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 4
Video Streaming

Introduction

Introduction

- **In this lab, we will be connecting a USB webcam to the RPi!**
- We'll need to build a program to enable webcam streaming from source
- Then we'll configure the RPi to automatically start all the programs needed to survive a reboot, just like a real security camera!

- We need to install one new program, **libjpeg8-dev**
  - **libjpeg8-dev** is a set of libraries required for efficient manipulation of JPEG images
    - We will be using the camera to generate a continuous stream of JPEGs

Installing Programs

Installing Programs

In your terminal, run:

- `sudo apt-get update`
- `sudo apt-get upgrade –y`
- `sudo apt-get install -y cmake libjpeg8-dev`

- As explained in the previous lab, these commands:
  1. Update the RPi's list of available packages
  2. Upgrade the packages installed on the RPi to the latest version
  3. Install the new packages specified

Installing Programs

- Now reboot the Pi

➢ `sudo reboot`

- Normally, you don't need to reboot after installing new packages, but since we updated the entire system, it's very likely a kernel upgrade has occurred

- Whenever the Linux kernel gets upgraded, there are a few side effects that persist until the next reboot
  - For example, **udev**, the device monitoring **daemon** (a daemon is a background process) usually stops recognizing new USB devices
  - As we are using a USB webcam, we need to reboot to fix this issue

Building and Installing mjpg-streamer

Building and installing mjpg-streamer

- **mjpg-streamer** is a command line application that **copies JPEG frames from one or more input plugins to multiple output plugins**
- **It can be used to stream JPEG files over an IP-based network**
  - From a webcam to various types of viewers such as Chrome, Firefox, Cambozola, VLC, mplayer, and other software capable of receiving MJPG streams

- The project page is:

  `https://github.com/jacksonliam/mjpg-streamer`

Building and installing mjpg-streamer

- Inside your home folder, please run the following:

➢ `git clone https://github.com/jacksonliam/mjpg-streamer.git`

- This command tells **git** to download a copy of the project from github.com
  - Fun fact: git is a source control tool, but it's formal description is "the stupid content tracker"

- Next, change the current directory into the source folder:

➢ `cd mjpg-streamer/mjpg-streamer-experimental`

Building and installing mjpg-streamer

- Next, we need to build the project into an executable:
  - `sudo make`
  - `sudo make install`

- Run **`ls`** and see the file listing as below:

```
pi@raspberrypi:~/mjpg-streamer/mjpg-streamer-experimental$ ls
_build              input_uvc.so           output_file.so  start.sh
cmake               LICENSE                output_http.so  TODO
CMakeLists.txt      makedeb.sh             output_rtsp.so  utils.c
Dockerfile          Makefile               output_udp.so   utils.h
docker-start.sh     mjpg_streamer          plugins         www
input_file.so       mjpg_streamer.c        postinstall.sh
input_http.so       mjpg_streamer.h        README.md
input_raspicam.so   mjpg_streamer@.service  scripts
pi@raspberrypi:~/mjpg-streamer/mjpg-streamer-experimental$
```

Building and installing mjpg-streamer

- Take a look at the files that have been produced
- First, see there are several files ending in **.so**

What are .so files?

- A **file** with the .**SO file** extension is a **Shared Library file**
- They contain information that can be used by one or more programs to offload resources **so** that the application(s) calling the **SO file** doesn't have to actually provide the **SO file**

- For example, one SO file might contain information and functions on how to quickly search through the whole computer
- Several programs can then call upon that SO file to use that feature in their own respective programs.

Building and installing mjpg-streamer

- **In order to use shared object files, we need to include them in our system's shared object library path**

- This path is a collection of folders that Linux searches when trying to load library files
- On Linux, this includes the **/lib** and **/usr/lib** folders by default

- There are two options to include these .so files in our library path:
  1. Copy these files to a folder already in our path (such as `/lib` or `/usr/lib`)
  2. Use an **environment variable** (such as **`LD_LIBRARY_PATH`**) to temporarily add the object files to the path (must be done every time)

Building and installing mjpg-streamer

We pursue Option 1:

- **Let's copy the library files into the `/usr/lib/` folder:**
  - `sudo cp *.so /usr/lib/`

- The **make install** command already **placed the binary** into our **`/usr/bin`** folder, which is in our PATH
  - You can verify this with the following command:
    - `which mjpg_streamer`

Building and installing mjpg-streamer

- **Tip:**
  - The **`which`** command looks for the **first executable inside your path** that matches the name you give it, and tells you the full file path
  - To see a full, colon-separated list of the directories included in your path, run:
    - echo $PATH

Testing the Camera

Testing the Camera

- Now that we have the executable built, we should test to make sure everything is working as expected
- For this step, connect your phone or laptop to the RPi's access point (see the previous lab if you need a refresher on how to do this)

- The command to start the webcam server is below:

➢ **`mjpg_streamer -i "input_uvc.so" -o "output_http.so -n"`**

- On the device you have connected to the RPi's hotspot, enter the following URLs to test the cameras:
  - ➢ `http://<ip>:8080/?action=stream`
  - ➢ `http://<ip>:8080/?action=snapshot`
- If everything is working, you should see the camera's output on your device
- **Note: the webserver built in to `mjpg_streamer` defaults to port 8080**

Testing the Camera

- **How it works?**
  - This command calls your new executable, telling it to record the stream from the USB camera (handled by the **input_uvc.so** library)
  - It tells the executable to output the stream to an unsecured micro webserver on port 8080 (handled by the **output_http.so** library)
  - The **-n** flag tells mjpg_streamer to disable camera commands (such as powering it on, panning, tilting, zooming, etc.) on the web interface, as this is a security risk

Configuring Linux's systemd

Configuring systemd

- **Background:**
  - **systemd** is the modern **init** system used by most Linux Operating System Distributions (**distros**)
  - An init system is a collection of tightly integrated scripts that work together to manage system resources, spawn new processes, and automate the launching of other scripts
  - The init script is the first userspace process that launches on the system, and the last one to exit at shutdown

- **It's systemd's ability to automatically launch processes that we want to take advantage of now**
  - **Our goal is to automatically enable the camera when the RPi boots up**
  - **If the RPi gets updated or crashes and reboots, we don't have to manually run the stream program**

Configuring systemd

- The authors of the mjpeg-streamer repo made this easy for us by providing a sample **unit file**, named **mjpg_streamer@.service**
- This unit file is in your project folder where the shared libraries were

- The provided service file is not perfect, but it is close to what we need
- Edit the file, and replace the line starting with "ExecStart" with the following (the last flag is a lowercase "L" not a "1"):
  - `ExecStart=/usr/local/bin/mjpg_streamer -i input_uvc.so -o "output_http.so -n -l localhost"`
    - Make sure the code is on one line!

Configuring systemd

- **How it works?**
  - This command is very similar to the one in previous slides, but we've added the **–l** flag
  - The **-l localhost** option tells mjpg_streamer to only expose the stream to localhost so other devices on the network cannot see the stream
  - Later, we'll use **nginx** to re-expose the stream, but behind a username and password
  - For now you won't be able to see any streams

Configuring systemd

- Additionally, we need to add three more lines directly under the `ExecStart` entry:
  - `RestartSec=3`
  - `Restart=always`
  - `Type=simple`
- These options tell systemd to keep trying to restart the service every 3 seconds if it fails, in case the camera gets unplugged temporarily

- **Remove the "user=…" line to enable root permissions on execution**

Configuring systemd

- Your completed unit file should look like the following, **with ‘pi’ replaced with your username:**

```
[Unit]
Description=A server for streaming Motion-JPEG from a video capture device
After=network.target

[Service]
User=pi
ExecStart=/usr/local/bin/mjpg_streamer -i input_uvc.so -o "output_http.so -n -l localhost"
RestartSec=3
Restart=always
Type=simple
[Install]
WantedBy=multi-user.target
```

- Now, we need to add this file to systemd's path to enable it

Configuring systemd

- Then we need to place it in **/lib/systemd/system**
  - `sudo cp mjpg-streamer-experimental/mjpg_streamer@.service /lib/systemd/system/mjpg-streamer.service`

- Now we need to enable the service:

- `sudo systemctl enable mjpg-streamer`

- Now it's time to reboot
- If you did everything correctly, you won't be able to see the stream, but there's another way you can test

Configuring systemd

- After rebooting, run:

➢ `sudo systemctl status mjpg-streamer`

- If you did everything correctly, you should see the following, which indicates the camera service is running!

```
pi@raspberrypi:~$ sudo systemctl status mjpg-streamer.service
● mjpg-streamer.service - A server for streaming Motion-JPEG from a video captur
   Loaded: loaded (/lib/systemd/system/mjpg-streamer.service; enabled; vendor pr
   Active: active (running) since Mon 2019-02-25 18:54:57 PST; 1min 42s ago
 Main PID: 1079 (mjpg_streamer)
   CGroup: /system.slice/mjpg-streamer.service
           └─1079 /usr/local/bin/mjpg_streamer -i input_uvc.so -o output_http.so

Feb 25 18:54:57 raspberrypi mjpg_streamer[1079]:  o: HTTP Listen Address..: loca
Feb 25 18:54:57 raspberrypi mjpg_streamer[1079]:  o: username:password....: disa
Feb 25 18:54:57 raspberrypi mjpg_streamer[1079]:  o: commands.............: disa
Feb 25 18:54:57 raspberrypi mjpg_streamer[1079]: MJPG-streamer [1079]: www-folde
Feb 25 18:54:57 raspberrypi mjpg_streamer[1079]: MJPG-streamer [1079]: HTTP TCP
Feb 25 18:54:57 raspberrypi mjpg_streamer[1079]: MJPG-streamer [1079]: HTTP List
Feb 25 18:54:57 raspberrypi mjpg_streamer[1079]: MJPG-streamer [1079]: username:
Feb 25 18:54:57 raspberrypi mjpg_streamer[1079]: MJPG-streamer [1079]: commands.
Feb 25 18:54:57 raspberrypi mjpg_streamer[1079]: MJPG-streamer [1079]: starting
Feb 25 18:54:57 raspberrypi mjpg_streamer[1079]: MJPG-streamer [1079]: starting
lines 1-17/17 (END)
```

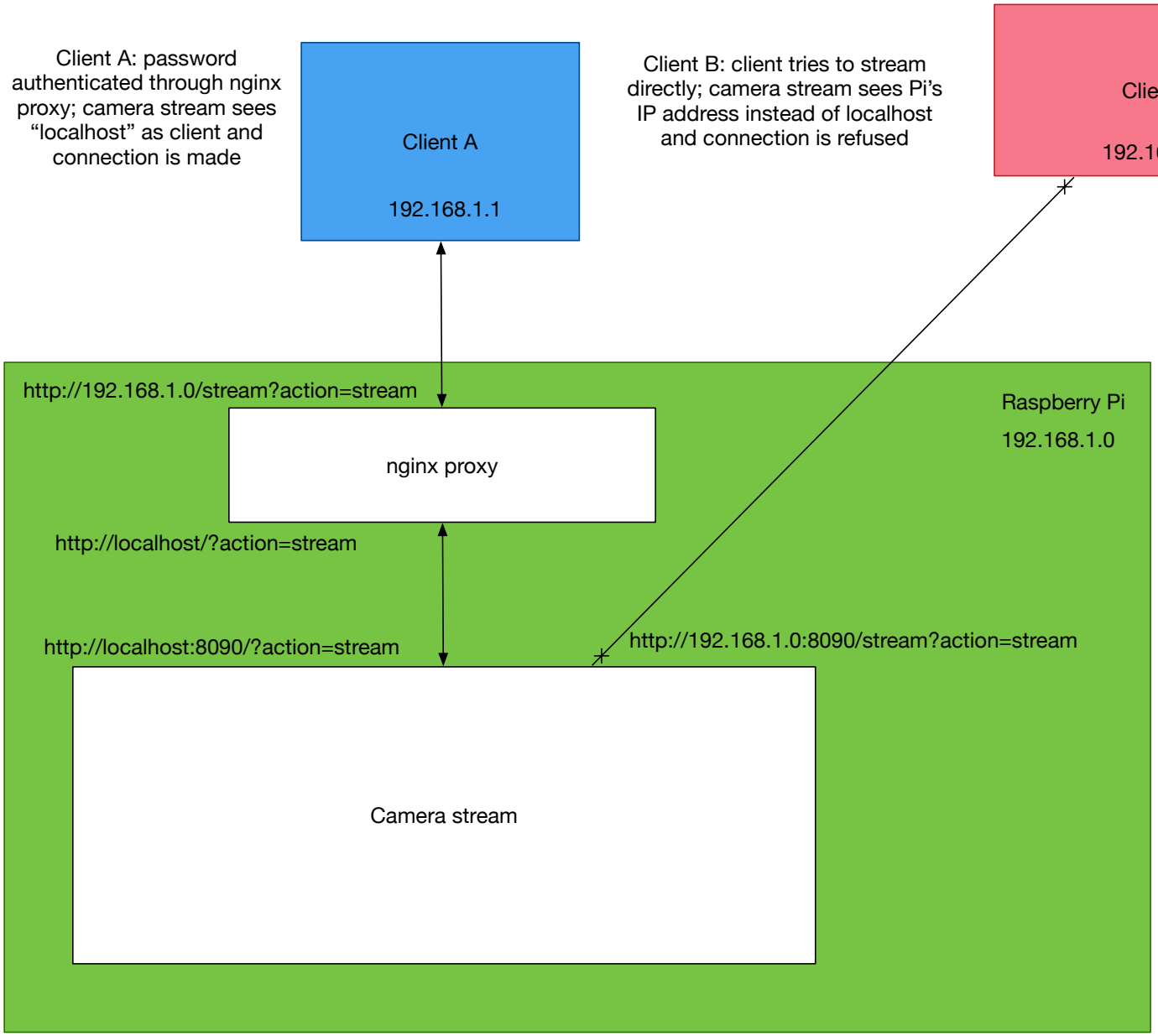
Proxying Video Through NGINX

Proxying Video through NGINX

- For security reasons, we disabled the video stream on all hosts other than "localhost"--that is to say only the RPi can see it's own stream, but any devices connected to it cannot
- That is very secure, but without a monitor or display attached to the Pi, it's not very useful

- **We want other devices to be able to see the stream, but only if they have the right credentials: your username and password**
- In an earlier lab, we set up Nginx to serve a website for the stream, without the video, and used an `.htaccess` file to configure logins
- Now, we can use that same login permission configuration with a **proxy_pass** directive to allow Nginx to serve the stream
- See the next diagram to understand what is happening

Proxying Video through NGINX



- **Note: The IP addresses are not realistic and you do not need to configure them**
- **IP addresses are assigned automatically to your connected systems!**

Proxying Video through NGINX

- As you can see in the diagram, **clients can view the stream only through the Nginx proxy, which requires password authentication**
- Let's add the necessary lines to the configuration file (located at `/etc/nginx/sites-enabled/default`), inside the server block but outside the `location` block:

```
location /stream/{
        proxy_pass  http://localhost:8080/;
}
```

- Now we have to reload the Nginx configuration file:

➢ `systemctl reload nginx`

➢ Now, using your phone you can navigate to the following address to see the live stream:

`http://<IP_Address>/stream/?action=stream`

Proxying Ultrasonic Data Through NGINX

Proxying Ultrasonic Data through Nginx

- This step is very similar to the previous step, as we are achieving the same goal: password protecting our data through Nginx proxying

- Open the nginx configuration file just as before, and add the following location block:

```
location /measure/{
        proxy_pass http://localhost:9000/measure/;
}
```

- Now **reload** nginx as you did in the previous step, and run
  - `python3 dashboard.py`

Proxying Ultrasonic Data through Nginx

- If you navigate to `http://<RPi's IP address>/measure/` you'll be able to see your sensor data with a password prompt
- The prompt may not appear if you've logged in recently in that browser
- To test it, use an Incognito tab or clear your cookies first

- However, you can still navigate to the old url (with the :9000) without a password, because Flask currently accepts connections from any host

- We want to limit the Flask server to localhost connections only, and use Nginx's proxy to open up the server to anyone with the correct password, just like the video stream

- Therefore, we need to change one line in `dashboard.py`
  - Replace `0.0.0.0` in the last line with `localhost`, and you've secured the server

Enabling Flask as a Service

Enabling Flask as a Service

- Nginx is enabled as a service, and we wrote a service file for mjpg-streamer, so both of those services will automatically start when the RPi is booted

- The final step in the project is to enable Flask as a service so the Ultrasonic Sensor survives reboots

- Without the ultrasonic data, only part of the system is operational after a reboot, so **let's write a unit file** and enable it

➢ `vim ultrasonic.service`

Enabling Flask as a Service

See the unit file below

```
[Unit]
Description=Ultrasonic Data Service
After=network.target
[Service]
ExecStart=/usr/bin/dashboard
TimeoutSec=3
Restart=always
Type=simple
[Install]
WantedBy=multi-user.target
```

Enabling Flask as a Service

- Now, copy the service file into system's path and enable it:

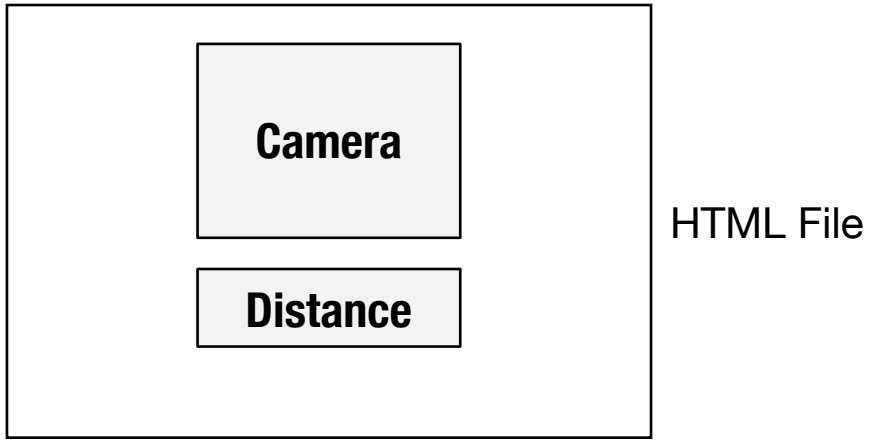
```
sudo cp ultrasonic.service /lib/systemd/system/ultrasonic.service
sudo systemctl enable ultrasonic.service
```

- Finally, we need to make our **dashboard.py** executable and place it into our PATH:

```
chmod +x dashboard.py
sudo cp dashboard.py /usr/bin/dashboard
```

Deliverable

- Task 4-1.
  - Write an HTML file to show the camera stream and ultrasound value as follows:



  - Note: The next few slides provide sample codes for the HTML file!

  - Demo the project to the TA

HTML File

Note: You do not have to use the provided code snippets
You can write your own simpler HTML file!

HTML File

- This is a sample HTML page we use to show the information:

```
<html>
    <head>
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <!-- Ensure the page is zoomed out when first loaded -->
    </head>
    <body>
        <center>
            <img id="stream" src = "/stream/?action=stream"
                 onerror="alert('Video stream disabled.')"/>
            <!-- Include the video stream in the page, alert user if it's not working -->
            <br>
            Distance:
            <div id="reading”>
                <!-- Create a placeholder for the measurement later -->
            </div>
            <br>
            <br>
            <a id="downloader" href="http://172.21.121.251/stream/?action=snapshot"
               download="">
                <!-- Create a download button for saving snapshots -->
                Save Snapshot
            </a>
            </center>
        </body>
        <script> … </script>
        <style> … </style>
</html>
```

Note:

- You do not have to use this file!
- You can write your own, simpler or fancier version of the HTML file!

HTML File

- Below, find the JavaScript code embedded in the HTML page
- Blue code represents the success state, red implies an error has occurred

```
<script>
function showMeasurement(){
var date = new Date();                                          // Get the current date
var reading = document.getElementById('reading');               // Store a reference to the measurement placeholder
var downloader = document.getElementById('downloader');         // Store a reference to the download button link
fetch('/measure/')                                              // Send a get request to the '/measure/' URL
  .then(response => response.json())                            // Convert the response text to JSON
  .then(function(response) {                                    // After it's converted
     var filename = date.toDateString()+date.toLocaleTimeString()   // Turn date into a string for the filename
     filename += "_cm_"+response.result+".jpeg"                  // Append the Sensor Distance measurement to the filename
     filename = filename.replace(/\s/g,"_");                    // Replace spaces in the dates with underscores
     downloader.download=filename;                          // Apply this filename to the downloader's download attribute
     reading.innerHTML= response.result + " cm";                // Update the measurement placeholder on the site
     reading.className = 'success';                             // Set the status to success (turn the text green)
  })
  .catch(function() {                                                   // Error Handling
    reading.innerHTML="Error Fetching Measurement";                     // Show error message in placeholder
    reading.className = 'error';                                        // Make placeholder text red
    var filename = date.toDateString()+date.toLocaleTimeString()    // Turn date into a string for the filename
    filename += "_cm_unknown.jpeg"                                  // Append measurement placeholder to the filename
    filename = filename.replace(/\s/g,"_");                         // Replace spaces in the dates with underscores
    downloader.download=filename;                                   // Set the status to success (turn the text green)
  });
}
window.setInterval(function(){ showMeasurement(); }, 500); // Tell the browser to call our function every 500ms
</script>
```

HTML File

- Below, find the CSS styling embedded in the html page to make the page fancier!

```
<style>
img#stream{
        /* Size the stream so it's no bigger
        than 90% of the screen, and no bigger
        than the camera resolution */
        width: 90%;
        max-width: max-content;
}
div#reading {
        /* Allows the placeholder and label
        to share a line in the html */
        display: inline-block;
}
.success {
        /* Turns text green */
        color: green;
}
.error {
        /* Turns text red */
        color: red;
}
</style>
```