Computer Networks - Assignment 2

Nimitt (22110169)

Sumeet Sawale (22110234)

Task 1: Comparison of congestion control protocols

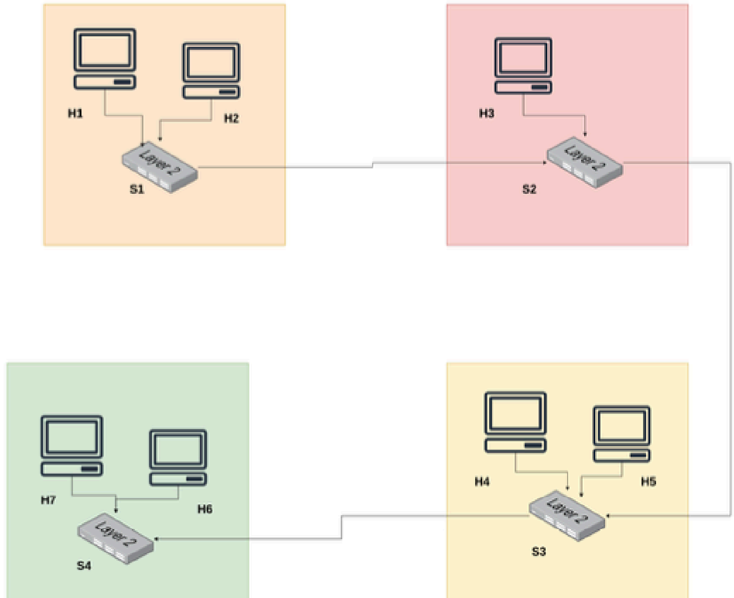
Setup and Tools Used

- UTM Virtual Machine Ubuntu 20.04
- Mininet
- Wireshark

Implementation

Setting Up the Network

- We use Mininet to setup the desired network configuration as per the input option:




```
def setup_topology(option, link_loss=None):
    class CustomTopo(Topo):
        def build(self):
            s1 = self.addSwitch('s1')
            s2 = self.addSwitch('s2')
            s3 = self.addSwitch('s3')
            s4 = self.addSwitch('s4')

            h1 = self.addHost('h1')
            h2 = self.addHost('h2')
            h3 = self.addHost('h3')
            h4 = self.addHost('h4')
            h5 = self.addHost('h5')
            h6 = self.addHost('h6')
            h7 = self.addHost('h7')

            self.addLink(h1, s1)
            self.addLink(h2, s1)
            self.addLink(h3, s2)
```

```
        self.addLink(h4, s2)
        self.addLink(h5, s3)
        self.addLink(h6, s3)
        self.addLink(h7, s4)

        self.addLink(s1, s2)
        self.addLink(s2, s3)
        self.addLink(s3, s4)

    net = Mininet(topo=CustomTopo(), controller=OVSController)
    net.start()

    for switch in ['s1', 's2', 's3', 's4']:
        net.get(switch).cmd('ovs-ofctl add-flow %s actions=normal' % switch)

    if option in ['c', 'd']:
        net.get('s1').cmd('tc qdisc add dev s1-eth1 root tbf rate 100Mbit burst 10kb la
        net.get('s2').cmd('tc qdisc add dev s2-eth2 root tbf rate 50Mbit burst 5kb la
        net.get('s3').cmd('tc qdisc add dev s3-eth3 root tbf rate 100Mbit burst 10kb

        if option == 'd' and link_loss is not None:
            net.get('s2').cmd(f'tc qdisc add dev s2-eth2 root netem loss {link_loss}%

    return net
```

Running Test

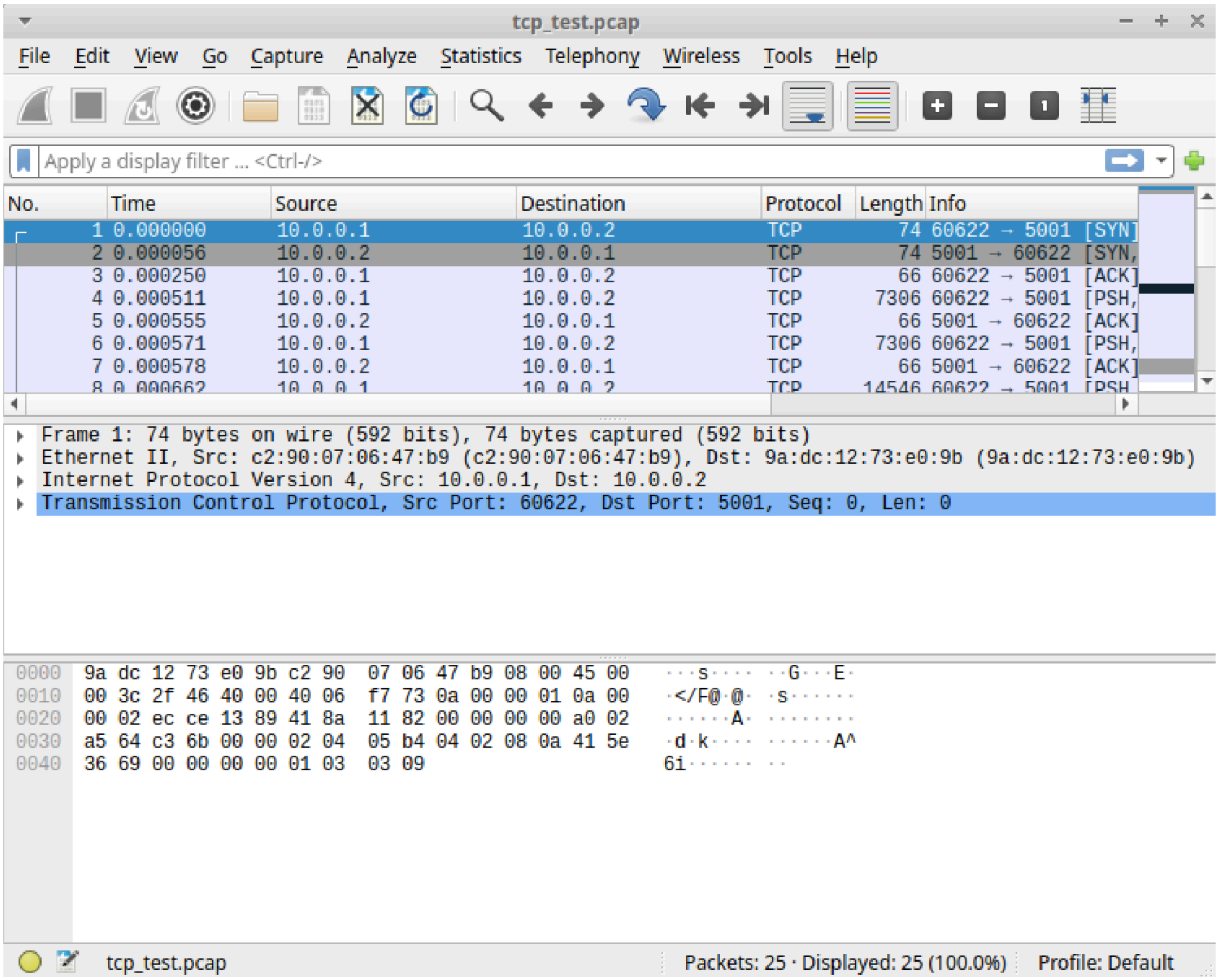
- As per the option, we set up the configuration and save the network flow statisitcs for further metric computation:

```
def run_iperf_test(net, option, cc_scheme, condition=None, link_loss=None):
    server = net.get('h7')
    server.cmd('tcpdump -i h7-eth0 port 5001 -w iperf_capture.pcap &')
    server.cmd('iperf3 -s -p 5001 &')
    time.sleep(2)
```

```
    clients = {
        'a': [('h1', 0, 150)],
        'b': [('h1', 0, 150), ('h3', 15, 120), ('h4', 30, 90)],
        'c': {  # Map conditions to client-server setups
            'a': [('h1', 0, 150), ('h2', 0, 150)],
            'b': [('h1', 0, 150), ('h3', 0, 150)],
            'c': [('h1', 0, 150), ('h3', 0, 150), ('h4', 0, 150)]
        },
        'd': {  # Loss configurations
            'a': [('h1', 0, 150), ('h2', 0, 150)],
            'b': [('h1', 0, 150), ('h3', 0, 150)],
            'c': [('h1', 0, 150), ('h3', 0, 150), ('h4', 0, 150)]
        }
    }

    test_clients = clients[option] if option in ['a', 'b'] else clients[option][condition]

    for host, delay, duration in test_clients:
        time.sleep(delay)
        client = net.get(host)
        client.cmd(f'iperf3 -c 10.0.0.7 -p 5001 -b 10M -P 10 -t {duration} -C {cc_sche

    time.sleep(160)
    server.cmd('pkill tcpdump')
    analyze_pcap()
    print("Tests completed! Results saved to iperf_capture.pcap and *_{cc_scheme
```

Analysing Metrics

- We analyse and compute the required metrics using saved network flow statistics using Wireshark GUI and tshark:




```
def analyze_pcap():
    print("Analyzing pcap file for performance metrics...")
    throughput = subprocess.getoutput("tshark -r iperf_capture.pcap -q -z io,stat,
    goodput = subprocess.getoutput("tshark -r iperf_capture.pcap -Y 'tcp.len > 0'
    packet_loss = subprocess.getoutput("tshark -r iperf_capture.pcap -q -z exper
    max_packet_size = subprocess.getoutput("tshark -r iperf_capture.pcap -T fiel

    print("Throughput:", throughput)
    print("Goodput (total data packets received):", goodput)
    print("Packet loss rate:", packet_loss)
    print("Maximum packet size achieved:", max_packet_size)
```

Results

Part (a)

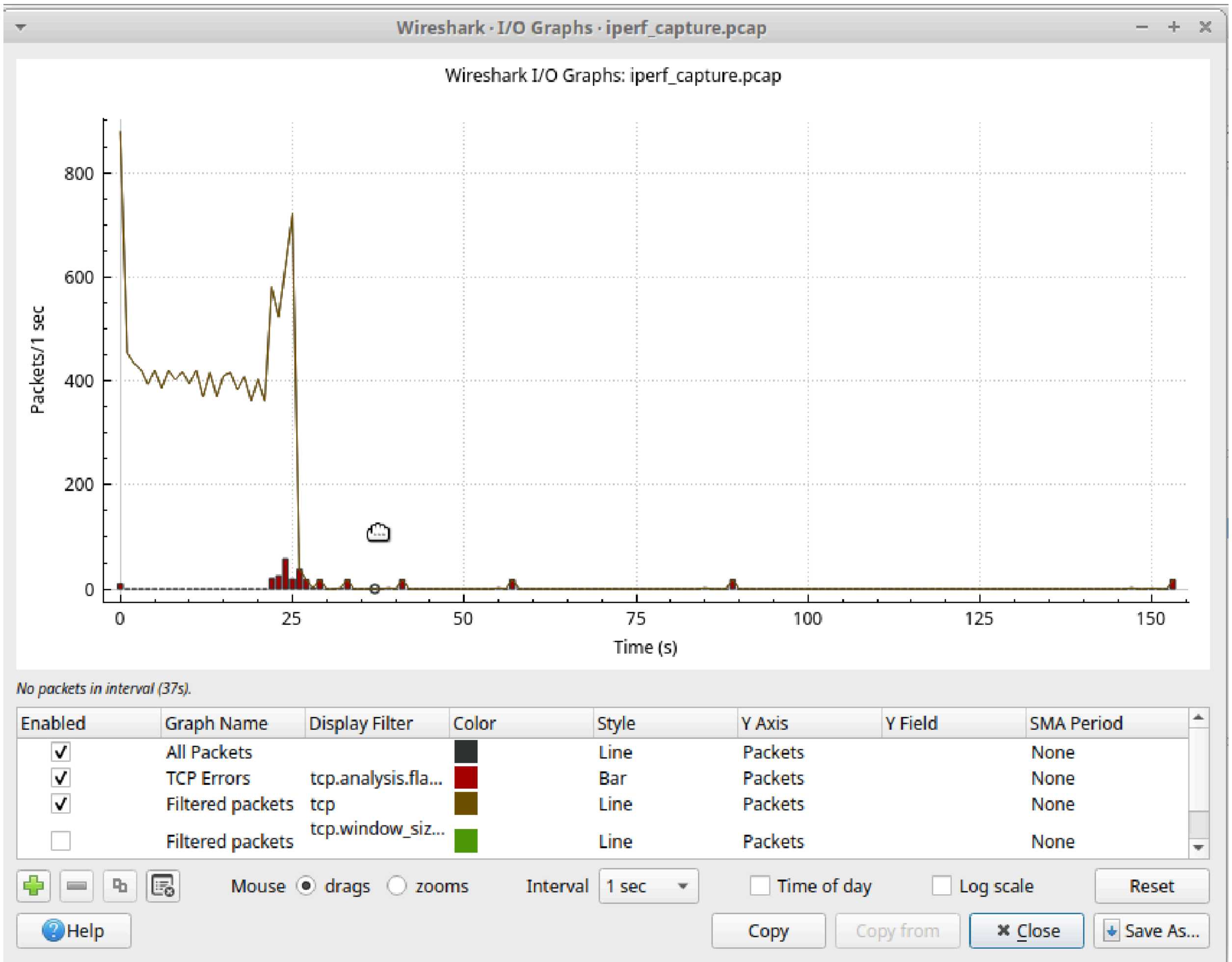
- Congestion Control = ‘yeah’

Throughput:

Using Tshark

Time Interval (s)	Packets	Bytes
0 <> 10	4626	126135170
10 <> 20	3947	124778902
20 <> 30	3284	72659524
30 <> 40	22	1452
40 <> 50	18	1188
50 <> 60	20	1320
60 <> 70	0	0
70 <> 80	0	0
80 <> 90	20	1320
90 <> 100	0	0
100 <> 110	0	0
110 <> 120	0	0
120 <> 130	0	0
130 <> 140	0	0
140 <> 150	2	132
150 <> Dur	20	1321

IO Graph



Goodput:

We define goodput as,

$$\frac{Good\ Packets}{Total\ Packets} * 100$$

= 67%

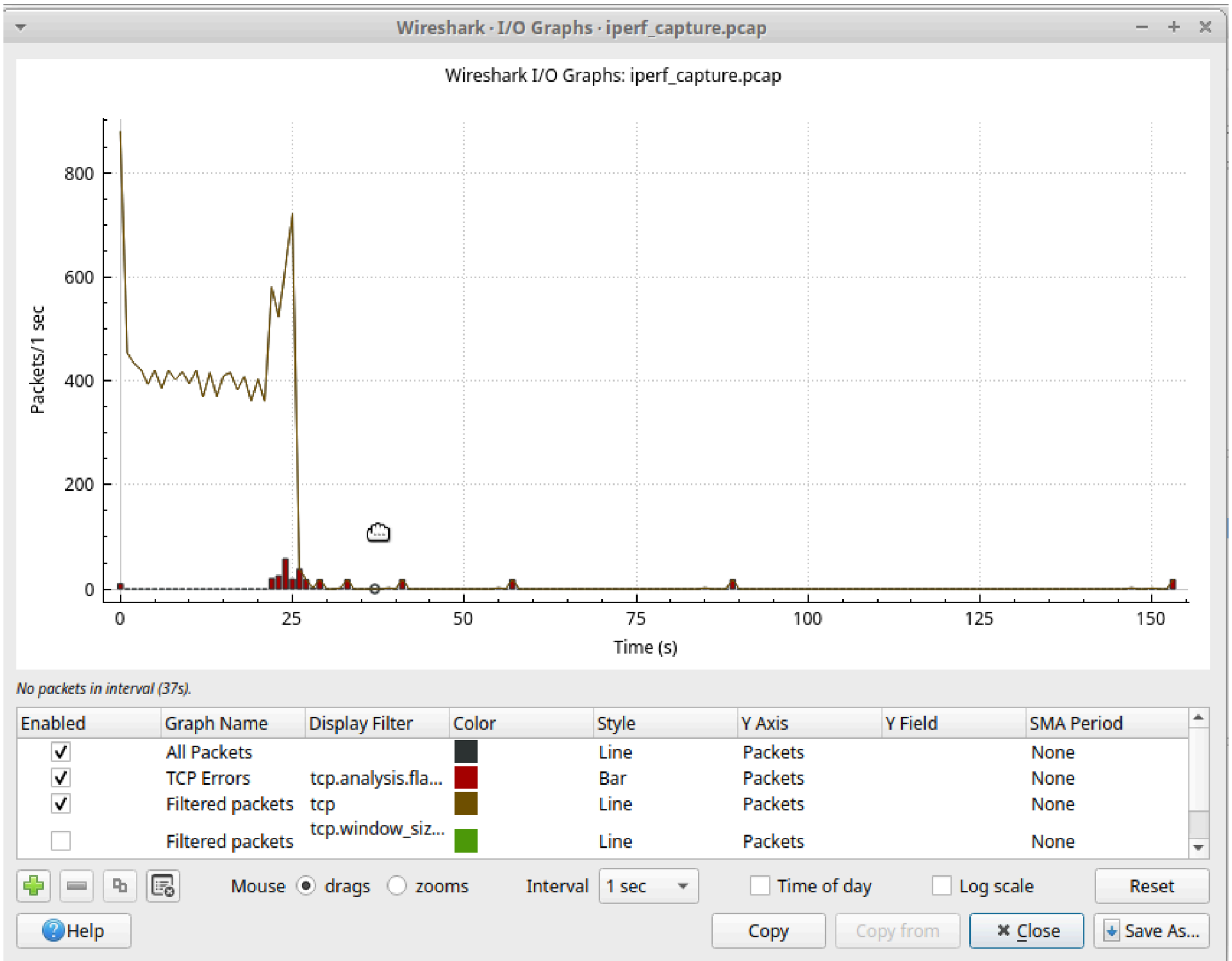
Packet Loss: 0%

Maximum Packet Size: 65226 bytes

- Congestion Control = 'bbr'

Throughput:

Time Interval (s)	Packets	Bytes
0 <> 10	4821	126154475
10 <> 20	4330	124804180
20 <> 30	4168	73388664
30 <> 40	89	1764594
40 <> 50	14	924
50 <> 60	18	1188
60 <> 70	2	132
70 <> 80	0	0
80 <> 90	16	1056
90 <> 100	4	264
100 <> 110	0	0
110 <> 120	0	0
120 <> 130	0	0
130 <> 140	0	0
140 <> 150	10	660
150 <> Dur	12	793



Goodput: 64%

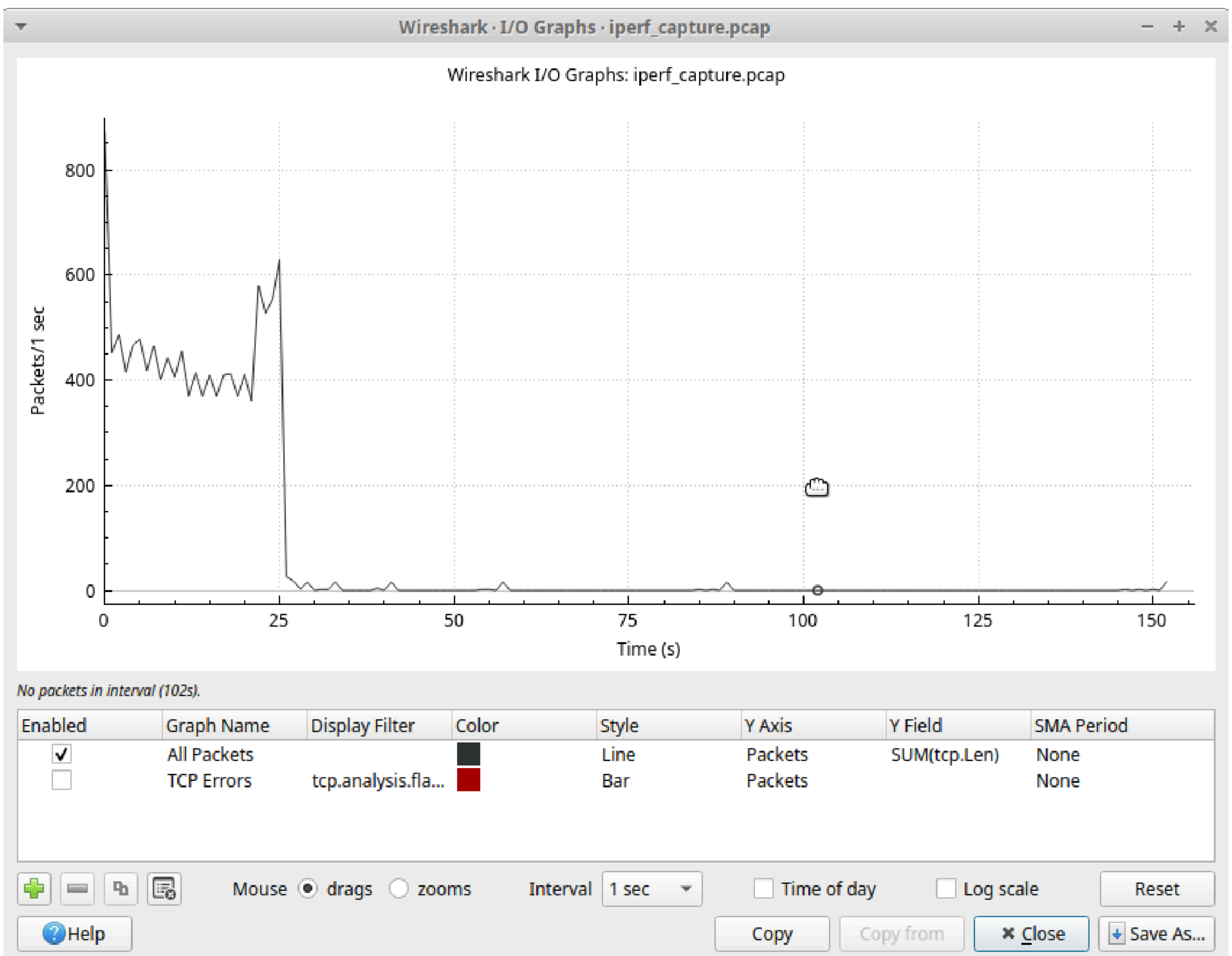
Packet Loss: 1%

- Congestion Control = ‘westwood’

Throughput:

Time Interval (s)	Packets	Bytes
0 <> 10	4895	126152928
10 <> 20	3984	124781344
20 <> 30	3124	70349044
30 <> 40	24	1584
40 <> 50	16	1056

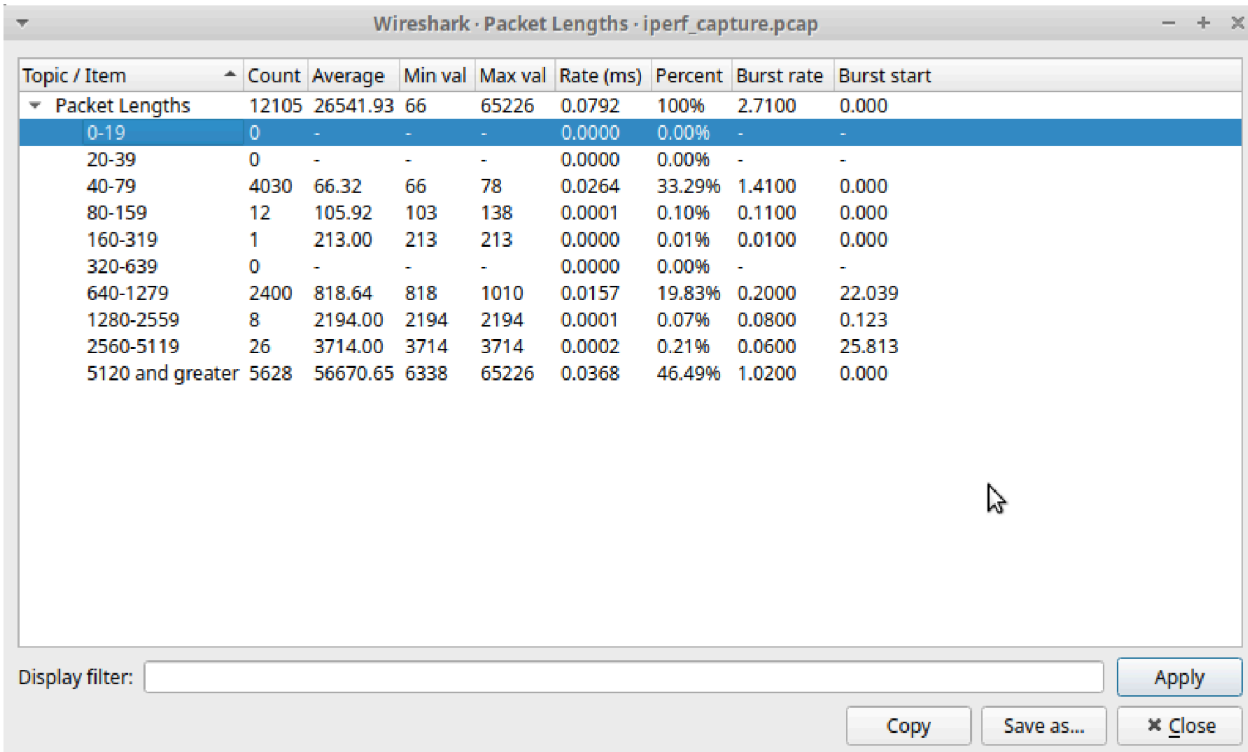
50 <> 60	20	1320
60 <> 70	0	0
70 <> 80	0	0
80 <> 90	20	1320
90 <> 100	0	0
100 <> 110	0	0
110 <> 120	0	0
120 <> 130	0	0
130 <> 140	0	0
140 <> 150	4	264
150 <> Dur	18	1189



Goodput: 69%

Packet Loss: 1

max window size: 61452 bytes



Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Packet Lengths	12105	26541.93	66	65226	0.0792	100%	2.7100	0.000
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	4030	66.32	66	78	0.0264	33.29%	1.4100	0.000
80-159	12	105.92	103	138	0.0001	0.10%	0.1100	0.000
160-319	1	213.00	213	213	0.0000	0.01%	0.0100	0.000
320-639	0	-	-	-	0.0000	0.00%	-	-
640-1279	2400	818.64	818	1010	0.0157	19.83%	0.2000	22.039
1280-2559	8	2194.00	2194	2194	0.0001	0.07%	0.0800	0.123
2560-5119	26	3714.00	3714	3714	0.0002	0.21%	0.0600	25.813
5120 and greater	5628	56670.65	6338	65226	0.0368	46.49%	1.0200	0.000

Obervations

- All the congestion control methods perform very similarly.
- WestWood gives the best throughput because of its conservative congestion window strategy.

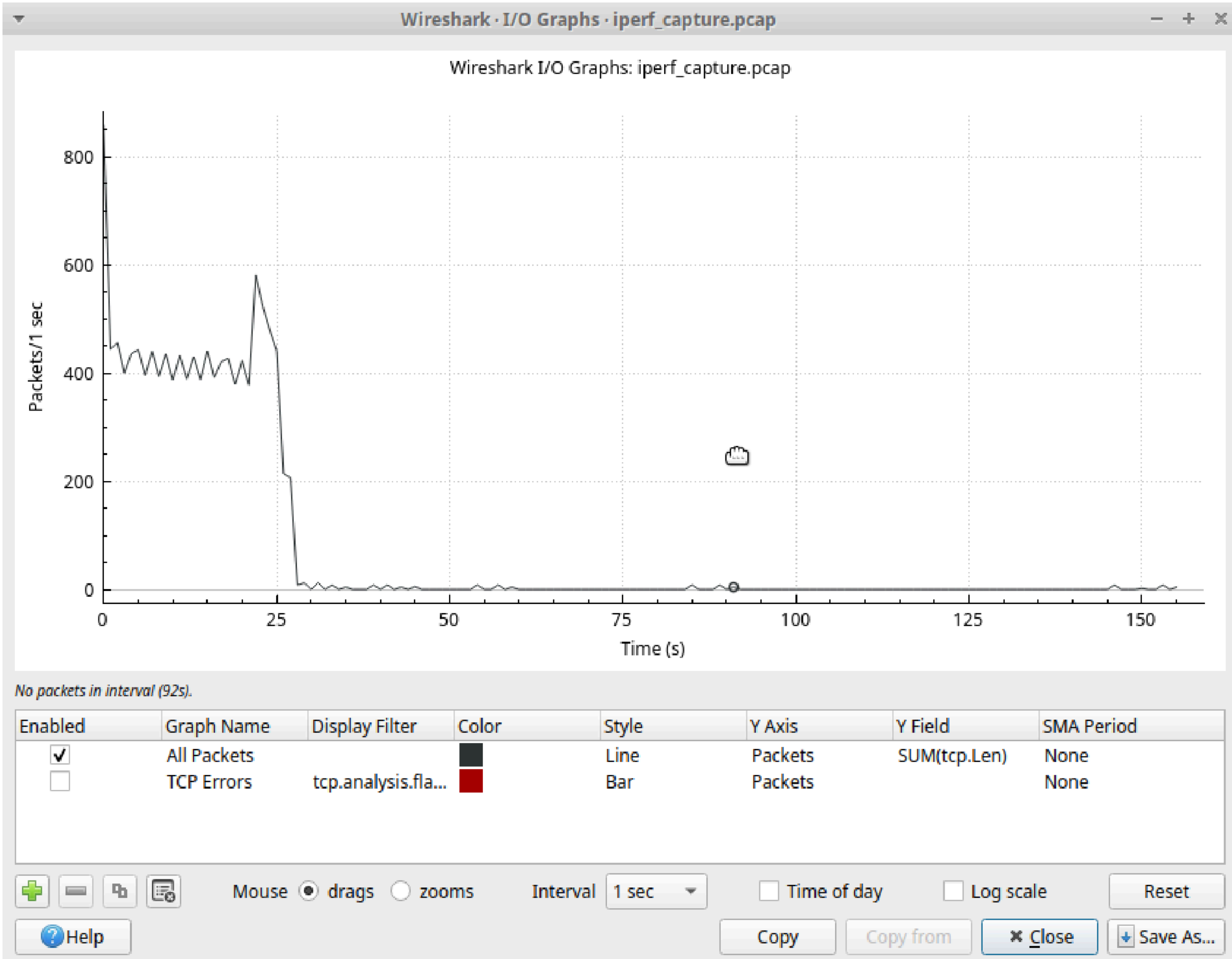
Part (b)

- Congestion Control = ‘yeah’

Throughput:

Time Interval (s)	Packets	Bytes
0 <> 10	4702	126140186
10 <> 20	4087	124788184

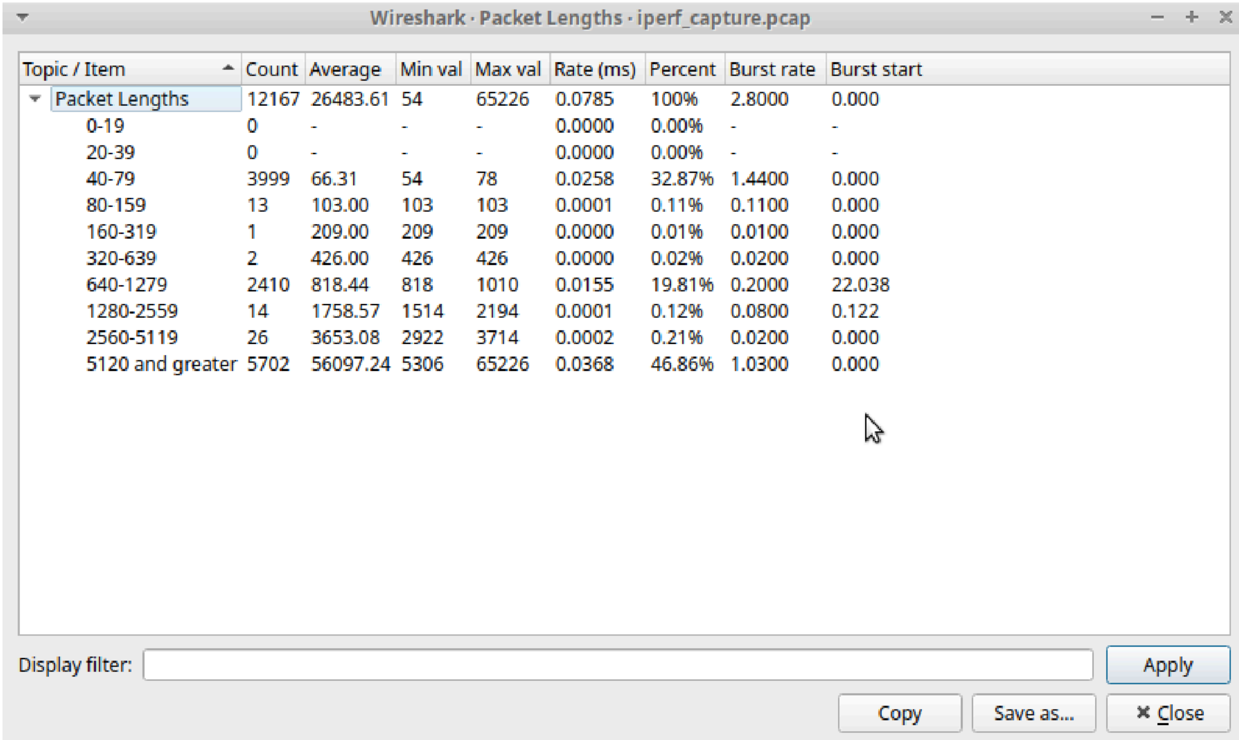
20 <> 30	3267	71290346
30 <> 40	32	2112
40 <> 50	17	1175
50 <> 60	20	1320
60 <> 70	0	0
70 <> 80	0	0
80 <> 90	16	1056
90 <> 100	4	264
100 <> 110	0	0
110 <> 120	0	0
120 <> 130	0	0
130 <> 140	0	0
140 <> 150	8	528
150 <> Dur	14	925



Goodput: 68%

Packet Loss: 1.5

Maximum Packet Size: 65226



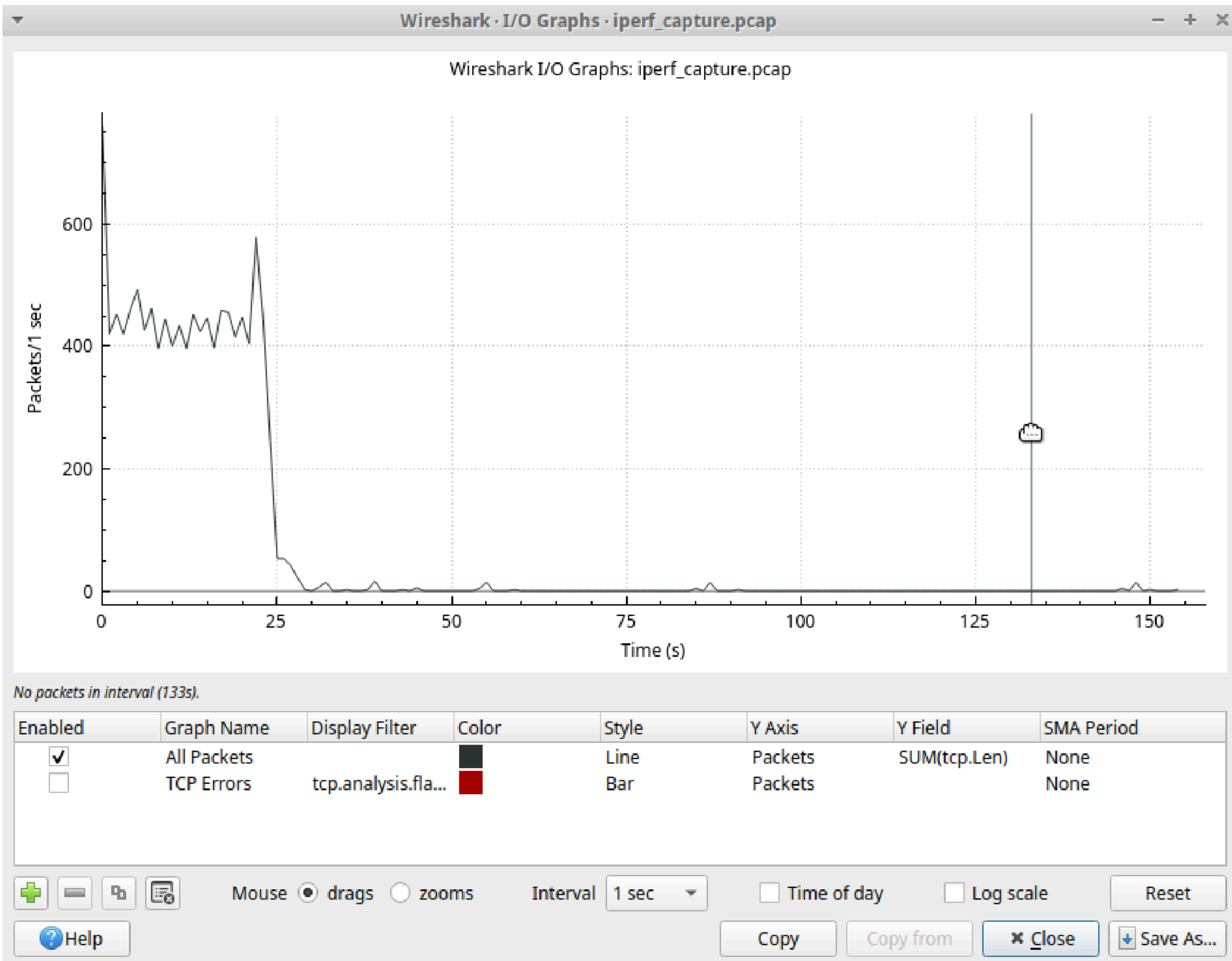
Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Packet Lengths	12167	26483.61	54	65226	0.0785	100%	2.8000	0.000
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	3999	66.31	54	78	0.0258	32.87%	1.4400	0.000
80-159	13	103.00	103	103	0.0001	0.11%	0.1100	0.000
160-319	1	209.00	209	209	0.0000	0.01%	0.0100	0.000
320-639	2	426.00	426	426	0.0000	0.02%	0.0200	0.000
640-1279	2410	818.44	818	1010	0.0155	19.81%	0.2000	22.038
1280-2559	14	1758.57	1514	2194	0.0001	0.12%	0.0800	0.122
2560-5119	26	3653.08	2922	3714	0.0002	0.21%	0.0200	0.000
5120 and greater	5702	56097.24	5306	65226	0.0368	46.86%	1.0300	0.000

- Congestion Control = ‘bbr’

Throughput:

Time Interval (s)	Packets	Bytes
0 <> 10	4733	126142231
10 <> 20	4279	124800868
20 <> 30	2288	60280764
30 <> 40	40	2640
40 <> 50	7	515
50 <> 60	20	1320
60 <> 70	0	0
70 <> 80	0	0
80 <> 90	18	1188
90 <> 100	2	132
100 <> 110	0	0

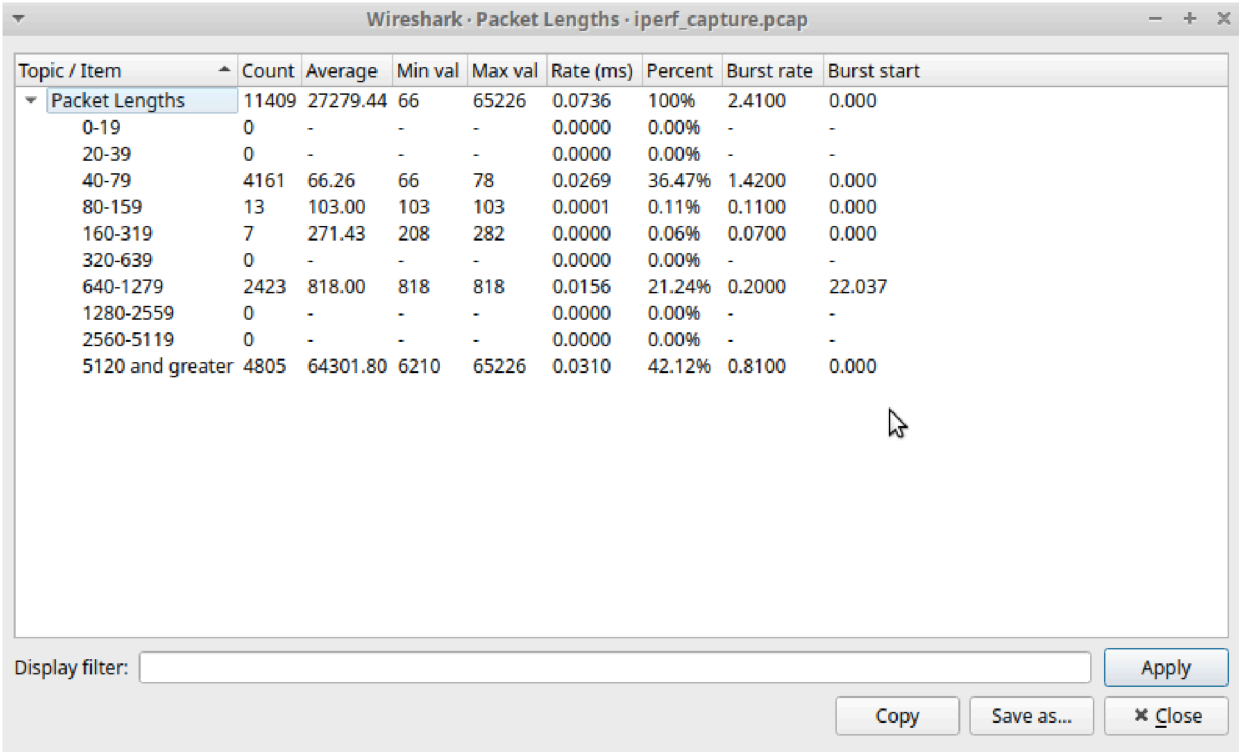
110 <> 120	0	0
120 <> 130	0	0
130 <> 140	0	0
140 <> 150	18	1188
150 <> Dur	4	265



Goodput: 60%

Packet Loss: 2%

Maximum Packet Size: 614579



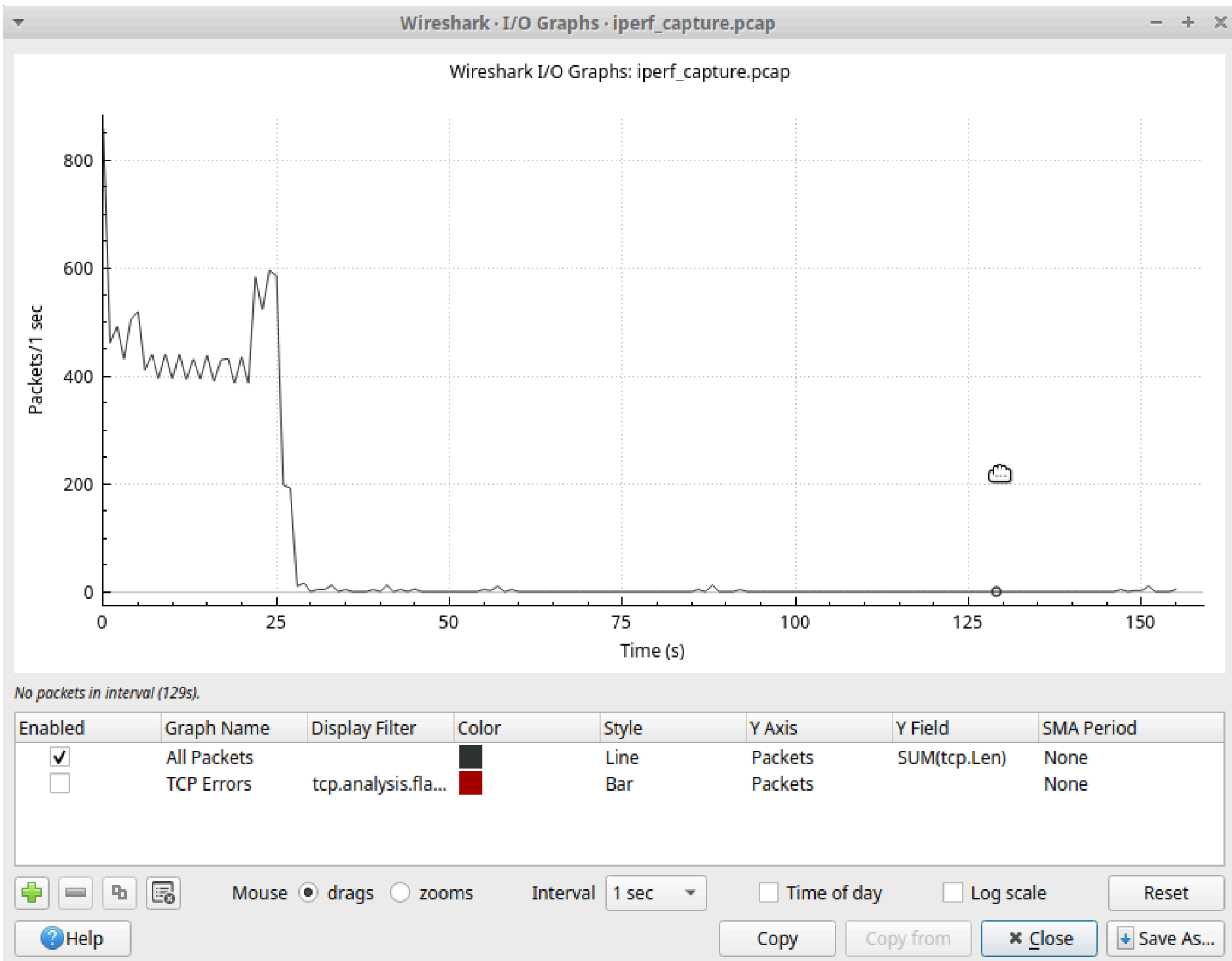
Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Packet Lengths	11409	27279.44	66	65226	0.0736	100%	2.4100	0.000
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	4161	66.26	66	78	0.0269	36.47%	1.4200	0.000
80-159	13	103.00	103	103	0.0001	0.11%	0.1100	0.000
160-319	7	271.43	208	282	0.0000	0.06%	0.0700	0.000
320-639	0	-	-	-	0.0000	0.00%	-	-
640-1279	2423	818.00	818	818	0.0156	21.24%	0.2000	22.037
1280-2559	0	-	-	-	0.0000	0.00%	-	-
2560-5119	0	-	-	-	0.0000	0.00%	-	-
5120 and greater	4805	64301.80	6210	65226	0.0310	42.12%	0.8100	0.000

- Congestion Control = 'westwood'

Throughput:

Time Interval (s)	Packets	Bytes
0 <> 10	4955	126163000
10 <> 20	4133	124791232
20 <> 30	3526	75895348
30 <> 40	28	1848
40 <> 50	21	1439
50 <> 60	20	1320
60 <> 70	0	0
70 <> 80	0	0
80 <> 90	16	1056
90 <> 100	4	264
100 <> 110	0	0

110 <> 120	0	0
120 <> 130	0	0
130 <> 140	0	0
140 <> 150	6	396
150 <> Dur	16	1057



Goodput: 65

Packet Loss: 1

Observations

- YeAH reduces bandwidth competitively so we see more contiuous spiky variations

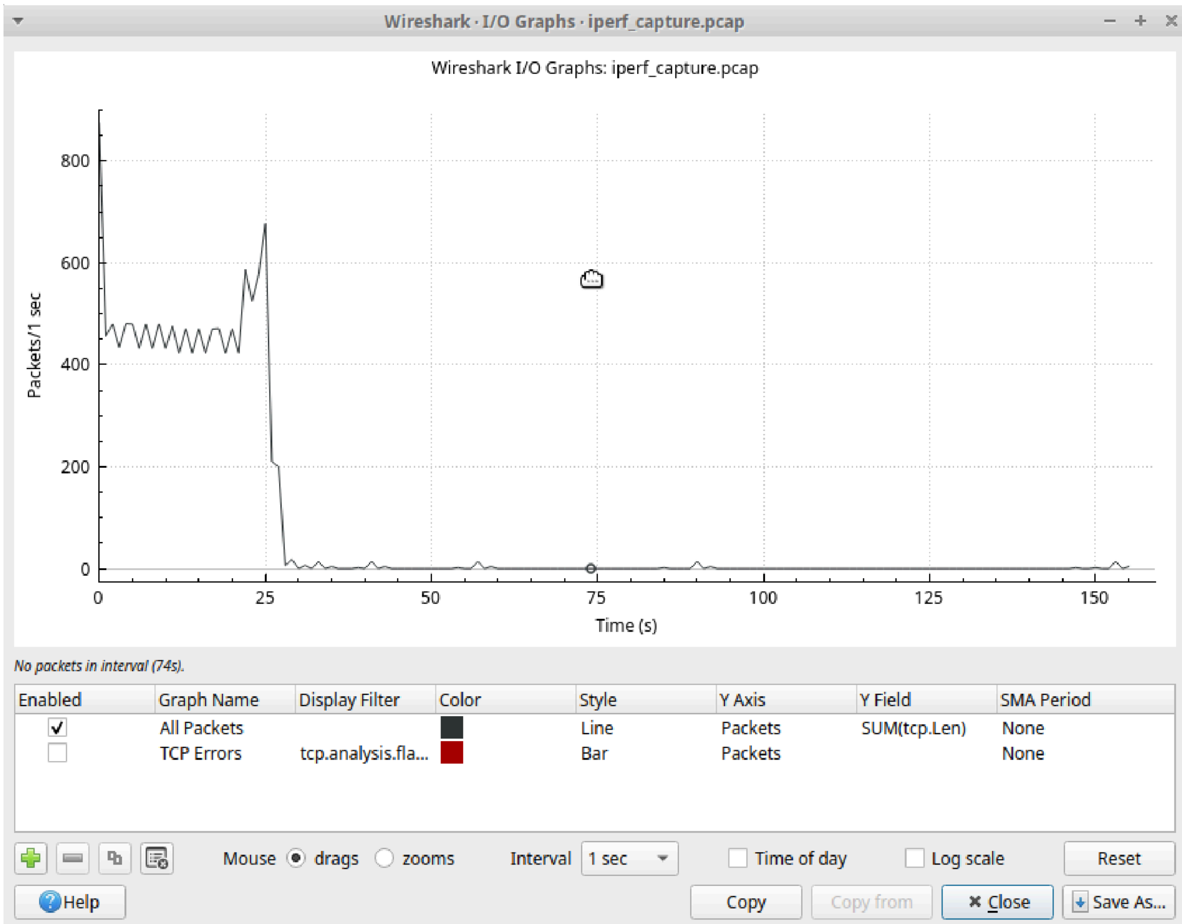
- BBR adjusts based on measured bandwidth and round-trip time, ensuring better stability.
- Westwood relies on estimated available bandwidth, meaning it takes longer to ramp up in shared environments.

Part (c)

- Condition 1
  - Congestion Control = ‘yeah’

  Throughput:

Duration Range	Count	Value
0 <> 10	5029	126161768
10 <> 20	4481	124814146
20 <> 30	3694	77055220
30 <> 40	26	1716
40 <> 50	18	1188
50 <> 60	20	1320
60 <> 70	0	0
70 <> 80	0	0
80 <> 90	2	132
90 <> 100	18	1188
100 <> 110	0	0
110 <> 120	0	0
120 <> 130	0	0
130 <> 140	0	0
140 <> 150	2	132
150 <> Dur	20	1321



Goodput: 60

Packet Loss: 1.5

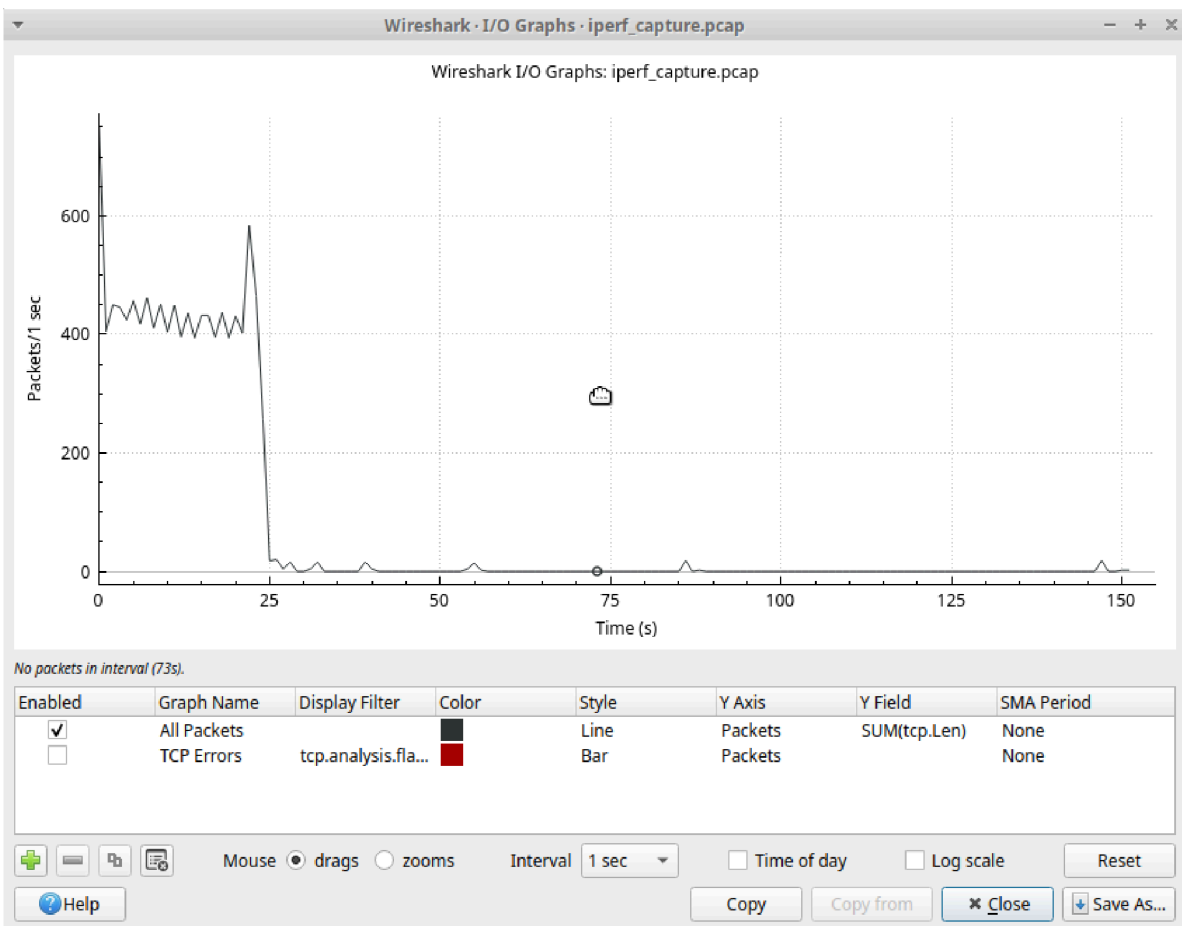
Mps: 65226

- Congestion Control = 'bbr'

Throughput:

Duration Range	Count	Value
0 <> 10	5029	126161768
10 <> 20	4481	124814146
20 <> 30	3694	77055220
30 <> 40	26	1716
40 <> 50	18	1188

50 <> 60	20	1320
60 <> 70	0	0
70 <> 80	0	0
80 <> 90	2	132
90 <> 100	18	1188
100 <> 110	0	0
110 <> 120	0	0
120 <> 130	0	0
130 <> 140	0	0
140 <> 150	2	132
150 <> Dur	20	1321



Goodput:65

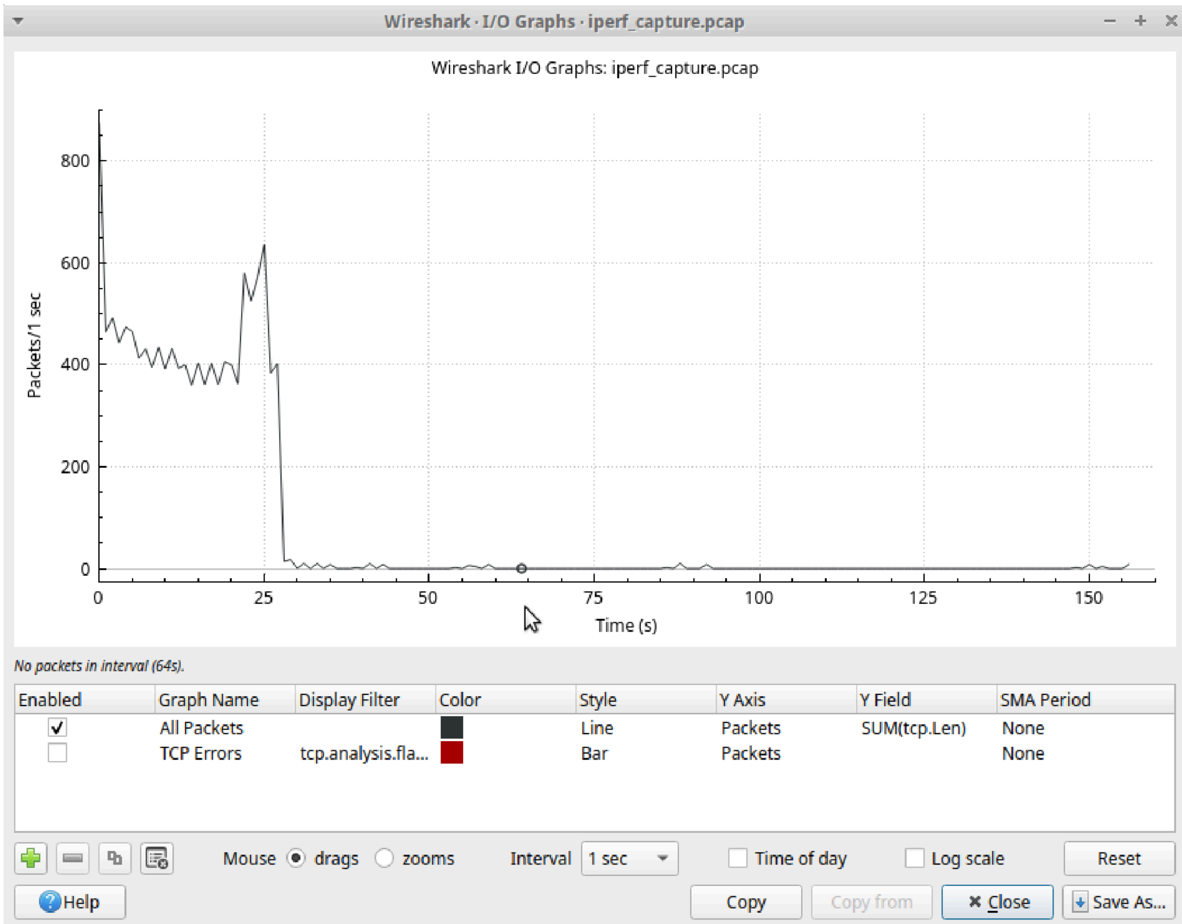
Packet Loss:0.5

MPS: 65226

- Congestion Control = ‘westwood’

Throughput:

Duration Range	Count	Value
0 <> 10	5029	126161768
10 <> 20	4481	124814146
20 <> 30	3694	77055220
30 <> 40	26	1716
40 <> 50	18	1188
50 <> 60	20	1320
60 <> 70	0	0
70 <> 80	0	0
80 <> 90	2	132
90 <> 100	18	1188
100 <> 110	0	0
110 <> 120	0	0
120 <> 130	0	0
130 <> 140	0	0
140 <> 150	2	132
150 <> Dur	20	1321



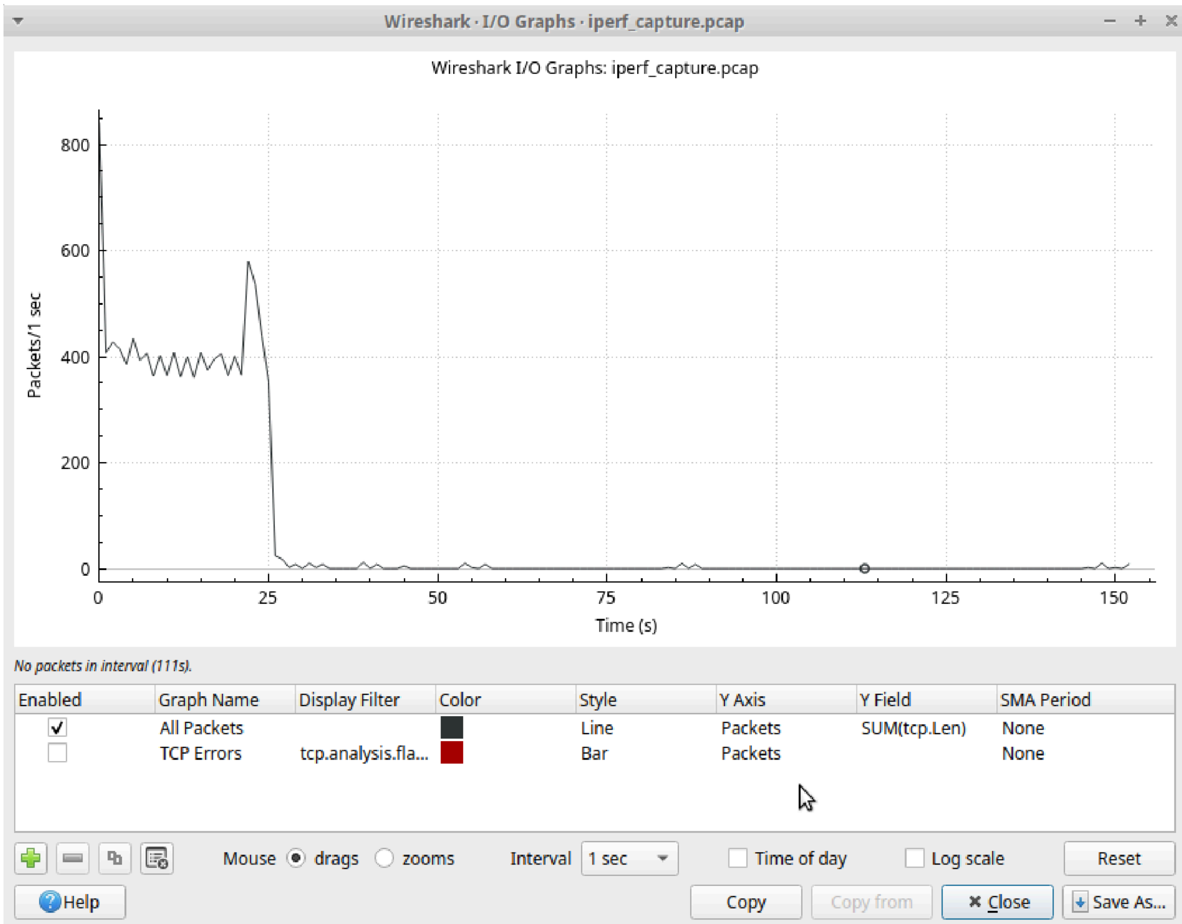
Goodput: 68

Packet Loss: 1.5

MPS: 65226

- Condition 2
  - Congestion Control = ‘yeah’

Throughput:



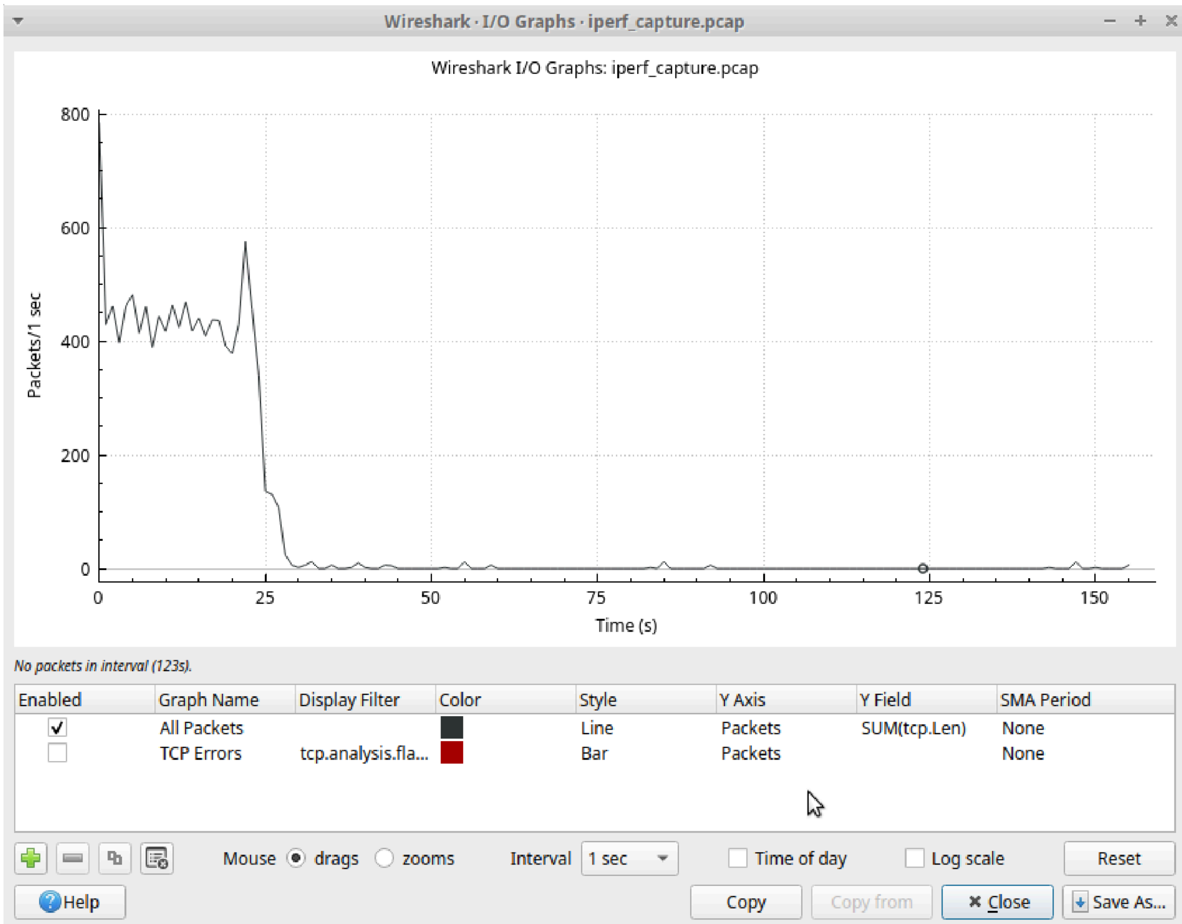
Goodput:65

Packet Loss:0.5

MPS: 65226

- Congestion Control = bbr’

Throughput:



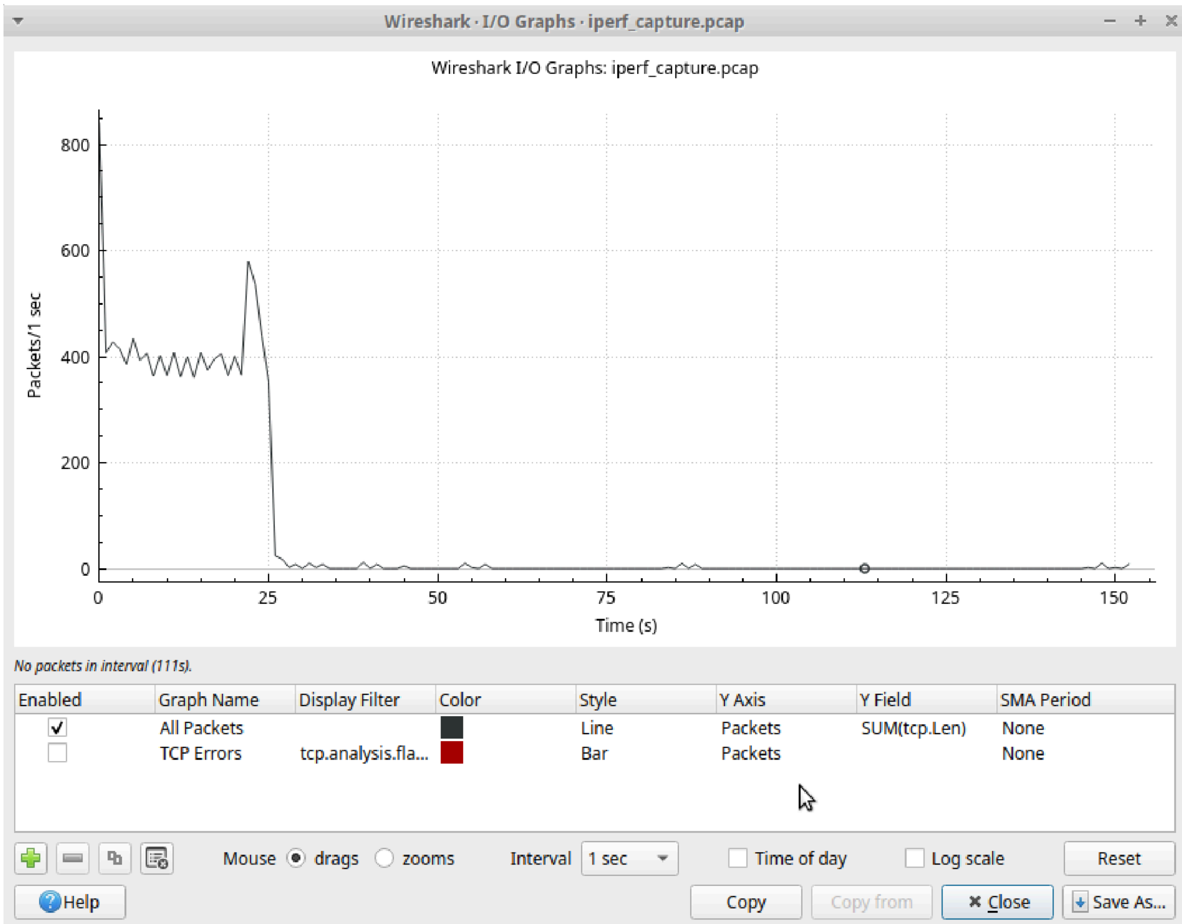
Goodput:65

Packet Loss:0.5

MPS: 65226

- Congestion Control = ‘westwood’

Throughput:



Goodput:65

Packet Loss:0.5

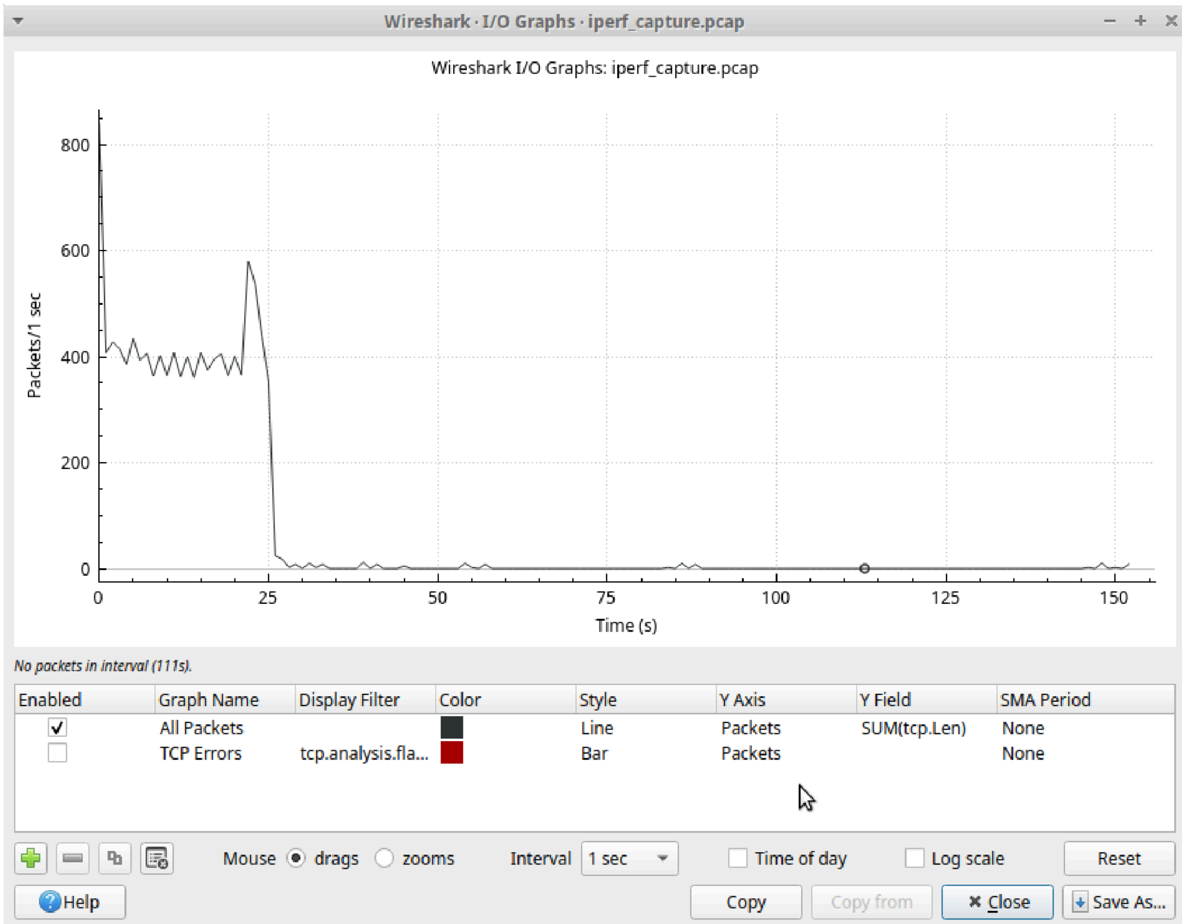
MPS: 65226

- Congestion Control = 'bbr'

Throughput:

- Condition 3
    - Congestion Control = 'yeah'

  Throughput:



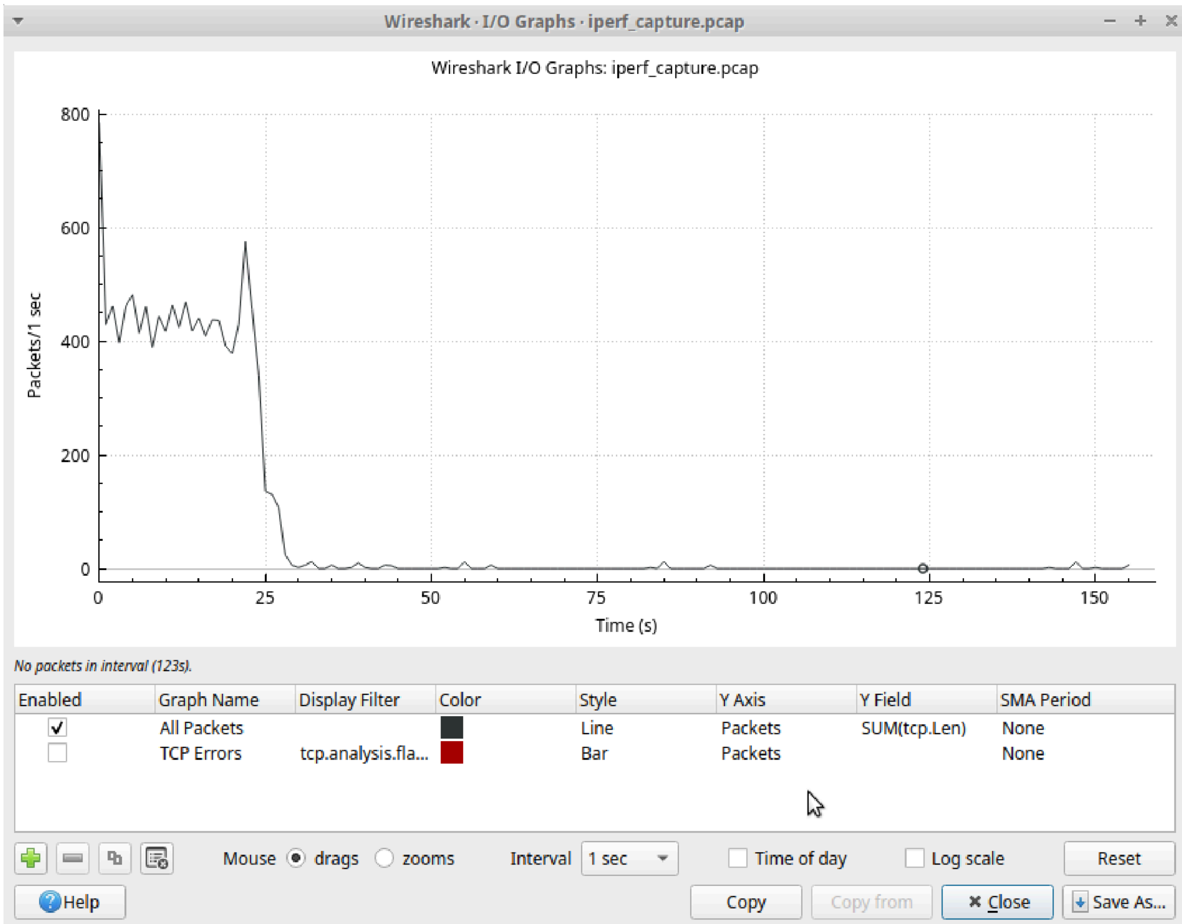
Goodput:65

Packet Loss:0.5

MPS: 65226

- Congestion Control = ‘bbr’

Throughput:

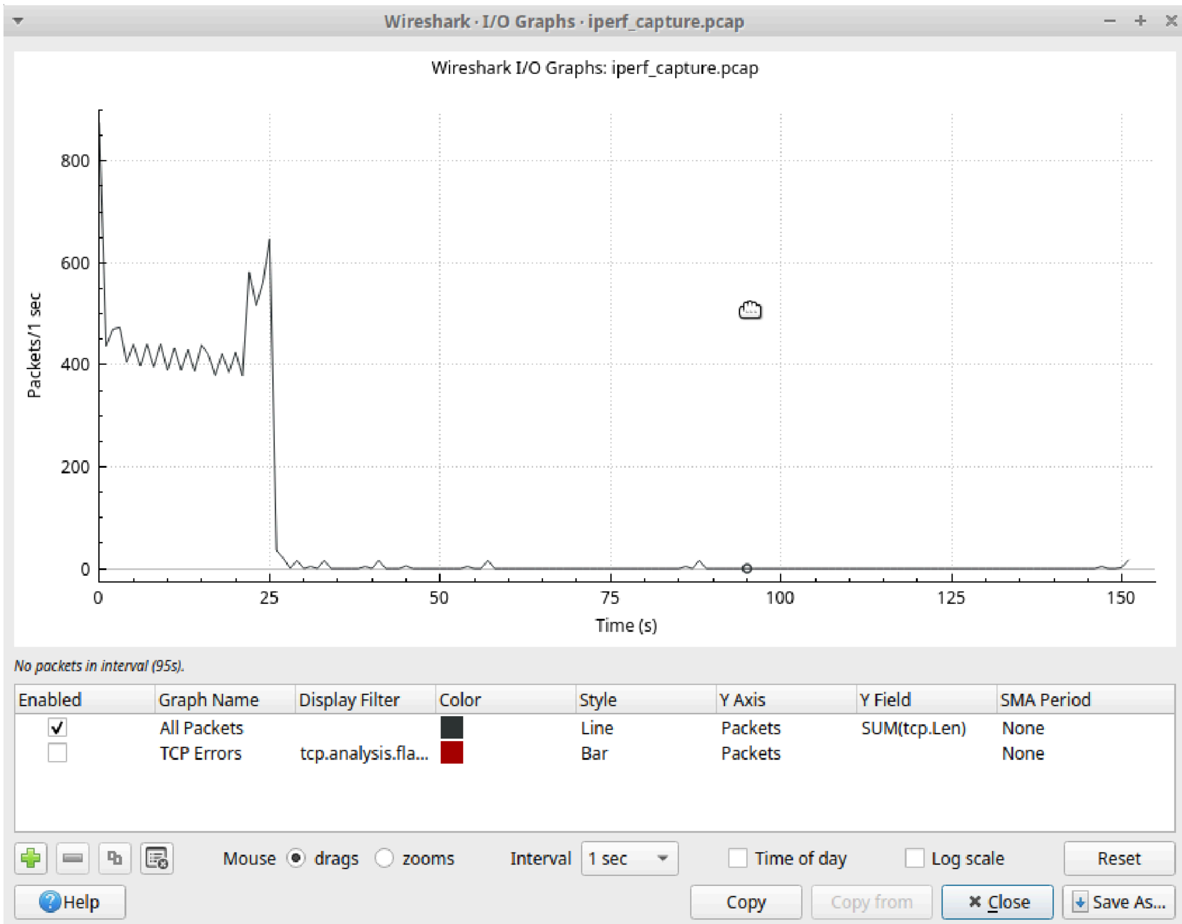


Goodput:61

Packet Loss:1

- Congestion Control = ‘westwood’

Throughput:



Goodput: 67

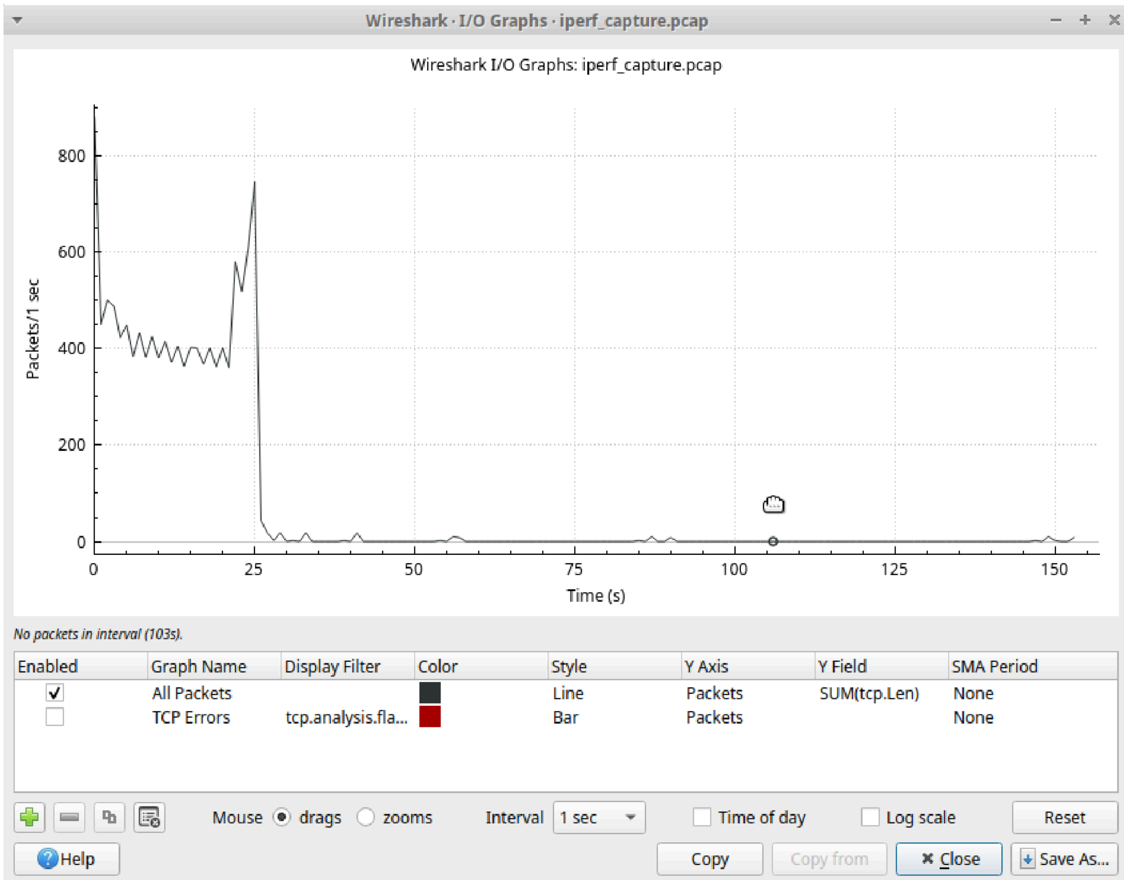
Packet Loss: 0.5

Obervations:

- YeAH is affected as it reacts to loss by switching modes.
- BBR remains steady because it does not use packet loss as a congestion signal.
- Westwood sees degradation since it estimates bandwidth based on successful transmissions.

Part (d)

- Condition 1: loss = 1%
    - Congestion Control = ‘yeah’



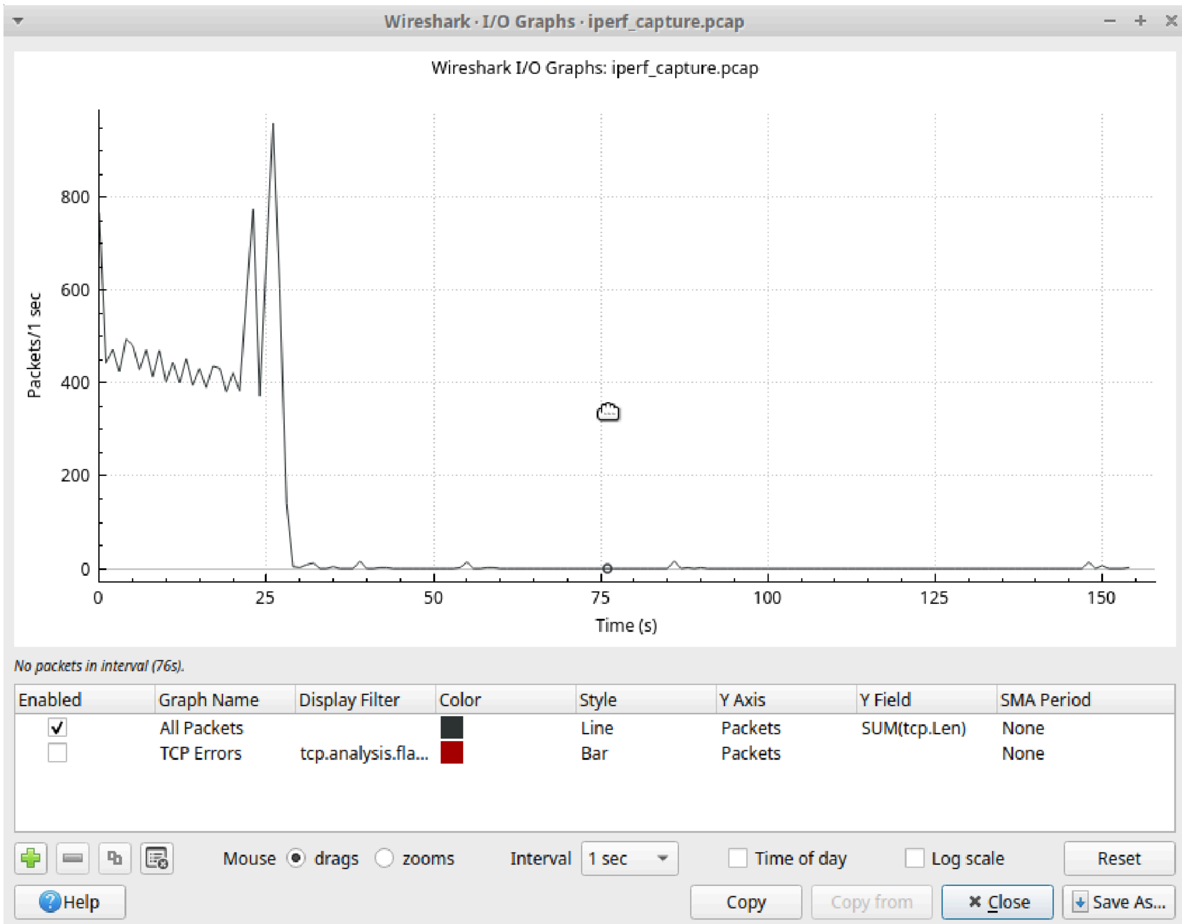
Throughput:

Goodput:

Packet Loss:

- Congestion Control = ‘bbr’

Throughput:



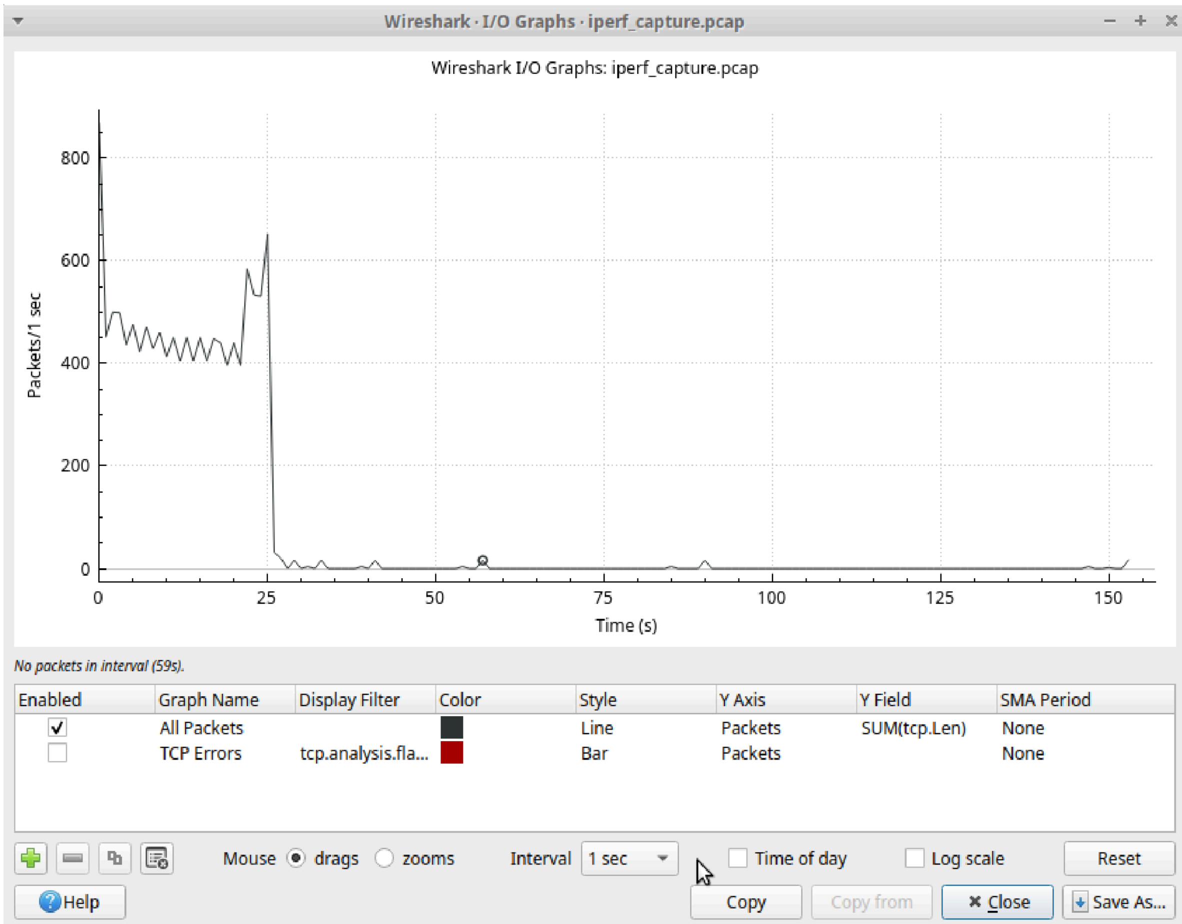
Goodput:

Packet Loss:

Maximum Packet Size:

- Congestion Control = 'westwood'

Throughput:



Goodput:

Packet Loss:

Maximum Packet Size:

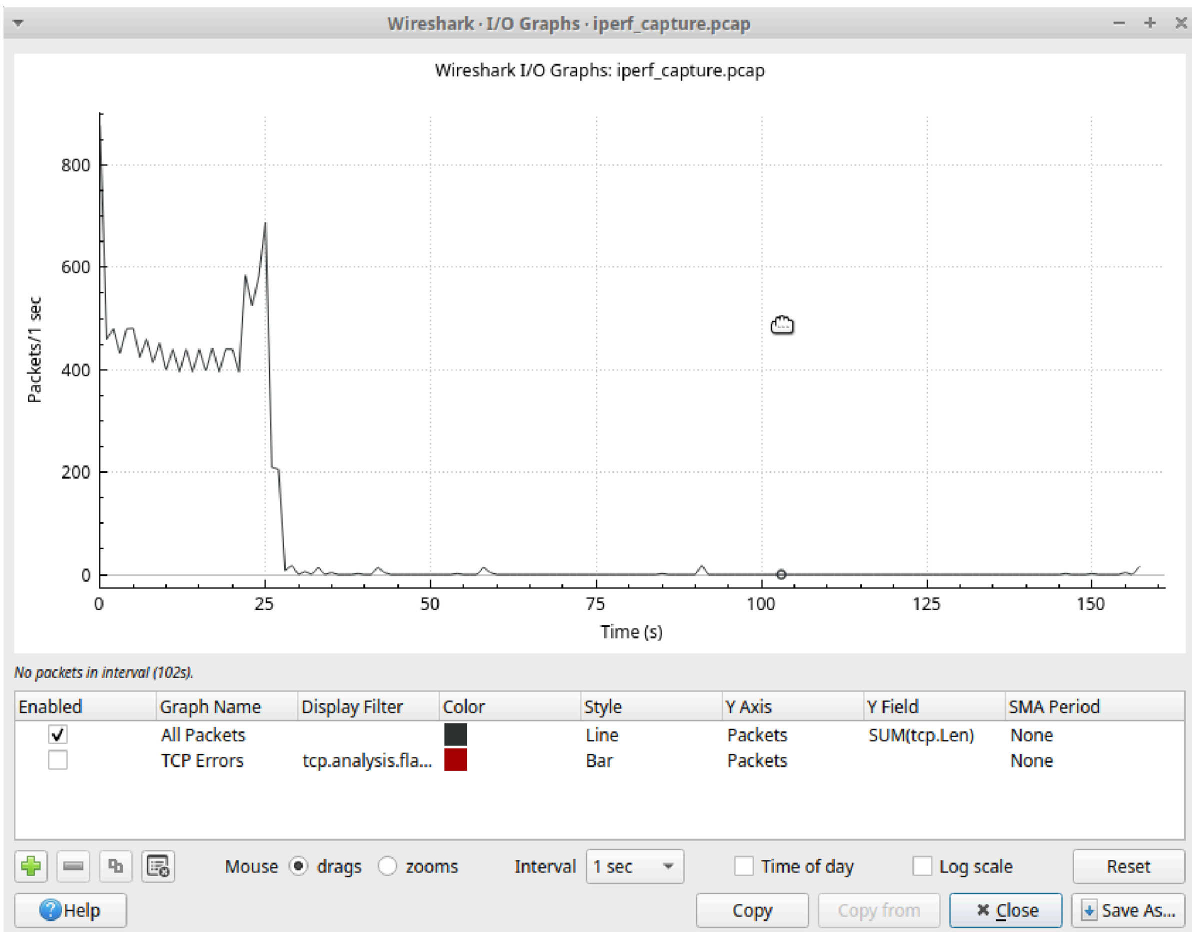
Condition 2: loss = 5%

- Congestion Control = ‘yeah’

Throughput:

0 <> 10	4958	126157082
10 <> 20	4188	124794808
20 <> 30	3658	77308764
30 <> 40	26	1716
40 <> 50	18	1188
50 <> 60	20	1320
60 <> 70	0	0

70 <> 80	0	0
80 <> 90	2	132
90 <> 100	18	1188
100 <> 110	0	0
110 <> 120	0	0
120 <> 130	0	0
130 <> 140	0	0
140 <> 150	2	132
150 <> Dur	20	1321



Goodput:

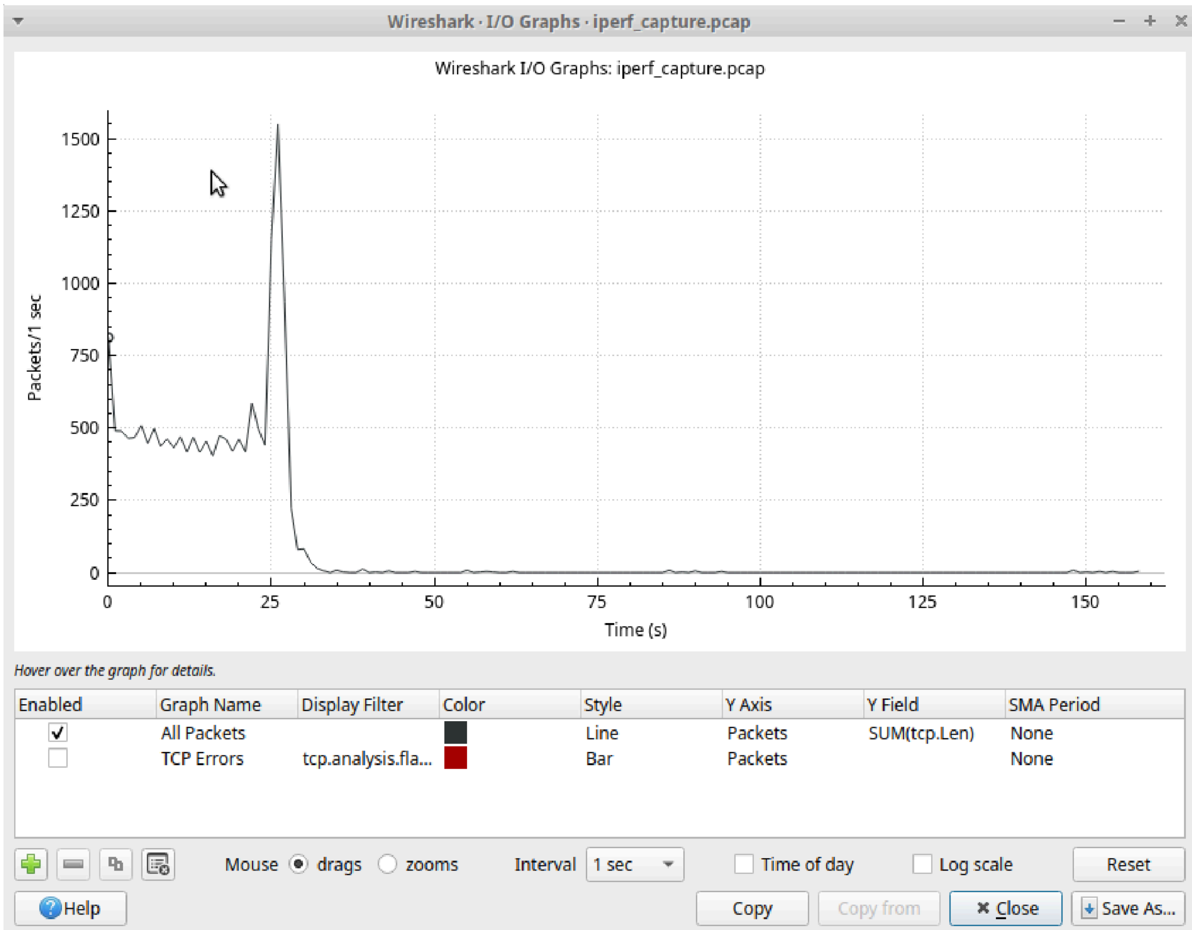
Packet Loss:

Maximum Packet Size:

- Congestion Control = ‘bbr’

Throughput:

0 <> 10	5076	126164869
10 <> 20	4415	124809790
20 <> 30	6342	84759756
30 <> 40	160	3523904
40 <> 50	12	792
50 <> 60	16	1056
60 <> 70	4	264
70 <> 80	0	0
80 <> 90	10	660
90 <> 100	10	660
100 <> 110	0	0
110 <> 120	0	0
120 <> 130	0	0
130 <> 140	0	0
140 <> 150	8	528
150 <> Dur	14	925



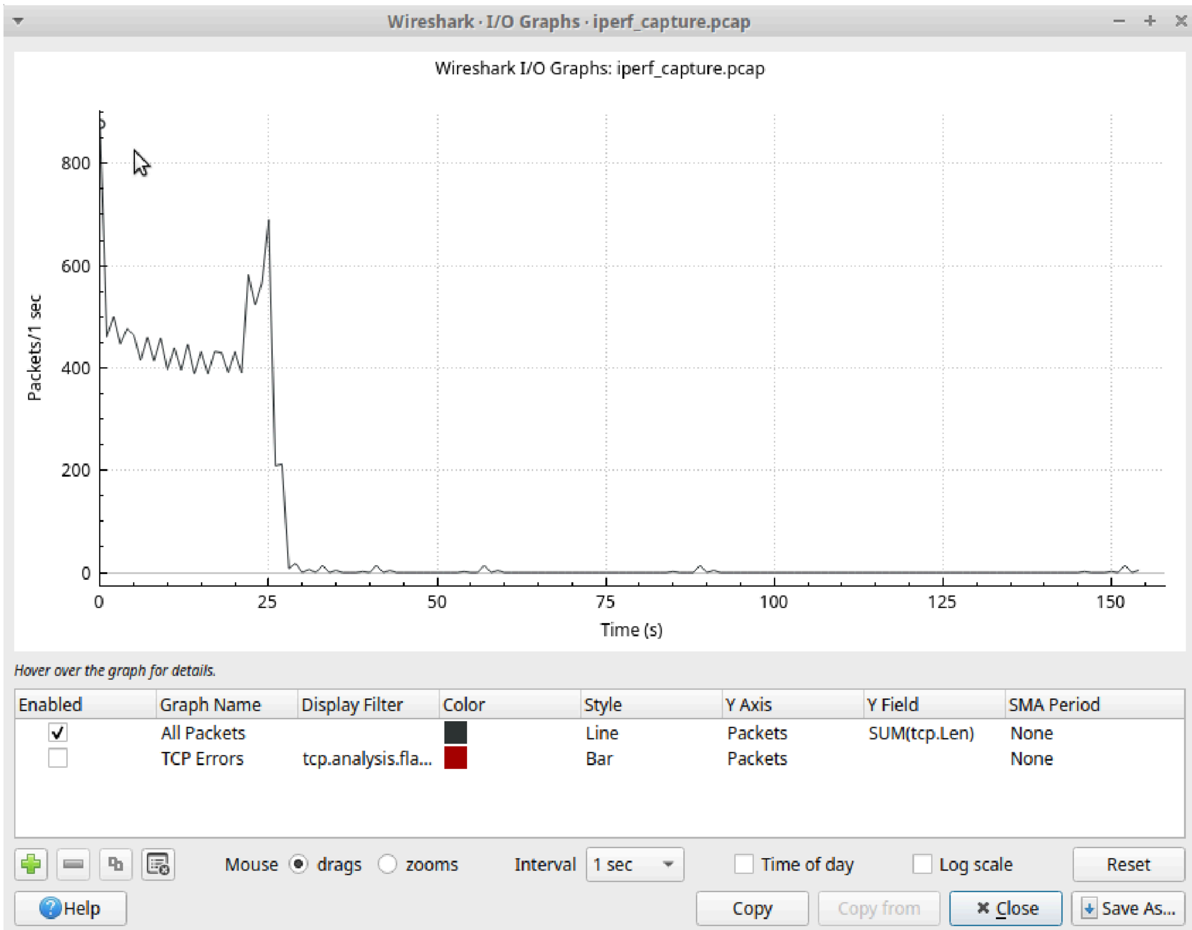
Goodput:

Packet Loss:

Maximum Packet Size:

- Congestion Control = ‘westwood’

Throughput:



Goodput:

Packet Loss:

Maximum Packet Size:

Observations

- YeAH is the best performer because it ignores loss-based congestion signals and estimates available bandwidth.
- BBR suffers more as loss increases because it switches between fast and slow modes.
- Westwood performs worst under high loss because it relies on bandwidth estimation, which gets inaccurate with lost ACKs.

Task 2: Implementation and mitigation of SYN flood attack

Setting up the environment for attack

1. Server
    a. disable cookies
    b. increase the number of retries
    c. decrease the size of backlog so that legitimate traffic is lost

```
sudo sysctl -w net.ipv4.tcp_syncookies=0
sudo sysctl -w net.ipv4.tcp_synack_retries=255
sudo sysctl -w net.ipv4.tcp_max_syn_backlog=16
```

2. Client
    a. Starting packet capture and legitimate traffic

    ```
    sudo tcpdump -i wlp0s20f3 host 172.20.10.2 -w Q2-part1.pcap > out.txt
    while true; do curl http://172.20.10.2:8000/;sleep 1;done
    ```

    b. Starting attack

    ```
    sudo iptables -A OUTPUT -p tcp --tcp-flags RST RST -j DROP
    sudo hping3 -S -p 8000 -i u10000 172.20.10.2
    ```

    - -S means SYN flag
    - -i means send 1 packet in 10000 micro sec = 100 packets per second

    c. Stopping attack

```
sudo iptables -D OUTPUT -p tcp --tcp-flags RST RST -j DROP
```

- **ctrl+c** to stop the terminal running **hping3**

For mitigation

1. Server
    a. Restoring the default settings

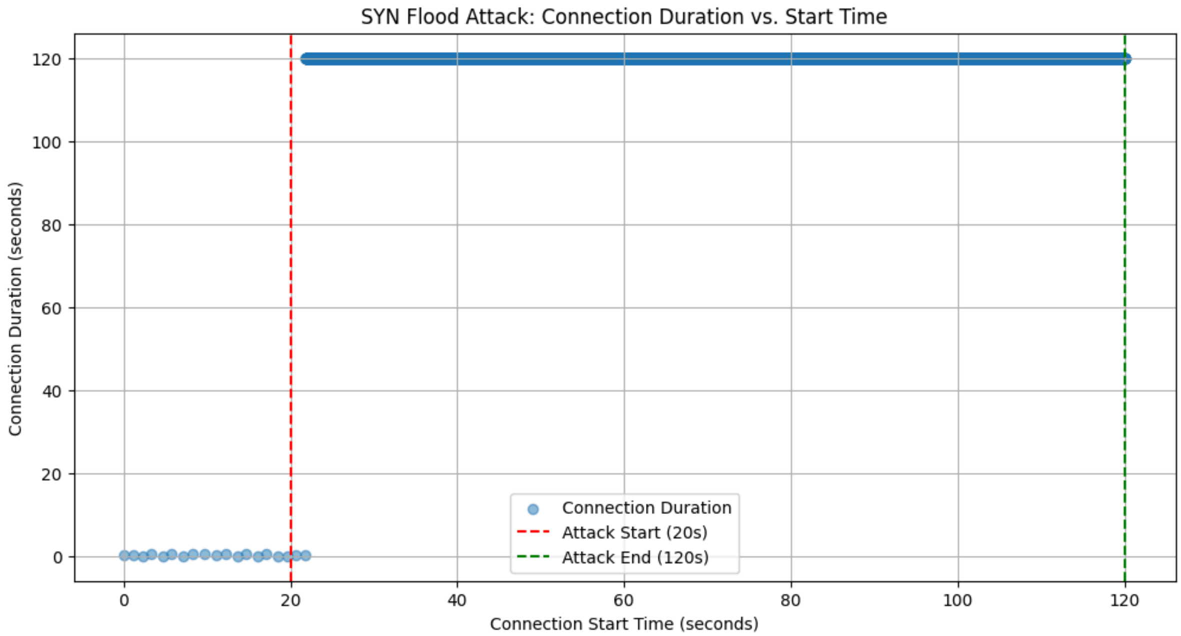
        sudo sysctl -w net.ipv4.tcp_syncookies=0
        sudo sysctl -w net.ipv4.tcp_synack_retries=255
        sudo sysctl -w net.ipv4.tcp_max_syn_backlog=16

```
sudo sysctl -w net.ipv4.tcp_syncookies=1
sudo sysctl -w net.ipv4.tcp_synack_retries=5
sudo sysctl -w net.ipv4.tcp_max_syn_backlog=128
```

2. Client remains same

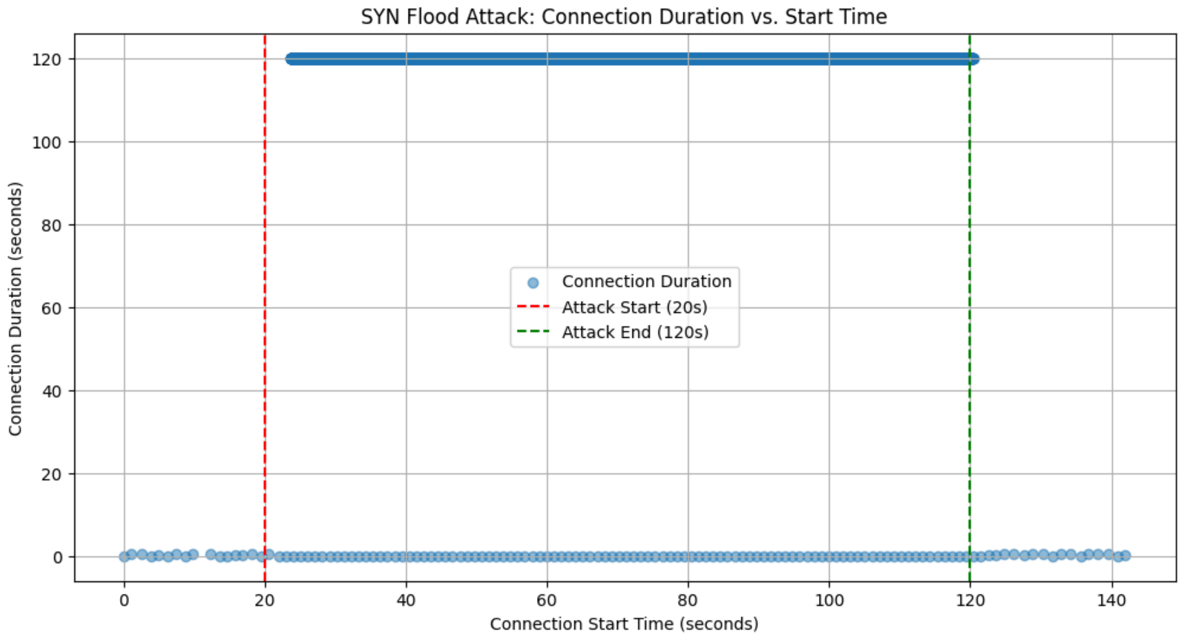
Results

1. Successful attack



- Very low connection duration in 0 - 20 seconds (legitimate traffic)
- Unclosed connections in 20 - 120 seconds (attack period)
- No connections accepted after 120 seconds because server is busy retrying the old attack packets
- Legitimate traffic stopped getting response after first 20 seconds. (DoS)

2. Mitigated attack



- The legitimate traffic is maintained throughout
- The connections opened in the attack interval are never close and stay at duration = 100 (default)
- Utilize cookies and bigger backlog buffer to store incoming requests and reduce the retries.

Task 3: Analyze the effect of Nagle's algorithm on TCP/IP performance

Setup and Tools Used

- UTM Virtual Machine Ubuntu 20.04
- Mininet
- Wireshark

Implementation

Setting Up Network

```
def setup_topology():

    class CustomTopo(Topo):
        def build(self):
            s1 = self.addSwitch('s1')
            h1 = self.addHost('h1')
            h7 = self.addHost('h7')

            self.addLink(h1, s1)
            self.addLink(s1, h7)

    net = Mininet(topo=CustomTopo(), controller=OVSController)
    net.start()

    net.get('s1').cmd('ovs-ofctl add-flow s1 actions=normal')
    return net


def configure_tcp_options(host, nagle, delayed_ack):

    if nagle:
        host.cmd("sysctl -w net.ipv4.tcp_nodelay=0")
    else:
        host.cmd("sysctl -w net.ipv4.tcp_nodelay=1")

    if delayed_ack:
        host.cmd("sysctl -w net.ipv4.tcp_delack_min=100")
    else:
        host.cmd("sysctl -w net.ipv4.tcp_delack_min=0")
```

Running Test

```
def run_tcp_test(net, nagle, delayed_ack):
```

```
    server = net.get('h7')
    client = net.get('h1')

    configure_tcp_options(server, nagle, delayed_ack)
    configure_tcp_options(client, nagle, delayed_ack)

    # Get correct IP for h7
    server_ip = server.IP()

    # Start tcpdump on the correct interface
    server.cmd('tcpdump -i h7-eth0 port 5001 -w tcp_test.pcap &')
    time.sleep(3)  # Ensure tcpdump starts before traffic

    # Start iPerf Server
    server.cmd('iperf -s -p 5001 &')
    time.sleep(2)  # Ensure server starts

    # Start iPerf Client
    client.cmd(f'iperf -c {server_ip} -p 5001 -t 120 -b 320bps')

    time.sleep(35)  # Ensure enough time for data transfer

    server.cmd('pkill iperf')
    time.sleep(2)
    server.cmd('pkill tcpdump')
    time.sleep(2)

    print("Test completed! Results saved to tcp_test.pcap.")
    analyze_pcap()
```

Analysing

```
def analyze_pcap():

    if not os.path.exists("tcp_test.pcap") or os.path.getsize("tcp_test.pcap") == 0:
```

```
        print("[ERROR] No packets were captured. Check tcpdump settings!")
        return

    throughput = subprocess.getoutput("tshark -r tcp_test.pcap -q -z io,stat,60 | g
    goodput = subprocess.getoutput("tshark -r tcp_test.pcap -Y 'tcp.len > 0' | wc
    packet_loss = subprocess.getoutput("tshark -r tcp_test.pcap -q -z expert,ip | g
    max_packet_size = subprocess.getoutput("tshark -r tcp_test.pcap -T fields -e

    print("Throughput:", throughput)
    print("Goodput (total data packets received):", goodput)
    print("Packet loss rate:", packet_loss)
    print("Maximum packet size achieved:", max_packet_size)
```

Results

Part (a) Nagle Disabled, Delayed ACK Disabled

Throughput:

Time Stamp| Packets| Total bytes transferred|

| 0.000 <> 0.032 | 8 | 4640 |

Goodput: 16.7

Packet Loss Rate: 0

Maximum Packet Size: 682

Part (b) Nagle Disabled, Delayed ACK Enabled

Throughput:

| 0.000 <> 0.012 | 8 | 4640 |

Goodput: 12.5

Packet Loss Rate: 0

Part (c) Nagle Enabled, Delayed ACK Disabled

Throughput:

| 0.000 <> 0.035 | 8 | 4640 |

Goodput: 12.5

Packet Loss Rate: 0

Part (d) Nagle Enabled, Delayed ACK Enabled

Throughput:

| 0.000 <> 0.032 | 8 | 4640 |

Goodput: 11.5

Packet Loss Rate: 0

Maximum Packet Size: 768

Explanation

1. Nagle Enabled, Delayed ACK Enabled

- **Nagle buffers packets** instead of sending immediately.
- **Delayed ACKs wait** before acknowledging, causing additional delays.
- **Result**: Very inefficient transfer, **low throughput and goodput**.

2. Nagle Enabled, Delayed ACK Disabled

- **Nagle still buffers**, but ACKs come faster since DA is off.
- **Result**: Slightly better performance but still **moderate delays**.

3. Nagle Disabled, Delayed ACK Enabled

- **Packets are sent immediately**, but ACKs are delayed.
- **Result**: More packets in flight, but **some unnecessary retransmissions** due to delayed ACKs.

4. Nagle Disabled, Delayed ACK Disabled

- **Every packet is sent immediately** (no waiting for Nagle).
- **ACKs are sent immediately** (no waiting for DA).

- **Result**: Best efficiency, **highest throughput and goodput**.

Observations

- IMP: Because TCP is a dynamic protocol, even when we limit the bandiwdth of TCP connection, the backend utilises the maximum possible bandwidth.
- We observe the outcomes of the execution similar to explanation above.
- Part (a) achieves highest goodput becuase of lower delays and there is no need for conservative congestinon control.

Acknowledgement

We would like to thank Prof. Sameer Kulkarni to provide this oppourtunity to work on this assignment. We also thank teaching assistants for their constant support.