CS 118 Spring 2018

Project 2 Report: Simple Window-Based Reliable Data Transfer

Shen Teng, 104758168

Goal

The purpose of this project is to use UDP Socket and C/C++ programming language to implement a reliable data transfer protocol (TCP). Since UDP itself is a not reliable data protocol, the goal of this project is to fully understand and learn how to implement a simple congestion window-based TCP protocol.

Overall Implementation

This section has brief overview of the implementation of our project. Details will be explained in later sections of the report. Firstly, we ensured the client and server establish a connection with handshake. Once the handshake is successfully executed, the client sends the filename to the server. Then the server finds the requested file and send the file to the client by reading only the maximum size into each packet and sending the packets until it gets the end of file. The client simply keeps getting the packets and writes the received data into a file "received.data". After the client gets all the packets, the client sends FIN signal to the server to close the connection. Once the server gets the packet with FIN signal, the server also sends a packet with FIN signal letting the client know that the connection is closed. Throughout the whole packet transmitting process, we used threads to implement timeout of packets.

Header Format

The header of our packet includes acknowledge number, sequence number, ACK flag that indicates that the packet includes acknowledge number, SYN flag that indicates the packet is for setting up the connection with handshake, and lastly FIN flag that indicates that the client or server is closing the connection. The total size of our header is 11 bytes which makes our payload size 1013 bytes.

Handshake

The implementation of handshake was done by making the client send a packet with a SYN flag. In case of the first handshake packet is lost before it reaches the server, we made the client set the timeout for the first handshake packet so that it will retransmit the first packet if the server does not reply. Once the server gets the packet with SYN flag set to 1, server also sets the SYN flag to 1 and sends the packet to the client indication that the connection is established. Then, the client finished handshake and sends the required filename to the server.

Messages

Whenever the client or the server receives or sends a packet to the other, it displays a corresponding message showing the ACK number or sequence number. The messages also indicates if the packet is retransmitted, or is establishing or closing the connection. In order to implement this, at each time the client or server receives or sends a file, we get the needed information from the header of the packet and include the corresponding information in the messages.

Timeouts

When the client and server establish the connection, it is the client who sends the first packet to the server. So, we made the client to set the timeout as it sends the first handshake packet to the server. Once the first handshake packet from the client successfully reaches the server, we made the server keep setting the timeouts for each packet it sends to the client.

As described before, we implemented the timeouts using threads. Basically, whenever the server (or client only for the first packet) sends a packet, we made it create a new thread and go to sleep for the specified time-out value (500 ms). At the same time, we made a set and put the sequence numbers of the sent packet into the set to keep track of sent packets. And whenever the server gets a packet from client that acknowledges that the packet was delivered, the server gets removes the sequence number from the set. This way, after the thread wakes up from the sleep, it checks if the sequence number of the corresponding sent packet is still in the set. If the set contains the sequence number, it repeats this process until it confirms that the packet is delivered to the other side.

Window-Based Protocol

For the window, we set the window size to 5, and we implemented it with a double-ended queue. In the beginning, the window queue is filled with null and each item in the queue corresponds with each packet sent. Once the client or server gets the correct ACK or SEQ number that corresponds to the sent packet, our application fills in the window queue with the packet.

As the above process repeats, our program keeps checking if the front of the window queue is null or filled in with a packet. If the front is not equal to null, it means the oldest packet sent in the current window is successfully delivered, so we pop the front, increase the queue at the end, and send a new packet. So in our window-based protocol, the window does not move until the oldest packet is confirmed to be transmitted to the other side.

Difficulties and How We Solved Them

1. Implementing timeout using threads
   When the server or the client need to send packets that require retransmission, they will create a thread after the packet is sent for the first time. The thread will use sleep for the TIME_OUT seconds and check if the packet has been ACKed or not.
2. Synchronizing the SEQ and ACK number with the window
   The double-ended queue is used to manage the window. The property that the elements can be accessed using index and the popping operation fit the use of window management well.