Programing Project #3 – Implementing Go-Back-N

In this programming assignment, you will be writing the sending and receiving transport-level code for implementing a simple reliable data transfer protocol. I recommend working in two phases, first implement basic communication (without dealing with any loss or corruption) and once that is working expand it to use Go-Back-N. I hope this project is fun, or at least interesting, since your implementation will differ very little from what would be required in a real-world situation.

We will be using a network simulator originally written by the textbook authors to approximate the behavior of a real-world network. The programming interface provided to your routines, i.e., the code that would call your entities from above and from below is very close to what is done in an actual UNIX environment. (Indeed, the software interfaces described in this programming assignment are much more realistic that the infinite loop senders and receivers that many texts describe). Stopping/starting of timers are also simulated, and timer interrupts will cause your timer handling routine to be activated.

Requirements

To receive full credit for the assignment your code must implement the Go-Back-N reliable data transfer protocol as described in class and the textbook using the provided simulator. Your solution must deliver all packets sent by the application layer on side A to the application layer on side B. You need to detect and correct for both loss and data corruption.

The simulator requires five command line arguments:

1. -n <number of messages>
2. -l <probability of loss>
3. -c <probability of corruption>
4. -t <average delay between messages sent by the application layer>
5. -d <debug level>

To simulate 10,000 messages with 1% probability of loss,1% probability of corruption, and average arrival time of 1 and the maximum debugging level you would use:

```
$> ./GoBackN -n 10000 -l 0.01 -c 0.01 -t 1 -d 5
```

Your solution must be able to handle any combination of input values. **You must assume that you only have space to cache 10 messages on the sender.**

The rubric with details of the grading criteria is included with the assignment on Canvas.

Getting Started: Download and test the simulator.

The code provided by the authors was in an antique K&R C format - replete with global variables and untyped functions. I've cleaned it up a bit and converted the simulator itself into a C++ class. The code is now broken up into four source files:

- The files **simulator.cc** and **simulator.h** contain the code for the simulator. You should not modify anything in these files. While reading over these files might help you understand how to use the simulator, the only thing you must know about these files are the structures "msg" and "pkt".
- The files **main.cc** and **main.h** contain some utility functions I added. You should not modify anything in these files.
- The files **GoBackN.cc** and **GoBackN.h** are the files you will modify.

The tabrall on Canvas contains these files along with a Makefile. Download the tarball, unpack it and compile by typing "make" inside the directory. You can run the program, but it will not send any data.

Helpful Information

- The document titled "My Approach to Project 3" as well as the recording of the lecture from October 15th provide detailed information about the simulator and one approach to the project.
- You should be able to complete this assignment by implementing the GoBackN algorithm exactly as it appears on in figure 3.20 from the textbook.
- You will need to find a way to estimate the Round-Trip Time so you can set the timeout correctly. The method **simulation::getSimulatorClock()** will return the current time inside the simulation. You can use that to find out when you send a packet and when the ACK arrives back.
- You will need to write your own checksum algorithm to detect corruption.
- There is a working version of the program in **~promig3/pub/bin/GoBackN**
- There is a small perl script name stressTest.pl in **~promig3/pub/bin** as well. This script runs the program under various conditions and checks the output. If all the packets arrive intact and in order it indicate that the test passed.

Extra Credit

For 10 points of extra credit you can modify your program for bi-directional communication - that is add code so that side B can and does send data to side A at the same time side A is sending data to side B. Note that you need to change the value of BIDIRECTIONAL in simulator.h from 0 to 1 for the simulator to generate messages on the B side.

What to submit

This assignment is due by the end of the day on Sunday, November 7th. There will be a 5% per day penalty for late submissions with a maximum late penalty of 40% (but remember we need time to grade your submission before the end of the semester).

You should submit a single tarball of a single directory containing a makefile, a README.txt with your name and any information I need to compile the program and the source files needed to build your program. The single directory must be named with your username. The grader should not need to do anything other than untar your files, cd into your directory, type make and begin testing.

Do not include any core, object or binary files in the tarball. The Makefile provided with the sample code includes a target named submit that will create the tarball in the format we are looking for. All you need to do is:

```
$> make submit
```

In addition to functionality, you will be graded on the quality of the code, including readability, comments, and the use of proper programing practices.