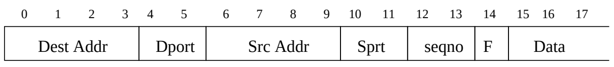
Programming Assignment #2 - HW #4

COMS W4119 - Computer Networks — Due Mar 6, 2006
Spring 2006 — Prof. Rubenstein

Programming Assignments must be e-mailed to the TA. The e-mail should contain a .zip file containing all files to be turned in. If desired, answers to questions can be turned in at the beginning of class on the day that the homework is due. CVN students have one additional day. Late assignments will not be accepted.

In this programming assignment, you will build sender and receiver implementations of Go-Back-N and Selective Repeat Reliable Transfer protocols. While full-duplex reliable delivery is implemented within TCP in the transport layer, we will design a half-duplex reliability protocol on top of UDP in the application layer.

Packet Formats



Each data packet you transmit should contain the following fields (in order, as pictured above):

- Destination IP Address: 32-bit, network byte order
- Destination Port: 16-bit, network byte order
- Source IP Address: 32-bit, network byte order
- Source Port: 16-bit, network byte order
- Sequence number: 16-bit: network byte order
- Flag: 8-bit: use described below.
- Data: up to 100 bytes in length. You can assume ASCII data with a byte value of 0 as a delimiter for the end of the data in a packet, and a byte value of 1 for the end of the transmission.

Note that an ACK packet should have the same format, except you can exclude the data field.

In order to test your program, we will provide you with an intermediate transmission point that implements a "lossy channel" that selectively drops packets. In other words, your receiver should reside at the destination address and port that you claim in your packet. But you should actually send the packet to another address/port that we will provide. This other address/port will then either forward your packet to the receiver address you specified (without further modification) or drop the packet.

Code details

You should create 4 program executables:

- `sender-GBN`: a sender that implements Go-Back-N. The program should take seven parameters:

  1. the window size (in packets) that the sender will use
  2. the time (in tenths of seconds) of the timeout timer for each transmitted packet (10 tenths recommended as a default)
  3. the name of a file. This file should contain ASCII text that the sender will transmit.
  4. the IP address of your receiver (using the standard representation for IP, e.g., 128.59.12.12)

5. the port of your receiver
6. the IP address of the lossy channel
7. the port of the lossy channel

- `receiver-GBN`: a receiver that implements Go-Back-N. The program takes a single parameter, which indicates the port to which the receiver should bind (should match parameter 5 above). You may decide whether or not the receiver buffers packets that are received out of order. The receiver should output to `stdout` the received data that was sent by the sender (i.e., the file should be output at the receiving end).
- `sender-SR`: a sender that implements selective-repeat. The set of parameters is the same as `sender-GBN`
- `receiver-SR`: a receiver that implements selective-repeat. The set of parameters is the same as `receiver-GBN`

Testing/Verification

As mentioned above, we will provide a "lossy channel" application to which you should actually forward your data when you want to test your program. How this lossy channel handles your data will depend on how you set the flag field in the packet. Note that it is important that you send the data and ACKs to the lossy channel (i.e., you use the lossy channel address and port in your call to `sendto`. It is also important that you use the right addresses of your sender and receiver in the source and destination fields in the packet, as this is where the lossy channel will forward packets that are not dropped.

The webpage `http://www.cs.columbia.edu/∼danr/4119/pa2-test.html` will contain additional information that we don't yet have available (e.g., address and port numbers of the lossy channel).

- Set to byte-value 0: the packet will be dropped
- Set to byte-value 1: the packet will be forwarded
- Set to any other value: the decision to drop the packet will be random.

The tester will also truncate your packet so that no more than 100 bytes of data can be forwarded within a given packet (including the end delimiters), so make sure not to try and forward more data than this in the packet.

Be careful: if you have a packet where the flag byte is set to 0, and you keep trying to retransmit this packet without modifying the byte value, then the packet will never make it through and the transfer will stall.

Note that you should also be able to connect your sender and receiver directly (i.e., make the 4th and 6th address parameters the same, and the 5th and 7th port parameters the same). It is unlikely that packets will be dropped, however, as networks these days exhibit very low loss.

What to turn in

The assignment should be turned in electronically, by e-mail to 4119-submit@cs.columbia.edu. You must turn in **as a single attachment** all work wrapped up together as a single gzipped tarfile with filename `pa2-CUID.tgz` (where `CUID` is your Columbia UNI). The tarfile should decompress to create a directory `pa2-CUID/` with the following files (example given in C):

- `sender-GBN.h`: header information
- `sender-GBN.c`: main code.
- `receiver-GBN.h`: header information
- `receiver-GBN.c`: main code.
- `sender-SR.h`: header information
- `sender-SR.c`: main code.

- `receiver-SR.h`: header information
- `receiver-SR.c`: main code.
- If you have additional c code that you wrote that is included by these files, include that code as well.
- `sender-GBN`: the GBN Sender executable compiled from the code above that **must** execute on CS Linux machines.
- `receiver-GBN`: the GBN Receiver executable compiled from the code above that **must** execute on CS Linux machines.
- `sender-SR`: the SR Sender executable compiled from the code above that **must** execute on CS Linux machines.
- `receiver-SR`: the SR Receiver executable compiled from the code above that **must** execute on CS Linux machines.
- `Makefile`: your makefile or `make.txt` that explains the procedures you followed to compile your file.
- A file named `test.txt` that includes some tests on which you ran your code.

`tar` is a Unix command. Suppose your current directory is `pa2-CUID`. Then

- `cd ..`
- `tar fvczh pa2-CUID.tgz pa2-CUID`

will generate the desired tarfile. To verify this worked correctly, create a directory `temp/`, enter that directory, and type `tar xvfz pa2-CUID.tgz` and the directory `pa2-CUID` and all its contents should be created (i.e., copied) into `temp/`.

Grading

Your grade will be based on *our* execution of your code against some pre-determined test cases (that may be slightly different) from the tests we provide above. So make sure that your code works, or we will not be able to test. We will also randomly spot-check the source code turned in to verify that it matches the object code, and will also compare object code across student assignments.

Again, *the code must execute on Columbia CS Linux boxes to receive credit.* More details on this forthcoming (e.g., a specific machine on which it must work).

Additional Comments

- You may choose how to do your own sequence numbering (e.g., whether to count packets or bytes, what sequence number to start with).
- Note that we skipped the connection setup phase where the sender and receiver agree on the starting sequence number. This value will therefore have to be hardcoded (since the sender's first data transmission might get lost).