

EE4204 Lab Report

Name: Bryan Castorius Halim

Student ID: A0255783R

Overview

I developed a UDP-based client-server socket program written in C for transferring a large message. The message used for this experiment is “myfile.txt,” containing text with a size of 59792 MB (Megabytes).

There are 4 main file components of this project:

- myfile.txt -> message to be sent in packets.
- headsock.h -> header file containing libraries and structs declarations.
- udp_server.c -> Server receiving the message from client
- udp_client.c -> Client sending the message to Server

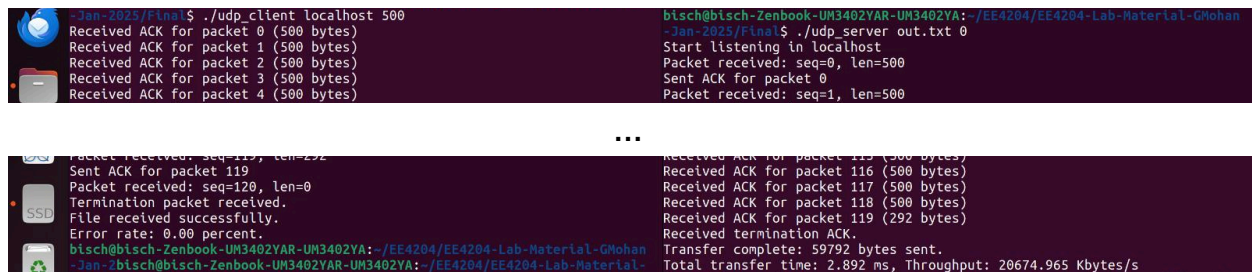
The UDP server needs to be run first with the following command:

`./udp_server <output_file> <error_rate 0 to 1>`

Finally, open a new terminal and run the UDP Client with the following command:

`./udp_client <IP address or localhost> <data unit size, capped at 1400>`

As seen below when both codes ran successfully:



```
blisch@blisch-Zenbook-UM3402YAR-UM3402YA:~/EE4204/EE4204-Lab-Material-GMohan
-Jan-2025/Final$ ./udp_client localhost 500
Received ACK for packet 0 (500 bytes)
Received ACK for packet 1 (500 bytes)
Received ACK for packet 2 (500 bytes)
Received ACK for packet 3 (500 bytes)
Received ACK for packet 4 (500 bytes)

blisch@blisch-Zenbook-UM3402YAR-UM3402YA:~/EE4204/EE4204-Lab-Material-GMohan
-Jan-2025/Final$ ./udp_server out.txt 0
Start listening in localhost
Packet received: seq=0, len=500
Sent ACK for packet 0
Packet received: seq=1, len=500

...

blisch@blisch-Zenbook-UM3402YAR-UM3402YA:~/EE4204/EE4204-Lab-Material-GMohan
-Jan-2025/Final$ ./udp_client localhost 500
Received ACK for packet 115 (500 bytes)
Sent ACK for packet 119
Packet received: seq=120, len=0
Termination packet received.
File received successfully.
Error rate: 0.00 percent.
blisch@blisch-Zenbook-UM3402YAR-UM3402YA:~/EE4204/EE4204-Lab-Material-GMohan
-Jan-2025/Final$ ./udp_server out.txt 0
Received ACK for packet 115 (500 bytes)
Received ACK for packet 116 (500 bytes)
Received ACK for packet 117 (500 bytes)
Received ACK for packet 118 (500 bytes)
Received ACK for packet 119 (292 bytes)
Received termination ACK.
Transfer complete: 59792 bytes sent.
Total transfer time: 2.892 ms, Throughput: 20674.965 Kbytes/s
```

Finally, if the process went smoothly, the output file (e.g. out.txt) will contain the same message as “myfile.txt” as seen below.



```
1
2
3 configure: exit 0
4 This file contains any messages produced by compilers while
5 running configure, to aid debugging if configure makes a mistake.
6
7 It was created by configure, which was
8 generated by GNU Autoconf 2.52.  Invocation command line was
9
10  $ /home/zhqun/otcl-1.8/configure
11
12 ## ----- ##
13 ## Platform.  ##
14 ## ----- ##
15
16 hostname = 79-9.priv19.nus.edu.sg
17 uname -m = i686
```

Program Overview

My program implements a UDP-based file transfer between a client and a server using stop-and-wait ARQ. The server creates a socket (family AF_INET, type SOCK_DGRAM, protocol 0), binds it to the server's address using the bind() function, and waits to receive packets using recvfrom(). Upon receiving a packet, it simulates transmission errors based on a given probability by the user. If an error is simulated, it replies with a NAK (flag == 0); otherwise, it writes the data to a file and sends an ACK (flag == 1), and the amount of actual error is tracked. If a termination packet (len == 0) is received, it sends a final ACK and ends the session, printing a success message and the actual error rate.

On the client side, a socket is created the same way as the server, and the file (myfile.txt) is read in chunks of size based on the user inputted *data unit size*. Each chunk is sent to the server using sendto(). The client then waits for a response using recvfrom(). If a valid ACK is received, it proceeds to the next packet; if not, it resends the current one. After all data is sent, the client transmits a termination packet and waits for its acknowledgment. Finally, transfer statistics such as elapsed time and throughput are computed at the end.

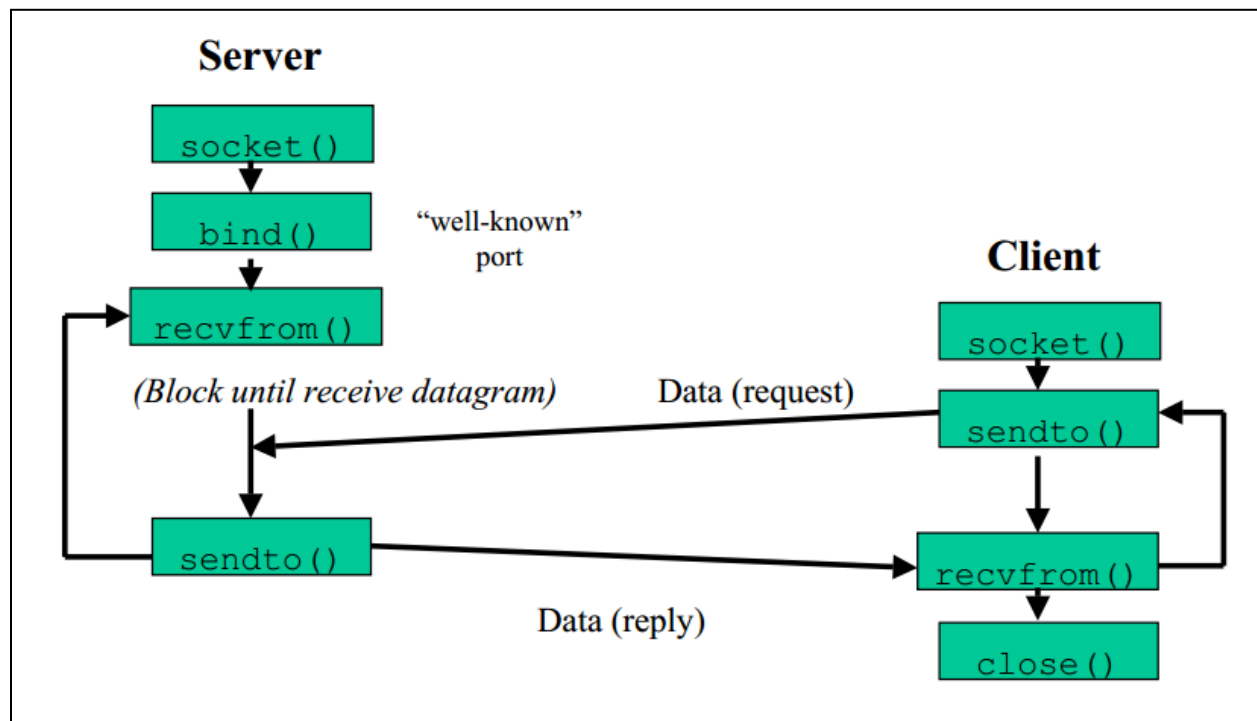


Image Source: Stackoverflow.

<https://stackoverflow.com/questions/23068905/is-bind-necessary-if-i-want-to-receive-data-from-a-server-using-udp>

Data Collection

During data collection, I commented out this line to provide a consistent error rate for different tries of the same args. Because I added this line to simulate an actual error rate, let's say the input is 0.2, the actual error rate can be 0.12 or 0.33, depending on the tries. Removing this gives a constant seed for the `c rand()`, making measurement more consistent.

```
// If don't add this, error rate will be constant in all tries of the same inputted rate  
srand(time(NULL));
```

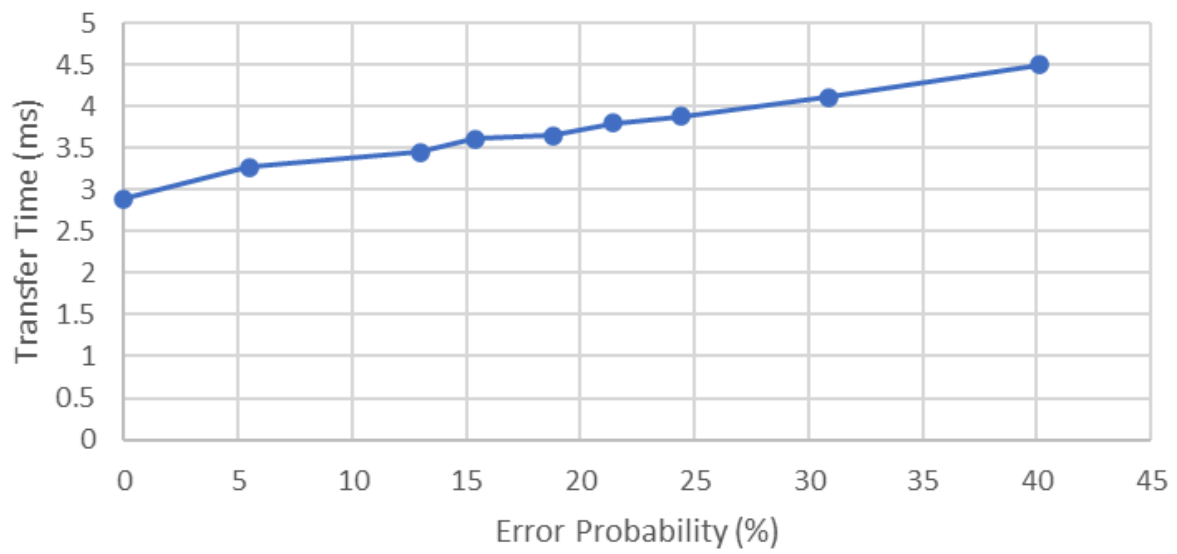
Highlighted in red are error values with bad seed (causing a big difference between the actual and set, we assume bad seed as a percentage discrepancy of 4 percent or grater).

1) Transfer time vs error probability (2 graphs; size = 500 and size = 1000 bytes)

a.) Packet size = 500 bytes

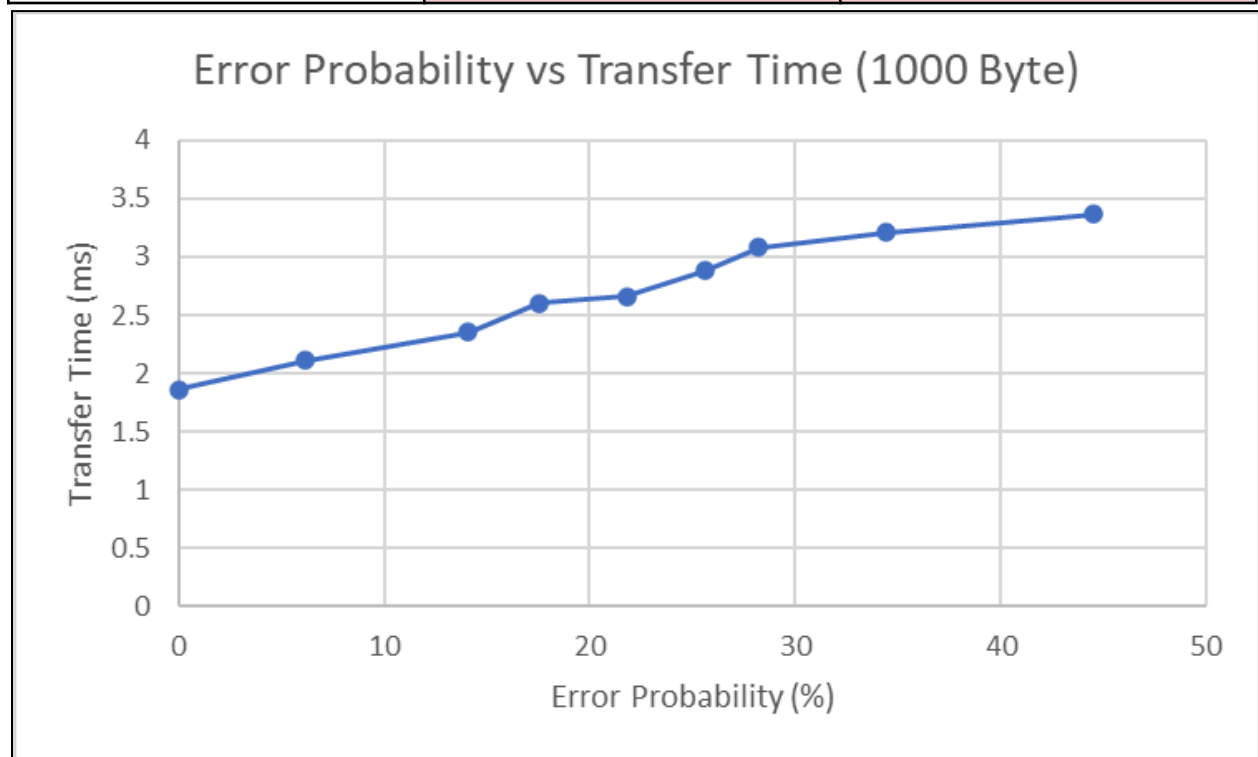
Transfer Time (ms)	Set Error Percentage (%)	Actual Error Percentage (%)
2.892	0	0
3.268	5	5.47
3.449	10	12.95
3.607	15	15.38
3.646	20	18.79
3.796	25	21.43
3.878	30	24.38
4.107	35	30.86
4.502	40	40.1

Error Probability vs Transfer Time (500 Byte)



b.) Packet size = 1000 bytes

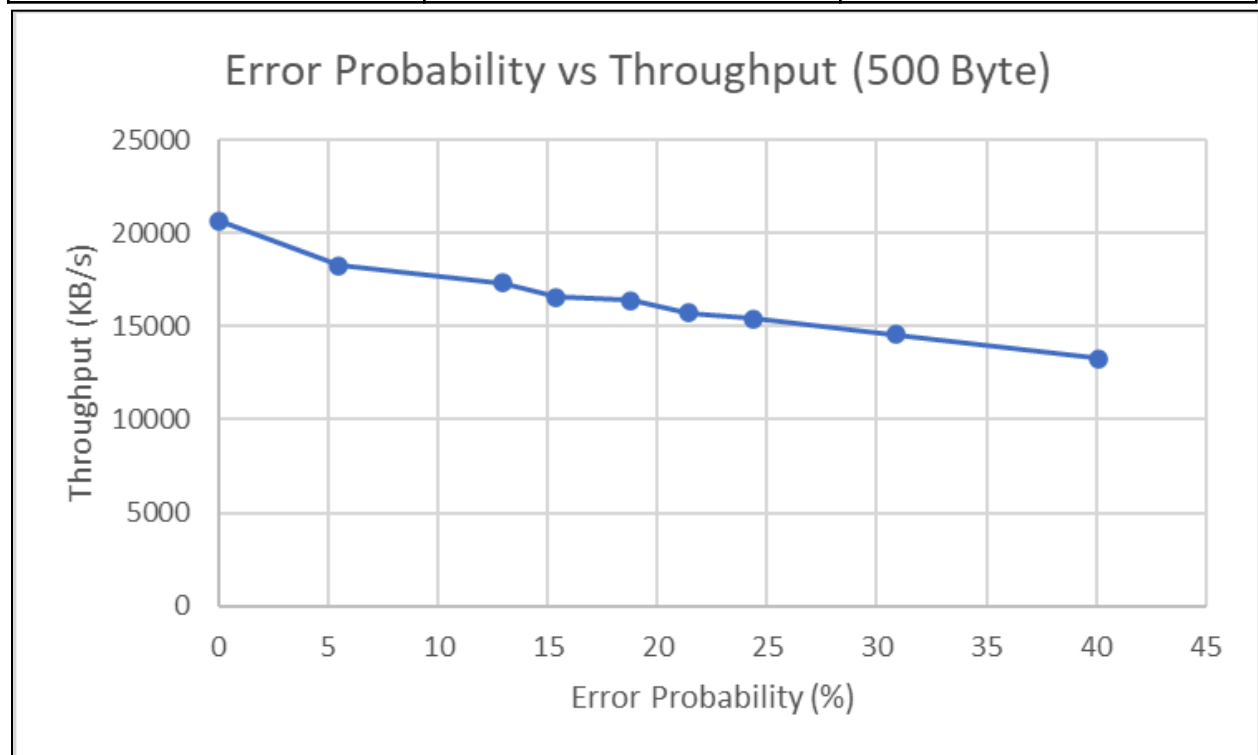
Transfer Time (ms)	Set Error Percentage (%)	Actual Error Percentage (%)
1.861	0	0
2.110	5	6.15
2.354	10	14.08
2.606	15	17.57
2.661	20	21.79
2.883	25	25.61
3.078	30	28.24
3.212	35	34.41
3.367	40	44.55



2) Throughput vs error probability (2 graphs; size = 500 and size = 1000 bytes)

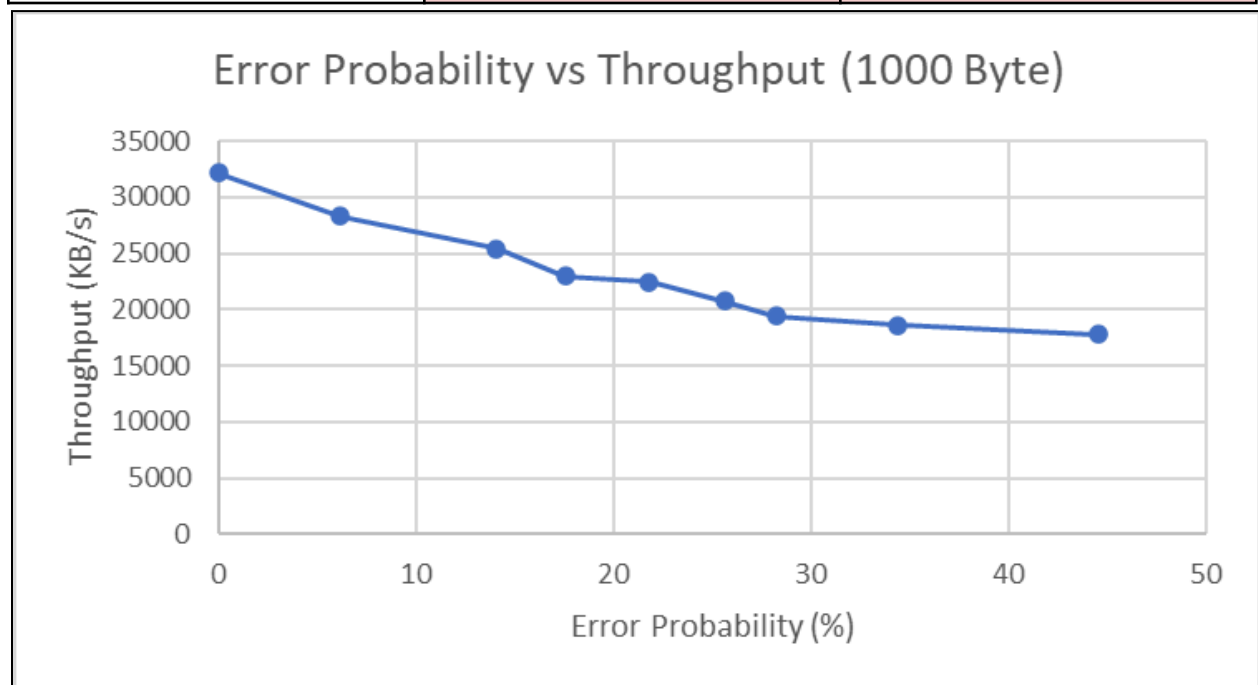
a.) Packet size = 500 bytes

Throughput (KB/s)	Set Error Percentage (%)	Actual Error Percentage (%)
20674.965	0	0
18296.206	5	5.47
17336.039	10	12.95
16576.657	15	15.38
16399.342	20	18.79
15751.317	25	21.43
15418.257	30	24.38
14558.559	35	30.86
13281.208	40	40.1



b.) Packet size = 1000 bytes

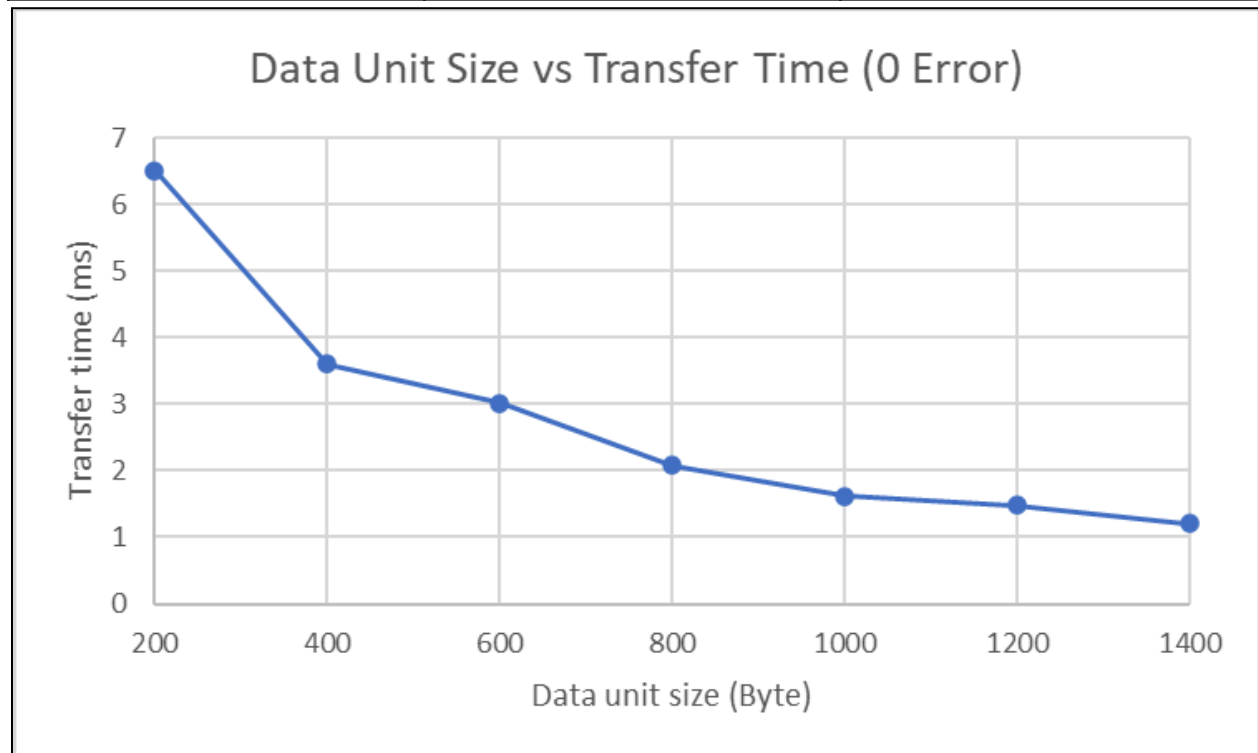
Throughput (KB/s)	Set Error Percentage (%)	Actual Error Percentage (%)
32128.963	0	0
28337.441	5	6.15
25400.170	10	14.08
22943.975	15	17.57
22469.748	20	21.79
20739.507	25	25.61
19425.601	30	28.24
18615.193	35	34.41
17758.242	40	44.55



3) Transfer time vs data unit size (2 graphs; error probability = 0, error probability = 0.1)

a.) Error rate = 0

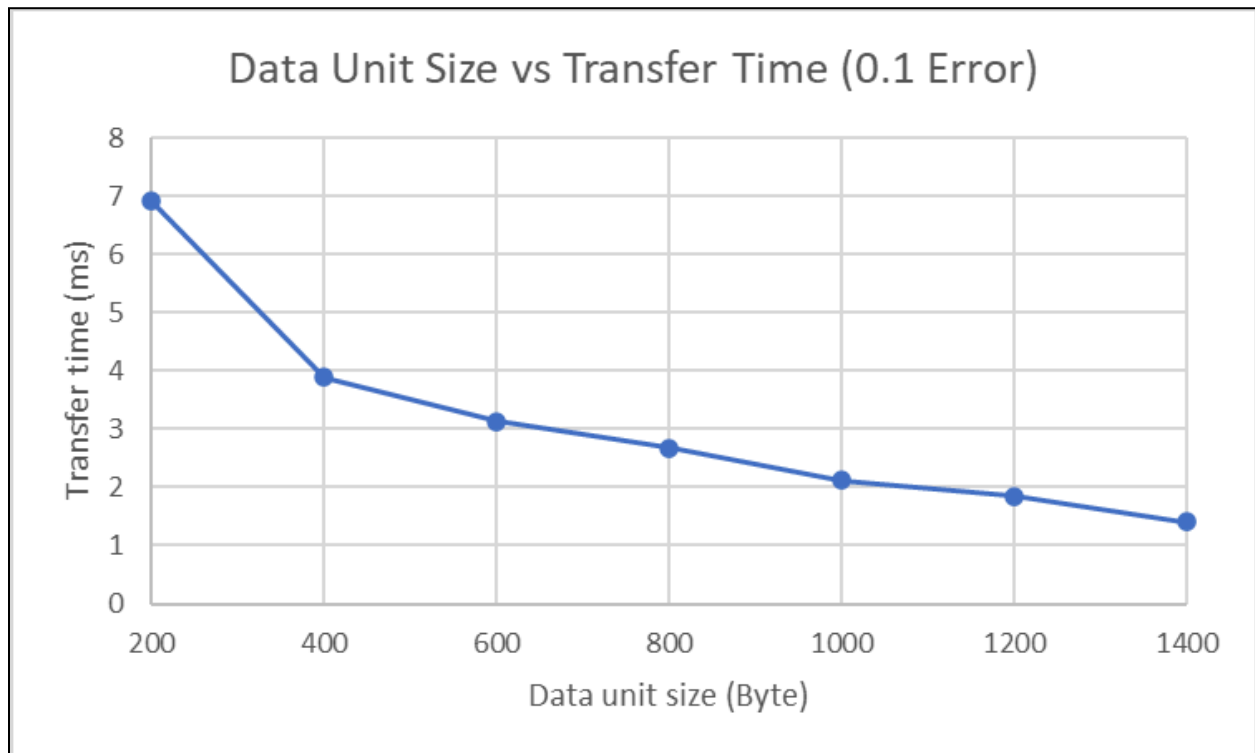
Transfer time (ms)	Data unit size (byte)	Actual Error Percentage (%)
6.508	200	0
3.604	400	0
3.022	600	0
2.075	800	0
1.612	1000	0
1.485	1200	0
1.204	1400	0



b.) Error rate = 0.1

Do note that due to the error probability being random, it can be less or higher than 0.1.

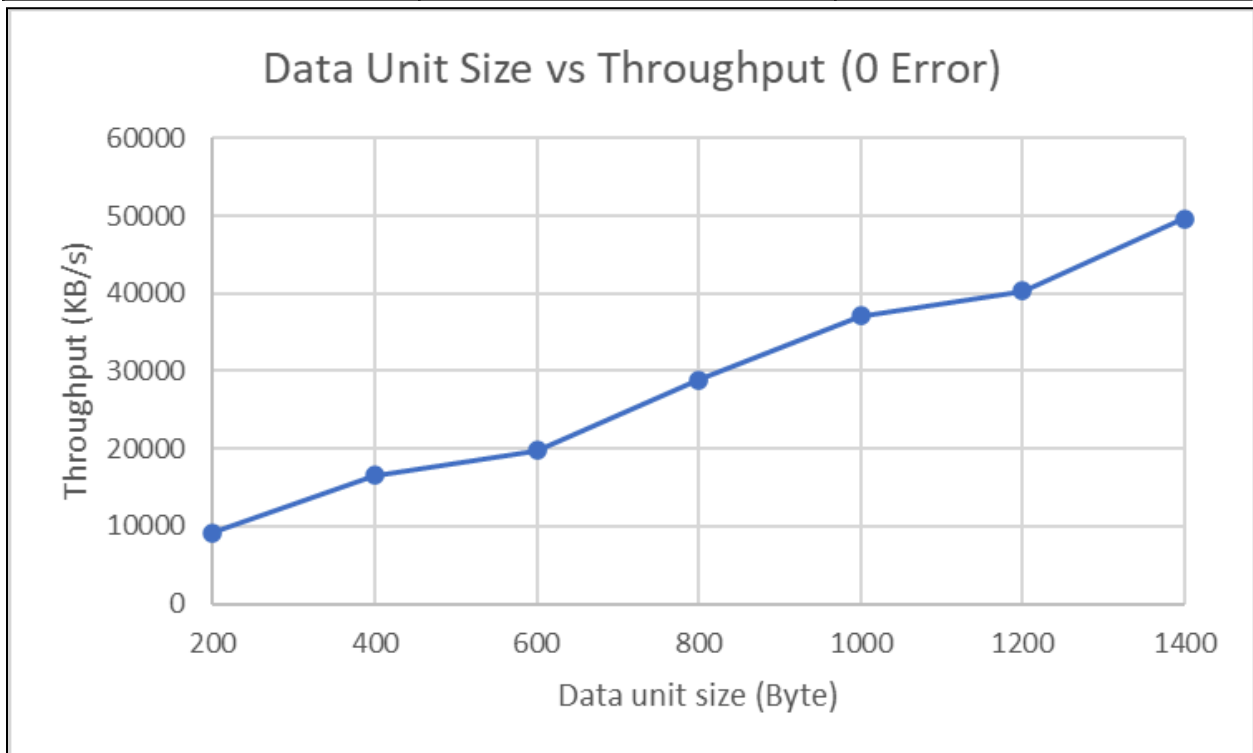
Transfer time (ms)	Data unit size (byte)	Actual Error Percentage (%)
6.923	200	9.09
3.887	400	11.07
3.130	600	13.68
2.680	800	13.64
2.121	1000	14.08
1.855	1200	15.00
1.410	1400	15.38



4) Throughput vs data unit size (2 graphs; error probability = 0, error probability = 0.1)

a.) Error rate = 0

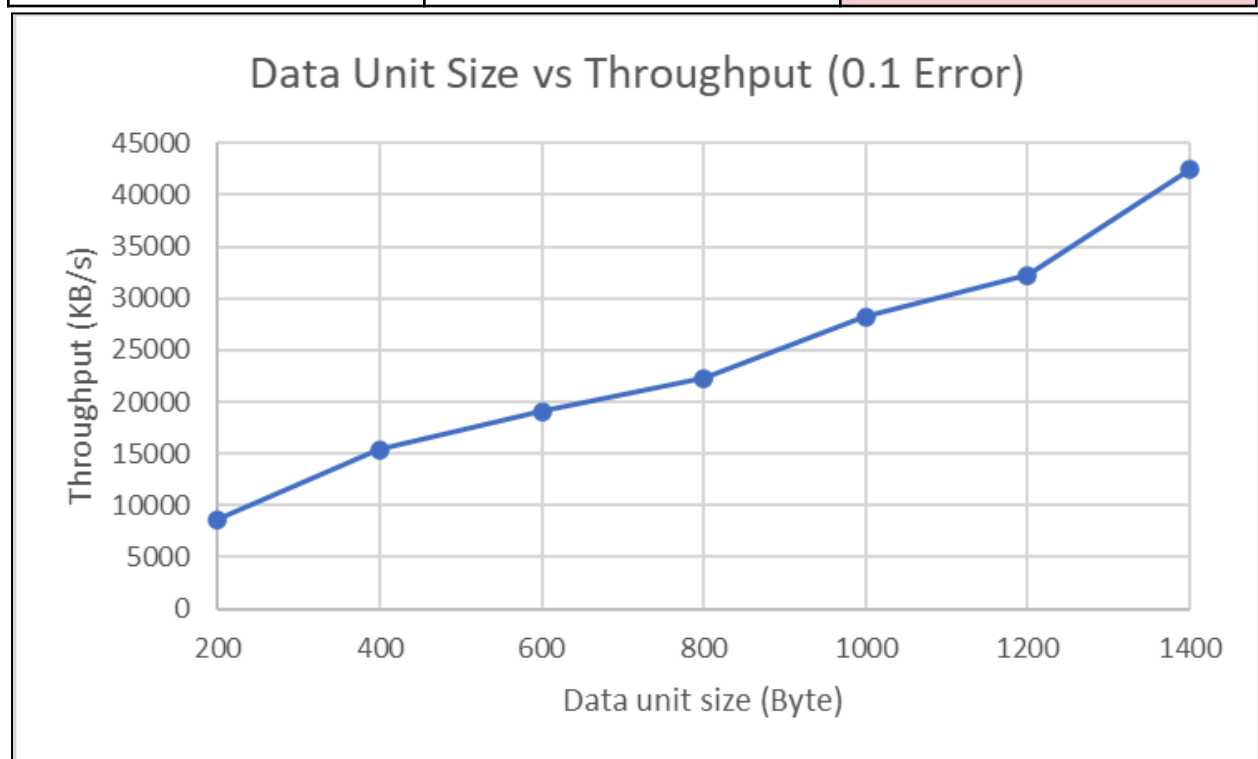
Throughput (KB/s)	Data unit size (byte)	Actual Error Percentage (%)
9187.462	200	0
16590.455	400	0
19785.572	600	0
28815.422	800	0
37091.811	1000	0
40263.973	1200	0
49661.130	1400	0



b.) Error rate = 0.1

Do note that due to the error probability being random, it can be less or higher than 0.1.

Throughput (KB/s)	Data unit size (byte)	Actual Error Percentage (%)
8636.718	200	9.09
15382.557	400	11.07
19102.875	600	13.68
22310.448	800	13.64
28190.476	1000	14.08
32232.884	1200	15.00
42405.674	1400	15.38



Observation and Conclusion

1. Throughput is inversely proportional to the transfer time, where the higher the throughput, the lower the transfer time. Throughput is calculated by dividing the total data sent by the actual transfer time.

$$\text{Throughput} = \frac{\text{Total data sent}}{\text{Transfer time}}$$

2. Error rate is proportional to transfer time; the higher the error, the higher the transfer time. A higher error rate causes more frequent retransmissions, and this retransmission causes the transfer time to increase. Let's say the error rate is 20%; approximately 20% of the total packets have to be retransmitted.
3. Error rate is inversely proportional to throughput; the higher the error, the lower the throughput. As explained in (1), throughput and transfer time are inversely proportional; as the error rate increases, transfer time increases, hence, throughput decreases.
4. Larger data unit size means less packet that needs to be sent due to the following:

$$\text{No of packets} = \frac{\text{Total data sent}}{\text{Data unit size}}$$

Less packet allows smaller transfer time, and smaller transfer time results in higher throughput. However, when an error occurs, the data to be retransmitted is higher for packets with large data unit size. Also, do note that for network LAN, UDP only supports a 1500 MTU (maximum transmission unit), hence, there is a limit to how big the packet can be.

5. Anomalies and random spikes may occur during testing. For example, given a 500 data unit size, 10 percent error, the transfer time is around 2 to 3 ms. However, there may be cases where I observed the output to go over 8 ms or even 10 ms. This may be caused by the following:
 - CPU & I/O overhead
 - I am printing every time a packet or ack is sent, even retransmissions, causing a lot of CPU overhead.
 - `fwrite()` may be affected by the disk cache and write speed.
 - UDP socket congestion
 - Some processes or background applications might be running and causing congestion to that port (localhost).
 - Function call overhead
 - We are calling functions such as `recvfrom()`, `gettimeofday()` that may come with inherent noise, leading to variability.