CSE 473 – Introduction to Computer Networks

Lab 4

Due 11/15/2017

As usual, review and follow the general instructions included in the Lab 1 writeup.

In this lab, you will be implementing a reliable data transfer protocol and testing it in ONL. The protocol is a sliding window protocol, similar to those discussed in Kurose and Ross, and includes the go-back-N feature. You will be provided with the basic framework for the protocol, but you will need to implement the mechanisms for detecting lost packets and retransmitting them, as needed. Unlike the descriptions in Kurose and Ross, your protocol will support data transfer in both directions. That is, both endpoints can operate as both "senders" and "receivers".

The provided framework is structured in three layers, an application layer, called *SrcSnk*, a transport layer called *Rdt* that implements the reliable data transfer protocol, and a lower layer, called the *Substrate* which deals with the socket communication. These three layers are each implemented as a separate java modules, each with their own threads, and they communicate with each other through queues. You will be provided with the *SrcSnk* and *Substrate* modules, as well as a skeleton of the *Rdt* module. There is also a "main" module called *TestRdt* that handles command-line arguments and launches the others. Finally, there is a *Packet* module that provides methods for accessing and setting packet fields and packing the packet fields into a buffer that can be sent over the network. You will find additional details in the provided source code, which is located in the *lab4* folder in your repository.

To complete the *Rdt* module, you will need to add a *send buffer* that stores packets that have been sent, but not yet acknowledged. The send buffer can be implemented as an array, with the index of each entry being its sequence number. You will also need to add a *receive buffer* for packets whose payloads have not yet been passed up to the application layer. Because packets can be lost and the payloads must be delivered in order, the receive buffer may need to hold a number of packets equal to the window size. You can implement the receive buffer as an array, indexed by sequence numbers.

Lastly, your go-back-N feature should implement cumulative ACKs (naturally) and triple-duplicate ACKs, *i.e.,* trigger a retransmission after three duplicate ACKs. Go-back-N has subtle behavior, so pay close attention to its specification. In particular, there is a single timer that triggers retransmissions and is reset whenever pending ACKs are received. In addition, while you will be using triple duplicate ACKs to detect lost packets early, because go-back-N retransmits all pending packets (up to a full window), you need to exercise some care not to repeatedly retransmit all those packets as you keep getting more duplicate ACKs, *e.g.,* by setting a flag that prevents such successive retransmissions and that is reset by a time-out or the receipt of the ACK for the missing packet.

The lab includes completing the *Rdt* module, demonstrating that it works correctly and then performing a series of experiments to get a better understanding of its performance characteristics. Detailed instructions are contained in the lab report template.