INTERNET OF THINGS
DEPARTMENT OF COMPUTER ENGINEERING
SANTA CLARA UNIVERSITY, CALIFORNIA
BEHNAM DEZFOULI, PHD

Santa Clara University
Internet of Things
RESEARCH LAB
SIOTLAB

Internet of Things

Lab 3
Setting up a WiFi Hotspot

Introduction

Introduction

- The **Raspberry Pi can be used as an access point**, acting to convert a wired Ethernet connection to a wireless one
- This can let you use the Raspberry Pi as a wireless router, similar to the one you are using in your home

- Linux allows you to do this very easily with two programs, **hostapd** and **`bridge-utils`**
  - **hostapd** allows you to configure device connections over the WiFi radio
  - **bridge-utils** bridges, or connects, the WiFi interface with your existing Ethernet connection

- In order to test our setup, we will also need to install **nginx** (a web server) and **apache-utils**, which will help us configure nginx

Installing Programs

Installing Programs

In your terminal, run:

- `sudo apt-get update`
- `sudo apt-get upgrade –y`
- `sudo apt-get install -y` **`hostapd bridge-utils apache2-utils nginx rng-tools`**
- **`reboot`**

Note: We install multiple packages in this line!

- These commands:
  - Update the RPi's list of available packages
  - Upgrade the packages installed on the RPi to the latest version
  - Install the new packages specified

Enable the AP

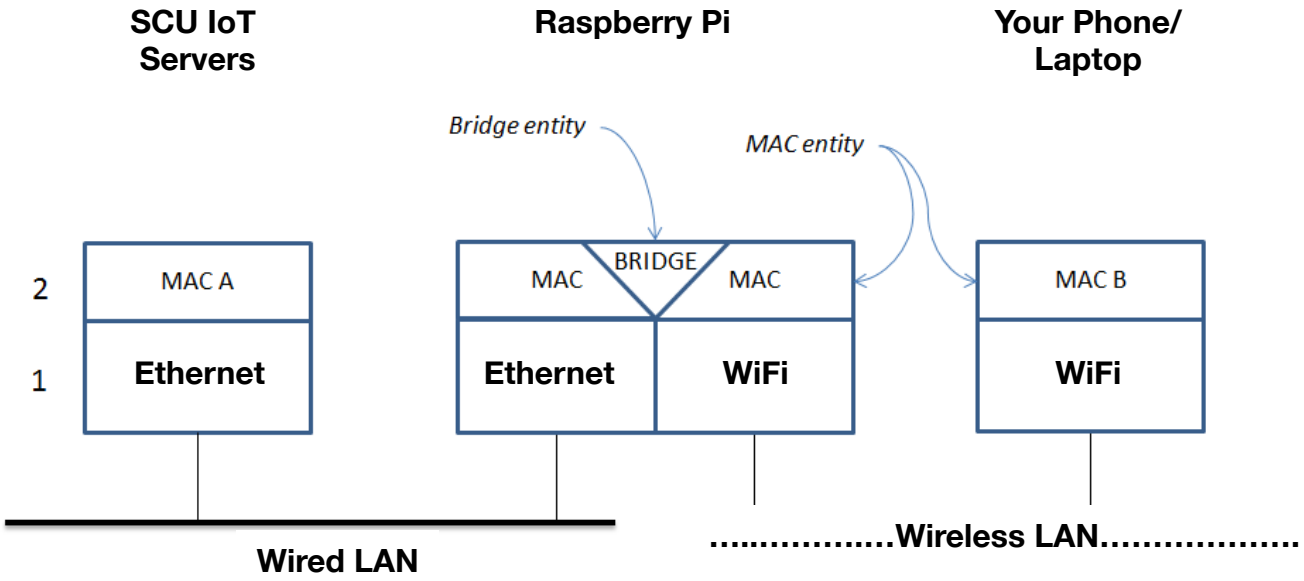
Introduction to Network Bridges

- A **network bridge** is a computer networking device that creates a single aggregate network from multiple communication networks or network segments
- This function is called **network bridging**
- Bridging is distinct from routing
  - Routing allows multiple networks to communicate independently and yet remain separate
  - Bridging connects two separate networks as if they were a single network

- **In the OSI model, bridging is performed in the data link layer (layer 2)**
- Therefore, the IP address of the bridged interfaces belong to the same subnet

Introduction to Network Bridges

The following figure shows the high level overview of bridging



Introduction to Network Bridges

- **How it works?**
  - Bridging requires two unused interfaces (such as **eth0**, you ethernet interface and **wlan0**, your WiFi interface) so you can connect them
  - You don't necessarily need to connect ethernet to WiFi, you could bridge ethernet and ethernet, WiFi and WiFi, or even involve Bluetooth interfaces
  - In this class, **we'll bridge Ethernet to WiFi**
  - Therefore**, both interfaces will be on the same IP subnet** and the **WiFi clients can use the DHCP server** we provided for you to receive their IP address

Enable the AP

Listing the Interfaces

- Please run `ifconfig` to see a list of interfaces available on the system
  - `ifconfig`
- You should see a list of interfaces:

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.16.230.20  netmask 255.255.255.0  broadcast 10.16.230.255
        inet6 fe80::4a8:81f8:7432:2433  prefixlen 64  scopeid 0x20<link>
        ether b8:27:eb:ae:42:9e  txqueuelen 1000  (Ethernet)
        RX packets 215159  bytes 283000560 (269.8 MiB)
        RX errors 0  dropped 31  overruns 0  frame 0
        TX packets 111702  bytes 9361223 (8.9 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 3  bytes 204 (204.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3  bytes 204 (204.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether b8:27:eb:fb:17:cb  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Enable the AP

Configuring the Bridge

- Now the `interfaces` file needs to be edited to adjust the various devices to work with bridging

➢ **`sudo vim /etc/network/interfaces`**

- At the bottom of the file, add:

➢ `#Bridge setup`

➢ `#bridge-utils must have been installed`

➢ `auto br0`

➢ `iface br0 inet dhcp`

➢ `bridge_ports eth0 wlan0`

- **Reboot the Raspberry Pi:**
  - ➢ **`sudo reboot`**

Enable the AP

Configure Hostapd

- Now, **we need to configure hostapd**:
  - **`sudo vim /etc/hostapd/hostapd.conf`**

- Add the following to the file (you don't need to add comments):

```
interface=wlan0                 # the interface used by the AP
bridge=br0                      # the bridge providing an internet connection
ssid=<username>                 # the AP ssid, replace <username> with yours
wpa_passphrase=<password>       # the AP password, replace <password>
hw_mode=g                       # use 2.4GHz WiFi
channel=7                       # the channel to use
wmm_enabled=0                   # enable Quality of Service
macaddr_acl=0                   # disable MAC address filtering
auth_algs=1                     # enable WPA authentication
wpa=2                           # set the version of WPA to use
wpa_key_mgmt=WPA-PSK            # some advanced WPA settings
wpa_pairwise=TKIP               # some advanced WPA settings
rsn_pairwise=CCMP               # some advanced WPA settings
```

DO NOT COPY AND PASTE! It wont work! Make sure there is not white space after each config line.

Enable the AP

Configure Hostapd

- Finally, we need to tell hostapd to use our new configuration file:
  - `sudo vim /etc/default/hostapd`
    - Find the line with #DAEMON_CONF, remove "#" and replace it with this:
  - `DAEMON_CONF="/etc/hostapd/hostapd.conf"`
    - vim tip: to jump to the line containing "DAEMON_CONF", enter command mode (<ESC>) and type:
      - /DAEMON_CONF<enter>
- Reboot the Raspberry Pi:
  - `sudo reboot`

Wrapping up Hostapd

- Once you reboot the system, hostapd should start as a background service if your configuration file does not include any errors
- You can use the following command to see if it is running:

➢ `service hostapd status`

- If the service is not active, then you can use the following command to start it:

➢ `service hostapd start`

- **You can also run hostapd manually using the following command to see any errors while starting this program:**

➢ `sudo hostapd /etc/hostapd/hostapd.conf`

Wrapping up Hostapd

- There should now be a functioning bridge between the wireless LAN and the Ethernet connection on the Raspberry Pi
- **Any device connected to the Raspberry Pi access point should have Internet access too**
  - **Connect your phone to the access point and test it!**

- If you run `ifconfig` again, the bridge, will have been allocated an IP address, set by the wired Ethernet's DHCP server

- **The `wlan0` and `eth0` no longer have IP addresses, as they are now controlled by the bridge**

Testing the Connection

Testing the Connection

Hosting the Webpage

- We don't just want internet through the RPi, we want to read data from the RPi too
- For this, **we need a webserver**, which is where **nginx** (which we installed in step 2)

Enabling nginx

- **NGINX**, pronounced like "**engine-ex**", is an **open-source web server**
- Since its initial success as a web server, is now also used as a reverse proxy, HTTP cache, and load balancer

- Because its roots are in **performance optimization at scale**, NGINX often outperforms other popular web servers in benchmark tests, especially in situations with static content and/or high concurrent requests

- NGINX is built to offer **low memory usage** and high concurrency
- While Apache is the most popular overall option, **NGINX is actually the most popular web server among high-traffic websites**

Testing the Connection

Enabling nginx

- You can verify NGINX installation by running the command below, which should print the latest version of NGINX

➢ `sudo nginx –v`

➢ `nginx version: nginx/1.6.2`

Enabling nginx

- Now you have a file in the default root folder of nginx, so let's enable the webserver and test it:

➢ `sudo systemctl enable nginx`

➢ `sudo systemctl start nginx`

- **How it works?**
  - You will learn more about `systemctl` in the next lab, but the above commands:
    - enable nginx to start at every reboot
    - start nginx immediately

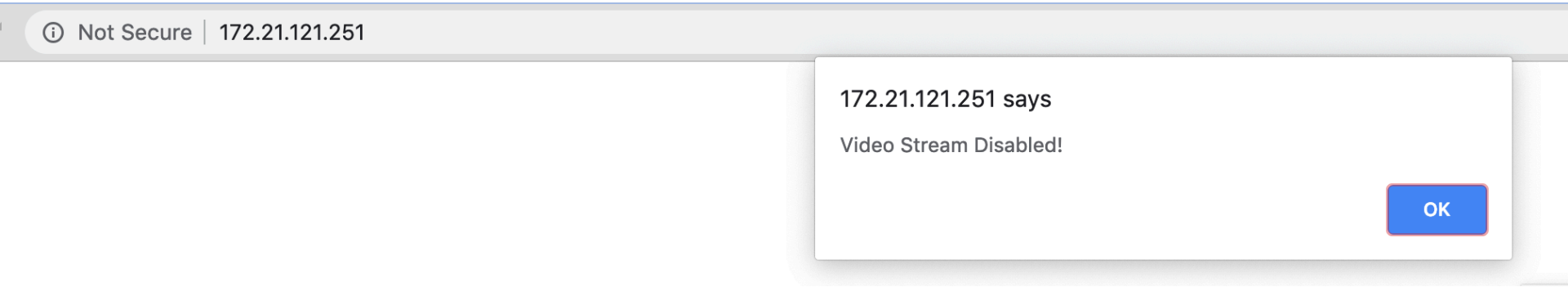
Testing the Connection

Testing the Server

- Now, determine the IP address of the Raspberry Pi:

➢ `ip addr`

- Navigate to that address in your connected device's web browser (such as phone/laptop)
- You should see a simple web page with a few buttons and an alert that pops up, saying `Video Stream Disabled!` which means everything is working as expected



Password Protecting the Server

Password Protecting the Server

Creating a Password

- While it's good to have an accessible web server, it's better to have control over who has access to your web server, **so let's add password protection**
- For this, we will use the **htpasswd** tool, which was installed alongside apache2-utils in step 2

- Run the following and **replace `pi` with your username**:

➢ `sudo htpasswd /etc/nginx/.htpasswd pi`

- **Then, type in your password twice when prompted**

- Now, we need to tell nginx where to find the .htpasswd file by editing the nginx config files

Password Protecting the Server

Configuring Nginx

- Open the main nginx config file:

```
sudo vim /etc/nginx/sites-enabled/default
```

- Inside this configuration file, you'll see the following lines:

```
server {
        listen 80 default_server;
        .........
}
```

- This is referred to as a **server block**

Password Protecting the Server

Configuring Password Prompts

- Insert the following two lines right under the second "listen" line to enable password protection:

```
auth_basic "Login Required";
auth_basic_user_file /etc/nginx/.htpasswd;
```

- Now write and close the file

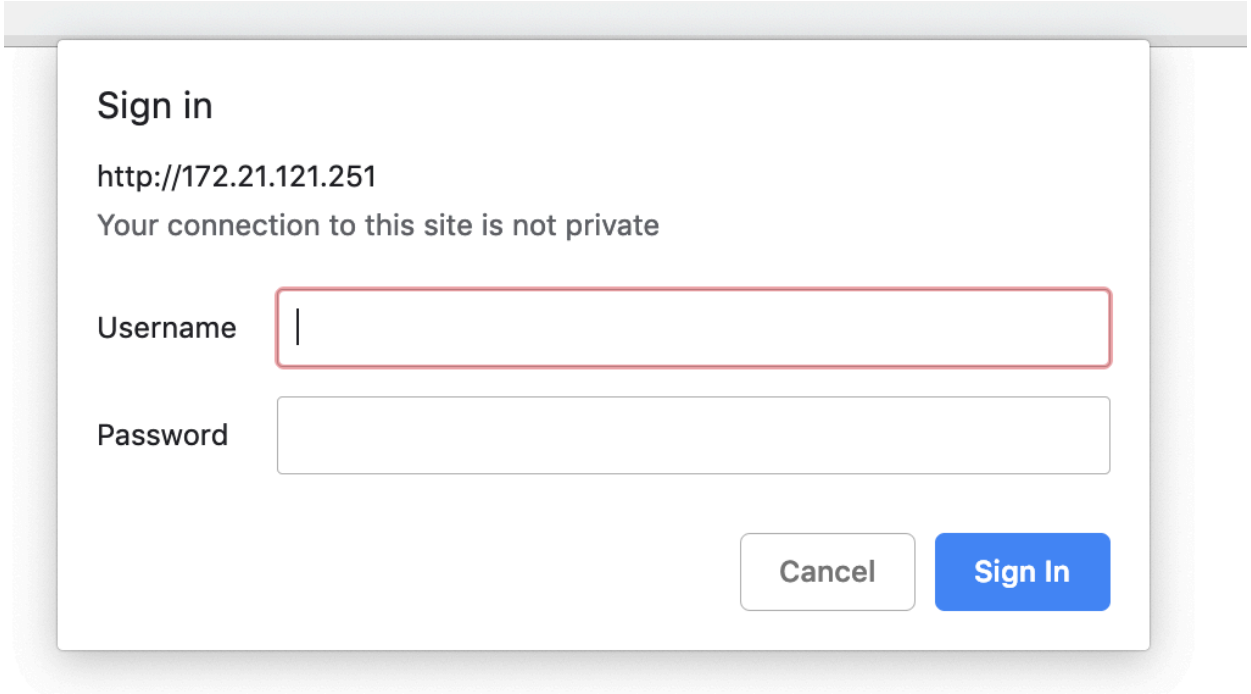
- Reload the nginx configuration file to apply your changes:

```
sudo systemctl reload nginx
```

Password Protecting the Server

Testing Password Authentication

- Now reload the page in your browser, and it should prompt you for a username and password, as shown below
- Enter your credentials and you should see the webpage
- This page by itself is useless until we are able to **pull data from the sensor**
- **To this end, we need to use Flask, a Python microserver**



Setting up Flask

Setting up Flask

Intro to Flask

- Flask is a small webserver that let's us run Python code when a user loads a page instead of only sending back a file

- **In our lab, we will use Flask to read data from our sensor every time the user requests data**
- Therefore, we must integrate Flask with our code from the last lab that read sensor data

Intro to Flask

- Flask is a **web application framework** written in Python
- Web Application Framework or simply Web Framework represents **a collection of libraries and modules**
  - Enables a web application developer to write applications without having to bother about low-level details such as protocols, thread management etc.

How a Flask app works

A typical Flask script looks as follows:

```
from flask import Flask


app = Flask(__name__)


@app.route('/')
def home():
    return "Hey there!"


if __name__ == '__main__':
    app.run(debug=True)
```

How a Flask app works

```
from flask import Flask
```

- You are making available the code you need to build web apps with flask
- **`flask` is the framework** here, while **`Flask` is a Python class** datatype
- In other words, *Flask* is the prototype used to create instances of web application or web applications

```
app = Flask(__name__)
```

- **Creates an instance of the *Flask* class** for our web app
- `__name__` is a special variable that gets as value the string "`__main__`" when you're executing the script

Setting up Flask

How a Flask app works

```
@app.route('/')
def home():
    return "Hey there!"
```

- The `route()` function tells the application **which URL should call the associated function**
- That **function is mapped to the home '/' URL**
- We are **defining a function that returns the string "Hey there!"**
- That means when the user navigates to localhost:5000, the `home()` function will run and it will return its output on the webpage
- If the input to the route method was something else, let's say '`/about/`', the function output would be shown when the user visited `localhost:5000/about/`

How a Flask app works

```
if __name__ == '__main__':
    app.run(debug=True)
```

- Python assigns the name "__main__" to the script when the script is executed
- That means the conditional statement is satisfied and the `app.run()` method will be executed

How a Flask app works

- Finally the **`run()`** method of Flask class runs the application on the local development server

➢ `app.run(host, port, debug, options)`

  - **`host`:** Hostname to listen on
    - Defaults to 127.0.0.1 (localhost)
    - **Set to '0.0.0.0' to have server available externally**
  - **`port`:** Defaults to 5000
  - **`debug`:** Defaults to false. If set to true, provides a debug information

Using Flask to Report Sensor Data

Starting the Flask App

- First, let's make a copy of the ultrasonic script from the last lab so we don't lose a working file:
  - `cp sample.py dashboard.py`

- Now, add the following lines to the very top of **dashboard.py**, above the `import` statements:

```
#!/usr/bin/env python3
from flask import Flask, request, jsonify
app = Flask(__name__)
```

- These lines import the required flask modules and allow us to run the script without specifying "python3" on the command line

Configuring Flask

- Now, we need to replace the infinite while loop in our function with Flask server code
- Leave the `measure()` function alone, but remove the loop at the very bottom
- Then, add the following after the measure() function:

```
@app.route('/measure/')
def getMeasurement():
   return jsonify({"result":measure()})

if __name__ == '__main__':
   app.run(host="0.0.0.0", port=9000, threaded=True)
```

- Lastly, remove all of the print lines, as we won't see the output on the web server anyway

Flask Explanation

- **How it works?**
  - This code assigned the url "`/measure`" to the result of the measure function, wrapped in JSON
  - The last line starts the server, allowing anyone to connect over port 9000 (we will go more into the host argument next lab)
  - The "threaded" argument is passed to allow compatibility with Google Chrome

Testing Flask

- If everything is working, you should be able to run the file with python3 **dashboard.py** and see the following:

```
* Running on http://0.0.0.0:9000/
(Press CTRL+C to quit)
```

- You can test your measurement over the WiFi connection by typing the following into your browser:

➢ http://<ip address>:9000/measure

- Replace "<ip address>" with the address of the Rpi
- Use the `ip addr` command on RPi to get its IP address

Tasks

- **Task 3-1.**
  - **What is the meaning of the following lines in the `interface` file?**
  - `auto br0`
  - `iface br0 inet dhcp`
  - `bridge_ports eth0 wlan0`

- **Task 3-2.**
  - WiFi's 5GHz band is less crowded compared to the 2.4GHz. **What needs to be changed to run your access point in the 5GHz band's channel 40?**
  - **How many channels are available in the U.S. in the 5GHz band?**

- **Task 3-3.**
  - Modify your code to show the **last 10 values of the ultrasound sensor** in the webpage.
  - **Demo the project to the TA.**