CptS/EE 455
Project #4 – Reliability with Sliding Window

Instructor: Adam Hahn
Due: 12/4/2019 at 11:59 pm

Deliverable:

Submit the code you developed and any information required to compile and run your program in a .zip/.rar/.gz to the class Blackboard page by the due date.

Assignment:

The project will explore how to utilize C sockets to implement a reliable communication over a simulated unreliable link using a *sliding window algorithm* as discussed in class in C to send a text file one line at a time between a client and server using UDP. You have 3 options for implementing this:

1) ***Extra Credit (20 pts)*** Extend your Project 3 program to implement UDP sockets and implement the proposed reliability requirements.
2) Use the provided template code (`client_udp.c, server_udp.c`) to unreliably transmit a file from the client to the server using UDP, you must use only UDP to send packets.
3) Use some other POSIX socket-based program language to send UDP-based packets, you must implement the high-level reliability using only UDP. Use the same command like arguments as shown in the provided code. 1

Test System:

Run your assignment in the `mininet` platform. Your system will run on a basic topology with two hosts, `h1 (IP: 10.0.0.1)`, and `h2 (IP: 10.0.0.2)`, which communicate through a switch, `s1`.

1. Download the `proj4_455.py` and `tux.txt` from the Github site.
2. Open three different terminal windows.
3. Start `mininet` in terminal #1 with the following command:

```
$ sudo mn –c
$ sudo mn ––mac ––switch ovsk  ––controller remote
```

4. Now, download POX into your home directory and start the POX SDN controller in terminal #2 utilizing the custom controller program, `proj4_555.py`. Download the controller and copy into the correct directory:

```
$ git clone http://github.com/noxrepo/pox
$ mv proj4_555.py  ~/pox/pox/misc/
```

Move to the "pox" directory and start the POX controller

```
$ cd pox
$ ./pox.py log.level --DEBUG misc.proj4_455
```

Verify that the controller connects to `mininet`, you should see the following output message:

```
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
```

5. Now go back to terminal #1 and run the compiled client and server code

```
mininet> h2 ./server_udp output.txt &
mininet> h1 ./client_udp 10.0.0.2 tux.txt
```

6. In terminal 3, view the newly created `ouput.txt` file. To verify it sent correctly, you should use the diff command as follows, the results should be empty

```
diff tux.txt output.txt
```

Recommendations:

The following suggestions may be helpful:

1) *Timeouts*: You may want to implement sockets that only block for a short period of time, therefore allowing you to resend a lost data frames. To do so, please utilize the setsockopt() function which enables you to set the amount of time the socket will block for during a recvfrom() call.

```
struct timeval tv;
tv.tv_sec = 0;
tv.tv_usec = 1000;

…
```

```
if (setsockopt(s,SOL_SOCKET, SO_RCVTIMEO,&tv,sizeof(tv)) < 0) {
    perror("PError");
}
```

2) Window Size=10: You should keep a sender buffer of 10 previous lines in case resends are needed.

3) Sequence Numbers & Acks: You'll need to use sequence numbers and acknowledgements, the number can just be represented by the first byte of each packet's payload.

4) Handle drops: The network will both randomly drop packets, so you should test and verify that your reliability mechanism can handle this.

5) Termination: To signal the termination of a connection, you can send a special sequence/acknowledgement number (e.g., 0xffff). It is possible that this is message is also dropped, but don't worry about acknowledging it.

6) Max line lengths: You can safely assume that each line sends at most 80 characters