

## Programming Assignment 2

Assigned: March 27, 2009

Due: April 13 (progress report) and April 20, 2009

In this programming assignment, you will implement a reliable data transfer protocol using emulab. You can modify the UDP echo server (UDPEchoServer.c) and UDP echo client (UDPEchoClient-Timeout.c) codes from the “TCP/IP Sockets in C” book. **You should not use TCP at all.** You can use the UDP echo client as the sender and the UDP echo server as the receiver. If you run the UDP codes on a pair of machines in the multilab, most likely you will not see any loss. Therefore, we need an environment that we can control the loss rate and create loss scenarios to test your codes. Fortunately, UK Laboratory for Advanced Networking has an emulab facility that meets these requirements.

The protocol you will implement in this assignment is go-back-N.

**Sender or Client**

You can modify the UDP client code to send packets to the receiver, which is a modified version of UDP Echo server. The sender will send data in a buffer using UDP. Its interface should be:

```
myclient server_IP server_port chunk_size window_size
```

where `server_IP` is the IP address (starting with “192.”) of the server, `server_port` is the port number the server is listening to, `chunk_size` is the default value for the length of data in each packet and it should be less than 1024, `window_size` is the maximal number of packets the sender can send before receiving an ack.

To avoid the burden of reading data from a file, you can declare a variable called `buffer` of type character array and initialize it with a constant. Here is an example:

```
char buffer[4096]='‘Here is the message. Put at least two thousand characters here!!!’’;
```

The message sent to the receiver is in the following format: type (4 bytes, int), sequence number (4 bytes, int), length (4 bytes, int) and data (the size of the data is specified by `datalen`). The field `type` should be 1 to indicate it is a data message. You can define the following structure for the messages you will send.

```
struct {
    int type;
    int sno;
    int datalen;
    char data[1024];
}
```

However, the size of message you will send should be  $4+4+4+\text{chunk\_size}$ .

In GBN, assume “base” is the smallest sequence number within the window. All packets up to “base”-1 have been acknowledged by the receiver. The number of packets the sender can send is `window_size`. The packets in the window are from “base” to “base”+“window\_size”-1. After sending them, it will set a timer (3 seconds). Then it has to wait for the ack from the receiver or a timeout.

1. If it is a duplicate ack, just ignore it.
2. If it is a new ack, clear the timer, update “base”, and send **new** packets allowed by the “window\_size”; if there are packets sent out but not acknowledged yet, the sender should restart the timer.
3. If it is a timeout, it will resend all packets (retransmission) from “base” to whatever packet allowed by the “window\_size” and set a timer. After 10 successive retransmissions, it will give up and declare failure.

After receiving the ack for the last message, the sender will send a special tear-down message with `type=4` (other fields irrelevant). This message possibly needs to be retransmitted for at most 10 times unless it receives an ack message of `type=8` from the receiver. Then the sender can exit.

The sender must print out the sequence numbers of all the packets transmitted and retransmitted in the format of “SENDING sno” and the sequence numbers of all the acks received in the format of “---- RECEIVING ACK ackno”. It should also print out the transmission/retransmissions of and the acks received for the special tear-down message.

## Receiver or Server

The modified UDP Echo server is the receiver and its interface should be:

```
myserver port_no chunk_size
```

where `port_no` is the port number that the server will listen to, and `chunk_size` is the default value for the length of data in each packet.

The server will keep a counter called “packet\_rcvd”. Starting with “-1”. After receiving a packet from the client, it will check whether the sequence number is equal to `packet_rcvd+1`. If not, just discard the packet and send an ack with `ackno=packet_rcvd`. If yes, add 1 to `packet_rcvd`, put the data in the packet to the correct location of a receiving buffer, and send an ack with `ackno=packet_rcvd`. Upon receiving the tear-down message of `type=4`, it will send an ack message with `type=8`. After that, it will wait for 7 seconds and exit. During the 7 second waiting time, it will send an ack with `type=8` for any tear-down message of `type=4` received and ignore all other messages.

The message sent to the sender is in the following format: `type` (4 bytes, int), acknowledgement number (4 bytes, int). The field `type` should be 2 (or 8 for the ack for the tear-down message) to indicate it is an ack. You can define the following structure for the acks to be sent.

```
struct {
    int type;
    int ackno;
}
```

The receiver must print out the sequence numbers of all the packets received in the format of “---- RECEIVING sno” and the sequence numbers of all the acks sent (including duplicate acks that repeat previous ackno) in the format of “SENDING ACK ackno”. It should also print out the tear-down message and the acks sent for it.

## Emulab

You can find all necessary information for using the Emulab at <http://www.uky.emulab.net/>. Click on the Documentation on the left hand side of the page, you will find instructions such as “how to get an account” and “how to get started”. We will also spend a class to show how to create a topology and how to set parameters of the experiment.

### 1. Getting an account

You will need to apply for an account at the emulab. The name of the project you will join is *cs471*. I will send you the name of the group you should join by email later. Since the emulab is undergoing an update, you should not start this process until April 6.

### 2. Create an experiment

A simple way to create an experiment is to use the graphical interface provided by the emulab software. After you login from the above web site, you can “Begin an Experiment.” On that page, you can “go to the NetBuild GUI” to create a topology.

### 3. Login to the machines of the experiment

After you start (or swap in) your experiment, you can login to the machine. The machine name will be in the format: `node_number.experiment_name.project_name.uky.emulab.net`. You can find this information (actual number and names) in the email sent to you.

After you login to the machine, it is just like a standard Linux machine. You can write programs and run them on these machines.

Note (very important): When you use nslookup, you can find an IP address of the node. This IP address is for you to access the node from the outside of the emulab. However, in your programs, you should always use the IP addresses sent in the email and they should start with “192.168.”

#### 4. Change parameters (loss rate, delay, bandwidth) of the experiment

After you start the experiment, you can see your experiment through “Experiment List”. Go to the experiment you want to make changes, click on “Modify Traffic Shaping” and it will allow you to change parameters such as loss rate, delay, and bandwidth.

#### 5. Testing

Because we have a lot of students using emulab, you are advised to test your program first on the machines in the multilab. After you debug your programs and believe that they run correctly, you can move your programs to emulab. The current guideline is that you run your experiment for at most 2 hours and then swap out your experiment if the number of free PCs in the emulab is less than 6. As soon as you finish testing your programs, swap out your experiment. The experiment will still be there and you can swap it in later when you need to test your programs. Always make your experiment swappable.

### Submission

Your **first submission** will be due on April 13. You should create an experiment and get yourself familiar with the emulab environment. Login to those machines and run the sample UDP programs. What you need to submit is a one-paragraph description of what you have done and the printout of the email received from the emulab about the experiment you created.

Your **final submission** is the programs and related documents. You need to provide a makefile that can compile and link your programs. Comments are required. Also write a README file to give general descriptions about your programs and instructions to run them, and state any limitations of the implementation. Mail your programs as an attachment to [fei@netlab.uky.edu](mailto:fei@netlab.uky.edu). You should tar all the files together instead of multiple attachments.

Final note: Dr. Finkel provided some useful information about programming. It is available at <http://www.cs.uky.edu/~raphael/programming.html>.