COL334 Assignment 3 Report
Part 1

Mrunal Kadhane - 2021CS10109
Kanishka Gajbhiye - 2021CS50131

1 AIM

The code creates a socket to receive data from a UDP server. The server internally samples from a uniformly random distribution and provides a constant packet loss rate. The server also implements a leaky bucket filter to force clients to not overwhelm the server with continuous send requests by temporarily reducing the rate of token generation and hence reducing the client's receiving rate. The packets received can get reordered on the network. Our goal was to receive data reliably from this server.

2 ALGORITHM

We are initially sending a 'SendSize' command to set the upper limit for the bytes we need to receive. The code implements **three parallel threads -**

1. sending offset requests to server
2. receiving data from the server
3. Filling the gaps in case of packet drops

2.1 Thread 1 : Send-handler

This thread continuously sends **offset requests** to the server without worrying about packet drops or anything. To prevent penalties we included a sleep time of **0.01 s** after each request. Offset requests are requests demanding a certain amount of bytes from a certain offset. Here we have kept our bytes as **1448** which is the maximum amount that can be transferred. We increment our offset after every request and exit after the offset crosses the maximum size expected to be received. This thread only focuses on sending data requests to server and reading replies is handled by receive- thread to increase efficiency.

2.2 Thread 2 : Receive-handler

This thread **accepts data from the server** and adds it into a dictionary for every new offset values as keys. It might not receive offsets in order because a packet might get reordered or a lost packet was asked again to the server and received later.

2.3 Thread 1 : Fill gaps-handler

We have implemented such that the key values in the dictionary will be multiples of 1448 (the numbytes set for the implementation). This thread goes through the dictionary and **checks for gaps** between the key values and then sends offset requests to the server. A sleeptime of **0.01 s** is added here as well between send requests. After going through the whole dictionary, if **len(dictionary)*1448 is less than the totalsize**, we again repeat the same process until the above condition is satisfied.

Once we have received the complete data, we assemble the file in increasing order of the offset values and generate an **MD5 hash** and submit it to the server to check for correctness. We were able to receive, compile and submit the data reliably in around **8 - 11 secs**.

3 Graphs and Analysis

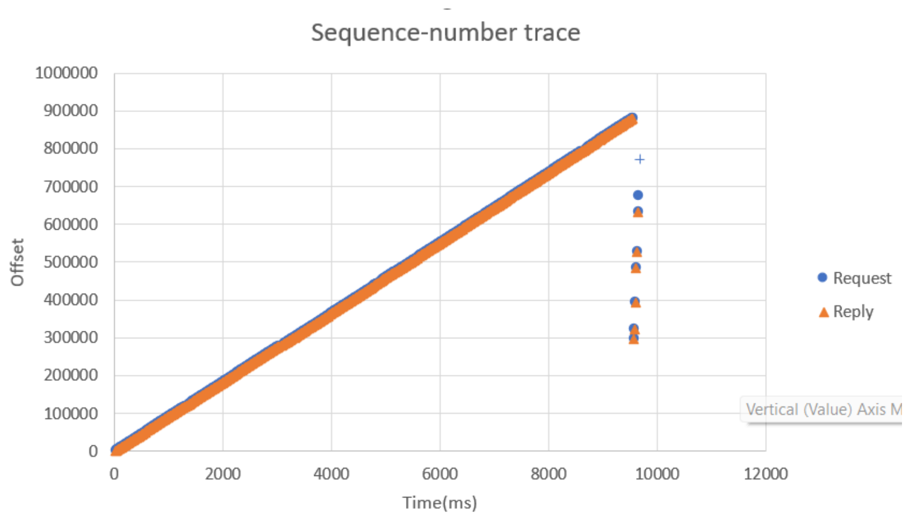

Figure 1: Sequence number trace

Above is the plot showing sequence trace of requests sent(blue) and replies received(orange). To a large extent, offset increases linearly with time for both offset requests sent and offset replies received. (indicated by a straight line).

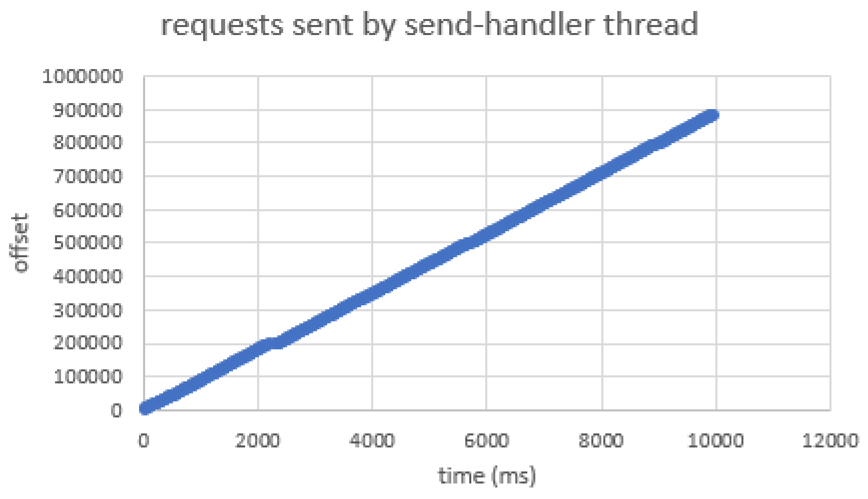

Figure 2: Sequence number trace of requests sent by main thread(send-handler)

For the send-handler thread, offsets only increases linearly with time since we are only incrementing it

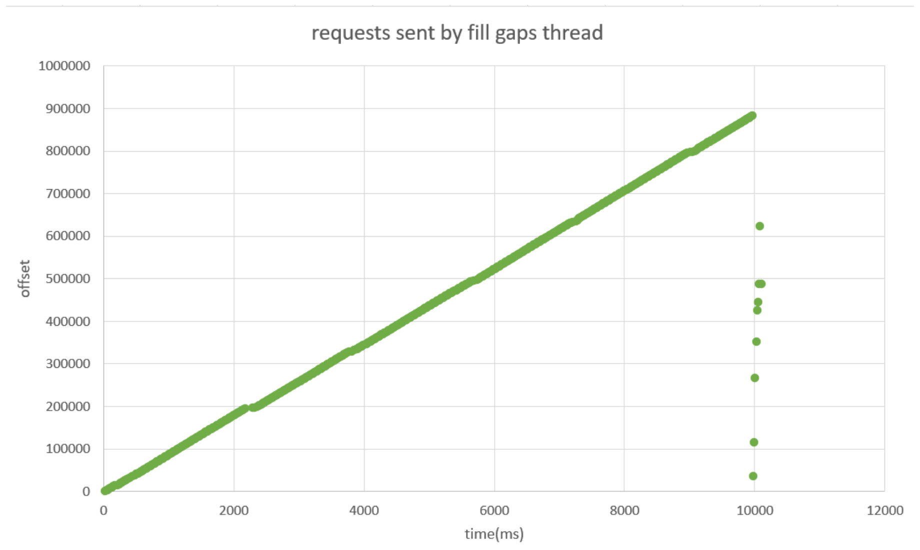

Figure 3: Sequence number trace of requests sent by fill gaps thread

The fill gaps thread also increases linearly for most part but since it iterates through the dictionary multiple times, it again requests for packets from a lower offset

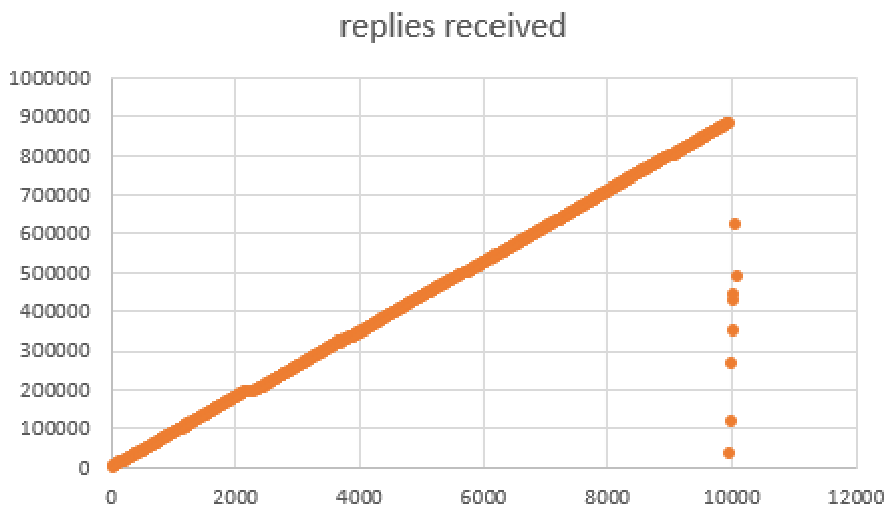

Figure 4: Sequence number trace of requests received by receiver thread

For the receive-handler thread, offsets increases linearly with time for most part can move down because of the fill gaps thread sending requests for lost packets

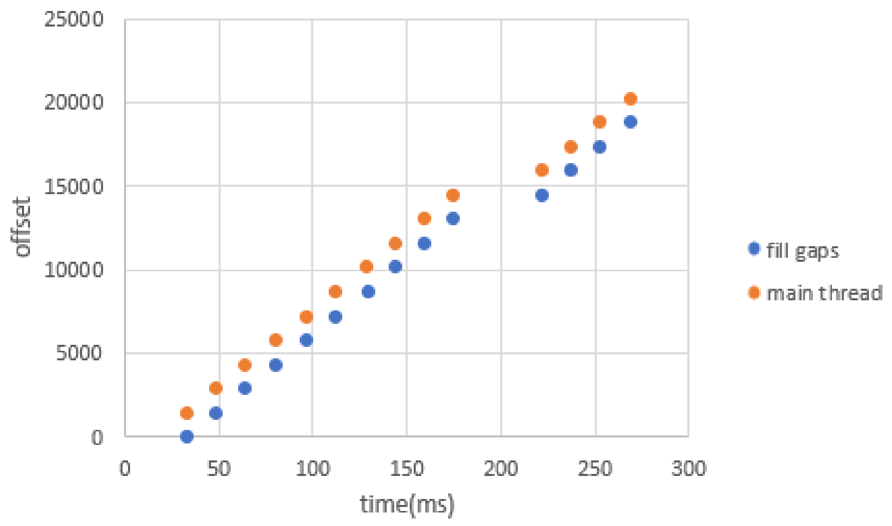

Figure 5: Sequence number trace of requests sent by fill gaps thread(blue) and main thread(orange) (Zoomed in graph)

The plot above is to compare the performance of our two sent threads for the first few requests. As we can see the fill gaps thread operates after the send thread to make up for its losses

Analysis on text file provided to test our code:
No. of lines = 10000
No. of requests sent by main thread = 610
No. of replies received by receiver thread = 610
Avg. time taken = 8 s