WES 237A: Introduction to Embedded System Design (Winter 2026)
Lab 4: Network Communication
Due: 2/16/2026 11:59pm

To report and reflect on your WES 237A labs, please complete this Post-Lab report by the deadline by submitting the following 2 parts:

- Upload your Lab 4 report, composed by a single PDF that includes your in-lab answers to the bolded questions in the Google Doc Lab and your Jupyter Notebook code. You could either scan your written copy or simply type your answer in this Google Doc. **However, please make sure your responses are readable.**
- Answer two short essay-like questions on your Lab experience.

All responses should be submitted to Canvas. Please also be sure to push your code to your git repo as well.

Locating IP Addresses of Devices in your Network

1. Connect the PYNQ board to the network switch over Ethernet.
2. 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.

There are two IPv4 addresses

The IPv4 address assigned to eth0 is 192.168.0.207. The netmask is 255.255.255.0

3. Connect your computer to the campus WiFi (UCSD-PROTECTED)
4. 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)
    a. **How many IPv4 addresses are assigned to this machine? What IPv4 address has the same netmask as the PYNQ board?**

2 IPv4 addresses are assigned to the machine.

The address that has the same netmask as the PYNQ board is 192.168.0.17

5. 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.**

Me: 192.168.0.207 Joy: 192.168.0.221

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

PYNQ-PYNQ Communication with Python

- Here, we're going to implement basic message-sending functionality in Python between two PYNQ boards.
- Download 'sockets_example.ipynb'
- Go through and complete the code. Answer the following questions.
    - **What does `socket.SOCK_STREAM` mean (hint: search the documentation link in the notebook)?**

It means that socket is going to use a TCP protocol

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

Create the socket Connect the socket to the server Send data Close socket when done sending data

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

Create the socket Bind the socket to the host and port Listen for a connection request Accept the connection Receive the data

Wireshark

1. On your local machine (or lab machine), install Wireshark
   If you are on Windows:
   a. Open the Firewall and Network Protection
   b. Click 'Allow an app through firewall.'
   c. Click 'Change Settings'
   d. Scroll down to 'Python'
   e. Select all 'Python' applications and all 'Public' boxes for each 'Python'

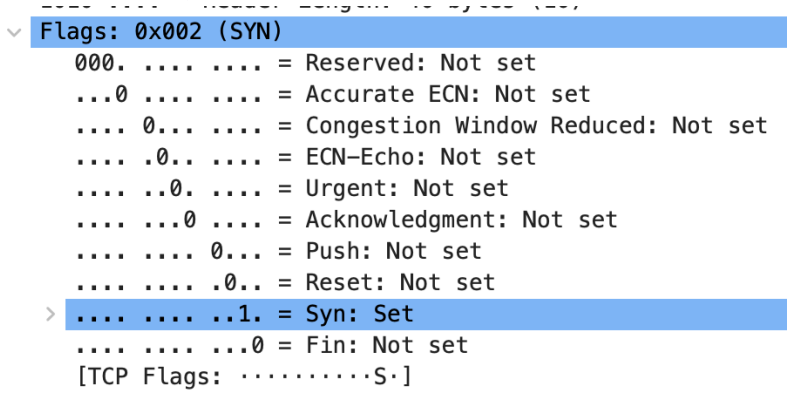
   f. Open the program `IDLE (Python 3.7 64-bit)`
   If you are on Linux / MacOS:
   g. Run the following command:
   sudo ufw status
   h. If you get Active status, run this:
   Sudo ufw allow 12345/tcp
   i. Run idle3 from the terminal (Linux) or search for IDLE in applications (Mac)
2. Click File->New 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()
```

3. Save the file, then select 'Run -> Run Module'. This is a slight variation of your server. It is waiting on port 12345 on the local lab machine.
4. From your PYNQ board, connect your client to
   a. IP: local lab IP
   b. Port: 12345

5. Send the message “Hello world\n”
6. You should see it displayed in the Python terminal
7. Now open Wireshark
8. Double click ‘Wi-Fi’ or ‘Ethernet’, depending on how you connected to the network. You’re now capturing a trace of the network traffic between your machine and the PYNQ board, via 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 connect the PYNQ board directly to your local machine.
9. Where it says ‘Apply a display filter’ at the top, type ‘tcp’
10. Repeat steps 9-11
11. 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?**

The relative sequence number is 0, the raw sequence number is 1005768029.
The segment that identifies the segment as a SYN segment is in the Flags section. The flag value is 0x002, which indicates that SYN is set.



    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.**

- The client sends a packet to the server to establish connection (SYN)
- The server sends a packet back to also initiated a connection, and also sends an ACK to acknowledge the client to server connection
- The client receives server connection and sends an ACK to confirm the server to client connection
- While connected the client sends [PSH, ACK] to tell the server to pass the data directly to the application, and the server responds with ACK to acknowledge that
- When the data finally sends with one of the [PSH, ACK] flags, the server responds with an ACK and the code running on the server processes the data and closes the connection
- When the connection is closed, the server sends a FIN to the client to indicate the connection is being torn down.

- For some reason with how the code is set up, the client does not send a FIN back to the server, which results in it sending a RST flag indicating the connection is being aborted, which then ends the connection between the server and the client

In [2]:
```
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 [11]:
```
conn.close()
```

In [12]:
```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# TODO:
# 1: Bind the socket to the pynq board <CLIENT-IP> at port <LISTENING-PO
CLIENT_IP = "192.168.0.207"
LISTENING_PORT = 8888

sock.bind((CLIENT_IP, LISTENING_PORT))


# 2: Accept connections
sock.listen(1)

conn, addr = sock.accept()
print(f"Accepted connection from {addr}")


# 3: Receive bytes from the connection
data = conn.recv(1024)


# 4: Print the received message
if data:
    print("Received message: ", data.decode('utf8'))
```

```
Accepted connection from ('192.168.0.221', 57098)
Received message:  hello
Received message:  my name is..
Received message:  i don't know...
Received message:  i need to sleep
Received message:  whatttttt
Client disconneted
```

Server accepting more than 1 message

In [ ]:
```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# TODO:
# 1: Bind the socket to the pynq board <CLIENT-IP> at port <LISTENING-PO
CLIENT_IP = "192.168.0.207"
LISTENING_PORT = 8888

sock.bind((CLIENT_IP, LISTENING_PORT))


# 2: Accept connections
sock.listen(1)

conn, addr = sock.accept()
print(f"Accepted connection from {addr}")


# 3: Receive bytes from the connection
# 4: Print the received message
while True:
    data = conn.recv(1024)

    if not data:
        print("Client disconneted")
        break
    print("Received message: ", data.decode('utf8'))

conn.close()
```

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 [ ]:
```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# TODO:
# 1: Connect the socket (sock) to the <SERVER-IP> and choosen port <LIST
SERVER_IP = '192.168.0.207'
LISTENING_PORT = 8888
sock.connect((SERVER_IP, LISTENING_PORT))

# 2: Send the message "Hello world!\n"
'''message = "Hello world!\n"
sock.sendall(message.encode('utf-8'))'''
# 2a: send messages before closing
for i in range(5):
    message = input("Your message?: ")
    sock.sendall(message.encode('utf-8'))

# 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