COMP 6461 - Fall 2014
Laboratory Assignment 2

Due: Friday, October 24, 2014
Demonstration date (Will be announced by the lab instructors; reservation is required similar to lab1).

File Transfer Protocol Using "Stop-and-Wait" and one timer

Specifications and requirements

In the first laboratory assignment, you used the TCP protocol to transfer a named file between two partners logged in at different computers. In this laboratory assignment you are to explore the use of *select()* to manage timeouts. You will also use a random number generator on the forward and reverse paths to drop a percentage of the packets. (This random dropping will be performed by a separate "router" program, see below.) In addition, you will use UDP (connectionless transport) to send the packets. (This is more in keeping with the idea of sending "packets"; it will also enable you to read in the next "packet" as a single unit, without knowing how large it may be, up to a maximum size. Given that packets may be "lost", this ability will be important.)

To keep the requirements simple, you will implement a very basic "Stop and Wait" protocol, so that there is minimal concern with windowing. The Stop-and-Wait protocol operates as follows: Each packet of your private protocol (which you developed in Laboratory 1) is sent as a UDP packet. If no loss occurs (see below), the peer host returns an acknowledgement packet, again using UDP. Once this acknowledgement has been received, the next data packet can be sent. If a packet is lost (on the forward path or on the reverse path), then the sending host will time out, and send it again. You are required to maintain sequence numbers in your solution, so that the case where the data are delivered but the acknowledgement is lost can be detected (i.e., the second reception of the data can be discarded). Since a single bit is sufficient for the correct operation of this protocol, you should add a single-bit field to your private protocol to carry this sequence number. For information on the initial value of this one-bit field, see below.

The timeouts must be implemented for all packets: "control" packets (where you exchange information that manages the transfer), and "data" packets (which contain the actual file being transferred). This is done using "select()".

To give experience in the three-way handshake, you are required to code a three-way handshake when you establish the connection. Because producing a random number on [0..1] will not give you much protection against errors, you must draw a random number on [0..255] on the client and again on the server, and use these two random numbers in the three-way exchange. Once the three-way exchange has been completed, you will use the least significant "n" bits of the random numbers (one number at each end of the connection) as the initial values for the sequence numbers. For this lab, "n" is equal to

one. In the third lab, "n" will be larger. It is not acceptable to "hard code" the initial sequence number on each side. It is also **not** acceptable to carry a "larger" sequence number in your private packet; *you must make your code work correctly with a single-bit sequence number* (after the three-way handshake is completed). To give a specific example, if the client side produces the random number 33 and the server side produces the random number 240, then the client side will send "33", the server side will acknowledge "33" and also send "240", and finally, the client side will acknowledge "240". This establishes "1" as the sequence number of the "last correctly received packet" from the client to the server, and "0" as the sequence number of the last correctly received packet from the server to the client. Therefore, the client will use "0" as its initial sequence number for sending to the server, and the server will use "1" as its initial sequence number for sending to the client.

The basic scenario unfolds in exactly the same way as it did for assignment 1. There is no difference visible "from the outside". In particular, the requirement to be able to both send ("put" a file) and receive ("get" a file) is maintained. Note that for the case of "put", the "client" and the "sending host" are the same for both the initial negotiation and the file transfer, while for the case of "get" the client is the "sending host" for the negotiation phase, but becomes the "receiving host" for the file transfer. (However, please note the requirement to inform the client and the server about the host where the intermediate program resides; see the link below.)

To permit "simulating" the actions of the Internet (i.e., the loss that occurs in the real Internet), you will use a random number generator in a separate "Router" program; the specification of this program is located here. The C++ text of this program is located here. Its C++ header file is located here. In this way you will simulate both cases of interest: 1) the original packet is lost, and 2) the acknowledgement is lost. The supplied program allows the user to set the loss percentages at run time. For this lab, set the "delay" parameter to disable delay; the Stop and Wait protocol is not capable of dealing with delay. *You are not allowed to modify the router program.* The marker will use the code provided above, and not any version of the router program that you may submit, in testing your program.

Log of transactions

The development of any distributed application is a complex endeavor that is subject to many errors. For Lab.2, you will make use of a technique that consists of generating execution traces stored in files that can then be scrutinized for debugging purposes (and also coincidentally by the marker to evaluate the quality of your programs). Subject to a settable switch (such as #define TRACE 1), you will generate a separate log file for each member of the communication group (client, router, server).

For the sender, the log file should include a listing of all noticeable events. For example:

Sender: starting on host aztec
Sender: sent packet 0

Sender: received ACK for packet 0
...
Sender: file transfer completed
Sender: number of effective bytes sent: 10000
Sender: number of packets sent: 100
Sender: number of bytes sent: 15000

The log file for the receiver should be organized in the same way as the one for the sender. For example:

Receiver: received packet 0
Receiver: sent an ACK for packet 0
...

Receiver: received packet 1
Receiver: sent an ACK for packet 1
Receiver: transfer completed
Receiver: number of bytes received: 10000

Both the client and the server should keep such a log; each log will have interleaved "Sender:" and "Receiver:" entries.

The router log is kept by the router program (see its specification).

Implementation guidelines

This second laboratory adds a few programming features that were not present for Lab1. One interesting new programming feature consists of using a timer at the sender. You can accomplish this by using the select() system call, which allows you to wait for either an arriving packet or a timeout, and then specify the timer value and the socket descriptor from which the receiver's feedback is expected. As select() returns the number of descriptors that are ready, one can discriminate between the INCOMING PACKET case (select() returns 1) and the TIMEOUT case (select() returns 0). Again, remember that both the client and the server can act as the sender, depending on the direction of file transfer.

You may have to experiment to find a proper timer value to use, which depends upon the round trip time (RTT) (use 300 ms as a starting point).

The second major difference is the use of UDP as your "delivery service". UDP is a connectionless protocol, so certain of the system calls made for the TCP case are no longer necessary. In addition, there are system calls that permit you to retrieve the "next packet", however large this will be. This will (possibly) simplify your code, because you will be able to preserve the packet boundaries (which you could not do with TCP). Note that you will have to specify an input buffer that is "at least as large" as the largest incoming packet.

In order to limit the size of the log files, you should use a data buffer size of 80 bytes and a file size of about 2000 bytes for your experimenting. Presume a drop rate of 10%, for each of the receiving and sending paths, but be ready to experiment on this side also.

For the log file, the following code construct could be used:

```
if (TRACE) {
  fprintf(logfile, "Sender: sent frame %d\n", seqno);
}
```

Deliverables (See course outline for submission format. More details are also available at the course website. You need to look at these details/instructions prior to submitting your assignment):

The deliverables are in three parts:

1. Do a study of the effect of the drop rate on the performance of the protocol. For drop rate percentages of 5% to 50% (in steps of 5%), run an actual file transfer, and then compute the ratio of the minimum number of packets required to send your file to the number of packets actually sent. (The extra packets are the re-sent ones.) Use a large enough file that there is a meaningful result. Plot the ratios against the loss rate at the router site. Create a file with your brief comments (1-2 paragraphs), and the graph, for later submission. These results do not have to be available at the time of the demonstration, but it clearly will be to your advantage to have done this work before your demonstration, so that you are confident of your work.
2. You must verify that your three programs work as expected. During the demo, only some basic messages should be displayed on the screen (of each member) before the file transfer begins and after it has ended. While the file transfer is in progress, one sample of data content has to be displayed on the screen and another one stored in a file at the reception point. You should also be able to demonstrate the transfer of a binary file; in this case the screen display should be suppressed. At the time of demonstration, the lab instructor will ask you questions about the functioning of the program; any student of the group must be able to answer any question. Part of the marks will be assigned for demonstrating compliance with requirements at this demonstration. Marks assigned to each member of the group may be different, depending on ability to answer the questions.
3. Submit your performance evaluation file, the three programs and the log files using the electronic submission facility located at https://eas.encs.concordia.ca/eas/authentication.jsp.

Demonstration Notes

You should pre-compile your programs, so that the demonstration time will be minimized. Remember that you will have only about 10 minutes to do the entire demonstration. If doing a file transfer for 2000 bytes takes significantly longer than a few seconds, there is something seriously wrong with your code.

Groups

For this lab assignment (and for the ones to come), a team of two is permitted, and no bonus is given for working alone. However, if it is discovered that groups larger than 2 have formed, the penalty will be a mark of zero (0) for *all members* of such larger teams.

NOTE: PLEASE do NOT make use of any "program generator" for this assignment. The purpose of the assignment is to ensure that you know how to write network programs using the socket interface, not to demonstrate your skills in fancy C++ programming. Please write in the most basic of C or C++ styles, and produce a program that would run in a plain DOS or Unix environment.