Reliable and Congestion Controlled Data Transfer over UDP

Student 1: Christine Li | 916857224 | A03
Student 2: Minh-Tu Nguyen | 917003682 | A03

Part 1: Adding Connection Support to UDP (UDP Putah)

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?

   The UDP socket utilizes two headers for multiplexing: destination IP and destination port. On the other hand, the TCP socket utilizes multiplexing four headers: source IP, source port, destination IP, and destination port. This is because the connection between the source and destination for TCP persists while the UDP is a connectionless protocol, where it only cares for sending the packets to its destination.

   If UDP were to use the same IP and Port number for transferring data to different clients, the server would have to handle responding to multiple different clients at the same time. As such, this will reduce the efficiency of the communication as the server would only be able to send one message at a time instead of multiple at a time. This may also cause a queue and lead to higher packet loss potential.

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?

   TCP requires three messages (SYN, SYNACK, ACK) to establish the connection because this ensures that both the client and server are able to transfer data as well as agree on the correct data socket port number. If we were to attempt to build the connection with two messages, only the SYN and SYNACK would be sent. This indicates that the server will never receive an acknowledgement that the data socket port number that the server sent to the client was ever received. As such, the server will not know if the client has connected with the data socket and if the client is ready to receive data.

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.

As the client is aware of the server's IP address and port number, the client initially sends a header with SYN = 1 to initiate the handshake connection with the server's welcoming socket. The server receives this packet and sends back to the client a header with SYN = 1, ACK = 1, and the destination_port = data socket port number. Once the client receives this packet, the client will now know to send any data to the provided data socket port. Then the client sends a header with ACK = 1 (SYN = 0) to the server to inform them that the client is ready to receive packets.

Part 2: Adding Reliability to our UDP Putah (UDP Solano)

1. Explain your implementation of sequence and acknowledge numbers. Why do both sender and receiver maintain separate sequence and acknowledgement numbers?

   The first packet that is sent from a sender to the receiver is initialized with the sequence number = ISN (a randomly generated number) and acknowledgement number = 0. Then on the receiver side, it receives the packet from the sender and initializes its first packet to be sent to the sender. This packet has its sequence number = ISN and acknowledgement number = received packet sequence number + 1 (to indicate the phantom byte that helps track the handshake communication). The packet is sent to the sender and then the sender receives the packet. The sender initializes its own packet which has header sequence number = received packet acknowledgement number and acknowledgement number = received packet sequence number + 1 (phantom byte).

   For the first packet that the sender sends to the receiver with data, it will send the same packet as above (as it did not receive an acknowledgement back from the receiver) with the data. Then the receiver will receive this packet with the data. Then, it will send back its own packet with the following headers: sequence number = received ack number, ack number = received sequence number + number of bytes the received data is.

   The pattern will continue as the following for packets being sent from either the client or the server: sequence number = received acknowledgement number, acknowledgment number = receiver sequence number + number of bytes the received data is.

   The purpose of the sequence number is to track how many bytes are sent. The acknowledgement number keeps track of how many bytes are received. This will help indicate if any data components have been lost. Both the server and receiver maintain a separate sequence and acknowledgment numbers to track their own bytes. The server will track how many bytes it has received and sent and the receiver will track how many bytes it has received and sent. This is to inform itself whether or not data was lost when it tried to communicate.

2. Explain and justify the additional header fields you needed to add compared to part 1 for this implementation.

   The additional header fields that we utilized in part 2 (and not in part 1) are unused and CWR. These were used as placeholders within the packet to ensure that the total number of bytes within a packet was equal to exactly 1000 bytes (when data is sent).

3. How would the performance of this file transfer have been affected if we did not use separate welcoming and connection sockets?

   The performance of the file transfer would have decreased significantly as there would only be one receiving socket that would be in charge of transferring data but also connecting with new clients. This will (most likely) cause a queue to form for the one socket to receive and send messages, leading to more packet loss and inefficiency.

Part 3: Adding Congestion Control to UDP Solano (UDP Berryessa)

1. Explain and justify the additions to your packet header to implement this part as compared to part 2.

   The additional header that we utilize is the receive window. The purpose of this header is for the sender to inform the receiver how many packets the receiver should expect to receive at one time before the receiver can send back any acknowledgement packets.

2. Based on the graphs that you generate, list down each time the state of your network changes (from a slow start to congestion avoidance and vice versa), explain the reason behind the change, and report the values of CWND and Slow Start Threshold (SSThreshold) before and after the state change.

   Every 5 rounds, Tahoe drops because it hits the maximum threshold. It reaches size 4 before it goes into congestion avoidance and drops to 1. For Reno, we go through a state change at round 25 and 85 because we hit the threshold at window size 8 and go into congestion avoidance, dropping to 4, which is half the previous window size.

3. Compare the bandwidth difference between TCP Tahoe and Reno in your experiments. Compare it with the bandwidth that you measured in part 2. Do you notice any significant

difference between these implementations? Explain why there is or why there is not a difference.

Tahoe achieved 2764 bits/second while part 2 we got 1240 bits/second so roughly half. Reno achieved 3681 bits/second, an even larger bandwidth than Tahoe. The implementations differ in that Tahoe will lose bandwidth coverage by going back down to 1 when we get duplicate acks whereas Reno will go to half the window size which saves a lot of bandwidth.

4. Based on your experience and results in designing these different models, what else can be done to improve the bandwidth usage of your network?

   To increase bandwidth we could utilize TCP cubic. Instead of wasting bandwidth by using Tahoe or Reno, we could remember the time when a "bad thing" happened like packet loss and aggressively go back to that point then conservatively go beyond that threshold until no packets have been lost for a while, at which point we can start being aggressive again until the next packet loss. We can also open up a bunch of parallel TCP servers and cheat by taking up more bandwidth this way. Additionally, we can start at a larger window size than 1, which gives us an advantage over competing clients.

Logging Function:

The file log.py merges the output files of our simultaneous clients. To run log.py, the arguments in the terminal must be the following: first_log.txt second_long.txt type_of_udp file_read. Type of UDP indicates putah, solano, or berryessa. File read indicates either big or alice or none.

This will create the merged log file in order of timestamp.

Best of luck

Submission Page

I certify that all submitted work is my own work. I have completed all of the assignments on my own without assistance from others except as indicated by appropriate citation. I have read and understand the university policy on plagiarism and academic dishonesty. I further understand that official sanctions will be imposed if there is any evidence of academic dishonesty in this work. I certify that the above statements are true.

Team Member 1:

Christine Li	[illegible]	11/22/22
Full Name (Printed)	Signature	Date

Team Member 2:

Minh-Tu Nguyen	[illegible]	Nov 22, 2022
Full Name (Printed)	Signature	Date