CS 498 Internet of Things
Lab 3

Team Ace

Alexey Burlakov (NetID: alexeyb2)
Christopher Lewis (NetID: calewis4)
Li Yi (NetID: liyi2)
Pui Sze Pansy Ng (NetID: ppn2)
Zhijie Wang (NetID: zhijiew2)

Introduction **3**

Task 1: Deploy the Network **3**

GNS3 Installation and GNS3 VM configurations 3

Implementation of OpenWRT Access points 3

Implementation of OpenWISP2 wireless controller 7

Implementation of IOT Gateway (MQTT Broker) 8

Implementation of Data warehouse (SQLite + Flask) 9

Installing Flask: 11

GNS3 Topology 12

Task 2: Deploy the Application **14**

Implementation of Shopper Gaze application 14

Implementation of MQTT publisher and MQTT gateway 16

Problems that encountered **18**

Contribution **18**

Introduction

In this lab, we will build an infrastructure management for IoT devices. We leverage GNS3 to run real router images on the local computer. We will design and create a large IoT infrastructure for large retailers, and simulate deploying a large number of sensors to track and monitor goods in the store, as well as gaze tracking to monitor shopper behavior and collect datasets to analyze how the product placement is doing. We will deploy a set of access points to communicate with your IoT devices and develop infrastructure to store this data locally in a data warehouse.

Task 1: Deploy the Network

GNS3 Installation and GNS3 VM configurations

We have downloaded VMware Player and VirtualBox to setup virtualization software. Then, we downloaded GNS3 for the GUI and import GNS3 image into VirtualBox

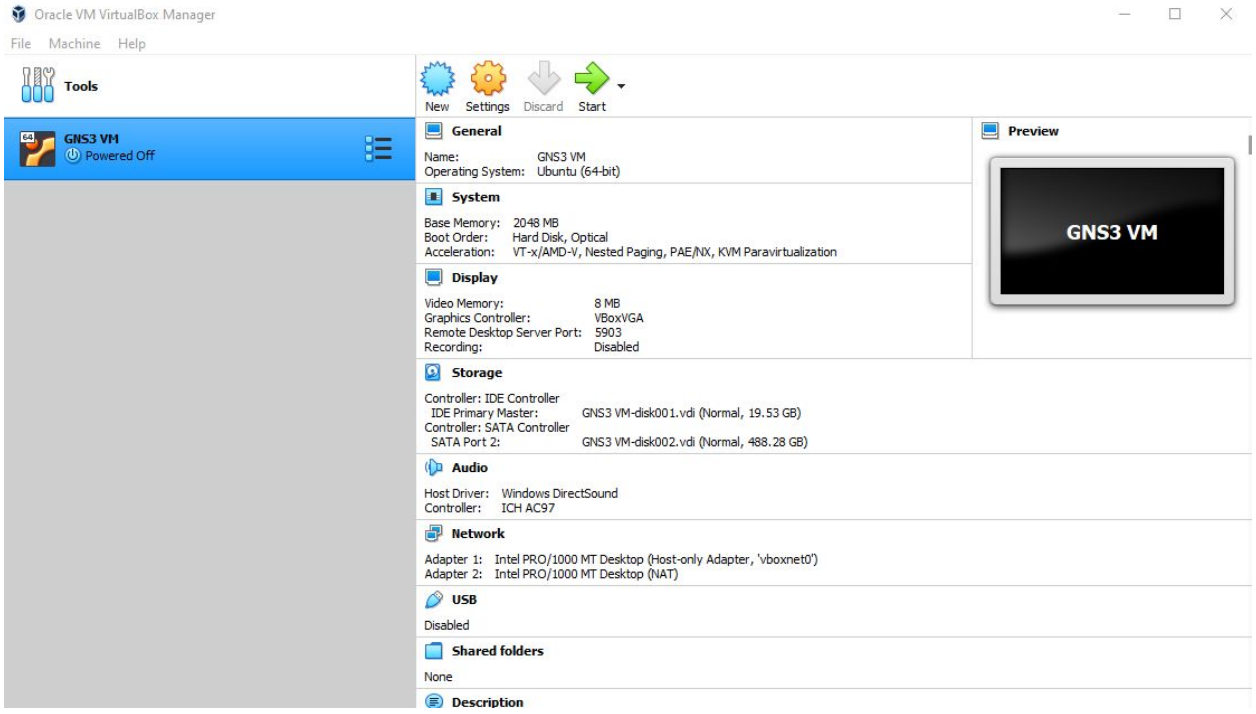

Figure 1.1: Screenshot of VirtualBox after import GNS3 image

Implementation of OpenWRT Access points

We followed https://openwrt.org/docs/guide-user/virtualization/virtualbox-vm to install and build OpenWRT VMs

First, we installed VirtualBox in Ubuntu 18.04 environment. After it, we selected combined-ext.img as OpenWrt image, and converted it to VBox drive.

```
VBoxManage convertfromraw --format VDI \
openwrt-18.06.1-x86-64-combined-ext4.img \
openwrt-18.06.1-x86-64-combined-ext4.vdi
```

We created a new VM by importing the VBOx drive file, and assigned 512MB RAM.
After the VM is successfully created, We changed Network settings for OpenWrt VM, added an additional network adapter (adapter 1) and attached it to Host-Only Adapter. The Adaptor Type is: Intel PRO/1000 MT Desktop (82540EM).
Starting the VM, change IP address to 192.168.56.2 by issuing

```
uci set network.lan.ipaddr='192.168.56.2'
uci commit
```

```
uci batch <<EOF
set network.mng=interface
set network.mng.type='bridge'
set network.mng.proto='static'
set network.mng.netmask='255.255.255.0'
set network.mng.ifname='eth0'
set network.mng.ipaddr='192.168.56.2'
delete network.lan
delete network.wan6
set network.wan=interface
set network.wan.ifname='eth1'
set network.wan.proto='dhcp'
EOF
```

```
Uci commit && reboot
```

uci setAfter Ip address and network configuration, we can SSH to OpenWrt VM by

```
Ssh root@192.168.56.2
```

The last step is: installing VM's luci: `opkg update && opkg install luci`

After all, we can verify the installation by open browser and connect to
http://192.168.56.2/cgi-bin/luci

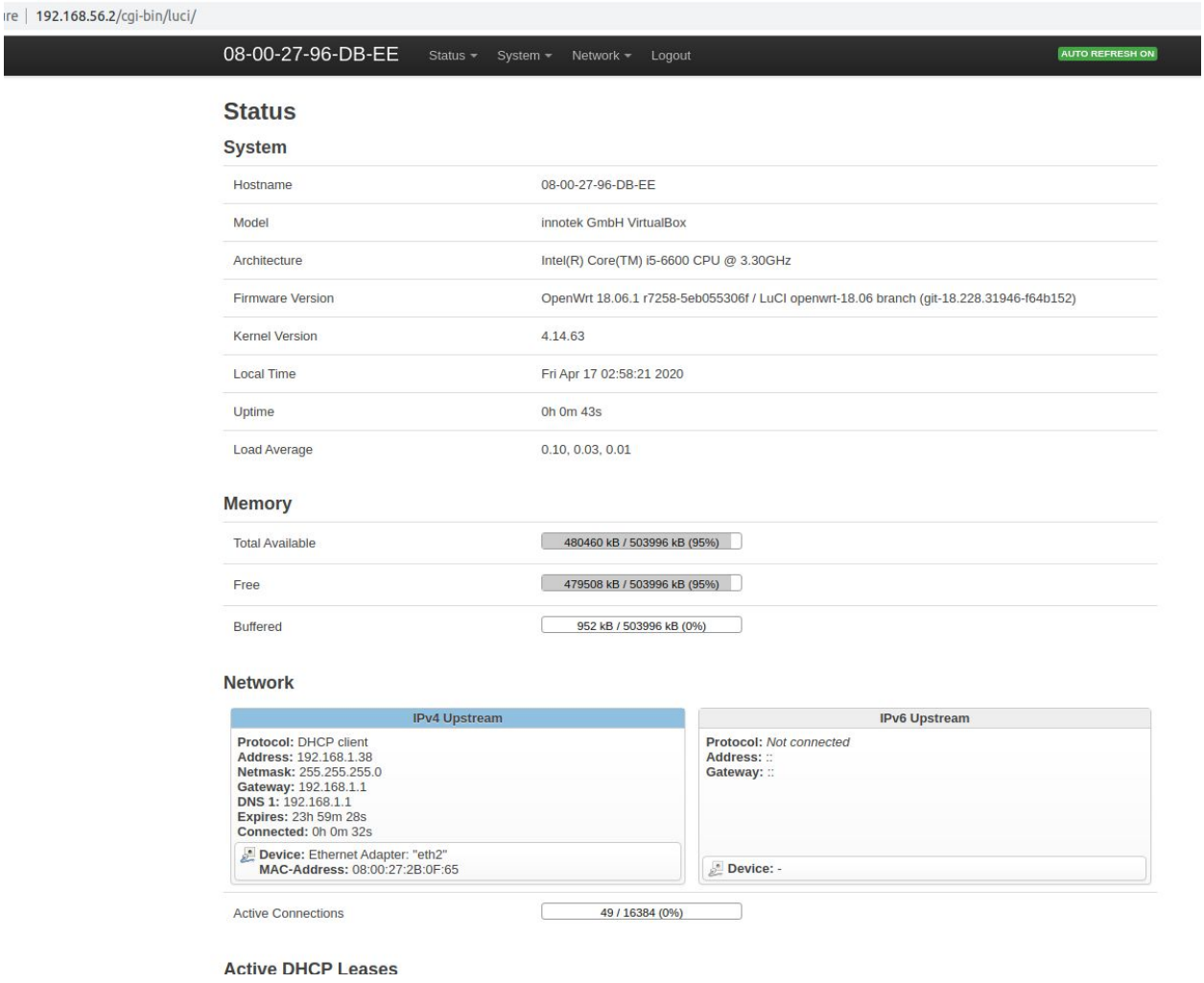

Figure 1.2: screenshot of 192.168.56.2

Next, we need to install OpenWisp-config on OpenWrt instances.

First, issuing command

```
opkg update
```

```
opkg install
http://downloads.openwisp.io/openwisp-config/latest/openwisp-config-openssl_0.4.6a-1_all.ipk
```

After OpenWisp-config has been installed with no errors, we can proceed to configure OpenWisp on OpenWrt instances.

```
vi /etc/config/openwisp
```

```
# For more information about the config options please see the README
# or https://github.com/openwisp/openwisp-config#configuration-options

config controller 'http'
    #option url 'https://openwisp2.mynetwork.com'
    #option interval '120'
    #option verify_ssl '1'
    #option shared_secret ''
    #option consistent_key '1'
    #option mac_interface 'eth0'
    #option merge_config '1'
    #option test_config '1'
    #option test_script '/usr/sbin/mytest'
    #option uuid ''
    #option key ''
    list unmanaged 'system.@led'
    list unmanaged 'network.loopback'
    list unmanaged 'network.@switch'
    list unmanaged 'network.@switch_vlan'
    # curl options
    #option connect_timeout '15'
    #option max_time '30'
    #option capath '/etc/ssl/certs'
```

We changed the url to ‘https://192.168.56.5’, changed shared_secret to the one shown in Organization settings. The list of organizations is at /admin/openwisp_users/organization/.

Run “`/etc/init.d/openwisp_config start`“ to start OpenWISP agent

After the above configurations, we can verify devices are added under Devices.

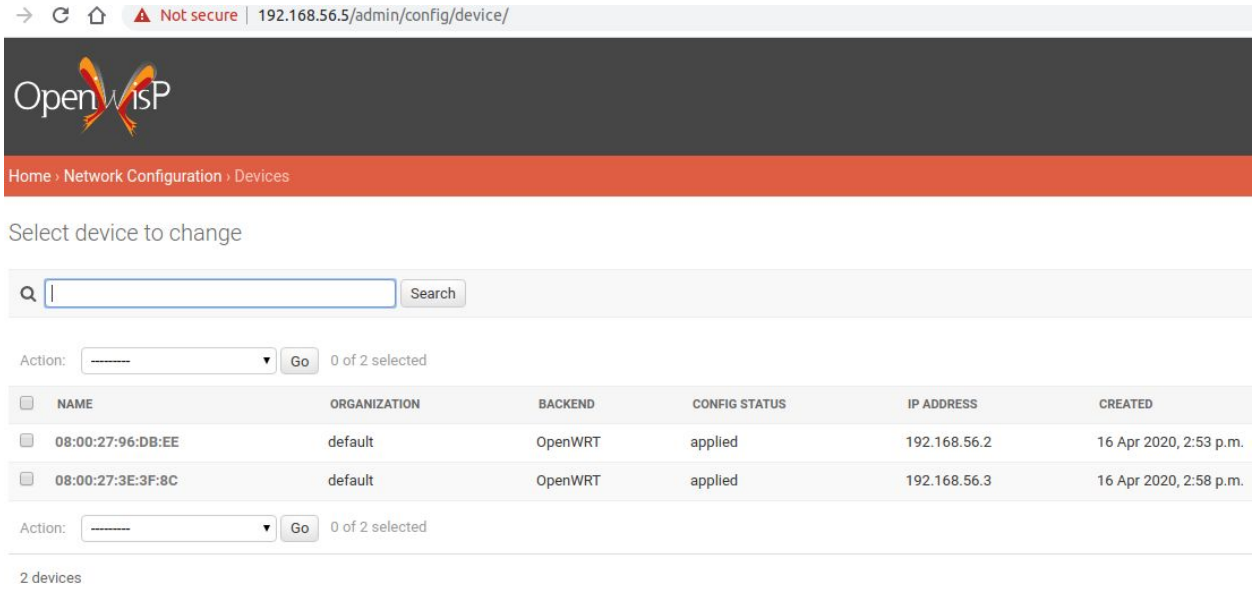

Figure 1.3: screenshot of OpenWISP

Implementation of OpenWISP2 wireless controller

We followed https://github.com/openwisp/vagrant-openwisp2 to create and install OpenWISP2 VM.
First, we downloaded and installed VirtualBox. After it, we installed Vagrant 2.2.6. The last step is installing Ansible

```
$ sudo apt update

$ sudo apt install software-properties-common

$ sudo apt-add-repository --yes --update ppa:ansible/ansible

$ sudo apt install ansible
```

After installing Ansible, we cloned the repository from Github. The vagrantfile and requirements.yml for is in the vagrant-openwisp2 directory.

Issuing command to install Ansible roles for this profile:

```
ansible-galaxy install -r requirements.yml
```

At last, issue command `vagrant up` to create and configure OpenWisp2 VM.

We verified the result by logging in `https://192.168.56.5/admin`

Implementation of IOT Gateway (MQTT Broker)

First of all, we use Vagrant to create a Ubuntu 14.04 virtual environment to install Mosquitto broker.
After Ubuntu 14.04 VM is created, issue command:

```
$ sudo apt-add-repository ppa:mosquitto-dev/mosquitto-ppa

$ sudo apt-get update
```

Issue command to install mosquitto

```
$ sudo apt-get install -y mosquitto lobmosquitto-dev mosquitto-clients
```

We can verify the service status by:

```
$ sudo service mosquitto status
```

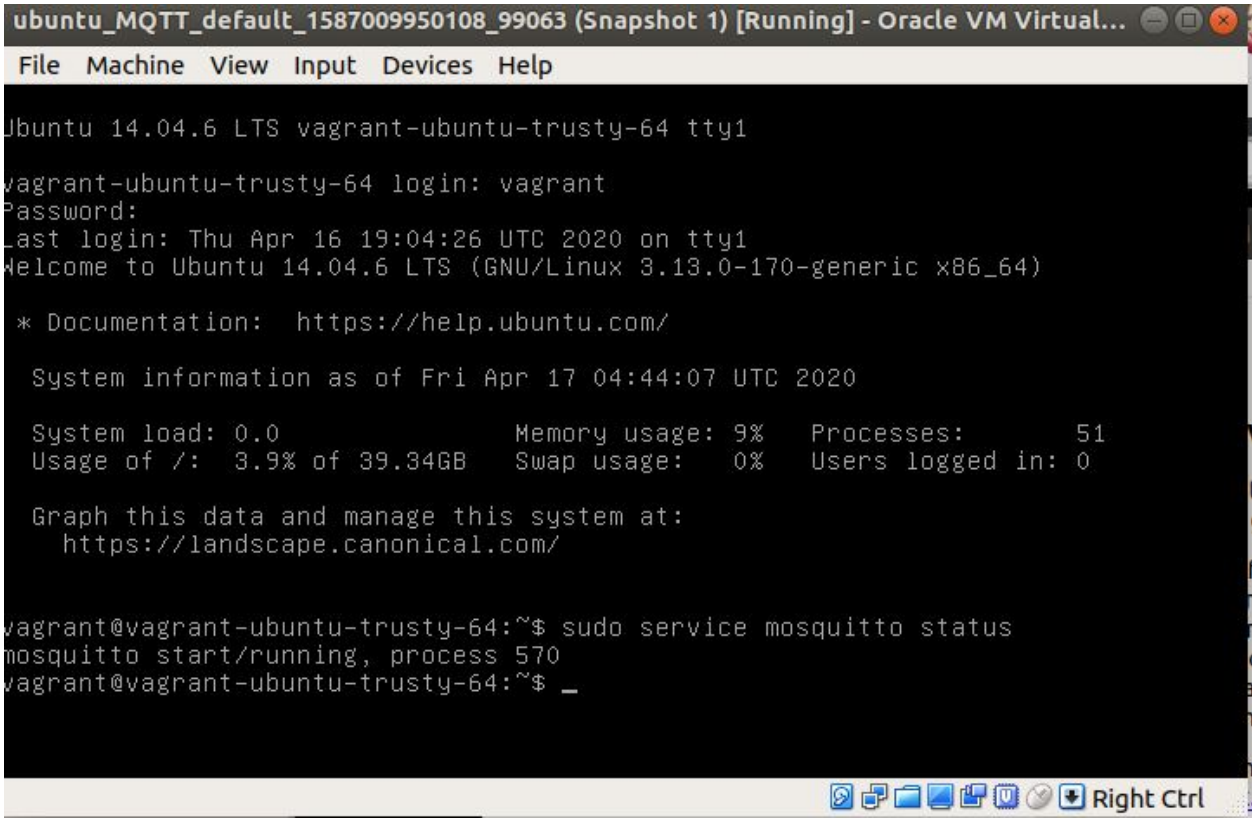

Figure 1.4: Screenshot of Ubuntu

Implementation of Data warehouse (SQLite + Flask)

We used a vagrantfile to create a ubuntu 18.04 VM hosted by VirtualBox. After it, we installed SQLite3 and Flask on Ubuntu VM. Here are commands:
First, update all packages on the VM. The **apt** package repository cache should be updated.

```
$ sudo apt-get update
```

Now to install SQLite 3, run the following command:

```
$ sudo apt-get install sqlite3
```

Now you can check whether **SQLite 3** is working with the following command:

```
$ sqlite3 --version
```

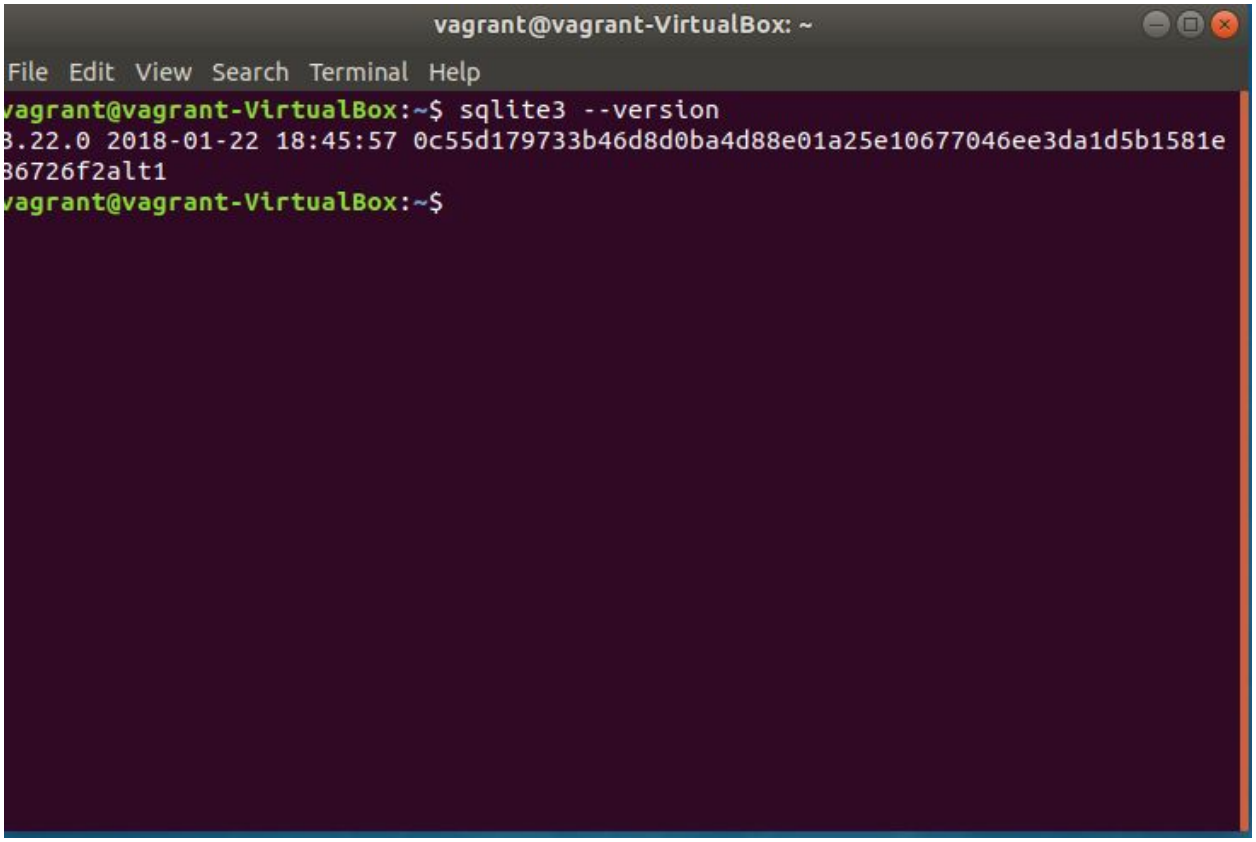

Figure 1.5: Screenshot of vagrant-VirtualBox

Installing SQLite Browser

Run the following command to install SQLite Browser:

```
$ sudo apt-get install sqlitebrowser
```

After installed SQLite broswer, we can create a database and table on GUI or by SQL command:

```
CREATE TABLE `traffic` (
      `time`     INTEGER,
      `shoppers` INTEGER,
      `lookers`  INTEGER
);
```

Installing Flask on Ubuntu 18.04 VM

First verify Python3 is installed on VM by:

```
python3 --version
```

The output should look like this:

```
Python 3.6.9
```

Second, install python3-venv package that provides the venv module:

```
sudo apt install python3-venv
```

Once the module is installed we created a virtual environment for Flask applications.

```
mkdir my_flask_app
```

```
cd my_flask_app
```

After in my_flask_app directory, creating a virtual environment:

```
python3 -m venv venv
```

And activate the virtual environment by:

```
source venv/bin/activate
```

Installing Flask:

After the virtual environment is activated, we installed Flask by:

```
pip install Flask
```

Verify the installation by command:

```
python -m flask --version
```

The result is:

```
Flask 1.1.2
Python 3.6.9 (default, Sep 12 2018, 18:26:19)
```

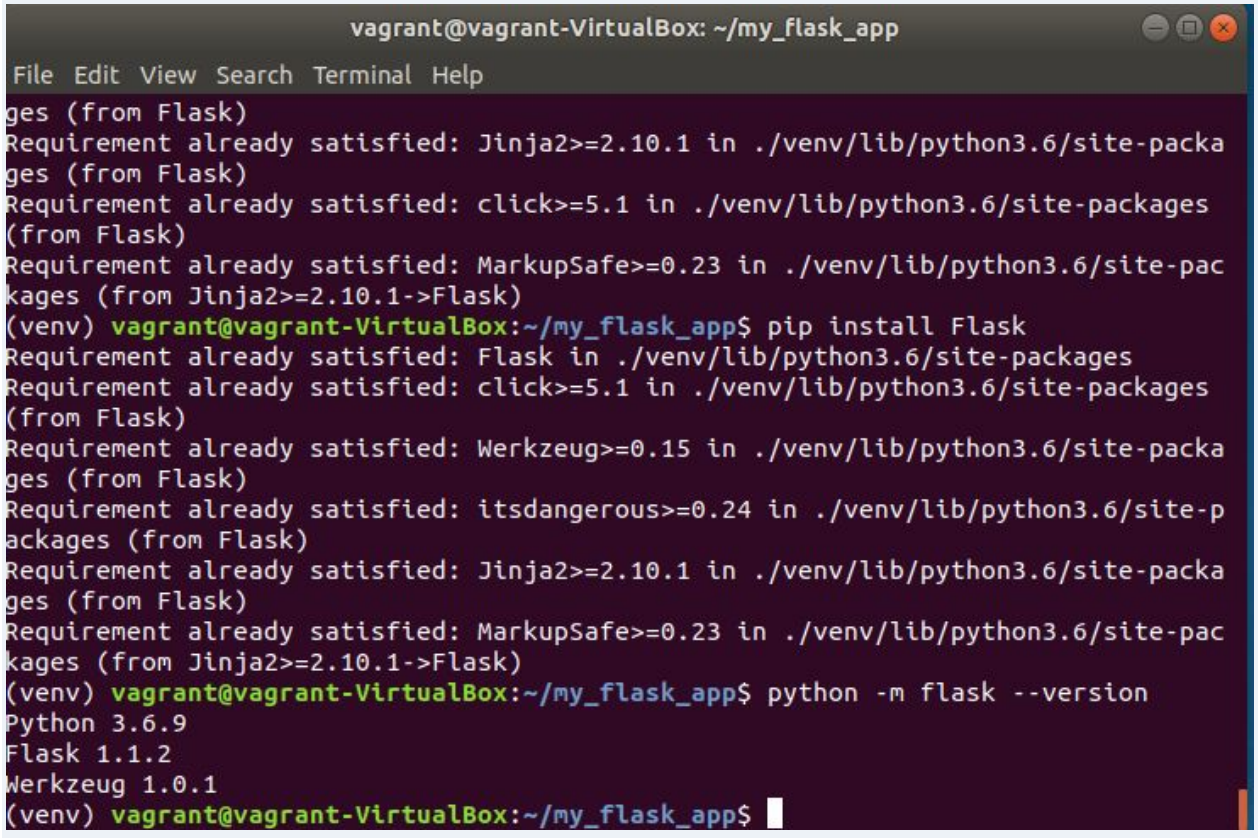

Figure 1.6: Screenshot of vagrant-VirtualBox after installing my_flask_app

GNS3 Topology

Please see the figures below:

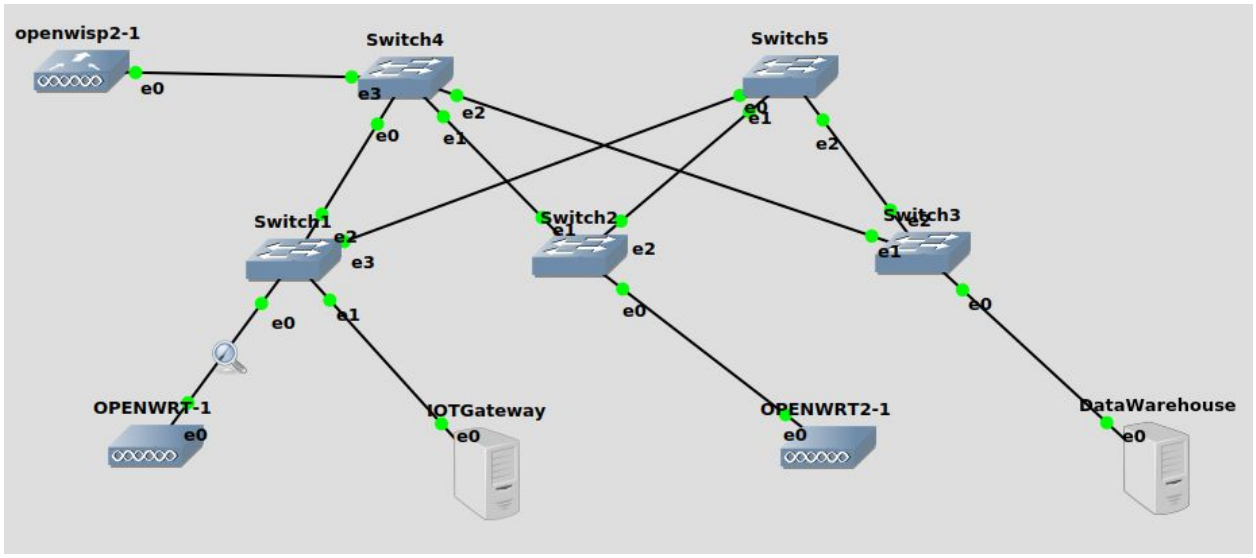

Figure 1.7

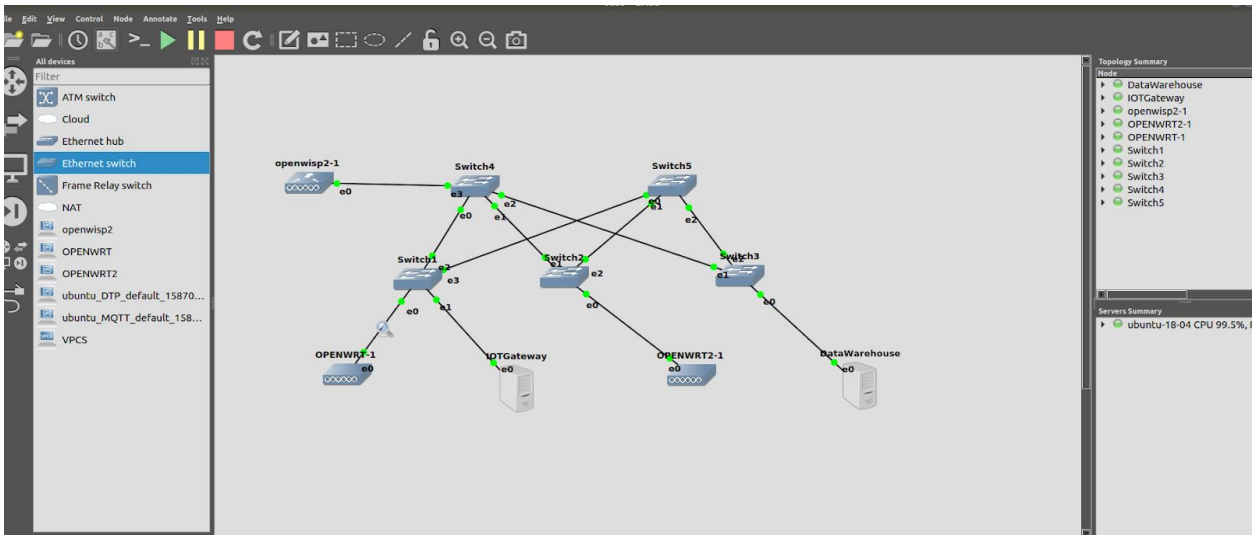

Figure 1.8

Task 2: Deploy the Application

Implementation of Shopper Gaze application

OpenVINO installation

Register and obtain serial number
SSH into Ubuntu VM

```
$ wget
http://registrationcenter-download.intel.com/akdlm/irc_nas/16612/l_openvino_toolkit_p_2020.2.120.tgz
$ tar -xvzf l_openvino_toolkit_p_2020.2.120.tgz
$ sudo ./l_openvino_toolkit_p_2020.2.120/install.sh
Follow the command prompt and install OpenVINO tool kit
```

Install Gaze Monitor

```
$ git clone https://github.com/intel-iot-devkit/shopper-gaze-monitor-python
$ cd shopper-gaze-monitor-python
$ chmod +x setup.sh
$ setup.sh
```

Setup USB passthrough from host device to VM

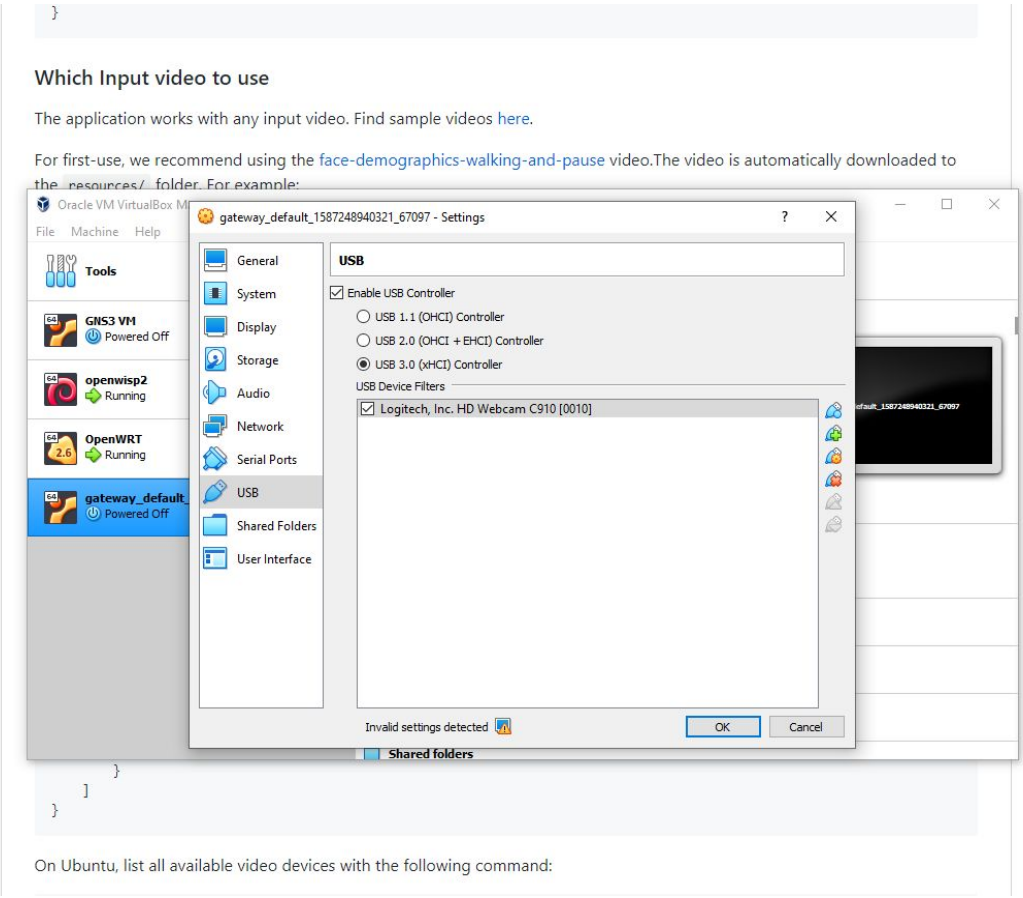

Figure 2.1

Verify Video Input, Verify Python Version, Initialize OpenVino Env with Correct Python

```
vagrant@vagrant:~$ ls /dev/video*
/dev/video0
vagrant@vagrant:~$ python3 --version
Python 3.6.9

Running the OpenVINO Shopper Gaze App requires GUI environment, the default VM box
“Hashicorp/Bionic” does not have desktop libs installed.
vagrant@vagrant:~@ sudo apt-get install -y ubuntu-desktop
```

Modify the configuration `~/shopper-gaze-monitor-python/resources/config.json` to

```
{
    "inputs": [
            {
            "video": "0"
        }
    ]
}
```

Create run.sh in `~/shopper-gaze-monitor-python/application`

```
source /opt/intel/openvino/bin/setupvars.sh -pyvaer 3.6
```

```
python3 shopper_gaze_monitor.py -m
/opt/intel/openvino/deployment_tools/open_model_zoo/tools/downloader/intel/face-detect
ion-adas-0001/FP32/face-detection-adas-0001.xml -pm
/opt/intel/openvino/deployment_tools/open_model_zoo/tools/downloader/intel/head-pose-e
stimation-adas-0001/FP32/head-pose-estimation-adas-0001.xml -d CPU
```

Make the Script executable.

```
$ chmod +x run.sh
```

Now to run the Application `./.run.sh`

Implementation of MQTT publisher and MQTT gateway

After shopper gaze application is successfully installed and run. We added a MQTT server to publish data.

First, we set environment variables:

```
export MQTT_SERVER=localhost:1883
export MQTT_CLIENT_ID=lee1234
```

Second, we downloaded the Paho Python MQTT Software from https://github.com/eclipse/paho.mqtt.python

And Installed it by:

```
Python setup.py
```

Now Paho Python MQTT software is installed on the server.

Starting Shopper gaze application, we modified client_sub.py in example folder to subscribe the topic that is published by shopper gaze application.

```
# mqttc.on_log = on_log

mqttc.connect("localhost", 1883, 60)

mqttc.subscribe("retail/traffic", 0)
```

```
    print(string)


# If you want to use a specific client id, use
# mqttc = mqtt.Client("client-id")
# but note that the client id must be unique on the broker. Leaving the client
# id parameter empty will generate a random id for you.
mqttc = mqtt.Client()
mqttc.on_message = on_message
mqttc.on_connect = on_connect
mqttc.on_publish = on_publish
mqttc.on_subscribe = on_subscribe
# Uncomment to enable debug messages
# mqttc.on_log = on_log
mqttc.connect("localhost", 1883, 60)
mqttc.subscribe("retail/traffic", 0)

mqttc.loop_forever()
```

Figure 2.2

The following screenshot (Figure 2.3) shows the shopper gaze application open webcam and publishing MQTT messages

```
retail /traffic 0 {“shoppers”: “1”, “lookers”: “1”}
```

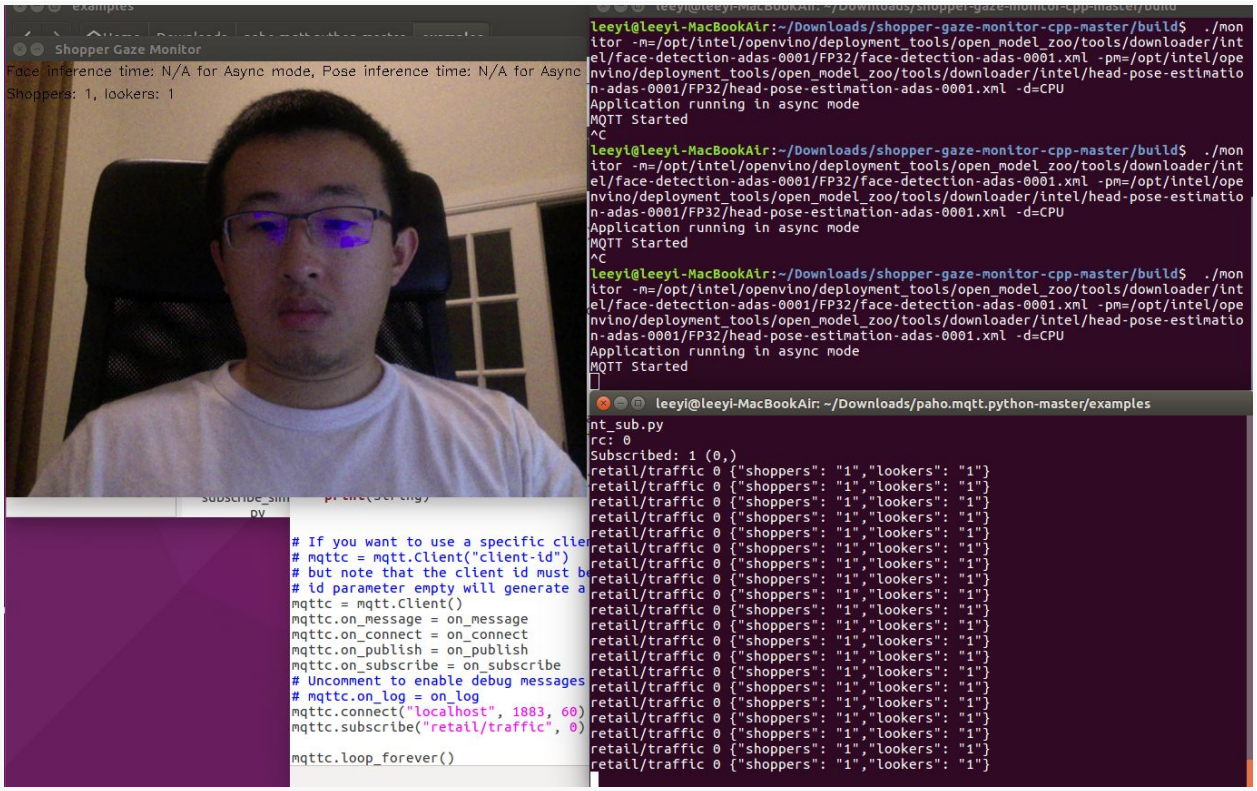

Figure 2.3

Problems that encountered

1. Installing the OpenVINO toolkit was not discovered by the Shopper Gaze App
   Solution: run installation with `sudo`
2. Default run command of the Shopper Gaze App used obsolete CPU extension library
   Solution: remove the argument from run command
3. GNS3 Application couldn't manage VirtualBox VM network interfaces
   Solution: Go to GNS3 Preferences->VirtualBox VM, select individual VMs and check `Allow GNS3 uses configured adapter`
4. OpenWISP Vagrant depends on Ansible for provisioning, yet Ansible is not very compatible with Windows.
   Solution: Some team members dual booted their PC with Ubuntu, some used a Mac laptop to provision and export.
5. Webcam laptop: For task 1, Li Yi is working on a desktop PC which does not have a connection to a webcam. For task 2, a Macbook Air is used to install Ubuntu 16.04, install Shopper Gaze app, install OpenVINO and install Mosquitto server etc.
6. Cloning Existing OpenWRT VM to be used with different IP address causes problem
   Situation: After provisioning 1 OpenWRT VM, cloning it to be used with a different IP address, it will not be recognized as a new device for OpenWISP. This is due to how automatic checksum and UUIDs are calculated for OpenWISP.
   Solution: Provision a new VM from ground up.

Contribution

Due to different locations of our team members, we worked on the project individually but shared findings and progress.
Alexey Burlakov - Alexey has implemented task 1 and task 2. On task 1, Cisco switches have been used instead of GNS switches.
Li Yi - Li has completed on task1 and task 2 using GNS3 switches, OpenWRT as APs, OpenWisp2 as wireless controllers, Mosquitto MQTT broker and Ubuntu 18.04/16.04 OS.
Pui Ng - Pui has implemented task1 and task2 with the guidance provided by lab 3 document. Vmware Workstation is used to host VMs. She also managed team communications including technical writing, structuring reports, and Slack communications.
Zhijie Wang - Zhijie has done on task1 and task 2. He worked on Mac environments, and installed VirtualBox to host GNS3 VM and other infrastructure VMs. He wrote the first half of the Task 2 steps.