COMP 6461 - Fall 2012 Laboratory Assignment 3

Due: November 30, 2012
Demonstration Dates: On or before December 7. You must schedule the demo time with the lab instructors.

File Transfer Protocol Using Windowing

Specifications and requirements

In the first laboratory assignment, you used the TCP protocol to transfer a named file between two partners logged in at different computers. In the second, you used a simple Stop-and-Wait protocol over UDP as a data delivery service, to examine the use of timers, with an intermediate router to mangle the packets. You also explored the effect of the drop rate on the performance of the protocol. In this laboratory assignment, you are to extend the capabilities of your protocol, by introducing Continuous RQ transmission (i.e., sliding window transmission), with Go-Back-N error recovery. The sliding window allows you to achieve error control with higher efficiency than that available with Stop-and-Wait. You will use a simplified version of Go-Back-N, so that it is not necessary to manage a list of timers. You will also continue to use UDP as your "delivery service", and to negotiate the starting sequence number, as you did in the second lab. The starting sequence number will be over a larger range, see below.

The basic scenario unfolds in exactly the same way as it did for laboratories 1 and 2. There should be no difference visible "from the outside". In particular, the requirement to be able to both send ("put" a file) and receive ("get" a file) is maintained.

You must write your program in C/C++, and you must be prepared to demonstrate your program in the XP environment in the lab. You may develop your programs elsewhere, but the demonstration must take place in the XP lab. In particular, Java and/or UNIX versions of the programs are NOT acceptable.

Topology

The role of the **sender part** (of the client or the server) consists of: (1) forwarding the data frames to the peer host; (2) protecting the *oldest* data frame with a timer for resending in case of loss; (3) reacting to ACKs and NAKs returned by the receiver, and (4) maintaining a log of its transactions.

The role of the **receiver part** (of the client or the server) consists of: (1) receiving the data frames from the sender; (2) issuing ACKs to the sender for acceptable frames, and NAKs for out-of-order frames; (3) maintaining a log of its transactions. Since the receiver part is always responding to a received message, the sending of which is protected by a timer, there is no need to use timers in the receiver part.

The dropping of packets by the router is meant to simulate the condition of a checksum not being correct or a packet being lost on the Internet. Since you are using a Local Area Net, the likelihood of such loss actually occurring is negligible, so we have to introduce artificial loss using the router (as we did in Lab 2). If a packet is dropped by the router, the arrival of the *next* packet at the receiver will cause it to issue a NAK because of the out-of-sequence packet. (Note that this could not occur in the second lab, because you never had more than one packet outstanding.) If a packet is delayed by the router, then mis-ordering may result. If delay is turned on in the router, the delay will not mis-order the packet by more than one (otherwise, there is no hope of your protocol working!).

Note that is also true that a retransmitted packet (or acknowledgement) may be subject to loss.

Finally, note well that the timer in the sender part protects the *oldest* data frame. For the specific case where the last packet sent is not protected by a timer, this protocol can fail. You must propose and implement a solution to this problem.

You may have to extend your protocol data structures, to accommodate new features.

The main parameters of your protocol are the window size, the packet size, and the timer time out value. You are required to do a study of the effect of drop rate and packet delay on the performance of your protocol, for a variety of window sizes. For maximum window sizes from 1 to 19, with step size 2, and for drop rate percentages of 5% to 50% (in steps of 5%), run an actual file transfer, and then compute the ratio of the minimum number of packets required to send your file to the number of packets actually sent. (This is the same calculation as for lab 2.) For each window size, plot the ratios against the drop rate at the router site. Repeat the experiment for delay probability percentages of 5% to 50% (in steps of 5%), holding the drop percentage at 0%, and varying the maximum window size from 3 to 19, with step size 2. Provide a brief commentary on your results (1-2 paragraphs) in a separate file which also contains the graphs and the tables with the obtained values. You have to submit this file along with the source code of your program.

In order to obtain reliable results, you have to use a file big enough to accommodate at least 5 windows of maximum size (19) with your chosen packet size. I suggest that you use 80 bytes as your data size for the frames. You may wish to use a smaller data size (e.g., less than 80 bytes) to observe the window rollover on your terminal while you are debugging. Start *small* (i.e., with a small file to send), and then increase the file size as you develop confidence in your solution. For the demonstration, the size of the file

should be large enough to guarantee that the sequence space is exhausted many times (at least twice).

For window size 1, packet delay must be disabled at the router. (This is because a window size of 1 is unable to deal with packet inversion.) This is why you are not asked to do the second experiment for window size = 1. For the larger maximum window sizes, packet delay must be enabled at the router (in the second experiment). You must ensure that your receiver part responds correctly to a late packet, for these cases. (If it is the one that you expect, respond positively; otherwise respond with a NAK or by discarding it, as appropriate).

There is a strong interaction between the sending of new packets and the receiving of the ACKs/NAKs. If you favor the sending of new packets, you will send all your packets until the window is full, then respond to the series of ACKs/NAKs. If you favor the receipt of ACKs/NAKs, then you will never get "ahead" of the current packet, which implies that your solution reduces to Stop-and-Wait. To ensure that you observe *both* cases in your simulation, you will have to cause the generation of new packets to "speed up" or "slow down" at the *source* of the packets (i.e., in the part of your code that reads the file that is to be sent). You should expect to be required to demonstrate *both* cases to your laboratory assistant (for one choice of window size, greater than 1) when you demonstrate this lab.

It is not acceptable to "cheat", and put a large sequence number somewhere in your packet! Your code must work correctly with the desired maximum window size (1, 3, 7, ...).

Deliverables & Demonstration Notes

The deliverables are in two parts:

1. Submit your assignment electronically.
2. Demonstrate that your two programs exhibit the required features. During the demo, only some basic messages should be displayed on the screen (of each end host) before the file transfer begins and after it has ended. While the file transfer is in progress, one sample of data content has to be displayed on the screen and another one stored in a file at the reception point. For these demos, you will need to log into any machine at the lab and prepare your assignment to run. You should pre-compile your programs, so that the demonstration time will be minimized. You will probably need different variants for each window size; the choice of a window size can be set using "#define". The marker will then go to your machine to perform the demo.

Groups

For this lab assignments, a team of two is allowed; and no bonus is given for working alone.

NOTE: PLEASE do NOT make use of any "program generator" for this assignment. The purpose of the assignment is to ensure that you know how to write network programs using the socket interface, not to demonstrate your skills in fancy C++ programming. Please write in the most basic of C or C++ styles, and produce a program that would run in a plain DOS or Unix environment.