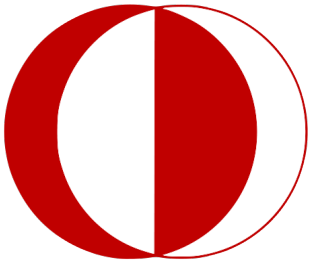
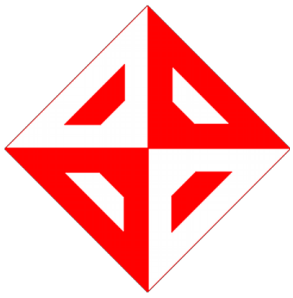




Socket Programming

CENG435: Data Communications and Networking

Gürhan İlhan Adıgüzel
2448025
gurhan.adiguzel@metu.edu.tr

Aybüke Aksoy
2448090
aybuke.aksoy@metu.edu.tr

Computer Engineering
Middle East Technical University

Academic Year 2023-2024

05 January 2024

Contents

1 Introduction

In this assignment, we have implemented a file transfer program using layer 4 protocols which are TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) talking across an unreliable channel. We have developed 2 socket applications. The first one uses TCP with single persistent connection to transfer the large and small objects consecutively and the second one implements a reliable pipelined data transport protocol over an unreliable channel UDP and tries to overcome head-of-line blocking and outperform TCP. Both implementations use client-server architecture.

Following the implementation, we have conducted experiments to compare the time taken by these 2 socket applications UDP, TCP and UDP Optimal version (which is obtained by changing to the approximately optimal timeout value during the implementation) to download the total number of 20 large and small files under different tc/netem rules.

Those rules consist of:

- Benchmark, no tc/netem rules applied
- Packet loss: $0\%, 5\%, 10\%, 15\%$
- Packet duplication: $0\%, 5\%, 10\%$
- Packet corruption: $0\%, 5\%, 10\%$
- Packet delay: 100ms - uniform distribution, 100ms - normal distribution

The experiments with each rule is run 30 times in isolation with only one parameter changed. Other parameters are set to default values.

The results of experiments can be found under the sections named by each rule.

2 Experiments

For the TCP Implementation:

• Packet size: 1500

For the UDP Implementation

• Selective Repeat Protocol is used.

• Packet size: 1500

• Window size: 120

• Default timeout on server's side for retransmission: 0.13s

Those values for packet size and window size are set as default in each experiment.

In Selective Repeat, the sender can send multiple packets before receiving an acknowledgment for the previously sent packets. The receiver acknowledges each correctly received packet, and if a packet is found to be corrupted or lost (in our case only lost as UDP checks for packet corruption), the sender retransmits that specific packet. It is more efficient than Go-Back N when loss or corruption occurs but also more complex.

In most SR implementations window size should be less than or equal to half of 2^m where m is the number of bits needed to represent the sequence number. This is crucial to avoid recognizing newly transmitted packets as retransmissions. In our case, when packet size is 1500, sequence number is 7 for small packets and 658 for large packets. So, the window size should be chosen as less than or equal to 4 for small packets and less than or equal to 512 for large packets. However, our implementation is not restricted by this as we have the packet number information and increasing the sequence number after sending a packet until it reaches to packet number value. Hence, we have decided on the window size experimentally. It is chosen large enough to improve efficiency of data transmission by exploiting available network bandwidth by also considering the memory usage, processing overhead, and error handling on the client side. It was observed that we had better results with window size around 100-120 packets.

0.13s for timeout is chosen since this value is the experimentally found approximate optimal value for Packet Delay rule and it will be used as default value for all rules to make comparison easier.

2.1 Benchmark

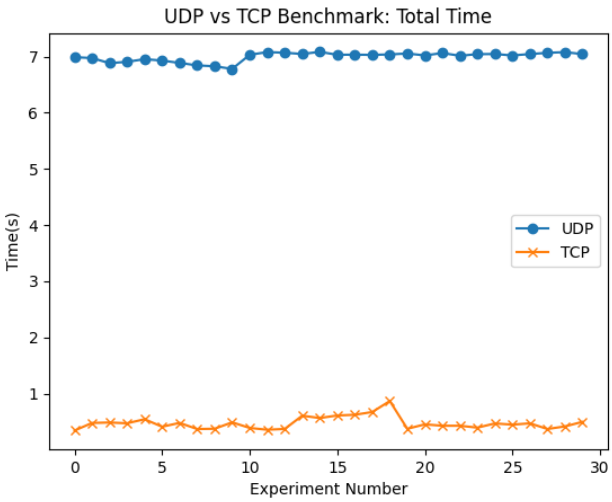
In this part, no tc/netem rule is applied. The total download time for TCP, UDP and UDP Optimal is taken over 30 runs.

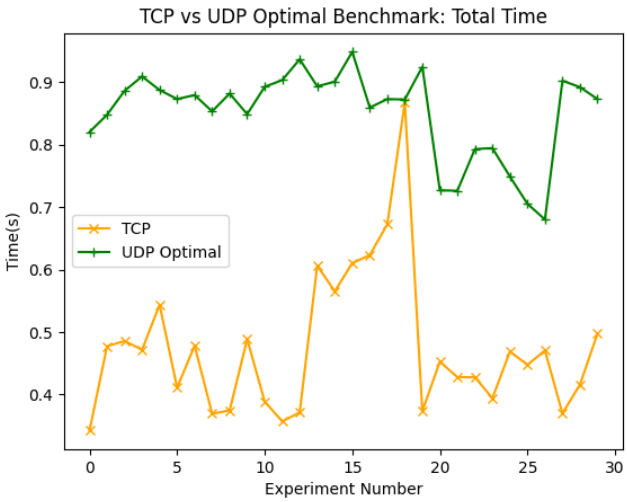
In UDP implementation's server side, timeout to retransfer the packets is set to 0.13s and in UDP Optimal implementation's server side, timeout to retransfer the packets is set to 0.01s.

0.01 is chosen according to average RTT since there is only a few retransmissions and timeouts with the quality of connection in default settings.

2.1.1 Total Download Time Over 30 Runs

The comparison plots of the total download time for UDP-TCP and TCP-UDP Optimal over 30 runs are as below:



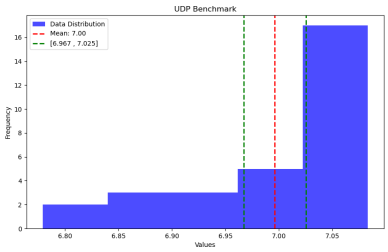


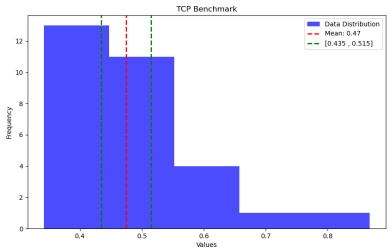
Retransmission counts for UDP in one of the runs:

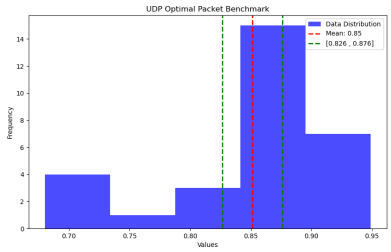
large-0	small-0	large-1	small-1	large-2
5	0	5	0	5
small-2	large-3	small-3	large-4 of	small-4
0	5	0	5	0
large-5	small-5	large-6	small-6	large-7
5	0	5	0	5
small-7	large-8	small-8	large-9	small-9
5	0	5	0	5

2.1.2 Confidence Intervals

The plots for the 95% confidence interval are as below:







2.1.3 Discussion

It can be seen from the results that UDP takes around 7s on average, TCP takes around 0.4s on average and UDP Optimal takes around 0.8s on average to transfer all files. The huge difference between UDP-TCP and UDP-UDP optimal is due to the default timeout chosen. The default timeout is way bigger than the average RTT for benchmark settings. The lost packets are being retransmitted later than they could as the server waits for timeout to resend even if there is no time needed for processing packets on client side.

UDP optimal overcomes this problem by setting the timeout to 0.01. The total download time decreases drastically yet it is still longer than TCP's. This is mainly due to TCP's various mechanisms such as congestion control, dynamic window sizing and flow control. TCP uses congestion control algorithms to manage the flow of data in the network by balancing its sending rate and it uses dynamic window sizing to adapt to network conditions by increasing its window size when there is few packet loss. Also it prevents

the sender from overwhelming the receiver by flow control. With these mechanisms, TCP can efficiently utilize available bandwidth in a stable network. However, our UDP implementations are lacking these dynamic parameters and hence it is slower than TCP.

2.2 Packet Loss

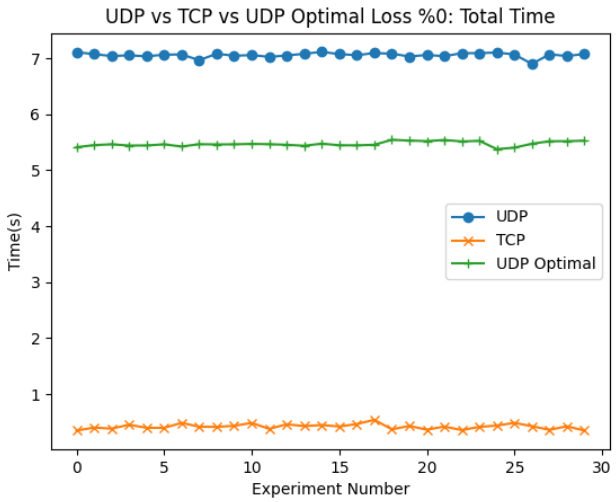
In this part, tc/netem rule for loss is applied. The total download time for TCP, UDP and UDP Optimal is taken over 30 runs.

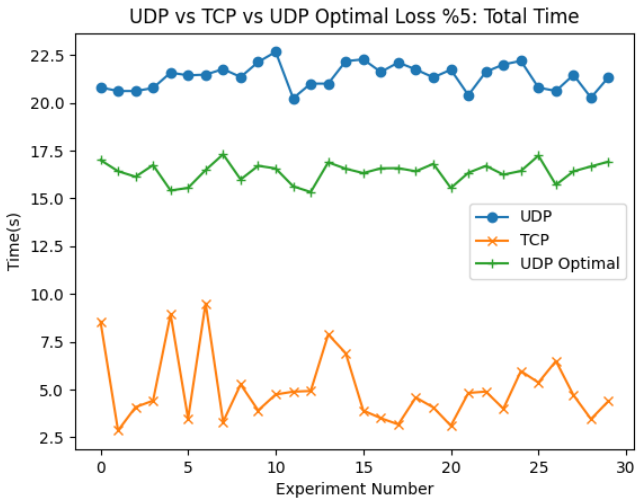
In UDP implementation's server side, timeout to retransfer the packets is set to 0.13s and in UDP Optimal implementation's server side, timeout to retransfer the packets is set to 0.10s.

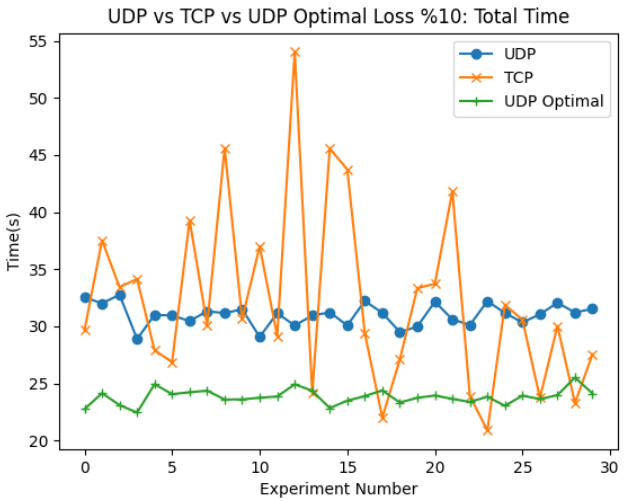
With more packets being lost, the processing time on the client side needed to recognize retransmitted and new coming packets is longer. Hence a longer timeout than the one in benchmark settings should be given. 0.10 is chosen according to average RTT and the number of retransmissions required in loss %15.

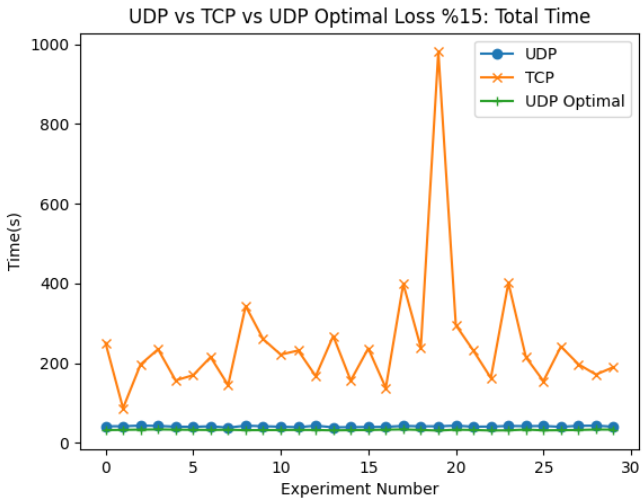
2.2.1 Total Download Time Over 30 Runs

The comparison plots of the total download time for UDP-TCP and TCP-UDP Optimal over 30 runs are as below and it can be seen from the results that:









For loss %0, UDP takes around 7s on average, TCP takes around 0.4s on average as they were on benchmark settings and UDP Optimal takes around 5.5s on average to transfer all files.

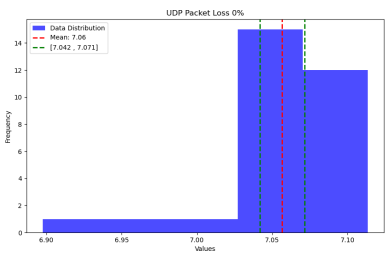
For loss %5, UDP takes around 21s on average, TCP takes around 5s on average and UDP Optimal takes around 16s on average to transfer all files.

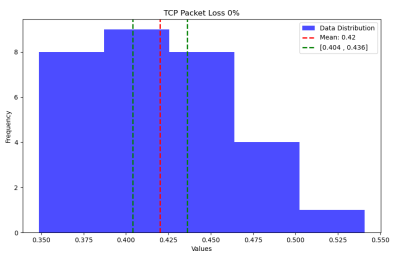
For loss %10, UDP takes around 31s on average, TCP takes around 32s on average and UDP Optimal takes around 24s on average to transfer all files.

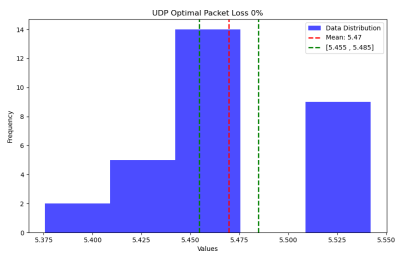
For loss %15, UDP takes around 41s on average, TCP takes around 245s on average and UDP Optimal takes around 32.5s on average to transfer all files.

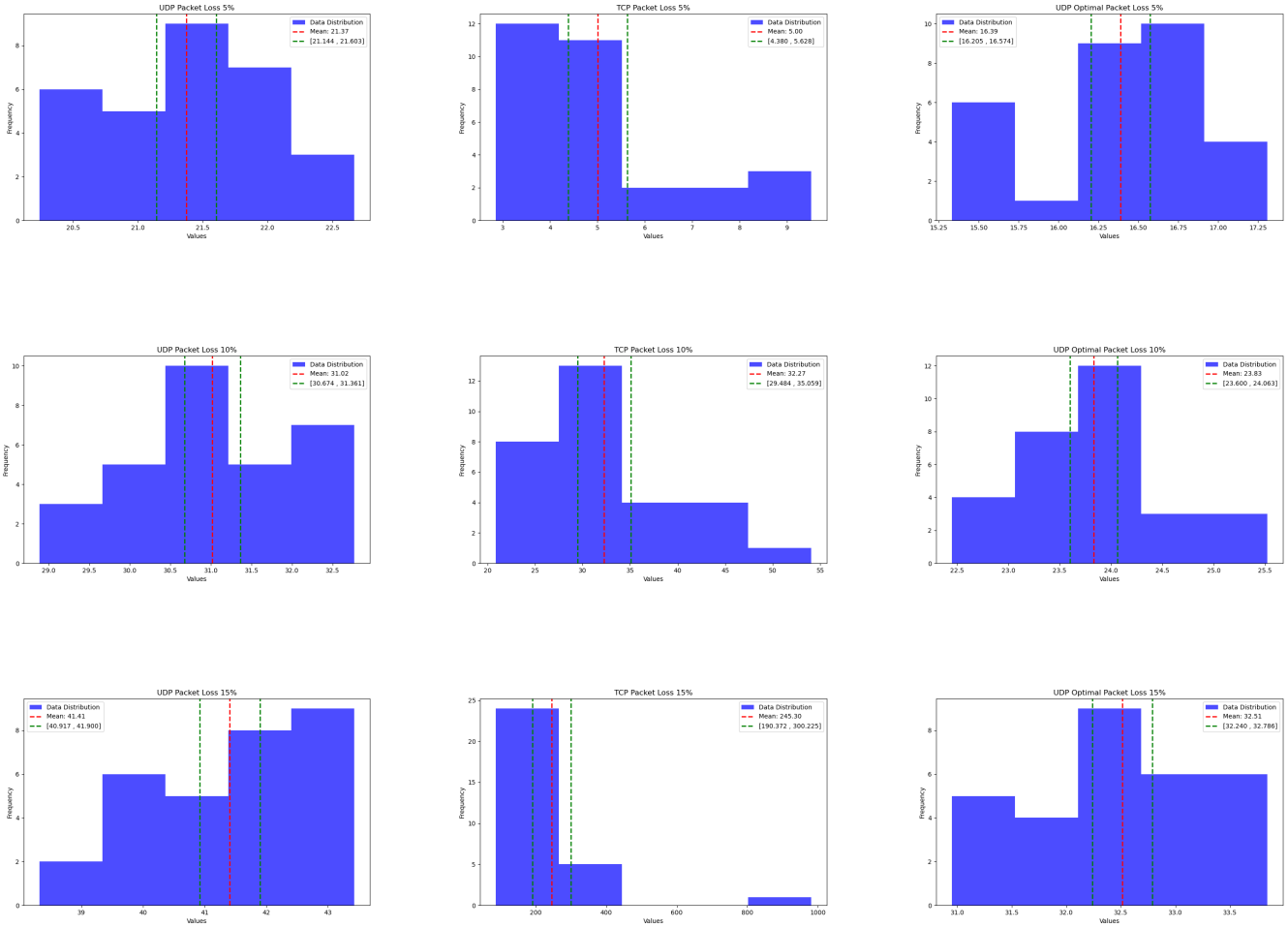
2.2.2 Confidence Intervals

The plots for the 95% confidence interval are as below:

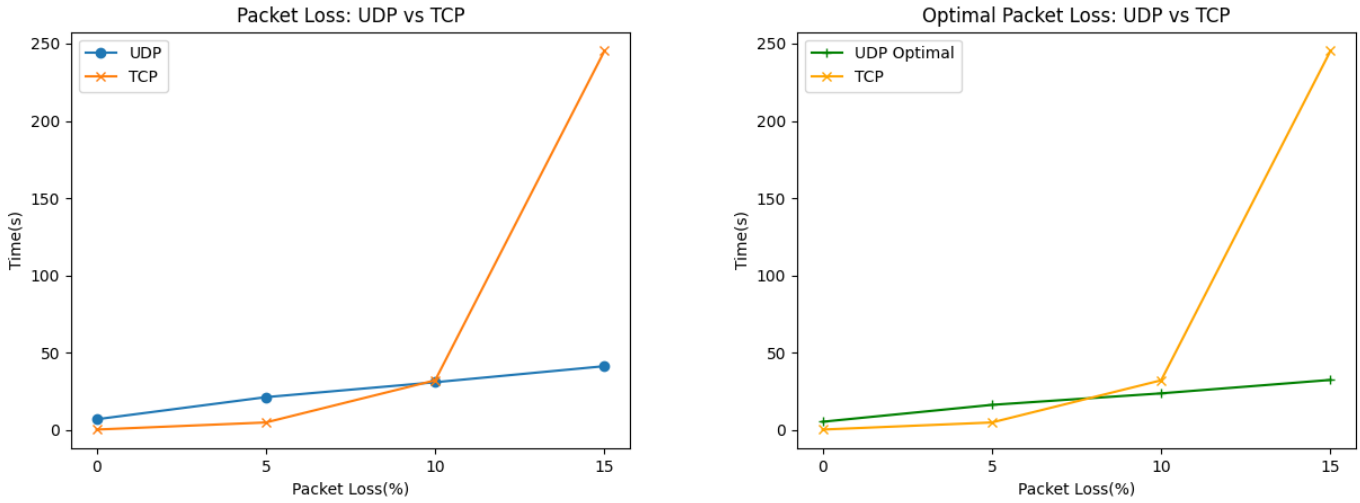








2.2.3 Total Download Time Over Loss Percentage



Retransmission counts for UDP in one of the runs for loss %5:

large-0	small-0	large-1	small-1	large-2
18	0	18	2	16
small-2	large-3	small-3	large-4 of	small-4
1	13	0	18	1
large-5	small-5	large-6	small-6	large-7
16	1	15	0	13
small-7	large-8	small-8	large-9	small-9
0	17	1	14	1

Retransmission counts for UDP in one of the runs for loss %15:

large-0	small-0	large-1	small-1	large-2
30	1	32	1	30
small-2	large-3	small-3	large-4 of	small-4
4	26	1	26	1
large-5	small-5	large-6	small-6	large-7
28	0	37	1	31
small-7	large-8	small-8	large-9	small-9
2	28	8	31	3

2.2.4 Discussion

We can observe that for 10% loss and higher, there is huge decline in TCP's performance and our RDT over UDP implementation and the optimal version outperforms TCP. With loss increasing, this performance gap gets bigger rapidly. TCP uses Selective Repeat Protocol as we do in our implementation. Hence the protocol choice is not the reason for the poor performance. This is mainly due to congestion control mechanism that TCP implements. The existing TCP control mechanisms rely on packet loss to detect congestion. With loss tc/netem rule, we are introducing external packet loss, when there is no congestion in the network. TCP slows down and tries to control the congestion but as retransmissions increase with external packet loss, the traffic burden on the network also increases and causes TCP to slow down more. After some point, router queue lengths might reach their maxima. As a result, routers might drop the packets and the network will suffer from more loss and more delays which causes TCP to end up with huge performance loss.

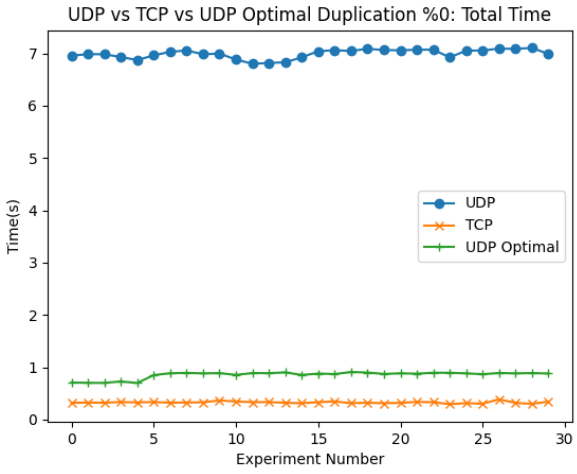
2.3 Packet Duplication

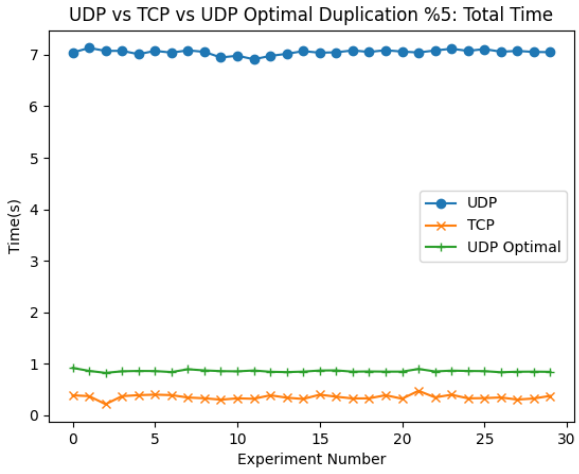
In this part, tc/netem rule for corruption is applied. The total download time for TCP, UDP and UDP Optimal is taken over 30 runs.

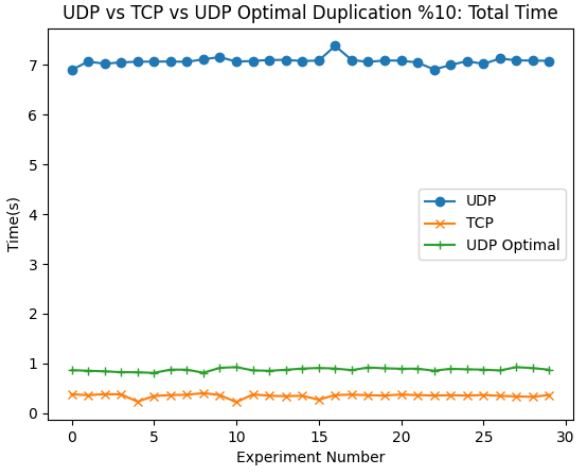
In UDP implementation's server side, timeout to retransfer the packets is set to 0.13s and in UDP Optimal implementation's server side, timeout to retransfer the packets is set to 0.01.

0.01 is chosen as duplication is not related to retransmission and hence is not closely related to timeout. It does not put burden on neither client nor the server side, so the values of the benchmark settings can be used efficiently.

2.3.1 Total Download Time Over 30 Runs



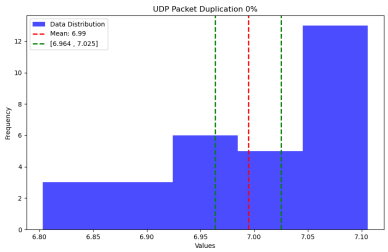


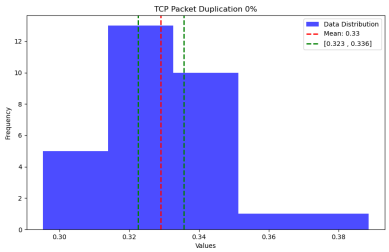


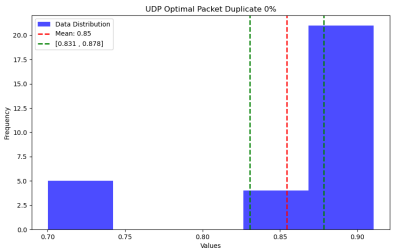
It can be seen that for all duplication percentages, the results are close to each other and difference is negligible. UDP takes around 7s on average, TCP takes around 0.3s on average and UDP Optimal takes around 0.8s on average to transfer all files.

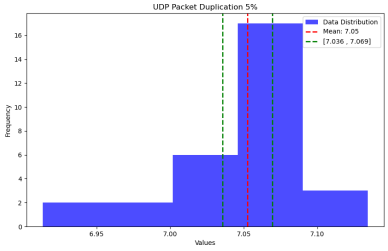
2.3.2 Confidence Intervals

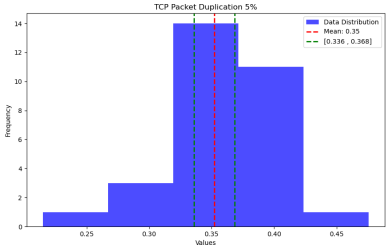
The plots for the 95% confidence interval are as below:

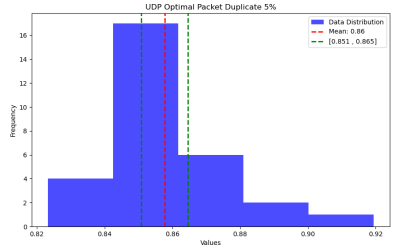


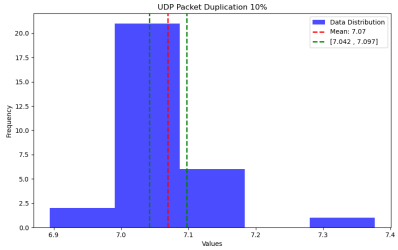


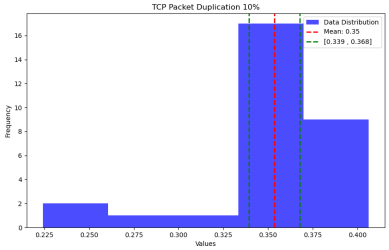


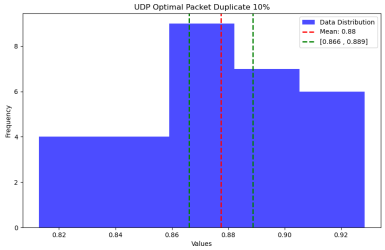




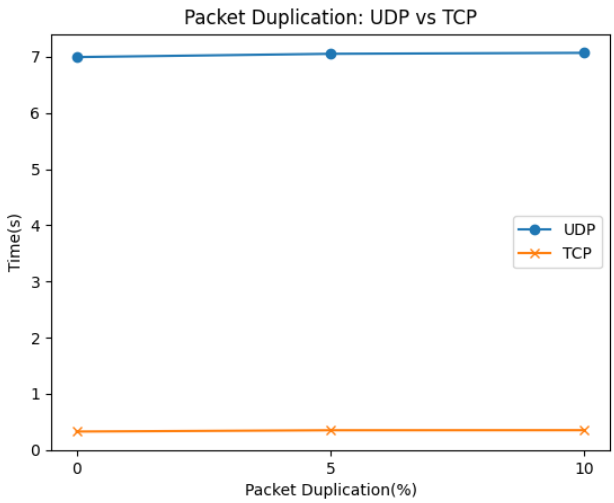


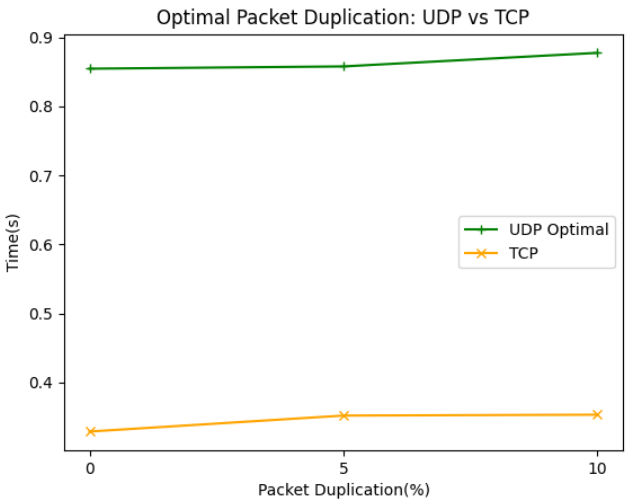






2.3.3 Total Download Time Over Duplication Percentage





Retransmission counts for UDP in runs for duplication $0\%, 5\%, 10\%$:

large-0	small-0	large-1	small-1	large-2
5	0	5	0	5
small-2	large-3	small-3	large-4 of	small-4
0	5	0	5	0
large-5	small-5	large-6	small-6	large-7
5	0	5	0	5
small-7	large-8	small-8	large-9	small-9
5	0	5	0	5

2.3.4 Discussion

Our results for this section is similar to our results in benchmark settings as 0.01 timeout is used. Also, the performance of all implementations are better than their performance in loss, corruption and delay. TCP uses unique sequence numbers for each packet to allow the receiver to identify and discard duplicates. We are also utilizing sequence number in our Selective Repeat implementation to ignore duplicate packets both in server and client side. Therefore, in UDP and TCP duplicate packets only cause a few additional condition checks which does not result in any remarkable delay nor timeouts in the network. Under this stable conditions, TCP can utilize its additional mechanisms and our RDT over UDP also can transfer the

files efficiently.

2.4 Packet Corruption

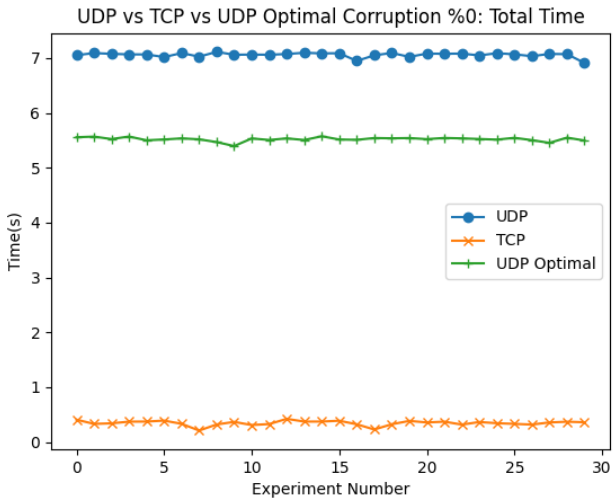
In this part, tc/netem rule for corruption is applied. The total download time for TCP, UDP and UDP Optimal is taken over 30 runs.

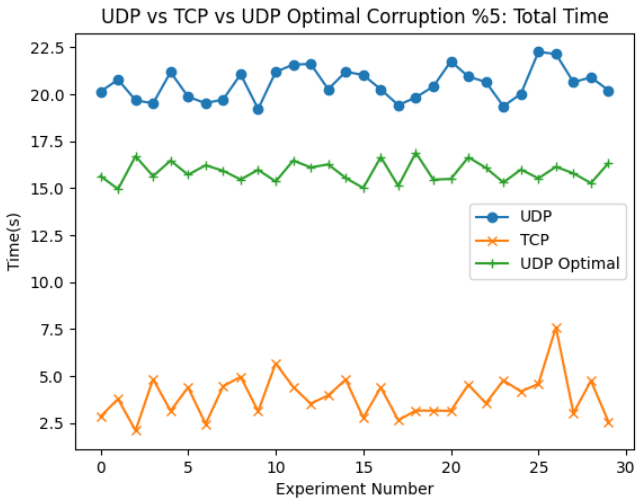
In UDP implementation's server side, timeout to retransfer the packets is set to 0.13s and in UDP Optimal implementation's server side, timeout to retransfer the packets is set to 0.10s.

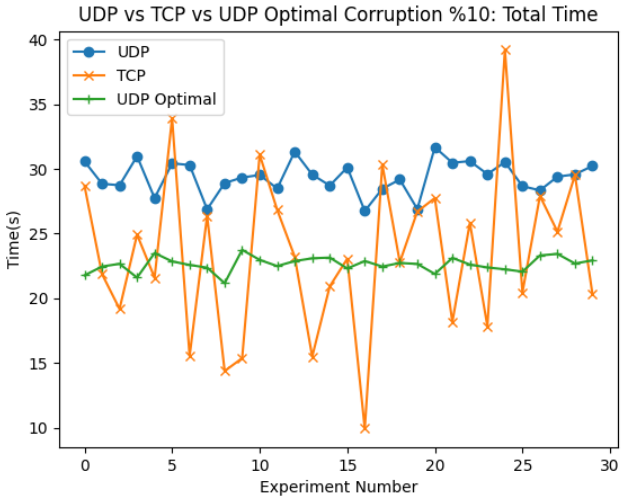
UDP discards the corrupted packets in lower layers, hence with more packets being corrupted, which results in more packets being lost, the processing time on the client side needed to recognize retransmitted and new coming packets is longer. Hence a longer timeout than the one in benchmark settings should be given. 0.10 is chosen according to average RTT and the number of retransmissions required in corruption %10.

2.4.1 Total Download Time Over Corruption Percentage

The comparison plots of the total download time for UDP-TCP and TCP-UDP Optimal over 30 runs are as below and it can be seen from the results that:







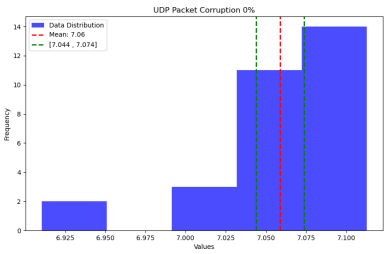
For corruption %0, UDP takes around 7s on average, TCP takes around 0.35s on average as they were on benchmark settings and UDP Optimal takes around 5.5s on average to transfer all files.

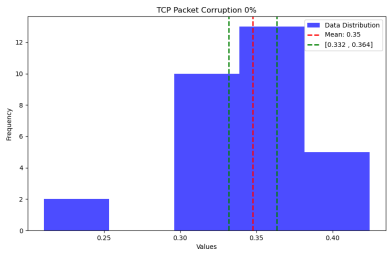
For corruption %5, UDP takes around 21s on average, TCP takes around 4s on average and UDP Optimal takes around 16s on average to transfer all files.

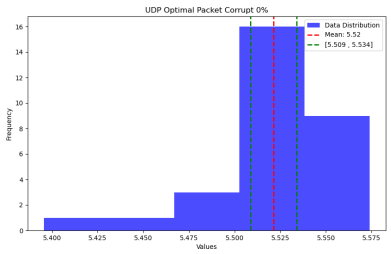
For corruption %10, UDP takes around 29s on average, TCP takes around 23.5s on average and UDP Optimal takes around 22.5s on average to transfer all files.

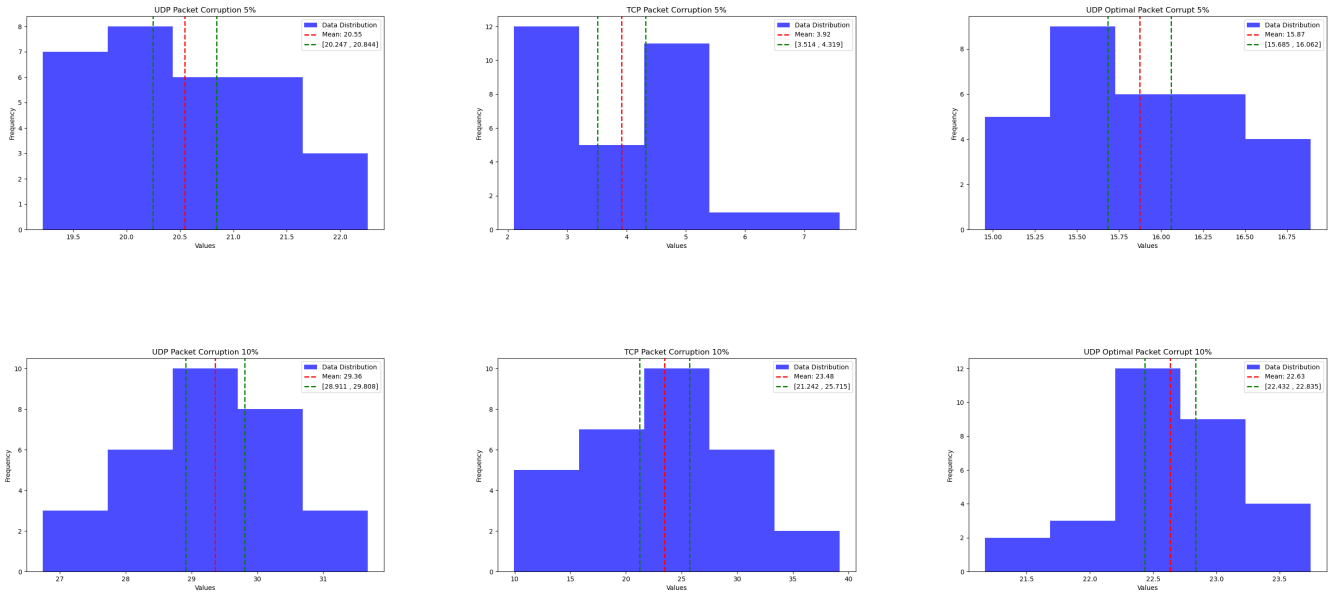
2.4.2 Confidence Intervals

The plots for the 95% confidence interval are as below:

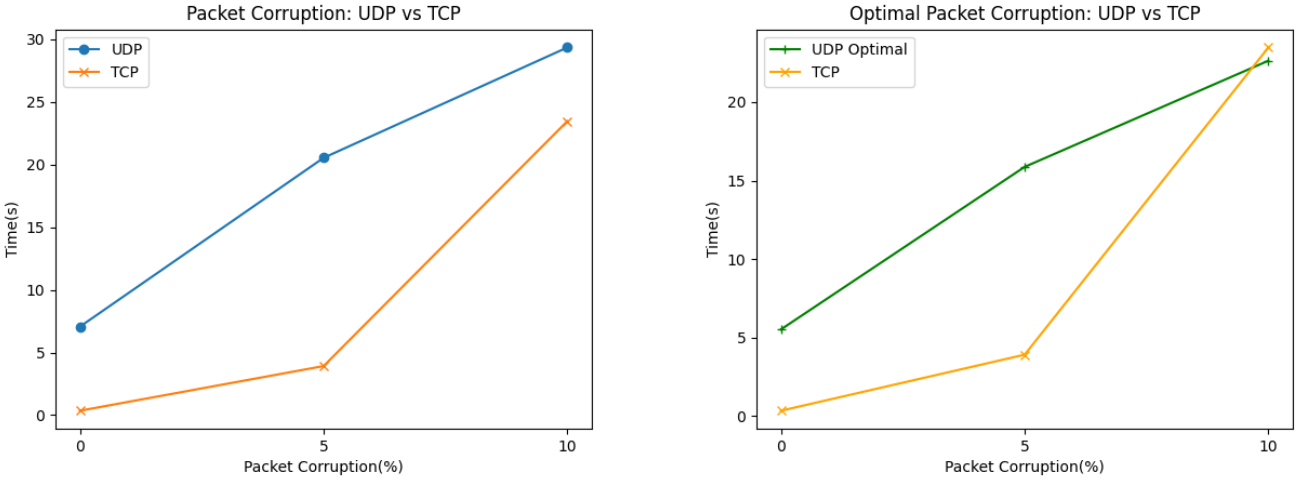








2.4.3 Times Over Corruption Percentage



Retransmission counts for UDP in one of the runs for corruption %5:

large-0	small-0	large-1	small-1	large-2
14	0	15	0	14
small-2	large-3	small-3	large-4 of	small-4
1	13	2	17	0
large-5	small-5	large-6	small-6	large-7
15	1	13	1	13
small-7	large-8	small-8	large-9	small-9
1	15	1	15	0

Retransmission counts for UDP in one of the runs for corruption %10:

large-0	small-0	large-1	small-1	large-2
22	1	24	3	22
small-2	large-3	small-3	large-4 of	small-4
2	20	2	17	2
large-5	small-5	large-6	small-6	large-7
18	1	20	0	20
small-7	large-8	small-8	large-9	small-9
2	22	1	20	2

2.4.4 Discussion

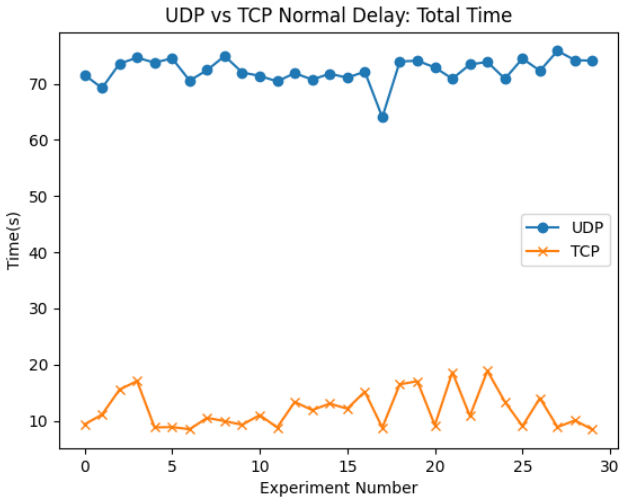
Comparing UDP and TCP, it can be observed that TCP has better performance. Yet, the time taken increase rates with increase in corruption shows that for higher corruption percentages, UDP is expected to pass TCP's performance.

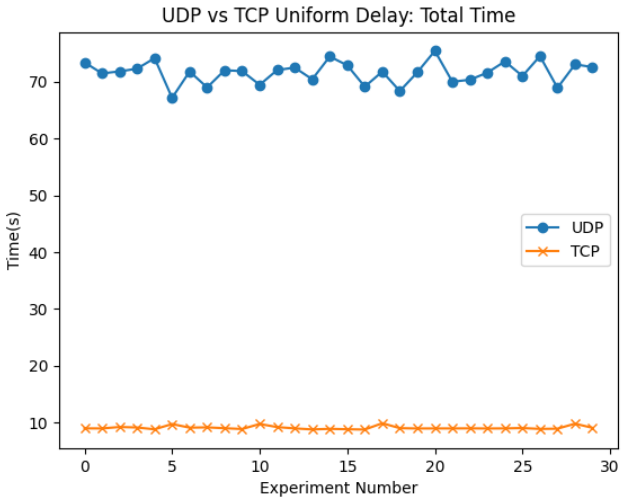
Looking at UDP Optimal vs TCP graph, we can see that UDP Optimal performs slightly better than TCP. This difference can also grow more for higher corruption percentages.

For corrupted packets, UDP has a mechanism called checksum and it is implemented in the lower layers. When a corrupted packet is recognized, UDP does not correct the packet but rather discards it completely. Hence, those corrupted packets will become lost packets for the upper layer. Our Selective Repeat implementation will request that packet to be retransmitted. TCP implements a similar logic with our RDT Protocol. It calculates a checksum to verify the integrity of the data. If the checksum does not match, the packet is considered corrupt and is discarded. Then, it requests that the sender retransmits the missing data. Hence, the results were expected to be similar. However, regardless of these, due to the same reasons explained in the packet loss section, TCP suffers from performance loss. The congestion control mechanism that TCP implements, collapse with external corruption.

2.5 Packet Delay

2.5.1 Total Download Time Over 30 Runs

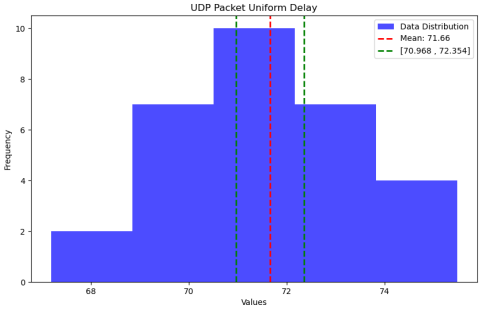


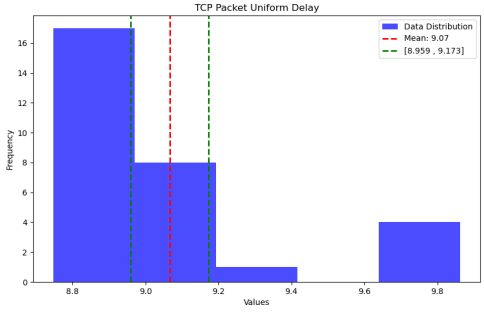


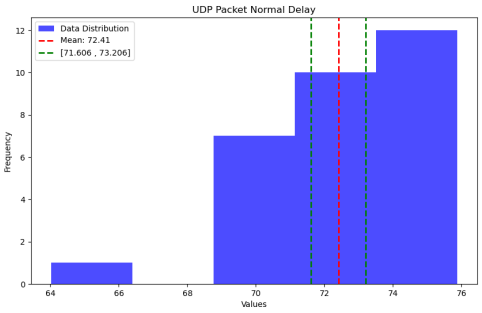
For delay with uniform and normal distribution, UDP takes around 72s on average and TCP takes around 10s on average. UDP optimal is not considered since the default timeout value in all other experiments was chosen according to the optimal value of timeout under delay rule.

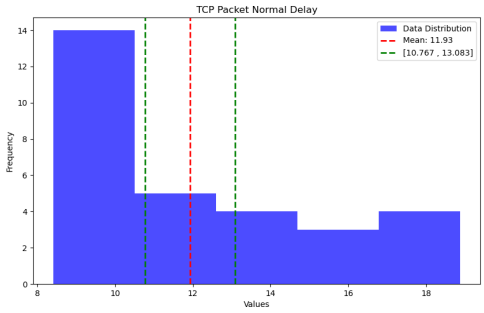
2.5.2 Confidence Intervals

The plots for the 95% confidence interval are as below:

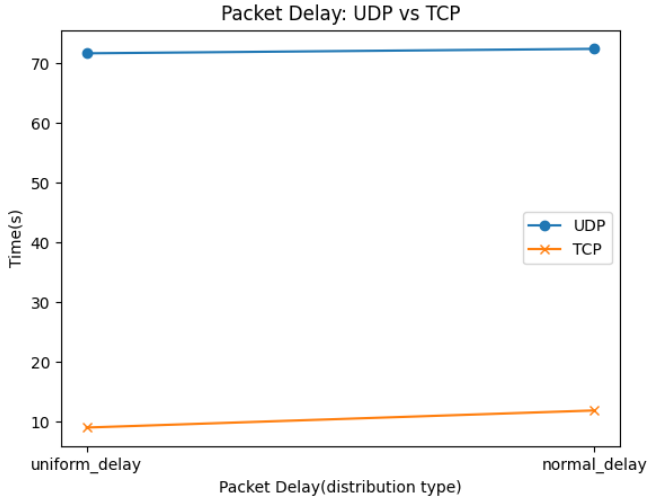








2.5.3 Total Download Time Over Delay Distribution



Retransmission counts for UDP in one of the runs for delay with uniform distribution:

large-0	small-0	large-1	small-1	large-2
21	0	22	0	23
small-2	large-3	small-3	large-4 of	small-4
0	80	0	24	0
large-5	small-5	large-6	small-6	large-7
24	0	22	0	77
small-7	large-8	small-8	large-9	small-9
0	30	0	29	0

Retransmission counts for UDP in one of the runs for delay with normal distribution:

large-0	small-0	large-1	small-1	large-2
23	0	21	0	22
small-2	large-3	small-3	large-4 of	small-4
0	68	0	28	0
large-5	small-5	large-6	small-6	large-7
22	0	23	0	71
small-7	large-8	small-8	large-9	small-9
0	22	0	22	0

2.5.4 Discussion

Between al tc/netem rules, 100ms delay was the one that decreases our RDT over UDP implementation's performance the most as the RTT's for each packet was extremely different. When the timeout held low to increase the efficieny for packets with low RTT, this caused the packets with delay to be recognized as lost packets and resulted in sender retransmitting the delayed packets and increased the total download time. When the timeout was held high enough to correctly recognize the delayed packets and not resend them, the few lost and corrupted packets in the network which would have low RTT's if transmitted, required more time to be recognized as lost packets on the server side and also increased the total download time. We tried to balance between those 2 situations with choosing a timeout of 0.13s. However, even with this experimentally optimal value, the performance of our implementation dropped significantly from 7s on benchmark to 70s.

TCP's performance dropped from 0.4s on benchmark to 10s. The change in the total download time is 25x which is much higher than our RDT over UDP's 10x. So, we can say even if TCP performs better under delay conditions in terms of total download time, it is more sensitive to delay than RDT over UDP. Again, as it was in the loss and corruption case, the reason for this is the TCP's congestion control mechanism. It slows down the rate of data transfer when it detects that the packets are delayed. However, the timeouts for different packets can be set dynamically with respect to delay, hence it performs better than our implementation in which the timeout is set as the same for every packet. We did not implement but our RDT over UDP performance can be improved by utilizing threading for timeouts.

Normal Distribution produces delays that follow a Gaussian distribution and values are centered around a mean value, with some deviation. On the other hand, Uniform Distribution generates delays uniformly across a specified range. Since normal distribution introduces more randomness and variability compared to a uniform distribution, both UDP and TCP performs slightly worse compared to uniform distribution. Also, it can be seen that standard deviation in the results for UDP with uniform delay is significantly less and our results are more stable throughout the runs.

3 Conclusion

During this assignment, we have implemented file transfer programs with TCP and UDP talking across an unreliable channel. During the coding process, since TCP is a reliable protocol, we did not have to worry about packet corruption or loss. The crucial part of the assignment was implementing Selective Repeat Protocol on unreliable protocol UDP. We prevented packet loss and corruption while trying to optimize the runtime performance. While trying to optimize, we had seen the effects of packet size, window size and timeout values on packet transfer.

During the experiment process, we have tried various tc/netem rules on the file transfer programs we have implemented. By measuring the performance on those different rules such as loss, corruption, duplication and delay, we have seen the outcomes of different problems that can occur during the packet transfer on the network. We have compared TCP and RDT over UDP in terms of these parameters and discussed the reasons behind the performance differences.

Our objective was to outperform TCP with our implementation and we have achieved this only when the loss and corruption was high. Hence, we can conclude while TCP is complex and is a better choice for stable networks where the advanced mechanisms like flow and congestion control can be utilized, our less complex RDT over UDP implementation might be a better choice for unstable and unreliable networks where many packet losses and corruptions occur.