CS118 Project 2 – Implementing Window-based Reliable Data Transfer Protocol

Yu-Kuan (Anthony) Lai 004445644

Elton Leong 204457607

In this project we implemented a Reliable Data Transfer Protocol (RDTP) that handle packet loss, delay, and reordering on top of UDP. Using RDTP, our application layer protocol, Basic File Transfer Protocol (BFTP) is able to transfer files between two hosts. The server and client programs then utilize both protocols to implement its side of the communication and file transfer.

Header format

The header of our Packets for RDTP is 5 bytes in size. It consists of a 16-bit Number field storing the sequence number, three 1-bit flag fields indicating SYN, ACK, and FIN, 5-bit padding, and a 16-bit Window field.

Messages

When the client first starts, it needs to connect with the server via a three-way handshake. Client will then send a packet with the requested file's filename. Server will respond with whether the file is found, and if so, the size of the file in bytes.

After sending all the file packets and receiving all the ACKs, the server will terminate the connection with the client using the FIN/FIN-ACK procedure. Client will wait for 2*RTO before closing its connection.

Timeouts

Following the requirements from the spec, the Retransmission Timeout value implemented by the RDTP is 500ms. The TIME-WAIT mechanism for terminating the connection use 2*RTO which is 1 second.

Window-Based Protocol

RDTP implements the Selective Repeat protocol to ensure reliable data transfer. The protocol uses a maximum window size to limit the amount of unACKed data sent by the server. If the sender receives an ACK for the packet with lowest sequence number within the sender's window, the sender's window slides forward, allowing more packets to be sent. If any unACKed packet within the sender's window timeout, that packet is resent by the sender and its timer is restarted.

Receiver of RDTP also has its own receiver's window. This window indicates the range of data it will accept and buffer, regardless if the data received are in-order, for eventual delivery to the application layer. If the receiver receives a packet whose sequence number equals to receive_base, all continuous data starting from receive_base will be delivered to the application layer, and the receiver's window will slide forward. If the receiver receives a packet within one full window above and below its current receive_base, it will respond with an ACK equal to the sequence number of the packet it receives.

For both sender's and receiver's windows, the window will rotate back to sequence number 0 when it reaches beyond maximum sequence number (i.e. 30720).

Difficulties

We ran into few segmentation faults and bugs. We were able to solve them with the help of GDB and Valgrind.

For other stupid bugs such as break inside switch statements and implicit type conversions, caffeine seems to do the trick.