Alvin Lim, 804 011 675
Tremaine Eto, 904 171 710
Professor Lu
CS 118
Fall 2015

Project 2: Simple Window-Based Reliable Data Transfer in C/C++

I. Introduction

The overall purpose of this project was to use UDP socket programming and C/C++ to implement a reliable data transfer protocol. Since stop-wait and stop-forward protocols were not allowed, we decided to go with a **Go-Back-N** protocol, a specific case of the sliding window protocol since it utilizes window sizes.

II. Set-up and Running

1. Once in the project directory, run "$ make" to execute the Makefile and create executables for both the sender and receiver.
2. First start the sender (sender.cpp) by running "`$ ./sender <port_number> <window_size> <loss_probability> <corruption_probability>`" where port_number is greater than 0, window_size is greater than 0, and both loss_probability and corruption_probability are between 0 and 1.0.
3. Then, start the receiver (receiver.cpp) by running in a separate terminal window "`$ ./receiver <host> <port_number> <file_name> <loss_probability> <corruption_probability>`" with the same conditions outlined in Step 2.

III. Implementation

1. Header Format

a. We opted to create a common packet structure (in its own packet.h that we included in both sender.cpp and receiver.cpp).
b. The first field is **char type.** Here, we set the type as 'R' for a request packet, 'D' for a data packet, 'A' for an ACK, and 'F' for a FIN (where we end communication).
c. The second field is **int seqNum.** Here, we set the sequence number of the packet.
d. The third field is **int size.** Here, we set the size of the packet data.
e. The fourth field is **char data[MAX_PACKET_SIZE].** Here, MAX_PACKET_SIZE is defined as 1KB, or 1024 bytes. It is essentially a character array to store the file data.

2. Time-outs

We handle time-outs by checking if the packet_number is either greater than or equal to the window size, which means it has gone through the while loop once to check for a new ACK. Also, we check if the number of the sum of the last_acked and the sequence number are greater than or equal to the total number of packets_needed. In these cases, we indicate that there has been a time-out and we print the sequence number.

3. Window-based Protocol

As mentioned in the introduction, we have chosen to utilize a Go-Back-N protocol. The reason why we opted for this particular ARQ protocol was for reliability purposes; there is a window of sequence numbers (specified by window_size by the user) that can be transmitted without acknowledgement. The receiver accepts only the next sequence number it is expecting while other sequence numbers are ignored. For further overview of how GBN was implemented, refer to section III.5, "Sender".

4. Sender

The sender initially waits for a request for a specified file from the receiver. It then checks to see if that file exists or not. If the file does not exists, then the sender automatically sends a FIN to end communication. If the file does exist, however, then we calculate how many packets are needed (by simply dividing by 1KB, or 1024B, and putting the remainder into another packet).

To implement Go-Back-N protocol, we have the sender send packets up until it hits the window size specified by the user (counted from the last ACK). At this point, a time-out occurs and a retransmission occurs.

When we actually receive an ACK that is expected (utilizing if checks for the type being 'A' and then checking the incoming packet's sequence number against the last ACKed number), then we slide the window over and the packet sequence number goes back to 0, as specified by GBN.

Finally, at the point that all packets have been sent over to the receiver and all ACKs have come, then a FIN message is sent to end communication.

5. Receiver

The receiver first builds a request packet (utilizing our build_packet() function) denoted with a type of 'R' for request. It then sends it over to the sender side, which checks if the file exists or not. If the file does not exist, a FIN is sent and communication halts.

If the file does exist, however, then the receiver begins to take packets of data by checking FD_ISSET() to check if our file descriptor is part of the set after our select() returns. Within the receiver, we simulate loss and corruption by simply utilizing rand() with the probabilities specified by the user.

We then ignore packets with incorrect sequence numbers (these packets would be out of order) as per GBN or packets that are not denoted with 'D' for data. We finally sent an ACK message back to the sender. Then, if a FIN message is received (denoted by a packet with 'F' as its type), then the receiver sends a FIN back and communication ends.

6. Loss and Corruption Simulation

We made the simulation of loss and corruption by taking the specified probabilities by the user and utilizing the rand() function to implement them. More specific to GBN, we simulated it in the receiver by taking the reply of the sender and having it have those probabilities of being lost or corrupted. In the case of corruption in the receiver, we had the ACK be retransmitted in an outgoing packet. On the sender side for loss and corruption simulation, we printed out an indication of a loss or corruption and then made sure to set the packet number back to 0 as specified by a GBN protocol.

IV. Difficulties

One difficulty was making sure that we understood how Go-Back-N protocol works; what really assisted us was looking at the graphical animation mentioned in lecture and the textbook to see how the window slid and seeing how the packets were sent according to the ACKs.

Another difficulty was determining the exact conditions to check with our if statements for time-outs and when to time out. We faced this again by going through the Go-Back-N protocol procedure and breaking it down into smaller cases.

Another difficulty was learning how to be accurate with file descriptors and making sure to find the right system calls and I/O functions. We had to find select() and ensure that we understood it as well as other file descriptor specific functions such as sendto(), recvfrom(), FD_ZERO(), FD_SET(), FD_ISSET(), etc. Linux man pages were very helpful in this sense, as we had to determine the exact use cases, parameters, and return values.