Project 3 Part 1 - UDP Putah Report

Nicholas Myers ID: 917617010 Section: A01

Filenames: **server_putah.py**
client_putah.py

1.What are the differences between multiplexing done in a UDP socket vs a TCP socket? What would happen if we were to use the same IP and Port number for transferring data to different clients in our implementation of UDP Putah? (3 points)

- UDP sockets are connectionless and unreliable unlike TCP which are connection-based and encourage reliability. This means that multiplexing in a UDP socket only utilizes two fields, dstIP and dstPort, while multiplexing in a TCP socket utilizes four fields, dstIP, srcIP, dstPort, and srcPort, since TCP connections need to keep track of which packet belongs to which connection. Thus since UDP is connectionless, it does not need to keep track of where packets come from, only where they are going.
- If we used the same dstIP and dstPort number for transferring data, then it would not correctly send the data to the desired clients and even cause socket binding issues since it is attempting to send data on an (IP,PORT) combo that is already in use.

2.Why does TCP require three messages to establish the connection? What would go wrong if you were to attempt building the connection with two messages? (3 points)

- TCP requires three messages to ensure reliability between endpoints, and this is in part due to how sequence and acknowledgement numbers play a part in ensuring that reliability. That reliability occurs due to the synchronization of their sequence numbers, which are randomly generated. There needs to be three messages to ensure this as the client SYN allows the server to synchronize with their sequence number, the server's SYN/ACK acknowledges the received sequence number and concurrently gives its own sequence number back to the client for them to synchronize with, and the final ACK from the client allows the server to assuredly know that the client has synchronized with its sequence number and can reliable begin sharing data between each other. One issue that can occur with only two messages is that only one side of the connection could establish and synchronize their sequence number as only one side could send a sequence number and the other acknowledge it with two messages. This causes the connection to only be

unidirectional and allow only one side to send data while the other can only receive it. This would not allow for an actual communicative bidirectional conversation.

3.Explain how you implemented the sharing of connection socket port numbers between the client and the server in your implementation. Justify your choice of header and message content in sharing this information. (4 points)

- The header I implemented was made up of srcPort, dstPort, and the 3 SYN, ACK, FIN 1-bit flags.
- The client would first send a SYN to the server's welcoming socket.
- The welcoming socket would receive the SYN and parse it. If it were recognized as a SYN, it would prepare to send a SYN/ACK. Meanwhile it also creates a new connection socket (however it is not utilizing it yet as the 3-way handshake is still occurring) and binds it to a random port.
- Then in order to relay the new port to the client in the SYN/ACK without adding another message or adding a payload, the server appends the new connection socket's port to the SYN/ACK header's source port.
- This does mean technically that the SYN/ACK's source port is not the welcoming sockets port number anymore (even though it is coming from the welcoming socket), however, I thought this would be an okay compromise since the client has already began communication with the server's welcoming socket and does not necessarily need its port number anymore. This saves us from sending another message in the handshake or from adding a payload to our message, which it should not have.
- The client then receives the SYN/ACK, parses it and checks if it's a SYN/ACK then sends back an ACK to the server.
- Now, the client creates a new connection socket and binds it to a random port. It now can set the "receivers" (IP, PORT) port value to the server's new connection socket port that was set inside the SYN/ACK's source port.
- The client now creates its message to be sent, the "Ping", and adjusts its own srcPort to that of the client's new connection socket's port, and its dstPort to that of the srcPort of the received SYN/ACK in the 3-way handshake (the port of the server's new connection socket).
- The server receives the first message in its connection socket and parses it. It now has the client's new connection socket's port from its srcPort field and now have both successfully shared each other's connection socket port numbers.