CS2031 - Telecommunications II

Assignment 1 Report

Client, Server, Gateway Communication

Introduction

Throughout this report I will discuss and explain the design and implementation of the protocol as described by the CS2031 Assignment 1. I will initially explain the design and implementation of the protocol. I will also explore the various obstacles and issues that I encountered throughout the assignment as well as various advantages and disadvantages associated with the approaches I decided to take.

Design and Implementation

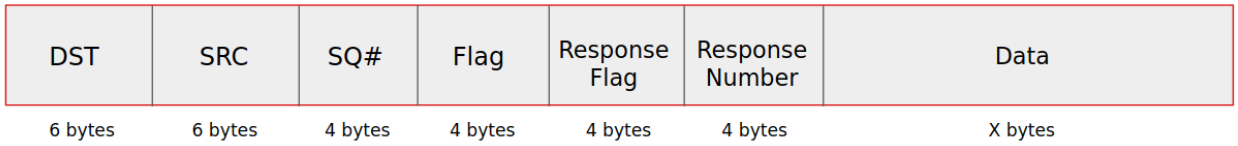
On the overall, the program I designed is simply composed of three main classes:

- Client
- Gateway
- Server

Within these classes I have designed and implemented various methods and protocols in order to achieve the desired output of having multiple clients that can communicate with a server via a gateway.

The program begins by asking the user to define the port on which they desire the client socket to be established on. A Client is then created on *localhost* at the disclosed port number.

The client then reads the desired data to be transmitted to the server as well as the port address of the server from the user via the terminal. It then encapsulates this data. The design of the data encapsulation protocol is as follows:



DST - Destination Address (Server)

SRC - Source Address (Client)

SQ# - Sequence Number of packet

Flag - Directional flag, (0 = from Client, 1 = from Server)

Response Flag - Response Flag (0 = NAK, 1 = ACK)

Response Number - Sequence Number of packet server expects next

Data - Byte array containing user disclosed data to be sent to server

Initial Client to Gateway Transmission

All of the above data is enclosed within a header buffer and then combined and encapsulated with the user defined data within the main buffer. A datagram packet is then created from this buffer and sent to the gateway via the socket created initially.

If the Client does not receive a response from the server within the defined time of *TIMER_VALUE*, then a SocketTimeoutException is thrown. This exception causes a method called *resendPacket()* to be called upon and the packet to be re-sent to the Server.

Data Handling at Gateway I

When a packet is received at the socket established at the Gateway an interruption is thrown which activates the onReceipt() method. It is here that the packet is processed and handled accordingly depending on whether it needs to be sent to the client or server.

The flag is extracted from the header and analysed.

Case 1 (Flag == 0) - If the flag is found to contain the value 0 this means that the packet has came from the client and needs to be sent on to the server. The destination address (*DST*) is then extracted from the header and the packet is sent to this port via the gateway socket.

Case 2 (Flag == 1) - If the flag is found to contain the value 1 this means that the packet has came from the server and needs to be routed back to the client. The source address (*SRC*) is then extracted from the header and the packet is sent to the client at the source address via the gateway socket.

Case 2 (Flag != 1 && Flag != 0) - If the flag is found to contain a value other than 0 or 1 an exception is then thrown and an error message printed to the terminal.

Data Handling at Server

When a packet is received at server socket, similar to the gateway, an interruption is thrown. This causes the onReceipt() method to be activated. The packet is first converted to a *StringContent* object in order for the encapsulated data to be processed and analysed.

The server begins by checking if the sequence number of the packet is the sequence number that the server had been expecting (initialised at 0). This can result in two outcomes:

Case 1 (sequenceNumber == expectedSequenceNumber) - In this case the sequence number of the packet matches that of the sequence number that the server had been expecting. This results in the server sending back a positive response ACK. This is done by setting the *Response Flag =1* within the header. The expected sequence number at the server is then incremented.

Case 2 (sequenceNumber != expectedSequenceNumber) - In this case the sequence number of the packet does not match that of the sequence number that the server had been expecting. This results in the server sending back a negative response NAK. This is done by setting the *Response Flag =0* within the header. The expected sequence number at the server remains the same.

The direction flag (*Flag)* is then changed to 1 to indicate that the new packet is coming from the server. This newly updated packet is then sent once again back to the gateway for it to be processed.

Data Handling at Gateway II

When the response packet is received at the gateway it is processed as discussed above in *Data Handling at Gateway I* and the flag is extracted, cross-checked and forwarded to the client or server accordingly. In this case, the packet is sent to the socket at the port address of the client.

Data Handling at Client

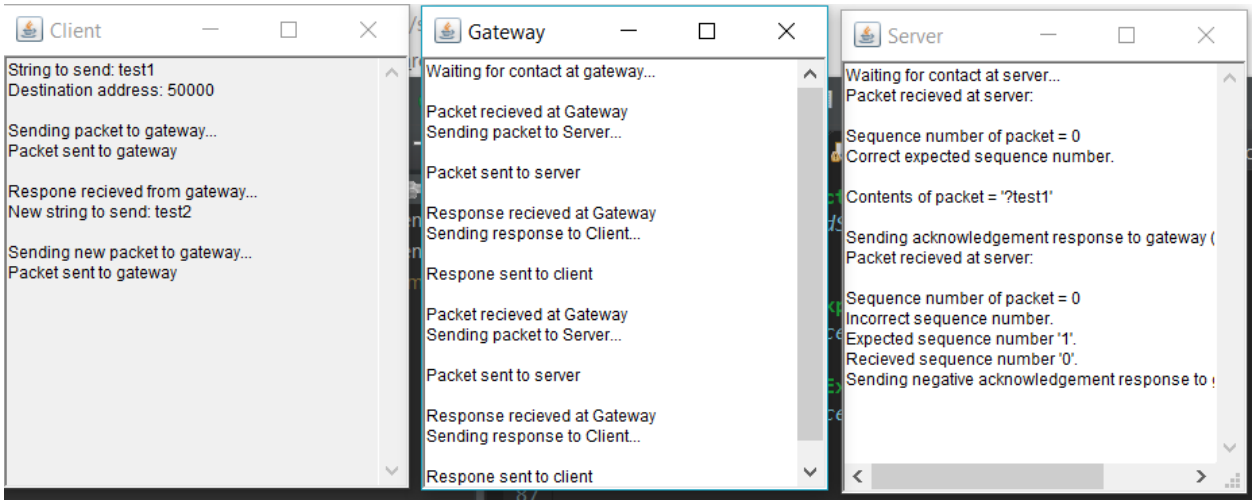
When the client itself receives a packet at its socket it causes an interruption to be thrown and once again the onReceipt() method is called upon.

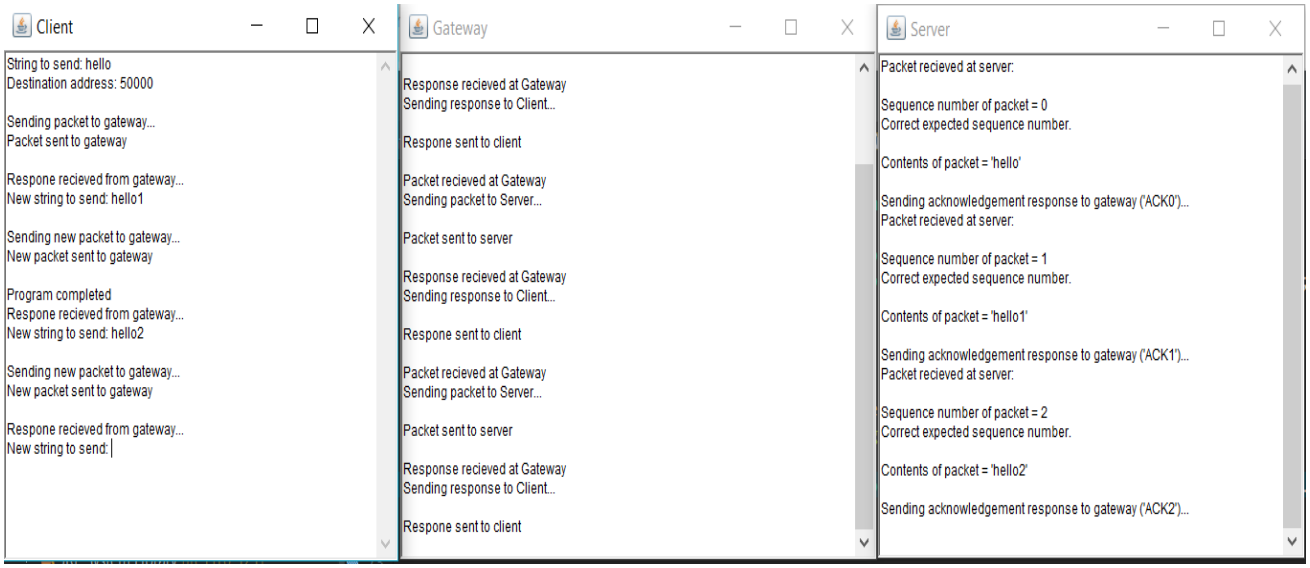
This time the client extracts the *Response Flag* from the packet and analyses it to see whether or not the packet contains an ACK or an NAK.

Case 1 (Response Flag == 1) - In this case the response flag of the packet indicates that the server has correctly received and acknowledged the packet successfully. This causes a carryOn() method to be called which allows the user to continue transmitting messages between client and server for as long as they desire.

Case 2 (Response Flag == 0) - In this case the response flag of the packet indicates that the server has incorrectly received and failed to acknowledge the packet successfully. This causes a resendPacket() method to be called which causes the packet to be re-transmitted to the server in an attempt to receive a positive transmission response ACK.

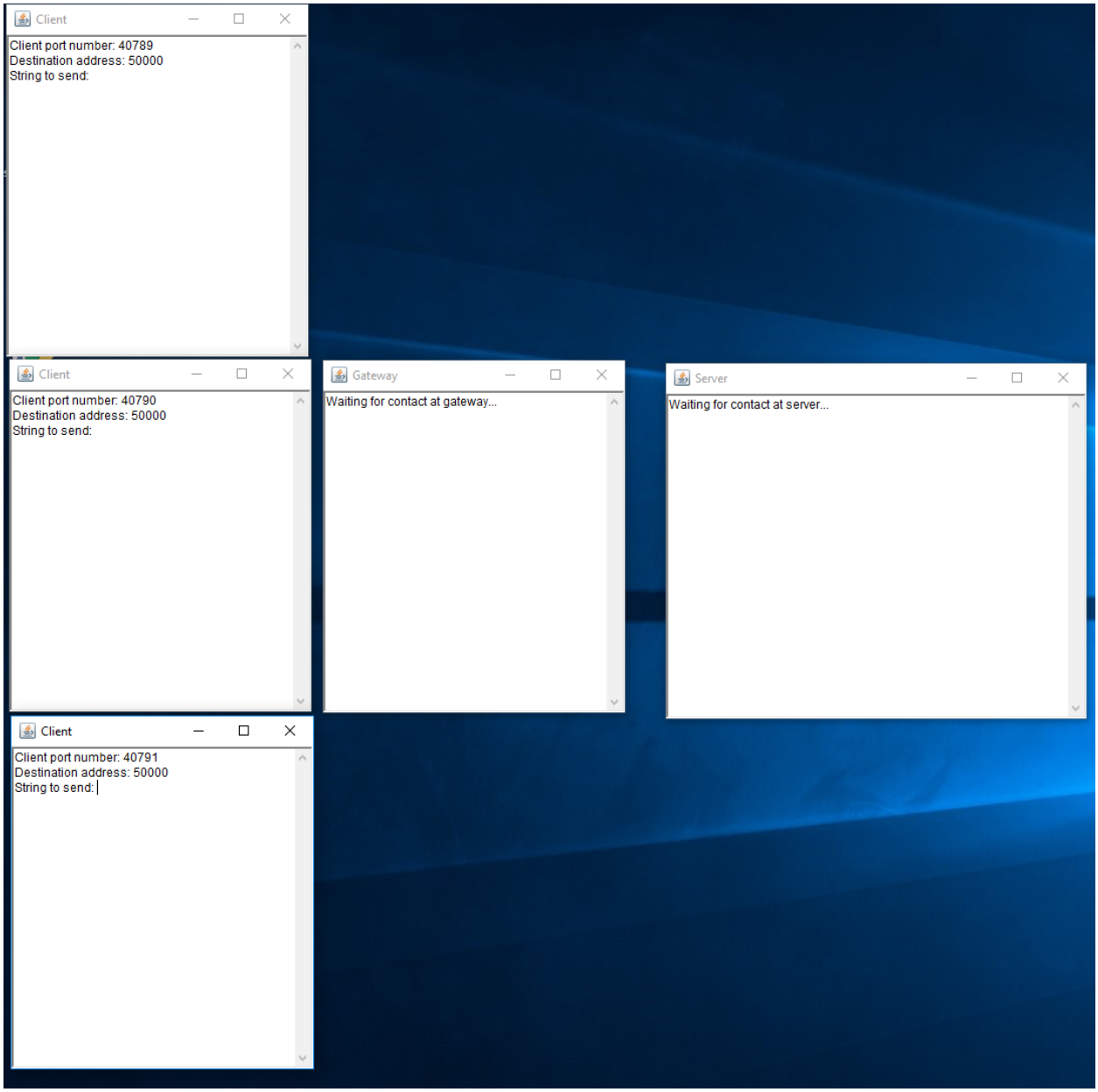
Screen Grabs (Early Versions)





Handling Multiple Clients

In order to allow the system to handle multiple clients I forced the user to define the source port as well as the destination port at the beginning of the program. This allows for a socket to be created at the given port and allow for further client instances to be created under various ports. (*See image below)*



This worked successfully in the beginning, however under further examination I realised that the server was unable to determine correctly the expected sequence number of the packets as it was receiving various packets from multiple sources containing various sequence numbers.

After further research I stumbled across a method that allowed me to solve this by using Hashmaps. When a transmission is first received at the server from a client, i.e *Sequence Number = 0*, a key and value pair are created in the local hashmap stored at the server for the new client in the form:

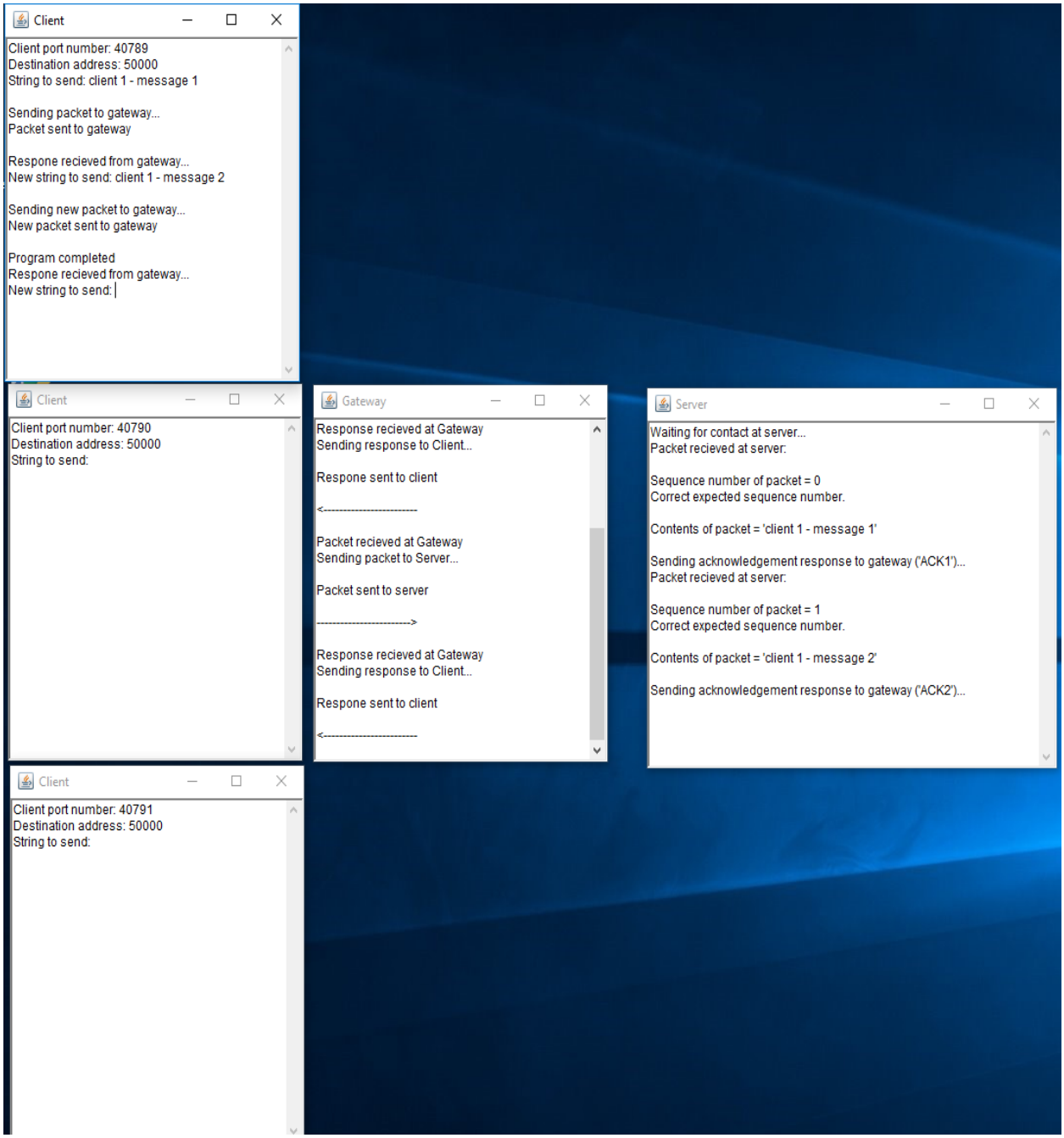
```
HashMap<Integer, Integer> hmap = new HashMap<Integer, Integer>();

hmap.put(SourceAddress, 0);
```

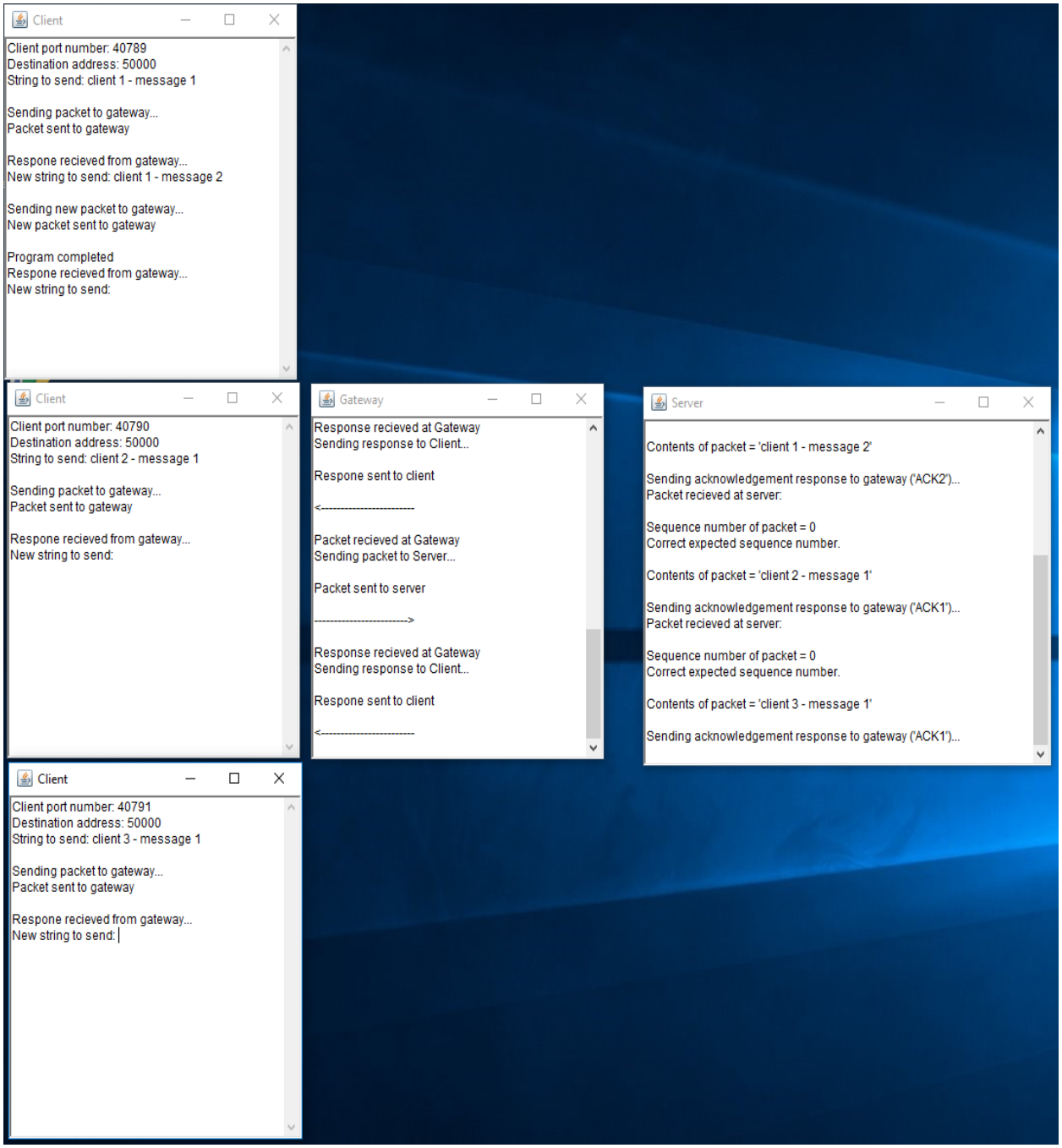
This enabled the server to store and process expected sequence number values for the various clients that would be in contact with it at any given time. The server checks to see if sequence number of the packet matches that of the expected sequence number stored in the hashmap for that client's address. If the values are the same, the correct packet has been received and the expected sequence number in the hashmap is updated. This is followed by an ACK being sent back to the client containing the sequence number the server expects next for this client.

The images below outline a communication between three various clients operating on different ports through the one gateway, communicating with the one server.

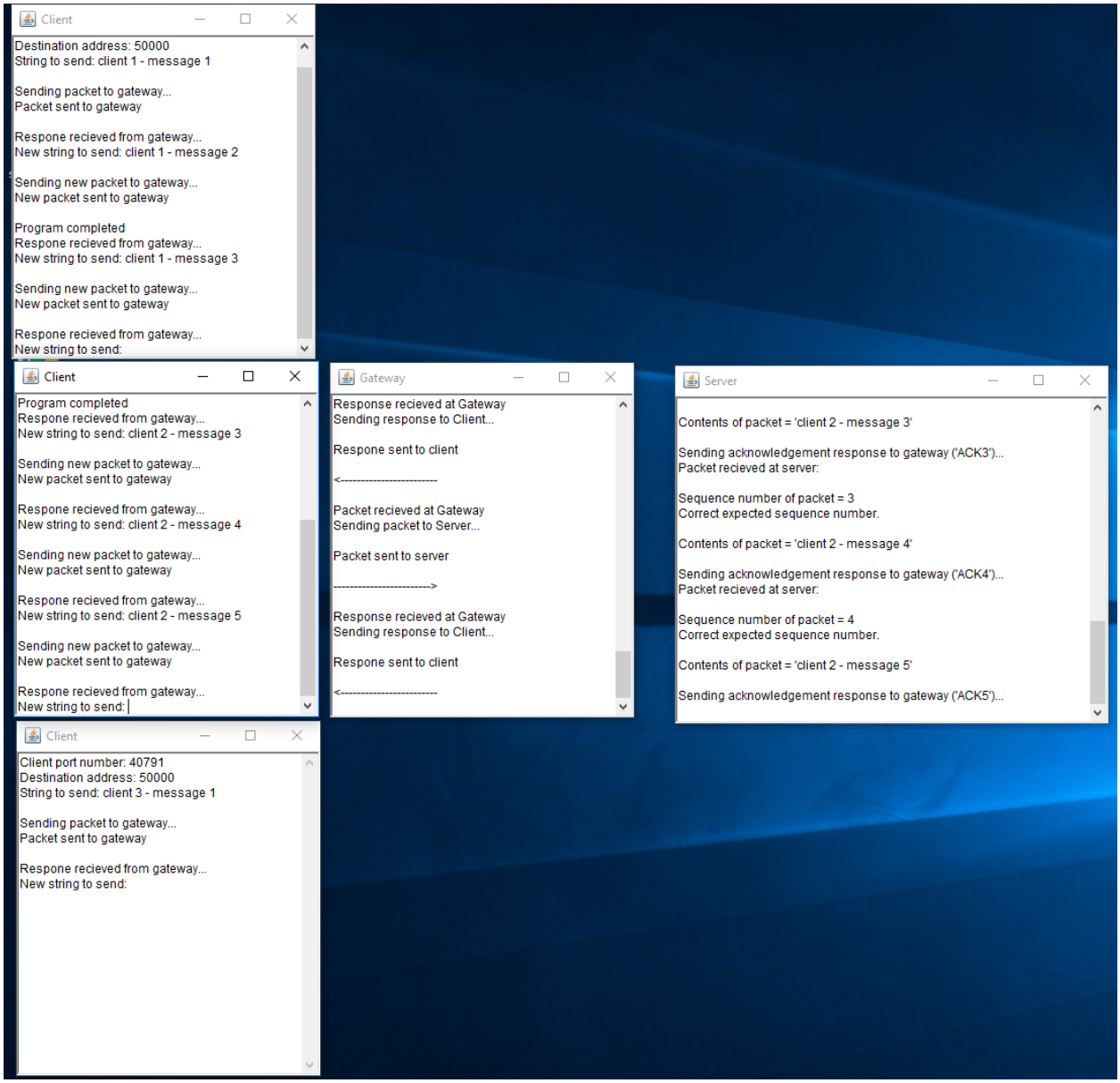
1. Client 1 (*Port: 40789*) communicates with the Server (*Port: 50000*) via the Gateway and successfully sends two messages and receives two ACK's from the server.



2. Client 2 (*Port: 40790*) then sends its first communication to the Server via the Gateway and successfully receives an ACK showing successful transmission. Similarly, Client 3 (*Port: 40791*) then also sends its first transmission to the Server. It too receives an ACK.



3. Clients 1 and 2 then proceed with their individual transmissions at various intervals, all of which are successfully processed and handled at the Server. All transmissions receive an ACK showing Server can now handle multiple clients.



Conclusion and Reflection

Overall, I found this assignment to be extremely beneficial in developing my understanding of how telecommunication networks work with regards to a simple Client -> Gateway -> Server communication.

At first the assignment was a bit daunting as we were just beginning to work our way through the content within the course lectures and I was unsure as to what was really going on. However, after further study and reading of both the course material and independent material found online i began to get my head around the system as a whole.

I encountered many challenges and obstacles throughout this assignment, many of which made me want to delete my entire code and start fresh. Fortunately, once the first packet managed to reach the Gateway and the correct data was displayed on the terminal I regained my confidence.

One of the first issues I had was that I was setting the Datagram packet's address to that of the Client port (e.g 40789) which lead to various errors and no successful transmission. It was only after reading into the documentation related to the Java Datagram class that I discovered it was in fact the port of the packet that needed to be set.

Getting the packet to reach the Server was the first hurdle. Getting a response packet back to the Client was another situation altogether. This lead me to designing the encapsulation protocol as described above which, after large amounts of debugging, frustration and trial and error, finally allowed me to get packets to go from Client -> Server -> Client (extremely rewarding and satisfactory).

If I was to do anything differently as a whole I would probably attempt to replicate an industry standard data encapsulation protocol e.g HDLC. If I had more time to work on the assignment I would also implement some error handling within the actual execution of the system itself i.e handling incorrect user input. However, on the whole i feel like I have implemented the system to the best of my ability and am extremely happy with the end result.

Brandon Dooley - *16327446*