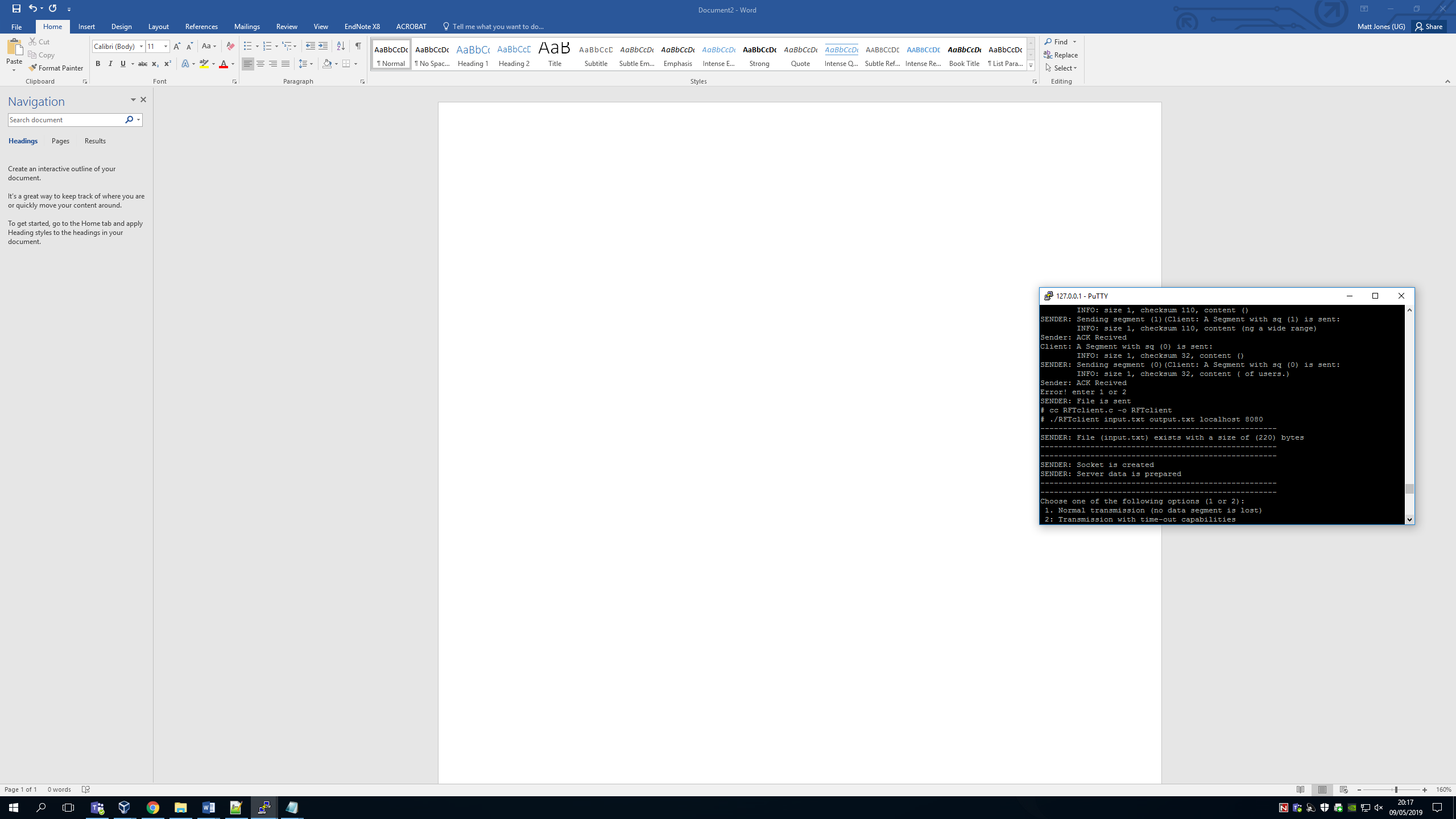
CSC2026 Networks Report

Parts completed:

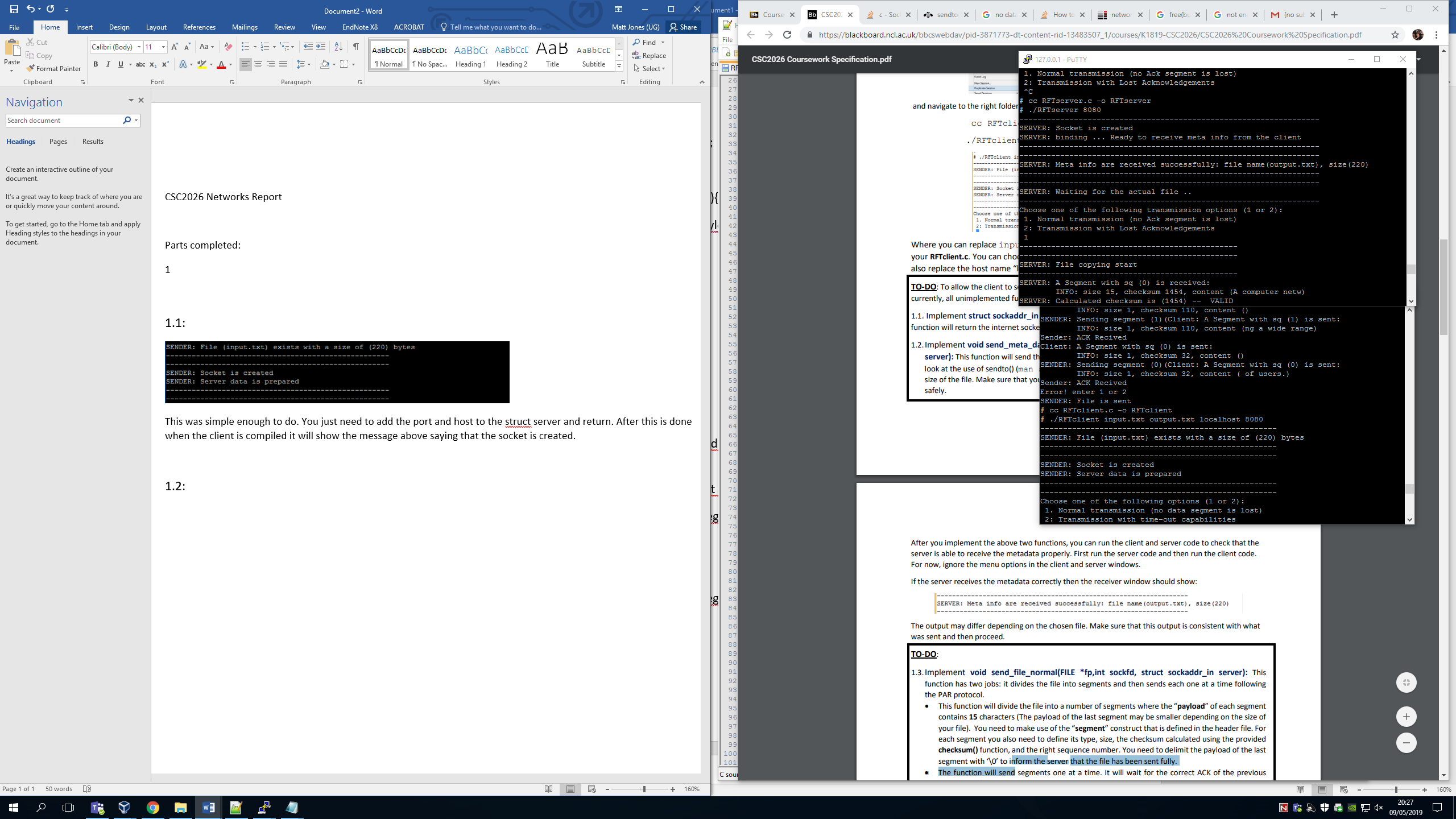
1,2

1.1:



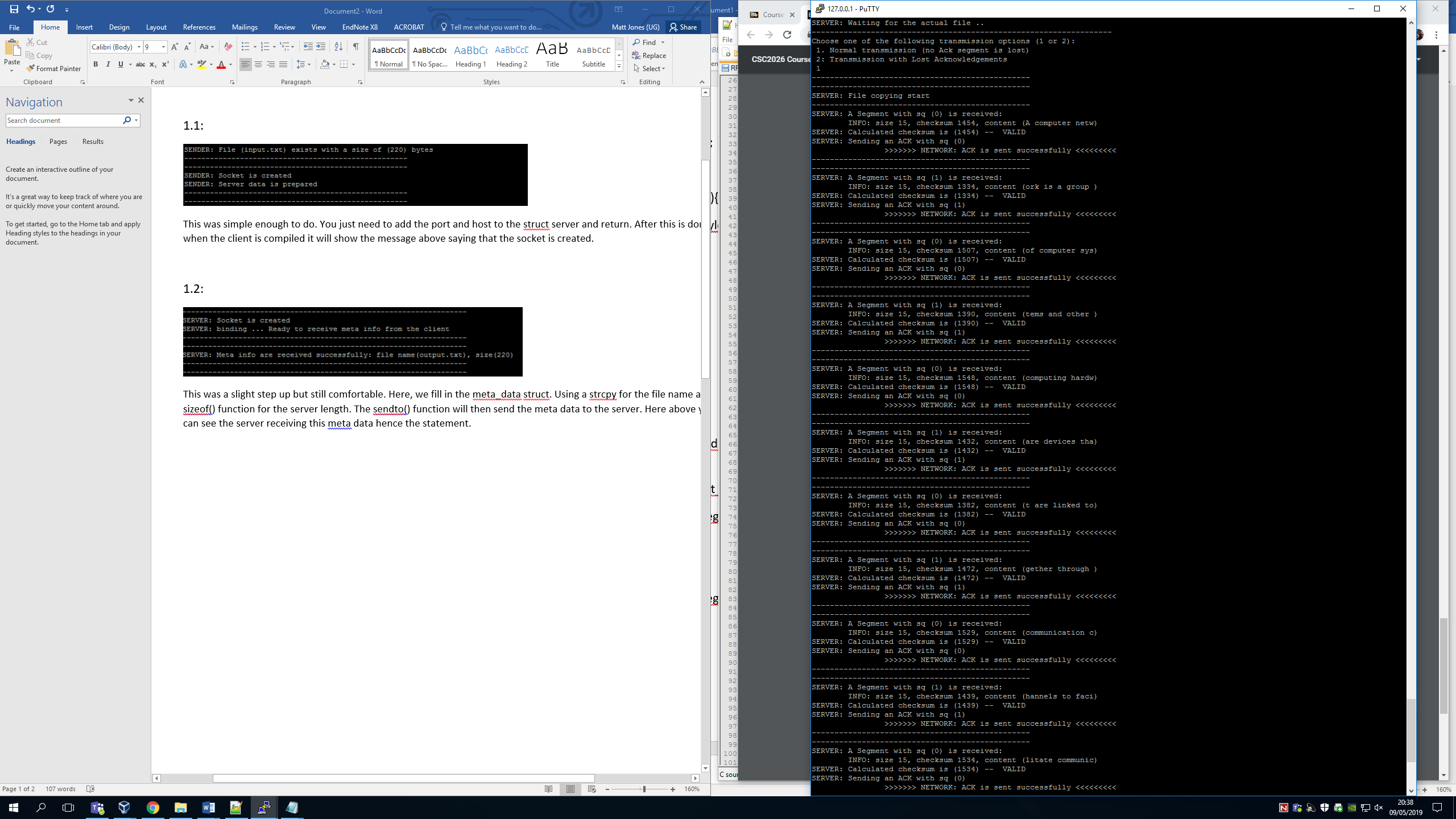
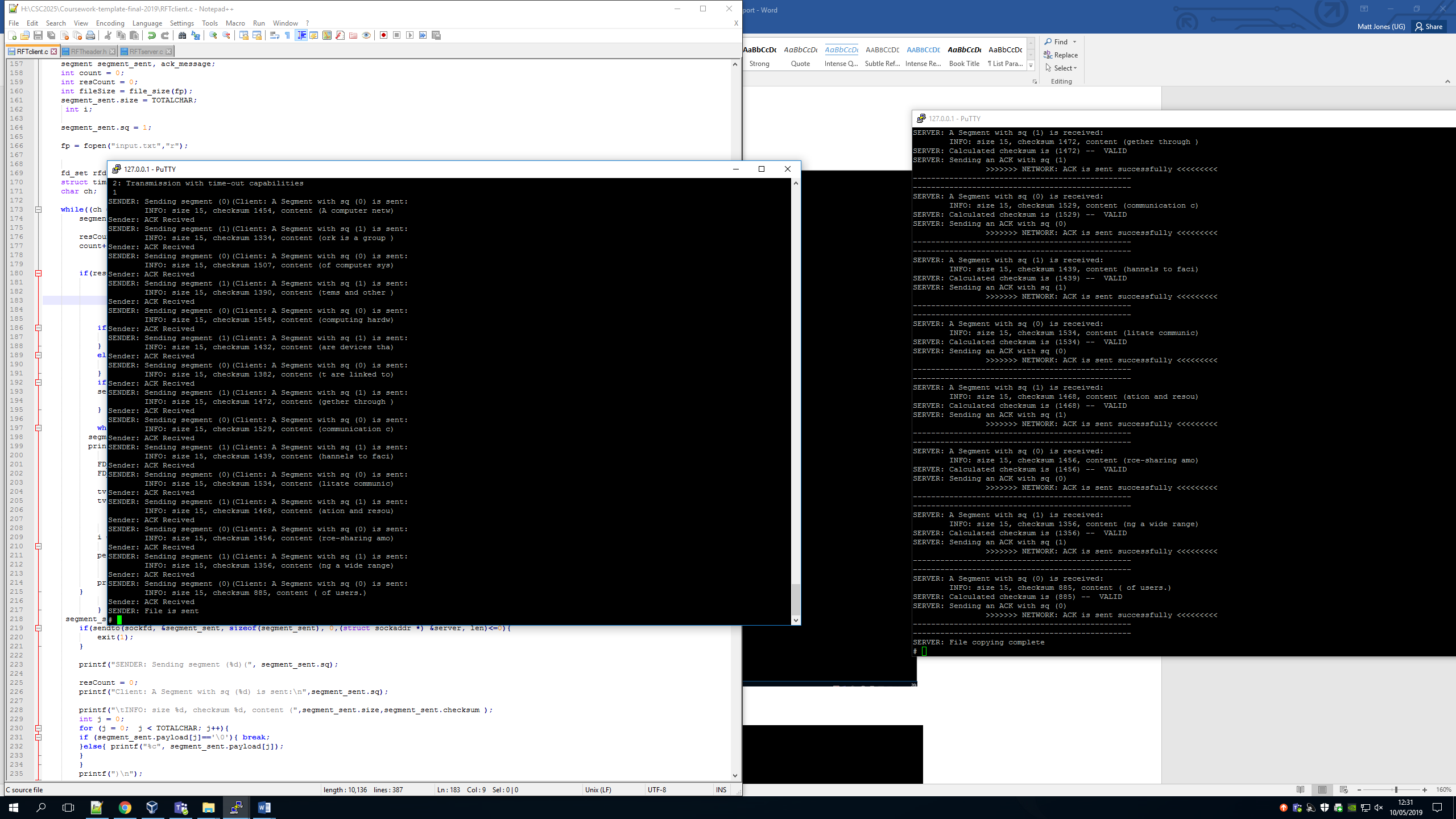
This was simple enough to do. You just need to add the port and host to the struct server and return. After this is done when the client is compiled it will show the message above saying that the socket is created.

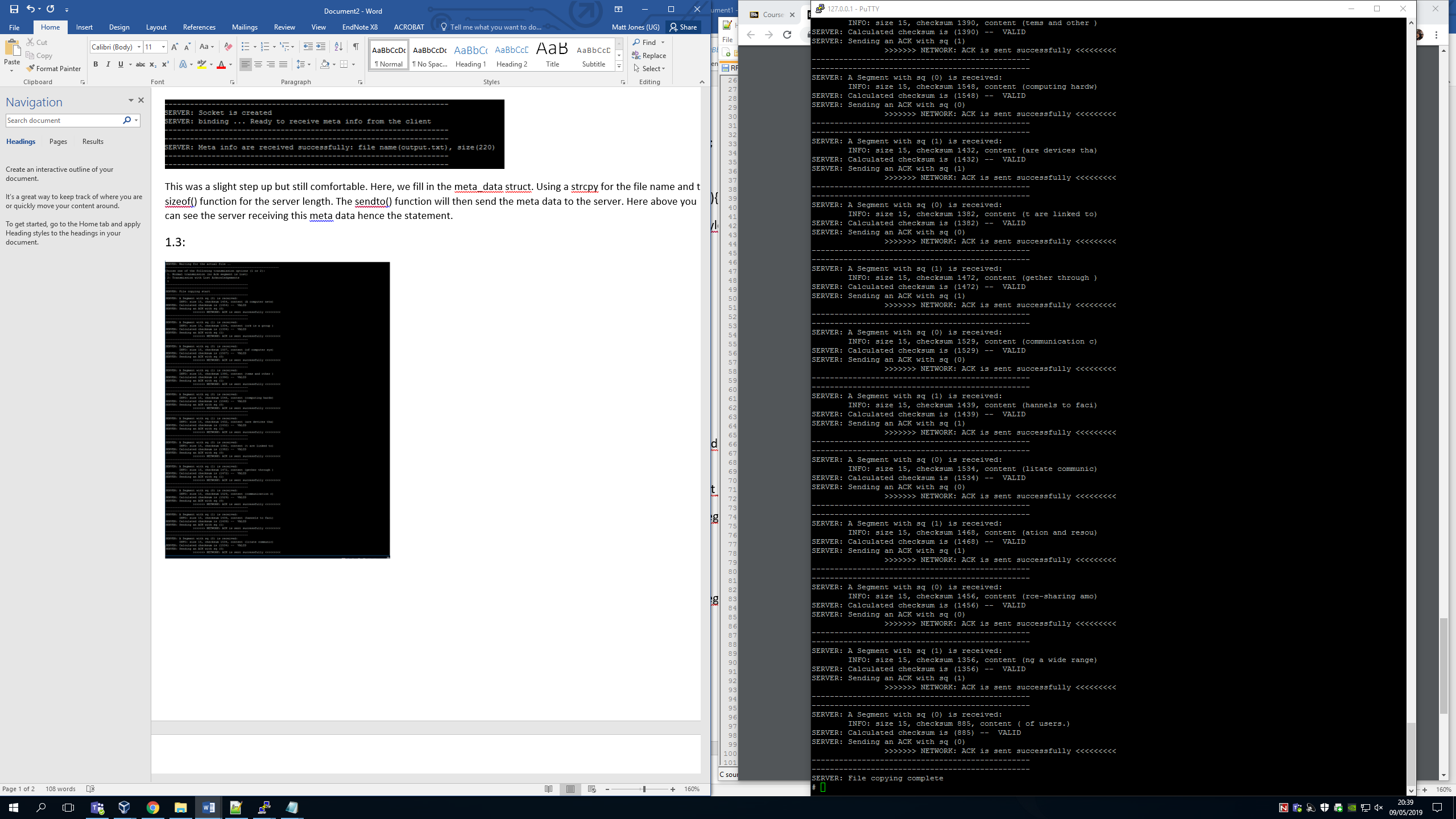
1.2:



This was a slight step up but still comfortable. Here, we fill in the meta\_data struct. Using a strcpy for the file name and the sizeof() function for the server length. The sendto() function will then send the meta data to the server. Here above you can see the server receiving this meta data hence the statement.

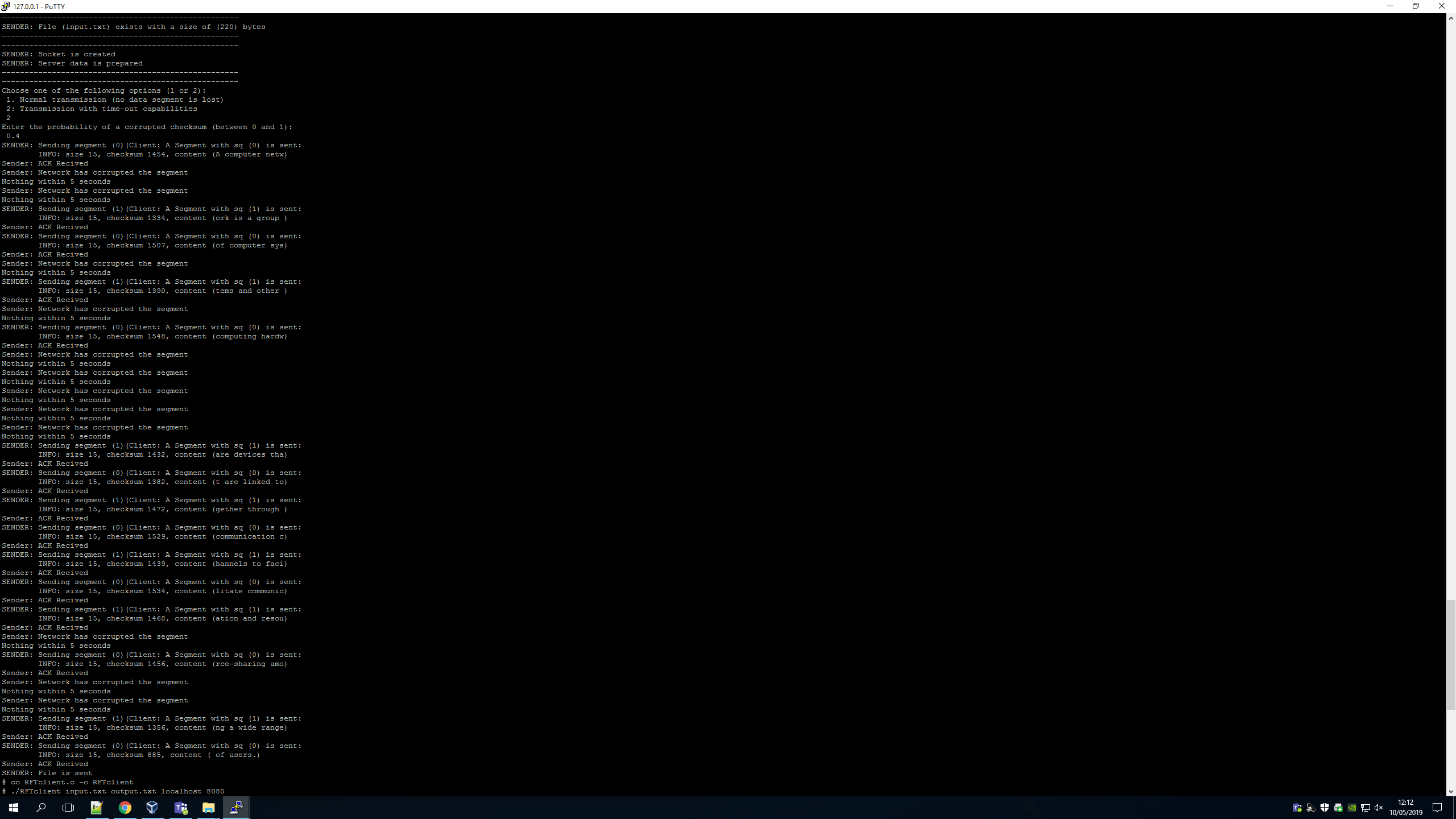
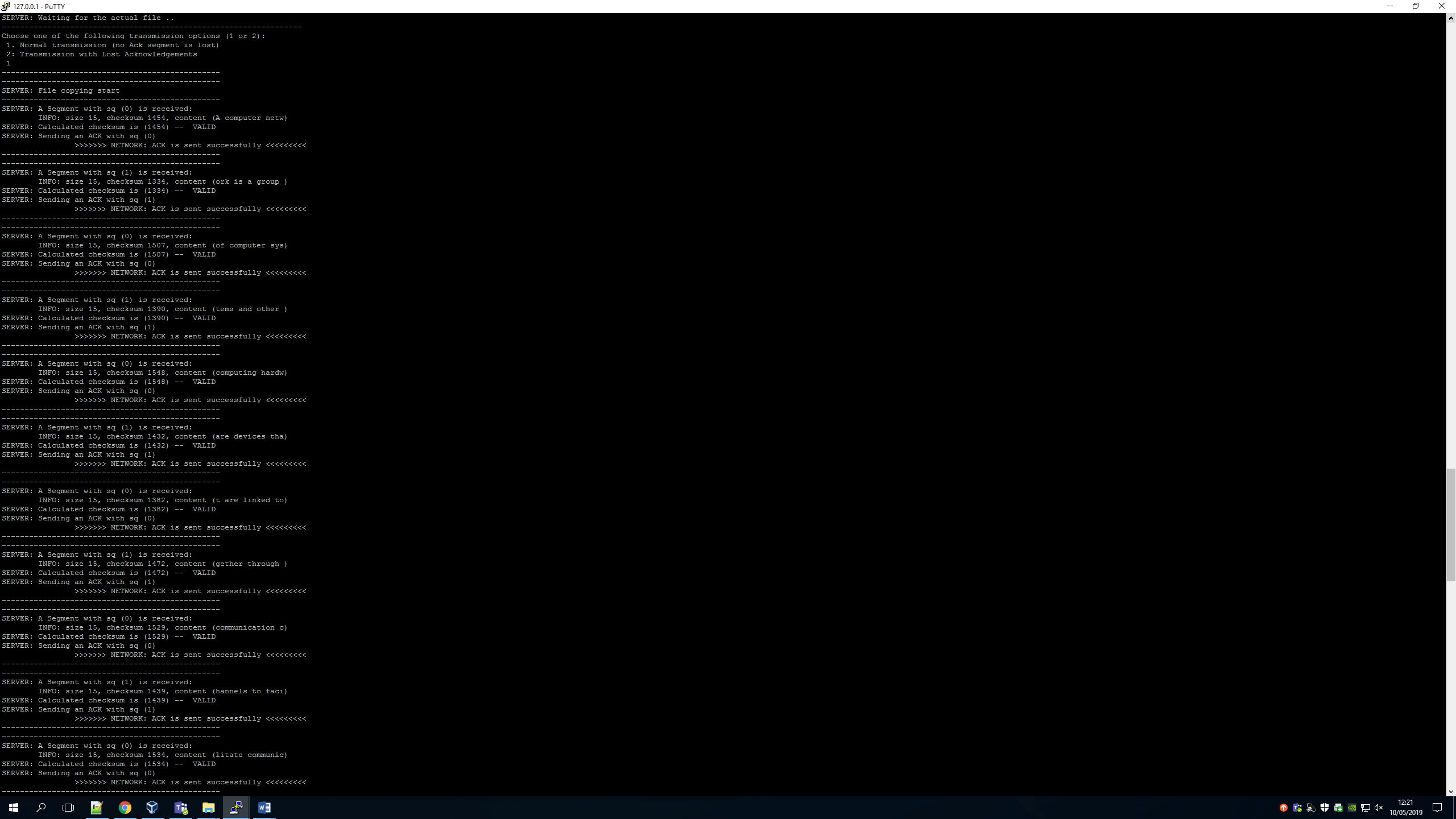
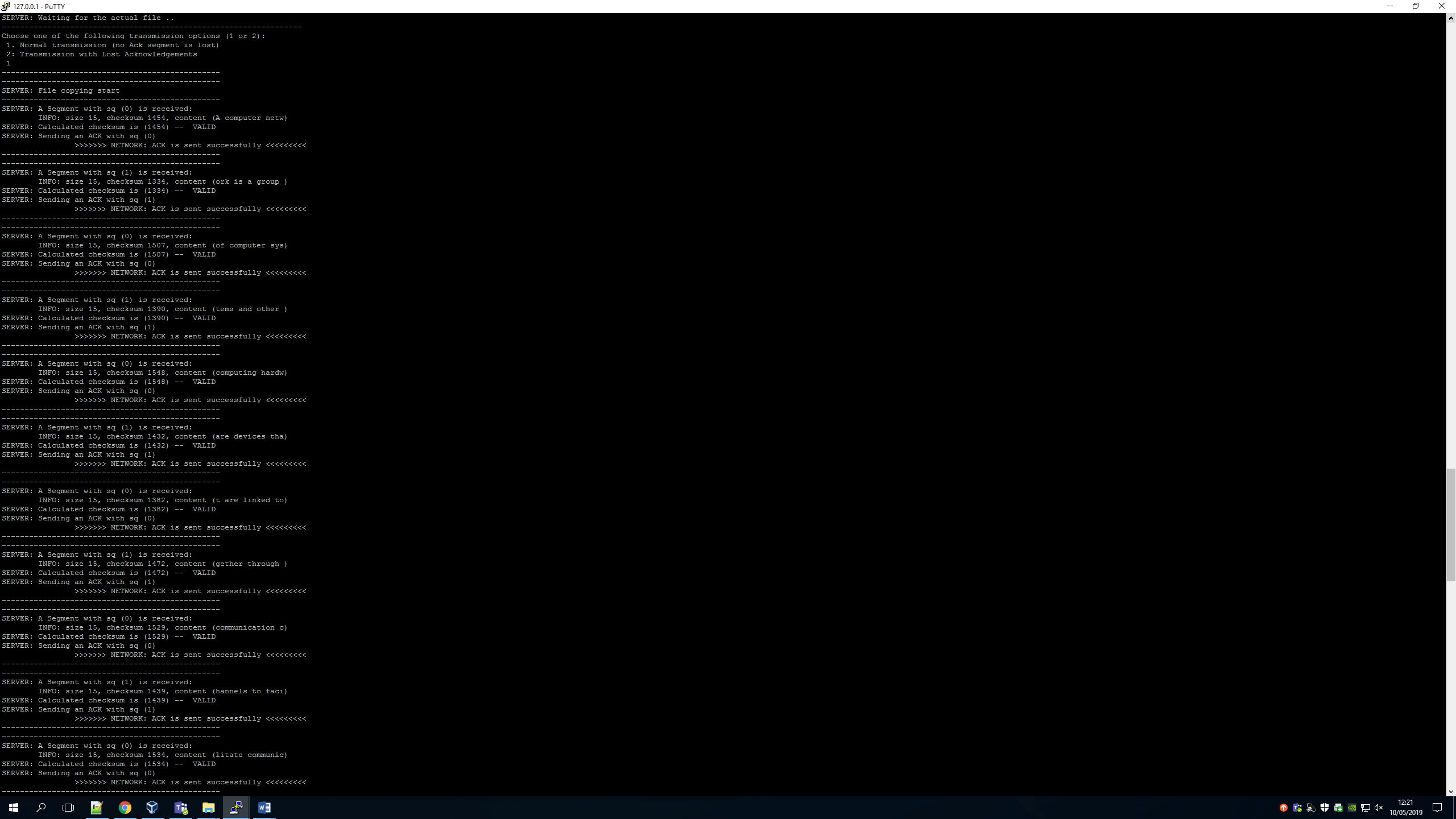
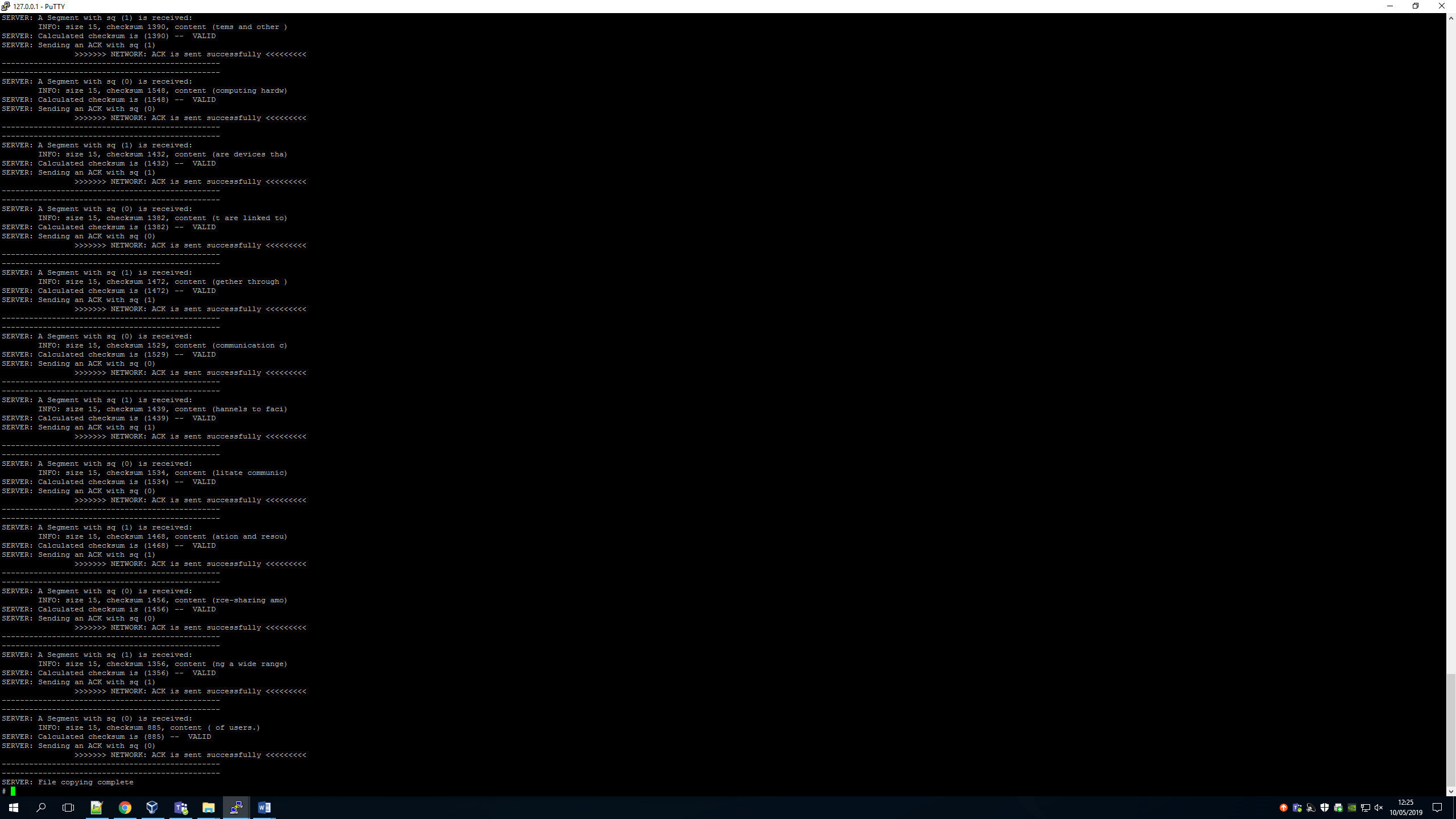
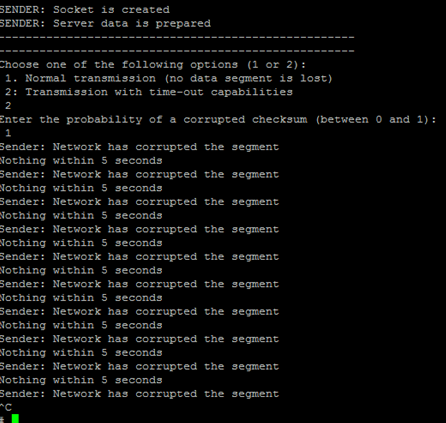
1.3:





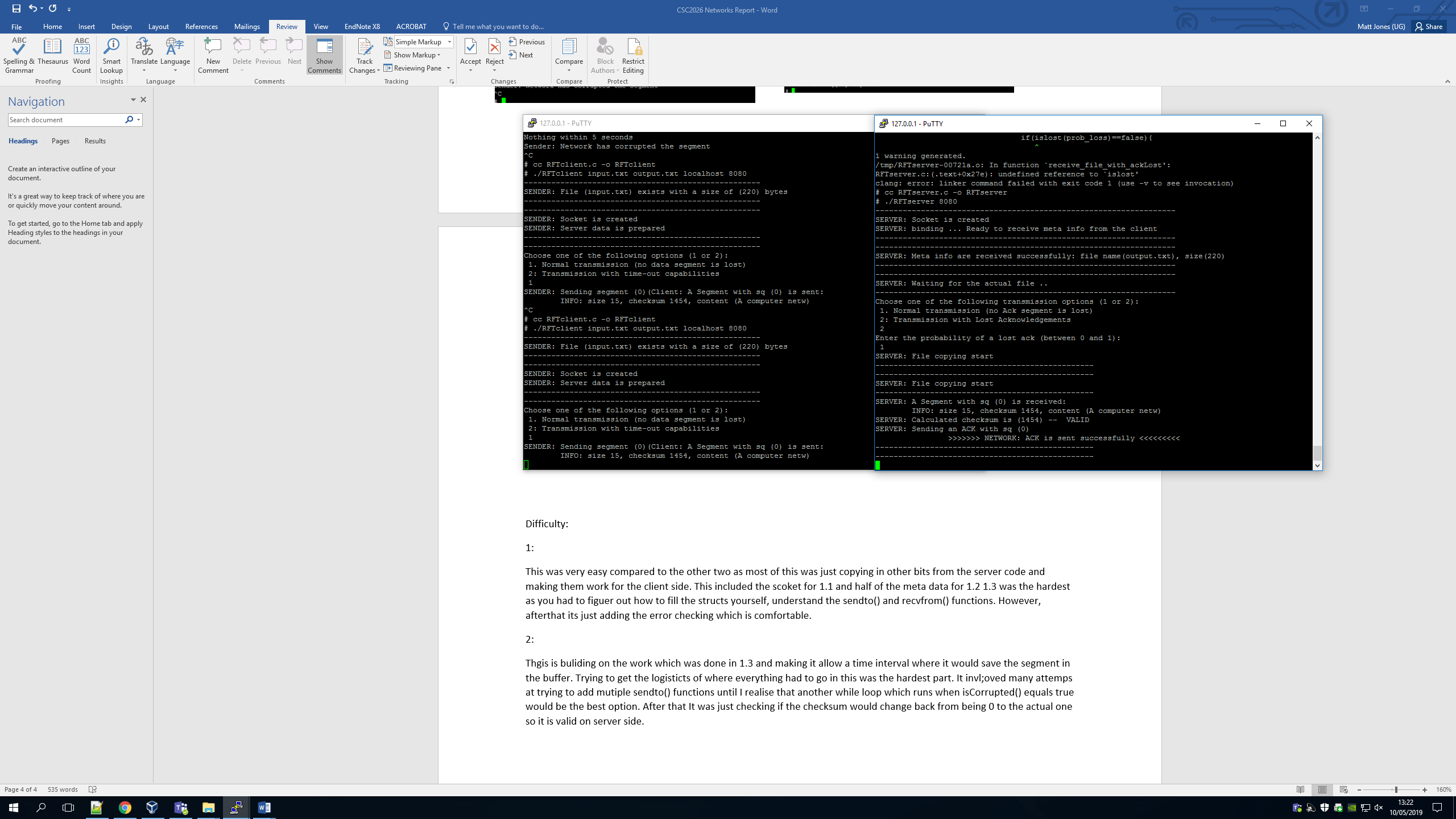
This was a lot harder than the last two steps, this is because when tackling this I didn’t not know how to make certain parts of the struct. First we had to open the file and use getc() to pull the individual characters out of the text file. This was then in a while loop placed into the payload. This when the struct segment is filled is then sent for each segment until the text file is done. After this then and acknowledgement is sent back once the checksum is made valid and the server prints the file is complete.

2:



This task wasn’t that bad at trying to figure out what to do, the hard part was trying to figure out where it went. This is because if the while loop which would wait with the buffer was in the wrong place it wouldn’t work. Above you can see that I gave it a probability of 0.4 for corruption. The first segment is sent fine then the second is corrupted. It has then a 5 second waiting period to try again. This will then try to send to file whilst segments are being corrupted until they are all sent fine. As you can see on both the client and server side the file sends. Also above I have shown what happens if you make the probability 1. Here it will try and send the segment but never will as it will always corrupt causing a infinite loop.

3:



In this we had to make the server side this time lose a message or have it be corrupt. This was done by using the isLost() method and seeing if it will return true or false regarding the probability we write in. Here as you can see above, the server gets the segment and thinks it send the acknowledgement back. However we can see on the client side it never receives it down to it being lost.

Difficulty:

1:

This was very easy compared to the other two as most of this was just copying in other bits from the server code and making them work for the client side. This included the socket for 1.1 and half of the meta data for 1.2. 1.3 was the hardest as you had to figure out how to fill the structs yourself, understand the sendto() and recvfrom() functions. However, after that its just adding the error checking which is comfortable.

2:

This is building on the work which was done in 1.3 and making it allow a time interval where it would save the segment in the buffer. Trying to get the logistics of where everything had to go in this was the hardest part. It involved many attempts at trying to add multiple sendto() functions until I realise that another while loop which runs when isCorrupted() returns true would be the best option. After that It was just checking if the checksum would change back from being 0 to the actual one so it is valid on server side.

3: This was fine at the start when adding in the isLost() method, but trying to check the sequence number was very hard. This is because I didn’t know a way to send the previous acknowledgement back as it had already been overwritten. This meant I could not complete this part fully before the deadline.