Lab #4 Report and Reflections

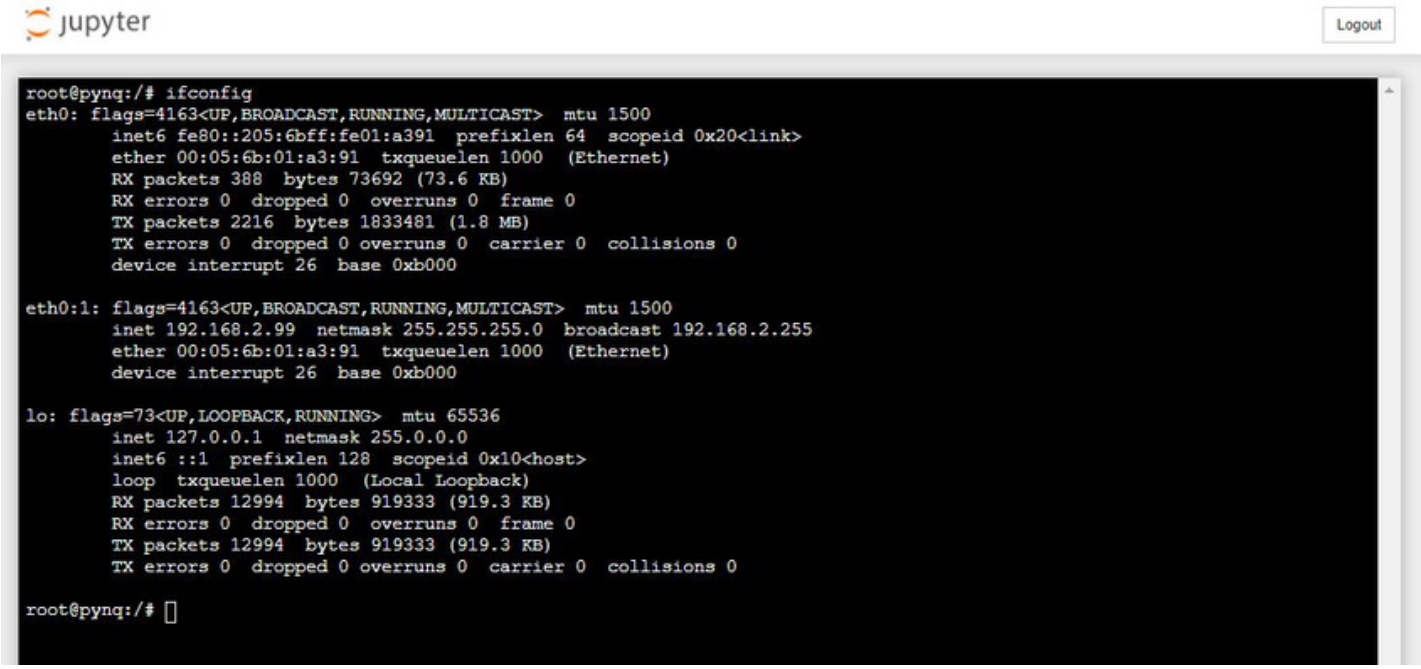
ABDULLAH AJLAN **A69028719**

WES 237A: Introduction to Embedded System Design (Winter 2024)
Lab 4: Network Communication

Locating IP Addresses of Devices in your Network

1. ~~Open a serial connection to your PYNQ board (see Lab3 if you forgot)~~
2. Connect the PYNQ board to the network switch over ethernet.
3. Run '$ ifconfig'. This is the *Interface Configuration* command and will tell you the different interfaces on your PYNQ board.
    a. **How many ipv4 addresses are assigned to the board? What is the ipv4 address assigned to the 'eth0' or ethernet interface? What is the netmask of this address?**
        i. Note: `eth0: 1` or `usb0` is a virtual interface through the USB cable. This assigns your board an IP address over USB. This is a static IP address so you can always reach your board from this IP address over USB.

The board has one IPv4 address assigned to the eth0 interface. The IPv4 address assigned to the eth0 interface is **192.168.2.99**. The netmask of this address is **255.255.255.0**.



4. ~~Use a lab computer or your personal computer with either of the following setups~~
    a. ~~Connected to the *WES237A_Private* wifi network (passwd: X!!!nxWes237A)~~
    b. ~~Connected directly to the network switch through ethernet cable~~
5. Open a command prompt and run `$ ipconfig` on windows and `$ ifconfig` on MAC/linux (it may take a second to connect so wait a minute and then run the command)
    a. **How many ipv4 addresses are assigned to this machine? What ipv4 address has the same netmask as the PYNQ board?**

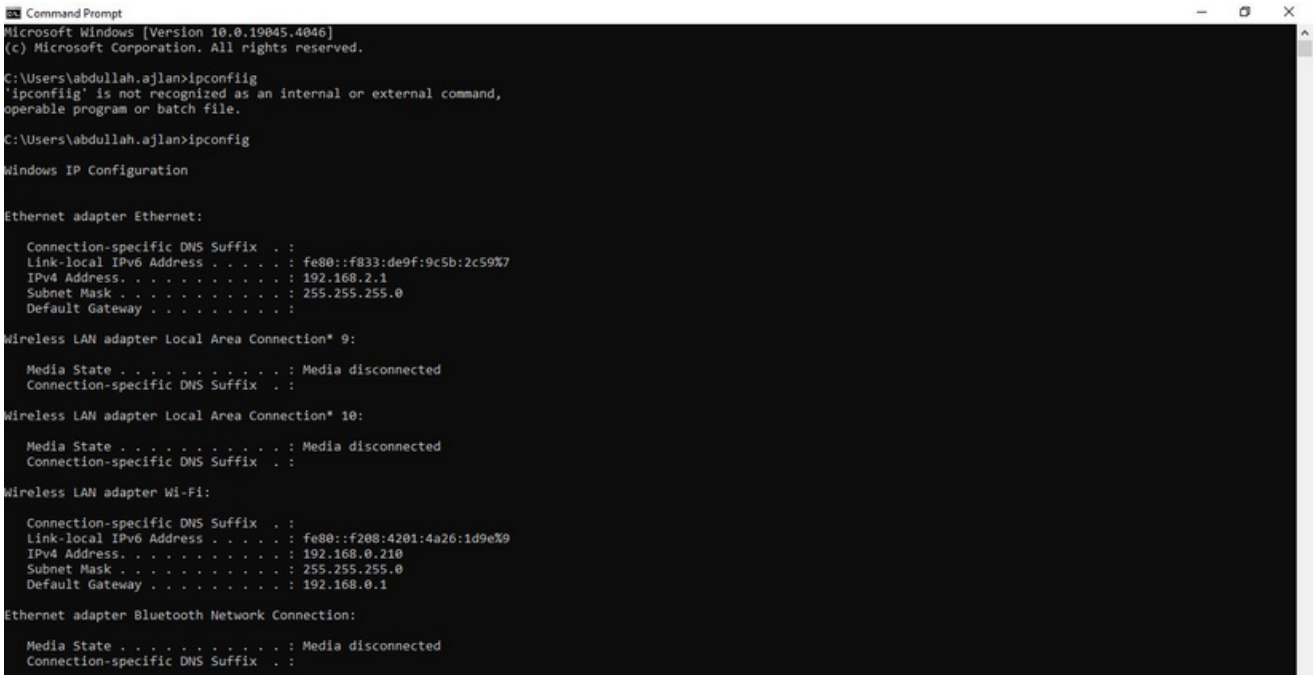
Based on the ipconfig output provided for Abdullah Ajlan's machine (local) , there are two IPv4 addresses assigned to this machine:
An IPv4 address assigned to the Ethernet adapter: 192.168.2.1
An IPv4 address assigned to the Wi-Fi adapter: 192.168.0.210

Both of these addresses have the same subnet mask of 255.255.255.0.
Comparing this to the PYNQ board's ifconfig output from the previous message, the PYNQ board has an IPv4 address of 192.168.2.99 with a subnet mask of 255.255.255.0.
The IPv4 address on Abdullah Ajlan's machine that has the same netmask as the PYNQ board is 192.168.2.1 (Ethernet adapter), as both have the subnet mask 255.255.255.0.

6. Right now, your local machine and your PYNQ board form a network! However, we're more interested in networking two PYNQ boards together rather than your local machine and your PYNQ board. Luckily, every device hooked up to the switch, is assigned an IP address on this network. That means we can communicate with any other board in the class. **Below, compile all the IP addresses of the PYNQ boards in your group.**

```
C:\Users\abdullah.ajlan>nmap -sn 192.168.2.0/24
Starting Nmap 7.94 ( https://nmap.org ) at 2024-02-18 18:31 Arab Standard Time
Nmap scan report for 192.168.2.99
Host is up (0.0039s latency).
MAC Address: 00:05:6B:01:A3:91 (C.P. Technology)
Nmap scan report for 192.168.2.1
Host is up.
Nmap done: 256 IP addresses (2 hosts up) scanned in 4.41 seconds
```

7. To access your PYNQ board jupyter notebooks, go to <PYNQ-IP>:9090




```
C:\Users\abdullah.ajlan>nmap -sn 192.168.2.0/24
Starting Nmap 7.94 ( https://nmap.org ) at 2024-02-18 18:31 Arab Standard Time
Nmap scan report for 192.168.2.99
Host is up (0.0039s latency).
MAC Address: 00:05:6B:01:A3:91 (C.P. Technology)
Nmap scan report for 192.168.2.1
Host is up.
Nmap done: 256 IP addresses (2 hosts up) scanned in 4.41 seconds
```

PYNQ-PYNQ Communication with Python

1. Here we're going to implement basic message sending functionality in python from one PYNQ board to another.
2. Download 'sockets_example.ipynb'
3. Go through and complete the code. Answer the following questions.
    a. **What does `socket.SOCK_STREAM` mean (hint: search the documentation link in the notebook)?**

The socket.SOCK_STREAM constant in Python's socket module defines the socket type as a stream socket, which is used for TCP communications, ensuring reliable, ordered, and error-checked delivery of data.

    b. **What is the order of operations for starting a client socket and sending a message?**

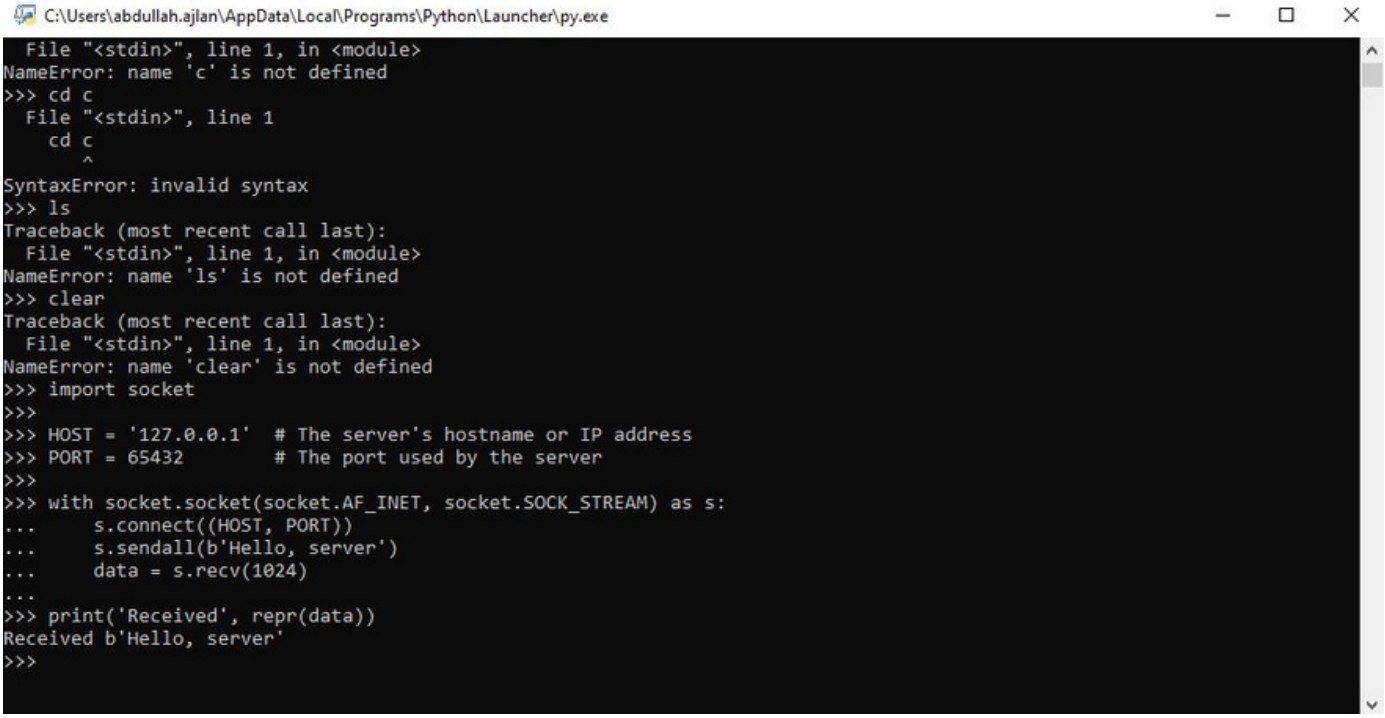
Import the socket module:
import socket

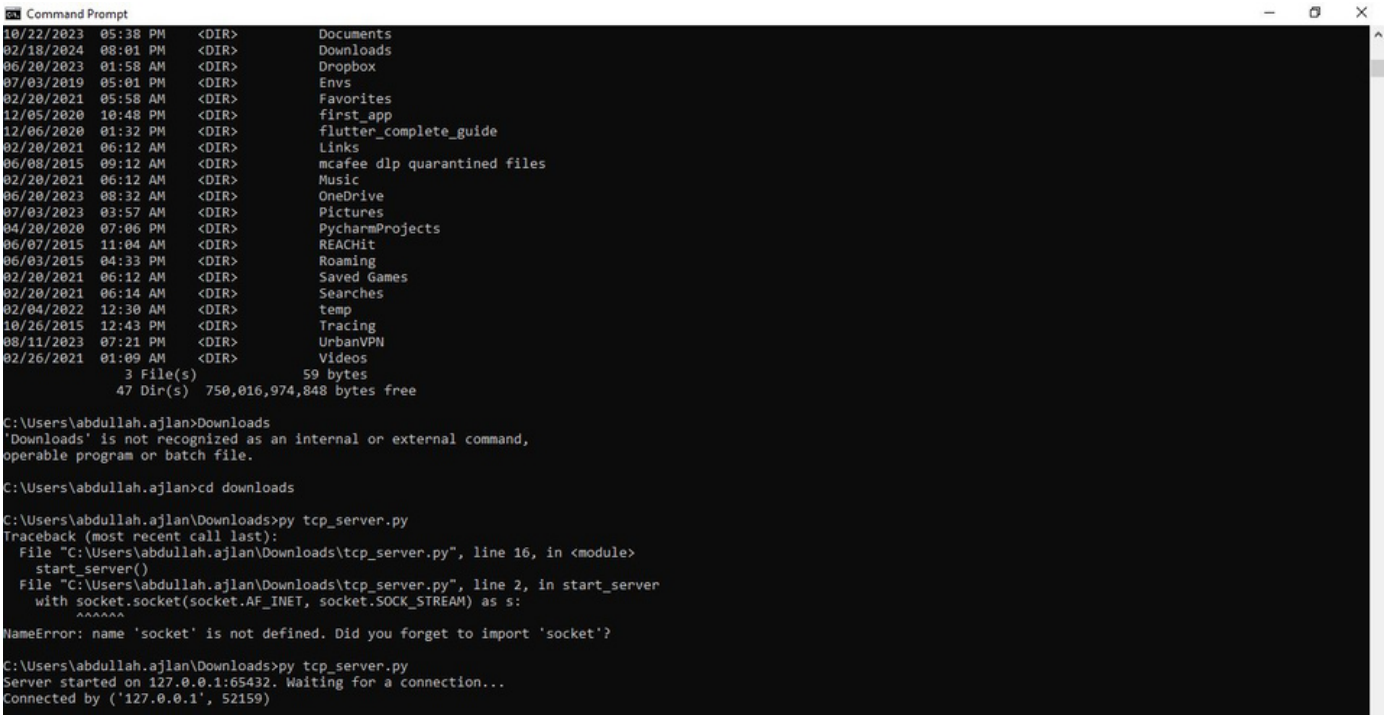
    c. **What is the order of operations for starting a server socket and receiving a message?**

```
def start_server(host='127.0.0.1', port=65432):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((host, port))
        s.listen()
        print(f"Server started on {host}:{port}. Waiting for a connection...")
        conn, addr = s.accept()
        with conn:
            print(f"Connected by {addr}")
            while True:
                data = conn.recv(1024)
                if not data:
                    break
                conn.sendall(data)

if __name__ == "__main__":
    start_server()
```


In [4]:
```
import socket

# Replace with the actual IP address of your laptop (the server)
server_ip = '192.168.2.1'
server_port = 12345

# Create a socket object
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect to the server
client_socket.connect((server_ip, server_port))

# Send the message
client_socket.sendall(b'Hello world\n')

# Close the connection
client_socket.close()
```

In [ ]:


C:\Users\abdullah.ajlan\AppData\Local\Programs\Python\Launcher\py.exe
File "<stdin>", line 1, in <module>
NameError: name 'c' is not defined
>>> cd c
File "<stdin>", line 1
cd c
^
SyntaxError: invalid syntax
>>> ls
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'ls' is not defined
>>> clear
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'clear' is not defined
>>> import socket
>>>
>>> HOST = '127.0.0.1' # The server's hostname or IP address
>>> PORT = 65432   # The port used by the server
>>>
>>> with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
...   s.connect((HOST, PORT))
...   s.sendall(b'Hello, server')
...   data = s.recv(1024)
...
>>> print('Received', repr(data))
Received b'Hello, server'
>>>


Command Prompt
10/22/2023 05:38 PM <DIR> Documents
02/18/2024 08:01 PM <DIR> Downloads
06/20/2023 01:58 AM <DIR> Dropbox
07/03/2019 05:01 PM <DIR> Envs
02/20/2021 05:58 AM <DIR> Favorites
12/05/2020 10:48 PM <DIR> first_app
12/06/2020 01:32 PM <DIR> flutter_complete_guide
02/20/2021 06:12 AM <DIR> Links
06/08/2015 09:12 AM <DIR> mcafee dlp quarantined files
02/20/2021 06:12 AM <DIR> Music
06/20/2023 08:32 AM <DIR> OneDrive
07/03/2023 03:57 AM <DIR> Pictures
04/20/2020 07:06 PM <DIR> PycharmProjects
06/07/2015 11:04 AM <DIR> REACHit
06/03/2015 04:33 PM <DIR> Roaming
02/20/2021 06:12 AM <DIR> Saved Games
02/20/2021 06:14 AM <DIR> Searches
02/04/2022 12:30 AM <DIR> temp
10/26/2015 12:43 PM <DIR> Tracing
08/11/2023 07:21 PM <DIR> UrbanVPN
02/26/2021 01:09 AM <DIR> Videos
3 File(s) 59 bytes
47 Dir(s) 750,016,974,848 bytes free
C:\Users\abdullah.ajlan>Downloads
'Downloads' is not recognized as an internal or external command,
operable program or batch file.
C:\Users\abdullah.ajlan>cd downloads
C:\Users\abdullah.ajlan\Downloads>py tcp_server.py
Traceback (most recent call last):
File "C:\Users\abdullah.ajlan\Downloads\tcp_server.py", line 16, in <module>
start_server()
File "C:\Users\abdullah.ajlan\Downloads\tcp_server.py", line 2, in start_server
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
^^^^^^
NameError: name 'socket' is not defined. Did you forget to import 'socket'?
C:\Users\abdullah.ajlan\Downloads>py tcp_server.py
Server started on 127.0.0.1:65432. Waiting for a connection...
Connected by ('127.0.0.1', 52159)

Wireshark

1. On your local machine (or lab machine), install Wireshark
2. Open the Firewall and Network Protection
3. Click 'Allow an app through firewall'
4. Click 'Change Settings'
5. Scroll down to 'Python'
6. Select all 'Python' applications and all 'Public' boxes for each 'Python'

7. Open the program `IDLE (Python 3.7 64-bit)`
8. Click File->New File and paste the following code **(Check for tab v space errors when copying and pasting)**

```
import socket
import time
import signal
import sys

def run_program():
      sock_l = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
      sock_l.bind(('0.0.0.0', 12345))
      sock_l.listen()
      print('Waiting for connection')
      conn, addr = sock_l.accept()
      print('Connected')
      with conn:
      data = conn.recv(1024)
      print(data.decode())

if __name__ == '__main__':
      original_sigint = signal.getsignal(signal.SIGINT)
      signal.signal(signal.SIGINT, exit)
      run_program()
```

9. Save the file, then select 'Run -> Run Module'. This is a slight variation to your server. It is waiting on port 12345 on the local lab machine.
10. From your PYNQ board, connect your client to
    a. Ip: local lab IP
    b. Port: 12345
11. Send the message "Hello world\n"
12. You should see it displayed in the Python terminal
13. Now open Wireshark
14. Double click 'Wi-Fi' or 'Ethernet' depending on how you connected to the network. You're now capturing a trace of the network which is only between your machine and the PYNQ board through the router. Look at a few of the traces. Notice which are between your PYNQ board and the local machine (check the IP addresses) and which involve the

router. There will only be a difference if you also connected the PYNQ board directly to your local machine.

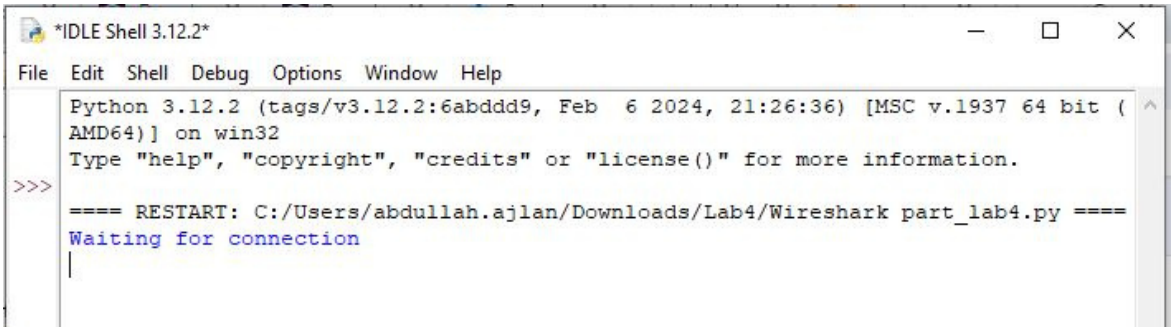
15. Where it says 'Apply a display filter' at the top, type 'tcp'
16. Repeat steps 9-11
17. Check the packet trace for any changes
    a. **What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the PYNQ board and the local machine? What is it in the segment that identifies the segment as a SYN segment?**

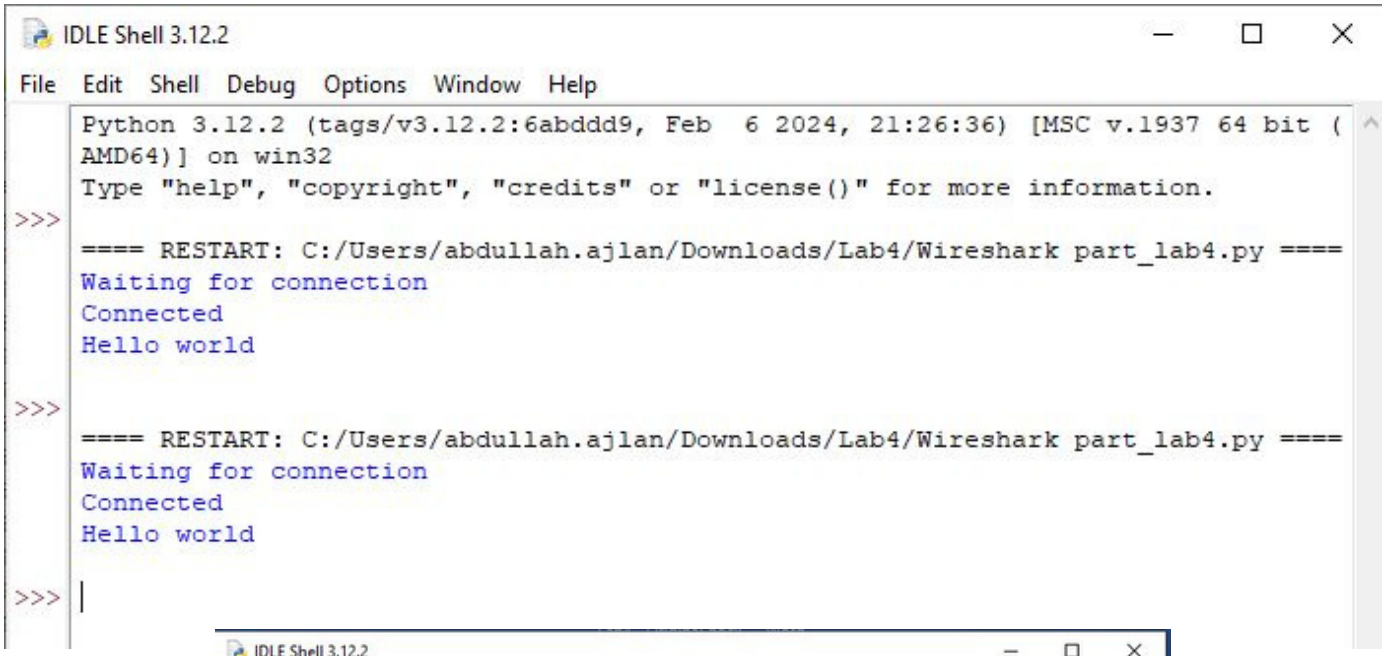
There are multiple TCP connections between the host at 192.168.2.99 and the host at 192.168.2.1, using different source and destination ports (9090, 51914, 52456, 51823, 41148, 12345).
The initial SYN segment for the connection involving port 41148 is seen in packet 17, indicating the start of a new TCP connection attempt from 192.168.2.99 to 192.168.2.1 with the source port 41148 and destination port 12345.
The SYN segment has the following characteristics:
Source IP: 192.168.2.99
Destination IP: 192.168.2.1
Source port: 41148
Destination port: 12345
Sequence number: 0
Flags: SYN
Window size: 64240
Options: MSS=1460, SACK_PERM, Timestamps (TSval and TSecr), Window scale (WS=64)
The SYN segment is retransmitted in packets 23, 26, and 30, indicating that the initial SYN has not been acknowledged by the destination (192.168.2.1), which could be due to packet loss, filtering, or the destination host not being available on port 12345.
Other connections (9090 ↔ 51914, 9090 ↔ 52456, 9090 ↔ 51823) are already established and exchanging data, as indicated by the PSH and ACK flags and the sequence and acknowledgment numbers.
The sequence and acknowledgment numbers in the packets indicate the relative byte ordering of the data being transferred. For example, in packet 1, the sequence number is 1, and the acknowledgment number is 1, indicating that one byte of data has been sent and one byte of data has been received, respectively.
The trace includes both data packets (indicated by PSH, ACK) and acknowledgment packets (indicated by ACK).
The window size advertised by the hosts (e.g., Win=1002, Win=8208) indicates the amount of buffer space available to receive data.

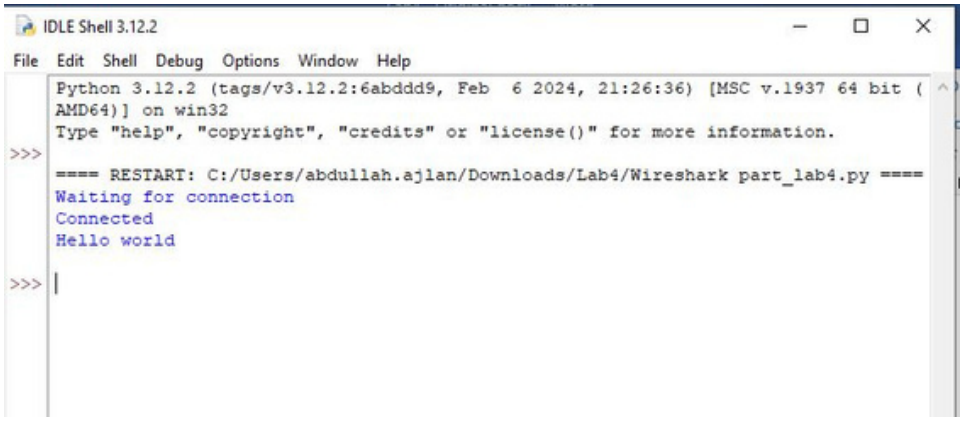
    b. Right click this trace and select 'Follow->TCP Stream'
    c. **Repeat a few times with different messages. Describe what's happening in the 5-10 steps of the TCP sequence for this communication. You can refresh your TCP flags here.**

TCP Three-Way Handshake (Connection Establishment)
SYN: The client sends a TCP segment with the SYN flag set to the server to initiate a new connection. This includes the initial sequence number for the connection.
SYN-ACK: The server responds with a TCP segment with both SYN and ACK flags set, acknowledging the client's SYN (with an ACK number set to the client's sequence number plus one) and providing its own initial sequence number.

ACK: The client sends an ACK segment back to the server, acknowledging the server's SYN segment (with an ACK number set to the server's sequence number plus one).
Data Transfer
After the handshake, data transfer can begin. The client and server exchange data packets with the PSH and ACK flags set. Each segment includes a sequence number, which is the byte number of the first byte of data in this segment, and an acknowledgment number, which is the next expected byte from the other side.
The receiver sends an ACK for the received segments. If multiple segments are received correctly, a single ACK can acknowledge all of them by specifying the next expected sequence number.
Connection Teardown (Graceful Close)
FIN: When the client has finished sending data, it sends a segment with the FIN flag set.
ACK: The server acknowledges the FIN segment from the client.
FIN: The server sends a FIN segment when it has finished sending data.
ACK: The client acknowledges the server's FIN segment.
In trace, can see the following steps related to data transfer and acknowledgment after the connection has been established:
The client (192.168.2.99) sends a PSH, ACK segment to the server (192.168.2.1) on port 52456, indicating that it is pushing data to the server.
The server acknowledges the receipt of this data with a PSH, ACK segment, also containing data to be pushed to the client.
The client sends an ACK segment, acknowledging the receipt of the server's data.
The sequence continues with data being pushed from the server to the client (packet 15), and the client acknowledging the data (packet 16). The client then attempts to initiate a new connection on a different port (packet 17), but there are retransmissions of the SYN segment (packets 23, 26, 30), indicating no response from the server for this new connection attempt.




IDLE Shell 3.12.2
File Edit Shell Debug Options Window Help
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb 6 2024, 21:26:36) [MSC v.1937 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==== RESTART: C:/Users/abdullah.ajlan/Downloads/Lab4/Wireshark part_lab4.py ====
Waiting for connection
Connected
Hello world
>>>
==== RESTART: C:/Users/abdullah.ajlan/Downloads/Lab4/Wireshark part_lab4.py ====
Waiting for connection
Connected
Hello world
>>>


IDLE Shell 3.12.2
File Edit Shell Debug Options Window Help
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb 6 2024, 21:26:36) [MSC v.1937 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==== RESTART: C:/Users/abdullah.ajlan/Downloads/Lab4/Wireshark part_lab4.py ====
Waiting for connection
Connected
Hello world
>>>