CSci4211: Introduction to Computer Networks Programming
Assignment II: TCP Analog
Due Date: Friday March 16, 2018 at 11:55p.m.

February 21, 2018

1 Your Task

Transmission Control Protocol (TCP) runs on top of the unreliable network-layer protocol Internet Protocol (IP). In order to provide reliability for TCP communication, some measures are employed such as error detection and retransmission of lost packets. These measures utilize checksums, time-outs, sequence numbers, acknowledgment, etc.

In this assignment, you will implement a TCP analog for transferring a file from a client to a server on top of a simulated network layer.

Stop & Wait protocol should be implemented. If you would like to do more things, try to implement a pipeline protocol (GBN or SR), which is much more difficult.

2 Details

Following are the details of the constructs and processes within the network layer simulator as well as the connection between network layer and TCP client/server.

2.1 Network Layer

The code for the simulated network layer has been provided for you and can be downloaded from the class website. The task of the network layer is to forward packets between the client and the server. However, given its unreliability nature, the network layer in this assignment, with a moderate probability, may drop packets and/or mangle bits within packets.

The network layer will accept connections from the TCP client and with a probability will forward packets to the TCP server (again may probabilistically mangle bits within the packet).

To run the network layer, you issue the following command: python networkLayer.py port maxNoPackets delay probDrop probMangle in which (you can check the networkLayer.py file for details):

- port is the port number that the network layer will be listening to. For simplification, we will use 5002.
- maxNoPackets is the maximum number of packet the network simulator can keep track of at any given time.
- delay is the amount of total delay beneath the transport layer. (Seconds, Float type)
- probDrop is the probability of dropping a packet. (Percentage. e.g. 1 for 1%)
- probMangle is the probability of mangling a bit within a packet. (Percentage)

2.2 TCP Client and Server

You will write a TCP client and a server in a SINGLE program file, the choice of which (either to run as a server or as a client) will depend on the number of arguments specified on the command line.

The TCP server will verify the validity of the packet (using checksums as will be discussed below). If the packet is valid, the packet is delivered to the upper layer (in our case, written to a file) and an acknowledgment is sent to the client. If the packet is invalid, the server ignores the packet. Therefore, the client eventually times out and retransmit the packet. Here are the steps of how your program should operate:

- You will first run the network layer simulator as specified in the previous section.
- You will run your program as a server binding and listening on port 5001.
- You will run your program as a client with a HOST (where your network simulator is running) and a port (5002) specifying that it should attempt a connection to the network layer simulator at that port. The client will also include a third additional argument which is a file that will be transferred to the server.
- Upon receiving connection from the client at port 5002, the network simulator will connect to your server at port 5001.
- Your program will endeavor to send all the contents of the file in increments (refer to the section which specifies the header structure of the packets to be transferred) from the client to the server (with retransmission when needed).
- The network layer will read from your client packets in blocks of 512 bytes with some packets randomly dropped (not forwarded to the server) or some of its bits mangled.
- If a packet is valid (based on the checksum), the packet will be delivered to the upper layer (copied to a file) and an acknowledgment will be sent back to the client.
- If a packet is invalid, the server will not send an acknowledgment back to the client. The client will time-out such packet, and will retransmit the same exact packet to the server.
- The ultimate goal is to transfer all the contents of the file from the client to the server.

2.3 Stop & Wait Protocol

Maybe you would like to try this first. In this protocol, each time we only send one packet to the socket, and client side is responsible for handling the transfer process (Recall that here the file is transmitted by the client). After sending, the client will wait for ACK for a certain amount of time. If there is a timeout event (or an ACK it does not want), the client should handle it properly. There are several different scenarios. A figure in Page 211 in the textbook may help you. Basically, four scenarios should be considered.

- Operation with no loss.
- A packet is lost.
- An ACK is lost.
- Premature timeout.

To handle a timeout event, you can use select() in Python, or a socket parameter setSoTimeout in Java.

2.4 Pipeline Protocols

If you are interested in implementing it, please discuss with us. It is much more difficult, and we do not require you to do this.

2.5 Packet Structure

The only requirement by the network layer on the packet structure is a size of 512 bytes. You can design the packet structure however you want. A suggested structure is shown below.

- 20-byte checksum (can use SHA-1) for checksum (Please refer to hashlib in Python, or MessageDigest in Java, for more information about programming with checksum.)
- 1-byte packet number, or sequence number (Note: If you are implementing a Stop & Wait protocol, one byte is enough. If you are for pipeline protocol, more bytes may be used.)
- 3-byte total size of the actual data
- 1-byte signifying if the packet is last
- 487-byte of data

3 Project Guidelines

Please keep the following in mind while doing the programming project:

- The file that you are going to transfer should only contain ascii characters.
- Any packet that should traverse the network layer should be of length 512. So any packet that is not of length 512 should be padded so that it is compatible with the network layer. For example in the server, instead of only sending "ACK" you will have to make a packet of length 512.
- You should NOT change the network layer in any manner (except for using java) because for grading we will use the network layer provided to you.
- For testing your project, just transfer a large file from the client to server and check whether the files are equivalent. Any file can be used and the file size should not be an issue. But we will not check files of more than 1MB.
- For the source code you have to submit only ONE file EXACTLY named 'program.ext' (where 'ext' can be py, java or any other file extension). The file will run as either as client or server based on the 'argv' parameters. For running server the arguments will be HOST (e.g. 'localhost', Though the host is unnecessary on server side in this project, you should just keep it for automatic testing) and PORT number; and for running client the arguments will be HOST (e.g. 'localhost'), PORT and filename. And based on the number or argument the file should run either as the client or server. So in all you will use three terminal windows to run your project (one for NetworkLayer, one for server and one for client). Failing to do such will result in the auto grader to not execute the program and points will be deducted.
- For the file to be transferred correctly the following three conditions should hold:
  - The name of files should be the same
  - The length of both of the files should be the same
  - The content of the files should be the same

4 What to submit

You should upload the project files to the Moodle site under Submissions section. Your submission (in a tar or zip format) should include the followings:

- The server/client source code (in one of the languages: Python, Java) and this should be in the proper format (named 'program.ext').

- Your test results, including client output, server outputs and network layer outputs, which should be txt files in a single folder.
- A brief readme.txt (less than 200 words) stating any compilation script you used, the machine you tested your program on, and any details that we should be aware of in order to compile and run your program. This should also include two or more paragraphs explaining what each segment of the client/server code does. This is not a line-by-line comments, but rather the logic of the code and what each segment does. You should also explain your design, the choice of packet structure and what you learned from the simulation, such as, the effect of longer delay on the rate of retransmission, how your program handled the errors,...etc.

5 Tips

- Start early to avoid last-minute issues.
- Ensure you can finish a Stop & Wait protocol before trying pipeline counterpart.
- Do not try to learn a new language, utilize knowledge of languages you are familiar with.
- Save and compile your work frequently.
- Try to make your program simple, commented and easy to read.
- Please read this description carefully. Feel free to ask questions.