Sam Strecker
Iler Hoepner
CS462
4/7/21

Design Document

[The protocol to be used]

We will use a simple text-based UI in the terminal to allow the user to pick between 4 different protocols to choose from. We will have Go-Back-N (GBN), Stop and Wait (SNW), and Selective Repeat (SR). Each protocol slightly alters how packets interact with the sliding window. For example, with GBN if the server does not receive an ack from the packet before the timeout it resends all packets in the window. Even if the server has already received an ack for the packet. Contrary to GBN Selective Repeat only resends the packets that it didn't receive an ack for. Finally Stop and Wait differs the most as it's window size is 1, so an ack is needed for every packet before the window can slide. For noisy environments GBN and SR both work best. But if it's a high bandwidth low latency noisy network GBN is the best fit. While SR would fit better in networks trying to conserve bandwidth as less packets are needed to be retransmitted. Finally SNW is perfect for mission critical low bandwidth networks where preventing data loss is a major priority.

[The size of your packet]

The size of the packet will again be chosen by the user in the UI. They will enter a number for the size of the packet in bits. This will then be used as the max buffer size of our packets. This dictates how much information we can place in the header and message area of the packet.

[Frame format, i.e., type of header information you want to keep]

Our frame format will be as follows

```
socketBucket, frame, frameLength, 0, (const struct sockaddr *)
&serverAddress, sizeof(serverAddress));

socketBucket = socket(AF_INET, SOCK_DGRAM, 0);

frame[MAX_FRAME_SIZE];

frameLength = createPacketFrame(sequenceNumber, frame, data,
dataLength, eot);

createPacketFrame(int sequenceNumber, char *frame, char *data, int
dataLength, bool end);
```

SocketBucket stores the needed information for the datagram socket, frame is calculated by the packet size the user entered, frameLength is the size our message area will be, and the rest is boiler plate from what's needed in datagram connections.

[Sequence numbers (do not use infinite sequence numbers]

The range of our sequence numbers is calculated in the while loop that checks our read status. We use a ternary operator with boolean values 0 and 1 to check if our sequence will be able to fit and if we need to roll over and start again. Since this while loop is within the send while our numbers are calculated again once we hit the size limit so our sequence numbers aren't an infinite sequence.

[Error detection]

For error detection we are using checksums that we send the output in the ACK as a boolean 0 or 1 to tell us if the data has been modified. We do this with a count that is the size in bytes of the frame to check. For each byte until count equals 0, it adds the [value at frame] to the sum and increments *frame. If the sum overflows a 16-bit value (larger than 0xFFFF), it truncates the output back to 16-bits (using bitwise AND of 0xFFFF).

[The size of your sliding window]

The size of the sliding window will be chosen by the user with the text-based UI. For SNW since the window size will always be 1 we don't prompt the user to choose the size if SNW is selected. For all other protocols the user has free reign of window size. This allows the user to test and see how changing the size of the sliding windows affect performance.

[The time-out interval (static or dynamically determined)]

The time-out interval is again statically user defined at the beginning of the program with the text-based UI. The time is in milliseconds so a user who inputs 15 will have a 15ms timeout

[How to implement timeouts]

Timeouts are implemented by comparing the time stamps of transmission and waiting for the ack with the chrono library utilizing the high_resolution_clock. We then check if we are over the timeout period and if so we resend the packet that failed to be acknowledged.