CSEE 4119: Computer Networks, Fall 2015

Programming Assignment 2: Simple TCP-like transport-layer protocol

Due Friday, November 6th 11:55 pm

Academic Honesty Policy

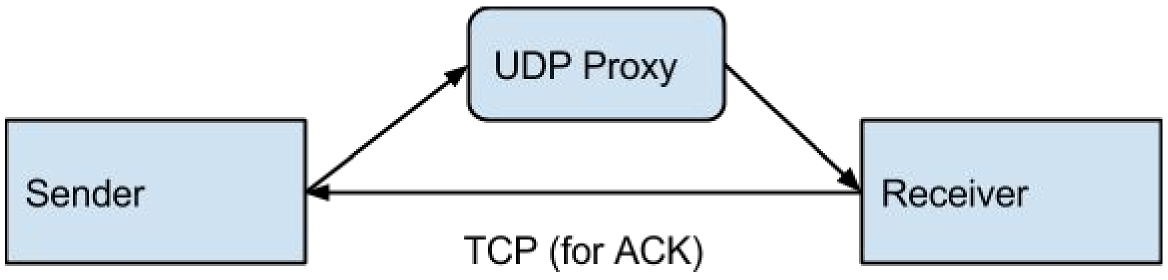
You are permitted and encouraged to help each other through Piazza's web board. This only means that you can discuss and understand concepts learnt in class. However, you may NOT share source code or hard copies of source code. Refrain from sharing any material that could cause your source code to APPEAR TO BE similar to another student's source code enrolled in this or previous years. Refrain from getting any code off the Internet. Cheating will be dealt with severely. Cheaters will be penalized. Source code should be yours and yours only. Do not cheat.

---

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 under Programming Assignment 2; 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 <filename> <listening_port> <sender_IP> <sender_port> <log_filename>`

```
%receiver file.txt 20000 128.59.15.37 20001 logfile.txt
Delivery completed successfully
%
```

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 (sender_IP) and port number (sender_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 <filename> <remote_IP> <remote_port> <ack_port_num> <log_filename> <window_size>
% sender file.txt 128.59.15.38 20000 20001 logfile.txt 1152
Delivery completed successfully
Total bytes sent = 1152
Segments sent = 2
Segments retransmitted = 0
%
```

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_num specifies the local port for the received acknowledgements.

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’).

The window_size is a parameter and window_size is measured in terms of the number of packets. Your sender should support variable window size, and it will be specified as the last parameter on command line. If no parameter is specified, the default window size value should be 1.

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
- 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.
- The link emulator can be run on UNIX/Linux vm (on Oracle VirtualBox or any virtual environment of your choice).
- Test your program over a wide range of network settings.
- Use file 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 your server will need to implement one more command called "GET", and this will indicate the start of the file transfer. The first assignment command line will need to be modified to accept new arguments. Mention the new command line arguments usage clearly in the README.
- For this assignment, assume that there is only one sender and one receiver for your file transfer.
- **Do not change the sequence/syntax of the input** parametes on command line. And also provide sample command line arguments (for sender and receiver) in README.
- TA's will compile and test the code you submit **on CLIC machines**. It is your responsibility to make sure your code is able to be compiled and runnable on CLIC machines.

  The **CLIC lab has following setup: Java 1.6, gcc version 4.6.3, python 2.7.3**

Testing point scheme:

	Issue	Deductions
1	Compilation and Execution related	
	Compilation fails	-100
	Sender not invoked in given fomat	-5
	Receiver not invoked in given fomat	-5
	No easy to read documentation / make file	-3

2	Functionality	
	In order delivery not handled	-10
	Packet loss not handled	-10
	Corrupted packets not handled	-10
	Packet delays not handled	-10
	Duplicate packets not handled	-10
	Log file not maintained on sender side	-5
	Log file not maintained on receiver side	-5
	Log file on sender does not follow given format	-5
	Log file on receiver does not follow given format	-5
	Statistics on sender is not printed properly	-5
	File on receiver is different than what sender sent	-10
	Variable window size supported	-10
	Any errors in above functions	-1 to -15 per error (TA discretion)
3	Extra features	Max +20
	Upto 10 points per extra feature (on TA discretion)	

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.
- Proxy references:
  - README http://www.cs.columbia.edu/~hgs/research/projects/newudpl/newudpl-1.4/newudpl.html
  - http://www.cs.columbia.edu/~hgs/research/projects/newudpl/