Lab 02: TCP, UDP, Socket Programming

Release Date: 28th January, 2025(Tuesday Section) && 30th February, 2025(Thursday Section)

Demo and Submission Date:4th February, 2025(Tuesday Section) | 6th February, 2025(Thursday Section)

Deliverables (submit compressed folder to vocareum):

- tcp_client.py or tcp_client.cpp
- Readme.md

Teams of 2.

Only one of the team members to do the submission with both full names and USC IDs.

Quick Links:

Primer on IP Addresses: IP addresses (article) | The Internet | Khan Academy

Introduction

In class, we've been discussing networking concepts such as TCP, UDP, and ports. However, they will be easier to understand once we start using them. In this lab, we will dive head first into socket programming. We handpicked examples we think will expose you to practical socket programming concepts that will help you understand and debug network problems you will see in future projects or see when using consumer applications at home.

Recommend for team members to manage your codes.
In order to connect to Github from the client, you'll need to set up an SSH key since password authentication has been deprecated. The easiest way to do this is to copy the contents of your SSH public key and pasting them into a new key in Github (see settings, then "SSH and GPG Keys" in the menu). If you don't already have a public/private ssh key pair, you can generate one by running the following:

```
ssh-keygen
```

For this class, you can leave all the default answers to the prompts (just keep pressing "enter"). This creates the public/private keys in ~/.ssh. To view your public key, run:

```
cat ~/.ssh/id_rsa.pub
```

For a more complete tutorial on ssh keys for github authentication, check out this article.

Evaluating TCP vs UDP

For this part we will highlight the differences of the UDP and TCP protocols. You will need to answer a few questions in a text file named 'Readme.md.'

First we will use the 'nc' command to test local TCP and UDP connections on our computer. We will see how it responds in a perfect environment, and then we will see how it responds in a lossy environment (50% loss)

To do this we will need a package **"Netcat", let's take a look at what it is before we start:**

What is Netcat?

- netcat is a versatile networking tool used for debugging, testing, and transferring data between hosts.

What Does This Test Do?

- In this test:
    - Terminal 1 listens for incoming messages on port 12345.
    - Terminal 2 sends a message to localhost (your own machine) on port 12345.
    - Terminal 1 displays the received message, demonstrating netcat's ability to listen for and process messages.

Run the following command to install Netcat:

- **Update the Package List:** Ensure your system has the latest package information
    - sudo apt update

- **Install netcat-openbsd,** use the following command to install the OpenBSD version of netcat:
    - sudo apt install netcat-openbs
- **Verify Installation,** Confirm that netcat-openbsd is installed:
    - dpkg -l | grep netcat

- **Testing netcat**

  1. Create a Listener(In Terminal 1, run the following command:)

  nc -l 12345

- This starts a listener on port 12345.
- The terminal will appear to hang, waiting for input.

  2. Connect and Send a Message(In Terminal 2, send a message to the listener:)

  echo "Hello, Netcat!" | nc localhost 12345

- **Check the Listener**
    - **Go back to Terminal 1, and you should see the message "Hello Netcat!"**

Evaluating UDP Reliability:

1. In a terminal, start a UDP server
   nc -u -l 10000
2. In a new terminal, connect a UDP client to the server
   nc -u localhost 10000
3. **Test UDP Communication:**

In **Terminal 2** (client), send a sequence of '1', '2', '3' … '10' to the server by typing directly into your nc client and hitting 'enter'

4. Now, in a third terminal, force a 50% loss on your local environment
   `sudo tc qdisc add dev lo root netem loss 50%`
5. In **Terminal 2**, send the sequence 1 to 10 multiple times. Check **Terminal 1** to observe that some numbers may not arrive at the server due to the induced packet loss.
6. When done, remove the packet loss simulation:
   `sudo tc qdisc del dev lo root netem loss 50%`

Evaluating TCP Reliability:

1. In a terminal, start a TCP server
   `nc -l 10000`
2. In a new terminal, Connect a TCP client to the server
   `nc localhost 10000`
3. Send a sequence of '1', '2', '3' … '10' to the server by typing directly into your nc client and hitting 'enter'
4. Now, in a third terminal, force a 50% loss on your local environment
   `sudo tc qdisc add dev lo root netem loss 50%`
5. Send a sequence of '1', '2', '3' … '10' to the server again, look for any changes. In **Terminal 2**, send the sequence 1 to 10 again multiple times. Observe the output in **Terminal 1**. Unlike UDP, TCP should handle the packet loss and deliver all messages, albeit with potential delays.
6. Remove the loss
   `sudo tc qdisc del dev lo parent root netem loss 50%`

Question 1: How did the reliability of UDP change when you added 50% loss to your local environment? Why did this occur?

Question 2: How did the reliability of TCP change? Why did this occur? Question 3:

How did the speed of the TCP response change? Why might this happen?

TCP Client & Server

For this lab you'll be working primarily in your VM and you'll need to get the codes in Lab2-Demo from Google Drive.

TCP Server

The server code is provided for you but you will need to **answer the questions in the code** in the 'Readme.md' file. Go through the code, and try to understand the basic logic. You will submit this text file as a part of Lab 02.

Compile tcp_server.c with the command `gcc -o server tcp_server.c` and execute the output binary with the command '`./server <PORT NUMBER>`'. If you have an error that says 'gcc command does not exist', you'll need to install it using `sudo apt install gcc`

TCP Client

Use Python (see this python sockets tutorial) or C++ (see this C++ sockets tutorial) to write a TCP client to interact with a TCP server. Python will be easier for you but you'll learn a lot if try it in C++. If you do it in C++, you'll also get 3 bonus points. Please implement the TODOs in the provided tcp_client.py or tcp_client.cpp file.

TIP: You can compile the C++ code with the command `g++ -o client tcp_client.cpp`' and execute the compiled binary with the command `./client`.

Test your TCP server and client together using **localhost** or your IP a

Remote Server (Use your Rpi as server & VM as client)

Now you're going to deploy a server onto a remote machine (your Raspberry Pi device)! Make sure your VM and the Raspberry Pi are powered on and connected to the same network and ssh into your Raspberry Pi.

`ssh <username>@<hostname or IP address>`

This is just like in Lab 01. Then, in another terminal, copy the server code into your raspberry pi by using the scp command (which is very similar to ssh):

`scp <path to file> <username>@<hostname/IP address>:<destination path>`

For example, if my RPi username is **pi** and my RPi hostname is **my-rpi**, I could

run: `scp ./tcp_server.c pi@my-rpi:~/tcp_server.c`

This would copy the file from my machine to the user **pi**'s home directory on the raspberry pi. Take a second to fully understand what's going on in the above command. Now, we can run the server on the Raspberry Pi and the client running on our PC can communicate with it over TCP! Re-compile and run the server code on the Raspberry Pi. Then, on your PC, change the hostname/IP address to that of the Raspberry Pi before running it. You should see the same output as in the previous part!

You could run this server anywhere and as long as you have the IP address of the machine it is running on, the client can communicate with it! If you're interested, you could deploy a VM on Microsoft Azure (you get $100 in free credits with the student developer pack, which is more than enough for this simple experiment). Check out this Video Tutorial for help on doing this.

Microsoft Azure Tips
Accessing your VM: Use ssh just like we do with the Raspberry Pi! Instead of a username/password, though, you'll use an ssh key. See here for detailed instructions on how to do this.

Firewall: In order to make sure your client will be allowed to communicate with the server running on your VM, you need to make sure the firewall is set up to allow TCP traffic on the appropriate ports. See these instructions for how to do this.

Bringing down your VM: You get charged by the minute for having your VM running, so make sure to remember to bring it down when you're done with it! You can find instructions on that here.

Save your code(Optional if team members are using Github to collaborate)

Use git to *add*, *commit*, and *push* your changes to your Github repository. See the **Add and commit changes** section from this tutorial for more information!

Demo and Code Grading Rubric

Points	**Description**
Demo	
3	The client code should be able to receive 'I got your message' from the server and print it in the terminal
3	The client code is able to read user input and send it to the server, which should then print that input.
2	The server runs properly on the Raspberry Pi.
2	The client runs properly with the server on the Raspberry Pi.

Code	Each Team much submit files to Vocareum
2	Team member names listed in Readme.
4	Code correctness (no python syntax errors/code compiles, sufficiently bug-free for the assignment, etc.)
3 (bonus)	Implemented in C++
Reflection	Answer Lab Questions in Readme.md
3	QA.1 to QA.3 (questions found in this lab document)
7	QC.1 to QC.7 (questions found in tcp_server.c)
	Total: 26