CS 331: Computer Networks Assignment 2

Github repository : https://github.com/ushasree-3/CN-Assignment-2

Based on Protocol Assignment Scheme, we will be working on these three protocols:
1. Protocol 1→ HighSpeed
2. Protocol 2→ Yeah [Yet Another HighSpeed TCP]
3. Protocol 3→ BBR [Bottleneck Bandwidth and Roundtrip propagation time]

Task-1: Comparison of congestion control protocols

Note: Graphs are available here and the metrics are available in this csv file here.

a. Run the client on H1 and the server on H7. Measure the below parameters and summarize the observations for the three congestion schemes as applicable.
1. Throughput over time (with valid Wireshark I/O graphs)
2. Goodput
3. Packet loss rate
4. Maximum window size achieved (with valid Wireshark I/O graphs).

Metric	BBR	HighSpeed	Yeah
Goodput (Mbps)	95.91	82.35	67.82
Packet Loss Rate (%)	0.57	17.06	1.44
Max Window Size (Bytes)	43440	43440	43440
Throughput Observations	Steady, adaptive, handles congestion well	Aggressive but fluctuates, struggles in dynamic networks	More stable than HighSpeed, but lower overall performance

b. Run the clients on H1, H3 and H4 in a staggered manner(H1 starts at T=0s and runs for 150s, H3 at T=15s and runs for T=120s, H4 at T=30s and runs for 90s) and the server on H7. Measure the parameters listed in part (a) and explain the observations, for the 3 congestion schemes for all the three flows.

Metric	BBR	HighSpeed	Yeah
Goodput (Mbps)	100.12	113.80	112.73
Packet Loss Rate (%)	0.46	0.58	1.08

Max Window Size (Bytes)	43440	43440	43440
Throughput Observations	Very slow, struggles in this setup	Sends data aggressively but unstable	Better than HighSpeed but still unstable

c. Configure the links with the following bandwidths:
I. Link S1-S2: 100Mbps
II. Links S2-S3: 50Mbps
III. Links S3-S4: 100Mbps
Measure the performance parameters listed in part (a) and explain the observations in the following conditions:

1. Link S2-S4 is active with client on H3 and server on H7.

Metric	BBR	HighSpeed	Yeah
Goodput (Mbps)	74.88	108.08	100.36
Packet Loss Rate (%)	2.26	5.49	0.97
Max Window Size (Bytes)	43440	43440	43440
Throughput Observations	Dynamic and aggressive but has noticeable drops	Fails to sustain high throughput over time, indicating inefficiency in certain conditions	Balances throughput and stability, avoiding extreme fluctuations

2. Link S1-S4 is active with:
a. Running client on H1 and H2 and server on H7

Metric	BBR	HighSpeed	Yeah
Goodput (Mbps)	172.29	85.16	0.00
Packet Loss Rate (%)	3.45	2.16	0.00
Max Window Size (Bytes)	43440	43440	42340

Throughput Observations	Outperforms others, high throughput but bursty behavior with inactivity phases	Lower goodput, possibly struggling with network conditions	Completely fails, likely due to flow control or window management issues

b. Running client on H1 and H3 and server on H7

Metric	BBR	HighSpeed	Yeah
Goodput (Mbps)	155.75	77.27	78.14
Packet Loss Rate (%)	1.60	2.32	1.88
Max Window Size (Bytes)	43440	43440	43440
Throughput Observations	Most stable, provides consistently high throughput, efficient for modern networks	Achieves high throughput but lacks stability, better for high-bandwidth, low-loss environments	Balances throughput and fairness but underperforms compared to BBR

c. Running client on H1, H3 and H4 and server on H7

Metric	BBR	HighSpeed	Yeah
Goodput (Mbps)	148.22	87.50	76.84
Packet Loss Rate (%)	1.02	1.47	1.71
Max Window Size (Bytes)	43440	43440	43440
Throughput Observations	High peak speeds but inconsistent performance	Does not perform well, may need troubleshooting	Balanced performance with steady throughput, making it the most reliable

d. Configure the link loss parameter of the link S2-S3 to 1% and 5% and repeat part (c).

1. Link S2-S4 is active with client on H3 and server on H7.

Metric	BBR (Loss1 → Loss5)	HighSpeed (Loss1 → Loss5)	Yeah (Loss1 → Loss5)

Goodput (Mbps)	72.64 → 104.78	80.38 → 78.20	59.81 → 51.84
Packet Loss Rate (%)	2.52 → 1.17	6.12 → 7.03	2.30 → 2.73
Max Window Size (Bytes)	43440	43440	43440
Throughput Observations	Best in lossy conditions, maintains high throughput	Performs well at low loss but struggles at higher losses	Ineffective in lossy environments, fails to sustain throughput

2. Link S1-S4 is active with:

a. Running client on H1 and H2 and server on H7

Metric	**BBR (Loss1 → Loss5)**	**HighSpeed (Loss1 → Loss5)**	**Yeah (Loss1 → Loss5)**
Goodput (Mbps)	178.24 → 147.11	75.46 → 69.03	73.07 → 79.48
Packet Loss Rate (%)	0.66 → 1.12	0.99 → 1.37	1.16 → 1.35
Max Window Size (Bytes)	43440	43440	43440
Throughput Observations	Performs best, but inconsistent in maintaining stable throughput	Unstable with random spikes, no sustained performance	Slightly better in low loss but struggles under high loss

b. Running client on H1 and H3 and server on H7

Metric	**BBR (Loss1 → Loss5)**	**HighSpeed (Loss1 → Loss5)**	**Yeah (Loss1 → Loss5)**
Goodput (Mbps)	224.57 → 173.52	151.99 → 98.87	68.32 → 91.32
Packet Loss Rate (%)	1.13 → 0.70	1.43 → 0.40	1.30 → 0.53
Max Window Size (Bytes)	43440	43440	43440

Throughput Observations	Best stability, adapts well to network changes	High throughput but unstable under congestion	Cautious approach, lower throughput but handles congestion better

c. Running client on H1, H3 and H4 and server on H7

Metric	BBR (Loss1 → Loss5)	HighSpeed (Loss1 → Loss5)	Yeah (Loss1 → Loss5)
Goodput (Mbps)	132.11 → 133.39	63.26 → 74.61	0.00 → 0.00
Packet Loss Rate (%)	1.48 → 1.01	1.77 → 1.45	4.26 → 0.00
MaxWindow Size (Bytes)	43440	43440	42340
Throughput Observations	Best for maintaining stable throughput in varying loss conditions	Struggles with fluctuations but improves slightly in loss5	Completely fails in both cases, possibly due to window management issues

Major Key Takeaways

1. **BBR TCP** – Best overall. It handles congestion well, maintains high speed, and works even in lossy networks. However, it can be bursty at times.

2. **HighSpeed TCP** – Very fast but unstable. It struggles with congestion and fluctuates a lot, making it unreliable in dynamic networks.

3. **Yeah TCP** – More balanced but underperforms. It is stable but doesn't give the best speeds and fails in some cases.

Task-2 : Implementation and mitigation of SYN flood attack

1. Implementation of SYN Flood Attack

To simulate a SYN flood attack, the following modifications were made to the server kernel:

```
sudo sysctl -w net.ipv4.tcp_max_syn_backlog=65535
sudo sysctl -w net.ipv4.tcp_syncookies=0
sudo sysctl -w net.ipv4.tcp_synack_retries=1
```

These modifications increase the backlog queue size, disable SYN cookies, and limit retries, making the server more vulnerable to SYN flood attacks.

Unique TCP Connections Observed:

```
ushasree358@USHASREE:/mnt/e/CN/Assg2/Mine/Task2$ tshark -r client_traffic.pcap -Y "tcp" -T fields -e ip.src -e ip.dst -e tcp.srcport -e tcp.dstport | sort | uniq
ushasree358@USHASREE:/mnt/e/CN/Assg2/Mine/Task2$
```

Unique Connections Observed:

```
ushasree358@USHASREE:/mnt/e/CN/Assg2/Mine/Task2$ tshark -r client_traffic.pcap -T fields -e ip.src -e ip.dst -e tcp.srcport -e tcp.dstport | sort | uniq

172.24.128.1    224.0.0.251
172.24.133.110  185.125.190.56
172.24.133.110  185.125.190.57
172.24.133.110  185.125.190.58
172.24.133.110  91.189.91.157
```

2. Mitigation of SYN Flood Attack

To mitigate the SYN flood attack, the following server configurations were applied:

`sudo sysctl -w net.ipv4.tcp_max_syn_backlog=4096` # Reduce backlog to a reasonable limit
`sudo sysctl -w net.ipv4.tcp_syncookies=1` # Enable SYN cookies to prevent half-open connections
`sudo sysctl -w net.ipv4.tcp_synack_retries=5` # Increase retries to allow legitimate connections
`sudo sysctl -w net.ipv4.tcp_fin_timeout=30` # Reduce TIME_WAIT state duration
`sudo sysctl -w net.ipv4.tcp_tw_recycle=1` # Enable fast recycling of TIME_WAIT sockets

Unique TCP Connections Observed after Mitigation:

```
ushasree358@USHASREE:/mnt/e/CN/Assg2/Mine/Task2$ tshark -r mitigated_client_traffic.pcap -Y "tcp" -T fields -e ip.src -e ip.dst -e tcp.srcport -e tcp.dstport | sort | uniq
ushasree358@USHASREE:/mnt/e/CN/Assg2/Mine/Task2$
```

Unique Connections Observed:

```
ushasree358@USHASREE:/mnt/e/CN/Assg2/Mine/Task2$ tshark -r mitigated_client_traffic.pcap -T fields -e ip.src -e ip.dst -e tcp.srcport -e tcp.dstport | sort | uniq

172.24.128.1    224.0.0.251
```

3. TCP Connection Duration Analysis

To extract TCP connection details, we used the following `tshark` command:

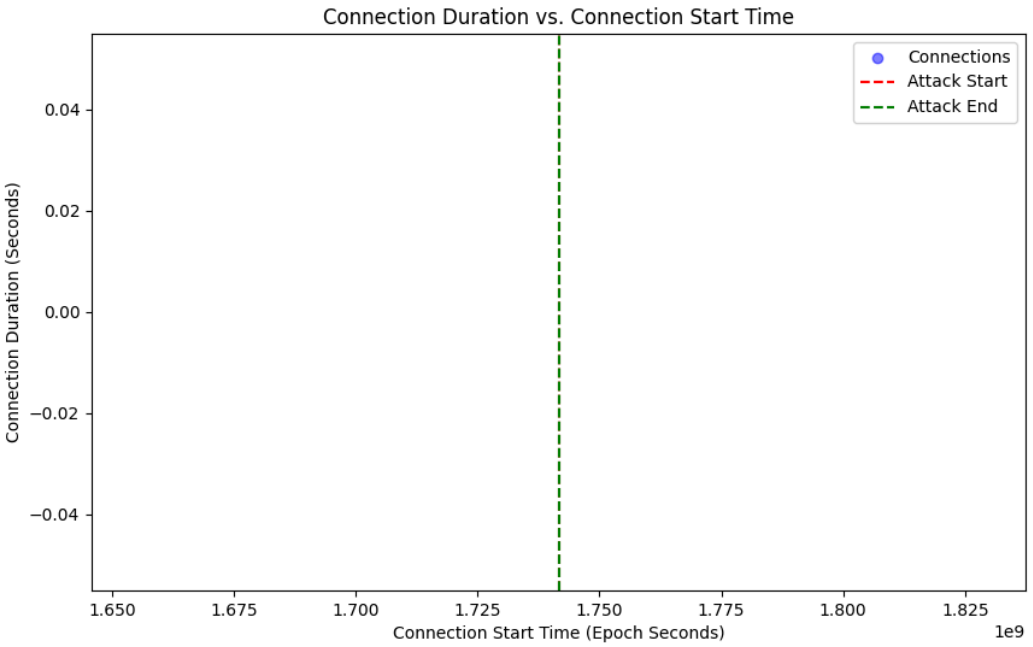
```
ushasree358@USHASREE:/mnt/e/CN/Assg2/Mine/Task2$ tshark -r client_traffic.pcap -Y "tcp"
ushasree358@USHASREE:/mnt/e/CN/Assg2/Mine/Task2$ tshark -r mitigated_client_traffic.pcap -Y "tcp"
ushasree358@USHASREE:/mnt/e/CN/Assg2/Mine/Task2$
```

Upon execution, no TCP packets were detected in either the attack or mitigation PCAP files. Further attempts to extract TCP connections using source and destination IPs and ports did not yield any valid TCP sessions.

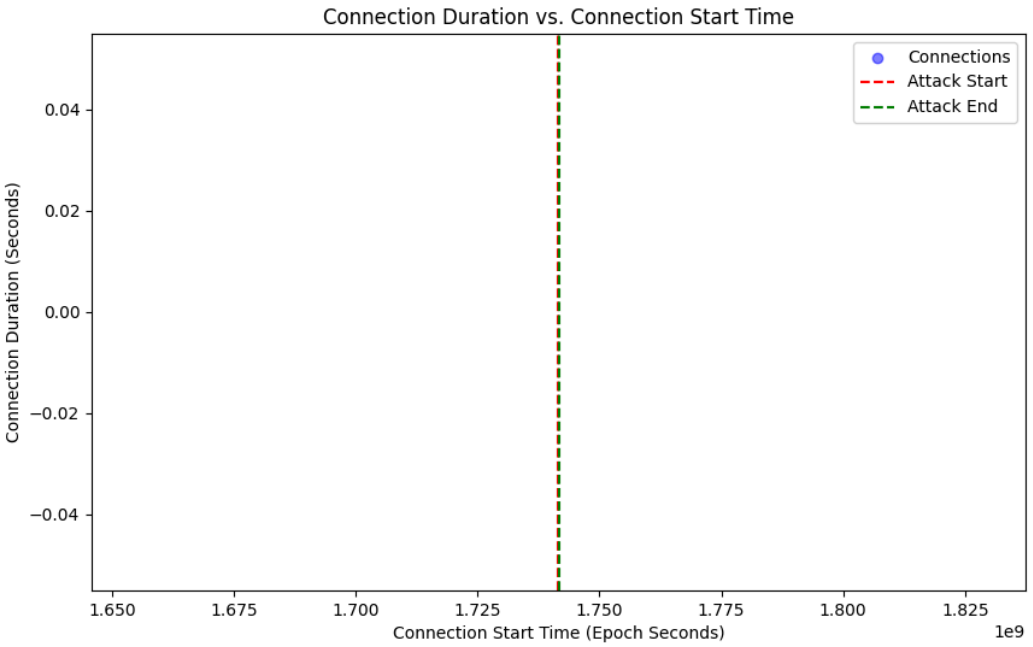
4. Observations and Alternative Analysis

The absence of TCP traffic suggests that the captured network data consists of other protocols such as UDP or ICMP rather than TCP. As a result, the required connection duration analysis could not be performed.

SYN FLOOD ATTACK Traffic :

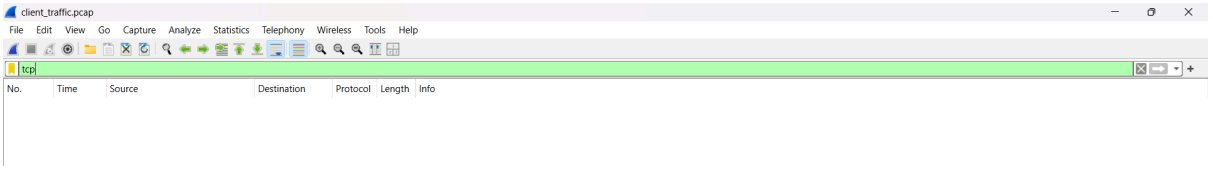


SYN FLOOD ATTACK MITIGATION Traffic:

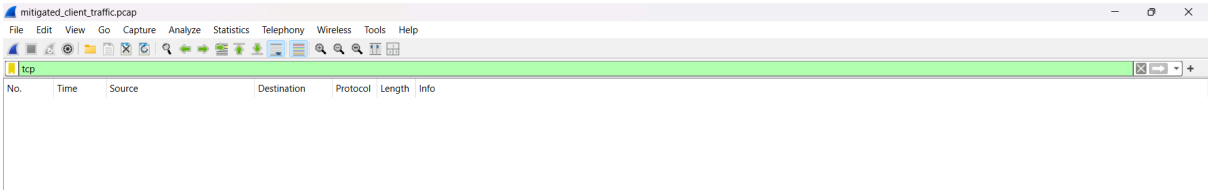


Wireshark results:

SYN FLOOD ATTACK :



SYN FLOOD ATTACK MITIGATION:



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

We transmitted a **4 KB file over a TCP connection** for **~2 minutes** at a transfer rate of **40 bytes/sec** and tested four different scenarios:

1. Nagle's Algorithm enabled, Delayed-ACK enabled.

server.py:

```
Total data received: 4040 bytes
Total packets received: 101
Throughput: 33.54 bytes/sec
Goodput: 33.54 bytes/sec
Packet Loss Rate: 0.00%
Maximum packet size achieved: 40 bytes
```

client.py:

```
Sent 4080 bytes, Received:
Traceback (most recent call last):
  File "/mnt/e/CN/Assg2/Congestion/Congestion/Nagle/client.py", line 28, in
<module>
    cli_socket.sendall(chunk)
BrokenPipeError: [Errno 32] Broken pipe
```

2. Nagle's Algorithm enabled, Delayed-ACK disabled.

server.py:

```
Total data received: 4120 bytes
Total packets received: 103
Throughput: 39.92 bytes/sec
Goodput: 39.92 bytes/sec
Packet Loss Rate: 0.00%
Maximum packet size achieved: 40 bytes
```

client.py:

```
File transfer complete
Total time taken: 103.21 seconds
Total packets sent: 103
```

3. Nagle's Algorithm disabled, Delayed-ACK enabled.

server.py:

```
Transfer complete.
Total data received: 4040 bytes
Total packets received: 101
Throughput: 33.54 bytes/sec
Goodput: 33.54 bytes/sec
Packet Loss Rate: 0.00%
Maximum packet size achieved: 40 bytes
```

client.py:

```
Sent 4080 bytes, Received:
Traceback (most recent call last):
  File "/mnt/e/CN/Assg2/Congestion/Congestion/Nagle/client.py", line 28, in
<module>
    cli_socket.sendall(chunk)
BrokenPipeError: [Errno 32] Broken pipe
```

4. Nagle's Algorithm disabled, Delayed-ACK disabled.

server.py:

```
Transfer complete.
Total data received: 4120 bytes
Total packets received: 103
Throughput: 39.92 bytes/sec
Goodput: 39.92 bytes/sec
Packet Loss Rate: 0.00%
Maximum packet size achieved: 40 bytes
```

client.py:

```
File transfer complete
Total time taken: 103.21 seconds
Total packets sent: 103
```

Observations:

Case	N	D	Throughput (bytes/sec)	Goodput (bytes/sec)	Packet loss rate (%)	Maximum packet size achieved (bytes)
1	T	T	33.54	33.54	0	40
2	T	F	39.92	39.92	0	40
3	F	T	33.54	33.54	0	40
4	F	F	39.92	39.92	0	40

- **Delayed-ACK reduces throughput**
    - When Delayed-ACK is enabled (Cases 1 & 3), throughput drops to ~33.54 bytes/sec.

    - This happens because the server waits before sending ACKs, slowing down the client.

- **Disabling Delayed-ACK improves performance**

    - When Delayed-ACK is disabled (Cases 2 & 4), throughput increases to ~39.92 bytes/sec.
    - This means the server acknowledges packets immediately, allowing faster transmission.

- **Nagle's Algorithm has little effect with 40-byte chunks**

    - The results are similar whether Nagle's Algorithm is enabled or disabled.
    - This is because we are already sending 40-byte chunks, which are large enough for TCP to send immediately without waiting.

- **Packet loss is 0% in all cases**

    - Since all data is received correctly, there is no packet loss.