CS3251 Computer Networks I

Fall 2010

Programming Assignment 2

Assigned: Nov. 2, 2010

Due: Nov. 23, 2010, 11:59pm

A Simple FTP Protocol:

For this homework you will implement a simple reliable file transfer protocol over UDP. You are responsible for achieving reliability over an unreliable service. The main objective of the assignment is to design and implement a stop-and-wait ARQ protocol and to experiment with the effect of different segment sizes and loss parameters on the performance of the protocol.

1- Implementation

You are to design and implement a reliable file-transfer application called KFT (in honor of Kermit, one of the first terminal and file transfer systems for PCs (see http:// http://en.wikipedia.org/wiki/Kermit_(protocol).) The KFT client can download a file from the server and save it in the local directory. A KFT server need only handle one client at a time, however, it should "run forever". In other words, the KFT server can be iterative rather than concurrent. The KFT client, on the other hand, should terminate after the successful completion of the download.

The KFT application should implement the basic functionality of a stop and wait transport protocol using sequence numbers, (positive) acknowledgments, retransmissions and timeouts. The application will be using UDP/IP sockets for transferring datagrams. Note that you should not use TCP sockets in this assignment.

To make things more interesting, a "packet dropper" will be introducing random packet losses. To use the packet dropper, you should use the function *sendto_dropper()* (provided by us), instead of the standard *sendto()* system call. The parameters of the *sendto_dropper()* function are the same as *sendto()*. You should initialize the dropper at the start of your program using the function *set_dropper()* that is also provided. For simplicity, these parameters are specified as integer percentages at the client and server command-line arguments.

The dropper.h and dropper.c files are attached along with this writeup.

You have significant flexibility in designing the underlying reliable transport functionality. You should however adhere to the following specifications:

- The file should be transferred reliably from the server to the client using a stop-and-wait ARQ protocol.

- The transfer should be reliable in the presence of packet losses.
- The maximum packet size will be a command-line argument at the client (in bytes). The KFT sender should be using maximum sized packets when possible.
- Both the server and the client should recognize an additional flag "-d" (i.e. debug mode) at the command line parameters. When running in debug mode, the two programs should report detailed information about their transfer status (e.g., sequence numbers of sent/received packets, retransmission timeouts, detection of corrupted packets, etc).
- Your client should be able to translate hostnames to IP addresses automatically, i.e., it should not matter whether we input a hostname or an IP address. Use *gethostbyname()* for the lookups.
- In your sender code you should start and stop a timer (hint: use *gettimeofday*()) to calculate the total elapsed time from the start of the transfer to the point where the final successful ACK is received from the server.
- The command lines of your client and server should look like:

>>kftserver <server_port_number> <loss_percent>
(e.g., kftserver 7777 3)

and

>>kftclient <server_name> <server_port_number> <remote_filename> <local_filename> <max_packet_size> <loss_percent>
(e.g. %kftclient ibis.cc.gatech.edu 7777 coolfile.txt mycopy.txt 1000 16384 2)

Note that the dropper can have different loss parameters at the client and server. The parameters specified at the client determine the error characteristics in the network path from the client to the server, while the parameters specified at the server determine the error characteristics from the server to the client.

2- Performance Evaluation

You have three parameters that can be tuned in your code: Client-to-server packet loss rate, serve-to-client packet loss rate and maximum segment size as indicated on the client command line. There is also another external parameter: the file size. Your task is to test how the performance of the file transfer protocol is affected by changes to these parameters. To this end you will report the results of four experiments:

1- Show how the transfer time (measured as instructed above) changes as the server-to-client loss percentage increases from 1% to 80%.
2- Show how the transfer time changes as the client-to-server loss rate increases from 1% to 80%
3- Show how the transfer time changes as the file size increases from 100Kbytes to 100Mbytes
4- Show how the transfer time changes as the maximum segment size changes from 100bytes to 50,000 bytes.

You should report the results of each experiment with a graph (use any available graphing software). Clearly label the x and y axes.

Each graph should include at least 10 points.

You should use the following nominal values for the parameters that are not being varied in your experiment:

Parameter	Nominal Value
File Size	10 Mbytes
Server-to-Client Loss Rate	5%
Client-to-Server Loss Rate	5%
Segment Size	1000 bytes

You are required to submit a 3-5page report that includes your graphs, description of the experiments, and discussion of any unusual events or bugs affecting your experiment. You should also discuss the effect of the parameter you are testing on the performance and provide an explanation for the observed effect.

Notes:

1. Your programs are to be written in standard C using the standard Socket libraries we have been covering in class. Your programs should run on the CoC Unix (or Linux) systems. **You must include a citation (text or web site) in your source code and project description if you use an external reference or existing code templates.**

2. Make sure that you do sufficient error handling such that a user can't crash your server. For instance, what will you do if a user provides invalid input?

3. You must test your program and convince us it works! Provide us with a sample output that shows all of your functions working.

4. You should implement two separate programs: a client and a server. Your final submission should include two executable programs. To simplify the grading process, please call the executables: *kftserver* and *kftclient*.
5. Focus on limited functionality that is fully implemented. Practice defensive programming as you parse the data arriving from the other end. Do not work on a fancy GUI, but focus on the protocol and data exchange. Make sure that you deal gracefully with whatever you or the TA might throw at it.

Submission Format:

Submit your programming assignments as a gzipped tarball (.tar.gz extension) with your Last Name as the file name. You must submit everything using t-square.

Your archive(.tar.gz) should include: your **well-documented source code**, a README file, a Makefile and sample output file called Sample.txt. The README file must contain:

1. Detailed instructions for compiling and running your client and server programs including a Makefile.
2. A description of your ARQ protocol (1-2 pages) with sufficient detail such that somebody else could implement your protocol
3. Any known bugs, limitations of your design or program

You will be graded on the correctness of the code and its readability and structure. As you write the documentation, imagine that I intend to implement your protocol in my own client that will interoperate with your server. Make sure we don't have to read your source code to figure out how to do this.

Grading Guidelines:

The grade will be split as follows: 70% for the program and accompanying documentation and 30% for the results of performance evaluation.

For the programming portion we will use the following guidelines in grading:

0% - Code is not submitted or code submitted shows no attempt to program the functions required for this assignment.

25% - Code is well-documented and shows a genuine attempt to program required functions but does not compile properly. The protocol documentation is adequate.

50% - Code is well-documented and shows a genuine attempt to program required functions. The protocol documentation is complete. The code does compile properly but fails to run properly (crashes, or does not handle properly formatted input or does not give the correct output).

75% - Code is well-documented. The protocol documentation is complete. The code compiles and runs correctly with properly formatted input. But the program fails to behave gracefully when there are user-input errors.

100% - Code is well-documented. The protocol documentation is complete. The code compiles and runs correctly with properly formatted input. The program is totally resilient to input or network errors.

For the experiment portion we will grade as follows:

60% - Graphs for all 4 experiments are provided and the graphs follow stated instructions

40% - Description of the experiments is complete and discussion of results provides insights.