TCP, UDP, and Port Forwarding

Intro

In class, we learned the concept behind TCP, UDP, and ports. However, understanding the concept of practical topics like computer networking is only enough to begin having a conversation about its implementation as well as real-world behaviors. In this workshop, 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.

GitHub/File Access

VM Processes

- Make sure to enable virtualization in BIOS
- Make sure under “Settings” the computer can run the VM
- Be on ee250 wifi
    - Password: ee250@vhe205
- Install sublime text
    - https://www.sublimetext.com/docs/3/linux_repositories.html#apt

1. Make a GitHub account!

- Install git on VM
    - *sudo apt update*
    - *sudo apt upgrade*
    - *sudo apt dist-upgrade*
    - *sudo apt install git gitk*
- Add ssh key to your github
    - *ssh-keygen -t rsa -b 2048 -C “ttrojan@usc.edu”*
        - Use their own email
    - Creates new key, when asked for file location, just hit “Enter”
    - When asked for passphrase, just hit “Enter”
    - Should receive identification
    - Add the key to github through Settings and “Add SSH Key”
    - Display id_rsa.pub file with *cat ~/.ssh/id_rsa.pub*
    - Copy them and paste into key field. Copy starting with “ssh-rsa” and should end with your email address
    - Add key, need to do for both the rpi and VM
    - Configure git
        - *git config --global user.name “Your name”*
        - *git config --global user.email “Your email”*
        - *git config --global push.default simple*

2. Clone repository

git clone git@github.com:git@github.com:jeff55235/hackiot2019-rpi-workshop.git
cd hackiot2019-rpi-workshop.git

3. Switch branches (on VM only)

git status
git checkout master
git checkout -b branchname
git push --set-upstream origin branchname .

4. Pull on RPi

git clone git@github.com:git@github.com:jeff55235/hackiot2019-rpi-workshop.git
cd hackiot2019-rpi-workshop.git
git pull
git checkout branchname

TCP Example

Go to...

hackiot2019-rpi-workshop/tcpExamples

Notice:

- The use of listening
- Once we send messages, the connection isn't broken

UDP Example

We will switch over to the UDP protocol and communicate between your VM and your Raspberry Pi over a Local Area Network (LAN) using sockets. Every machine/device typically has a single IP address on a LAN. To send packets to a specific process on a machine, you need to use ports at the transport layer (or L4). The examples we will run here will illustrate how ports work.

Go to…

hackiot2019-rpi-workshop/udpExamples

UDP Part 1

Run udpServer1 and udpServer2
Pro tip: the hotkey for spawning a terminal in Ubuntu is Ctrl+Alt+t.

Both of them will have the same port starting off in the code. What we will see is that we can't have multiple servers occupying the same port. If one is in use the other can't connect.

Now change udpServer2's port to 9001. We can run it now. Try changing the ports within the 9000's. As long as they are different numbers we're fine.

This is important because it shows that ports must be separate for UDP and these separate ports will help for less latency later on.

UDP Part 2

Now run both servers on different ports in the 9000's.
Try running the udpClient file. You will see you can't. Change the port to 1024. You'll see that you can run it now.
There are actually defined ports for TCP and UDP.
https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

So you will essentially have to establish ports that are out of the established ones. This is why we use 9000's because nothing would generally use 9000.

Try changing the port in udpClient to 5001 and now try sending messages to the different UDP servers.

UDP Part 3

Now, we will run python code on two separate machines (your VM and rpi). Login to your rpi and clone the git repository. We want to send packets from rpi to VM over the local area network. Because the VM is a guest OS on your host OS, this will not be straightforward.

1. Find RPI IP Address: While SSH'ed into RPi, type "ifconfig" and under "wlan0" next to "inet"
2. Find VM IP Address: In terminal on VM, type "ifconfig" and under "enp0s3" next to "inet"
3. In Host OS, type "ipconfig" (for Windows) or "ifconfig" (for Mac) into terminal and under "Wireless LAN adapter Wi-fi:" next to "IPv4 Address"
4. Set up Port Forwarding on VirtualBox
    a. Open VirtualBox Network Settings - Click on "Network" in the VirtualBox window (not the running VM)

b. Click “Advanced”
c. Click “Port Forwarding”
d. Click the “+” icon
e. Then paste into the rule
    i. Protocol -> UDP
    ii. Host IP -> IP Address from Step 3
    iii. Host Port -> 8050, 8051, … , 805x
    iv. Guest IP -> IP Address from Step 2
    v. Guest Port -> 9000, 9001, …, 900x
f. Do part e again for second server

VirtualBox Port Forwarding:
Find the VirtualBox network settings for your VM and add two port forwarding rules, one for each UDP server script you want to run on your VM. You should not have to shutdown or restart your VM, but doing so may help you debug any issues. Below is an example scenario followed by a set of port forwarding rules:

Example Scenario
Host OS IP: 192.168.1.183 *
VM IP: 10.0.2.15
RPi IP: 192.168.1.55 (not important)
udpServer1.py listening port: 9000 (you choose this)
udpServer2.py listening port: 9001 (you choose this)

Idea of Port Forwarding:

The default networking mode of VirtualBox (and most other virtual machine managers) is to enable something called a Network Address Translator (NAT). Commercial routers you can purchase for your home run NATs as well. Most commercial routers allocate private Local Area Network (LAN) IP addresses to connected devices that look something like `192.168.1.100`. This is how the VHE 205 router behaves. Since LAN IPs are internal IP addresses, you would not be able to connect to any device on these router networks from the outside by using these private addresses. Instead, you would need to use the Wide Area Network (WAN) address (aka the static or dynamic IP allocated by your Internet Service Provider (ISP)). However, this is not straightforward because trying to connect to a WAN address will actually lead to connecting to just the router and not the device connected to the router. To address a specific device behind a router, you would need to setup something called port forwarding in a router's firmware.

Think of it this way. Right now if your RPi tried to send messages to your VM. Because your VM is essentially a computer within a computer, messages would be received by your host OS and not your VM. So, we need to setup port forwarding so that we can send packets from our RPis connected to the router to a VM running on a PC connected to the router.

VirtualBox Port Forwarding Coding Setup

Please use ports 8050 and 8051 for host OS (8000 and 8001 may possibly be used by Windows/macOS applications).

1. On the VM
    a. Set the “host” variable in both server scripts to be the IPv4 address of the VM
    b. Have the ports be different 9000s numbers
2. On the RPi
    a. Change the host variable to the RPi’s LAN IP address
    b. Change the “server” variable to the host OS’s LAN IP address
3. But you will send messages to ports 8050 and 8051 as per port forwarding they will go to the 9000 ones on the VM.

Application Example

Ultrasonic Sensor and UDP:

Develop an application in the ultrasonicClient.py and ultrasonicServer.py files provided which streams the distance output from the ultrasonic ranger sensor connected to your rpi to your VM every 200ms using UDP packets (use time.sleep() to add the interval).

Main Idea: Since we are constantly streaming data, UDP is a good protocol choice here because we are not too worried about lost packets and we are looking for the most updated value with the least amount of delay.

Things to Have:

- ultrasonicClient.py running on the RPi should send packets to port 8050
- ultrasonicServer.py running on your VM should be listening in on port 9005.
- Sleep every 200ms
- Will need to use port forwarding rules to make get your RPi to connect to your VM.

What it should do:

ultrasonicClient.py should print() the stream of ultrasonic ranger readings in addition to sending the data over to your VM via UDP packets. The format of ultrasonicClient.py should be in the following format where [VALUE] is the actual value read from the ultrasonic ranger:

```
RPi: [VALUE]
cm
```

ultrasonicServer.py should also print out the received ultrasonic ranger reading packets. The format should be in the following format where [VALUE] is the value received in the UDP packet.

```
VM: [VALUE]cm
```

LED and TCP:

In this second application, we want to reliably control a Grove LED of a color of your choosing attached to your RPi from your VM.

Main Point: To achieve this, we will use a TCP socket to increase message reliability. This is because now we are only sending singular messages that are important.

Things to have:

- This time, you can use any available ports that work.
- Your RPi should run ledServer.py,
- Your VM should run ledClient.py.
- The strings to be sent should be user-inputted and should only turn on and off the Grove LED when it detects the correctly formatted string.

What it should do:
You should turn on the LED when you send the string "LED_ON" from VM to rpi and turn off the LED when you send the string "LED_OFF".

It is always a good practice to insert a feedback mechanism to your code to help debug. Your application must have the RPi reply to every message it receives to give feedback.
More specifically, the RPi should respond with "LED_ON Success", “LED_OFF Success”, and "Command Not Recognized" via a TCP packet to the VM.
The VM should print these received messages accordingly. This acknowledgement and print out feature will help you verify if your application works.