Lab 2: Threading, Asyncio, and Tunnels

Due date: May 12, 2023 (Anywhere on Earth, i.e., May 13, 5 AM PDT)

TA Contact Info: Stan Lyakhov, lyakhovs@oregonstate.edu

1. **Multithreaded Server** In Lab 1 we implemented a simple server and client that communicated back and forth. However, our server could only accept one connection at a time, and the rest of the clients that connect to the server have to wait for all the previous clients to be serviced. Now that we learned about threads we can fix this issue!
   (a) Copy your *lab1_server.py* and rename it to *lab2_server.py*. Modify the code to use *threading*. Note that a thread should be created and started after the line that calls *server_socket.accept()*.
   Hint: the functionality of sending intro message/receiving long message should be moved into a separate function so that it can be called by the thread! Starting the thread should be the last thing that the *main* function does.
   (b) To test that the server works as expected, make sure that you can still communicate with it using *lab1_client.py*.
   (c) Try to connect to the server by starting the client **twice** (you should have 3 python files running. The server and two clients). If threading is working correctly, both clients should be receiving the intro message from the server before either enters the long message!

2. **Congestion Testing** Now that we made a multithreaded server in Part 1, we can do some load testing by making client that will repeatedly connect to the server!
   You have two choices here:
   - You can use the provided *async_client.py* to do the testing below.
   - **BONUS 5 points**: make your own client using **threading**. You may use *lab1_client.py* as a starting point and add threads. For the long message part, specify a long message ahead of time (see how *async_client.py* does it for inspiration) and send it. Don't use *input()* to get the long message for each client.

   (a) We will want to measure congestion using Wireshark, so start it up listening on the loopback adapter.
   (b) Run the server from Part 1. Try connecting the client with different number of threads/tasks. Recommended values are (10, 20, 50, 100, 200, 500, and 1000). If you are using *async_client.py*, you need to find how to change this value in the code!
   (c) In Wireshark, stop capture. Go to *statistics -> IO Graphs*. Paste the graph here and describe what you see. Are there any TCP errors? When do they start to appear (i.e. how many clients have to connect before the network starts getting overwhelmed). **Attach a screenshot**. If there are no TCP errors, you may want to try connecting even more client threads until they start to appear.
   (d) Find the packets with TCP Errors in the main Wireshark window (you can usually do this by pressing the red bars in the IO Graph view). What sort of TCP errors are they? **Attach a screenshot**.

3. **Finally reaching the internet** So far in the labs we only worked on localhost: that is not very satisfying! Let's actually use the internet for once.

   (a) Time to reveal my secret plan: we are going to use *netcat* to send messages back and forth between two different laptops by using the flip servers as a *proxy* (grab a friend! Please let me know if this is a problem). To make it more fun, we will be doing this **without** connecting to the OSU VPN!

   So, first make sure you are **not** connected to the OSU VPN.

   (b) To start off, we need to run a *netcat* server on flip.
   Laptop 1 should connect to flip (*SSH ONID1@flip2.engr.oregonstate.edu*) and then start the netcat server:
   nc -lv localhost RANDOMPORT. Make sure to replace "ONID1" with your actual ONID (mine is lyakhovs) and "RANDOMPORT" with a some random port value where you'll be hosting the server (try to choose something above 5000).

   (c) Laptop 2 will now create the SSH tunnel.
   Run the command *SSH -L localhost:8080:localhost:RANDOMPORT ONID2@flip2.engr.oregonstate.edu*. Now replacing "ONID2" with the ONID of the second user and "RANDOMPORT" with the same port used above.

   What this does is create a *tunnel* between "localhost:8080" on Laptop 2 and "localhost:RANDOMPORT" on the flip servers. Conceptually, this means that every time Laptop 2 connects to "localhost:8080", it gets *forwarded* to "localhost:RANDOMPORT" on the server! As a bonus, this data gets encrypted on your computer and decrypted when it reaches the server.

   (d) Laptop 2 can now connect to netcat: without closing the terminal that has the open tunnel, open a new terminal and run *nc localhost 8080*. If you can't install netcat, you can also use *telnet* by replacing *nc* with *telnet* (it might be easier to get *telnet* on windows).

   (e) Connection should be established, and both Laptop 1 and Laptop 2 should be able to send each other messages! Isn't this fun? Chat with your friend.

   (f) **Attach screenshots** of your communication between Laptop 1 and Laptop 2. Try to make it obvious that the communication happens between two separate laptops.