Simple Window-based Reliable Data Transfer

Lab Report

Zhouyang Xue 104629708

Yang Li 904642975

Manual

1. Made to create the executable server and client. Server is the server side of this program which sends out the file to the client whereas the client is the client side to receive the data.
2. ./server <#port> to run the server
3. ./client hostname <#port> filename to get the file from the server

Logic and Design

1. As the packet has header and payload, we design a data struct for the packet. The variable includes the data type to indicate it's SYN or Fin or Ack or Data or Retrans to show what to do for the client and server.
2. Our client first sends a SYN to the server on requesting the connection. We use select to find if there is package coming into the socket to determine if the SYN is timed out.
3. The server keeps waiting for the SYN from the client. If it received a SYN from client, the server will sends a SYN-ACK.
4. The server keeps track of if client sends a file name to the server. If the server received a file name, it will look at the filename within the current directory.
5. If the server did not get the file name it will retrans SYN-ACK to request a filename.
6. If the server did not find the file with specified filename the server will respond to the client with a packet with type TYPEERROR to indicate the filename not found.
7. After that the server divides the packet into several packet and we have the window size as 5 packet each packet with 1024 byte, header 32 byte and payload size 972.
8. In the transmission process. We have a variable to indicate the first not transferred packet within the window. We also had a array of 5 packet to keep track of the pacet. After the packet has been transferred, the type will be then set to 3 to indicate it has been transferred and next time it's retrans.
9. For simplicity, we keep track of the first packet in the window for timeout. Only if it's timeout we retransfered it. For the rest, we did not immediately sent packet retrans and check timeout.
10. After the server received all the ack and at the end of the file. It will send a fin to the client and expected client to send a fin to close the socket.
11. The client will respond and close.
12. The server will close the socket after 3 retrans fin no matter whether or not it received fin from the client.

Challenges and Difficulties

1. To design proper FIN message communication is with no doubts one of the most challenging part of this project. We spent a lot time trying which of the client and server should send the first FIN, and util which message the connection should be shut down. In the end, our server sends the first FIN after all the data packets have been ACK'ed, and

waits for a FIN-ACK from the client. Once the client received the FIN message, it responds with a FIN-ACK. Then, the server shuts down right after receiving the FIN-ACK. If timed-out, server resend FIN. In case of packet loss of the FIN message, the server should send at most 2 FIN, and if it doesn't receive any response from the client, it shuts down anyway. Likewise, client shuts down if the connection idles for more than 2ROTs.

2. We also encountered the challenge of implementing 404-not-found behaviors. Originally, in our design, the server sends a 404 signal back to the client and shut down immediately. However, in a unstable connection where loss rate is high, it is highly possible that the 404 signal gets dropped. In this situation, the server has already closed the connection, but the client is still waiting for the first data packet to be transmit. Therefore, the client goes into an infinite loop retransmitting the filename to the server. Our method to overcome this difficulty is instead of shutting down server immediately, after we send the 404 signal, we start the FIN communication. Since we have already implemented a safe FIN protocol, this approach would allow as to safely shut down both server and client.

Test

1. We first tested a trivial test.txt under good-condition connection.
2. We then tested a larger test file under good-condition connection.
3. We tested a large test file under good-condition connection where the sequence number of packets got reused.
4. We installed Oracle Virtual Box and installed a Ubantu virtual machine.
5. We used tc command to test loss rate 10%, 20%, 50%, and 80% on trivial files.
6. We also used tc command to test loss rate 10% and 50% on regular sized files.
7. We finally tested all "bad behaviors", such as FILE-NOT-FOUND, SYN message loss, and FIN message loss.