COM SCI 118
Computer Network Fundamentals
Winter 2017

Project 2: Implementing Window-based Reliable Data Transfer Protocol

Shannon Tom (UID: 904475976)
Michelle Tran (UID: 304448702)

Design

In this project, we implement a simple window-based reliable data transfer protocol modeling the Selective Repeat protocol on top of UDP. We do not implement TCP congestion control. This project includes three files: server.c for the server that sends data, client.c for the client that receives and reorders packets, and packet.h for the declaration of a packet data structure.

The client and server communicate using a UDP socket. The client connects to the server and requests a file through the server's IP address and associated port number. The server then checks if it has the requested file; if yes, the server sends the file to the client in 1024 byte packets.

The server and client should be run using the following command line arguments, respectively:

```
./server <portnumber>
./client <server_IP> <server_portnumber> <filename>
```

Implementation

Packet Format:

The packet structure defined in packet.h contains header fields and a character array. In total the packet is a constant size of 1024 bytes.

The packet header includes 14 bytes of data, distributed amongst 4 header fields:

1) seq - The sequence number of the packet. This sequence number is between 0 and 30720 and is incremented by the number of bytes of file data the server has sent. When the sequence number reaches the maximum value of 30720, it resets by wrapping around from 0.
2) nextSeq - The next expected sequence number of the packet. This indicates which packet is sequential to the current packet and is used to determine packet reordering by the client.
3) length - The number of bytes of data payload in the packet.
4) type - A short from 1 to 5 that corresponds to the type of packet. The type definitions are:
    1 - DATA
    2 - SYN
    3 - ACK
    4 - FIN
    5 - SYNACK

The remaining 1010 bytes of the packet are allocated by a character array, which represents the data field. The server will read the file data directly into this character array before sending the packet.

Messages:
The client and server print out messages in the following format when they send or receive packets. Text in square brackets must always be present, while text in parenthesis vary dependent on the action taken.
Server:

"Sending packet" [Sequence number] [WND] ("Retransmission")("SYN")("FIN")
"Receiving packet" [ACK number]

Client:

"Sending packet" [ACK number] ("Retransmission")("SYN")("FIN")
"Receiving packet" [Sequence number]

Timeouts:
Timeouts are implemented using the select() function defined in <sys/select.h>. This function takes a set of file descriptors and a timeout value as arguments and returns the number of file descriptors ready to be read or written within the timeout value. If the select() function returns 0, then no file descriptors are ready before timeout occurs, which results in the retransmission of the packet. The select() function decrements the timeout variable as it counts down to 0 s.

In order to ensure that the correct retransmission timeout value of 500 ms is applied, each packet within the server window is marked with a timestamp at the time the packet is sent, recorded through clock_gettime(). The remaining timeout value is calculated by the difference between the oldest packet timestamp (the first packet within the window) and the current time.

Packet Window:
On the server end, the window-based protocol is implemented using an array of window structures. This window structure contains three elements: a packet structure, a timestamp corresponding to the last transmission time of the packet, and an integer named "isAcked" that is set either to 0 or 1 to indicate if the packet has been acknowledged.

Once a packet is sent, it is appended to the end of the window array along with its current timestamp. Received acknowledgements from the client are checked against the packets in the window. If their sequence numbers match, then isAcked is set to 1, then the window array will be updated accordingly by moving the unACKed window structures forward down the array.

Packet Buffering:
The client buffers the received, out-of-order packets in order to handle cases when packets are lost and must be retransmitted, or when received packets are not directly sequential.

Once a packet has been received, it will either be written to file immediately or stored in the packet buffer. A “last ordered packet” is maintained. If the incoming packet is the very first packet of the data, it is written to file. If the packet is not the first packet, it is checked against the last ordered packet to determine if the current packet is the expected next packet. If it is, then the current packet is written to file. If not, it is buffered.

If there are packets in the out of order buffer, the received packet’s expected next sequence number will be checked against each buffered packet’s sequence number; if there is a match, the buffered packet will be written to file and the buffer itself will be updated accordingly by shifting the subsequent packets up by one index, and then subtracting one from the number of out of order packets. The newly written packet will now be marked as the “last ordered packet” and checked against the updated buffer. The buffer will be checked until there is no match for the next expected sequence number. The packet is ACKed and the next packet will be received.

An ACK buffer is also maintained in order to check for received duplicate packets; if the packet has already been ACKed, it can simply be ACKed again.

Difficulties

One difficulty that arose was figuring out how to implement timeouts. We first attempted timeouts using select(), but due to a logic error on our end, we shelved this idea and tried implementing timeouts with timer_create(), which sends signals that can be handled by a signal handler once the timer expires. However, we encountered difficulties with this implementation because of race conditions in the signal handler. Finally, we re-examined the select() implementation and caught the logic error.