CSE 3461 Lab 3

Instructor: Adam C. Champion, Ph.D.
Due: Monday, October 16, 2017, 11:59 p.m. (40 points)

Write a file transfer application using UDP sockets in Python. The file transfer protocol includes a server called `ftps.py` and a client called `ftpc.py`. In the following, we assume the client is running on `beta.cse.ohio-state.edu` and the server is running on `gamma.cse.ohio-state.edu`.[1] First, start the server on `gamma.cse.ohio-state.edu` using the command

```
python3 ftps.py <local-port-on-gamma>
```

Then, start `troll` on `beta.cse.ohio-state.edu` with the command (on one line)

```
troll -C <IP-address-of-beta> -S <IP-address-of-gamma> -a <client-port-on-beta>
      -b <server-port-on-gamma> -r -s 1 -t -x 0 <troll-port-on-beta>
```

Next, start `ftpc.py` on `beta.cse.ohio-state.edu` with the command (on one line)

```
python3 ftpc.py <IP-address-of-gamma> <remote-port-on-gamma>
      <troll--port-on-beta> <local-file-to-transfer>
```

The `ftpc.py` client will send all bytes of the local file to the `troll` process, which should forward the packets to `ftps.py`. The `ftps.py` server should receive the file and then store it. Ensure that you do the following:

- **Send all segments to troll, not the `ftps.py` server.** If you send to the server, troll will receive no segments;
- On the client, `bind()` to the `<client-port-on-beta>`. Normally, in UDP socket programming, only the server calls `bind()`, but `troll` requires binding client and server ports beforehand.
- The new file created by `ftps.py` must be *in a different directory* (i.e., `recv`) to avoid overwriting the original file, since all the CSE machines have your root directory mounted.

The file transfer application will use a simple protocol. The payload of each UDP segment will contain the remote IP (4 bytes), remote port (2 bytes), and a flag (1 byte), followed by a data/control field as explained below. The flag takes 3 possible values depending on the data/control field:

- **First segment (4 bytes):** The first segment should contain the number of bytes in the file to follow (in network byte order). The flag is set to a value of 1.
- **Second segment (20 bytes):** The second segment should contain 20 bytes which is the name of the file (assume the name can fit in 20 bytes). The flag is set to a value of 2.

[1] In your lab, you can substitute any machine on `stdlinux` (e.g., `zeta.cse.ohio-state.edu` or `epsilon.cse.ohio-state.edu`) for `beta.cse.ohio-state.edu` and `gamma.cse.ohio-state.edu`, respectively. You should do so to avoid "overloading" these machines.

– **Other segments:** The other segments will contain data bytes from the file to be transferred. Each segment can have up to 1,000 data bytes. The flag is set to a value of 3.

The `troll` process will drop packets with a drop rate percentage as indicated in the command line. With a drop rate of `0`, the file should be delivered intact to `ftps.py`. Submit well-documented code using the following command:

```
submit c3461aa lab3 <code-directory-name>
```

No buffer, array, or data structure should exceed 1,000 bytes. **You'll need to "chunk" the file into "pieces," each of which is at most 1,000 bytes. Submit a** `README.txt` **file with your lab.** Your program should work for *binary* files (images, etc.) of arbitrary size and the client and server should be running on different machines on `stdlinux`. You can use the sample images on the course webpage to test your program. Use `diff` or `md5sum` at the command line to ensure that the transferred file is bitwise identical to the original one. To avoid overrunning UDP buffers, you can call `sleep()` for a short time between sending packets.