MobIntel: Sensing and Analytics Infrastructure for Urban Mobility Intelligence

Stepan Mazokha, Fanchen Bao, Jiannan Zhai, Jason O. Hallstrom
I-SENSE
Florida Atlantic University
Boca Raton, FL, USA
{smazokha2016, fbao2015, jzhai, jhallstrom}@fau.edu

***Abstract*—Mobility monitoring in urban environments can provide valuable insights into pedestrian and vehicle movement. Understanding the causes and effects of changing mobility patterns can help city officials and businesses optimize operations and support economic development. In this paper, we present MobIntel, an alternative to visual surveillance technologies for mobility monitoring. We deployed multiple radio frequency sensors in downtown West Palm Beach and enabled a system for providing valuable metrics concerning pedestrian activity patterns. We discuss several obstacles to accurate trajectory monitoring, such as MAC address randomization and sensor range issues.**

***Index Terms*—Mobility Intelligence, Mobility Monitoring, WiFi Probe Requests, IoT.**

I. Introduction

The Internet of Things (IoT) has brought a renewed interest in analyzing behavior in public spaces. Sensing in public environments creates a host of opportunities, including an ability to analyze mobility patterns, occupancy trends, and more. The collected data can be used to improve the ways common spaces are used and foster economic development.

Consider the "Supermanzana" approach [1] developed in the Poblenou neighborhood of Barcelona as an example. The project focused on improving the liveability and sustainability of public spaces. Even without integrating sensing solutions, the city government was able to orchestrate speed limits and redesign streets to merge smaller city blocks into larger "superblocks." These actions ultimately helped decrease automobile traffic, in favor of more pedestrians, and increased small business development.

Nowadays, city governments interested in adopting such changes have access to reliable data about city traffic. In addition to enabling more accurate changes, solutions providing such insights also offer tools for observing immediate changes in public spaces and analyze the effectiveness of adopted decisions. Such data could be supplied by monitoring traffic patterns, which would serve as the basis for robust pre/post analysis to guide city improvements.

One example of a system that provides this type of data is MotionLoft [2, 3, 4, 5, 6], a vision-based system for counting pedestrians and vehicles within a city. The system is in production in several locations, including the City of West Palm Beach, Florida.

Unfortunately, MotionLoft suffers from a number of complications, which can limit its adoption in larger cities. For instance, in May of 2019, the City of San Francisco banned the use of facial recognition in public spaces [7]. This turn of events may be a watershed moment for smart city applications built on visual analysis. On the other hand, it opens up opportunities for further research into alternative tools built with privacy constraints at the core of the design.

In this work, we present the first iteration of the MobIntel platform [8], which addresses privacy concerns and supports relatively inexpensive deployment within a city. MobIntel does not use visual analysis; instead, it uses RF-based probing activities of WiFi and Bluetooth devices. Our small set of sensors, deployed in downtown West Palm Beach, passively collect RSSI data from probing activities, as well as the hashed MAC addresses of emitting devices. We emphasize the system's use of hashing as an initial step toward providing anonymity to devices and owners.

Collected probe data is aggregated into sessions, periods of time during which a sensor continuously collects probe requests sent by the same device. Sessions form the basis for tracking outdoor occupancy and persistence of pedestrians in designated sensing areas. We later compare these findings with metrics provided by MotionLoft. Based on discrepancies in this comparison, we present future steps for system improvement. This includes approaches such as MAC address de-randomization, pedestrian trajectory tracking, improved positioning using Kalman filters, and more.

II. Related Work

Wireless probe analysis represents an affordable and scalable solution to urban mobility monitoring. The area has received significant attention over the last decade. Here we summarize prior work in the area, as well as alternative approaches, such as video surveillance and others.

A. RF-based Solutions

Using probing packets sent by WiFi and Bluetooth hardware for urban mobility monitoring has been considered by other researchers over the last several years. The growth in sensing infrastructure and use of personal mobile peripherals creates opportunities to better understand mobility patterns in public environments.

978-1-7281-6997-2/20/$31.00 ©2020 IEEE
DOI 10.1109/SMARTCOMP50058.2020.00034

Longo et al. [9] consider occupancy estimation using readily available WiFi and Bluetooth sniffing hardware. Based on probing activities (both classic and low-energy extensions), their system delivers results in the form of a visual dashboard and messenger bot, which together allow users to quickly find an available room for work and offers metrics for owners of indoor workspaces. The authors discuss two obstacles to monitoring accuracy. The first issue is the inability of a sniffer to distinguish probing requests from devices in a particular area under observation. The second issue is a procedure of generating virtual MAC addresses, implemented by manufacturers to complicate device tracking across multiple networks.

Redondi et al. [10] discuss the use of passive RF measurements from wireless hardware. The authors consider device localization, profiling, and classification. They present a set of extracted features from probe requests, including dwell and presence time. The work is based on data collected from devices emitting physical MAC addresses and emphasizes the increased adoption of virtual MAC addresses.

Similarly, Mikkelsen et al. [11] discuss RF sensing for occupancy estimation. The authors apply the approach to public transportation vehicles. The intent is to provide bus drivers and potential passengers with information on vehicle load. However, the solution suffers from several issues, including MAC address randomization, passengers carrying more than one probe-emitting device, and sensors capturing probing activities outside of the vehicle.

Luo et al. [12] describe the use of RSSI values to locate devices in a construction site and compare their results with data obtained from inside a building. They test four different localization algorithms: MinMax, Maximum Likelihood, Ring Overlapping Circle RSSI, and k-Nearest Neighbor. The results indicate that MinMax offers the best overall accuracy. The authors argue that with an appropriate density of RF-based sensors, such approach is efficient and cost-effective.

Vattapparamban et al. [13] demonstrate the viability of probe request sniffing as a solution for occupancy monitoring in smart buildings. Similar to the work above, the authors present a solution that applies linear least squares to estimate a device's position, and then refine the estimate using k-nearest neighbor clustering to group discovered devices into one of 8 defined zones within the building. Randomization of MAC addresses is again a focal point of the discussion.

B. Alternative Approaches

Occupancy tracking and localization are not limited to RF-based solutions. Although probe-based solutions are among the most efficient and low-cost, issues such as MAC address randomization, RSSI thresholding, multipath propagation effects, channel uncertainty, and the possibility of multiple devices per person limit the achievable accuracy. Here we discuss alternative solutions.

Göçer et al. [14] study occupancy patterns and walking routes during different seasons on a university campus using video surveillance and spatiotemporal mapping. The team applies machine learning and spatial statistical analysis to detect and track pedestrians using data transferred to a geographic information system (GIS).

Steyer et al. [15] describe an object tracking approach based on a dynamic occupancy grid. The observed area is partitioned into a matrix of cells and observed objects are mapped to these cells. False-positive mappings are avoided by determining object tracks using clustering and velocity variance analysis of adjacent occupied cells based on their speed of movement. The authors also consider object size as a factor in areas with a high density of pedestrians and moving vehicles. Results demonstrate the robustness of object tracking in highly dense environments, even in the presence of object occlusion.

III. Pilot Implementation

The pilot implementation consists of sensing platform, cloud middleware, and web portal. The pilot was deployed in downtown West Palm Beach, FL.

A. Sensing Platform

Regarding hardware, a Raspberry Pi 4 Model B (RPi4) serves as the foundation of the sensing platform [Figure 1A]. It is equipped with a Broadcom BCM2711 SoC, with a 1.5 GHz, 64-bit, quad-core, ARM Cortex-A72 processor, and 4GB RAM. It is portable, powerful, and affordable. The operating system used is Raspbian Buster with kernel version 4.19.

To perform probe request sniffing, the sensor requires a WiFi chip capable of entering *monitor* mode, one of eight modes a WiFi chip can operate in. It allows the chip to capture all wireless packets on a given channel, regardless of whether the packets are associated with an access point (or network). The built-in RPi4 chip does not support monitor mode natively. It is possible, however, to patch the firmware to enable monitor mode via Nexmon [16], an open-source project with a C-based firmware patching framework for Broadcom/Cypress WiFi chips. Unfortunately, we were not able to identify a stable configuration during testing — it generally supported less than 30 minutes of continuous probe request sniffing before halting. Hence, the Alfa AWUS036NHA 802.11n wireless USB adapter was incorporated as the probe request sniffer [Figure 1A]. It is equipped with the Atheros AR9271 chipset, which offers native support for monitor mode.

Battery backup is important to prevent data loss or hardware damage due to power disruption. To meet this requirement, the 18650 UPS HAT & Safe Power Management Expansion Board (X750) [17] and four rechargeable lithium-ion batteries were integrated into the sensor enclosure [Figure 1A]. With the X750, the sensor can remain operational for at least ten hours after losing external power. Further, the X750 allows software to read battery voltage from the built-in MAX17040 Fuel Gauge System via I2C, which enables the sensor to perform safe shutdown when the battery voltage drops below a threshold.

The X750 requires 5V DC power to charge the batteries. To connect the X750 to a regular 110V AC outlet, the RS-15-5 Single Output AC to DC Power Supply is used [18] [Figure 1A].

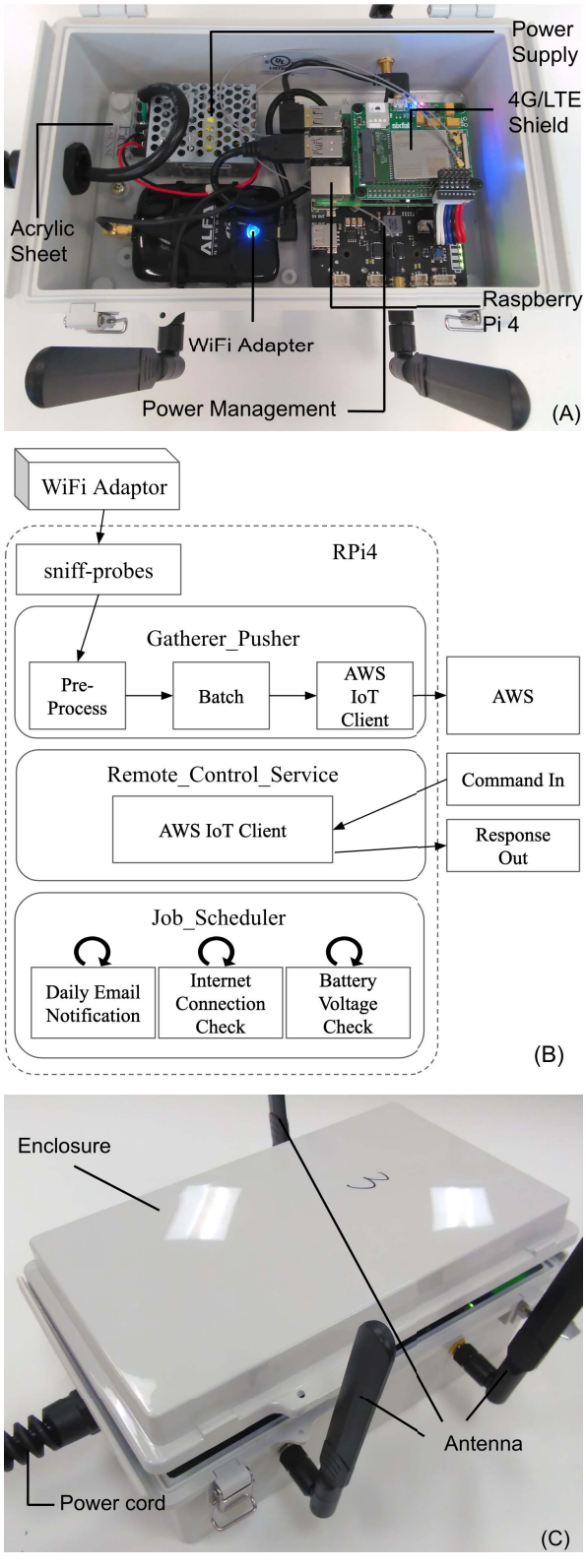

Fig. 1. Sensing Platform Setup

To upload data to AWS, the sensor requires a stable Internet connection. Our design uses the Raspberry Pi 4G/LTE Shield Kit [19] to access a cellular Internet connection [Figure 1A]. The kit contains a Quectel EC25 Mini PCIe 4G/LTE Module (internet connection) and a 4G/LTE Base Shield V2 (interface between the 4G/LTE Module and the RPi4).

The 4G/LTE Shield has two power ports, one through the 5V PWR pin, and the other via USB. The original design connected power to both ports — the 5V PWR pin to the X750 power management board, and the USB to the RPi4. Since power from the X750 was always available, the 4G/LTE Shield would never lose power, even if the RPi4 was shut down. This design led to a problem where the WiFi adapter, which was also connected to the RPi4 via USB, frequently failed to turn on after a RPi4 reboot. This was caused by power leakage from the 4G/LTE Shield to the RPi4. When the RPi4 reboots, a power cycle occurs across all of its USB ports which shuts down, and then turns on USB-connected devices. Since the 4G/LTE Shield never loses power, during a RPi4 reboot, power from the 4G/LTE Shield can leak to the RPi4's USB ports. This potentially interrupts the USB power cycle and prevents the WiFi adapter from turning on.

The problem was resolved by restricting the power supply to the 4G/LTE Shield via USB only (i.e. no power to the 5V PWR pin). This way, during the RPi4 reboot, the 4G/LTE Shield loses power completely. With no power leakage from the shield, the RPi4 performs a proper power cycle on all USB ports, and then turns on the WiFi adapter.

All hardware components are assembled within an enclosure to shield from environmental exposure. Device layers are stacked and mounted to an acrylic sheet, along with the WiFi adapter and power supply. The acrylic sheet is secured within a WQ-50 NEMA Hinged Enclosure [20] designed for outdoor applications. Additional openings are drilled through the walls of the enclosure to accommodate three antennas and one power cord [Figure 1C]. All openings are sealed with waterproof cable glands and SMA adapters.

Regarding software, the sensing software was developed in Python 3.7. It consists of three main applications, each running under a separate process: *Gatherer_Pusher*, *Remote_Control_Service*, and *Job_Scheduler* [Figure 1B].

Gatherer_Pusher is designed to gather, pre-process, batch, and push probe request data to Amazon Web Services (AWS) for storage. Probe request collection is achieved using the open-source *sniff-probes* [21] which uses *tcpdump*. *Tcpdump* is a free packet analyzer that displays TCP/IP packets transmitted over a network. It is a popular command-line interface tool for Unix-like systems to analyze network data. After raw data is collected, the application extracts MAC address, timestamp, WiFi channel, and RSSI from each probe request to form a data record. Once a batch of data records is available (default = 1000), they are pushed to AWS IoT via MQTT messaging.

Remote_Control_Service was created to send a pre-registered remote control command to a specific sensor in order to execute predefined functions and receive a response. For instance, sending a health check command, such as

cpu_temp, results in a sensor responding with its current CPU temperature. Similarly, sending an operational command, such as *reboot*, prompts the sensor to perform a safe reboot, which saves all memory-bound probe request data to a local database, terminates *sniff-probes*, and turns off the WiFi adapter before rebooting the RPi4. *Remote_Control_Service* provides a scalable way to remotely interact with the sensors programmatically. It is mainly used to query a sensor's health state, troubleshoot issues, and perform over-the-air software updates.

Job_Scheduler runs a registered job repeatedly at a given period. Three main jobs are registered to provide health checks on the sensor:

- *Daily Email Notification:* This job sends an email containing three days of logs each morning. The logs contain health status messages recorded by the application throughout its execution.
- *Internet Connection Check:* This job verifies the sensor's Internet connection every ten minutes. If the sensor loses connectivity, it performs a safe reboot.
- *Battery Voltage Check:* This job examines the sensor's battery voltage. If the voltage is below a predefined threshold (default = 3.2 V), the sensor will send an alert email and perform a safe shutdown.

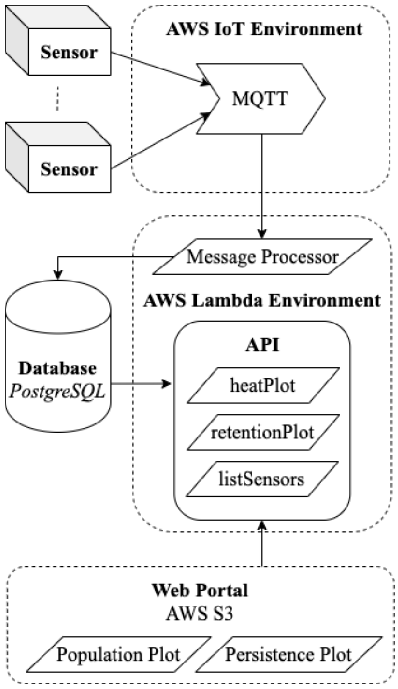

Fig. 2. MobIntel Cloud Architecture

B. Cloud Middleware

The architecture of the MobIntel platform supports a large number of deployments, discussed later in more detail. The system is built on AWS and consists of five main components. They are illustrated in Figure 2: IoT Core, Processing, Database, API, and Web Portal.

MobIntel collects multiple readings every minute. These sensors generate messages containing a unique hardware ID and an array of recorded probing packets. Each transmission happens as soon as a required number of readings is collected. In the current implementation, this number is set to 1,000 packets per message due to the maximum size of the MQTT payload, 128 KB [22].

Next, the message is collected by AWS IoT Core. Each message is submitted to a designated MQTT topic, to which the Processor module is subscribed. The module later processes each of the probing packets in the payload and saves them to the database, both as raw data and sessions. The latter record information about the duration of each device's stay within the sensor's detection range. This data is stored in a managed PostgreSQL database.

The system is capable of adapting to rapid changes in the amount of data captured by deployed sensors. This is particularly important in areas with transient crowds (e.g. weekend street markets). The design relies on the AWS Lambda and API Gateway services, which support independent trigger functions, each capable of fast scalability and parallel data processing, independent of other system components. As a result, third party requests can be executed without causing bottlenecks in data retrieval. Additionally, the system implements sharding for specific functions in the API, which increases the speed of data retrieval. Performance of these endpoints is discussed later.

Finally, a user-facing web portal was built using React.js, deployed on AWS S3. It allows users to easily access all of the processed messages, as well as to compare collected data with information obtained from MotionLoft.

C. Deployments

The pilot deployment includes two sensors installed along Clematis Street, the historical heart of Downtown West Palm Beach, Florida. Figure 3A shows a sensor mounted on a light pole; Figure 3B illustrates the locations of the two sensors. As a vibrant cultural and business center, Clematis Street boasts ample pedestrian traffic; it is an ideal location for pilot testing.

IV. Evaluation

We now present an evaluation of the MobIntel implementation. First, we evaluate the sensor's probe request capture rate. We then analyze the performance of the live system by testing response times of the various API endpoints.

A. Probe Request Capture Rate

The probe request capture rate is defined as the number of probe requests captured by a sensor divided by the total number emitted within the sensor's sensing range. It is used to assess a sensor's capability of handling a high volume of probe requests.

To send probe requests, we created a stressor application using an open source network packet crafting tool, *scapy* [23]. *Scapy* can send probe requests at arbitrary rates, up to the

Fig. 3. MobIntel Sensor Deployment

sending device's physical limit. To capture probe requests, a deployment-ready sensor was used. To ensure that most probe requests reached the sensor, we placed the stressor and the sensor 1.5 m apart and configured both to work on channel 10.

During each trial, the sensor was turned on approximately one minute before the stressor. The stressor continuously sent mock probe requests for one minute at pre-configured probe request rates (2^0, 2^1, ..., 2^{15} counts per second). Each probe request contained a distinct marker identifying the probe request rate. After the mock probe requests were sent at all the rates, the sensor would send additional 1,000 sentinel probe requests to signal the completion of its task. The experiment was terminated when at least one such sentinel was present in the database on AWS.

We conducted two rounds of tests: one with a single stressor (SS), and the other with double stressors (DS). Each round consisted of four separate trials. After each trial, we collected the number of probe requests sent (for each request rate) from *scapy*'s output and obtained the number of captured requests by counting the stored requests that contained the corresponding identifier. The numbers of sent and captured probe requests were averaged over all trials. Results are summarized in Figure 4. The x-axis denotes the stressor probe request rate, and the y-axis denotes the capture rate.

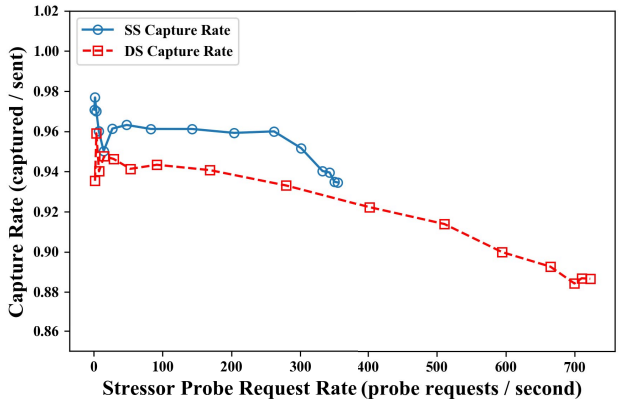

Fig. 4. Sensor Capture Rate

The SS data points cluster around 350 probe requests per second, indicating that the stressor reached its physical limit. This is confirmed by a similar clustering of the DS data points around 700 probe requests per second. When the probe request rate is similar for both SS and DS, the capture rate under DS is always lower than under SS, because DS involves parallel probe request transmission (i.e. two stressors sending at the same time), which at high transmission rates, can result in collisions loss.

The SS curve is also stable, maintaining above 95% until the probe request rate passes 260 probe requests per second. Beyond 260, it drops toward the DS curve. This drop might be due to the same limitation in the sensor as described above. Specifically, when the probe request rate goes above 260 probe requests per second, it is likely that the probe requests arrive at the sensor faster than they can be processed. Once the sensor exhausts its buffer, any further probe requests might not be captured. Although the sensor is under SS, this situation resembles the one under DS. Hence, the trend of the SS curve dropping toward the DS curve. We speculate that if the stressor has a higher physical limit, we will observe the SS curve decreasing further, and probably merging with the DS curve.

Despite the decline in capture rate under DS, the sensor still maintains a decent capture rate (88%), even at 700 probe requests per second. This suggests the sensor is capable of handling at least 2,520,000 probe requests per hour. Given that a smartphone typically sends up to 2,000 probe requests per hour [24], our sensor can monitor at least 1,260 smartphones over an hour. Since the sensor covers approximately 2,290 m^2, with a 27 m radius, it is expected to accommodate a device density of at least 0.55 $smartphones/m^2$.

B. API Throughput

As discussed in Section 3.2, MobIntel receives messages from sensors through AWS IoT Core, and upon arrival, triggers

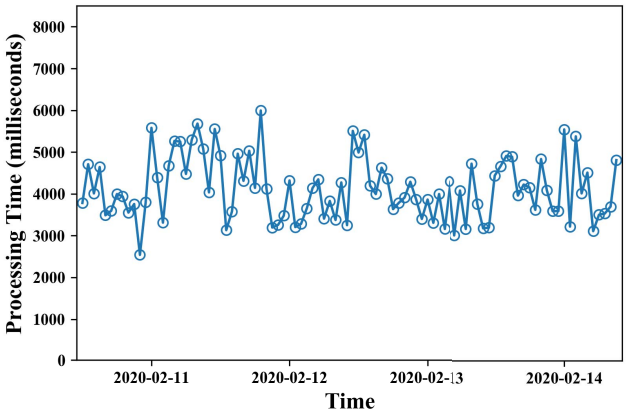

Fig. 5. Lambda Function Processing Time

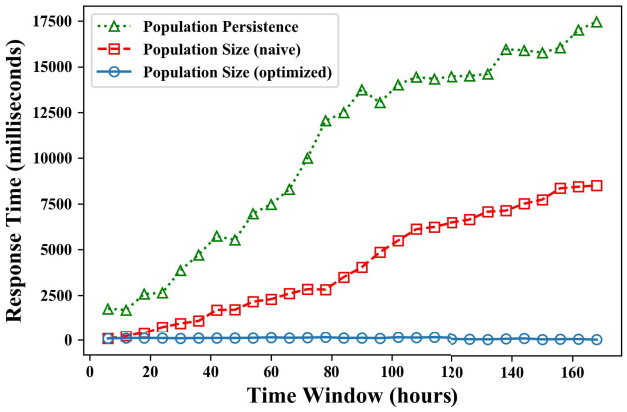

Fig. 6. API Response Time

a designated AWS Lambda function. The processed data is stored in a PostgreSQL database hosted on the AWS Relational Database Service (RDS). This solution has the capability of scaling up to the limitations of these services.

AWS Lambda is limited to 1,000 concurrent executions, with a 15-minute maximum function timeout [25]. AWS IoT Core is able to handle 10,000 inbound messages per second [26]. On average, the triggered Lambda function processes one message in 4.1 seconds, which results in a maximum throughput rate of 244 messages per second. Performance results were sampled using AWS CloudWatch during active use of the system. A summary of the sampled results is illustrated in Figure 5. The x-axis denotes the time at which the function's performance was recorded, and the y-axis denotes the processing time in milliseconds. Given these specifications, the system is capable of capturing up to 14,640,000 probe requests per minute.

C. API Response Time

On average, MobIntel saves about 450,000 new probe requests per day, with just two sensors. The number will grow as more sensors are deployed.

To evaluate API response time, we implemented an application that calls the API endpoints with an increasing date span. The resulting response time is calculated as an average of 5 requests, with the same set of parameters executed on 5 independent machines. Twenty-five measurements are calculated for each time window.

First, we evaluate the performance of the Population Size API. This endpoint returns the total number of recorded probes within the time range specified by the user. Our initial implementation was naive, aggregating probe counts upon request using SQL queries. The solution suffered long processing times as more data was recorded. The performance is illustrated in Figure 6, labeled Population Size (naive). The x-axis denotes the requested time window, and the y-axis denotes the API response time. As the time range increases, the response time increases significantly.

To improve performance, the revised solution calculates probe counts on each sensor for a set of predefined densities, starting with 5 minutes, up to 12 hours. The results are then processed and stored in the database independently of the raw probe requests. As a result, the system performs only a simple query to retrieve the aggregated counts saved in the database. The performance is illustrated in Figure 6, labeled Population Size (optimized).

Next, we evaluate the performance of the Population Persistence API. The API yields a variable number of sessions discovered in the area based on a *dynamically* chosen population (i.e., set of anonymized MAC addresses). As a result, the precomputation discussed above is not viable. The evaluation results are summarized in Figure 6, labeled Population Persistence. The response time of the endpoint increases significantly as the time range increases. Optimization of this endpoint is the focus of future work.

D. System Accuracy

Between November 20, 2019, and December 20, 2019, roughly 4.7 million probe requests were captured, and 97.7% of the probe requests use virtual MAC addresses. Over 2.3 million probe sessions were observed, and 1.9 million unique MAC addresses were extracted.

The MotionLoft system was used for comparison, shown in Figure 7. The x-axis denotes the time span in which the measurements were recorded, and the y-axis denotes the number of recorded probes. The figure shows a strong correlation between the data sets, but MobIntel captures significantly larger counts. This can be attributed to three factors. First, MobIntel sensors have an omnidirectional sensing range, while the MotionLoft cameras are pointed directly towards the pedestrian route on the street. Second, 97.7% of the probe requests captured by MobIntel include virtual MAC addresses, so a device can be interpreted as multiple entities, as its address changes with time. Third, each pedestrian can carry more than one signal-emitting device. We observed requests from phones, laptops, and smart watches, which are all captured by our system.

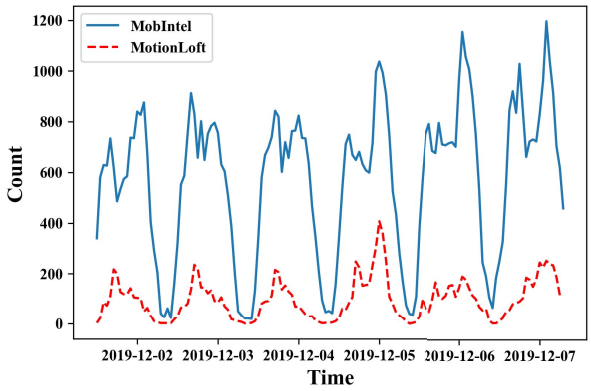

Fig. 7. MobIntel & MotionLoft Comparison

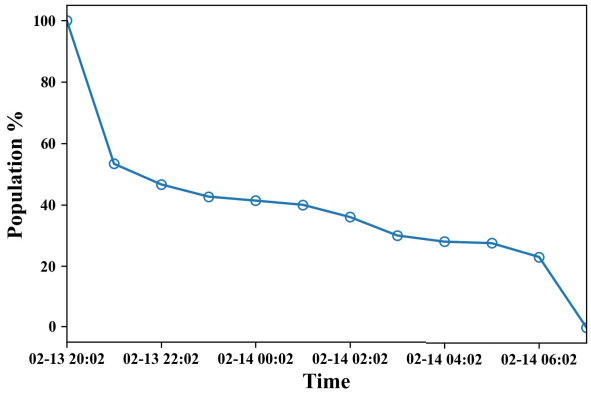

Fig. 9. Population Persistence

We also noticed the presence of constant signal emitters, including routers, point of sale systems, etc. These devices include physical MAC addresses in their probe requests and can be easily identified. The MobIntel Web Portal allows users to remove these devices from the population counts. Their probing activity compared to the MotionLoft data is shown in Figure 8, where the x-axis denotes the time span in which the measurements were recorded, and the y-axis denotes the number of recorded probes. The correlation is weaker, with relatively little variability in MobIntel counts during daytime hours.

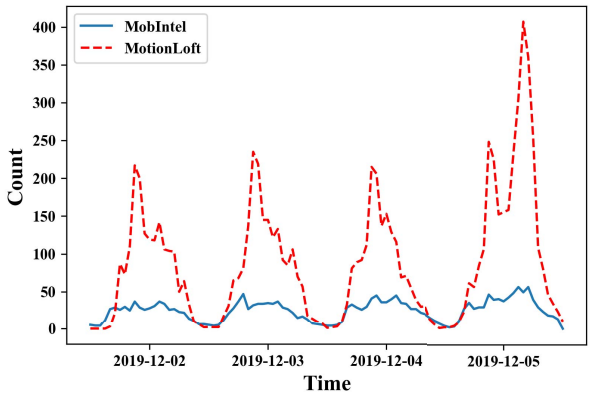

Fig. 8. Physical MACs vs MotionLoft Readings

As noted previously, we have also implemented a technique to visualize population persistence in an observed area. The plots demonstrate how long a group of devices stay in the observed area without leaving. One example is shown in Figure 9. The x-axis denotes the time at which the initial population is recorded, and the y-axis denotes population persistence, which is computed as the percentage of the remaining devices compared to the initial population. The graph illustrates how long unique devices (and their owners) stay in the area visible to a sensor. For instance, at time point 02-14 00:02, the population persistence is approximately 40%, meaning that 40% of the initially detected devices still remain in the sensing area. As time passes, the population percentage decreases from 100%, eventually converging to 0% (i.e. none of the original devices are present in the area). During our observations, we discovered dwell times ranging from 5 seconds, up to 25 consecutive days, with an average persistence of 12 minutes and 7 seconds. It is important to recall that this includes virtual MAC addresses, so the data doesn't directly correspond with device departures.

V. Conclusion

In this paper, we described an approach to monitoring pedestrian and vehicle activities in urban environments based on analysis of wireless probing activities. It consists of a network of RF sensors and cloud infrastructure for processing and visualizing the collected data.

A study on the performance of the system was provided. Our load tests demonstrate a sensor's ability to maintain a sufficient capture rate (88%) under a constant probe request load of 700 counts per second. The sensors are capable of handling a device density of over 0.55 $device/m^2$ in their detection range every hour.

We also evaluated API response time under different request time ranges. The original implementation demonstrated slow performance due to an on-demand calculation strategy. The optimized alternative relies on population size pre-calculation and demonstrates significantly improved performance.

Our future work will focus on further exploration of the load limits of our sensing platform. This can be achieved by either building a stressor with a higher physical limit on probe request rate, or using more stressors in the load test. We will examine whether the correlation between the probe request rate and the capture rate follow the trend observed in the current study. This includes validating whether our speculation on the merging of the SS into the DS curve occurs if the stressor extends its emission rate beyond 350 probe requests per second; confirming whether the sensor's capture rate continues to drop at low probe request rates when more stressors are involved; and exploring whether the decrease in

capture rate, as observed in the DS curve, experiences a sudden change after some critical point.

Future work on system performance will involve four tracks. The first will focus on the de-randomization of virtual MAC addresses. This requires collection of additional information before MAC address hashing, such as vendor identifier. The second focus on geospatial analysis for device identification. By tracing each device's path over time, it becomes possible to match trajectories and locations, even with virtual addresses. Third, we plan to add positioning and velocity measurement. With an area covered by multiple sensors, if a device's probe requests are captured by at least two sensors, its position and velocity can be estimated. Finally, we plan to add support for Bluetooth probes. In combination with WiFi, we expect this to enable better device identification within a covered area.

VI. Acknowledgement

This research is supported by the City of West Palm Beach, the Knight Foundation, and the Community Foundations of Martin and Palm Beach Counties. The authors would like to thank Chris Roog, Director of Economic Development in the city of West Palm Beach, for his collaboration in realizing MobIntel.

References

[1] J. Scudellari, L. Staricco, and E. Vitale Brovarone, "Implementing the supermanzana approach in barcelona. critical issues at local and urban level," *Journal of Urban Design*, pp. 1–22, 2019.

[2] MotionLoft Inc. Motionloft - meaningful data from vehicle and people counts. [Online]. Available: https://motionloft.com/

[3] H.-H. Lee, C.-J. Lee, and C.-P. Lo, "Electronic device and method for managing traffic flow," Oct. 31 2013, uS Patent App. 13/854,168.

[4] M. Cuban, J. Reitman, and P. McAlpine, "Object detection sensors and systems," Jan. 31 2019, uS Patent App. 15/659,198.

[5] M. Cuban, J. Reitman, and P. McAlpine, "Object detection and tracking," Jan. 31 2019, uS Patent App. 15/659,239.

[6] M. Cuban, J. Reitman, and J. O. Pritchard, "Object detection and analysis via unmanned aerial vehicle," Jun. 5 2018, uS Patent 9,989,965.

[7] New York Times. (2019) San francisco bans facial recognition technology. [Online]. Available: https://www.nytimes.com/2019/05/14/us/facial-recognition-ban-san-francisco.html

[8] F. A. U. I-SENSE. (2020) Mobintel platform. [Online]. Available: https://www.mobintel.org

[9] E. Longo, A. E. Redondi, and M. Cesana, "Accurate occupancy estimation with wifi and bluetooth/ble packet capture," *Computer Networks*, vol. 163, p. 106876, 2019.

[10] A. E. Redondi and M. Cesana, "Building up knowledge through passive wifi probes," *Computer Communications*, vol. 117, pp. 1–12, 2018.

[11] L. Mikkelsen, R. Buchakchiev, T. Madsen, and H. P. Schwefel, "Public transport occupancy estimation using wlan probing," in *2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM)*. IEEE, 2016, pp. 302–308.

[12] X. Luo, W. J. O'Brien, and C. L. Julien, "Comparative evaluation of received signal-strength index (rssi) based indoor localization techniques for construction jobsites," *Advanced Engineering Informatics*, vol. 25, no. 2, pp. 355–363, 2011.

[13] E. Vattapparamban, B. S. Çiftler, I. Güvenç, K. Akkaya, and A. Kadri, "Indoor occupancy tracking in smart buildings using passive sniffing of probe requests," in *2016 IEEE International Conference on Communications Workshops (ICC)*. IEEE, 2016, pp. 38–44.

[14] Ö. Göçer, K. Göçer, B. Özcan, M. Bakovic, and M. F. Kıraç, "Pedestrian tracking in outdoor spaces of a suburban university campus for the investigation of occupancy patterns," *Sustainable cities and society*, vol. 45, pp. 131–142, 2019.

[15] S. Steyer, G. Tanzmeister, and D. Wollherr, "Object tracking based on evidential dynamic occupancy grids in urban environments," in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 1064–1070.

[16] M. Schulz, D. Wegemer, and M. Hollick. (2017) Nexmon: The c-based firmware patching framework. [Online]. Available: https://nexmon.org

[17] Geekworm. X750 - all for raspberry pi. [Online]. Available: http://raspberrypiwiki.com/index.php/X750

[18] MeanWell Web. Mean well rs-15-5. [Online]. Available: https://www.meanwell-web.com/en-gb/ac-dc-single-output-enclosed-power-supply-output-rs–15–5

[19] Sixfab. Raspberry pi 4g/lte hat kit. [Online]. Available: https://sixfab.com/product/raspberry-pi-4g-lte-shield-kit/

[20] Polycase. Wq-50 nema hinged enclosure. [Online]. Available: https://www.polycase.com/wq-50

[21] B. Dorsey. Sniff probes - plug-and-play bash script for sniffing 802.11 probes requests. [Online]. Available: https://github.com/brannondorsey/sniff-probes

[22] Amazon Web Services. AWS iot core endpoints and quotas. [Online]. Available: https://docs.aws.amazon.com/general/latest/gr/iot-core.html

[23] Scapy. [Online]. Available: https://scapy.net/

[24] J. Freudiger, "How talkative is your mobile device? an experimental study of wi-fi probe requests," in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, 2015, pp. 1–6.

[25] Amazon Web Services. AWS lambda limits. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/limits.html

[26] Amazon Web Services. AWS IoT core increases default limits for customers. [Online]. Available: https://aws.amazon.com/about-aws/whats-new/2018/03/iot-core-increases-default-limits/