00-10.00  sec  2.50 MBytes  21.0 Mbits/sec    0    734 KBytes
[  5]  10.00-11.00  sec  1.25 MBytes  10.5 Mbits/sec    0    799 KBytes
[  5]  11.00-12.00  sec  1.25 MBytes  10.5 Mbits/sec    0    918 KBytes
[  5]  12.00-13.00  sec  2.50 MBytes  21.0 Mbits/sec    0   1.08 MBytes
[  5]  13.00-14.00  sec  1.25 MBytes  10.5 Mbits/sec    0   1.30 MBytes
[  5]  14.00-15.00  sec  2.50 MBytes  21.0 Mbits/sec    0   1.55 MBytes
[  5]  15.00-16.00  sec  2.50 MBytes  21.0 Mbits/sec    0   1.83 MBytes
[  5]  16.00-17.00  sec  2.50 MBytes  21.0 Mbits/sec    0   2.15 MBytes
[  5]  17.00-18.00  sec  2.50 MBytes  21.0 Mbits/sec    0   2.48 MBytes
[  5]  18.00-19.00  sec  1.25 MBytes  10.5 Mbits/sec    0   2.89 MBytes
[  5]  19.00-20.00  sec  1.25 MBytes  10.5 Mbits/sec    0   3.31 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-20.00  sec  33.2 MBytes  13.9 Mbits/sec    0             sender
[  5]   0.00-22.29  sec  24.6 MBytes  9.26 Mbits/sec                  receiver

iperf Done.

mininet>
```

8. You need to store the output of a command (parts 3- 7) in a file (Just try it on one command) (take screenshot).



```
    # Q8: store the output of a command (parts 3-7)
    with open('output.txt', 'w') as f:
        f.write('Q3 Output:\n')
        f.write(q3 + '\n\n')

        f.write('Q4 Output:\n')
        f.write(q4 + '\n\n')

        f.write('Q5 Output:\n')
        f.write(q5 + '\n\n')

        f.write('Q6 Output:\n')
        f.write(q6 + '\n\n')

        f.write('Q7 Output:\n')
        f.write(q7 + '\n\n')
```

“mininet_a2.py” code

```
# Andre Hei Wang Law
# 4017 5600
# coen366 FL-X
# mininet a2

from mininet.net import Mininet
from mininet.node import Controller
from mininet.cli import CLI
from mininet.link import TCLink

def create_topology():
    net = Mininet(controller=Controller, link=TCLink)

    # Q1: add a controller
    c0 = net.addController('c0')

    # add hosts
    h1 = net.addHost('h1')
    h2 = net.addHost('h2')
    h3 = net.addHost('h3')

    # add switches
    s1 = net.addSwitch('s1')
    s2 = net.addSwitch('s2')

    # create links
    net.addLink(h1, s1, bw=20, delay='10ms')
    net.addLink(h2, s1, bw=25, delay='10ms')
    net.addLink(s1, s2, bw=11, delay='40ms')
    net.addLink(s2, h3, bw=15, delay='7ms')

    # build
    net.build()
    c0.start()
    s1.start([c0])
    s2.start([c0])

   # Q2: test the reachability between every host
    net.pingAll()

    # Q3: run UDP traffic between H1 and H2 for 10 seconds
    h2.cmd('iperf3 -s &')
    q3 = (h1.cmd('iperf3 -c {} -u -t 10'.format(h2.IP())))
    print(q3)
    # Q4: run UDP traffic between H1 and H3 for 20 seconds
    h3.cmd('iperf3 -s &')
    q4=  (h1.cmd('iperf3 -c {} -u -t 20'.format(h3.IP())))
    print(q4)

    # Q5: run UDP traffic sending 1 Gbytes between H1 and H2
    h2.cmd('iperf3 -s &')
    q5 = (h1.cmd('iperf3 -c {} -u -n 1'.format(h2.IP())))
    print(q5)

    # Q6: run UDP traffic sending 2 Gbytes between H1 and H3
    h3.cmd('iperf3 -s &')
    q6 = (h1.cmd('iperf3 -c {} -u -n 2'.format(h3.IP())))
    print(q6)

    # Q7: run TCP traffic between H1 and H3 for 20 seconds
    h3.cmd('iperf3 -s &')
    q7 = (h1.cmd('iperf3 -c {} -t 20 -i 1'.format(h3.IP())))
    print(q7)

    # Q8: store the output of a command (parts 3-7)
    with open('output.txt', 'w') as f:
        f.write('Q3 Output:\n')
        f.write(q3 + '\n\n')

        f.write('Q4 Output:\n')
        f.write(q4 + '\n\n')

        f.write('Q5 Output:\n')
        f.write(q5 + '\n\n')

      f.write('Q6 Output:\n')
        f.write(q6 + '\n\n')

        f.write('Q7 Output:\n')
        f.write(q7 + '\n\n')

    CLI(net)
    net.stop()

if __name__ == '__main__':
    create_topology()
```

“output.txt” file

```
Q3 Output:

Connecting to host 10.0.0.2, port 5201

[  5] local 10.0.0.1 port 34000 connected to 10.0.0.2 port 5201

[ ID] Interval           Transfer     Bitrate         Total Datagrams

[  5]   0.00-1.00   sec   129 KBytes  1.05 Mbits/sec  91

[  5]   1.00-2.00   sec   127 KBytes  1.05 Mbits/sec  90

[  5]   2.00-3.00   sec   129 KBytes  1.05 Mbits/sec  91

[  5]   3.00-4.00   sec   127 KBytes  1.04 Mbits/sec  90

[  5]   4.00-5.00   sec   129 KBytes  1.05 Mbits/sec  91

[  5]   5.00-6.00   sec   127 KBytes  1.04 Mbits/sec  90

[  5]   6.00-7.00   sec   129 KBytes  1.05 Mbits/sec  91

[  5]   7.00-8.00   sec   129 KBytes  1.05 Mbits/sec  91

[  5]   8.00-9.01   sec   122 KBytes   987 Kbits/sec  86

[  5]   9.01-10.00  sec   134 KBytes  1.11 Mbits/sec  95

- - - - - - - - - - - - - - - - - - - - - - - - -

[ ID] Interval           Transfer     Bitrate         Jitter
Lost/Total Datagrams

[  5]   0.00-10.00  sec  1.25 MBytes  1.05 Mbits/sec  0.000 ms  0/906
(0%)  sender

[  5]   0.00-10.08  sec  1.25 MBytes  1.04 Mbits/sec  0.761 ms  0/906
(0%)  receiver



iperf Done.
```

```
Q4 Output:

Connecting to host 10.0.0.3, port 5201

[  5] local 10.0.0.1 port 45960 connected to 10.0.0.3 port 5201

[ ID] Interval           Transfer     Bitrate         Total Datagrams

[  5]   0.00-1.00   sec   129 KBytes  1.05 Mbits/sec  91

[  5]   1.00-2.00   sec   127 KBytes  1.04 Mbits/sec  90

[  5]   2.00-3.00   sec   129 KBytes  1.05 Mbits/sec  91

[  5]   3.00-4.00   sec   129 KBytes  1.05 Mbits/sec  91

[  5]   4.00-5.00   sec   124 KBytes  1.02 Mbits/sec  88

[  5]   5.00-6.01   sec   130 KBytes  1.06 Mbits/sec  92

[  5]   6.01-7.02   sec   127 KBytes  1.02 Mbits/sec  90

[  5]   7.02-8.00   sec   130 KBytes  1.09 Mbits/sec  92

[  5]   8.00-9.00   sec   127 KBytes  1.04 Mbits/sec  90

[  5]   9.00-10.00  sec   129 KBytes  1.06 Mbits/sec  91

[  5]  10.00-11.11  sec   112 KBytes   822 Kbits/sec  79

[  5]  11.11-12.03  sec   123 KBytes  1.09 Mbits/sec  87

[  5]  12.03-13.01  sec   148 KBytes  1.25 Mbits/sec  105

[  5]  13.01-14.00  sec   126 KBytes  1.04 Mbits/sec  89

[  5]  14.00-15.05  sec   127 KBytes   998 Kbits/sec  90

[  5]  15.05-16.00  sec   130 KBytes  1.12 Mbits/sec  92

[  5]  16.00-17.00  sec   129 KBytes  1.06 Mbits/sec  91

[  5]  17.00-18.00  sec   129 KBytes  1.05 Mbits/sec  91

[  5]  18.00-19.01  sec   126 KBytes  1.03 Mbits/sec  89
[  5]  19.01-20.02  sec   129 KBytes  1.04 Mbits/sec  91

- - - - - - - - - - - - - - - - - - - - - - - - -

[ ID] Interval           Transfer     Bitrate         Jitter
Lost/Total Datagrams

[  5]   0.00-20.02  sec  2.50 MBytes  1.05 Mbits/sec  0.000 ms  0/1810
(0%)  sender

[  5]   0.00-20.18  sec  2.50 MBytes  1.04 Mbits/sec  8.731 ms  0/1810
(0%)  receiver



iperf Done.




Q5 Output:

Connecting to host 10.0.0.2, port 5201

[  5] local 10.0.0.1 port 46557 connected to 10.0.0.2 port 5201

[ ID] Interval           Transfer     Bitrate         Total Datagrams

[  5]   0.00-0.00   sec  1.41 KBytes   827 Mbits/sec  1

- - - - - - - - - - - - - - - - - - - - - - - - -

[ ID] Interval           Transfer     Bitrate         Jitter
Lost/Total Datagrams

[  5]   0.00-0.00   sec  1.41 KBytes   827 Mbits/sec  0.000 ms  0/1
(0%)  sender

[  5]   0.00-0.09   sec  0.00 Bytes  0.00 bits/sec  0.000 ms  0/0 (0%)
receiver



iperf Done.
```

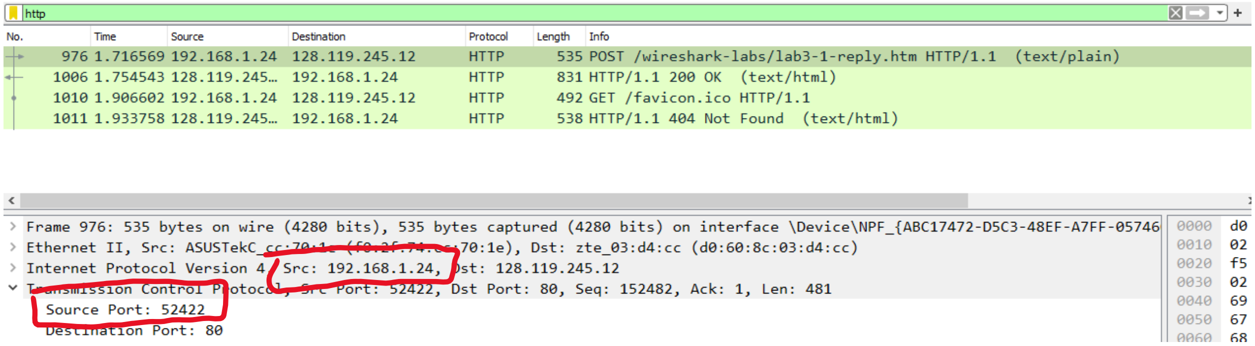
```
Q6 Output:

Connecting to host 10.0.0.3, port 5201

[  5] local 10.0.0.1 port 52885 connected to 10.0.0.3 port 5201

[ ID] Interval           Transfer     Bitrate         Total Datagrams

[  5]   0.00-0.00   sec  1.41 KBytes  1.05 Gbits/sec  1

- - - - - - - - - - - - - - - - - - - - - - - - -

[ ID] Interval           Transfer     Bitrate         Jitter
Lost/Total Datagrams

[  5]   0.00-0.00   sec  1.41 KBytes  1.05 Gbits/sec  0.000 ms  0/1
(0%)  sender

[  5]   0.00-0.17   sec  0.00 Bytes  0.00 bits/sec  0.000 ms  0/0 (0%)
receiver



iperf Done.




Q7 Output:

Connecting to host 10.0.0.3, port 5201

[  5] local 10.0.0.1 port 36542 connected to 10.0.0.3 port 5201

[ ID] Interval           Transfer     Bitrate         Retr  Cwnd

[  5]   0.00-1.00   sec  1.25 MBytes  10.4 Mbits/sec    0    202
KBytes

[  5]   1.00-2.00   sec   445 KBytes  3.65 Mbits/sec    0    223
KBytes

[  5]   2.00-3.00   sec  1.55 MBytes  13.1 Mbits/sec    0    286
KBytes
```

```
[  5]   3.00-4.00   sec  1.24 MBytes  10.4 Mbits/sec    0    349
KBytes

[  5]   4.00-5.00   sec  2.24 MBytes  18.8 Mbits/sec    0    413
KBytes

[  5]   5.00-6.00   sec   891 KBytes  7.29 Mbits/sec    0    478
KBytes

[  5]   6.00-7.00   sec  2.05 MBytes  17.2 Mbits/sec    0    542
KBytes

[  5]   7.00-8.00   sec  1.12 MBytes  9.39 Mbits/sec    0    605
KBytes

[  5]   8.00-9.00   sec  1.24 MBytes  10.4 Mbits/sec    0    670
KBytes

[  5]   9.00-10.00  sec  2.50 MBytes  21.0 Mbits/sec    0    734
KBytes

[  5]  10.00-11.00  sec  1.25 MBytes  10.5 Mbits/sec    0    799
KBytes

[  5]  11.00-12.00  sec  1.25 MBytes  10.5 Mbits/sec    0    918
KBytes

[  5]  12.00-13.00  sec  2.50 MBytes  21.0 Mbits/sec    0   1.08
MBytes

[  5]  13.00-14.00  sec  1.25 MBytes  10.5 Mbits/sec    0   1.30
MBytes

[  5]  14.00-15.00  sec  2.50 MBytes  21.0 Mbits/sec    0   1.55
MBytes

[  5]  15.00-16.00  sec  2.50 MBytes  21.0 Mbits/sec    0   1.83
MBytes

[  5]  16.00-17.00  sec  2.50 MBytes  21.0 Mbits/sec    0   2.15
MBytes

[  5]  17.00-18.00  sec  2.50 MBytes  21.0 Mbits/sec    0   2.48
MBytes

[  5]  18.00-19.00  sec  1.25 MBytes  10.5 Mbits/sec    0   2.89
MBytes
```

```
[  5]  19.00-20.00  sec  1.25 MBytes  10.5 Mbits/sec    0   3.31
MBytes

- - - - - - - - - - - - - - - - - - - - - - - - -

[ ID] Interval           Transfer     Bitrate         Retr

[  5]   0.00-20.00  sec  33.2 MBytes  13.9 Mbits/sec    0
sender

[  5]   0.00-22.29  sec  24.6 MBytes  9.26 Mbits/sec
receiver



iperf Done.
```

2. Wireshark Assignment 3

1. What is the IP address and TCP port number used by the client computer (source) that is transferring the file to gaia.cs.umass.edu? To answer this question, it's probably easiest to select an HTTP message and explore the details of the TCP packet used to carry this HTTP message, using the "details of the selected packet header window" (refer to Figure 2 in the "Getting Started with Wireshark" Lab if you're uncertain about the Wireshark windows.



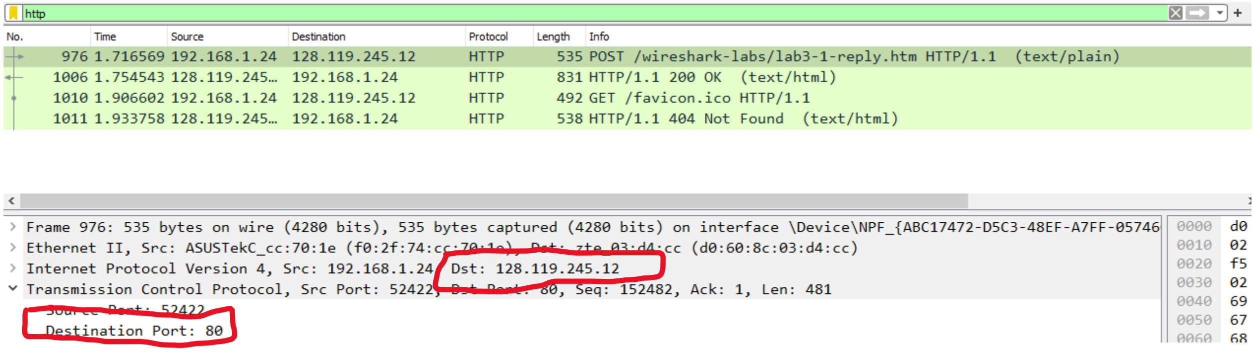
```
http
No.    Time     Source         Destination     Protocol  Length  Info
  976 1.716569 192.168.1.24   128.119.245.12  HTTP      535 POST /wireshark-labs/lab3-1-reply.htm HTTP/1.1  (text/plain)
 1006 1.754543 128.119.245…   192.168.1.24    HTTP      831 HTTP/1.1 200 OK  (text/html)
 1010 1.906602 192.168.1.24   128.119.245.12  HTTP      492 GET /favicon.ico HTTP/1.1
 1011 1.933758 128.119.245…   192.168.1.24    HTTP      538 HTTP/1.1 404 Not Found  (text/html)
```

```
> Frame 976: 535 bytes on wire (4280 bits), 535 bytes captured (4280 bits) on interface \Device\NPF_{ABC17472-D5C3-48EF-A7FF-05746
> Ethernet II, Src: ASUSTekC_cc:70:1e (f0:2f:74:cc:70:1e), Dst: zte_03:d4:cc (d0:60:8c:03:d4:cc)
> Internet Protocol Version 4, Src: 192.168.1.24, Dst: 128.119.245.12
v Transmission Control Protocol, Src Port: 52422, Dst Port: 80, Seq: 152482, Ack: 1, Len: 481
    Source Port: 52422
    Destination Port: 80
```

```
0000  d0
0010  02
0020  f5
0030  02
0040  69
0050  67
0060  68
```

Source IP Address: 192.168.1.24, Source Port: 52422

2. What is the IP address of gaia.cs.umass.edu? On what port number is it sending and receiving TCP segments for this connection?



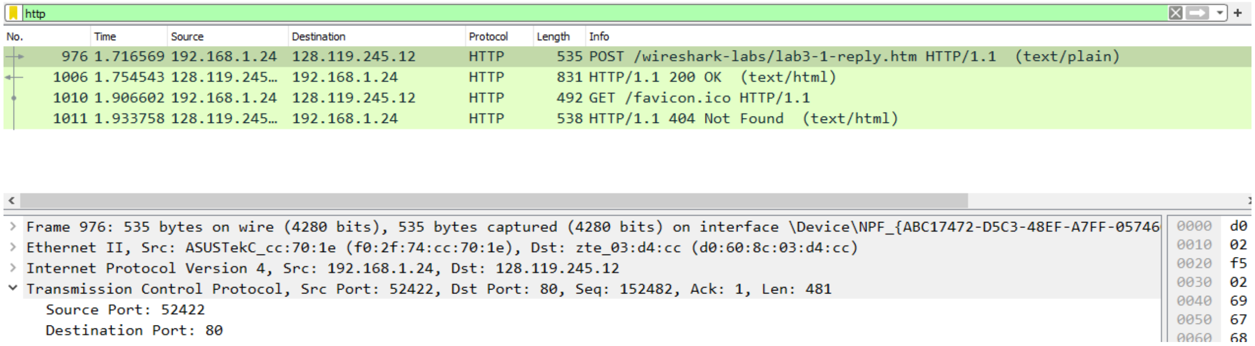
```
http
No.    Time     Source         Destination     Protocol  Length  Info
  976 1.716569 192.168.1.24   128.119.245.12  HTTP      535 POST /wireshark-labs/lab3-1-reply.htm HTTP/1.1  (text/plain)
 1006 1.754543 128.119.245…   192.168.1.24    HTTP      831 HTTP/1.1 200 OK  (text/html)
 1010 1.906602 192.168.1.24   128.119.245.12  HTTP      492 GET /favicon.ico HTTP/1.1
 1011 1.933758 128.119.245…   192.168.1.24    HTTP      538 HTTP/1.1 404 Not Found  (text/html)
```

```
> Frame 976: 535 bytes on wire (4280 bits), 535 bytes captured (4280 bits) on interface \Device\NPF_{ABC17472-D5C3-48EF-A7FF-05746
> Ethernet II, Src: ASUSTekC_cc:70:1e (f0:2f:74:cc:70:1e), Dst: zte_03:d4:cc (d0:60:8c:03:d4:cc)
> Internet Protocol Version 4, Src: 192.168.1.24, Dst: 128.119.245.12
v Transmission Control Protocol, Src Port: 52422, Dst Port: 80, Seq: 152482, Ack: 1, Len: 481
    Source Port: 52422
    Destination Port: 80
```

```
0000  d0
0010  02
0020  f5
0030  02
0040  69
0050  67
0060  68
```

Destination IP Address: 128.119.245.12, Destination Port: 80

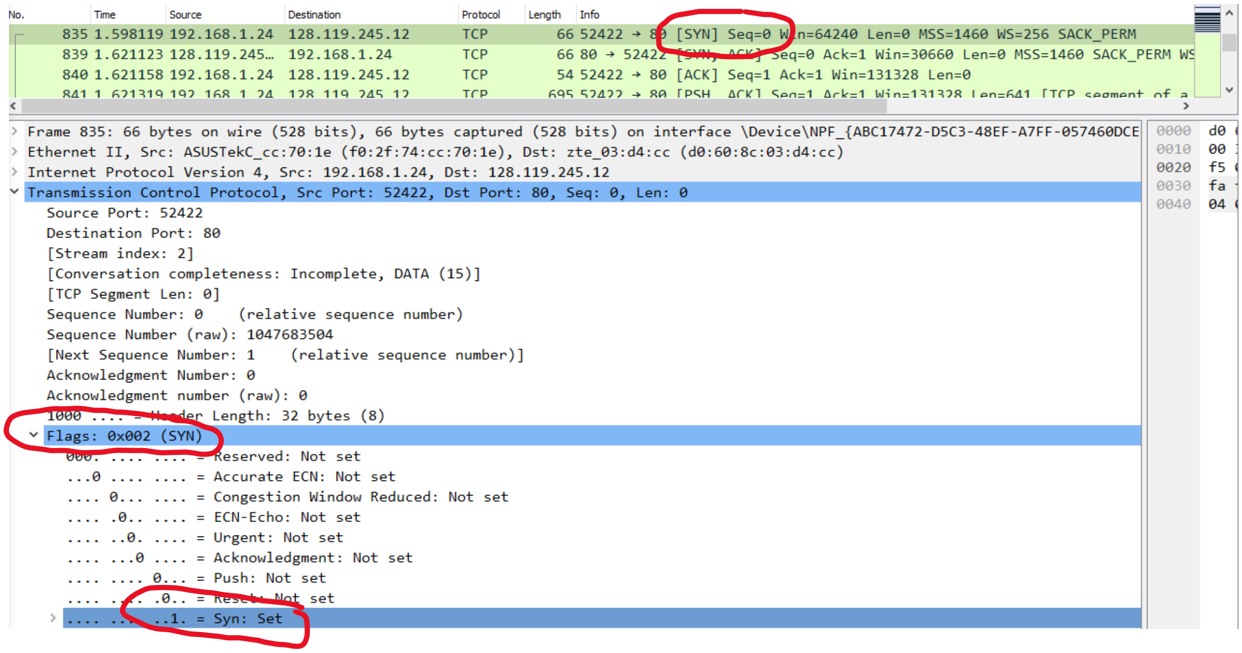
3. What is the IP address and TCP port number used by your client computer (source) to transfer the file to gaia.cs.umass.edu?



```
http

No.     Time     Source          Destination      Protocol  Length Info
  976 1.716569 192.168.1.24    128.119.245.12   HTTP       535 POST /wireshark-labs/lab3-1-reply.htm HTTP/1.1  (text/plain)
 1006 1.754543 128.119.245…    192.168.1.24     HTTP       831 HTTP/1.1 200 OK  (text/html)
 1010 1.906602 192.168.1.24    128.119.245.12   HTTP       492 GET /favicon.ico HTTP/1.1
 1011 1.933758 128.119.245…    192.168.1.24     HTTP       538 HTTP/1.1 404 Not Found  (text/html)

> Frame 976: 535 bytes on wire (4280 bits), 535 bytes captured (4280 bits) on interface \Device\NPF_{ABC17472-D5C3-48EF-A7FF-05746   0000  d0
> Ethernet II, Src: ASUSTekC_cc:70:1e (f0:2f:74:cc:70:1e), Dst: zte_03:d4:cc (d0:60:8c:03:d4:cc)                                0010  02
> Internet Protocol Version 4, Src: 192.168.1.24, Dst: 128.119.245.12                                                            0020  f5
v Transmission Control Protocol, Src Port: 52422, Dst Port: 80, Seq: 152482, Ack: 1, Len: 481                                   0030  02
    Source Port: 52422                                                                                                          0040  69
    Destination Port: 80                                                                                                        0050  67
                                                                                                                                0060  68
```

Source IP Address: 192.168.1.24, Source Port: 52422

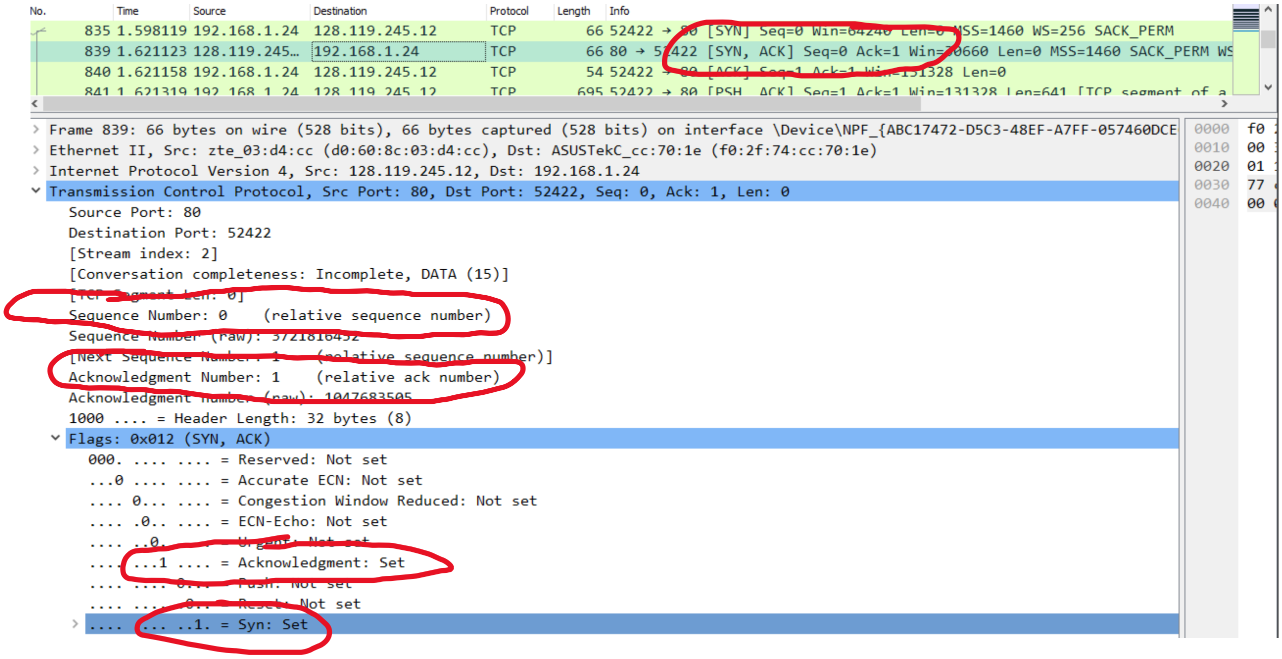
4. What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and gaia.cs.umass.edu? What is it in the segment that identifies the segment as a SYN segment?



```
No.     Time     Source          Destination      Protocol  Length Info
  835 1.598119 192.168.1.24    128.119.245.12   TCP         66 52422 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
  839 1.621123 128.119.245…    192.168.1.24     TCP         66 80 → 52422 [SYN, ACK] Seq=0 Ack=1 Win=30660 Len=0 MSS=1460 SACK_PERM WS
  840 1.621158 192.168.1.24    128.119.245.12   TCP         54 52422 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
  841 1.621319 192.168.1.24    128.119.245.12   TCP        695 52422 → 80 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=641 [TCP segment of a

> Frame 835: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{ABC17472-D5C3-48EF-A7FF-057460DCE   0000  d0
> Ethernet II, Src: ASUSTekC_cc:70:1e (f0:2f:74:cc:70:1e), Dst: zte_03:d4:cc (d0:60:8c:03:d4:cc)                                0010  00
> Internet Protocol Version 4, Src: 192.168.1.24, Dst: 128.119.245.12                                                            0020  f5
v Transmission Control Protocol, Src Port: 52422, Dst Port: 80, Seq: 0, Len: 0                                                  0030  fa
    Source Port: 52422                                                                                                          0040  04
    Destination Port: 80
    [Stream index: 2]
    [Conversation completeness: Incomplete, DATA (15)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 1047683504
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    1000 .... = Header Length: 32 bytes (8)
  v Flags: 0x002 (SYN)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Accurate ECN: Not set
      .... 0... .... = Congestion Window Reduced: Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...0 .... = Acknowledgment: Not set
      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
    > .... .... ..1. = Syn: Set
```

Sequence number of SYN is 0. “Syn” flag is “Set” means that it is a SYN segment.

5. What is the sequence number of the SYNACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN? What is the value of the Acknowledgement field in the SYNACK segment? How did gaia.cs.umass.edu determine that value? What is it in the segment that identifies the segment as a SYNACK segment?

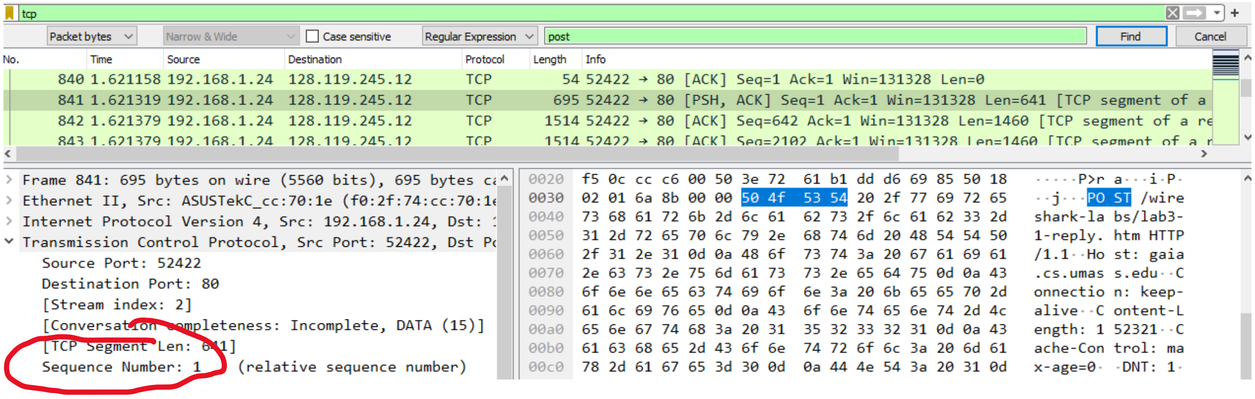


```
No.     Time     Source          Destination     Protocol  Length  Info
    835 1.598119 192.168.1.24    128.119.245.12  TCP          66 52422 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
    839 1.621123 128.119.245…    192.168.1.24    TCP          66 80 → 52422 [SYN, ACK] Seq=0 Ack=1 Win=[illegible]0660 Len=0 MSS=1460 SACK_PERM WS
    840 1.621158 192.168.1.24    128.119.245.12  TCP          54 52422 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
    841 1.621319 192.168.1.24    128.119.245.12  TCP         695 52422 → 80 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=641 [TCP segment of a
```

```
> Frame 839: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{ABC17472-D5C3-48EF-A7FF-057460DCE   0000  f0
> Ethernet II, Src: zte_03:d4:cc (d0:60:8c:03:d4:cc), Dst: ASUSTekC_cc:70:1e (f0:2f:74:cc:70:1e)                                   0010  00
> Internet Protocol Version 4, Src: 128.119.245.12, Dst: 192.168.1.24                                                                0020  01
v Transmission Control Protocol, Src Port: 80, Dst Port: 52422, Seq: 0, Ack: 1, Len: 0                                               0030  77
    Source Port: 80                                                                                                                  0040  00
    Destination Port: 52422
    [Stream index: 2]
    [Conversation completeness: Incomplete, DATA (15)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 3721816432
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 1047683505
    1000 .... = Header Length: 32 bytes (8)
  v Flags: 0x012 (SYN, ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Accurate ECN: Not set
      .... 0... .... = Congestion Window Reduced: Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
    > .... .... ..1. = Syn: Set
```

Sequence number of SYNACK segment is 0. The Acknowledgement field is 1. It is determined by incrementing the initial sequence by 1. The segment that identifies the segment as a SYNACK segment is the Acknowledgment "Set" and Syn "Set" flags.

6. What is the sequence number of the TCP segment containing the HTTP POST command? Note that in order to find the POST command, you'll need to dig into the packet content field at the bottom of the Wireshark window, looking for a segment with a "POST" within its DATA field.

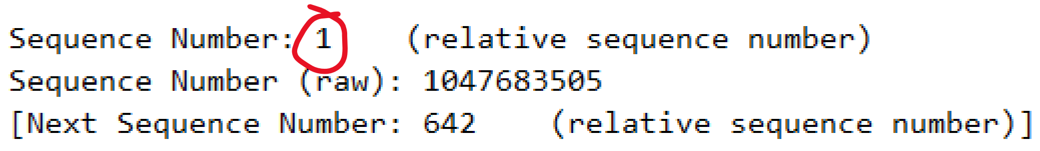


```
tcp
Packet bytes    Narrow & Wide    ☐ Case sensitive    Regular Expression    post    Find    Cancel

No.   Time     Source        Destination    Protocol Length Info
  840 1.621158 192.168.1.24  128.119.245.12 TCP        54 52422 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
  841 1.621319 192.168.1.24  128.119.245.12 TCP       695 52422 → 80 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=641 [TCP segment of a
  842 1.621379 192.168.1.24  128.119.245.12 TCP      1514 52422 → 80 [ACK] Seq=642 Ack=1 Win=131328 Len=1460 [TCP segment of a re
  843 1.621379 192.168.1.24  128.119.245.12 TCP      1514 52422 → 80 [ACK] Seq=2102 Ack=1 Win=131328 Len=1460 [TCP segment of a r

> Frame 841: 695 bytes on wire (5560 bits), 695 bytes ca
> Ethernet II, Src: ASUSTekC_cc:70:1e (f0:2f:74:cc:70:1e
> Internet Protocol Version 4, Src: 192.168.1.24, Dst: 1
v Transmission Control Protocol, Src Port: 52422, Dst Po
    Source Port: 52422
    Destination Port: 80
    [Stream index: 2]
    [Conversation completeness: Incomplete, DATA (15)]
    [TCP Segment Len: 641]
    Sequence Number: 1    (relative sequence number)

0020  f5 0c cc c6 00 50 3e 72  61 b1 dd d6 69 85 50 18   ·····P>r a···i·P·
0030  02 01 6a 8b 00 00 50 4f  53 54 20 2f 77 69 72 65   ··j···PO ST /wire
0040  73 68 61 72 6b 2d 6c 61  62 73 2f 6c 61 62 33 2d   shark-la bs/lab3-
0050  31 2d 72 65 70 6c 79 2e  68 74 6d 20 48 54 54 50   1-reply. htm HTTP
0060  2f 31 2e 31 0d 0a 48 6f  73 74 3a 20 67 61 69 61   /1.1··Ho st: gaia
0070  2e 63 73 2e 75 6d 61 73  73 2e 65 64 75 0d 0a 43   .cs.umas s.edu··C
0080  6f 6e 6e 65 63 74 69 6f  6e 3a 20 6b 65 65 70 2d   onnectio n: keep-
0090  61 6c 69 76 65 0d 0a 43  6f 6e 74 65 6e 74 2d 4c   alive··C ontent-L
00a0  65 6e 67 74 68 3a 20 31  35 32 33 32 31 0d 0a 43   ength: 1 52321··C
00b0  61 63 68 65 2d 43 6f 6e  74 72 6f 6c 3a 20 6d 61   ache-Con trol: ma
00c0  78 2d 61 67 65 3d 30 0d  0a 44 4e 54 3a 20 31 0d   x-age=0· ·DNT: 1·
```

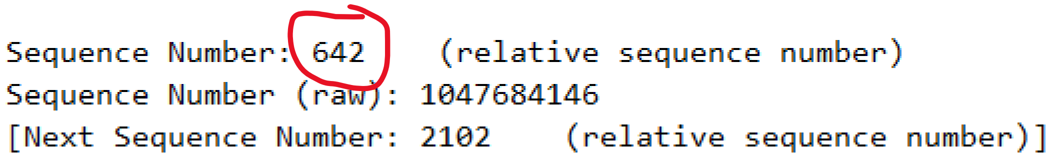
Sequence number of the TCP segment containing the HTTP POST is 1.

<u>7. Consider the TCP segment containing the HTTP POST as the first segment in the TCP connection. What are the sequence numbers of the first six segments in the TCP connection (including the segment containing the HTTP POST)? At what time was each segment sent? When was the ACK for each segment received? Given the difference between when each TCP segment was sent, and when its acknowledgement was received, what is the RTT value for each of the six segments? What is the EstimatedRTT value (see Section 3.5.3, page 242 in text) after the receipt of each ACK? Assume that the value of the EstimatedRTT is equal to the measured RTT for the first segment, and then is computed using the EstimatedRTT equation on page 242 for all subsequent segments.</u>

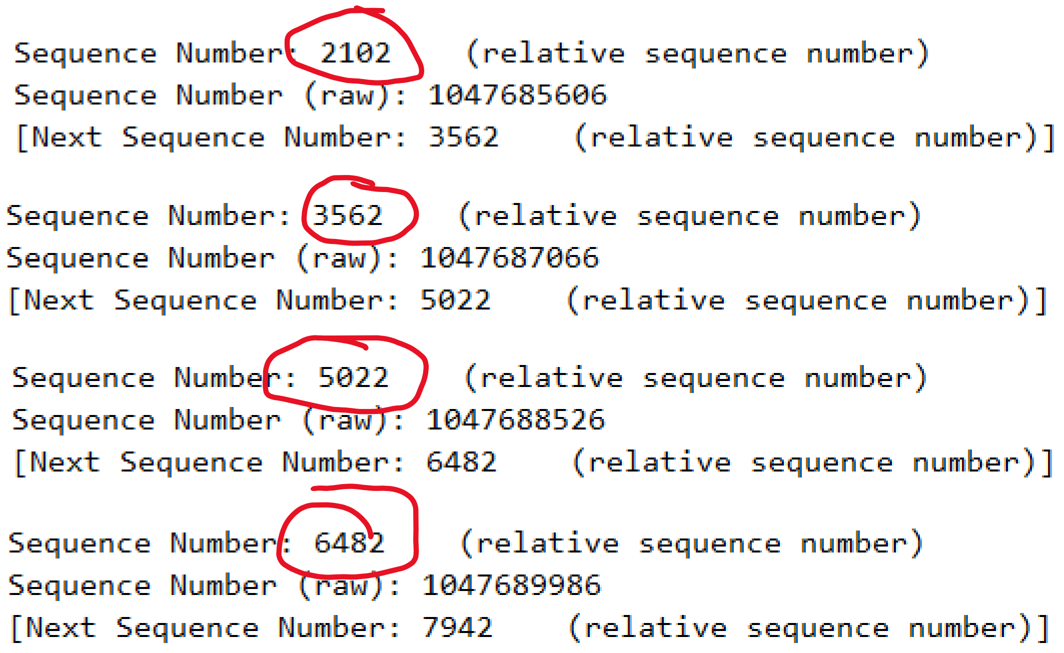
-Sequence numbers of the first six segments:



```
Sequence Number: 1    (relative sequence number)
Sequence Number (raw): 1047683505
[Next Sequence Number: 642    (relative sequence number)]
```

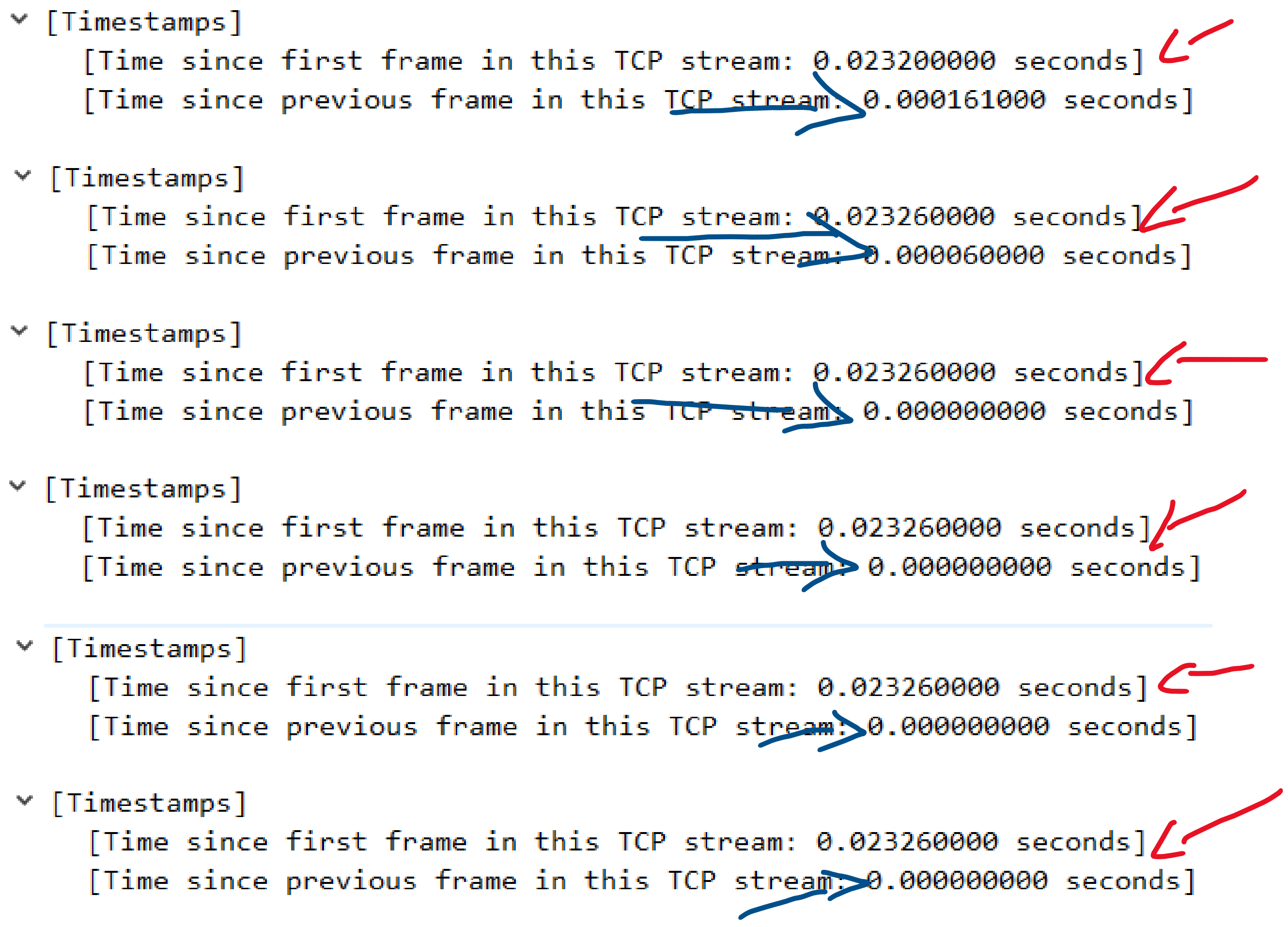


```
Sequence Number: 642    (relative sequence number)
Sequence Number (raw): 1047684146
[Next Sequence Number: 2102    (relative sequence number)]
```



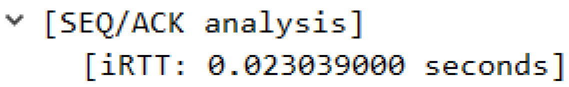
```
Sequence Number: 2102    (relative sequence number)
Sequence Number (raw): 1047685606
[Next Sequence Number: 3562    (relative sequence number)]

Sequence Number: 3562    (relative sequence number)
Sequence Number (raw): 1047687066
[Next Sequence Number: 5022    (relative sequence number)]

Sequence Number: 5022    (relative sequence number)
Sequence Number (raw): 1047688526
[Next Sequence Number: 6482    (relative sequence number)]

Sequence Number: 6482    (relative sequence number)
Sequence Number (raw): 1047689986
[Next Sequence Number: 7942    (relative sequence number)]
```

-Time of each segment SENT (red) and RECEIVED (blue):

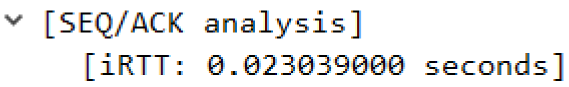


```
⌄ [Timestamps]
    [Time since first frame in this TCP stream: 0.023200000 seconds]
    [Time since previous frame in this TCP stream: 0.000161000 seconds]

⌄ [Timestamps]
    [Time since first frame in this TCP stream: 0.023260000 seconds]
    [Time since previous frame in this TCP stream: 0.000060000 seconds]

⌄ [Timestamps]
    [Time since first frame in this TCP stream: 0.023260000 seconds]
    [Time since previous frame in this TCP stream: 0.000000000 seconds]

⌄ [Timestamps]
    [Time since first frame in this TCP stream: 0.023260000 seconds]
    [Time since previous frame in this TCP stream: 0.000000000 seconds]

⌄ [Timestamps]
    [Time since first frame in this TCP stream: 0.023260000 seconds]
    [Time since previous frame in this TCP stream: 0.000000000 seconds]

⌄ [Timestamps]
    [Time since first frame in this TCP stream: 0.023260000 seconds]
    [Time since previous frame in this TCP stream: 0.000000000 seconds]
```

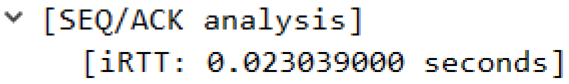
-RTT value for each six segment (subtraction between sent and received):



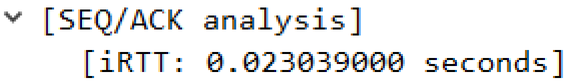
```
⌄ [SEQ/ACK analysis]
    [iRTT: 0.023039000 seconds]
```



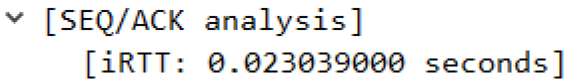
```
⌄ [SEQ/ACK analysis]
    [iRTT: 0.023039000 seconds]
```



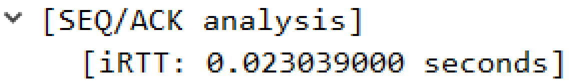
```
⌄ [SEQ/ACK analysis]
    [iRTT: 0.023039000 seconds]
```



```
⌄ [SEQ/ACK analysis]
    [iRTT: 0.023039000 seconds]
```

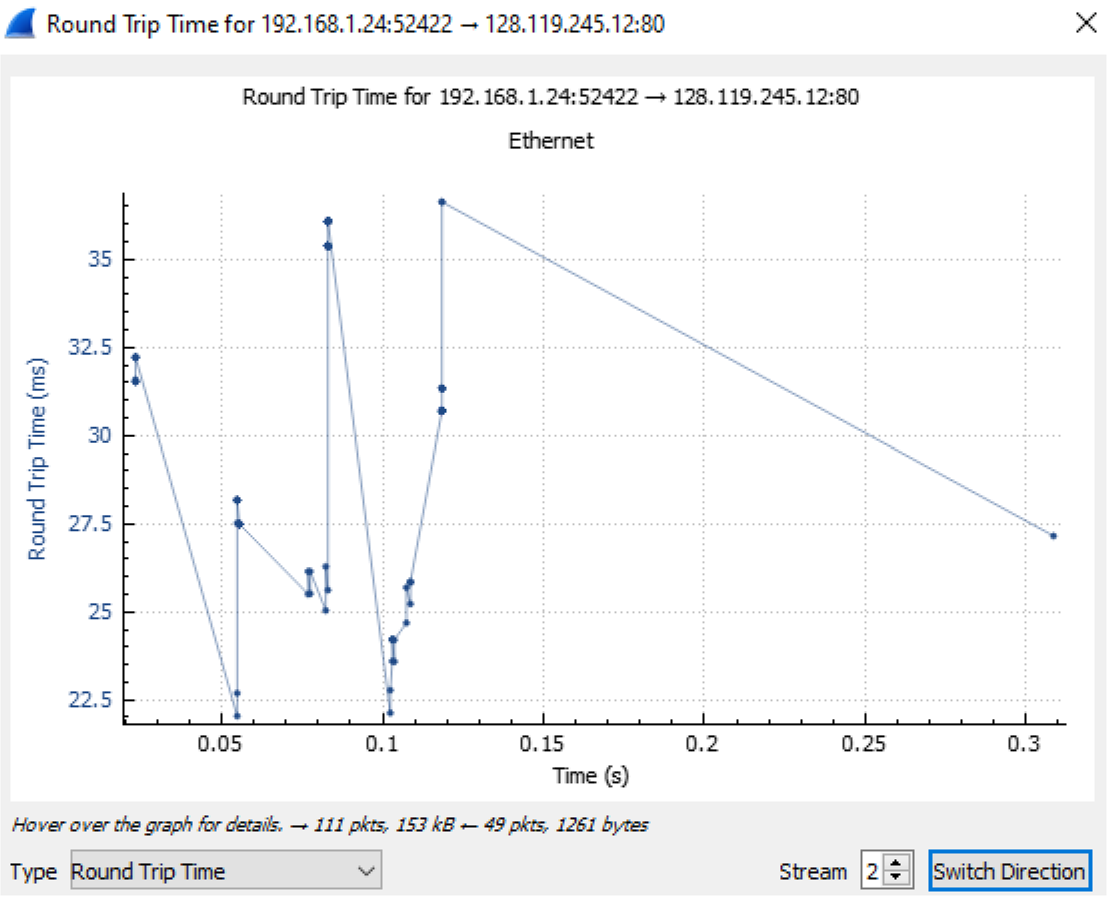


```
⌄ [SEQ/ACK analysis]
    [iRTT: 0.023039000 seconds]
```



```
⌄ [SEQ/ACK analysis]
    [iRTT: 0.023039000 seconds]
```

-Estimate RTT time: The average RTT time is 0.023039 seconds



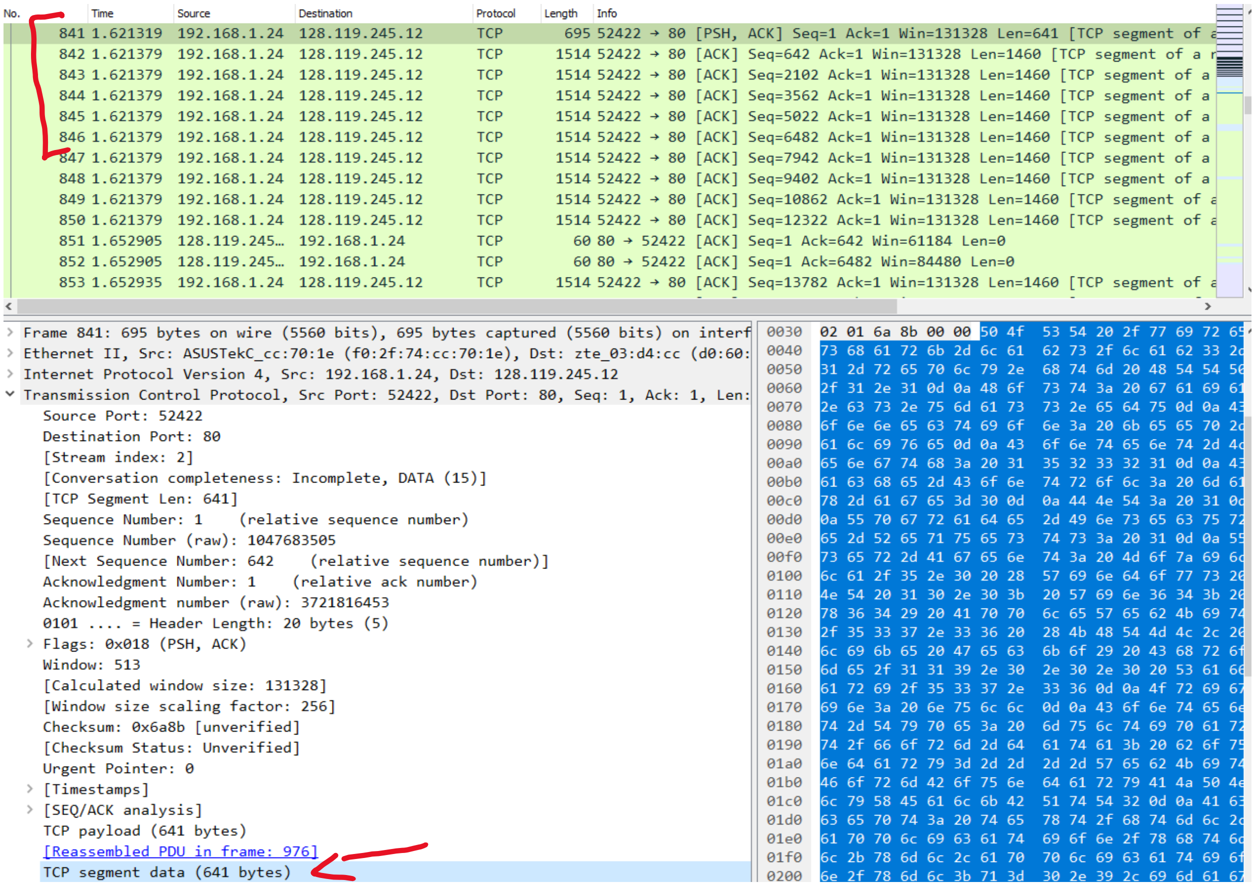
Hover over the graph for details. → 111 pkts, 153 kB ← 49 pkts, 1261 bytes

Type Round Trip Time

Stream 2

Sample image of the round-trip graph of the first HTTP POST segment.

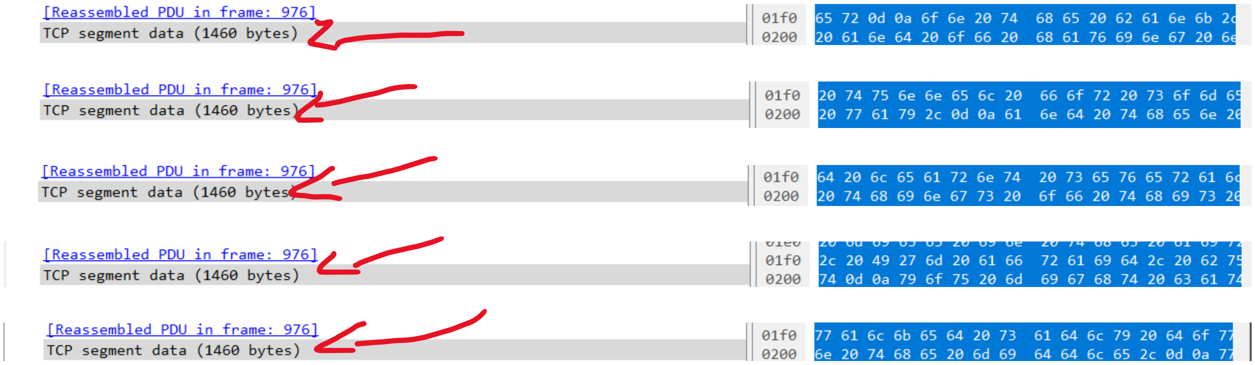
8. What is the length of each of the first six TCP segments?[4]



```
No.  Time     Source         Destination    Protocol Length Info
 841 1.621319 192.168.1.24   128.119.245.12 TCP       695 52422 → 80 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=641 [TCP segment of a
 842 1.621379 192.168.1.24   128.119.245.12 TCP      1514 52422 → 80 [ACK] Seq=642 Ack=1 Win=131328 Len=1460 [TCP segment of a r
 843 1.621379 192.168.1.24   128.119.245.12 TCP      1514 52422 → 80 [ACK] Seq=2102 Ack=1 Win=131328 Len=1460 [TCP segment of a
 844 1.621379 192.168.1.24   128.119.245.12 TCP      1514 52422 → 80 [ACK] Seq=3562 Ack=1 Win=131328 Len=1460 [TCP segment of a
 845 1.621379 192.168.1.24   128.119.245.12 TCP      1514 52422 → 80 [ACK] Seq=5022 Ack=1 Win=131328 Len=1460 [TCP segment of a
 846 1.621379 192.168.1.24   128.119.245.12 TCP      1514 52422 → 80 [ACK] Seq=6482 Ack=1 Win=131328 Len=1460 [TCP segment of a
 847 1.621379 192.168.1.24   128.119.245.12 TCP      1514 52422 → 80 [ACK] Seq=7942 Ack=1 Win=131328 Len=1460 [TCP segment of a
 848 1.621379 192.168.1.24   128.119.245.12 TCP      1514 52422 → 80 [ACK] Seq=9402 Ack=1 Win=131328 Len=1460 [TCP segment of a
 849 1.621379 192.168.1.24   128.119.245.12 TCP      1514 52422 → 80 [ACK] Seq=10862 Ack=1 Win=131328 Len=1460 [TCP segment of a
 850 1.621379 192.168.1.24   128.119.245.12 TCP      1514 52422 → 80 [ACK] Seq=12322 Ack=1 Win=131328 Len=1460 [TCP segment of a
 851 1.652905 128.119.245… 192.168.1.24   TCP        60 80 → 52422 [ACK] Seq=1 Ack=642 Win=61184 Len=0
 852 1.652905 128.119.245… 192.168.1.24   TCP        60 80 → 52422 [ACK] Seq=1 Ack=6482 Win=84480 Len=0
 853 1.652935 192.168.1.24   128.119.245.12 TCP      1514 52422 → 80 [ACK] Seq=13782 Ack=1 Win=131328 Len=1460 [TCP segment of a
```

```
> Frame 841: 695 bytes on wire (5560 bits), 695 bytes captured (5560 bits) on interf
> Ethernet II, Src: ASUSTekC_cc:70:1e (f0:2f:74:cc:70:1e), Dst: zte_03:d4:cc (d0:60:
> Internet Protocol Version 4, Src: 192.168.1.24, Dst: 128.119.245.12
v Transmission Control Protocol, Src Port: 52422, Dst Port: 80, Seq: 1, Ack: 1, Len:
    Source Port: 52422
    Destination Port: 80
    [Stream index: 2]
    [Conversation completeness: Incomplete, DATA (15)]
    [TCP Segment Len: 641]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 1047683505
    [Next Sequence Number: 642    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 3721816453
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window: 513
    [Calculated window size: 131328]
    [Window size scaling factor: 256]
    Checksum: 0x6a8b [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
  > [SEQ/ACK analysis]
    TCP payload (641 bytes)
    [Reassembled PDU in frame: 976]
    TCP segment data (641 bytes)
```

```
0030  02 01 6a 8b 00 00 50 4f  53 54 20 2f 77 69 72 65
0040  73 68 61 72 6b 2d 6c 61  62 73 2f 6c 61 62 33 2d
0050  31 2d 72 65 70 6c 79 2e  68 74 6d 20 48 54 54 50
0060  2f 31 2e 31 0d 0a 48 6f  73 74 3a 20 67 61 69 61
0070  2e 63 73 2e 75 6d 61 73  73 2e 65 64 75 0d 0a 43
0080  6f 6e 6e 65 63 74 69 6f  6e 3a 20 6b 65 65 70 2d
0090  61 6c 69 76 65 0d 0a 43  6f 6e 74 65 6e 74 2d 4c
00a0  65 6e 67 74 68 3a 20 31  35 32 33 32 31 0d 0a 43
00b0  61 63 68 65 2d 43 6f 6e  74 72 6f 6c 3a 20 6d 61
00c0  78 2d 61 67 65 3d 30 0d  0a 44 4e 54 3a 20 31 0d
00d0  0a 55 70 67 72 61 64 65  2d 49 6e 73 65 63 75 72
00e0  65 2d 52 65 71 75 65 73  74 73 3a 20 31 0d 0a 55
00f0  73 65 72 2d 41 67 65 6e  74 3a 20 4d 6f 7a 69 6c
0100  6c 61 2f 35 2e 30 20 28  57 69 6e 64 6f 77 73 20
0110  4e 54 20 31 30 2e 30 3b  20 57 69 6e 36 34 3b 20
0120  78 36 34 29 20 41 70 70  6c 65 57 65 62 4b 69 74
0130  2f 35 33 37 2e 33 36 20  28 4b 48 54 4d 4c 2c 20
0140  6c 69 6b 65 20 47 65 63  6b 6f 29 20 43 68 72 6f
0150  6d 65 2f 31 31 39 2e 30  2e 30 2e 30 20 53 61 66
0160  61 72 69 2f 35 33 37 2e  33 36 0d 0a 4f 72 69 67
0170  69 6e 3a 20 6e 75 6c 6c  0d 0a 43 6f 6e 74 65 6e
0180  74 2d 54 79 70 65 3a 20  6d 75 6c 74 69 70 61 72
0190  74 2f 66 6f 72 6d 2d 64  61 74 61 3b 20 62 6f 75
01a0  6e 64 61 72 79 3d 2d 2d  2d 2d 57 65 62 4b 69 74
01b0  46 6f 72 6d 42 6f 75 6e  64 61 72 79 41 4a 50 4e
01c0  6c 79 58 45 61 6c 6b 42  51 74 54 32 0d 0a 41 63
01d0  63 65 70 74 3a 20 74 65  78 74 2f 68 74 6d 6c 2c
01e0  61 70 70 6c 69 63 61 74  69 6f 6e 2f 78 68 74 6d
01f0  6c 2b 78 6d 6c 2c 61 70  70 6c 69 63 61 74 69 6f
0200  6e 2f 78 6d 6c 3b 71 3d  30 2e 39 2c 69 6d 61 67
```



```
[Reassembled PDU in frame: 976]
TCP segment data (1460 bytes)
01f0  65 72 0d 0a 6f 6e 20 74  68 65 20 62 61 6e 6b 2c
0200  20 61 6e 64 20 6f 66 20  68 61 76 69 6e 67 20 6e
```

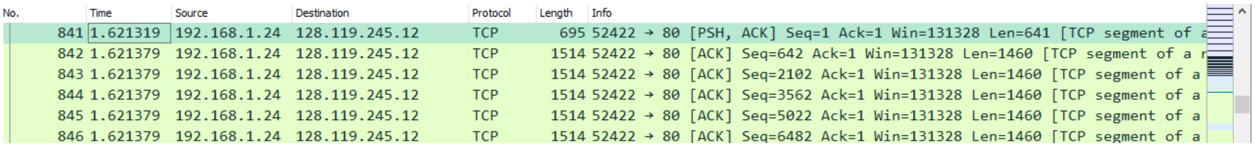
```
[Reassembled PDU in frame: 976]
TCP segment data (1460 bytes)
01f0  20 74 75 6e 6e 65 6c 20  66 6f 72 20 73 6f 6d 65
0200  20 77 61 79 2c 0d 0a 61  6e 64 20 74 68 65 6e 20
```

```
[Reassembled PDU in frame: 976]
TCP segment data (1460 bytes)
01f0  64 20 6c 65 61 72 6e 74  20 73 65 76 65 72 61 6c
0200  20 74 68 69 6e 67 73 20  6f 66 20 74 68 69 73 20
```

```
[Reassembled PDU in frame: 976]
TCP segment data (1460 bytes)
01f0  2c 20 49 27 6d 20 61 66  72 61 69 64 2c 20 62 75
0200  74 0d 0a 79 6f 75 20 6d  69 67 68 74 20 63 61 74
```

```
[Reassembled PDU in frame: 976]
TCP segment data (1460 bytes)
01f0  77 61 6c 6b 65 64 20 73  61 64 6c 79 20 64 6f 77
0200  6e 20 74 68 65 20 6d 69  64 64 6c 65 2c 0d 0a 77
```

9. What is the minimum amount of available buffer space advertised at the received for the entire trace? Does the lack of receiver buffer space ever throttle the sender?



No.	Time	Source	Destination	Protocol	Length	Info
841	1.621319	192.168.1.24	128.119.245.12	TCP	695	52422 → 80 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=641 [TCP segment of a
842	1.621379	192.168.1.24	128.119.245.12	TCP	1514	52422 → 80 [ACK] Seq=642 Ack=1 Win=131328 Len=1460 [TCP segment of a r
843	1.621379	192.168.1.24	128.119.245.12	TCP	1514	52422 → 80 [ACK] Seq=2102 Ack=1 Win=131328 Len=1460 [TCP segment of a
844	1.621379	192.168.1.24	128.119.245.12	TCP	1514	52422 → 80 [ACK] Seq=3562 Ack=1 Win=131328 Len=1460 [TCP segment of a
845	1.621379	192.168.1.24	128.119.245.12	TCP	1514	52422 → 80 [ACK] Seq=5022 Ack=1 Win=131328 Len=1460 [TCP segment of a
846	1.621379	192.168.1.24	128.119.245.12	TCP	1514	52422 → 80 [ACK] Seq=6482 Ack=1 Win=131328 Len=1460 [TCP segment of a

Since Win are all 131328, the minimum amount of buffer space is 131328. It does not throttle.

10. Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?

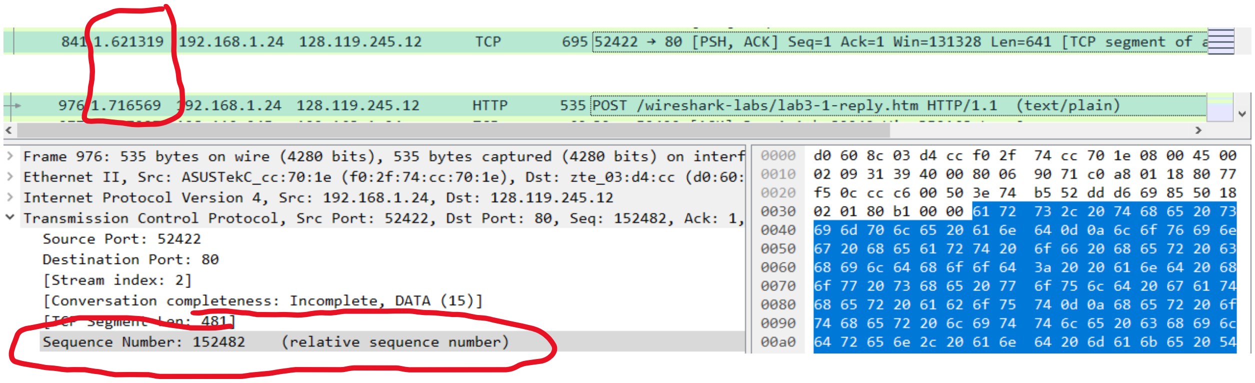
There are no retransmitted segments in the trace file. This can be explained by the fact that no same sequence number appears at two different times, thus no re-requests of previous segments.

11. How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing every other received segment (see Table 3.2 on page 250 in the text).

The ACK number increases 1420 bytes each time. I couldn't find in my cases of a receiver ACKing every other received segment.

12. What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

The throughput is the number of bytes transferred per time in which the number of bytes transferred represents the difference between the first and last segment numbers. Knowing that the first segment number is 1 and the last segment number is 152482, we can calculate the data transferred to be 152481 bytes. As for the time difference, it is 1.716569-1.621319 = 0.09525. As such, the throughput is 152482/0.09525 = 1600860.89239 bytes per second.

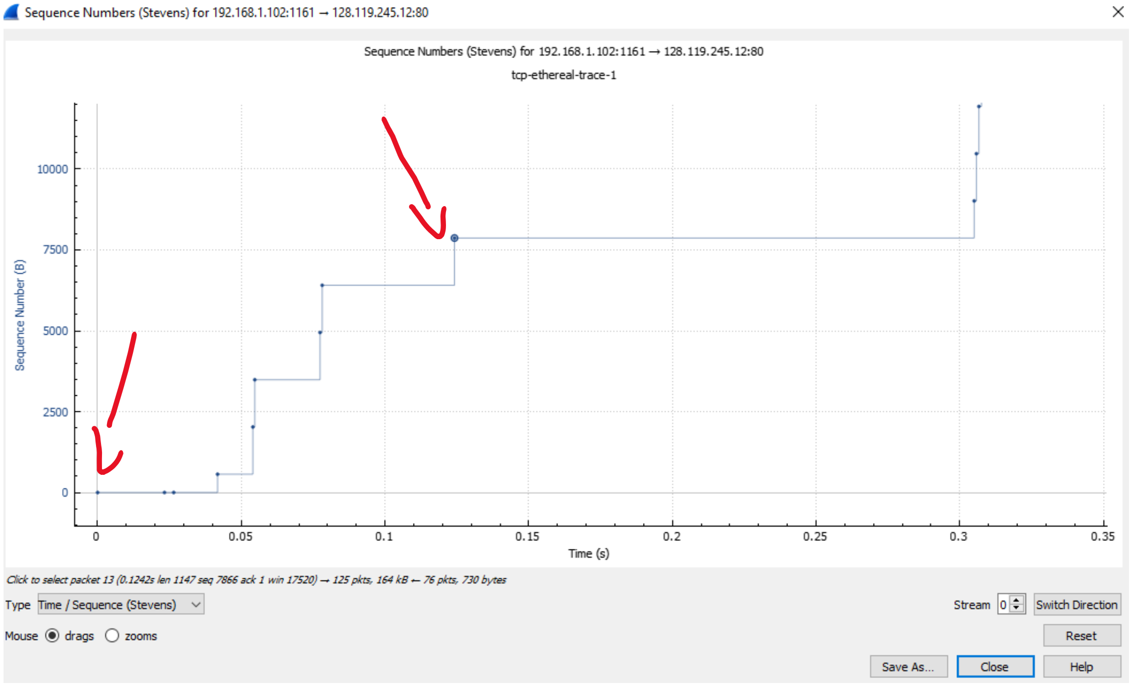


```
 841 1.621319 192.168.1.24  128.119.245.12   TCP       695 52422 → 80 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=641 [TCP segment of a

 976 1.716569 192.168.1.24  128.119.245.12   HTTP      535 POST /wireshark-labs/lab3-1-reply.htm HTTP/1.1  (text/plain)

> Frame 976: 535 bytes on wire (4280 bits), 535 bytes captured (4280 bits) on interf
> Ethernet II, Src: ASUSTekC_cc:70:1e (f0:2f:74:cc:70:1e), Dst: zte_03:d4:cc (d0:60:
> Internet Protocol Version 4, Src: 192.168.1.24, Dst: 128.119.245.12
v Transmission Control Protocol, Src Port: 52422, Dst Port: 80, Seq: 152482, Ack: 1,
    Source Port: 52422
    Destination Port: 80
    [Stream index: 2]
    [Conversation completeness: Incomplete, DATA (15)]
    [TCP Segment Len: 481]
    Sequence Number: 152482    (relative sequence number)
```

```
0000  d0 60 8c 03 d4 cc f0 2f  74 cc 70 1e 08 00 45 00
0010  02 09 31 39 40 00 80 06  90 71 c0 a8 01 18 80 77
0020  f5 0c cc c6 00 50 3e 74  b5 52 dd d6 69 85 50 18
0030  02 01 80 b1 00 00 61 72  73 2c 20 74 68 65 20 73
0040  69 6d 70 6c 65 20 61 6e  64 0d 0a 6c 6f 76 69 6e
0050  67 20 68 65 61 72 74 20  6f 66 20 68 65 72 20 63
0060  68 69 6c 64 68 6f 6f 64  3a 20 20 61 6e 64 20 68
0070  6f 77 20 73 68 65 20 77  6f 75 6c 64 20 67 61 74
0080  68 65 72 20 61 62 6f 75  74 0d 0a 68 65 72 20 6f
0090  74 68 65 72 20 6c 69 74  74 6c 65 20 63 68 69 6c
00a0  64 72 65 6e 2c 20 61 6e  64 20 6d 61 6b 65 20 54
```

13. Use the Time-Sequence-Graph (Stevens) plotting tool to view the sequence number versus time plot of segments being sent from the client to the gaia.cs.umass.edu server. Can you identify where TCP's slowstart phase begins and ends, and where congestion avoidance takes over? Comment on ways in which the measured data differs from the idealized behavior of TCP that we've studied in the text.

-The slowstart phase begins at time 0.00 seconds and ends at 0.1242 seconds.



Sequence Numbers (Stevens) for 192.168.1.102:1161 → 128.119.245.12:80

Sequence Numbers (Stevens) for 192.168.1.102:1161 → 128.119.245.12:80
tcp-ethereal-trace-1

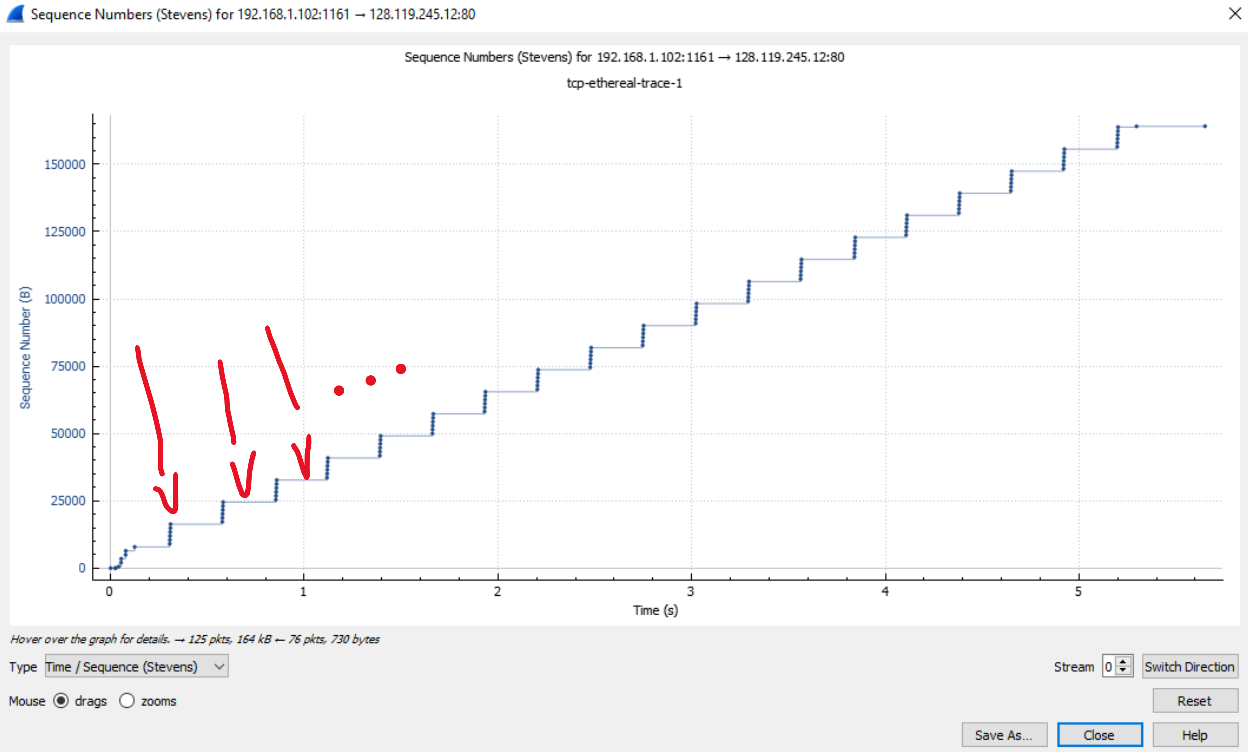
Click to select packet 13 (0.1242s len 1147 seq 7866 ack 1 win 17520) → 125 pkts, 164 kB ← 76 pkts, 730 bytes

Type Time / Sequence (Stevens)

Stream 0

Mouse ● drags ○ zooms

-It is in congestion (horizontal line) every 6 points


Sequence Numbers (Stevens) for 192.168.1.102:1161 → 128.119.245.12:80

Sequence Numbers (Stevens) for 192.168.1.102:1161 → 128.119.245.12:80

tcp-ethereal-trace-1

Hover over the graph for details. → 125 pkts, 164 kB ← 76 pkts, 730 bytes

Type Time / Sequence (Stevens)

Stream 0

Mouse ● drags ○ zooms

-The ideal behavior of a TCP connection is to allow data to be transmitted as fast as possible with no lost in data while also not too fast which results in continuous queuing delay. In this case, there is a difference where the duration of each congestion has a fluctuation, thus different delay each time. The slowstart can also vary depending on the TCP connections.

3. <u>Concepts Learned from this Lab</u>

For Mininet lab, I learned abut the effect of bandwidth and delay on a network. I also practiced with “iPerf3”, a real-time throughput measurement tool, to create a two switch and three hosts topology network. For Wireshark lab, I practice on a basic TCP protocol and learned about sequence and acknowledgement numbers. Finally, I worked on TCP congestion control in action.