Project 3: Reliable File Transfer Over UDP

Type of Project: Individual or Pair
Deadline: 2017-03-30, 11:59pm
Language: C/C++
Points: 25 points

Submission Guidelines: Submit as a single .tar.gz or .zip file via ELC. Name the project directory as "**LastNameFirstName**-udp_reliable_transfer" (e.g., "PerdisciRoberto-udp_reliable_transfer"). Submit the source code, includeing a "**Makefile**", so that the code can be compiled simply by executing 'make'. Also, make sure the compiler will output the following executable files

urft-client
urft-server

Along with the source code, you need to submit a document that describes **your client-server communication protocol** (e.g., format of the packet header, in what cases you send ACKs or NAKs, what is the length of time-out, etc.)
All files need to be under the same main directory (i.e., do not create subdirectories under "**LastNameFirstName**-udp_reliable_transfer").

NOTE: project submissions that do not follow the guidelines risk to be assigned 0 points.

Project Description: In this project, you are required to **design a reliable file transfer protocol** and **write a client application and a server application** that communicate with each other using UDP use a stop-and-wait approach. In particular, the client needs to contact the server, pass the file name to be uploaded to the server, and send the file to the server. The client-server communication **must withstand packet loss**.

To simulate packet loss, I will test your software using a "transparent proxy" [urft-proxy]. An example of how I will run your code is reported below:

urft-server <server port>
$ ./urft-server 20000

urft-proxy <loss probability> <proxy port> <server ip> <server port>
$ ./urft-proxy 0.10 10000 10.0.0.2 20000

urft-client <proxy ip> <proxy port> <file name>
$ ./urft-client 10.0.0.1 10000 file1.txt

Once the server is started, it will wait for file download requests. The proxy will forward any packets received on port 10000 to the server, and then it forward the server's response packets back to the client. The client will send a request to upload a file (e.g., "file1.txt") to the proxy, and the proxy will fortward this request to the server. Notice that from the client's point of view there is no proxy, meaning that the client acts as if it was directly talking to the server (i.e., as if the program listening on port 10000 was the server itself, and not the proxy). Also, from the point of view of the server, the

proxy is the client (i.e., the server does not know that the proxy is forwarding packet on behalf of the client).

NOTE:

- Max Datagram Size = 512 bytes. This is the maxium length of the UDP payload that you should use.
- Max Time-out = 1sec.

TESTING YOUR CODE

To test your code I will create two directories: "client_dir" and "server_dir". The "client_dir" will contain the client program "urft-client" and a number of file that need to be uploaded from the client to the server. The "server_dir" will contain the server program "urft-server". The files uploaded by the client need to be stored inside the "server_dir" directory (do not create any sub-directories under it).

Notice that you can run the server, proxy, and client on the same machine using the loopback address. For example, you can run the proxy and server as:

```
$ ./urft-server 20000
$ ./urft-proxy 0.10 10000 127.0.0.1 20000
```

Once the server and proxy have been launched, you can run the client as

```
$ ./urft-client 127.0.0.1 10000 test_file1
```

where 127.0.0.1 and 10000 are the IP and UDP port of the proxy, respectively. In this example, the file "test_file1" will be split into datagrams and sent to the server, which will sotre it in the "server_dir" directory (i.e., the same dir from which the "urft-server" is launched). You can then compare the md5sum of "client_dir/test_file1" and "server_dir/test_file1" to make sure they match!