Zachary Canann

CSE 160

11/18/2014

# Project 3

In this project two main components were added: one for setting up and tearing down a connection, and another for the transfer of data. For the set up, a SYN is repeatedly sent to the server until responded to by a SYN/ACK. The base sequence and acknowledgement numbers are established in this process. At this point data transfer has begun the client is established. The server however will not be established until the first packet of data arrives. To check if a connection has been in a state (such as having sent a SYN) for an unreasonable amount of time, there is a single timer that advances the elapsed time spent in the current state for each connection. If messages are sent but not responded to, eventually the connection will time out and close. The teardown works in the same way as the 3 way handshake, with an eventual time out if needed.

For the data transfer portion it was decided to use two parallel arrays for the sliding window. One containing the actual data, and another containing a boolean indicating if data exists in that location. I found this much easier to manage and debug than attempting to juggle variables such as the next byte expected, last byte sent, etc. Instead, these variables could be inferred from the array of booleans when needed. A very fast timer is constantly going off prompting the node to both read and send data, and of course this is only doable when there is data to read and if there is data to send. Data is popped off the receiving end as it is read, and popped off the sender end as it is acknowledged. There is a hard time out for sent data, and it will be sent again if not acknowledged. If however the destination reports a window size of zero, the sender will wait a small amount of time before sending one packet of data. The acknowledgement of this packet will let the client know the new window size if it has changed. This can be inefficient, as the window could be completely open and a single packet is sent to learn the window size. Alternatively the receiver could resend the most recent ACK with the new window size, however the first approach was chosen since it was more simple to implement.

## Problem 1

Choosing a random number has more benefits. For example the case where a node sends a SYN, sends data and crashes, reboots and sends a new SYN. If the sequence number started at 1, it is possible old data could be confused with the new connection, whereas it is less likely in a case where a random sequence number is chosen.

## Problem 2

In ideal conditions it would probably be best to have a receive buffer at least twice the size of a sending buffer. This allows for the receiver to accept a full send window, then report to the sender that it still has enough space for another window. If the receiver can process all of the information fast enough, the sender will then be able to constantly be sending a full window.

## Problem 3

If the source address and port ID are the same, only one connection will be opened at once and they will overwrite the old connection. If they use different ports each time they can cause the server to create a new connection for each, and these will remain open until they time out. The best way to deal with this is to limit how many SYNs can be accepted from a particular host within a certain time frame.

## Problem 4

Currently connections will not time out and will remain open. A reasonable way to deal with this is to prevent too many connections to a host and time out connections after not receiving data for too long.