CS 118

Project 2

Reliable Data Transfer over UDP using Selective Repeat

Gabriel Powlowsky

UID: 204486619

Phuoc La

UID: 104567852

Summary

Our server-client pair implement reliable data transfer over UDP using the selective repeat protocol.

Implementation

Basics

Once the server is started, the client sends a file request to the server. The server then opens the file and calculates how many packets it needs to divide the file into. Then, it begins transmitting the file in chunks (packets). Each packet consists of a sequence number (given in bytes), packet type (0 for request, 1 for ACK, 2 for data and 3 for FIN), content length and data. Each packet is 1024 bytes maximum, and the maximum for the data portion of the packet is 1012 bytes.

We use the size of a packet for the sequence numbers. So, the first sequence starts at 0 and then grows to 1024, then 2048 and so on. The maximum sequence number is 29696. After the maximum is received, the sequences restart at 0.

Selective Repeat

For implementing selective repeat, we follow the protocol described in Kurose and Ross. The server has a given window that limits the number of outstanding packets that can be sent. After sending the maximum number of packets, it cannot send more until it has received ACK packets from the client. If the window size is set as 5120, five packets can be in transmission to the client at any given time. As soon as a packet has been ACK'ed, another packet can be sent.

The client also has a window it bases to limit the packets it receives. If a packet falls within the window range of the client, the client will send an ACK to the server containing the sequence number of the packet, regardless of whether it is the next expected packet or not. If the packet sequence equals the base (the next expected sequence number) of the client or above, it is buffered. If the arriving packet sequence number equals the base, it is written to file, as are any other contiguous packets that were previously buffered.

Loss and corruption

Both the client and server simulate loss or corruption of arriving packets. For both the client and server, the arriving packet is ignored if there is loss or corruption. We use a random number generator to simulate loss and corruption. Both result in timeouts, at which point the server will resend the packet. To achieve this, each packet has a timer associated with it, and if a packet does not arrive within the timeout window, the packet is marked as timed out and resent.

Closing transmission

After the server finishes sending all the packets, it sends a FIN packet to the client. The client then sends a FIN packet to the server and then closes the file.

Difficulties

Debugging was challenging for this project, as we each designed different parts. Gabriel worked on the server, and Phuoc worked on the client and the extra credit. Without having built both parts, it was hard to know whether a bug was caused by the server or by the client. We each had to check the errors to make sure that what we had built was working as expected. In particular, it was challenging to debug errors when there is loss or corruption in both the server and client, as this creates a confusion situation where it's hard to know where an error originated. Also, we had to understand each other's code in order to figure out what could go wrong. In addition, modifying someone else's code is much more difficult than modifying code you wrote yourself, as understanding existing code can be as difficult as writing the code yourself.