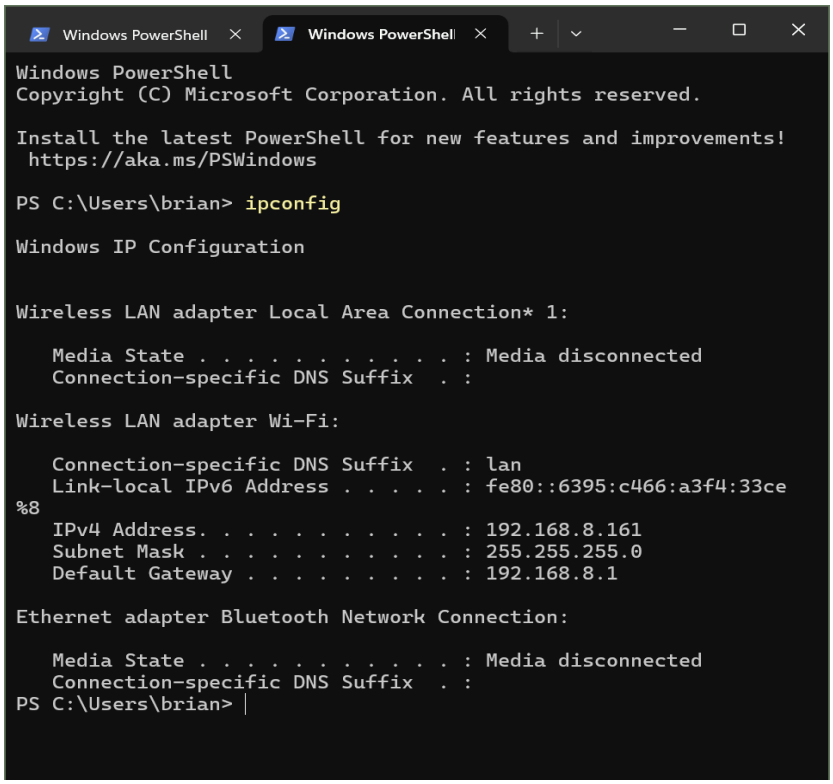
Lab 4 Report: Network Communication

Locating IP Addresses of Devices in your Network

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?
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.
Answer: 2 IP addresses

a. eth0
    i. Ipv4: 192.168.8.133
    ii. netmask 255.255.255.0
b. lo: Loopback
    i. Ipv4: 127.0.0.1
    ii. Netmask: 255.0.0.0

On your computer, 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)
Question: How many IPv4 addresses are assigned to this machine? What IPv4 address has the same netmask as the PYNQ board?



Question: Below, compile all the IP addresses of the PYNQ boards in your group:

1. My PYNQ board:
    a. IPV4: 192.168.8.133
2. Partners PYNQ board:
    a. IPV4: 192.168.8.165

PYNQ-PYNQ Communication with Python

Go through and complete the code. Answer the following questions.

Question: What does `socket.SOCK_STREAM` mean (hint: search the documentation link in the notebook)?

Answer: socket.SOCKET_STREAM means establish a TCP connection.

Question: What is the order of operations for starting a client socket and sending a message?

Answer:

1. Initialize the socket object
2. Call the connect() object and pass in an IP address as a string and a listening port as an integer.
3. Use the sendall() function to send a byte literal b'...''
4. Close the socket with .close()

Question: What is the order of operations for starting a server socket and receiving a message?

Answer:

1. Initialize the socket object
2. Call the bind() object and pass in an IP address as a string and a listening port as an integer.
3. Use the listenl() function to begin listening on that port.
4. Call the accept() function to accept any message from a given IP address
5. Call recv(BUFFER_SIZE) and specify the buffer size in the parameter
5. Close the socket with .close()

Wireshark

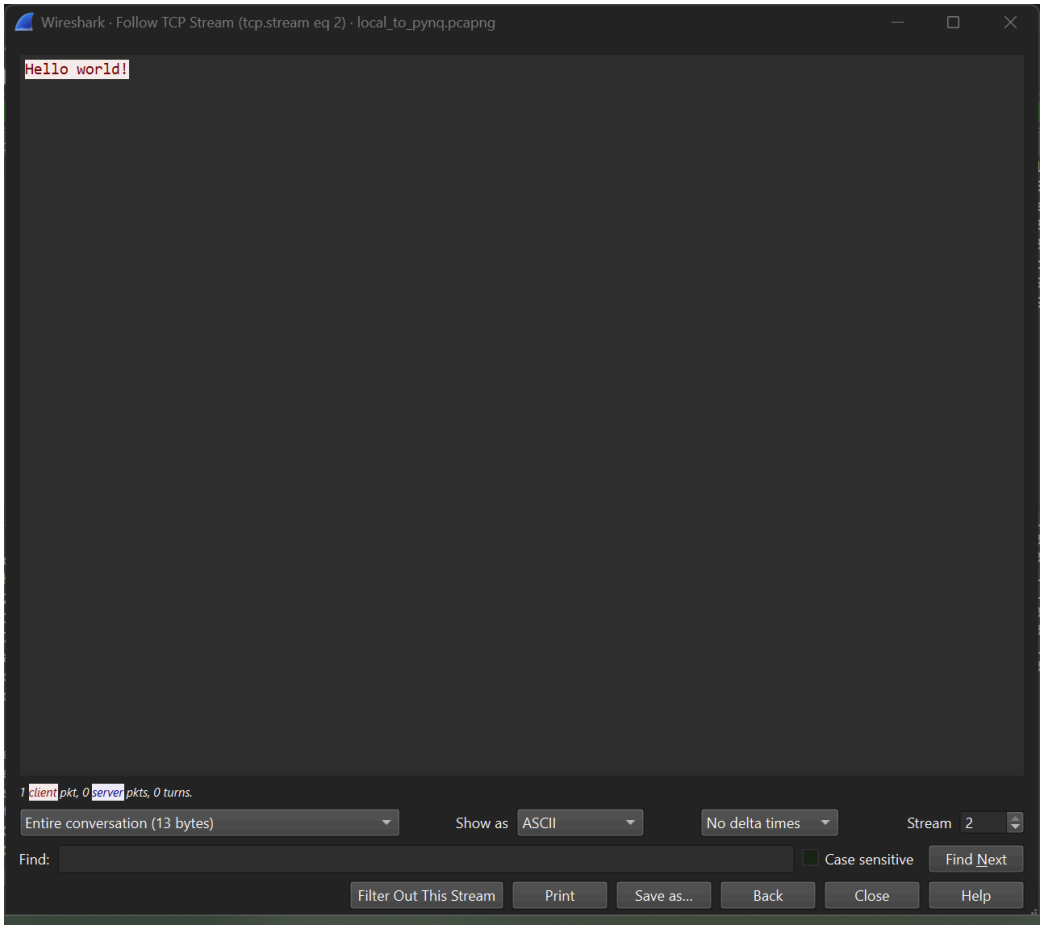
Question: What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the PYNQ board and the local machine? What is it in the segment that identifies the segment as a SYN segment?

Answer:

1. Sequence number for TCP SYN Segment(raw): 3485764467
2. Flags: 0x002 (SYN)

Question: Follow the Stream:

Answer:



In [1]:
```
import time
from pynq.overlays.base import BaseOverlay
import socket

base = BaseOverlay("base.bit")
btns = base.btns_gpio
leds = base.leds
```

Sockets

This notebook has both a client and a server functionality. One PYNQ board in the group will be the client and SENDS the message. Another PYNQ board will be the server and RECEIVES the message.

Server

Here, we'll build the server code to LISTEN for a message from a specific PYNQ board.

When we send/receive messages, we need to pieces of information which will tell us where to send the information. First, we need the IP address of our friend. Second, we need to chose a port to listen on. For an analogy, Alice expects her friend, Bob, to deliver a package to our back door. With this information, ALICE (server ip address) can wait at the BACK DOOR (port) for BOB (client ip address) to deliver the package.

Format of the information ipv4 address: ###.###.###.### (no need for leading zeros if the number is less than three digits) port: ##### (it could be 4 or 5 digits long, but must be >1024)

Use the socket documentation (Section 18.1.3) to find the appropriate functions https://python.readthedocs.io/en/latest/library/socket.html (https://python.readthedocs.io/en/latest/library/socket.html)

In [2]:
```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# TODO:
# 1: Bind the socket to the pynq board <CLIENT-IP> at port <LISTENING-PORT>
sock.bind(('', 12345))
sock.listen(1)
# 2: Accept connections
conn, addr = sock.accept()
print(f"Connected by  {addr}")
# 3: Receive bytes from the connection
while True:
    data = conn.recv(1024)
    if not data:
        sock.close()
        break
# 4: Print the received message
    print(data)
sock.close()
```

```
Connected by  ('192.168.8.165', 60818)
b'a'
b'b'
b'c'
b'd'
b'e'
```

Client

Now, we can implement the CLIENT code.

Back to the analogy, now we're interested in delivering a package to our friend's back door. This means BOB (client ip address) is delivering a package to ALICE (server ip address) at her BACK DOOR (port)

Remember to start the server before running the client code

In [4]:
```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# TODO:
# 1: Connect the socket (sock) to the <SERVER-IP> and choosen port <LISTEN

SERVER_IP = "192.168.8.161"
LISTENING_PORT = 12345

sock.connect((SERVER_IP, LISTENING_PORT))

# 2: Send the message "Hello world!\n"
try:
    sock.sendall(b'Hello world!\n')
except:
    sock.close()

# 3: Close the socket
sock.close()
```

On your server, you should see the message and then the server will shutdown! When we close a socket, both the client and the server are disconnected from the port.

Instead, change the function above to send 5 messages before closing.

The pseudocode looks like this

- connect the socket
- for i in range(5)
  - msg = input("Message to send: ")
  - send the message (msg)
- close the socket

Type *Markdown* and LaTeX: α^2