Customer Tracking Through an Affordable Consumer Device Array

June 8, 2015

A Senior Project
presented to
the Faculty of the Department of Computer Science
California Polytechnic State University, San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Science

By
Daniel Nishi
June, 2015

Abstract

Commercial person tracking systems that use wifi packet analysis is currently very costly. In order to reduce the barrier of entry and allow small businesses to reap the benefits of indoor trilateration systems and monitor consumer traffic patterns, I am building out a system that will run on low-cost hardware and be deployable without a monthly service fee. By using consumer hardware, we are able to collect and analyze wifi management packets from smartphone sources and use it to extract actionable business information.

Contents

1 Introduction

Business owners are always looking for ways to improve the efficiency of their business. Today's consumers carry around a device every day that allows us to take analytics about their behavior – their smartphone. Solutions for tracking users through their smartphones and using location-based analytics exist for larger businesses through contractors, however there does not exist an inexpensive solution for smaller businesses. For example, the Navizon Indoor Triangulation System's cheapest package costs $350 a month to operate [2].

This project uses inexpensive hardware to imitate the features offered by commercial location tracking suites. Using a combination of Raspberry Pis and affordable USB adapters capable of being set into monitor mode, wifi packets sent out by smartphones are aggregated and analyzed to extract location information. Smartphones utilize these packets to determine if known wifi networks are accessible. [5] By mining these packets, we can determine a unique identifier for the smartphone, its distance from the collection node, and the smartphones targeted wifi network. The analysis software uses collection logs from several nodes in a trilateration analysis to determine the devices location. Over time, this process can find trends in person traffic around the collection nodes and group people by visit times and demographics based on their previously connected wifi networks.

2 Problem Statement

Location-based analytics using smartphone tracking has a high cost of entry. This does not allow small businesses or interested individuals to try out the technology and reap the benefits of using it to improve their business. By creating a low-cost alternative to existing commercial solutions, we can reduce the barrier to using location-based analytics and allow

enterprising small businesses to analyze visitor behavior.

3 Software

The project is broken down into two different programs. The first is a data collection program which runs locally on the collection node (i.e. Raspberry Pi or other computer). The second is an analysis web app, which can be hosted either on a local network or publicly online. Logs form the data collection program are uploaded into the analysis application, which provides a web UI for analyzing the data.

3.1 Collection Software

The collection software is implemented in Python. It uses a two process architecture: the first process listens to the packets being received by the USB wifi adapter and logs them using the SQLAlchemy ORM into a local SQLite database log; the second process forces the USB wifi adapter to change which wifi channel it is on, in order to collect wifi information being broadcast on all channels. I will explain the most important components of the implementation in this section.

Before the execution of the program, we need to initialize our connection to the database.

```
# database.py
import os, sys, datetime
from sqlalchemy import Column, Integer, String, DateTime, ForeignKey
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import create_engine

Base = declarative_base()
```

```
class Packet(Base):
    __tablename__ = 'packet'
    id = Column(Integer, primary_key=True)
    mac = Column(String(17), nullable=False)
    ssid = Column(String(32))
    time = Column(DateTime, default=datetime.datetime.now())
    signal = Column(Integer)

def createNewDatabase(filename):
    engine = create_engine('sqlite:///%s.db' % filename, convert_unicode=True)
    Base.metadata.create_all(engine)

# wifi_monitor.py
engine = create_engine('sqlite:///log.db')
engine.raw_connection().connection.text_factory = str
Base.metadata.bind = engine

DBSession = sessionmaker(bind=engine)
session = DBSession()
```

We declare our database schema declaratively with an object model using the SQLAlchemy declarative library. This automatically sets up our schema and allows us to directly commit objects to our SQLite database. Using this schema (Base), we set up a SQLite connection to the "log.db" file. This database engine will be used to persist the packet data as we collect it. Take special note of changing the text factory of the engine's SQLite connection to use the Python str class – this is because the default SQLAlchemy connection will crash

upon hitting an unexpected unicode character without using the Python string as its string class.

```
# wifi_monitor.py
if __name__ == '__main__':
    print "Starting interface into monitor mode..."
     iface = sys.argv[0]
    if not set_iface_to_monitor_mode(iface):
        print "Error starting %s into monitor mode" % iface
        sys.exit(1)

    print "Beginning packet capture..."
    p = Process(target = hop_channels)
    p.start()
    signal.signal(signal.SIGINT, signal_handler)
    sniff(iface=iface, prn = CollectPackets, store=0)
```

When the program is started, it takes a network device specified in the parameters and attempts to place it into monitor mode. Monitor mode allows the device to listen to every incoming packet, regardless of its intended destination. Assuming this process succeeds, the program starts up the second process, which is discussed later. The program then enters a sniffing mode and awaits the USB wifi adapter to detect packets to record. Once the interface is in monitor mode and the channel hopper process is started, we use the sniff command out of the Scapy library with the "CollectPackets" function to record our data. "store=0" is defined to not store the packets in memory after they are recorded (i.e. by persisting 0 packets in memory) – if this is not specified, the machine will run out of memory eventually.

```
def set_iface_to_monitor_mode(interface):
    try:
        err = os.system('ifconfig %s down' % interface)
        err = err + os.system('iwconfig %s mode monitor' % interface)
        err = err + os.system('ifconfig %s up' % interface)
        return (err == 0)
    except Exception as e:
        print e
        return False
    return True
```

In order to set the interface into monitor mode, we use several unix commands directly. The interface's mode cannot be changed if it is currently active, so we use ifconfig to disable the interface temporarily. We then use iwconfig to change the mode of the interface into monitor. After this, we use ifconfig to re-activate the interface. If any exceptions occur, this process returns false the program execution halts. Each of the os.system calls should return 0 on success, so by adding the return codes together we can say that all of them were successful if the error is 0.

```
PROBE_REQUEST_TYPE = 0
PROBE_REQUEST_SUBTYPE = 4


def CollectPackets(pkt):
    if pkt.type == PROBE_REQUEST_TYPE and pkt.subtype == PROBE_REQUEST_SUBTYPE:
        mac_address = pkt.addr2
        signal = -(256 - ord(pkt.notdecoded[-4:-3]))
        ssid = pkt.getlayer(Dot11ProbeReq).info
```

```
new_packet = Packet(mac=mac_address,
    ssid=ssid,
    signal=signal,
    time=datetime.datetime.now())
session.add(new_packet)
session.commit()
```

In order to listen to not overload our device with packets that lack the information we need, we restrict the collection to only probe request packets by pre-checking the packet type as it arrives. The probe request packet contains the transmitting device's MAC address (addr2), the signal strength between the device and the collection node, and, potentially, the SSID that the device is trying to connect to [3]. The Scapy API's packet object model pulls out the MAC address and the data structure which contains the SSID, allowing for easy recording.

The signal strength is hidden within the undecoded information of the packet, and we need to index directly into the raw byte structure to pull out the data. One of the fields present within the probe request packet is the "dBm_AntSignal" field, which contains the received signal strength indicator, which we use to determine the signal strength received in dBm. This field is a one-byte field, meaning it ranges up to 256, so we find the real value by subtracting the field's value from 256. We will later use this resultant number as our signal strength using a Free-space Path Loss formula. This value is stored as a positive value, but it actually a negative one.

This data is used to construct a new ORM Packet object which contains the MAC address, signal strength, the time it was collected, and the SSID that the device was trying to connect to, if it exists. The object is committed to the database to save it and the collection continues.

For best operation, all collection nodes need to be synced to the same time. Please ensure that all collection nodes have their clocks synchronized to an online clock before beginning collection, because clock synchronization needs to be within a few seconds for accurate operation.

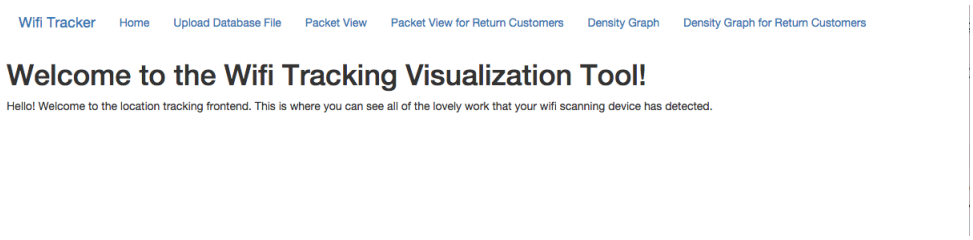
3.1.1 The Channel Hopper

```
'''
Channel hopper.
'''
MAX_CHANNEL = 14
CHANNEL_DELAY = 5 #seconds
def hop_channels():
    channel = 0
    while True:
        try:
            channel = (channel % MAX_CHANNEL + 1)
            os.system('iw dev %s set channel %d' % (iface, channel))
            print 'setting channel to %d' % (channel)
            time.sleep(CHANNEL_DELAY)
        except Exception as e:
            print "Channel hopping ceased."
            print e
            break
```

802.11 wifi has several different channels that it uses to communicate [7]. In order to capture packets from as many devices as possible, the collector node needs to listen to

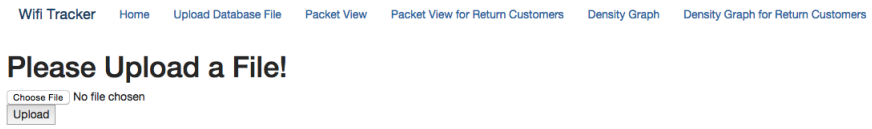
every single channel. The collector node's wifi adapter can only listen to a single channel at any given time. In order to combat this, the collector has a second process which hops channels every few seconds. This allows the collector to listen to packets on every channel, even though it will miss some on the other channels. Most probe requests are sequentially sent out on every channel, however, so we will detect the majority of packets [4].

3.2 Analysis Server

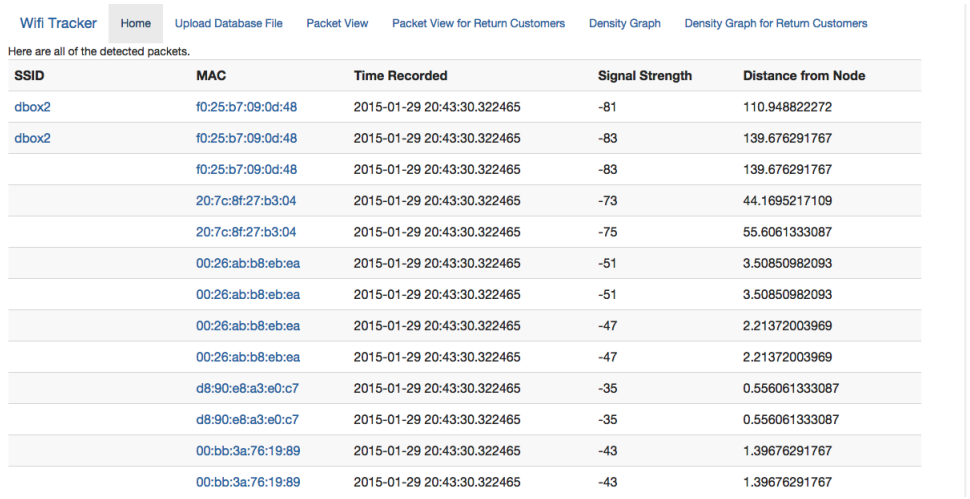


The analysis server is a Python Flask web server which uses SQLAlchemy to unpack the data collection logs generated by the collection nodes. It allows the user to view the data in a web application built using Bootstrap, the Jinja2 templating system, and Google Charts.

3.2.1 Packet Decoding and Analysis



When the user starts up the analysis server and browses to it in their web browser, they must upload a collection log database from one of the collection nodes. This is done through a web page that simply has a file input HTML field. When the file is submitted to the server, Flask catches the POST request and moves the file into a uploaded file folder.



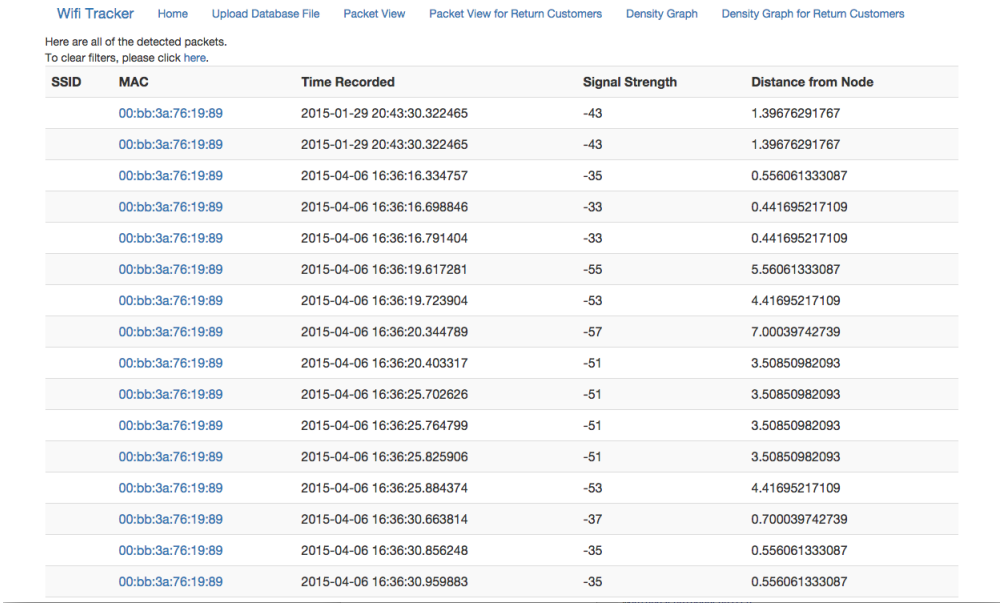
Wifi Tracker | Home | Upload Database File | Packet View | Packet View for Return Customers | Density Graph | Density Graph for Return Customers

Here are all of the detected packets.

SSID	MAC	Time Recorded	Signal Strength	Distance from Node
dbox2	f0:25:b7:09:0d:48	2015-01-29 20:43:30.322465	-81	110.948822272
dbox2	f0:25:b7:09:0d:48	2015-01-29 20:43:30.322465	-83	139.676291767
	f0:25:b7:09:0d:48	2015-01-29 20:43:30.322465	-83	139.676291767
	20:7c:8f:27:b3:04	2015-01-29 20:43:30.322465	-73	44.1695217109
	20:7c:8f:27:b3:04	2015-01-29 20:43:30.322465	-75	55.6061333087
	00:26:ab:b8:eb:ea	2015-01-29 20:43:30.322465	-51	3.50850982093
	00:26:ab:b8:eb:ea	2015-01-29 20:43:30.322465	-51	3.50850982093
	00:26:ab:b8:eb:ea	2015-01-29 20:43:30.322465	-47	2.21372003969
	00:26:ab:b8:eb:ea	2015-01-29 20:43:30.322465	-47	2.21372003969
	d8:90:e8:a3:e0:c7	2015-01-29 20:43:30.322465	-35	0.556061333087
	d8:90:e8:a3:e0:c7	2015-01-29 20:43:30.322465	-35	0.556061333087
	00:bb:3a:76:19:89	2015-01-29 20:43:30.322465	-43	1.39676291767
	00:bb:3a:76:19:89	2015-01-29 20:43:30.322465	-43	1.39676291767

If multiple collection nodes are used, the user must specify a location relative to the first node when uploading their file database. This location is used to combine the collection logs to determine the location of the devices detected by the collection nodes.

3.2.2 Packet View



Wifi Tracker | Home | Upload Database File | Packet View | Packet View for Return Customers | Density Graph | Density Graph for Return Customers

Here are all of the detected packets.
To clear filters, please click here.

SSID	MAC	Time Recorded	Signal Strength	Distance from Node
	00:bb:3a:76:19:89	2015-01-29 20:43:30.322465	-43	1.39676291767
	00:bb:3a:76:19:89	2015-01-29 20:43:30.322465	-43	1.39676291767
	00:bb:3a:76:19:89	2015-04-06 16:36:16.334757	-35	0.556061333087
	00:bb:3a:76:19:89	2015-04-06 16:36:16.698846	-33	0.441695217109
	00:bb:3a:76:19:89	2015-04-06 16:36:16.791404	-33	0.441695217109
	00:bb:3a:76:19:89	2015-04-06 16:36:19.617281	-55	5.56061333087
	00:bb:3a:76:19:89	2015-04-06 16:36:19.723904	-53	4.41695217109
	00:bb:3a:76:19:89	2015-04-06 16:36:20.344789	-57	7.00039742739
	00:bb:3a:76:19:89	2015-04-06 16:36:20.403317	-51	3.50850982093
	00:bb:3a:76:19:89	2015-04-06 16:36:25.702626	-51	3.50850982093
	00:bb:3a:76:19:89	2015-04-06 16:36:25.764799	-51	3.50850982093
	00:bb:3a:76:19:89	2015-04-06 16:36:25.825906	-51	3.50850982093
	00:bb:3a:76:19:89	2015-04-06 16:36:25.884374	-53	4.41695217109
	00:bb:3a:76:19:89	2015-04-06 16:36:30.663814	-37	0.700039742739
	00:bb:3a:76:19:89	2015-04-06 16:36:30.856248	-35	0.556061333087
	00:bb:3a:76:19:89	2015-04-06 16:36:30.959883	-35	0.556061333087

The most simple view of the data is the packet view. This page displays every single

packet in the collection logs in their raw form. In order to pull the packets out of the collection log, the server applies a SQLAlchemy declarative database schema onto the data to convert them into objects. The server then applys a minimal amount of processing to the packets to convert the RSSI signal strength into a distance from the collection node.

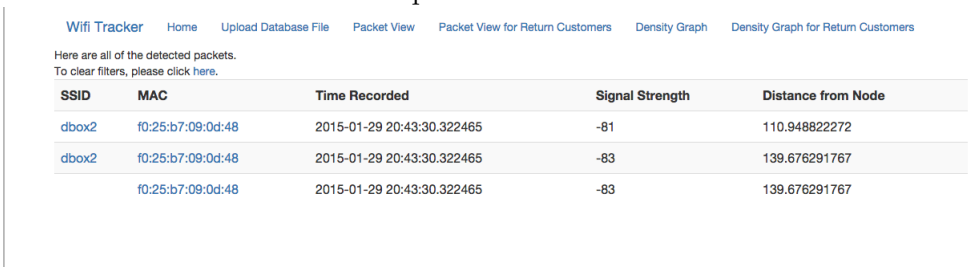
```
# Returns the distance in meters.
def get_distance(dbm):
    exponent = (27.55 - (20 * math.log10(2412)) + math.fabs(dbm)) / 20.0
    return 10 ** exponent
```

The server applies the above distance formula to every single packet's RSSI dbm. The algorithm is derived from the Free Space Path Loss (FSPL) algorithm, which determines the signal loss over free space for a given frequency and distance [6]. For distance in meters and frequency in megahertz using the 802.11 specification of 2412mhz for wifi, we get the exponent formula listed above [7]. Thus, this formula allows us to calculate the distance of the packet's device from the collection node.

```
def _filterPackets(ssid, mac):
    query = session.query(database.Packet)
    if ssid is not None:
        query = query.filter(database.Packet.ssid == ssid)
    if mac is not None:
        query = query.filter(database.Packet.mac == mac)
    return query
```

The user can filter the packets by detected MAC address and the SSID that the device attempted to connect into. This is done by performing a query against the SQLite database collection log, as soon above. By using SQLAlchemy's built-in query system, the analysis server constructs a SQL query and displays the packet view page with only the packets

that adhere to the filter. On the UI side, this is done by clicking on the MAC address or SSID that the user wants to filter upon.

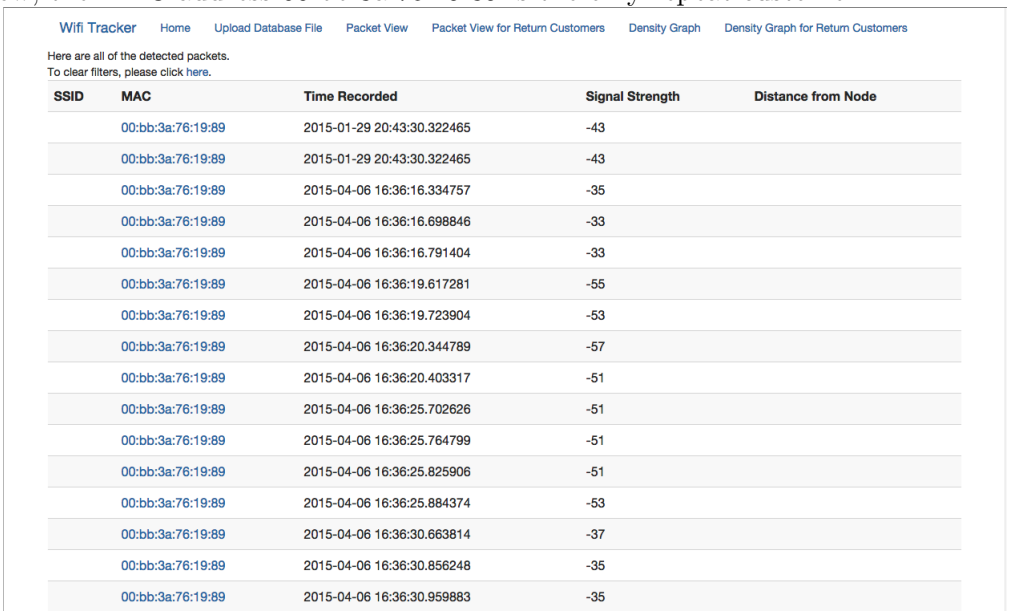


Wifi Tracker Home Upload Database File Packet View Packet View for Return Customers Density Graph Density Graph for Return Customers

Here are all of the detected packets.
To clear filters, please click here.

SSID	MAC	Time Recorded	Signal Strength	Distance from Node
dbox2	f0:25:b7:09:0d:48	2015-01-29 20:43:30.322465	-81	110.948822272
dbox2	f0:25:b7:09:0d:48	2015-01-29 20:43:30.322465	-83	139.676291767
	f0:25:b7:09:0d:48	2015-01-29 20:43:30.322465	-83	139.676291767

The server provides a secondary filter for only viewing repeat customers. In the figure below, the MAC address 00:bb:3a:76:19:89 is the only repeat customer.

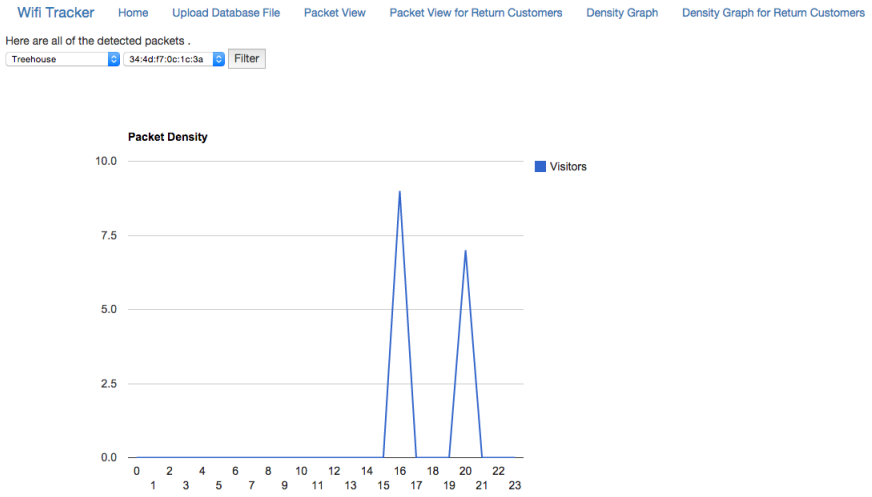


Wifi Tracker Home Upload Database File Packet View Packet View for Return Customers Density Graph Density Graph for Return Customers

Here are all of the detected packets.
To clear filters, please click here.

SSID	MAC	Time Recorded	Signal Strength	Distance from Node
	00:bb:3a:76:19:89	2015-01-29 20:43:30.322465	-43	
	00:bb:3a:76:19:89	2015-01-29 20:43:30.322465	-43	
	00:bb:3a:76:19:89	2015-04-06 16:36:16.334757	-35	
	00:bb:3a:76:19:89	2015-04-06 16:36:16.698846	-33	
	00:bb:3a:76:19:89	2015-04-06 16:36:16.791404	-33	
	00:bb:3a:76:19:89	2015-04-06 16:36:19.617281	-55	
	00:bb:3a:76:19:89	2015-04-06 16:36:19.723904	-53	
	00:bb:3a:76:19:89	2015-04-06 16:36:20.344789	-57	
	00:bb:3a:76:19:89	2015-04-06 16:36:20.403317	-51	
	00:bb:3a:76:19:89	2015-04-06 16:36:25.702626	-51	
	00:bb:3a:76:19:89	2015-04-06 16:36:25.764799	-51	
	00:bb:3a:76:19:89	2015-04-06 16:36:25.825906	-51	
	00:bb:3a:76:19:89	2015-04-06 16:36:25.884374	-53	
	00:bb:3a:76:19:89	2015-04-06 16:36:30.663814	-37	
	00:bb:3a:76:19:89	2015-04-06 16:36:30.856248	-35	
	00:bb:3a:76:19:89	2015-04-06 16:36:30.959883	-35	

3.2.3 Density Graph

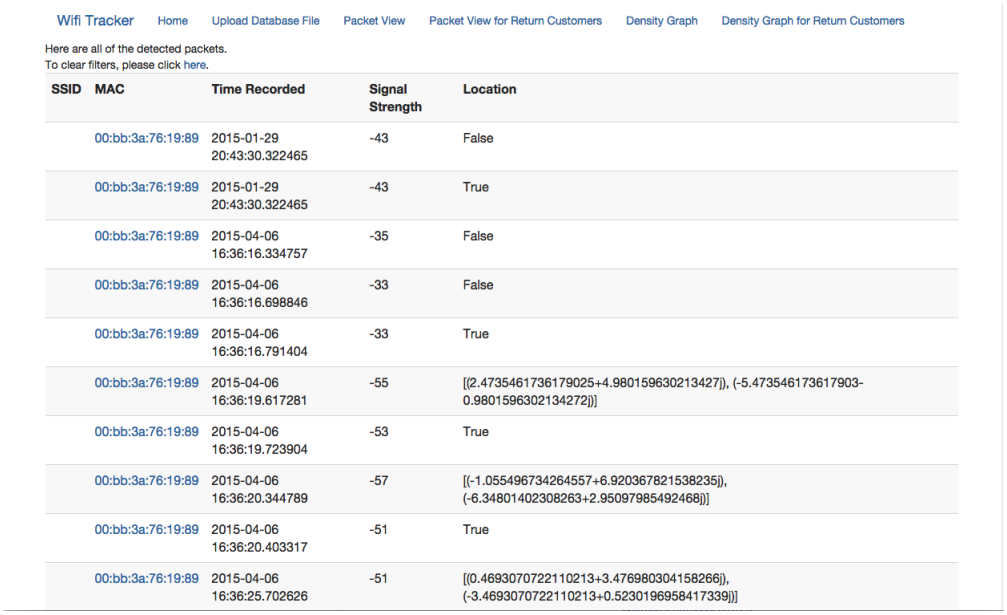


The Density Graph page displays the information from the packet view broken down by time period. This allows us to see which hours of operation had the most traffic around the collection node. As with the Packet View, the Density Graph page allows for filters to be set on the data in order to focus on individual devices or SSID groups. The analysis server passes in the packets into the Jinja2 templating engine for rendering. The Jinja2 template constructs a JS object of packets for visualization.

3.2.4 Single Node Operation

In single node operation, a single collection log is uploaded into the analysis server. Because each packet represents a single data point, the analysis server cannot perform any trilateration on the data. It can, however, calculate the distance from the node and use this information to determine if the device has been seen close to the collection node and at what times. This operation is pictured above with Packet-By-Packet View figure.

3.2.5 Double Node Operation



Wifi Tracker Home Upload Database File Packet View Packet View for Return Customers Density Graph Density Graph for Return Customers

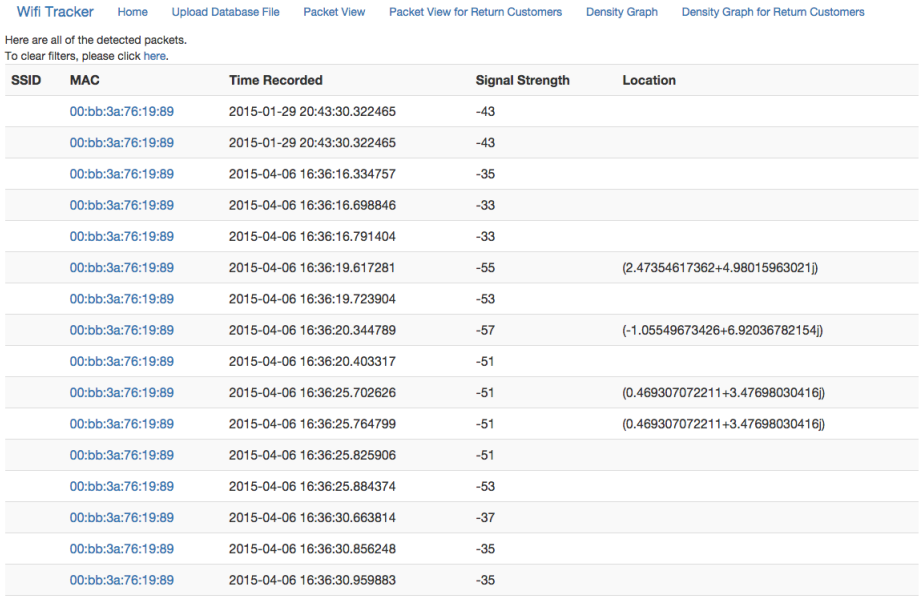
Here are all of the detected packets.
To clear filters, please click here.

SSID	MAC	Time Recorded	Signal Strength	Location
	00:bb:3a:76:19:89	2015-01-29 20:43:30.322465	-43	False
	00:bb:3a:76:19:89	2015-01-29 20:43:30.322465	-43	True
	00:bb:3a:76:19:89	2015-04-06 16:36:16.334757	-35	False
	00:bb:3a:76:19:89	2015-04-06 16:36:16.698846	-33	False
	00:bb:3a:76:19:89	2015-04-06 16:36:16.791404	-33	True
	00:bb:3a:76:19:89	2015-04-06 16:36:19.617281	-55	[(2.4735461736179025+4.980159630213427j), (-5.473546173617903-0.9801596302134272j)]
	00:bb:3a:76:19:89	2015-04-06 16:36:19.723904	-53	True
	00:bb:3a:76:19:89	2015-04-06 16:36:20.344789	-57	[(-1.055496734264557+6.920367821538235j), (-6.34801402308263+2.95097985492468j)]
	00:bb:3a:76:19:89	2015-04-06 16:36:20.403317	-51	True
	00:bb:3a:76:19:89	2015-04-06 16:36:25.702626	-51	[(0.4693070722110213+3.476980304158266j), (-3.4693070722110213+0.5230196958417339j)]

In double node operation, two collections logs from two different locations are uploaded into the analysis server. Distance from the node is calculated for every packet in both databases. These databases are then reconciled into a single one for display.

In order to merge the two databases, the analysis server iterates over the first database's packets. For each packet in the first database, the server iterates over the second database finding the chronologically closest packet with the same MAC address. This iteration is done in chronological order, calculating the time delta between the two packets and then selecting the packet with the smallest difference and the same MAC address. Using these two points and the location of the second collection node relative to the first node, the server draws two circles around each node with the distance of the points. The two intersection points are returned as the two possible locations of the device. Without a third node, the server cannot determine which of these locations is valid.

3.2.6 Triple Node Operation



Wifi Tracker Home Upload Database File Packet View Packet View for Return Customers Density Graph Density Graph for Return Customers

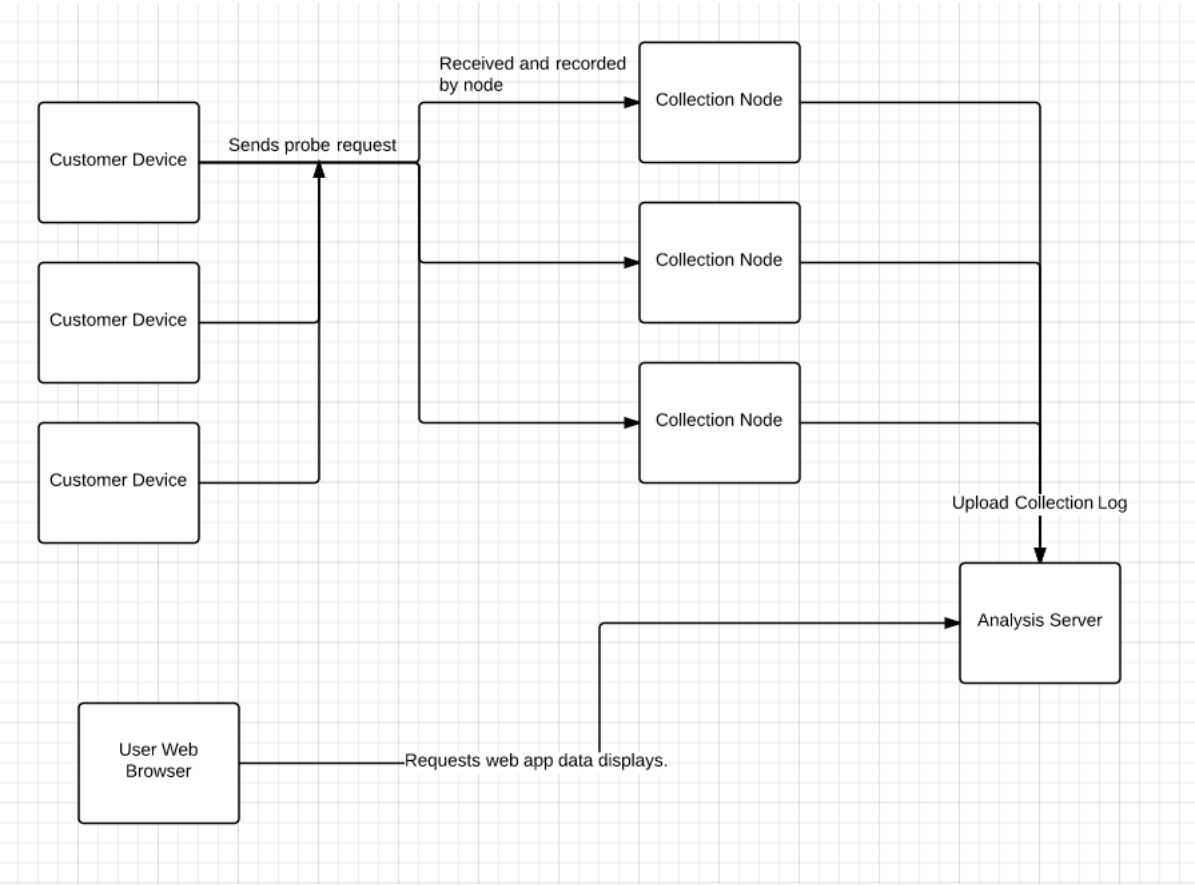
Here are all of the detected packets.
To clear filters, please click here.

SSID	MAC	Time Recorded	Signal Strength	Location
	00:bb:3a:76:19:89	2015-01-29 20:43:30.322465	-43	
	00:bb:3a:76:19:89	2015-01-29 20:43:30.322465	-43	
	00:bb:3a:76:19:89	2015-04-06 16:36:16.334757	-35	
	00:bb:3a:76:19:89	2015-04-06 16:36:16.698846	-33	
	00:bb:3a:76:19:89	2015-04-06 16:36:16.791404	-33	
	00:bb:3a:76:19:89	2015-04-06 16:36:19.617281	-55	(2.47354617362+4.98015963021j)
	00:bb:3a:76:19:89	2015-04-06 16:36:19.723904	-53	
	00:bb:3a:76:19:89	2015-04-06 16:36:20.344789	-57	(-1.05549673426+6.92036782154j)
	00:bb:3a:76:19:89	2015-04-06 16:36:20.403317	-51	
	00:bb:3a:76:19:89	2015-04-06 16:36:25.702626	-51	(0.469307072211+3.47698030416j)
	00:bb:3a:76:19:89	2015-04-06 16:36:25.764799	-51	(0.469307072211+3.47698030416j)
	00:bb:3a:76:19:89	2015-04-06 16:36:25.825906	-51	
	00:bb:3a:76:19:89	2015-04-06 16:36:25.884374	-53	
	00:bb:3a:76:19:89	2015-04-06 16:36:30.663814	-37	
	00:bb:3a:76:19:89	2015-04-06 16:36:30.856248	-35	
	00:bb:3a:76:19:89	2015-04-06 16:36:30.959883	-35	

In triple node operation, three collection logs from three different locations are uploaded into the analysis server. Distance from the node is is calculated in all three logs. These logs are then reconciled into a single one for display.

The first two logs are merged using the method specified in double node operation. This operation results in two possible points for the device. In a similar process to the double node operation, the third log is analyzed for the packet that is chronologically closest with the same MAC address to the one being analyzed. If it exists, the server compares the distance specified by the third log's packet to the proposed locations calculated from the first two nodes. The one closest to the third log's packet's distance is chosen as the best estimate of the device's location.

4 Diagram of Network Communication



For the system to work fully, several collection nodes must be active at the same time. These collection nodes do not need to be on any network, because they are storing the collection data locally. After their collection period, the collection log file must be uploaded to the analysis server. This can be done either by collecting the collection files from the nodes and uploading them from a central machine or done by uploading them directly from the collection nodes. Because the operation of the collection nodes works best from a GUI-less Linux deployment, it is likely better to upload from a central machine.

In order to upload to the analysis server, it must be online. This can either be done by hosting it on the local network by executing the server.py file. This can also be done in a

deployment to a cloud instance, such as an AWS instance. I did not use an analysis server in the cloud for this project.

In both cases, the files are uploaded and stored on the database for analysis. To access this data, the user must browse to the analysis server through a web browser and access the web app. The analysis stage occurs whenever a page is requested from the analysis server that displays the data. This analysis is done on the server and served to the client machine over the network.

5 Budget and Bill of Materials

My budget for this project was $100. In order to properly test the trilateration, I needed three collection nodes. Since I already owned a laptop capable of running Linux, I used it as the first collection node. I purchased two Raspberry Pis – one Raspberry Pi B+ and one Raspberry Pi 2. Both are capable of performing the collection task. In order to pull off this project in production with standard devices, one would need three Raspberry Pis at $25 and three USB wifi adapters capable of going into monitor mode, which cost about $6 a piece. It is theoretically possible to build out the system and test the trilateration for under the $100 budget.

5.1 USB Wifi Adapters

Many USB wifi adapters are incapable of going into monitor mode. I discovered this when I attempted to use a wifi adapter that I already owned. The adapter used a Realtek chipset that did not support monitor mode. The most common wifi adapter would fit this project is currently the Ralink RT5370. In a stock Raspbian installation, the Ralink RT5370's x86 and ARM drivers allow it to go into monitor mode, making it usable for our purposes.

Do note that all Ralink RT5370 chipset wifi adapters are not equal. One adapter I

purchased as a "niceEshop Mini 150M" adapter, which had an external antenna. This particular adapter gave significantly different power results than the other Ralink RT5370 adapters that I used for this project. For consistency purposes, all of the wifi adapters should be purchased from the same source at the same time.

Finally, many sellers that advertise Ralink RT5370 adapters do not actually sell the RT5370. One adapters I purchased on Amazon claimed to be a Ralink RT5370 chipset and had reviews that claimed that it was an RT5370. Upon inspection, the adapter I received was actually an Realtek RTL8192, which is incapable of going into monitor mode. Ensure that you check the most recent Amazon reviews in order to determine if the seller is actually selling an RT5370 adapter.

5.2 Bill of Materials

Item	Cost	Justification
Raspberry Pi B+	$25	Inexpensive device capable of running Linux and powerful enough to perform collection.
Raspberry Pi 2	$35	Inexpensive device capable of running Linux and powerful enough to perform collection.
Ralink RT5370 x 3	($3.39 + $1.49 shipping) x 3 = $15	Least expensive wifi adapter capable of monitor mode.

6 Lessons Learned

I learned a lot about the 802.11 wifi packet specifications as well as about setting up a web server. Although most of the heavy lifting was done by the Scapy Python library, I

investigated parsing the packets directly for performance reasons. Initially, the Raspberry Pi B+ did not seem to be able to handle the load of collecting packets and storing them to the SQLite database, but this turned out to be resultant from me using the XFCE desktop management software instead of running out of only the terminal. Although I did not end up coding the direct parsing, I had to deeply analyze the spec in order to pull out the RSSI data from the undecoded segments and also in preperation for the worst performance case. Altogether, this made for a great learning experience with Python, interop with actual hardware, and trying to generate useful information.

If I were to do this project differently, I would have worked on attempting to find location through means other than trilateration with three wifi nodes. Double and triple nodes are particularly susceptible to problems with obstructions. Since RSSI, which I used to calculate the distance of the devices from the collection nodes, is the signal strength, it is weakened by an obstruction. This means that the algorithm I used to convert a signal strength to distance based upon signal strength loss through space would interpret an obstruction as the signal simply being further away. With only three nodes, even a simple obstruction would poison the data enough to make the trilateration unable to find a point.

Furthermore, wifi was a poor choice of signal to track for precise locations. I did not discover that the accuracy of the signal I was tracking was too low for precise indoor positioning until I had already implemented the system. Wifi is roughly accurate within 2m to 10m [1]. Although this is good enough for the original goal of fingerprinting customers by their devices' MAC addresses, this was not good enough for precise indoor location through trilateration.

With this in mind, I would have restructured the project to focus more on analytics from a single collection node. The information added by adding double and triple node support ended up being only accurate enough for room-to-room precision, rather than

within-room precision.

7 Conclusion

Although this project did not reach the level of existing commercial systems, I believe it provides a framework for building out a system that could compete with them. The project was a great exploration into wifi data collection and building out a data visualization system. As a proof-of-concept and a base, I believe that this could be built into an app that would generate a lot of actionable business information for small businesses.

Working with this project greatly aided my Python and JS experiences and to understand the challenges of working with information coming in from physical hardware. Obtaining and getting useful information out of the collection nodes, while working to keep performance at a level that the Raspberry Pi devices could manage, proved to be a challenging task. This project tied together many of my skills across several domains and programming languages.

8 Appendix: Code

8.1 Dependencies

In order to deploy this product, Twitter Bootstrap is required. This requires the bootstrap.css file to be placed in app/static/css directory and bootstrap.js to be placed in the app/static/js directory.

8.2 Collection Node

8.2.1 wifimonitor.py

```
'''
In order to properly run, ensure that |iface| is in monitor mode.
Furthermore, run this Python script as sudo.

Dependencies: Scapy, SQLite, SQLAlchemy.
'''

import sys, os, signal, datetime
from scapy.all import *
from multiprocessing import Process
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from database import Base, Packet


# Database setup.
```

```
# TODO: Make this mutable.
engine = create_engine('sqlite:///log.db')
engine.raw_connection().connection.text_factory = str
Base.metadata.bind = engine

DBSession = sessionmaker(bind=engine)
session = DBSession()

PROBE_REQUEST_TYPE = 0
PROBE_REQUEST_SUBTYPE = 4

# Set non-statically in the future.
iface = 'en0'
channel = 1

seen_mac_addresses = set()
seen_SSIDs = set()

def PacketHandler(pkt):
    if pkt.haslayer(Dot11):
        if pkt.type == PROBE_REQUEST_TYPE and pkt.subtype ==\
                PROBE_REQUEST_SUBTYPE and pkt.getlayer(Dot11ProbeReq).info != "'
            seen_mac_addresses.add(pkt.addr2)
            seen_SSIDs.add(pkt.getlayer(Dot11ProbeReq).info)
            print "MAC address: %s, SSID: %s, Signal Strength(-100 to 0): %d" %\
```

```
                    (pkt.addr2, pkt.getlayer(Dot11ProbeReq).info,
                     -(256-ord(pkt.notdecoded[-4:-3])))

def AllPacketHandler():
    data = {'last_mac': None}
    def packetHandler(pkt):
        if pkt.haslayer(Dot11) and pkt.addr2 != data['last_mac'] and
              pkt.addr2 not in seen_mac_addresses:
              data['last_mac'] = pkt.addr2
              seen_mac_addresses.add(pkt.addr2)
              print -(256-ord(pkt.notdecoded[-4:-3]))
              print "MAC address detected: %s" % (pkt.addr2)
    return packetHandler

def PrintPackets(pkt):
    if pkt.type == PROBE_REQUEST_TYPE and pkt.subtype == PROBE_REQUEST_SUBTYPE:
        mac_address = pkt.addr2
        signal = -(256-ord(pkt.notdecoded[-4:-3]))
        ssid = pkt.getlayer(Dot11ProbeReq).info
        new_packet = Packet(mac=mac_address,ssid=ssid,signal=signal,
                                time=datetime.datetime.now())
        session.add(new_packet)
        session.commit()
        print "MAC address: %s, SSID: %s, Signal Strength(-100 to 0): %d" %\
              (mac_address, ssid, signal)
```

```
'''
Channel hopper.
'''
def hop_channels():
    channel = 0
    while True:
        try:
            channel = (channel % 14 + 1)
            os.system('iw dev %s set channel %d' % (iface, channel))
            print 'setting channel to %d' % (channel)
            time.sleep(5)
        except Exception as e:
            print "Channel hopping ceased."
            print e
            break

def signal_handler(signal, frame):
    p.terminate()
    p.join()
    print "Terminating monitoring."
    print "Now displaying detected IP addresses..."
    for address in seen_mac_addresses:
        print address
```

```
    print "Now printing observed SSID probe requests"
    for ssid in seen_SSIDs:
        print ssid

    print "-- %d MAC addresses detected --" % len(seen_mac_addresses)
    print "-- %d SSIDs addresses detected --" % len(seen_SSIDs)
    sys.exit(0)


def set_iface_to_monitor_mode(interface):
    '''
    try:
        # TODO(dhnishi): Make this fail if an ifconfig or iwconfig fails.
        err = os.system('ifconfig %s down' % interface)
        err = err + os.system('iwconfig %s mode monitor' % interface)
        err = err + os.system('ifconfig %s up' % interface)
        return (err == 0)
    except Exception as e:
        print e
        return False
    '''
    return True

if __name__ == '__main__':
    print "Starting interface into monitor mode..."
```

```
    iface = sys.argv[0]
    if not set_iface_to_monitor_mode(iface):
        print "Error starting %s into monitor mode" % iface
        sys.exit(1)

    print "Beginning packet capture..."
    p = Process(target = hop_channels)
    p.start()
    signal.signal(signal.SIGINT, signal_handler)
    sniff(iface=iface, prn = PrintPackets, store=0)
```

8.2.2 database.py

```
import os, sys, datetime
from sqlalchemy import Column, Integer, String, DateTime, ForeignKey
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import create_engine

Base = declarative_base()
'''
class MACAddress(Base):
    __tablename__ = 'mac'
    # We need an id as address is not a good primary key, due to mutability.
    id = Column(Integer, primary_key=True)
    address = Column(String(17), nullable=False)
```

```
class SSID(Base):
    __tablename__ = 'ssid'
    id = Column(Integer, primary_key=True)
    # SSIDs have a 31 or 32 character limit in name length.
    name = Column(String(32), nullable=False)
'''
class Packet(Base):
    __tablename__ = 'packet'
    id = Column(Integer, primary_key=True)
    mac = Column(String(17), nullable=False)
    ssid = Column(String(32))
    time = Column(DateTime, default=datetime.datetime.now())
    signal = Column(Integer)


def createNewDatabase(filename):
    engine = create_engine('sqlite:///%s.db' % filename, convert_unicode=True)
    Base.metadata.create_all(engine)
```

8.3 Analysis Server

8.3.1 run.py

```
from app import app
app.run(debug=True)
```

8.3.2 app/__init__.py

```
from flask import Flask
```

```
import os

APP_ROOT = os.path.dirname(os.path.abspath(__file__))
UPLOAD_FOLDER = os.path.join(APP_ROOT, 'static/upload')
DATABASE = os.path.join(APP_ROOT, 'static/upload/packetDatabase.db')
DATABASE2 = os.path.join(APP_ROOT, 'static/upload/packetDatabase2.db')
ALLOWED_EXTENSIONS = set(['db'])

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['ALLOWED_EXTENSIONS'] = ALLOWED_EXTENSIONS
app.config['DATABASE'] = DATABASE
app.config['DATABASE2'] = DATABASE
from app import server
```

8.3.3 app/server.py

```
from flask import render_template, request
from app import app
import os
from functools import wraps
from collections import defaultdict

# Mock stuff.
import mock_database
```

```
# Database imports.
from sqlalchemy.orm import sessionmaker
from sqlalchemy import inspect, MetaData
import database

# TODO: Remove this shit.
engine = None
engine2 = None
session = None
session2 = None

location_2 = (-3, 4)
location_3 = (2, 2)

# Wrappers.
def databaseUploadRequired(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        print app.config['DATABASE']
        if not os.path.isfile(app.config['DATABASE']):
            return render_template('databaseNotUploaded.html', title="Uh-oh!")
        initializeDatabaseIfNotInitialized()
        return f(*args, **kwargs)
    return decorated_function
```

```
def database2UploadRequired(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        print app.config['DATABASE2']
        if not os.path.isfile(app.config['DATABASE2']):
            return render_template('databaseNotUploaded.html', title="Uh-oh!")
        initializeSecondDatabaseIfNotInitialized()
        return f(*args, **kwargs)
    return decorated_function

def initializeDatabaseIfNotInitialized():
    global engine
    if engine is None:
        engine = database.buildEngine(app.config['DATABASE'])
        print "engine built!"
        m = MetaData()
        m.reflect(engine)
        for table in m.tables.values():
            print table.name
            for c in table.c:
                print c.name

        database.Base.metadata.create_all(engine)
        Session = sessionmaker(bind=engine)
        global session
```

```
        session = Session()
        print "session set up"

def initializeSecondDatabaseIfNotInitialized():
    global engine2
    if engine2 is None:
        engine2 = database.buildEngine(app.config['DATABASE2'])
        print "engine2 built!"
        m = MetaData()
        m.reflect(engine2)
        for table in m.tables.values():
            print table.name
            for c in table.c:
                print c.name

        database.Base.metadata.create_all(engine2)
        Session = sessionmaker(bind=engine2)
        global session2
        session2 = Session()
        print "session set up"



@app.route('/')
@app.route('/index')
```

```
def index():
    return render_template('index.html', title='Home')

# TODO: Add in the ability to have parameters to further drill down the views.
@app.route('/packetView')
@databaseUploadRequired
def packetView():
    title = 'Packet-by-Packet View'
    template = 'packetView.html'

    maybe_ssid = request.args.get("ssid")
    maybe_mac = request.args.get("mac")

    packets = _filterPackets(maybe_ssid, maybe_mac)
    packets = addDistanceToPackets(packets.all())
    #packets = session.query(database.Packet)
    return render_template(template, title=title, packets=packets, ssid=maybe_ss

@app.route('/packetViewRepeat')
@databaseUploadRequired
def repeatPacketView():
    title = 'Packet-by-Packet View'
    template = 'packetView.html'

    packets = session.query(database.Packet)
```

```
    repeat_users = get_return_users(packets)
    rackets = _filterPacketsByUsers(repeat_users)
    addDistanceToPackets(rackets.all())
    print rackets
    return render_template(template, title=title, packets=rackets)


def _filterPackets(ssid, mac):
    query = session.query(database.Packet)
    if ssid is not None:
        query = query.filter(database.Packet.ssid == ssid)
    if mac is not None:
        query = query.filter(database.Packet.mac == mac)
    return query

def _filterPacketsByUsers(macs):
    query = session.query(database.Packet)
    query = query.filter(database.Packet.mac.in_(macs))
    return query

def addDistanceToPackets(packets):
    for packet in packets:
        packet.distance = get_distance(packet.signal)
    return packets
```

```
@app.route('/repeatDensityGraph')
@databaseUploadRequired
def densityGraph():
    title = 'Density Graph'
    template = 'densityGraph.html'
    packets = session.query(database.Packet)
    repeat_users = get_return_users(packets)
    packets = _filterPacketsByUsers(repeat_users)
    hour_count = countByHour(packets)

    # Attempt to get the real database.
    initializeDatabaseIfNotInitialized()
    return render_template(template, title=title, packets=packets, hour_count=h



@app.route('/densityGraph')
@databaseUploadRequired
def userDensityGraph():
    title = 'Density Graph'
    template = 'userDensityGraph.html'

    maybe_mac = request.args.get("mac")
    maybe_ssid = request.args.get("ssid")
    packets = _filterPackets(maybe_ssid, maybe_mac)
    all_packets = session.query(database.Packet)
```

```
    hour_count = countByHour(packets)

    # Attempt to get the real database.
    initializeDatabaseIfNotInitialized();

    print get_return_users(all_packets);
    return render_template(template, title=title, packets=packets, hour_count=ho
                           users=flatten_users(all_packets), macs=flatten_macs(a

"""
countByHour creates a count of unique MAC addresses encountered and breaks it do
"""
def countByHour(packets):
    hour_counter = defaultdict(set)
    for packet in packets:
        hour_counter[packet.time.hour].add(packet.mac)

    visitors = {h: len(v) for (h, v) in hour_counter.iteritems()}
    for i in xrange(0,24):
        if i not in visitors:
            visitors[i] = 0

    print visitors
    return visitors
```

```
@app.route('/loadDatabase', methods=['GET', 'POST'])
def uploadDatabase():
    uploadCompleteTitle = 'Upload Complete!'
    uploadCompleteTemplate = 'uploadComplete.html'

    uploadTitle = 'Upload a SQLite Database'
    uploadTemplate = 'loadDatabase.html'

    # Handle the file upload request.
    if request.method == 'POST':
        file = request.files['file']
        if file and allowed_file(file.filename):
            filename = "packetDatabase.db"
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
            print "wooho"
            return render_template(uploadCompleteTemplate, title=uploadComplete
        return render_template(uploadTemplate, title=uploadTitle, error="There v

    # Default to showing the upload.
    return render_template(uploadTemplate, title=uploadTitle)


@app.route('/loadDatabase2', methods=['GET', 'POST'])
def uploadDatabase2():
    uploadCompleteTitle = 'Upload Complete!'
```

```
    uploadCompleteTemplate = 'uploadComplete.html'

    uploadTitle = 'Upload a SQLite Database'
    uploadTemplate = 'loadDatabaseWithLocation.html'

    # Handle the file upload request.
    if request.method == 'POST':
        print request.form
        file = request.files['file']
        if file and allowed_file(file.filename):
            filename = "packetDatabase2.db"
            global location_2
            location_2 = (float(request.form['x']), float(request.form['y']))
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
            return render_template(uploadCompleteTemplate, title=uploadComplete
        return render_template(uploadTemplate, title=uploadTitle, error="There v

    # Default to showing the upload.
    return render_template(uploadTemplate, title=uploadTitle)


# Helper functions for location.
def addLocationToPackets(first, second):
    closestPointIter = iter(second)
    for packet in first:
```

```
            packet.distance = get_distance(packet.signal)
            closestPoint = find_most_recent_point(packet, closestPointIter)
            if (closestPoint is None):
                continue
            closestPoint.distance = get_distance(closestPoint.signal)
            packet.location = IntersectPoints(complex(0, 0), complex(location_2[0],
            if packet.location == True:
                packet.location = IntersectPoints(complex(0, 0), complex(location_2
        return first

def reduceToSingleLocation(first, third):
        closestPointIter = iter(third)
        for packet in first:
            closestPoint = find_most_recent_point(packet, closestPointIter)
            closestPoint.distance = get_distance(closestPoint.signal)

            # Find distance difference between third point.
            closestPoint.location = [(distance(p, complex(location_3[0], location_3
        return first

@app.route('/packetView2')
@databaseUploadRequired
@database2UploadRequired
def packetViewWithLocation():
        title = 'Packet-by-Packet View'
```

```
    template = 'packetViewWithLocation.html'

    maybe_ssid = request.args.get("ssid")
    maybe_mac = request.args.get("mac")

    packets = _filterPackets(maybe_ssid, maybe_mac)
    packets = addLocationToPackets(packets.all(),
        session2.query(database.Packet))
    #packets = c
    return render_template(template, title=title, packets=packets,
                           ssid=maybe_ssid, mac=maybe_mac)


'''
####################
##HELPER FUNCTIONS##
####################
'''
def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1]\
                               in app.config['ALLOWED_EXTENSIONS']

def flatten_users(packets):
    ssids = set()
    for packet in packets:
```

```
            ssids.add(packet.ssid)
        return ssids

def flatten_macs(packets):
        macs = set()
        for packet in packets:
            macs.add(packet.mac)
        return macs

def get_return_users(packets):
        user_day_map = defaultdict(set)
        for packet in packets:
            user_day_map[packet.mac].add(packet.time.date())

        macs = []
        print user_day_map
        for mac in user_day_map:
            if len(user_day_map[mac]) > 1:
                print "found one", mac
                macs.append(mac)

        return macs

import math
```

```
# Returns the distance in feet.
def get_distance(dbm):
    exponent = (27.55 - (20 * math.log10(2412)) + math.fabs(dbm)) / 20.0
    return 10 ** exponent

# Determines whether two circles collide and, if applicable,
# the points at which their borders intersect.
# Based on an algorithm described by Paul Bourke:
# http://local.wasp.uwa.edu.au/~pbourke/geometry/2circle/
def IntersectPoints(P0, P1, r0, r1):
    if type(P0) != complex or type(P1) != complex:
        raise TypeError("P0 and P1 must be complex types")
    # d = distance
    d = math.sqrt((P1.real - P0.real)**2 + (P1.imag - P0.imag)**2)
    # n**2 in Python means "n to the power of 2"
    # note: d = a + b

    if d > (r0 + r1):
        return False
    elif d < abs(r0 - r1):
        return True
    elif d == 0:
        return True
    else:
        a = (r0**2 - r1**2 + d**2) / (2 * d)
```

```
        b = d - a
        h = math.sqrt(r0**2 - a**2)
        P2 = P0 + a * (P1 - P0) / d

        i1x = P2.real + h * (P1.imag - P0.imag) / d
        i1y = P2.imag - h * (P1.real - P0.real) / d
        i2x = P2.real - h * (P1.imag - P0.imag) / d
        i2y = P2.imag + h * (P1.real - P0.real) / d

        i1 = complex(i1x, i1y)
        i2 = complex(i2x, i2y)

        return [i1, i2]

def find_most_recent_point(my_point, points):
    soonest = None
    try:
        soonest = points.next()
        while True:
            point = points.next()
            soonest_difference = abs(soonest.time - my_point.time)
            current_difference = abs(point.time - my_point.time)
            if current_difference <= soonest_difference:
                soonest = point
            else:
```

```
                break
        except Exception as e:
            return soonest

        return soonest

def distance(P0, P1):
    return math.sqrt((P0.real - P1.real)**2 + (P0.complex - P1.complex)**2)
```

8.3.4 app/database.py

```
import os, sys, datetime
from sqlalchemy import Column, Integer, String, DateTime, ForeignKey
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import create_engine

Base = declarative_base()
'''
class MACAddress(Base):
    __tablename__ = 'mac'
    # We need an id as address is not a good primary key, due to mutability.
    id = Column(Integer, primary_key=True)
    address = Column(String(17), nullable=False)

class SSID(Base):
    __tablename__ = 'ssid'
```

```
    id = Column(Integer, primary_key=True)
    # SSIDs have a 31 or 32 character limit in name length.
    name = Column(String(32), nullable=False)
'''
class Packet(Base):
    __tablename__ = 'packet'
    id = Column(Integer, primary_key=True)
    mac = Column(String(17), nullable=False)
    ssid = Column(String(32))
    time = Column(DateTime, default=datetime.datetime.now())
    signal = Column(Integer)

def createNewDatabase(filename):
    engine = create_engine('sqlite:///%s.db' % filename)
    Base.metadata.create_all(engine)

def buildEngine(filepath):
    return create_engine("sqlite:///" + filepath, convert_unicode=True)
```

8.3.5 app/templates/base.html

```
<html>
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">
```

```
    {% if title %}
    <title>{{ title }} - Wifi Scanning Device</title>
    {% else %}
    <title>Wifi Scanning Device</title>
    {% endif %}

    <!-- Bootstrap -->
    <link href="/static/css/bootstrap.min.css" rel="stylesheet">
    <!--<link href="/static/css/wifiscanner.css" rel="stylesheet"/> -->
</head>
<body>
    <!-- Header -->
    <div class="container">
        <div class="navbar-header">
          <button type="button" class="navbar-toggle collapsed"
                  data-toggle="collapse" data-target="#navbar"
                  aria-expanded="false" aria-controls="navbar">
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
          </button>
          <a class="navbar-brand" href="#">Wifi Tracker</a>
        </div>
        <div id="navbar" class="navbar-collapse collapse">
```

```
            <ul class="nav navbar-nav">
              <li><a href="/index">Home</a></li>
              <li><a href="/loadDatabase">Upload Database File</a></li>
              <li><a href="/packetView">Packet View</a></li>
              <li><a href="/packetViewRepeat">Packet View for Return
                  Customers</a></li>
              <li><a href="/densityGraph">Density Graph</a></li>
              <li><a href="/repeatDensityGraph">Density Graph for Return
                  Customers</a></li>
            </ul>
          </div><!--/.nav-collapse -->
        </div>

        {% block content %}{% endblock %}

        <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.
        <!-- Include all compiled plugins (below), or include individual files a
        <script src="static/js/bootstrap.min.js"></script>
    </body>
</html>
```

8.3.6 app/templates/databaseNotUploaded.html

```
{% extends "base.html" %}
{% block content %}
```

```
<div class="container">
   <h1>Please <a href="/loadDatabase">upload a database</a>
        to begin visualizations.</h1>
</div>
{% endblock %}
```

8.3.7 app/templates/densityGraph.html

```
{% extends "base.html" %}
{% block content %}
<div class="container">
  <script src="/static/js/chart.js"></script>
  <div>
      Here are all of the detected packets.
  </div>
  <div>
      <!--Load the AJAX API-->
      <script type="text/javascript" src="https://www.google.com/jsapi"></script
      <script type="text/javascript">

        // Load the Visualization API and the piechart package.
        google.load('visualization', '1.0', {'packages':['corechart']});

        // Set a callback to run when the Google Visualization API is loaded.
        google.setOnLoadCallback(drawChart);
```

```
// Callback that creates and populates a data table,
// instantiates the pie chart, passes in the data and
// draws it.
function drawChart() {

  // Create the data table.
  var data = new google.visualization.DataTable();
  data.addColumn('string', 'Time');
  data.addColumn('number', 'Visitors');
  data.addRows([
  {% for hour, count in hour_count.iteritems() %}
    ['{{hour}}', {{count}}],
  {% endfor %}
  ])

  // Set chart options
  var options = {'title':'Packet Density',
                 'width':800,
                 'height':600};

  // Instantiate and draw our chart, passing in some options.
  var chart = new google.visualization.LineChart(
          document.getElementById('chart_div'));
  function selectHandler() {
    var selectedItem = chart.getSelection()[0];
```

```
            if (selectedItem) {
              var time = data.getValue(selectedItem.row, 0);
              console.log('The user selected ' + time);
            }
          }

          google.visualization.events.addListener(chart, 'select', selectHandler
          chart.draw(data, options);
        }
      </script>
      <div id="chart_div"></canvas>
</div>
</div>
</div>
{% endblock %}
```

8.3.8 app/templates/index.html

```
{% extends "base.html" %}
{% block content %}
<div class="container">
   <h1>Welcome to the Wifi Tracking Visualization Tool!</h1>
   <div>
        Hello! Welcome to the location tracking frontend.
         This is where you can see all of the lovely work that your
         wifi scanning device has detected.
```

```
    </div>
</div>
{% endblock %}
```

8.3.9 app/templates/loadDatabase.html

```
{% extends "base.html" %}
{% block content %}
<div class="container">
    {% if error %}
      <h1 class='error'>{{ error }}</h1>
    {% endif %}
    <h1>Please Upload a File!</h1>
    <form action="" method=post enctype=multipart/form-data>
      <p><input type=file name=file>
         <input type=submit value=Upload>
    </form>
</div> <!-- Container -->
{% endblock %}
```

8.3.10 app/templates/loadDatabaseWithLocation.html

```
{% extends "base.html" %}
{% block content %}
<div class="container">
    {% if error %}
      <h1 class='error'>{{ error }}</h1>
    {% endif %}
```

```
    <h1>Please Upload a File!</h1>
    <form action="" method=post enctype=multipart/form-data>
        <input type=file name=file>
        <input type="number" id="myNumber" value="2" name="x">
        <input type="number" id="myNumber2" value="2" name="y">
        <input type=submit value=Upload>
    </form>
</div> <!-- Container -->
{% endblock %}
```

8.3.11 app/templates/packetView.html

```
{% extends "base.html" %}
{% block content %}
<div class="container">
    <div>
        Here are all of the detected packets.
    </div>
    {% if mac is not none or ssid is not none %}
    <div>
        To clear filters, please click <a href="?">here</a>.
    </div>
    {% endif %}
    <div class="table-responsive">
        <table class="table table-striped">
        <tr>
```

```
            <th class='packet_field ssid'>SSID</th>
            <th class='packet_field mac'>MAC</th>
            <th class='packet_field time'>Time Recorded</th>
            <th class='packet_field signal'>Signal Strength</th>
            <th class='packet_field distance'>Distance from Node</th>
        </tr>
        {% for packet in packets %}
        <tr>
            <td class='packet_field ssid'><a href="?ssid={{ packet.ssid }}
            {% if mac is not none %}&mac={{mac}}{% endif %}">{{ packet.ssid }}
            </a></td>
            <td class='packet_field mac'><a href="?mac={{ packet.mac }}
            {% if ssid is not none %}&ssid={{ssid}}{% endif %}">{{ packet.mac }}
            </a></td>
            <td class='packet_field time'>{{ packet.time }}</td>
            <td class='packet_field signal'>{{ packet.signal }}</td>
            <td class='packet_field distance'>{{ packet.distance }}</td>
        </tr>
        {% endfor %}
        </table>
    </div>
</div> <!-- Container -->
{% endblock %}
```

8.3.12 app/templates/packetViewWithLocation.html

```
{% extends "base.html" %}
{% block content %}
<div class="container">
    <div>
        Here are all of the detected packets.
    </div>
    {% if mac is not none or ssid is not none %}
    <div>
        To clear filters, please click <a href="?">here</a>.
    </div>
    {% endif %}
    <div class="table-responsive">
        <table class="table table-striped">
        <tr>
            <th class='packet_field ssid'>SSID</th>
            <th class='packet_field mac'>MAC</th>
            <th class='packet_field time'>Time Recorded</th>
            <th class='packet_field signal'>Signal Strength</th>
            <th class='packet_field distance'>Location</th>
        </tr>
        {% for packet in packets %}
        <tr>
            <td class='packet_field ssid'><a href="?ssid={{ packet.ssid }}
            {% if mac is not none %}&mac={{mac}}{% endif %}">{{ packet.ssid }}
            </a></td>
```

```
            <td class='packet_field mac'><a href="?mac={{ packet.mac }}
            {% if ssid is not none %}&ssid={{ssid}}{% endif %}">{{ packet.mac }}
            </a></td>
            <td class='packet_field time'>{{ packet.time }}</td>
            <td class='packet_field signal'>{{ packet.signal }}</td>
            <td class='packet_field distance'>{{ packet.location[0] }}</td>
        </tr>
        {% endfor %}
        </table>
    </div>
</div> <!-- Container -->
{% endblock %}
```

8.3.13 app/templates/uploadComplete.html

```
{% extends "base.html" %}
{% block content %}
<div class="container">
   <h1>Upload complete!</h1>
   <div>You will now be able to utilize the rest of the site's functionality.
    </div>
</div> <!-- Container -->
{% endblock %}
```

8.3.14 app/templates/userDensityGraph.html

```
{% extends "base.html" %}
{% block content %}
```

```
<div class="container">
    <script src="/static/js/charts.js"></script>
    <div>
      Here are all of the detected packets
      {% if mac is not none %} for the MAC address {{mac}}{% endif %}
      {% if mac is not none and ssid is not none %} and{% endif %}
      {% if ssid is not none %} for the SSID {{ssid}}{% endif %}.
    </div>

    <div>
        <select id="ssidSelect">
            {% for _ssid in users %}
                <option value="{{ _ssid }}">{{ _ssid }}</option>
            {% endfor %}
        </select>

        <select id="macSelect">
            <option value=""></option>
            {% for _mac in macs %}
                <option value="{{ _mac }}">{{ _mac }}</option>
            {% endfor %}
        </select>

        <button id="filterButton">Filter</button>
    </div>
```

```
<div>
    <!--Load the AJAX API-->
    <script type="text/javascript" src="https://www.google.com/jsapi">

    </script>
    <script type="text/javascript">

    // Load the Visualization API and the piechart package.
    google.load('visualization', '1.0', {'packages':['corechart']});

    // Set a callback to run when the Google Visualization API is loaded.
    google.setOnLoadCallback(drawChart);

    // Callback that creates and populates a data table,
    // instantiates the pie chart, passes in the data and
    // draws it.
    function drawChart() {

      // Create the data table.
      var data = new google.visualization.DataTable();
      data.addColumn('string', 'Time');
      data.addColumn('number', 'Visitors');
      data.addRows([
      {% for hour, count in hour_count.iteritems() %}
        ['{{hour}}', {{count}}],
```

```
  {% endfor %}
  ]);

  // Set chart options
  var options = {'title':'Packet Density',
                 'width':800,
                 'height':600};

  // Instantiate and draw our chart, passing in some options.
  var chart = new google.visualization.LineChart(
          document.getElementById('chart_div'));
  function selectHandler() {
    var selectedItem = chart.getSelection()[0];
    if (selectedItem) {
      var time = data.getValue(selectedItem.row, 0);
      console.log('The user selected ' + time);
    }
  }

  google.visualization.events.addListener(chart, 'select', selectHandler
  chart.draw(data, options);
}
</script>
<div id="chart_div"></canvas>
</div>
```

```
    </div>
</div>
{% endblock %}
```

8.3.15 app/static/js/charts.js

```
/**
 * Created by dhnishi on 4/13/15.
 */

window.onload = function() {
    var button = document.getElementById('filterButton');
    button.onclick = function() {
        var ssid = document.getElementById('ssidSelect').value;
        var mac = document.getElementById('macSelect').value;
        console.log(ssid, mac);

        var url= "/densityGraph?";
        if (ssid !== "") {
            url += "ssid=" + ssid + "&";
        }
        if (mac !== "") {
            url += "mac=" + mac;
        }
        location.href = url;
    }
```

```
};
```

References

[1] "Mobile location technologies." [Online]. Available: http://courses.cs.washington.edu/courses/csep590b/11wi/lectures/110124-location.pdf

[2] D. Alba, "Navizon's new location technology lets you find your friends indoors."

[3] darkAudax, "Tutorial: Wpa packet capture explained." [Online]. Available: http://www.aircrack-ng.org/doku.php?id=wpa_capture

[4] Dot11.info, "802.11 management frames," May 2015. [Online]. Available: http://dot11.info/index.php?title=Chapter_4_-_802.11_Management_frames

[5] A. Musa, "Tracking unmodified smartphones using wi-fi monitors," Master's thesis, Unniversity of Illinois at Chicago, 2012.

[6] Wikipedia, "Free-space path loss — wikipedia, the free encyclopedia," 2015, [Online; accessed 30-May-2015]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Free-space_path_loss&oldid=651671832

[7] ——, "List of wlan channels," *Wikipedia, The Free Encyclopedia*, 2015.