Assignment 3 : Receiving data reliably

Rajat Golechha, Satyam Kumar

October 15, 2023

1 What We Did

In this milestone of the assignment, we implemented reliable data transfer using the UDP protocol via two threads: one that receives the messages and another that sends the messages according to the protocol.

1.1 How We Did It

First, we requested the server for the size of the file to be obtained. Based on the file size, we created a list of requests that needed to be sent to the server over a period of time. We then initiated the receiving and sending threads.

1.1.1 Receiving Thread

The receiving thread continuously concatenated the received messages into a buffer. We parsed the string for complete datagrams. When a complete datagram was found, we removed that part from the string and stored it in a dictionary. Simultaneously, we removed the corresponding element from the request list.

```
    while True:
        with reqs_lock:
            # print(len(reqs))
            if len(reqs) == 0:
                break
        try:
            recv = client.recvfrom(bufferSize)
            recv = recv[0].decode()
            all_data += recv
            while all_data.count('Offset')>0:
                x = all_data.find('Offset')
                y = all_data.find('NumBytes')
                offset = int(all_data[x+8:y-1])
                m1 = all_data[y+9::]
                z = m1.find('\n')
                num_bytes = int(m1[0:z])
                if(z+2+num_bytes > len(m1)):
                    break
                m2 = m1[z+2:z+2+num_bytes]
                with datadict_lock:
                    datadict[offset] = m2
                    all_data = m1[z+2+num_bytes::]
                element = [offset, num_bytes]
                with reqs_lock:
                    if element in reqs:
```

```
                        offset_time[offset] = time.time() - start_time
                        reqs.remove(element)

        except socket.timeout:
            continue
```

1.1.2 Sending Thread

In the sending thread, we sequentially requested messages from the server based on the contents of the request list. This list was dynamically updated based on incoming messages.

```
    i = 0
    while True:
        with reqs_lock:
            if len(reqs) == 0:
                break
        with reqs_lock:
            x = i % len(reqs)
            i = i + 1
            print(x)
            req_msg = "Offset: "+str(reqs[x][0])+"\n"+"NumBytes: "+str(
reqs[x][1])+"\n\n"
            M.append((time.time()-start_time,reqs[x][0]))
        client.sendto(req_msg.encode(),serverAddressPort)
        time.sleep(0.01)
```

Once all the messages in the request list were received, we concatenated them and generated an MD5 hash for the entire message. This hash was then submitted to the server.

2 Graphs

On analysing the data and converting them to graphs, we obtain the following plots :

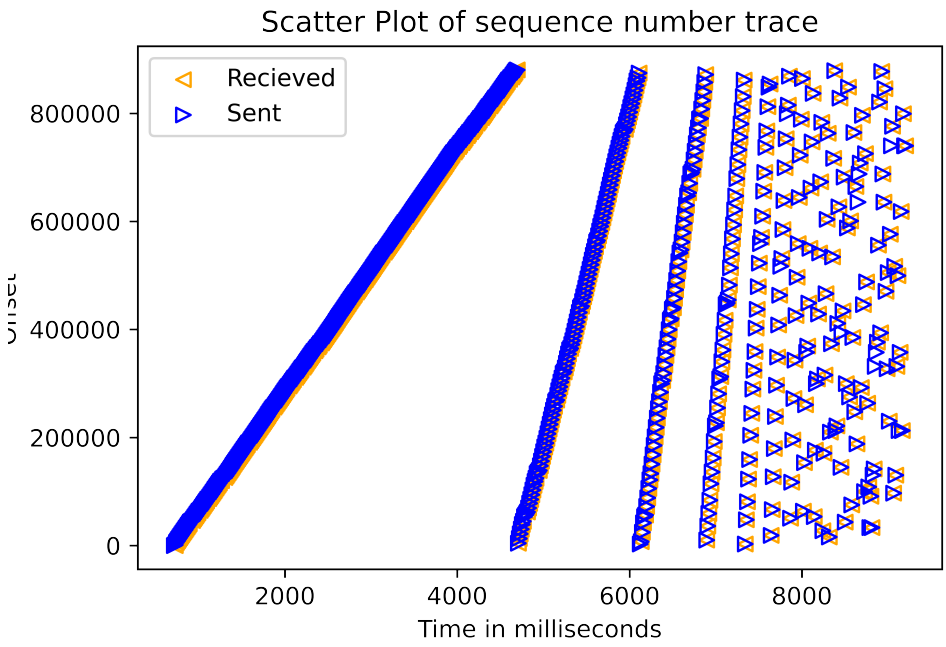

Figure 1: Offset-Time plot

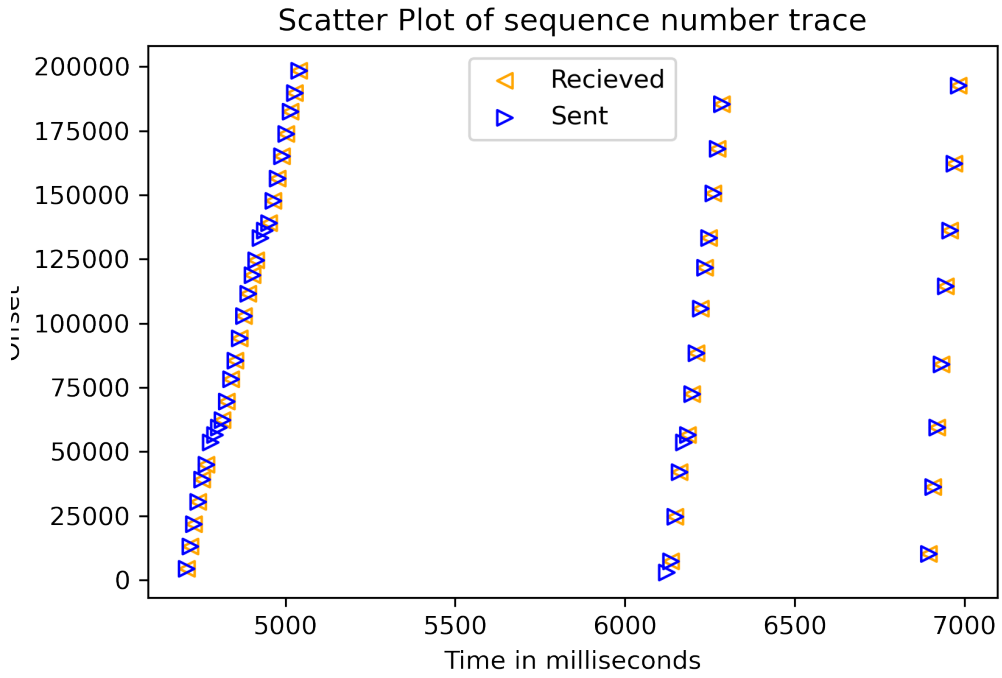

Figure 2: Zoomed in Offset-Time plot

3 Graph Analysis

- In the plots, the blue mark represents the sending of a request for a packet, and the orange mark represents the receiving of a packet by the receive thread.
- In the first graph, we can observe multiple straight lines, each representing different iterations of requests. Every subsequent line corresponds to packets that were either dropped, skipped, or lost in the previous iterations.
- We also observe in the second zoomed in graph how for some points there is a sent request but no data received, these points denote the packet loss in the UDP transfer.
- To ensure the reliability of UDP transfer, multiple iterations are performed. This iterative process guarantees that every packet is received, addressing any potential issues associated with packet loss.