CSE 473 – Introduction to Computer Networks

Studio 4

(*Adapted from Jon Turner's and Roch Guerin's Studios*)

In this studio, you'll be building a program that monitors a running application and prints out a short report when the application terminates. The *Monitor* is implemented as a Java module with its own thread and it operates between two other modules, an *application module* that generates packet payloads and receives payloads sent by a remote peer, and a *substrate module* that handles socket IO.

In your svn repository, you will find an ONL configuration file and several Java files, *SrcSnk.java, Monitor.java, Substrate.java, TestMonitor.java* and *Packet.java*. Although we won't be using *onl* for the first part of the studio, one person in each group should still login to *onl* and commit the reservation for your assigned account. This must be done in the first 30 minutes of the studio, or *onl* will discard the reservation. In the first part of the studio, you will be working with the student sitting next to you.

1. Review the source code with your neighbor. Make sure you understand what all the parts are doing and how they fit together. Make a diagram that shows a client and server instance of the program and shows how they communicate. Your diagram should include blocks for the *SrcSnk*, *Monitor*, *Substrate, Sender* and *Receiver* modules.
2. Next, fill in the missing parts of the *Monitor*. Your finished program should keep track of the number of packets and chars sent and received. It should include a copy of the number of packets and chars sent so far in each packet sent. At the end, it should print a short report summarizing the number sent, the number received, and the number that were lost in transit.
3. Test your program running on a single computer (say shell.cec.wustl.edu). Both of you should login to the computer where you're going to do your testing. One should start the server, and the other start the client. Try different values for the command line arguments and observe the results. Fix any problems you discover in your program. While doing this testing, choose a server port number that is not being used. You can use *netstat* to check if a given port number is in use. After you are satisfied that things are working correctly, run the provided *script0* as well (note that *script0* expects the server port number as a command line argument).
4. In this part you will be testing your program in ONL. Each pair should create a directory to work in. Name these directories studio4a and studio4b. Take turns with the other pair in your group to carry out this testing. You should test both versions of the *Monitor*. Run the provided *script1* to verify that things are working correctly in this environment (*script1* is similar to *script0*). You will need to edit *script1* to use the proper directory.
5. Once both versions of the program are working correctly, you should test them together. That is, you should use the program produced by one pair as the server, and the program produced by the other pair as the client. You will have to modify *script1* to allow you to carry out this test. Correct any problems you run into. Make sure that the pair of programs works the same way no matter which of the two is the server and which is the client.
6. Start an *iperf* udp server on *h6x1* using the provided *udpRcvr* script. Then run *script2* (after editing the directory names) with an argument of 1.5 (*script2* runs *TestMonitor* and *iperf* simultaneously and this argument is the *iperf* client's sending rate). Notice what happens on the monitoring displays. What percentage of the *TestMonitor* traffic sent gets lost when running *script2*? What percentage of

the *iperf* traffic is lost? Are they the same or different? Any idea why? Adjust the *iperf* sending rate until there is no queueing on the inter-router link. What is the largest *iperf* sending rate that you can sustain without any loss? If you increase the *iperf* sending rate to 3, what percentage of the *TestMonitor* traffic is lost? What percentage of the *iperf* traffic is lost?