CS118 Spring 2018
Project 2: Simple Window-Based Reliable Data Transfer

Sean Langley 504-661-838
Susan Krksharian 104-584-663

June 8 2018

1 Implementation description

1.1 TCP Header

Our TCP packet headers contain a sequence number, acknowledgement number, flags field, and data length. The flags field contains options for whether or not the packet is an ACK packet, DATA packet, SYN packet, SYNACK packet, FIN packet, RETRANS packet, or an EOF packet. Sequence numbers are only used in data packets and ack numbers are only used in ack packets. A possible implementation may have not needed to have both sequence and ack numbers in the packet header, but it made the design simpler and easier to understand.

1.2 Three Way Handshake

When the server program starts up, the server initializes a socket and waits for a SYN packet on its service port. When the client program starts up, it sends a SYN packet to the server which includes its initial sequence number. When the server gets the SYN packet, it sends back a SYNACK packet, which acknowledges the client's SYN packet and also contains its own initial sequence number. The client then responds with an ack. After the server receives the ack, the client and server can begin the file transfer. A better implementation sets the client and server's initial sequence numbers to random values, which increases the security of the system against packet sniffers.

1.3 File Request

After the server and client establish a connection, the client sends a file request to the server. This file request consists of a single TCP packet with the file name. This implementation is limited such that a file name must be smaller than the maximum segment size of a packet, which is 1011 bytes.

Once the server gets the file name, it looks in its own directory, finds the file, and then parses the file into packets. Since the parsing of packets is done before packets are sent out, packets are associated with sequence numbers as they are created rather than when they are transmitted.

1.4 File Transfer

After the server splits the file into TCP packets, it commences a Selective Repeat protocol with a window size of 10. Three threads are established - the main thread which handles sending the packets in the current window, a timeout thread, and a thread that receives acks. In order to implement the timeout and ack thread, it was necessary to have meta data about every data packet that was in the queue. The packet meta data structure contains a pointer to the packet that it refers to, the time that it was sent, and a Boolean representing whether or not the packet has been acked yet.

1.4.1 Timeout Thread

When the main thread sends out a packet, its packet meta data is updated with the time the packet was sent out using the C++ chrono library. This information is used in the timeout thread. The timeout thread sleeps for the re transmission timeout time (set at 500 milliseconds) which prevents wasting CPU cycles during redundant checks of the transmission times. Once the thread wakes up, it loops through the server's data structure of unacked packets. It compares every unacked packet's transmission time with the current time, and if that time is greater than 500 milliseconds, then it re transmits the packet.

1.4.2 Ack Thread

This thread's role is to receive acks. It initially blocks, because it calls recvfrom() which blocks the thread until an incoming packet is detected. Once the thread detects an incoming packet, it updates the server's meta data and marks the packet as acked.

1.5 Receiving the File

In order delivery of packets is not guaranteed. In order to ensure correct ordering of packets, first the client receives every packet and stores them in a data structure. Once the client has all the packets, it loops through every packet's sequence number to find the lowest sequence number. It then loops through the received packets, reordering them by received sequence number. Since the packets are now in order, the client should have the correct file and it writes the file to the current directory.

1.6 Tearing Down

Once the server sends the last data packet along with an EOF packet to the client, it initiates a tear down. It does this by sending a packet marked with a FIN flag. Our program doesn't allocate any heap variables or buffers, so the client and server doesn't have to do anything except send each other FIN/ACK packets when the client receives the FIN flag.

2 Implementation Difficulties

After creating a program that was able to send each other UDP packets, managing all of the information about every packet that was sent out became increasingly complex. It was nearly impossible to create this program without some sort of meta data structure about the server and the client. This programming project became a lot simpler once the packet meta data structure was created, so ever packet's ack and send time could be taken into account.