

# 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 the Alternating-Bit-Protocol version 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.

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

## Details

The details of this assignment (and the original source code for the simulator) are provided by the textbook authors. You can find the original text of the assignment on the companion website (http://media.pearsoncmg.com/aw/aw_kurose_network_3/labs/lab5/lab5.html ). This document contains some very useful information about how to work with the simulator.

I recommend approaching the project in the following steps (you don't have to follow my suggestions as long as the final submission works).

## Step 1 – 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. In fact, the only thing you to look at in 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 project.cc and project.h are the files you will modify.

To help you get started I suggest doing the following:

1. Look at the two structures at the top of the simulator.h file (msg and pkt).
2. You will make all your changes in the project3.h and project3.cc files. Don't change the other files.
3. The function `A_output(struct msg message)` is called each time the application has data it wants you to send to the B side. To get you started I've already added code to A_output so that it:
    1. Creates a new packet using the struck pkt datatype. (At this point the value for seqnum, acknum and checksum don't matter, so I just set them to zero).
    2. Sends the new packet to the network for delivery to the other side using the function void tolayer3(int AorB, struct pkt packet). Note that the authors provided two defined constants "A" and "B", so you can call the function with something like "tolayer3(A,packet)"
4. When the packet arrives on the B side, the simulator will call the function B_input. To get you started I've already added code to B_input so that it:
    1. Takes the data out of the packet and copies it into a message.
    2. Sends the message to the application by calling tolayer5(B,message);

At this point if you run the program with no corruption or loss you will see the data arriving at the B side. If you run the perl script I have provided called "stressTest.pl" the program should pass the first test.

## Step 2 - Implement the alternating bit protocol.

Now you are ready to implement the alternating bit protocol. You will need to add code that buffers the messages sent from the application layer to your transport layer. I would start by doing something like:

1. Create a structure to hold the messages that will be sent to the B side. I recommend one of the standard C++ container classes. This will be used to store messages you get them from the application while you are waiting for confirmation they have been received on the other side.
2. Add sequence numbers to the packets you send to the B side. Since this is alternating bit you can just use 0 and 1.
3. On the B side, created and send back and ACK packet each time you receive a packet. To send data from the B side to the A side you simply call "tolayer3(B,packet)"
4. On the A side, the simulator will called "A_input()" each time it gets an ACK packet. You need to modify the "A_input() function to check and see if there are messages waiting to be sent, and if there are send the next message in a new packet each time you get the ACK packet.

As before, if you run the program without loss or corruption you should be able to see the ACK packets arrive at layer4, and can verify that your code is working.

## Step 3 – Add timers and deal with loss.

Once you are sure your algorithm sends all the data correctly, you need to turn on the timers and start dealing with lost packets. Since we are not year dealing with corruption and this is an alternating bit protocol the only way to detect loss is when a timer goes off.

1. You only need timers on the A side. You can set the timer by calling "starttimer(A,X);", where X is how long the timer should be. After X time steps the simulation will call "A_timerinterrupt()".For now just start the timer each time you send a packet.
2. Fill in the code for so that it retransmists "A_timerinterrupt()" the lost packet (i.e., the last one you sent).
3. Add code that stops the timer when the A side receives an acknowledgment.

Your code should now be able to deal with packets that are lost.

## Step 4 – Add checksums

Once your code can deal with loss at the network layer, you need to develop a checksum algorithm. You can use any algorithm you like; it doesn't need to be too fancy. In your submission please include a README.txt where you describe what types of corruption your algorithm can and cannot detect.

Once you have a checksum algorithm you can detect corruption in the network. You need to modify your code so that it treats a corrupt packet the same way it would a lost packet (but you don't have to wait for a timmer).

## Step 5 – Convert to Go-Back-N

Finally, you need to modify your code so that it can pipeline multiple packets at a time (i.e., modify from alternating bit to Go-Back-N). You should be able to follow the algorithm described in the text.

## Extra Credit

For 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.

## Submission

This assignment is due by the end of the day on Sunday, November 5th. 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.

Don't forget to add a sentence or two in your README.txt file about how you do error detection and what kinds of corruption you can or can't detect.

Do not include any core, object or binary files in the tarball. Note: you use "make submit" it might create the submission file for you.

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