Homework 3 Part 3

Question 1

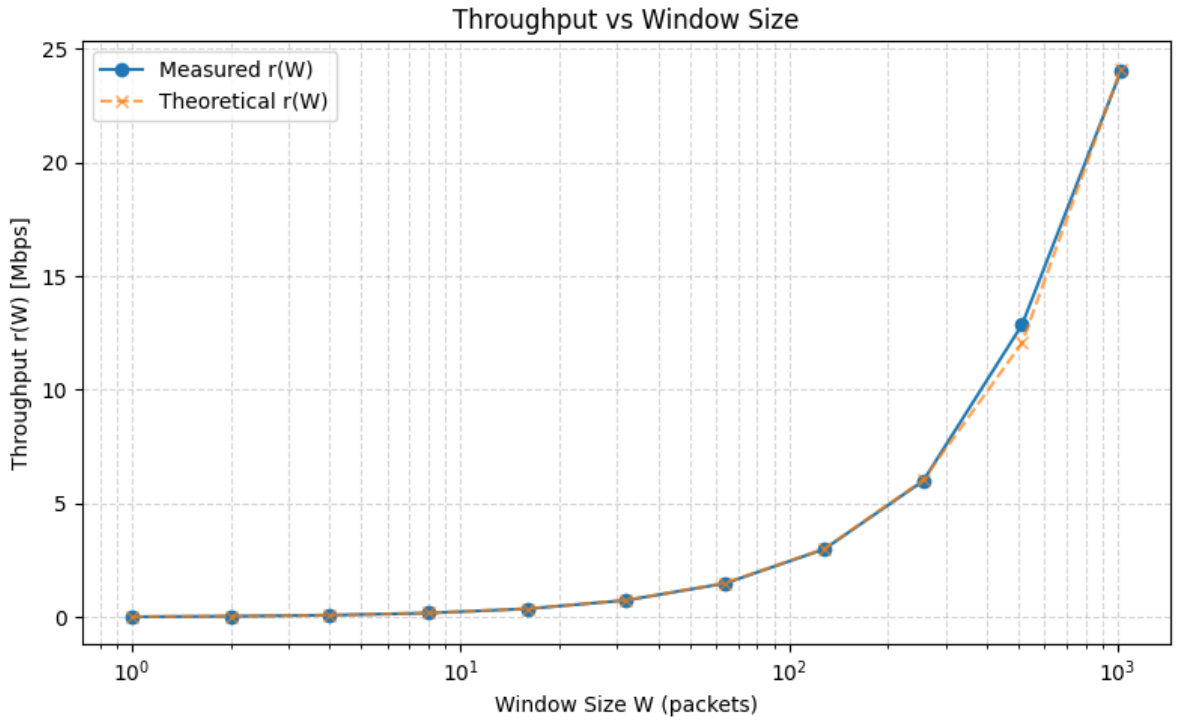
We measured the steady-state throughput r(W) for a range of window sizes W=1,2,4,...,1024, with RTT fixed at 0.5 seconds, no packet loss, and link speed set to 1 Gbps. The results show:

- Throughput grows linearly with window size
- The observed rate matches the model:

$$r(W) = \frac{W \cdot MSS \cdot 8}{RTT} = 0.0235 \cdot W\ Mbps$$

- At W=1024, measured throughput was ~24 Mbps, matching expected behaviour.

Python Code in Appendix



Question 2

We measured the steady-state throughput r(RTT)r(RTT)r(RTT) for a fixed window size W=30W = 30W=30 packets, while varying the RTT from 10 ms to 5120 ms. The actual measured RTT values were used to account for emulation and OS-induced deviations.

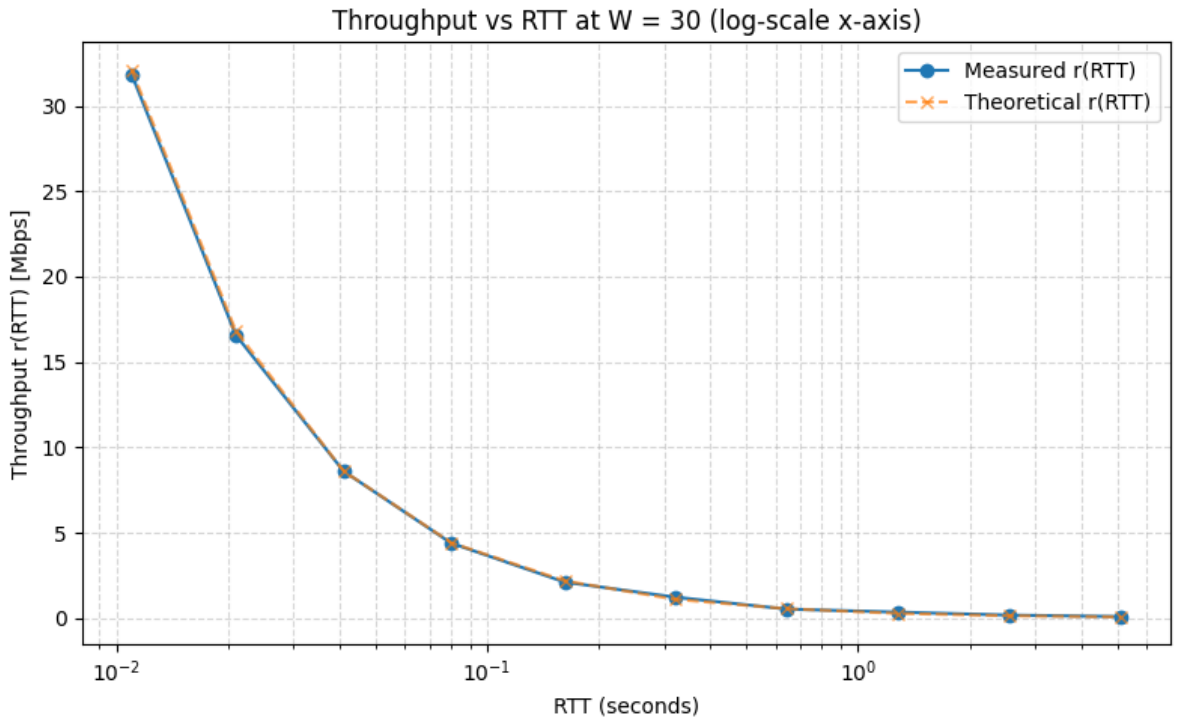
The results show:

- **Throughput decreases inversely with RTT**, consistent with the theoretical model
- The observed rate closely matches the model:

$$r(RTT) = \frac{W \cdot MSS \cdot 8}{RTT} = \frac{30 \cdot 1472 \cdot 8}{RTT} = \frac{353280}{RTT}\ bps$$

- These findings validate the relationship r(RTT) ∝ 1/RTT, though minor deviations exist due to queuing and system overheads in the test environment.

Code in Appendix



Question 3

The experiment measured throughput on a local dummy receiver with window size W=8000W = 8000W=8000. With standard 1472-byte packets, the peak throughput reached approximately **845 Mbps**, remaining below the 1 Gbps threshold. In contrast, using 9KB jumbo packets resulted in a peak throughput of approximately **4.7 Gbps**. This difference illustrates how larger packet sizes reduce per-packet processing overhead and significantly improve achievable throughput, especially on high-speed links. The test system used an Intel Core i5-9400 CPU (6 cores @ 2.90 GHz) with 16 GB RAM.

Question 4

```
Main:   sender W = 300, RTT = 0.200 sec, loss 0 / 0.1, link 10 Mbps
Main:   initializing DWORD array with 2^23 elements... done in 64 ms
Main:   connected to s3.irl.cs.tamu.edu in 0.200 sec, packet size 1472 bytes
[   2] B             256 (    0.4 MB) N      512 T    0 F    0 W    256 S   1.499 Mbps RTT 0.273
[   4] B            1937 (    2.8 MB) N     2237 T    0 F    0 W    300 S   9.839 Mbps RTT 0.353
[   6] B            3636 (    5.3 MB) N     3936 T    0 F    0 W    300 S   9.944 Mbps RTT 0.353
[   8] B            5335 (    7.8 MB) N     5635 T    0 F    0 W    300 S   9.939 Mbps RTT 0.353
[  10] B            7034 (   10.3 MB) N     7334 T    0 F    0 W    300 S   9.944 Mbps RTT 0.354
[  12] B            8733 (   12.8 MB) N     9033 T    0 F    0 W    300 S   9.944 Mbps RTT 0.353
[  14] B           10433 (   15.3 MB) N    10733 T    0 F    0 W    300 S   9.950 Mbps RTT 0.353
[  16] B           12132 (   17.8 MB) N    12432 T    0 F    0 W    300 S   9.944 Mbps RTT 0.353
[  18] B           13832 (   20.2 MB) N    14132 T    0 F    0 W    300 S   9.950 Mbps RTT 0.353
[  20] B           15532 (   22.7 MB) N    15832 T    0 F    0 W    300 S   9.950 Mbps RTT 0.353
[  22] B           17232 (   25.2 MB) N    17532 T    0 F    0 W    300 S   9.950 Mbps RTT 0.353
[  24] B           18933 (   27.7 MB) N    19233 T    0 F    0 W    300 S   9.956 Mbps RTT 0.353
[  26] B           20634 (   30.2 MB) N    20934 T    0 F    0 W    300 S   9.951 Mbps RTT 0.353
[  28] B           22334 (   32.7 MB) N    22634 T    0 F    0 W    300 S   9.950 Mbps RTT 0.353
[  30] B           22920 (   33.6 MB) N    22920 T    0 F    0 W    300 S   3.430 Mbps RTT 0.353
[ 57.71] <-- FIN-ACK 22920 window D70096AB
Main:   transfer finished in 28.582 sec, 9391.77 Kbps, checksum D70096AB
Main:   estRTT 0.353, ideal rate 10004.26 Kbps
```

```
Main:   sender W = 300, RTT = 0.200 sec, loss 0 / 0, link 10 Mbps
Main:   initializing DWORD array with 2^23 elements... done in 64 ms
Main:   connected to s3.irl.cs.tamu.edu in 0.200 sec, packet size 1472 bytes
[   2] B             256 (    0.4 MB) N      512 T    0 F    0 W    256 S   1.498 Mbps RTT 0.275
[   4] B            1939 (    2.8 MB) N     2239 T    0 F    0 W    300 S   9.851 Mbps RTT 0.354
[   6] B            3638 (    5.3 MB) N     3938 T    0 F    0 W    300 S   9.944 Mbps RTT 0.353
[   8] B            5336 (    7.8 MB) N     5636 T    0 F    0 W    300 S   9.939 Mbps RTT 0.353
[  10] B            7036 (   10.3 MB) N     7336 T    0 F    0 W    300 S   9.950 Mbps RTT 0.354
[  12] B            8735 (   12.8 MB) N     9035 T    0 F    0 W    300 S   9.939 Mbps RTT 0.353
[  14] B           10434 (   15.3 MB) N    10734 T    0 F    0 W    300 S   9.944 Mbps RTT 0.353
[  16] B           12133 (   17.8 MB) N    12433 T    0 F    0 W    300 S   9.944 Mbps RTT 0.353
[  18] B           13833 (   20.3 MB) N    14133 T    0 F    0 W    300 S   9.950 Mbps RTT 0.353
[  20] B           15534 (   22.7 MB) N    15834 T    0 F    0 W    300 S   9.956 Mbps RTT 0.353
[  22] B           17234 (   25.2 MB) N    17534 T    0 F    0 W    300 S   9.950 Mbps RTT 0.353
[  24] B           18934 (   27.7 MB) N    19234 T    0 F    0 W    300 S   9.950 Mbps RTT 0.353
[  26] B           20635 (   30.2 MB) N    20935 T    0 F    0 W    300 S   9.956 Mbps RTT 0.353
[  28] B           22335 (   32.7 MB) N    22635 T    0 F    0 W    300 S   9.950 Mbps RTT 0.353
[  30] B           22920 (   33.6 MB) N    22920 T    0 F    0 W    300 S   3.422 Mbps RTT 0.353
[ 48.76] <-- FIN-ACK 22920 window D70096AB
Main:   transfer finished in 28.581 sec, 9392.09 Kbps, checksum D70096AB
Main:   estRTT 0.353, ideal rate 10000.67 Kbps
```

In both the loss-free and the reverse-path loss (10%) scenarios, the protocol achieves nearly the same throughput — around 9390 Kbps — because the sender can tolerate some ACK loss without stalling. When an ACK is lost, it doesn't always require a timeout or retransmission; a later ACK (for a higher sequence number) can still acknowledge earlier packets and advance the window. This cumulative ACK behavior allows the sender to continue transmitting without waiting for every single ACK to arrive. Additionally, with a large window size of 300 packets, the sender has enough in-flight packets to keep the pipeline full, maintaining high throughput even when a few ACKs are dropped. Since the RTT estimate and retransmission timeout (RTO) adapt over time, occasional ACK loss has little impact on overall performance.

Question 5

The receiver uses **TCP flow control** to advertise its current receive window in each ACK. This window is based on the remaining space in the receiver's buffer and limits how much data the sender can transmit without acknowledgment. This technique ensures the sender does not overwhelm the receiver. The advertised window has a fixed **upper bound** determined by the size of the receiver's buffer. Without window scaling, this upper bound is 65,535 bytes (64 KB). With TCP Window Scaling, this can be extended to 1 GB (1,073,741,824 bytes).

65,535 bytes (64 KB) — Default Maximum Without Scaling

- **TCP headers have a 16-bit field for the Window Size in the TCP header.**
- **A 16-bit field can hold values from 0 to 216−1=65,5352^{16} - 1 = 65,535216−1=65,535.**
- **So, without window scaling, the max advertised window is 65,535 bytes (just under 64 KB).**

This is defined in RFC 793 (TCP specification).

1,073,741,824 bytes (1 GB) — With Window Scaling

- **In RFC 1323, TCP introduced Window Scaling to support high-bandwidth networks.**
- **Window scaling adds a scale factor (shift count) from 0 to 14, applied to the 16-bit window.**
- **This is negotiated during the TCP 3-way handshake using the Window Scale option.**

Appendix

Question 1

```
import matplotlib.pyplot as plt
import numpy as np

# Data
W = np.array([1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024])
r = np.array([0.023, 0.047, 0.093, 0.187, 0.374, 0.748, 1.493, 2.992,
5.991, 12.859, 24.020])

# Theoretical throughput (MSS = 1472 bytes, RTT = 0.5s)
MSS = 1472
RTT = 0.5
theoretical = (W * MSS * 8) / RTT / 1e6

# Plot
plt.figure(figsize=(8, 5))
plt.semilogx(W, r, 'o-', label='Measured r(W)')
plt.semilogx(W, theoretical, 'x--', label='Theoretical r(W)', alpha=0.7)
plt.xlabel('Window Size W (packets)')
plt.ylabel('Throughput r(W) [Mbps]')
plt.title('Throughput vs Window Size')
plt.grid(True, which='both', linestyle='--', alpha=0.5)
plt.legend()
plt.tight_layout()
plt.savefig('throughput_plot.png')  # <- Save the plot
plt.show()
```

Question 2

```
import matplotlib.pyplot as plt
```

```
import numpy as np


# Measured values

RTT = np.array([0.011, 0.021, 0.041, 0.080, 0.162, 0.321, 0.642, 1.281,
2.561, 5.122])  # in seconds

r_measured = np.array([31.841, 16.582, 8.604, 4.390, 2.106, 1.229, 0.527,
0.351, 0.176, 0.091])  # in Mbps


# Theoretical model: r(RTT) = (W * MSS * 8) / RTT

W = 30

MSS = 1472  # bytes

r_theory = (W * MSS * 8) / RTT / 1e6  # Mbps


# Plot

plt.figure(figsize=(8, 5))

plt.semilogx(RTT, r_measured, 'o-', label='Measured r(RTT)')

plt.semilogx(RTT, r_theory, 'x--', label='Theoretical r(RTT)', alpha=0.7)

plt.xlabel('RTT (seconds)')

plt.ylabel('Throughput r(RTT) [Mbps]')

plt.title('Throughput vs RTT at W = 30 (log-scale x-axis)')

plt.grid(True, which='both', linestyle='--', alpha=0.5)

plt.legend()

plt.tight_layout()

plt.show()
```