WiSenseHub: Architecture to deploy a building-scale WiFi-sensing system

Pratyaksh Mundra[∨], Zhengyu Huang[‡], William Hunter[‡], Aditya Arun[‡], Dharmi Khadela†, Prachi Sinha†, Dinesh Bharadia[‡], and Roshan Ayyalasomayajula[†]
[∨] Vellore Institute of Technology
India
[†] University at Buffalo-SUNY, [‡] University of California, San Diego
USA

Abstract

The smart buildings of the future need to understand the movement and occupancy of the people in the environment. Using cameras to provide this context can be privacy-invasive. Alternatively, installing dedicated hardware to sense the environment can be cost-prohibitive and limit ubiquitous adoption. WiFi-based sensing has hence been championed to provide this building-scale sensing, as it allows for both privacy and is ubiquitously deployed in most buildings. However, industry-translatable research in this space has been challenging as no building-scale systems can provide WiFi sensing data. Consequently, many real-world challenges of deploying these sensing systems remain a mystery. To overcome this veil of mystery, we develop and open-source WiSenseHub, a building-scale WiFi-sensing system. We build our system on commercially available WiFi radios, deploy our backend services to collect data on infinitely scalable AWS cloud or a local server desktop, and build a front-end phone-based interface to collect diverse WiFi sensing data. We deployed multiple WiFi radios in our building and collected data for user devices for over 38 hours.

CCS Concepts

• **Information systems** → **Information integration**; *Location based services*; **Sensor networks**.

Keywords

Wi-Fi Localization, Wi-Fi channel, Data Open-Sourcing, Server

ACM MobiCom '24, November 18–22, 2024, Washington D.C., DC, USA

ACM ISBN 979-8-4007-0489-5/24/11…$15.00
https://doi.org/10.1145/3636534.3697313

ACM Reference Format:
Pratyaksh Mundra[∨], Zhengyu Huang[‡], William Hunter[‡], Aditya Arun[‡], Dharmi Khadela†, Prachi Sinha†, Dinesh Bharadia[‡], and Roshan Ayyalasomayajula[†]. 2024. WiSenseHub: Architecture to deploy a building-scale WiFi-sensing system. In *The 30th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '24), November 18–22, 2024, Washington D.C., DC, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3636534.3697313

1 Introduction

Ubiquitous Wi-Fi deployments have made Wi-Fi-based sensing very popular for GPS-denied environments, specifically for localization, navigation, and tracking. With the increasing deployment of indoor robotics and home automation, Wi-Fi-based sensing has become a leading research topic for autonomous systems [3, 9, 13]. There is still a huge gap between research and deployment of these Wi-Fi-based solutions, specifically due to the lack of an open-sourced data pipeline for streaming, processing, storage, and retrieval of Wi-Fi sensing data for user devices and robot devices that also ensure real-time deployment and is/are privacy-preserving.

In contrast to existing Wi-Fi sensing systems, most robotics sensor deployments like camera and LiDAR have systematized data management pipelines and established ROS (Robot Operating System) platforms, crucial for the reproducibility of novel sensing algorithms. While our past work developed a Wi-Fi sensing ROS wrapper [9] based on Nexmon-CSI[7], it does not accommodate a seamless Wi-Fi sensor data management pipeline. Unlike camera and LiDAR sensors which are ego sensors [1] that measure the environment of the user or robot device, Wi-Fi data for a given device (user device or robot) is collected at the servers connected to infrastructure-deployed Wi-Fi Access Points (APs). These third-party server systems that perform data management demand (a) a centralized data-management server system

[1]provide inside-out perspective for the robot, where the robot measures and estimates its surroundings

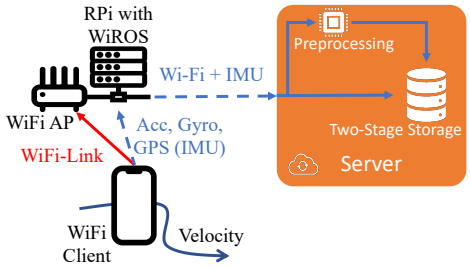

Figure 1: WiSenseHub's Architecture We design WiSenseHub to ensure that we can collect both Wi-Fi sensor data labeled device data: Gyroscope, Accelerometer, and GPS (labeled IMU)

including real-time data streaming and preprocessing compute (either on a cloud or local server), and (b) data privacy of the data pipeline for a user device.

Many commercial players, such as Cisco-Spaces [16] and Qualcomm's Atheros Boards, have developed Wi-Fi-based sensing platforms for data collection and development. Most of these platforms require proprietary hardware or subscriptions to their SaaS systems, which demands a need for open-sourced data aggregation for Wi-Fi sensing research and deployment. Many existing research-oriented open-sourced toolboxes aid in extracting Wi-Fi sensor data like PicoScenes [11], Nexmon-CSI[7], WiROS [1] and SWAN [22]. These platforms enable streaming data to an external port from the Wi-Fi hardware, but the data needs to be aggregated and managed by a third-party server for wider open-sourcing, research, and development. In WiSenseHub, we try to bridge this final gap in enabling scalable and deployable research platforms for Wi-Fi sensing. We develop an open-source framework[2] for scalable data collection across multiple infrastructure Wi-Fi access points and user devices and demonstrate it via an open-sourced Wi-Fi extraction toolbox [1] for the purposes of this paper. WiSenseHub's framework and WiROS's ROS wrapper are easily extendable to other open-sourced toolboxes like PicoScenes [11], SWAN [22], Intel 802.11n toolbox [8], or AXCSI [6].

WiSenseHub is a centralized server framework that integrates user device data and Wi-Fi sensor data for a building-scale infrastructure access point (AP) deployment. To achieve this we design a data pipeline system that can, for a given target user device:

(a) Collect Wi-Fi signatures of the device measured across multiple Wi-Fi Access Points at a single cloud or local platform.

(b) Collect the device ego-information like the accelerometer, gyroscope, and GPS/GNSS location data from the user device using the open-sourced SensorServer [17].

(c) Synchronize and anonymize the user and sensor data across the Wi-Fi infrastructure to create user self-correlated and privacy-compliant datasets.

To design a centralized server framework for data collection we retrofit COTS ASUS routers [2] with the open-sourced WiROS [9] platform that can publish Wi-Fi sensor data using ROS. Unfortunately, the ROS publisher-subscriber model is hard to scale for either cloud or local platforms. So, we create a ROS to MQTT wrapper that creates a lightweight MQTT publisher at each Wi-Fi router. We then design both an open-sourced local framework and an AWS cloud framework to create a dual-stage data storage platform that can collect data for each user or robot device at distinct timestamps across multiple Wi-Fi APs at each frame.

We need ground truth location labeled data for any Wi-Fi sensing application. While GPS data is not accurate indoors, one can perform GPS and motion-based dead-reckoning for accurate ground truth. So, we need to collect device-specific data that includes accelerometer, gyroscope, and GPS data for each instance of user or robot device. At the same time, this device information has to be synchronized with the Wi-Fi sensor data. So we modify the existing WiROS implementation to integrate the data coming from a WebSocket provided by an open-source Android App [17]. This integrated data is then streamed from one of the Wi-Fi APs via a Raspberry Pi, and the Wi-Fi APs that do not receive the data from the WebSocket stream only the Wi-Fi messages from WiROS. Additionally, we use a hash function on the device hardware MAC address of the user device to anonymize user identity but at the same point use it as a unique key to store and retrieve user or robot device data as required. In our system, the entire process of data collection is only initiated through the user or robot device using a Wi-Fi Ping. So, the user device or the robot is always in control of data flow to ensure user or robot device data privacy.

Finally, we deploy WiSenseHub using ASUS routers mounted with RPi-4 that run both the WebSocket and WiROS node. These modified WiROS nodes stream the Wi-Fi+device data and Wi-Fi data to a local LINUX server and also an AWS EC2 instance for cloud processing and storage. Utilizing this setup, we deploy 4 different Wi-Fi APs and test them across 2 different Android phones. We then test to ensure that the data collation is consistent over long periods and get an estimate of the average packet rate or frames-per-second (FPS) of data, which is about 20 received packets per second per AP for Wi-Fi data integrated across 4 routers. We also send the device's motion sensors and GPS location information used in a typical localization task. We estimate construction of a typical localization dataset would use 2.1GB/Hour/User of memory storing the data on our system and running a localization server for a building in the cloud would cost about 1.45 USD per hour with our system.

[2]Index Page for WiSenseHub: https://github.com/WS-UB/WiSenseHub

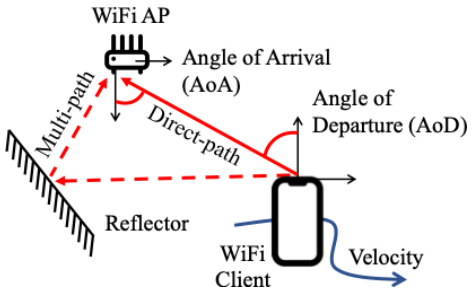

Figure 2: Signal parameters measurable by CSI The direct path's (solid line) and multipath's (dotted line) angles of arrival and departure can be measured in the local coordinates of the APs. The robot's velocity may also be measured.

2 Background

To understand the need for a distinct centralized server deployment for labeled Wi-Fi sensor data collection, we first give a quick overview of how most of the Wi-Fi sensing systems are designed. As shown in Figure 2, Wi-Fi sensor measurements that critically involve Wi-Fi connection information include: the Hardware MAC address of the connected device, the frequency of connection, channel of operation, bandwidth, MIMO settings (number of Tx, number of Rx, and number of streams), and quintessentially the received signal strength (RSSI) and the channel state information (CSI). RSSI and CSI are used to estimate the path parameters, like the Angle of Arrival (AoA), Angle of Departure (AoD), attenuation, and delay [3], for both the directed and reflected paths. These path parameters are then used for various localization and sensing tasks [13]. Critically, that vast set of state-of-the-art algorithms [1, 3, 12, 13, 18] uses angle-of-arrival (AoA) measurements to perform localization, tracking, and navigation estimation. These AoA measurements are estimated from the CSI measurements made at the Access Points (AP). The CSI measurement for a single packet can be up to 30kB, and the algorithms to compute AoA can require significant computation, meaning they are unfeasible to do at the edge. This motivates the need for a centralized, building-scale sensing pipeline.

Additionally, in contrast to camera and LiDAR sensors, which measure the environment on the ego (inside-out measurements) device that houses the sensor, Wi-Fi sensor measurements of a user or robot device must be made at the infrastructure deployed Wi-Fi routers, and typically require CSI measurements made across multiple APs, which makes a simple ROS wrapper like WiROS [13] difficult to use for building-scale deployments. While there are 10-100s of APs within a building the user or robot device is usually within range for only for 3-4 APs for Wi-Fi sensing measurements as long as the APs are within an enterprise infrastructure. To overcome, this we design WiSenseHub, a centralized server framework using open-sourced toolboxes that can easily scale to streaming data from multiple APs while being independent of which of the building APs are sensing the user or robot device data.

[3]absolute from 802.11mc [4] and relative from channel state information (CSI)

3 Design

We design WiSenseHub's framework to enable simultaneous, scalable, and privacy-preserving data collection. We create the data-management framework as shown in Figure 3 using both AWS and open-sourced toolboxes. Where the Wi-Fi and Phone data are recorded at the Raspberry Pi (RPI) connected to the Wi-Fi as described further in section 3.2, which is then streamed and stored in either the AWS cloud platform or a local server deployment described in section 3.1. Finally, we describe how the data is managed within the AWS server and the local server in section 3.3.

3.1 High-level Workflow

The primary motivation to build WiSenseHub is to democratize the deployment of large-scale WiFi sensing systems. In this vein, we open source all our backend server and user-facing frontend systems and build on inexpensive commercially available hardware. An end-end workflow of WiSenseHub is provided in fig. 3. The example workflow showcases N access points deployed in the environment. These access points monitor over-the-air WiFi packets originating from various clients. For each received packet, they compute the channel state information (CSI), which contains rich and fine-grained information about the interactions of the WiFi signal with the environment [9]. The CSI data is transmitted over Ethernet via a UDP socket to our cloud-connected Raspberry Pi's (RPI). We utilize openly available WiROS [9] platform to stream this CSI information in real-time. Finally, this data is uploaded to the cloud via an MQTT broker. We specifically use AWS's IOT core to ensure real-time and robust transmission of the WiFi CSI data to the cloud. This AP + RPI framework is repeated across multiple APs which can be deployed widely in any environment. To facilitate easy bring up and monitoring of this infrastructure, we again utilize AWS's IOTCore to onboard new RPIs, set up security keys, and monitor data upload across RPIs.

The APs [9] we deploy have the capability to measure CSI on any 802.11ac compliant WiFi packets. However, we find that for many scenarios, we need the transmission of dedicated WiFi packets at specific WiFi channels, bandwidth, MCS rate, or packet rate. Moreover, we need an efficient way to tag the packets arriving from multiple clients and associate the same packets received across multiple APs (as shown in fig. 3). To facilitate these requirements, we design an Android application to allow for the controlled transmission of WiFi packets and a real-time database table

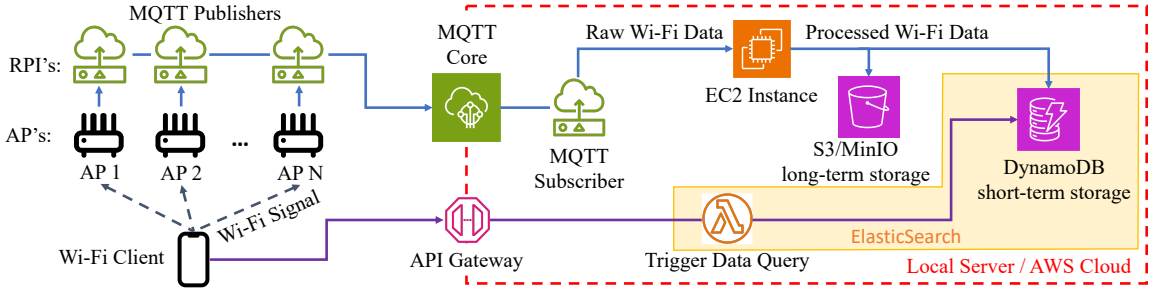

Figure 3: WiSenseHub's system block diagram: WiFi packets captured at the APs are offloaded to the Raspberry Pi's (RPI) connected at the edge. These RPIs push the data via the MQTT Core to the EC2 compute instance for additional processing. In our example, we compute the user device's location via triangulation. Raw and processed WiFi information is stored in S3 (MinIO) for long-term storage, and tabulated meta-data about the packets and user locations are stored in DynamoDB (ElasticSearch) for short-term storage and easy retrieval. API calls from the user devices can access this information readily.

joining routine on the cloud to create a cohesive dataset across multiple APs in real-time. In addition to this data upload pipeline, we develop a data download pipeline to query WiFi sensing information from the cloud to serve the end-user on the phone. We achieve this by developing a simple AWS Amplify API gateway and a serverless AWS Lambda trigger to query the stored data. Additional details of this front-end infrastructure are provided in section 3.2.

In the AWS cloud, we have three primary components. The first is the aforementioned Lambda instance to query and serve WiFi sensing data to the end user. The second is an Elastic Compute (EC2) instance to ingest WiFi sensing data from the MQTT broker and process it in real-time. And third, is a two-tiered storage system comprising DynamoDB and Simple Storage Service (S3) to efficiently categorize and store the WiFi data collected across multiple APs and clients. Details about these building blocks are provided in section 3.3.

3.2 User and Infrastructure Frontend Systems

We use the existing implementation of open-sourced WiROS platform [9] that collected CSI of all packets within a specific WiFi channel using *monitor* mode. We install WiROS across all the Raspberry Pis (RPIs) connected to the APs we have deployed. We also need the ground truth measurements from the device for each of the Wi-Fi sensor measurements. A simple way could be to send the data from the phone over the internet to the server directly, but since the ROS server on the RPIs and the AWS (local) server are not synchronized, the data association would be more troublesome. So, instead, if we can synchronize the Wi-Fi-sensor data and the IMU data before it enters the server over the IOTCore we would not have this issue.

So, we get the accelerometer, gyroscope, and GPS readings of the device at one of the RPIs over the internet. To implement this into our system, we use the SensorServer application [17][4]. This Android app allows us to stream real-time sensor data from the phone to multiple WebSocket clients, enabling them to monitor the device's movement, environment, and position in real time. Clients can connect to the app using a WebSocket URL to receive data from various sensors, such as the accelerometer and gyroscope, in JSON format. This data is then processed by a ROS node, implemented in Python, which subscribes to the WebSocket server and publishes the sensor data as ROS topic.

So now we have the new ROS node publishing the GPS, accelerometer, and gyroscope measurements and WiROS also publishes Wi-Fi messages using a ROS framework. We, then implement a ROS node which plays a crucial role in synchronizing IMU data with Wi-Fi messages. The node subscribes to Wi-Fi and IMU data topics and publishes synchronized data to designated ROS topics. Specifically, it listens to Wi-Fi messages on the `/csi` topic and ground truth device data on the `/imu` and `/gps` topics. Upon receiving new Wi-Fi data, the node updates its internal state and publishes the Wi-Fi message along with the most recent ground truth device data to the `/sync/csi`, `/sync/imu` and `/sync/gps` topics, respectively. These ROS topics are then subscribed to the MQTT publisher that delivers them to the server over the IOTCore API.

3.3 Backend Server Framework

In this subsection, we first describe the various components of the AWS cloud framework for WiSenseHub's framework in section 3.3.1-3 and then describe the open-sourced toolboxes we use as alternatives for local server deployment in section 3.3.4.

3.3.1 Lambda Data query: AWS Lambda is an event-driven computing service, which is primarily used to perform specific individual tasks. For WiSenseHub, it serves to retrieve the relevant information from the AWS storage. An example data retrieval task we have implemented is to query the

[4]https://github.com/umer0586/SensorServer

phone's location by utilizing WiFi-based localization and displaying this location on the map. In this case, the Android application utilizes Amplify's gateway to trigger the Lambda endpoint and requests location information for its user. The locations, computed in the cloud using the WiFi sensing information, are tagged with this user's unique application key, and the request is served again via Amplify.

3.3.2 EC2 Post Processing Server: The Lambda endpoint serves well to perform simple low-compute tasks. However, for more involved processes, we require a scalable compute resource. For this, we depend on AWS's Elastic Cloud Compute (EC2). The EC2 instance serves two primary purposes. One, it runs an instance of WiROS's processing node[5]. This processing node is responsible for taking the raw CSI information transmitted by the MQTT broker, calibrating and sanitizing the CSI data, and computing the AoA-ToF profile, which is often useful for various localization and WiFi sensing applications [13]. Second, it is responsible for storing the processed and raw WiFisensing information in the two-tiered storage system comprised of DynamoDB and S3. In addition, this server instance can be extended to deploy localization algorithms like SpotFi [12], MDTrack [21] or DLoc [3] to test the validity of these localization systems in building-scale scenarios.

3.3.3 Two-tiered Storage System: One of WiSenseHub's primary deliverables is to provide researchers with an open-source and easy-to-deploy WiFi sensing data collection solution for their research. An important part of this is to store vast amounts of this data in an inexpensive yet easily accessible manner. To balance between these two core properties we chose to develop a two-tiered data storage solution. DynamoDB is a serverless NoSQL database system that allows for quick storage and retrieval of ordered data. It is often best suited to track inventory lists for customers or store user data. A key aspect of these applications is although there may be millions of rows of data, each row contains limited information (inventory quantity, description, and unique ID).

However, each CSI data can vary between $8-30$ KB and any additional processed transformation stored (eg. AoA-ToF image) can double the row size. However, DynamoDB becomes cost-ineffective when storing large row sizes. On the other hand, we find that S3 (Simple Storage Solution) provides an inexpensive but harder-to-access data storage solution. Hence, to bring the best from both worlds, we store in DynamoDB the universal file links to the WiFi sensing data files stored in S3. This reduces the row's data size while also providing the quick lookup benefits of DynamoDB.

3.3.4 Local AWS deployment: To ensure the scalability of our system we deploy the server locally on a docker. We leverage dockers to integrate the framework, where we take data from multiple RPIs through MQTT. The MQTT core component is responsible for collecting the received messages through the publishers to the subscriber within the server. The server has a pre-processing Python pipeline that processes the Wi-Fi data and transfers the data to a long-term storage platform managed through MinIO [14]. We use MinIO as the local server alternative to the S3 buckets to organize the data and upload it to the bucket for further querying. MinIO is used to provide high-performance storage solutions for large volumes of streaming data.

For querying the WiFi and User device data we are using ElasticSearch for advanced querying and analysis, as an alternative to the DynamoDB and Lambda function-based querying. Instead of DynamoDB-based indexing, ElasticSearch is used to index and query the user data. Since both MinIO and ElasticSearch are free open-sourced toolboxes compatible with the AWS framework, the local server deployment can also scale to build scale data collection platform for Wi-Fi sensor data and the device data across multiple devices.

4 Evaluation

We now give a brief description of how the data looks like and evaluate the consistency of the data streaming capabilities of our server. We also provide the estimates in terms of USD for costs incurred by running the AWS server, and also the memory consumed for storage.

Table 1: Data Entries: Shows the data entries, their data types, memory occupied, and description of the parameter. This table is only for one AP_i's Wi-Fi measurements.

Variable Name	Data Type	Memory	Description
User Dev ID	String	17 B	Hashed ID (TX_{MAC})
Timestamp	Decimal	20-28 B	Unix Timestamp
Bucket Name	String	~ 20 B	S3 Bucket
File Name	String	~ 40 B	Binary File
Offset	Decimal	10 B	Offset in File
Sequence	Decimal	8-12 B	Packet Sequence
RSSI	Decimal	7 B	Signal Strength
Rx ID	String	16-20 B	Rx IP Addr
AP_i Channel	Decimal	5-8 B	Channel Number
AP_i Bandwidth	Decimal	4 B	MHz
Accelerometer	Float	24 B	Float Array
Gyroscope	Float	9 B	Float Array
GPS	Float	30 B	Float Array
AP_i CSI	Float	8-30 KB	Float Array
Total (no Floats)	N/A	$\leq$ 0.2 KB	Entry Size on DB

[5]https://github.com/ucsdwcsng/wiros_processing_node

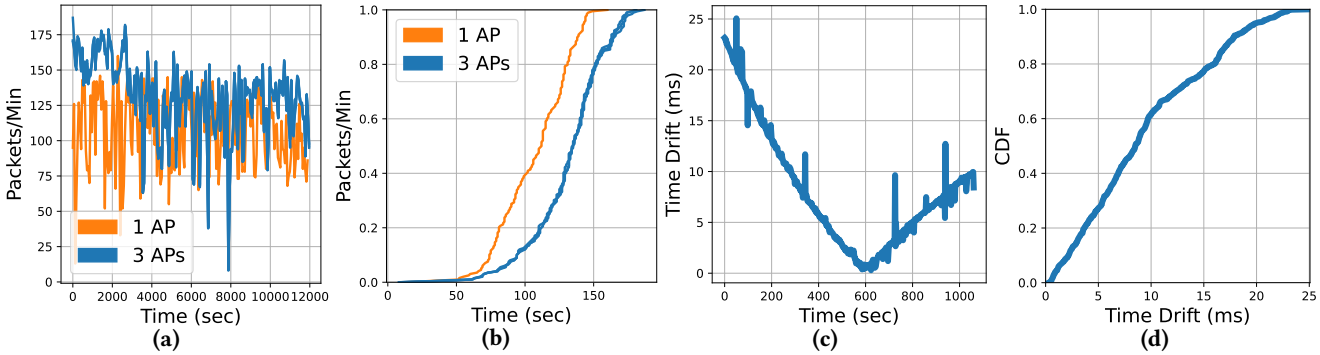

Figure 4: (a-b) Data-Consistency Evaluation: Shows the frames rates (FPS) into the server storage of data collection for the AWS and local server when collecting WiFi data across 1 AP and 3APs and plot the time-series of the FPS per AP and the CDFs of the FPS per AP. (c-d) Data-Synchronization Evaluation: Shows the time difference (deviation in seconds) for the arrival of the frames from different APs in both time series and CDF.

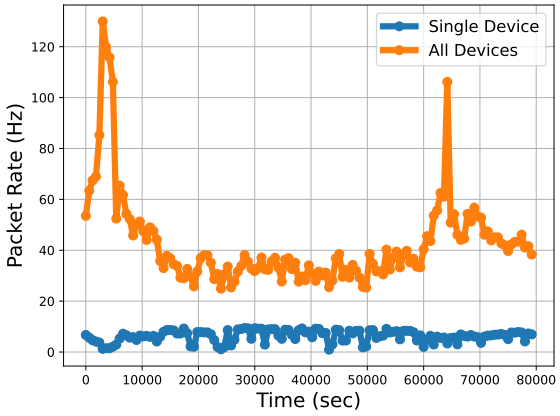

Figure 5: MAC Address Association: Shows how average FPS changes when the server blindly listens to all the packets, but using WiSenseHub's MAC address hashing we can still associate them irrespective of MAC randomization of the user device.

4.1 Table Entries

Firstly, in Table 1 we list out all the table entries received for each frame for a user device, the description of the data, data type, memory required for each entry, and description of the entry. These table entries are indexed using the Hashed Device ID and the timestamp of the data collected. The entries in Table 1 are only for one Wi-Fi AP in the environment (AP_i), and a typical Wi-Fi sensing algorithm requires data across 3 to 4 APs, and most of the time only 3 to 4 APs are within the range of the user device that can measure the device's Wi-Fi signals, and only one of the APs receiving the device data.

4.2 Building level Scalability

We further evaluate the consistency of our data collection platform as a test of the performance of our system, we have run the data collection pipeline on both AWS and the local server for multiple long-term runs to evaluate the synchronization of data across multiple APs, and steady rate data collection from multiple APs and the user devices data. In total, we have run the local and AWS servers for 40 Hrs. We first evaluate the claim of WiSenseHub being scalable to the building level. To demonstrate that we scale the number of APs simultaneously measuring Wi-Fi data. As mentioned earlier most of the algorithms need at least 3 APs and in most realistic deployments, the user device is in range for only 3 APs at best, we demonstrate the change in average FPS across 1 AP and 3 AP data collection scenarios as shown in Figure 4(a-b). To evaluate streaming consistency and scalability, we calculate the mean Frames/entries stored per second (FPS) per AP within the S3 storage bucket (MinIO storage). We can see here that irrespective of using 1 APs or 3APs the FPS per AP remains almost the same. This demonstrates that scaling to multiple APs across the building should not adversely affect the performance of WiSenseHub. We can also further see from Figure 4(b), where the average FPS does not change significantly, except due to localized drops from transient interference, demonstrating the scalability of WiSenseHub.

4.3 Data-Synchronization Consistency

To then evaluate data-synchronization consistency, we look at a few consecutive data samples across time for the same packet sent. We then measure the time difference between the minimum and maximum of the timestamps between the data samples coming from all the APs and the phone for a given user device's transmission. That is we are in essence measuring the time taken to get all the data corresponding to one data point within our dataset. We then plot the time series and CDF of the difference in time to measure all the data for one point as shown in Figure 4(c) and Figure 4(d) respectively. We can see from Figure 4(d) that on average, it takes about 9 msec to get all the data for one single data point. Further, from the time series in Figure 4(c), we can

see that it takes only 25 msec to collect one data point using WiSenseHub. This shows that the user is quasi-static for each data-sample making the data across each point consistent for a given user location.

4.4 User device Privacy Vs Track-ability

As discussed previously, since Wi-Fi data is being collected by a third party, we need to ensure that the user device is the one in control of disseminating the information. While the GPS and other device data is shared at the user device's discretion the Wi-Fi sensor data is collected by the Wi-Fi APs. So to enable user-centered data collection, WiSenseHub is also designed to ensure that data collection only happens when the user pings the router and not at all times. Further, we only use an anonymized hash of the user's MAC address to identify a user's device. To further verify that we extract a given user's data from the data collected across all the user devices and show the FPS as shown in Figure 5. Where we can see that WiROS can publish and the server ingests data up to 100 FPS on average per AP. In contrast, with WiSenseHub's filtering with the user device pinging at a constant rate of 5Hz, the data ingestion rate within the server is restricted to $10 - 20$ FPS depending on the amount of channel contention.

Table 2: Projected Costs: Shows the rate of memory dump you get from a single Wi-Fi AP @20 FPS and a single user device data @100 FPS. The costs are shown in terms of AWS S3 storage, and the amount of local memory consumed per hour of data collection.

Sensor Data	Bandwidth (KB/s)	AWS S3 Cost (USD/hr)	Local Memory per hour (B)
Wi-Fi (1APs)	$620kB/s$	0.05	$2.1GB$
User Device	$12.8kB/s$	0.0012	$46.0MB$

4.5 Memory and Costs

Finally, we also give an estimate of the costs incurred by this data collection framework provided by WiSenseHub in terms of both price and memory required. We give an estimate based on the CSI frames stored when we run WiSenseHub for 1 hour. We show a memory estimate for the data collected over 1 hour for a single Wi-Fi AP as shown in Table 2, which can be used to linearly scale the memory consumed during the data collection. We also estimate the memory consumed during the AWS data storage and convert that to the cost incurred in USD which is also shown in Table 2. Thus, in WiSenseHub we provide a rough order of magnitude (ROM) costs that anyone can use to estimate their costs before deploying WiSenseHub to scale for their building for Wi-Fi-based sensing algorithms for researchers and developers. Note that storage of the CSI is only necessary for evaluation or model training purposes and is not necessary for location inference.

5 Related Work

Wireless sensing researchers commonly rely on open-sourced datasets or COTS systems that furnish physical-layer WiFi channel state information. However, we find the datasets and systems, for example the Human activity recognition datasets [19, 23–26] or localization datasets [3, 5], in place today do not allow researchers to quickly bring up building-scale testbeds to conduct and verify their research.

To enable more freedom, existing WiFi sensing systems can be deployed instead of relying on open-sourced datasets. These systems can be categorized into those which provide non-real-time and real-time CSI information.

WiFi measurement tools: There are two widely used open-source WiFi measurement toolkits [8, 20] which support the IEEE 802.11n protocol. However, this protocol is outdated and consequently not as widely supported in current WiFi deployments. Alternatively, newer toolkits that leverage the IEEE 802.11ac [7] and IEEE 802.11ax [6, 11] exist. Unfortunately, these systems do not expose WiFi measurements in real time; instead, they store them on the device in specialized files. This requires additional post-processing and time-synchronization, precluding real-time system integration, which is necessary for debugging and building real-world systems.

ROS-supported Tools: To circumvent these challenges, a few tools integrate WiFi measurements into MQTT-based frameworks (Robot operating system, ROS, specifically). [15] provides only the WiFi signal strength (RSSI) measurements in real-time via an MQTT-topic, which can be used as a proxy between a transmitter and receiver. However, RSSI is a very coarse-grained, environment-sensitive measurement, whereas richer WiFi measurements exist that can expand the use of WiFi sensors. These richer measurements (channel state information, CSI) are exposed in the tools above and help determine the arrival and departure angles of a WiFi signal, the velocity of the WiFi sensor, or even fine-grained information about the environment [13]. Figure 2 details these physical measurements. A recent work [10] looked at providing real-time support for the 802.11n chipsets [8, 20] mentioned previously. However, this system requires active collaboration with the WiFi infrastructure, requiring deployed WiFi APs to perform 'round-robin' packet exchanges. This active involvement by the infrastructure creates additional overhead in the congested network and precludes ubiquitous deployment of WiFi sensing in indoor environments. Recently, WiROS [1] overcomes some challenges to develop a "monitor" based CSI acquisition framework. Their system is primarily focused on providing CSI at a single robot client. Instead, many infrastructure-centric sensors require CSI collection at multiple APs deployed in the building. WiROS, however, cannot be deployed at a building scale.

WiSenseHub primarily seeks to bridge the gap in research by open-sourcing an infrastructure framework, built on off-the-shelf hardware and software, to quickly, efficiently, and cost-effectively bring up building-scale test beds to help further the state-of-art in WiFi-based sensing.

6 Discussion and Conclusion

So, in WiSenseHub, we design and open-sourced our code bases [6] for creating local and AWS servers and also the web app extension for WiROS that can stream device data in tandem with Wi-Fi sensor data from across APs. WiSenseHub's framework is easily scalable to building scale of Wi-Fi APs, with the help of RPI-enabled ASUS routers. We deployed WiSenseHub on cloud and local servers to demonstrate the stability of WiSenseHub's data collection rate and ensure that the data collected is user-defined and device data secure.

References

[1] Aditya Arun, William Hunter, Roshan Ayyalasomayajula, and Dinesh Bharadia. 2024. WAIS: Leveraging WiFi for Resource-Efficient SLAM. In *Proceedings of the 22nd Annual International Conference on Mobile Systems, Applications and Services*. 561–574.

[2] ASUS. 2022. Asus RT-AC86U router. https://www.asus.com/us/networking-iot-servers/wifi-routers/asus-wifi-routers/rt-ac86u/, journal=ASUS USA.

[3] Roshan Ayyalasomayajula, Aditya Arun, Chenfeng Wu, Sanatan Sharma, Abhishek Rajkumar Sethi, Deepak Vasisht, and Dinesh Bharadia. 2020. Deep learning based wireless localization for indoor navigation. In *ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*. 1–14.

[4] IEEE Computer Society LAN/MAN Standards Committee et al. 2007. IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11ˆ* (2007).

[5] Nir Dvorecki, Ofer Bar-Shalom, Leor Banin, and Yuval Amizur. 2020. Intel Open Wi-Fi RTT Dataset. https://doi.org/10.21227/h5c2-5439

[6] Francesco Gringoli, Marco Cominelli, Alejandro Blanco, and Joerg Widmer. 2022. AX-CSI: Enabling CSI extraction on commercial 802.11 ax Wi-Fi platforms. In *Proceedings of the 15th ACM Workshop on Wireless Network Testbeds, Experimental evaluation & CHaracterization*. 46–53.

[7] Francesco Gringoli, Matthias Schulz, Jakob Link, and Matthias Hollick. 2019. Free your CSI: A channel state information extraction platform for modern Wi-Fi chipsets. In *Proceedings of the 13th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*. 21–28.

[8] Daniel Halperin, Wenjun Hu, Anmol Sheth, and David Wetherall. 2011. Tool release: Gathering 802.11 n traces with channel state information. *ACM SIGCOMM Computer Communication Review* 41, 1 (2011), 53–53.

[9] William Hunter, Aditya Arun, and Dinesh Bharadia. 2023. WiROS: WiFi sensing toolbox for robotics. *arXiv preprint arXiv:2305.13418* (2023).

[10] Ninad Jadhav, Weiying Wang, Diana Zhang, Swarun Kumar, and Stephanie Gil. 2022. Toolbox Release: A WiFi-Based Relative Bearing Framework for Robotics. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 13714–13721. https://doi.org/10.1109/IROS47612.2022.9981230

[11] Zhiping Jiang, Tom H Luan, Xincheng Ren, Dongtao Lv, Han Hao, Jing Wang, Kun Zhao, Wei Xi, Yueshen Xu, and Rui Li. 2021. Eliminating the barriers: demystifying Wi-Fi baseband design and introducing the PicoScenes Wi-Fi sensing platform. *IEEE Internet of Things Journal* 9, 6 (2021), 4476–4496.

[12] Manikanta Kotaru, Kiran Joshi, Dinesh Bharadia, and Sachin Katti. 2015. SpotFi: Decimeter level localization using Wi-Fi. In *ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. 269–282.

[13] Yongsen Ma, Gang Zhou, and Shuangquan Wang. 2019. WiFi sensing with channel state information: A survey. *ACM Computing Surveys (CSUR)* 52, 3 (2019), 1–36.

[14] minIO. 2024. minIO Object Storage. https://min.io/. Accessed: 2024-07-08.

[15] Eitan Marder-Eppstein Scott Hassan. 2010. WiFi DDWRT Tool. https://github.com/ros-drivers/wifi_ddwrt

[16] Cisco Spaces. 2023. Cisco Spaces: Turn your Buildings into Smart Spaces. https://www.cisco.com/c/en/us/solutions/collateral/enterprise-networks/at-a-glance-c45-741653.html?oid=aagen014817

[17] Umer Farooq. 2024. Sensor Server v6.3.1. https://github.com/umer0586/SensorServer. Accessed: 2024-07-08.

[18] Deepak Vasisht, Swarun Kumar, and Dina Katabi. 2016. {Decimeter-Level} localization with a single {WiFi} access point. In *13th USENIX symposium on networked systems design and implementation (NSDI 16)*. 165–178.

[19] Dazhuo Wang, Jianfei Yang, Wei Cui, Lihua Xie, and Sumei Sun. 2022. CAUTION: A Robust WiFi-based Human Authentication System via Few-shot Open-set Gait Recognition. *IEEE Internet of Things Journal* (2022).

[20] Yaxiong Xie, Zhenjiang Li, and Mo Li. 2015. Precise Power Delay Profiling with Commodity WiFi. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking* (Paris, France) *(MobiCom '15)*. ACM, New York, NY, USA, 53–64. https://doi.org/10.1145/2789168.2790124

[21] Yaxiong Xie, Jie Xiong, Mo Li, and Kyle Jamieson. 2019. mD-Track: Leveraging multi-dimensionality for passive indoor Wi-Fi tracking. In *ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*. 1–16.

[22] Yaxiong Xie, Yanbo Zhang, Jansen Christian Liando, and Mo Li. 2018. Swan: Stitched wi-fi antennas. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. 51–66.

[23] Jianfei Yang, Xinyan Chen, Han Zou, Dazhuo Wang, and Lihua Xie. 2022. AutoFi: Towards Automatic WiFi Human Sensing via Geometric Self-Supervised Learning. *arXiv preprint arXiv:2205.01629* (2022).

[24] Jianfei Yang, Xinyan Chen, Han Zou, Dazhuo Wang, Qianwen Xu, and Lihua Xie. 2022. Efficientfi: Towards large-scale lightweight wifi sensing via csi compression. *IEEE Internet of Things Journal* (2022).

[25] Siamak Yousefi, Hirokazu Narui, Sankalp Dayal, Stefano Ermon, and Shahrokh Valaee. 2017. A survey on behavior recognition using WiFi channel state information. *IEEE Communications Magazine* 55, 10 (2017), 98–104.

[26] Yi Zhang, Yue Zheng, Kun Qian, Guidong Zhang, Yunhao Liu, Chenshu Wu, and Zheng Yang. 2021. Widar3. 0: Zero-effort cross-domain gesture recognition with wi-fi. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

[6] https://github.com/WS-UB/WiSenseHub