Program 3
Stop-and-Wait ARQ

Prof. Kredo

Due: By 23:59 Friday, November 14

Introduction

In this program you will accomplish several goals:

- Create a file transfer program over an unreliable network
- Implement reliable communication using stop-and-wait ARQ

All programming assignments must be done in pairs.

Requirements

This program has you write a socket program to transfer a file across the network. However, your program will experience random packet losses that you must overcome using stop-and-wait ARQ.

Your programs must transfer a file from the server to the client using the same procedures as Program 2 (i.e., using the same command line arguments, error conditions, etc.).

Your program will experience packet losses because you must use a special function, `packetErrorSend`, which has the same arguments as `send(2)`. `packetErrorSend` has the same functionality as `send(2)`, except:

- it randomly drops packets, but returns a non-zero value
- it sends at most 1300 Bytes

A header file and library are available on Learn to enable you to use `packetErrorSend` in your program. You **must** use the `packetErrorSend` function; any program that uses `send(2)` or equivalent functions will receive no credit.

The implementation of `packetErrorSend` is available in a library file on Learn. Link the library with your code by using the `-L` and `-l` options to gcc/g++. (HINT: Put `-lpacketErrorSend` last in your compile command.) There are two libraries, one for 64-bit systems and one for 32-bit systems; copy the appropriate library for your system into the same directory as your code and rename it `libpacketErrorSend.a`. **Your Makefile must link to a file named** `libpacketErrorSend.a`**.** The OCNL 340 PCs are 32-bit systems and `jaguar` is 64-bit. You can use the `uname -m` command to determine your machine type.

Your programs need to perform timeouts while waiting for packets using `select(2)`. You must select an appropriate timeout for your program and document why you selected this timeout in your code. Do not actively measure RTT.

Additional requirements for this program include:

- Do not send small packets; it's just wasteful.
- You must include a Makefile with your submission that compiles both the client and server programs upon the command `make`.
  - The client program must be called `snw-client` and take three arguments. The first argument must be an IP address or hostname, the second argument must be a port number, and the third argument must be the file to retrieve from the server.

  - The server program must be called `snw-server` and take one argument, the port number on which to listen.
  - Have your code link against a library with the filename `libpacketErrorSend.a` in the same directory as your source code.
- Your Makefile should assume `packetErrorSend.h` is in the same directory as the sources files.
- Your programs must compile cleanly on the lab computers in OCNL 340 or `jaguar` and use the gcc/g++ argument `-Wall`.
- Check all function calls for errors. Points will be deducted for poor code.
- Put both group member names, the class, and the semester in the comments of all files you submit.
- Submit your program files through Learn. Do not submit copies of the header file or library for `packetErrorSend`.

Input/Output

The input and output of the client and server for Program 3 should match those for Program 2.

Grading

Your program will be graded based on the following, which is also a suggested order of implementation.

- 10 points: Filename sent to server
- 10 points: Partial (due to loses) file transferred to client
- 20 points: Acknowledgments sent
- 25 points: Stop-and-wait used to recover from packet losses
- 15 points: Packet duplication prevented
- 20 points: Code quality and neatness

Hints

- Make no assumptions about the file contents. Any bit pattern may appear at any point in the file. **Do not use string functions to handle file data!**
- Test and debug in steps. Start with a subset of the program requirements, implement it, test it, and then add other requirements. Performing the testing and debugging in steps will ease your efforts. In general, the Grading requirements are ordered in a way to ease this implementation process.
- Your programs (client and server) should close when the file has been transferred. The answer on how to do this lies in `recv(2)` (HINT:"orderly shutdown" means to close a connection).
- Use Wireshark to see what your programs actually send and receive.
- C++ programs will need to use

```
extern "C" {
    #include"packetErrorSend.h"
}
```

- Programs that are functional, but implement fewer requirements are likely to earn more points than programs that are not functional and attempt to implement many requirements. Programs that do not compile or do not run correctly will receive almost no credit.
- You may have an optional command line argument that enables debugging output (for an example, see the server program from Program 1). The debug argument must be optional (meaning that your program must run when it is not specified) and the debug output in your code must be cleanly organized.