David Gaulke

ICS 460 – Networks and Security

Project 2 Technical Paper

Several design questions arise when implementing Sliding Window protocol. Computer networks involve many parts, and to represent one involves many coexisting variables that all interact in different ways. In this paper, I will discuss my implementation and the reasons for the decisions that I made.

The application begins with a Sender and a Receiver. Each prompts the user to introduce network transmission errors which consist of dropping, damaging, and/or delaying frame delivery.

The Sender then prompts the user to select a file to transmit. The Receiver creates a ServerSocket to listen on port number 49152. The Sender connects to receiver and delivers the file as a stream of into the Socket as a stream of bytes. The Receiver reads a stream of bytes from the resulting Socket connection and writes it into an output file. I chose to echo a subset of the Java Networking API because it seemed like the most desirable way to abstract away the transmission details. It also gave me the advantage of being able to test the Sender and Receiver classes with the real API before replacing it.

ServerSocket creates a Socket that knows what port to listen on, but not which address or port to send to. It waits until it receives a connection that carries this information. The connecting Socket knows what address and port to send to, but does not need to provide a port to listen on. It chooses one at random from the range of undesignated UDP ports. If it receives a BindException when it tries to listen, it tries again until an unused port is found. The Sender and Receiver now both have Sockets. The Socket receives bytes from the Sender until it has buffered enough for a complete Packet or until flush is called, then it creates a Packet and sends it to its instance of DataLink. Packet size is arbitrary and can be changed with option -p. I chose 1024 bytes, but in practice it would be a good idea to find a value based on how often frames are lost or damaged. If the packets are large, it is a waste of bandwidth to resend. When frames are received on the other end, the DataLink class passes Packets up to Socket, which feeds it to Receiver as a byte stream.

The DataLink class implements Sliding Window protocol. When it receives a Packet from a Socket, it assigns a frame numbers, piggybacks a frame acknowledgment, calculates a checksum, and hands it to an instance of PhysicalLayer to deliver to the recipient. PhysicalLayer returns frames from the other end to DataLink, which reads the sequence and ack numbers and acts in accordance with go-back-n protocol. The amount of sequence numbers is the same as the window size. If there were fewer, it would be possible for messages to get confused. There could be more, but it would not be necessary. I chose a window size of 8, which is arbitrary in the simulation, but in practice it is best to use as many frames as allowed by the network. It can be changed in the Sender with option -w. The Receiver will learn the window size from the first frame that it receives.

In order to implement timeouts for resending of frames, I created a Clock class that exists as a member of DataLink. Clock provides an interface for DataLink to start and stop timers. If Clock has no active timers when one is registered, it starts a new thread to wait out the timer. If additional timers are added, the thread keeps running until they are all expired. If a new Ack timer is added with a shorter duration than the existing timer set to expire, it interrupts the thread and inserts itself in the first position. I chose a timeout value of 10 seconds because frames are only sent every 2 seconds and this allows it time to catch up. The timeout value can be set with option -t.

At the bottom, the PhysicalLayer class listens for Datagram packets and passes them as frames to DataLink. It also receives frames from DataLink and transmits them to the other end. PhysicalLayer contains methods to set the likelihood that each type of network error will occur.