Marauder's Map: Techincal Whitepaper

Samuel Anklesaria, Jason Brooks, Alejandro Carrillo

Yale University • CPSC 183

samuel.anklesaria@yale.edu • jason.brooks@yale.edu • alejandro.carrillo@yale.edu

Abstract

In this paper, we describe a system to enable MAC address-based tracking of Wifi devices across Yale campus. Low-cost packet sniffing drones are distributed across Yale and programmed to report any unique devices they see communicating over WiFi. Reports are merged in a central database in the cloud. A website front-end enables users to input their smartphone or laptop's MAC address and see a history of their travel across the campus.

I. Introduction

In order for electronic devices communicating over WiFi to establish identity and maintain reliable connections, they depend on MAC addresses to uniquely identify who is sending and who is to receive what packets. Note that, in principle, *all* devices in range of a WiFi transmitter receive the packets of a sender; in typical operation, packets that are addressed to a MAC address other than that of a listener on a network are ignored by that listener. It is the habit of all WiFi-enabled devices to broadcast their unique MAC address to the world that enables us to build a database of when and where MAC addresses are seen. Furthermore, by operating our wireless chipset in monitor mode, such that we are not associated with any access point but instead listen passively to all observable traffic, we can see devices communicating on every network within range without ever being seen.

II. Methods

I. Overview

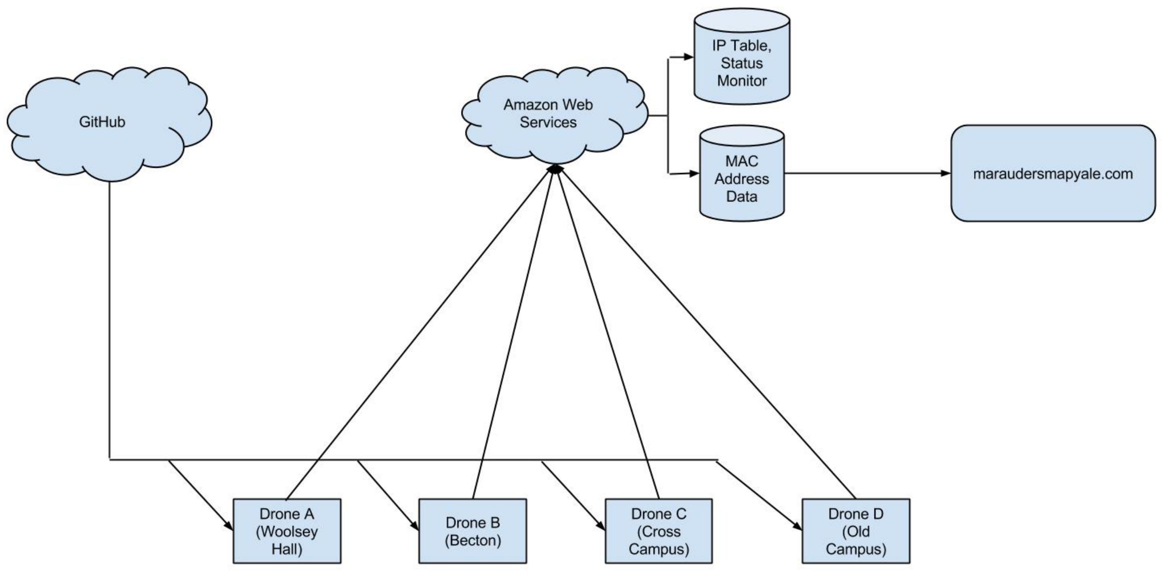


Figure 1: *An overview of the Maurader's Map backend*

At the frontlines of our data collection system are our drones, distributed across campus in strategic locations. Each drone pulls in the latest drone code from GitHub, updates an Amazon DynamoDB table with its latest LAN IP address, and begins reporting the times at which it sees each MAC address that comes within range of the drone. These reports are stored in another DynamoDB table, which our front-end web application uses for content.

II. Drone Construction

Each node (or drone) in our monitoring system consists of a small, low-cost Raspberry Pi computer and an Alfa wireless USB adapter. In some cases a powered USB hub is included in order to accomodate the needs of the powerful USB WiFi adapter. Each drone can be encapsulated in a large RadioShack project enclosure. (See figure 2)

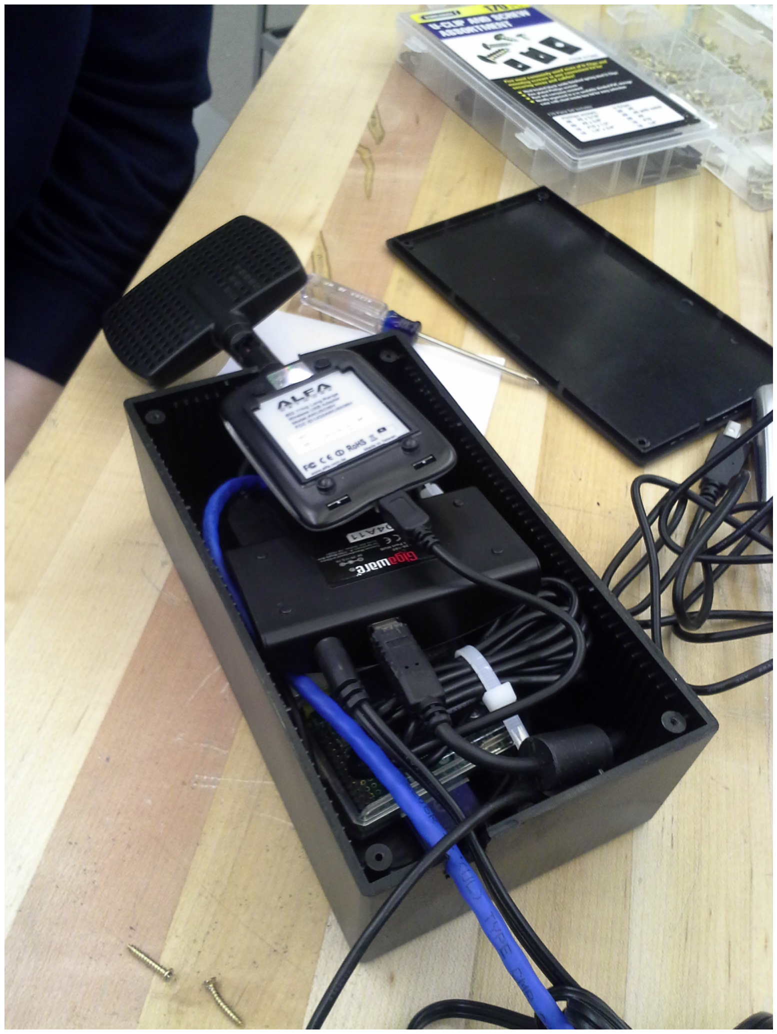
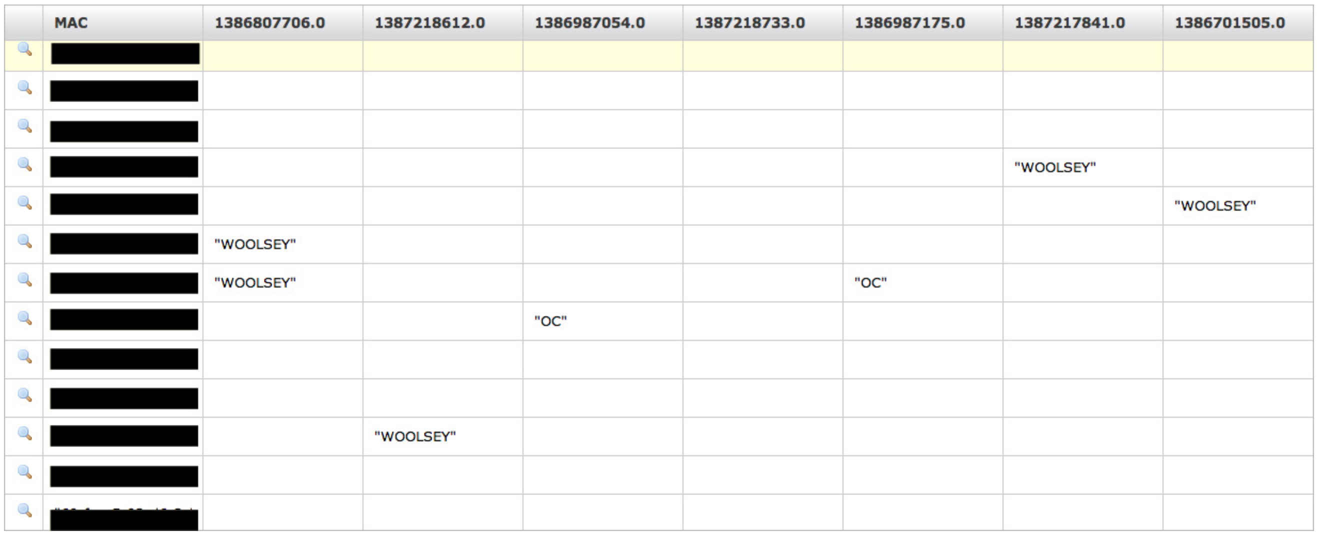


Figure 2: *A look inside a drone.*

The Pi computers are running the default Debian distribution for the Raspberry Pi, "wheezy." Upon boot, a script in /etc/init.d instructs the Pi to pull the latest source code release from GitHub. The bootstrapper then invokes our scraper program. The scraper program first updates an item in a DynamoDB table to report its IP address on the wired LAN. This action functions not only to give us an address through which we can service each drone during operation, but also as a periodic "heartbeat" indicating the health status of the Pi. Our program then reads packet dumps from stdin, parses out MAC addresses and timestamps, and organizes the information so that it can be neatly inserted into our central database. Our packet dumps are generated using stdout from "tcpdump" and are piped directly to our scraper. The scraper periodically writes its data buffer to the central database. Figure 3 shows what a typical set of database items looks like, with MAC addresses redacted for privacy reasons. Note that DynamoDB is a NoSQL database, which enables us to arbitrarily add timestamps as attributes to an item that represents a single MAC address.



MAC	1386807706.0	1387218612.0	1386987054.0	1387218733.0	1386987175.0	1387217841.0	1386701505.0
[redacted]							
[redacted]							
[redacted]							
[redacted]						"WOOLSEY"	
[redacted]							"WOOLSEY"
[redacted]	"WOOLSEY"						
[redacted]	"WOOLSEY"				"OC"		
[redacted]			"OC"				
[redacted]							
[redacted]							
[redacted]		"WOOLSEY"					
[redacted]							
[redacted]							

Figure 3: *Sample items from our MAC address table. The leftmost column represents the MAC address. Successive columns represent time intervals (in Unix time) over which a MAC might have been seen. If a MAC has been seen at a particular timestamp, the location at which it was observed is noted as the valu.*

The NoSQL "schema" described in Figure 3 was chosen in order to optimize the dataset for selecting MAC addresses. Of course it is possible to format the data in a more traditional sense, in which the keys (or columns) represent attributes (like "timestamp") and not values (like 1387233645) in themselves. Such organization then requires a query search over the database to find all the locations and times a specified MAC was observed. While this would likely perform fast enough for our purposes, we chose the former database layout for the simplicity of updating and retrieving MAC items.

III. Marauder's Map Website

A website was created using NodeJS on the backend and AngularJS on the front end for the purpose of displaying the information from the NoSQL database to a user. A user is prompted to enter their MAC address, and then an ajax call is made to the database to see if that MAC address exists in our system. If the address does exist, AngularJS handles a user interface which allows a slider to scroll between all locations and times for that address. If the address does not exist, an error is returned, and the user is prompted to enter a different MAC address.