Telecommunications Assignment 1

Name: Marek Betka
Student Number: 16324334

Introduction:

The goal of this implementation was to have multiple clients that can communicate to a gateway which would let them forward their data to a server. The server would then send an acknowledgement of the packet if it was the one it expected, and if not it would send a negative acknowledgement to help the client correct itself and send the appropriate packet. I chose to do this assignment in python 3 instead of java, as I find it better to work in even though a basic program was provided in the java implementation.

Setup

Since I was working in python, I had to make the server and client myself. This proved not too challenging as python comes with a module called "socket", which was used to make this work. I decided to use a TCP protocol which meant a connection had to be made between the client and gateway, and gateway and server. TCP is usually used for packet transfers in which, reliability is more important than the transmission time. This seems to apply better to what kind of data we would be sending should be sent reliably from one "Internet of things" sensor to the server.

At first, I made a basic client and server to figure out how to use the socket module to let programs communicate between each other. This first implementation was extremely limited as there could only be one client and one server, and there was no timeout for if no messages were being received. This meant if somehow the client sent a packet which got lost in transmission somehow, the server would be waiting to receive that packet indefinitely.

Handling multiple clients with threads

After making this most basic implementation, I decided to work on having multiple clients being able to communicate to the server. The way this was done was by using threads. The server code was changed so that it waits for a connection, after finding one it runs a method as a new thread, forwarding the connection to that method.

The client code had to be changed too to only assign a port to the client that was not being used by another (i.e. Two clients whose ports are both 5555 could not connect to the server). This worked relatively well, however if a client was closed, the server would keep the thread alive until the whole program was closed.

This was solved by using "try" and "except" statements, now if the server or client had a connection failure, instead of the program crashing or running indefinitely, the thread would be closed.

Gateway

The next thing that was worked on was the gateway, the client was modified to allow the user to connect to a server directly, or this gateway by default. The gateway was made so that it had a list of known servers that the client could connect to, this was done this way to let new, future servers be added to the gateway if needed. One way this could be improved upon was to let the client know the list of servers the gateway has, however this modification was never made.

The code for the gateway used a lot of code from both the client and server, as it basically must function as both. It waits for a connection to be made, before running a method as a new thread with that function in the same way the server did. After this however, the gateway waits for the client to tell it the name of the server it wants to connect to. If it receives this data, it looks through a list of servers to find a match. If a match is found, the gateway attempts to establish a connection to that server. However, if a match is not found, the gateway sends out a packet back to the client to tell it that it could not find a match for the given input and proceeds to close the connection and the thread.

Once connections with the gateway have been established, the gateway repeatedly waits for a message from the client, then server until a disconnect occurs. At this point there were some issues with the gateway as to who should be sending data, and who should be receiving data. It was clear that at this point in the implantation some kind of time out was needed to sync up the client and server.

A timeout exception was added to the gateway, which let the server and client communicate through the gateway, instead of directly. Some adjustments needed to be made however to the time out exception, namely if it timed out a certain number of times it would close the thread and assume that either the client or server disconnected unexpectedly (i.e. If the gateway couldn't reach the client and timed out 10 times, the connection and thread would be closed).

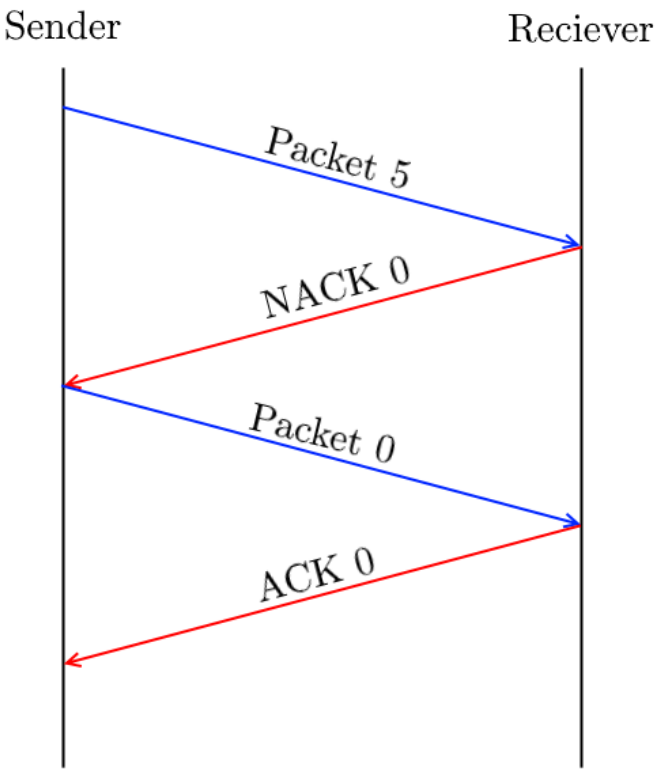
One flaw that could have been improved upon was the fact that there was no expected way to disconnect, and when the gateway closed the connection because of 10 time outs, it would only do so on its end. This means the server would still be trying to communicate to the gateway, and when it too, timed out a number of times would close its connection and thread. Although this seems like a simple fix, it would

require a special packet to be sent from the gateway to the server/client letting it know the connection is closed. This special packet would also need to not be spoofed by the client so the modification was never made.

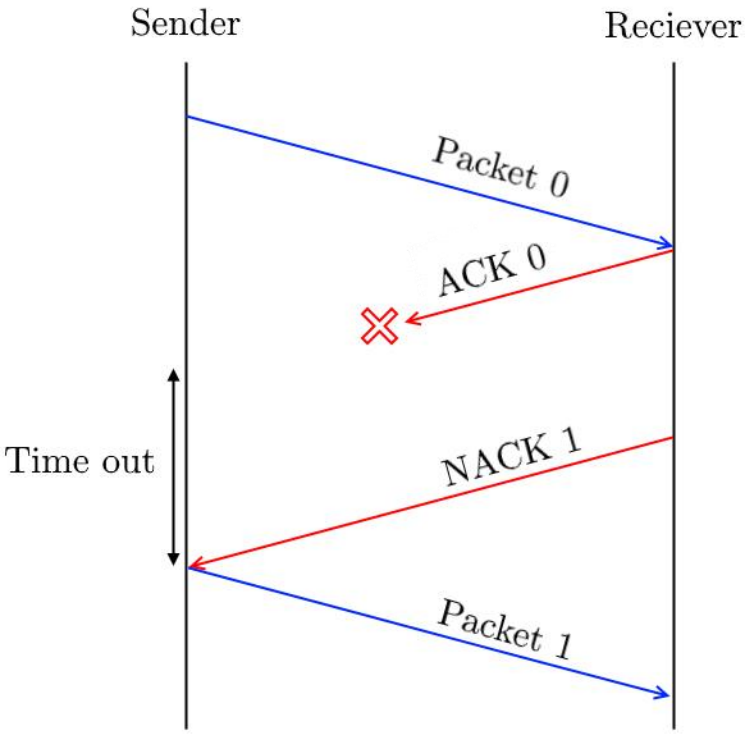
Packet sequencing

Finally that all the connections were working well, the implementation for sequencing the packets could be done. The packets that were sent were just the sequence number for that packet, however I did add functionality to use a delimiter to split the data from the sequence number. This functionality was never used but it is available for use.

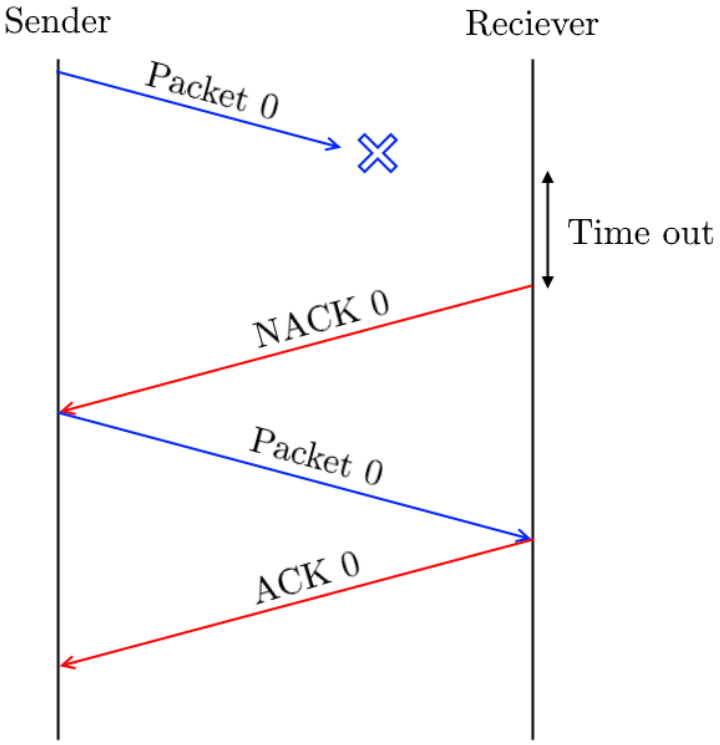
After the client receives confirmation it has connected to the server, it sends its first packet, then the server acknowledges it received that packet or tells the client which sequence number it wanted instead. The client then sends the packet of sequence number that the server wanted. This can be seen in the following diagram:



If the client sends a packet to the server, and the server sends a packet back which gets lost in transmission, client will wait for a NACK of which packet it wants. It was chosen that the client should wait for the server as usually the connection issue is with the client not the server, and the client doesn't know if it failed to get its packet, or it just failed to get an acknowledgement. This can be seen in the following diagram:



If a client sends a packet to the server and the packet gets lost in transmission, meaning the server did not receive it. The server will send out a negative acknowledgement, letting the client know that it expected a packet, but it did not arrive, letting the client also know which sequence number it expected. This can be seen in the following diagram:



It should be noted, all these diagrams are a simplification of the actual process that goes on, considering that the gateway is also involved in this. However, for understanding how the packets get sent and correct themselves using stop and wait these diagrams are far more practical. Lastly a small piece of code was implemented on the server and client to let the packets fail every so often (Around 1/5 chance to fail). This was to demonstrate the system functioning. Now here are some actual examples of my client, gateway and server sending/receiving data to/from each other.

Examples of transmitted packets

Note: For clarity sake the client is the one with blue text, the server with red text, and the gateway with green text.

This example shows the client connecting to the server through the gateway, and sending the a packet with sequence number 5 when server expected one with 0.

```
Type in your ip:port or press enter to connect to gateway:
Which server would you like to connect to? Type 'main' for main server: main
Recieved: Connected!
Sending: 5
Time out: Server did not respond
Recieved: NACK:0
Resending: 0
Recieved: ACK:0
Sending: 1
Recieved: ACK:1
```

Here is an example of what the gateway sees when starting and connecting multiple clients to a server.

```
Listening...
Client 127.0.0.1:65160 has connected to the gateway!
Client has been connected to localhost:9999
Client 127.0.0.1:65163 has connected to the gateway!
Client has been connected to localhost:9999
Client 127.0.0.1:65166 has connected to the gateway!
Client has been connected to localhost:9999
```

Here is an example of the client failing to send a packet to the server from, from the clients perspective and the servers perspective:

```
Recieved: ACK:14
Sending: 15 (has failed)
Time out: Server did not respond
Recieved: NACK:15
Resending: 15
```

```
Sending: ACK:14
There was an error with the connection!
Sending: NACK:15
Recieved: 15
```

Here is an example of the server failing to send an acknowledgement, from the clients perspective and the servers perspective:

```
Sending: 8
Time out: Server did not respond
Recieved: NACK:9
Resending: 9
Recieved: ACK:9
```

```
Recieved: 8
Sending: ACK:8 (has failed)
There was an error with the connection!
Sending: NACK:9
Recieved: 9
Sending: ACK:9
```

Conclusion

In conclusion, this assignment was a really good practical way of learning and understanding sockets and helped me know the theory of sockets much better than I could have by just reading about them. The biggest difficulty I had was setting up and understanding the basic sockets for my first client/server system.

I probably could have improved the gateway to send a packet for the client or server to disconnect, as mentioned previously. I could have added the functionality for the client to request a list of servers from the gateway before attempting to connect to one.

In total, including the documentation, the assignment took me around 17 hours to complete. The biggest part of this was spent learning about sockets in python and also fine tuning everything at the end.