Programming Assignment 2

Due Date: Friday, December 10th, 11:59PM

In this programming assignment, you will be writing the sending and receiving transport-level code for implementing a simplified version of TCP. Your version will use a network loss, delay and corruption link emulator provided by us. Data is exchanged via UDP, i.e., you will be running TCP "on top of" UDP. The network can drop, corrupt, reorder and delay packets. You will need to run one copy of the emulator, as acknowledgements are assumed to arrive without delay or loss. Please use version 1.7 of the emulator.

You can run your sender, receiver and the link emulator either on one, two or three machines. The link emulator acts like a "proxy", i.e., the sender is configured to send packets there and the proxy is configured to send packets to the receiver. The receiver process sends its acknowledgements directly to the data sender.

For testing, run the following command:

```
newudpl -iinput_host_address -ooutput_host_address -L 50
```

You will write a one-way ("simplex") version of TCP, without the initial connection establishment, but with a FIN request to signal the end of the transmission. Sequence numbers start at zero. Your program only has to handle one set of packets (one "file"). You do not have to worry about congestion or flow control, so the sender window size should be a fixed parameter that should be provided as command line input. However, you should adjust your retransmission timer as per the TCP standard (although it may be advisable to use a fixed value for initial experimentation). Your sender application should read data from a file and your receive application should write data to a file, both specified as command-line parameters.

The TCP data receiver should be invoked as follows:

```
tcpserver file listening_port address_for_acks port_for_acks
```

The server receives data on the `listening_port`, writes it to the file `file` and sends ACKs to `ip_address_for_acks`, port `port_for_acks`.

The client (data sender) has the following interface:

```
tcpclient file address_of_udpl port_number_of_udpl windowsize
ack_port_number
```

The *`ack_port_number`* is used to receive acknowledgements. The window size is measured in bytes and you can use any default value. You can assume that the TAs will specify all arguments when testing your client and server.

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 (with the checksum set to zero); this does not quite correspond to the correct way of doing it (which includes parts of the IP header), but is close enough.

The data is sent to the channel emulator, with its address and port specified on the command line of your data sender. The emulator will forward the packets to a different port and, possibly, host. If you want to, you can run the sender, receiver and emulator on one host, but you may find it easier to debug things if they're run on different hosts.

You can choose a reasonable value for the maximum segment size, e.g., 576.

There is extra credit for well documented code. Extra credit can boost your score to a maximum of 100. i.e. if you lost points in the written part, it will make up for that.

Please provide the following in your submission:

1. README.txt file which lists the following:
    - all the files submitted and a short description of each of them
    - the commands needed for running of the programs.
    - list of any known bugs, features, etc.
      The TA should be able to run your program on any host and port number by providing this information from the command line. If you've tested your code on a particular machine that you want the TA to also use, then mention this in your README. Any program that requires any editing of code on the TA's part will be rejected. If the README doesn't contain the necessary information, there will be a penalty.
2. A separate (typed) document of a page or so describing the overall program design, a verbal description of ``how it works'', and design tradeoffs considered and made.
3. A program listing containing in-line documentation
4. A screen dump of a typical client server interaction

Downloading the Link Emulator:

The link emulator tool we will use is called **newudpl**. On the Courseworks website, navigate to the “Files” page. We have uploaded two files: “newudpl” which is a Linux binary, and “newudpl-1.7.tar”, which contains the installation files for version 1.7 of the emulator. If your operating system is directly compatible (you have the right version of Linux), you can download and run the “newudpl” binary directly. Otherwise, download and unzip the “newudpl-1.7.tar” file. Follow the directions in INSTALL to install the emulator.