CSEE W4119.001
Computer Networks
Professor Vishal Misra

Due: November 11th, 2013

Programming Assignment 2
Simple TCP-like transport-layer protocol

1. Introduction

In this programming assignment, you will implement a simplified TCP-like transport layer protocol. Your protocol should provide reliable, in order delivery of a stream of bytes. It should recover from in-network packet loss, packet corruption, packet duplication and packet reordering and should be able cope with dynamic network delays. However, it doesn't need to support congestion or flow control.

To test your implementation you will use a link emulator provided by us (see Proxy.zip attached; contains a README that guides usage). Data is exchanged via UDP, i.e., you will be running your TCP version "on top of" UDP. In the real-world, a network can drop, corrupt, reorder, duplicate and delay packets. To mimic this behavior you will use the link emulator. Specifically, you will invoke the emulator so that it lies in the sender-to-receiver path, whereas the acknowledgements (i.e., the packets on the receiver-to sender path) will be sent directly from the receiver to the sender, i.e., you can assume that packets arrive without a loss on this path. You can run your sender, receiver and link emulator either on one, two or three machines. The link emulator acts like a "proxy", i.e., the sender sends packets towards the emulator and the emulator should forward the packets to the receiver. The receiver process should send its acknowledgements directly to the data sender.

This programming assignment can be implemented in either of the following two ways.

a) As a stand-alone independent implementation.
b) As an extension to the Programming Assignment 1. (This will be considered only for extra credit)

You can use either **C, C++, Java or Python** your implementation, using standard UNIX or Java datagram socket calls (for brevity we refer only to UNIX network APIs in the assignment description).

2. Usage scenario and functionality requirements

You will implement a one-way ("simplex") version of TCP. Your program has to handle only one set of packets, i.e., the delivery of a single file. Your implementation will be composed of two separate programs, sender and receiver. The sender reads data from a file and uses the sending services of the TCP-like protocol to deliver it to the remote host. The receiver uses the receiving services of the TCP-like protocol to reconstruct the file.

The TCP receiver should be invoked as follows:

%receiver file.txt 20000 128.59.15.37 20001 logfile.txt

command line exec with filename, listening_port remote_IP, remote_port, log_filename

Delivery completed successfully

>% *program finished: shell prompt*

The receiver receives data on the *listening_port*, writes it to the specified file (*filename*) and sends ACKs to the remote host at IP address (*remote_IP*) and port number (*remote_port*). In the above example the ACKs are sent to the sender (128.59.15.37) with an ACK port 20001. The IP address is given in dotted-decimal notation and the port number is an integer value. The receiver will log the headers of all the received and sent packets to a log file (*log_filename*) specified in the command-line. The log entries should be ordered according to the timestamps of the packets, from lowest to highest, and should be displayed to the standard output if the specified log filename is "stdout". The output format of a log entry should be as follows.

timestamp, source, destination, Sequence #, ACK #, and the flags

The receiver should indicate whether the reception was successful, and report file I/O errors (e.g., 'unable to create file').

The data sender should be invoked as follows:

% sender file.txt 128.59.15.38 20000 20001 1152 logfile.txt

command line exec with filename, remote_IP, remote_port, ack_port_number, window_size, log_filename

Delivery completed successfully
Total bytes sent = 1152
Segments sent = 2
Segments retransmitted = 0

>% *program finished: shell prompt*

The sender sends the data in the specified file (*filename*) to the remote host at the specified IP address (*remote_IP*) and port number (*remote_port*). In the above example the remote host (which can be either the receiver or the link emulator proxy) is located at 128.59.15.38 and port 20000. The command-line parameter *ack_port_number* specifies the local port for the received acknowledgements. The *window_size* is measured in bytes and your sender should support values from 1-65535. As before a log filename is specified. The log entry output format should be similar to the one used by the receiver, however, it should have one additional output field (append at the end), which is the estimated RTT. At the end of the delivery the sender should indicate whether the transmission was successful, and print the number of sent and retransmitted segments. The sender should report file I/O errors (e.g., 'file not found').

3. Requirements

- You will implement a one-way version of TCP without the initial connection establishment, but with a FIN request to signal the end of the transmission.
- Sequence numbers should start from zero.
- You do not have to worry about congestion or flow control, and thus the sender window size should be a configurable command-line specified fixed parameter.
- You should adjust your retransmission timer as per the TCP standard (although it may be advisable to use a fixed value for initial experimentation)
- You need to implement the 20-byte TCP header format, without options.
- You do not have to implement push (PSH flag), urgent data (URG), reset (RST) or TCP options.

- You should set the port numbers in the packet to the right values, but can otherwise ignore them.
- The TCP checksum is computed over the TCP header and data; this does not quite correspond to the correct way of doing it (which includes parts of the IP header), but is close enough.

4. Tips

- You should test your programs with and without the link emulator. When using the link emulator, specify its IP address and port number in the command line of the data sender.
- Test your program over a wide range of network settings.
- Use diff on the sent file and the received file to check if they are the same.
- You can choose a reasonable value for the maximum segment size, e.g., 576.

5. Submission guidelines

Submission will be done by courseworks. Please post a single **<UNI>_<Language>.zip** (Ex. zz1111_java.zip) file to the Programming Assignment 2 folder. The file should include the following:

- README.txt. This file contains the project documentation; program features and usage scenarios; a brief description of (a) the TCP segment structure used (b) the states typically visited by a sender and receiver (c) the loss recovery mechanism; and a description of whatever is unusual about your implementation, such as additional features or a list of any known bugs. If you are extending your existing project, then mention it in the README with sufficient details of implementation.
- Makefile. The make file is used to build your application. Please try to keep this as simple as possible. (For Python, please mention the complete command to run your code in the README).
- The source code files.
- Script to run your code. You should submit a simple shell script to invoke the respective methods from your code. The interface for this is same as described above. This is required so that we can run tests on your code uniformly independent of the language/invocation.

Please make sure that your programs compile and run correctly. To grade your program we will extract the received zip file, type make, and run the programs using the same syntax given in section 2. Please make sure that your submission successfully passes this simple procedure.

Additional Notes:

- Your programs will be compiled and tested on a CLIC machine (clic.cs.columbia.edu), so please verify that your programs are written in versions that operate correctly in this environment. (For example, Java 7 is not available on CLIC machines. So please use Java 6). During our grading of PA1, some submissions are only compilable on Mac OS. We didn't deduct points for that in PA1. But you may lose some points in PA2. So please make sure that your code is compilable and runnable on CLIC machine.
- Do not simply submit the entire eclipse/your favourite project folder. Submit only the relevant files.

6. References

- 'Computer Networking: A Top-Down Approach Featuring the Internet' (3rd Edition) by James F. Kurose and Keith W. Ross. See Chapter 3 for a good description of transport-layer protocol fundamentals (with TCP as an illustrative example).
- 'TCP/IP Illustarted Vol I' by W. Richard Stevens. An excellent guide to the TCP/IP protocol.
- 'Beej's Guide' which can be found at http://www.beej.us/guide/bgnet/. An online tutorial for socket programming.
- Unix man pages for socket, bind, sendto, recvfrom.