Proteus: An Easily Managed Home-based Health Monitoring Infrastructure

Mengjing Liu, Mohammed Elbadry, Yindong Hua, Zongxing Xie, Suvab Baral, Isac Park, Fan Ye

***Abstract*—A data collection infrastructure is vital for generating sufficient amounts and diversity of data necessary for developing algorithms in home-based health monitoring. However, the manageability—deployment and operation efforts—of such an infrastructure has long been overlooked. Even a small size of a dozen homes may incur enormous manual efforts on the research team. In this paper, we present Proteus, an easily managed infrastructure designed to automate much of the work in deploying and operating such systems. We develop new components and combine with mature technologies to minimize the human efforts required. Proteus includes: *i)* scalable, continuous deployment, operation and update of devices with automatic bootstrapping; *ii)* automatic fault and error monitoring and recovery with watchdogs and LED feedback, and complementary edge and cloud storage backups; and *iii)* an easy-to-use data-agnostic pipeline for integrating new modalities. We demonstrate our system's robustness through different sets of experiments: 3 sensor nodes running for 24 days sending data (17.4 Mbps aggregate rate), 10 sensor nodes for 14 days (58 Mbps aggregate rate), and 32 emulated sensors (419.2 Mbps aggregate rate). All such experiments have data loss rates less than 1%. Further we reduce human efforts by 25-fold and code required for adding new data modality by 25-fold. We also share our experience and lessons learned during the design, development, and pilot deployment of Proteus. Our results show that Proteus is a promising solution for enabling research teams to effectively manage home-based health monitoring at small to medium sizes.**

***Index Terms*—Application Platform, Device Management, Connected Health, Home-based Health Monitoring, Data Collection Infrastructure.**

I. Introduction

Home-based health monitoring has the potential to revolutionize the way we manage our health [1], [2]. The continuous collection and analysis of multi-modal sensing data could lead to early and precise interventions and management of a broad spectrum of diseases [8], [9], [10]. However, the development of robust, generalizable algorithms and models critically depends on the amount and diversity of data from real homes, far beyond well controlled lab environments [11]. The lack of a *manageable* infrastructure that can easily collect such data from real homes is a fundamental barrier to practical home-based health monitoring [12].

Mengjing Liu, Mohammed Elbadry, Yindong Hua, Zongxing Xie, Suvab Baral, Isac Park, and Fan Ye are with the Department of Electrical and Computer Engineering, Stony Brook Univeristy, New York 11790, United States (e-mail: {mengjing.liu, Mohammed.Elbadry, yindong.hua, Suvab.Baral, Isac.Park, zongxing.xie, fan.ye}@stonybrook.edu.

This work is supported in part by NSF grants 1951880, 2119299.

The *manageability* in deploying and operating such an infrastructure is a great challenge, especially for most research teams of small sizes (e.g., one faculty member plus a few students). Significant manual efforts and special expertise are needed throughout all stages of the process: *i)* configuring sensor, edge devices in multiple homes, installing data collection and analytic software to create the testbed, and constantly updating the algorithms and models running on the sensor, edge and cloud upon improved software or changed health conditions; *ii)* automatic monitoring of the system status and performance, and timely detection, resolution of network, device faults and errors to ensure minimal data loss in 24/7 data collection; and *iii)* integrating heterogeneous sensing hardware, modalities with varying data formats and rates (e.g., from Kbps of activities to tens of Mbps of RF baseband) which may require separate and distinct engineering efforts for data streaming, processing, and storage [13].

While there exist a few research infrastructures for health monitoring purposes (e.g. [5], [14], [4], [15]), they do not consider manageability issues of intensive human efforts required in deployment (configuring, installing) and operation (updating, troubleshooting), and they support mainly traditional sensors (e.g. video, wearable devices, ambient sensors such as temperature and humidity) or sensors for specific diseases (e.g. breath sensor for asthma monitoring), not easy to extend and add new modalities. There are works (e.g. [3], [16]) focusing on clinical data/device management. While they effectively address the needs of managing clinical data, they do not consider the intensive management efforts of devices and codes at the edge/home and issues in the error-prone home environment (e.g. network outage, hardware failure) which can result in significant data loss. Existing data collection or IoT infrastructures from industry (e.g. KAA [6], Thingspeak [7]) support new modalities by data-agnostic transportation and storage utilizing MQTT and HTTPS protocols. However, they mostly focus on cloud-side features such as data transfer, storage and analysis. Intensive manual efforts for configuring sensor, edge devices, and frequent update of algorithms or models needed for home health-monitoring, are not addressed. Neither do they consider sufficiently the faults, errors that happen frequently (e.g. network outage, device faults) in home environments, causing serious data losses and manual efforts to remedy.

In this paper [1]. , we present Proteus, a manageability-focused data collection infrastructure that minimizes manual efforts so a small research team can efficiently deploy and operate longitudinal home-based health monitoring. To

[1]A portion of this work was published in ACM BCB'23[1] proceedings.

TABLE I: Comparison with existing work, including both IoT platforms (i.e., KAA[6], ThingSpeak[7]) and applications developed on top of IoT platforms (i.e., SMART-on-FHIR[3], Welcome[4]). Our proposed system (in cyan), Proteus, advances the state of the art (in both categories) by enabling a comprehensive set of functions as a whole to allow research teams to effectively manage home-based health monitoring.

System	SMART-on-FHIR[3]	**Welcome** [4]	**SPHERE** [5]	**KAA** [6]	**ThingSpeak** [7]	**Proteus**
In-home	✗	✓	✓	✓	✓	✓
Automatic Installation	✗	✗	✗	✗	✗	✓
Automatic Connectivity	✗	✗	✗	✗	✗	✓
OTA Deployment/Update	✗	✗	✓	✗	✗	✓
Edge Status Monitoring	✗	✗	✓	✓	✓	✓
Watchdog on Edge/Cloud	✗	✗	✗	✗	✗	✓
Complementary Edge Storage	✗	✗	✗	✗	✗	✓
Data-agnostic Pipeline	✓	✗	✓	✓	✓	✓
End-to-end Data Loss	—	—	—	—	—	$< 1\%$
End-to-end Latency	—	—	—	—	—	< 3.65 seconds
Reduction in Manufacturing Time	—	—	—	—	—	25-fold

[1] For cells with a '-', it means that the corresponding results are not provided in the respective paper.

minimize human efforts, we develop new design components, and identify, combine mature technology pieces, engineering practices to optimize the overall workflow, including: *i)* a scalable, continuous deployment and management pipeline with IoT device management solution (DMS) for remote, batch deployment and frequent update, automated node registration and networking setup upon first boot, and automated connection on edge; *ii)* automatic monitoring and watchdog mechanisms on edge for timely recovery from errors, and complementary edge storage backup for resilience against network failures; and *iii)* a data-agnostic pipeline that uses publish-subscribe (pub-sub) to transfer heterogeneous data formats to the cloud and store the data in database appropriately. In addition, we share our experience and lessons learned during its development, pilot deployment and evaluation, which are useful for improving the manageability of other similar infrastructures.

Preliminary experiences show that we can achieve 25-fold reduction in manual configuration of 7 sensor nodes from 420 minutes to 17 minutes. A small scale stress test shows that Proteus works reliably with 32 simulated sensor nodes, collecting data at an aggregate rate of 419.2 Mbps, with negligible data loss. The infrastructure we deploy in a one-bedroom, one-bathroom simulated home environment runs 3 real sensor nodes continuously for 40 days, and 10 real sensor nodes for 2 weeks without glitches. Despite many faults and errors (e.g. 6 sensor nodes running for 1 month, 2 network outages, 11 low data transfer rates, lasting 2 hours in total), the watchdog mechanisms can always quickly restore the normal operation, leading to minimal data loss ($< 1\%$). Results show that greatly improved manageability of Proteus enables small research teams to efficiently deploy and operate longitudinal, continuous data collection systems.

We summarize our contributions as follows:

- We identify three sources of intensive manual efforts in manageability challenges for deploying and operating longitudinal home-based health monitoring: *i)* configuring sensor nodes and edge servers, installing and updating their software and analytics; *ii)* detecting, resolving network and hardware faults and errors to ensure minimal data loss; *iii)* integrating new, unforeseen data modalities.
- We design new components and identify, combine mature techniques to minimize human efforts for greatly improved manageability for small research teams to deploy and operate such an infrastructure, including *i)* automated, continuous deployment, operation and update of the infrastructure with automatic self-bootstrapping; *ii)* automatic monitoring and recovery with watchdogs and complementary edge storage, requiring minimal manual efforts while ensuring minimal data loss; and *iii)* easy integration of new, unforeseen modalities with a data-agnostic pipeline.
- We have deployed the infrastructure in a simulated home environment. We find the manual configuring efforts are cut down by 25 fold, and lines of code for new modality integration is cut down by 25 fold. 10 real sensor nodes run continuously for 2 weeks without any errors, and 32 simulated nodes collecting data at an aggregate rate of 419.2 Mbps have negligible losses.
- We share our experience and lessons learned during the design, development, and pilot deployment of Proteus, which includes four aspects: *i)* data loss within sensor nodes before network delivery; *ii)* unexpected issues on sensor nodes due to imperfect hardware and software; *iii)* out of order data written into database; *iv)* Instructions for ADL data collection in a lab.

II. Background and Related Work

We identify existing techniques that are useful for cutting down manual efforts, including Enterprise Device Management Solutions (DMS) for remote, continuous deployment and management of codes and models at scale, containers and microservices to enable convenient duplicates on edge and cloud, and pub sub communication to support robust, asynchronous data transmission and easy integration of new modalities. Below we provide a brief description of these technologies that are critical for designing Proteus.

Enterprise Device Management Solutions. There exist enterprise solutions [17] for IoT device management and field deployment, like Amazon's AWS Greengrass IoT, Azure IoT device Management, which provide services like monitoring node status remotely with a cloud dashboard, pushing updates to remote nodes, containerized deployment of codes, version control, watchdog services for automatic quick-recovery, and etc. We leverage AWS Greengrass in Proteus for status monitoring and remote update of sensors and edge servers.

Containers and Microservices. Containerizing the code (e.g., docker) is a popular cloud technology that packages a full userspace environment with complex library, code dependencies so it can be accurately and easily duplicated in large quantities with simple scripts in forms of microservices. We leverage them to facilitate duplicate deployment of code and models at scale at the edge and cloud.

Pub-Sub Communication. A pub-sub transport provides robust asynchronous communication where data is cached and resent under intermittent connections, and flexibility to support data of different formats/content as opaque loads tagged with different topics. There exist mature solutions for constrained nodes at edge (e.g., MQTT [18]). With Quality of Service (QoS) 2, MQTT guarantees once and only once data delivery over network. An MQTT application typically consists of three main components: publishers, subscribers, and a broker. Publishers send data to the broker, which receives and stores the data, and then sends it to subscribers based on matching topic subscriptions. Additionally, there are pub sub microservices to share data among processes (e.g., Redis [19]). We leverage them for data-agnostic transferring pipeline in Proteus to reduce modality integration efforts.

Related Work. While these technologies help with manageability of the intended infrastructure, building and running such a system still requires significant efforts in an error-prone environment with imperfect embedded devices. There is not enough discussion and experience sharing about minimizing manual efforts for manageability of such systems. We categorize and compare the related work in Table I.

Home-based data collection infrastructure has been created, but not focused on manageability. Spatial Data infrastructures emphasize large scale data integration and search from existing geospatial resources without considering reducing manual efforts of the research team [20]. Advantech, in collaboration with Microsoft, built a WISE-Cloud platform that integrates IoT software and a cloud platform to provide services to industries, e.g., seamless sensor information collection, remote management of devices, and big data analytics, which forms an enterprise solution not suitable for small teams [21]. VitalCore [16] has developed an analytics and support dashboard and eliminated the tall pipeline of reading HL7 format health data. Multi-modal sensor infrastructure [5] supports various sets of data sent to the cloud, but does not handle edge computing, or maintaining units in the field. [3] introduces a generalized multi-site and multi-modality cloud infrastructure for clinical data management which establishes a standardized and secure data repository along with a user-friendly graphical user interface. However, it is important to note that this work is not specifically designed for home-based health monitoring, thus intensive human efforts for installing, monitoring devices and remedy on failures (e.g. network outage, hardware failures) which can easily occur in home environments are not considered. Furthermore, the paper does not provide a quantitative estimation of the infrastructure's robustness and efficiency. WELCOME [4] proposes a integrated care platform using wearable sensing and smart cloud computing to monitor and manage patient's conditions. But they don't consider the intensive management efforts of such a big, complex infrastructure in home environments. Many such data collection works could use Proteus to reduce manual efforts for greatly improved manageability (e.g., Parkinson data collection [22]).

There are also existing IoT platforms from industry (KAA [6], Thingspeak [7]), which support multi-modal data transportation with MQTT or HTTPS protocols, and various data analysis and visualization tools on cloud. However, they do not consider manual efforts in configuring, installing individual sensor, edge devices, and detecting, troubleshooting for faults, errors (e.g. network outage, device faults) in edge environments, which are fundamental barriers for small research teams to conduct longitudinal home-based health monitoring.

III. System Design

A. Goals and Assumptions

1) Goals: The goals of our infrastructure design are to minimize manual efforts of a research team in different stages: *i)* deploying the infrastructure for initial setup and continuous operating for data collection and analysis. Even a dozen homes may incur significant efforts on a research team in installing, configuring sensors, edge devices and their software, frequent updating of their algorithms and models, and continuous monitoring of the system performance; *ii)* detecting network, device faults and errors, and recovering timely to facilitate longtime, continuous running with minimum data loss. Hardware sensors, embedded systems, and home networks are generally imperfect; unforeseeable crashes, faults and errors always occur in the field; and *iii)* coding work on the pipeline to integrate multiple, possibly unforeseen types of sensors (e.g., base band radio data) for health monitoring needs.

2) Assumptions: We make the following assumptions: *i)* there exists sufficient wireless connection to the Internet at homes to handle the data throughput; *ii)* most of the data to be collected are time series and events (e.g., vital signs, activities with timestamps) that can be stored in relational and time series databases (e.g., MySQL, InfluxDB); and *iii)* for certain sensitive data, the user may prefer storage, processing locally on edge servers, not cloud.

B. System Overview

In this section, we first describe the 3-layer framework of Proteus including the data path and control path of the pipeline. We then provide a comprehensive description of the efforts required to deploy and operate the infrastructure, and our design to minimize such efforts.

Figure 1 shows the three-layer structure of Proteus and it data/control paths. In each home (i.e., edge), embedded systems (e.g., Raspberry Pi's) gather data directly from connected sensors (e.g., Ultra-Wide Band (UWB) radios, or Neulog sensors via USB), and send data through the home WiFi to an edge server. The edge server, a data transfer gateway, aggregates data from all sensor nodes in one home, pushes data to the cloud. It temporarily stores the data in a local database when the Internet is disconnected, and resends new incoming data upon reconnection. Sensitive data not allowed to leave the home is also stored locally. Further,

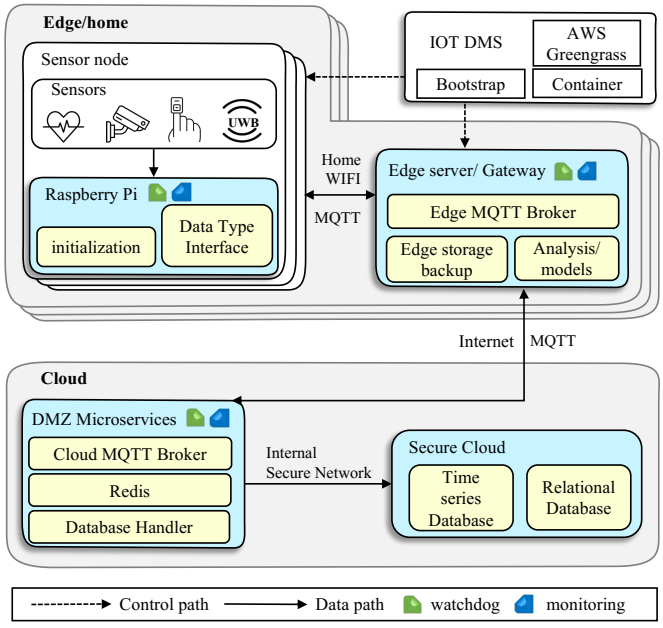


Fig. 1: Proteus Design Overview. In each home, there are multiple Raspberry Pi's connected to sensors. They use MQTT to publish readings to an Edge Server Gateway, which then aggregates the data and sends it to the cloud MQTT Broker which stores it in databases appropriately. Both edge and cloud have watchdogs, cloud monitoring and OTA (Over the Air) update capabilities.

analytic algorithms and machine learning models can be deployed on the edge server, enabling edge processing whenever needed. After data is sent to a cloud MQTT broker in a DMZ (Demilitarized Zone facing public Internet and blocking potential malicious traffic), some microservices receive and push the data to appropriate databases (either time-series or relational) in the secure cloud behind DMZ. The control path consists of two parts: *i)* IoT DMS to remotely deploy and update containerized codes and models on edge, sensor nodes, including an automatic bootstrap mechanism to minimize configuration efforts (e.g. automatic network configuration on boot). *ii)* System-wide monitoring and watchdogs to monitor system status and performance to ensure timely detection and recovery from faults, errors (e.g. network outage).

We divide our design into three parts: *i) Automated and Continuous Deployment* to minimize human efforts in the setup of the infrastructure, which enables large-scale deployment for small research teams; *ii) Maintenance* to monitor system performance, detect and restore from unexpected faults, error automatically, which ensures the system can operate seamlessly over an extended period, even in error-prone environments; and *iii) Data-agnostic Pipeline* to support easy integration of new modalities, allowing for the collection of multi-modal data to meet comprehensive research needs. We roughly estimate the amount of work required for each part in minutes or LOC, which serves a ballpark figure but not accurate description.

C. Automated and Continuous Deployment

The deployment of the infrastructure can be divided into three phases: *i) Installation* which entails installing and configuring each component (operating ssytem, software and hardware on sensor/edge, and software on the cloud); *ii) Connectivity* where the sensor nodes communicate with the edge server and with the cloud automatically with little manual intervention. *iii) OTA Deployment and Update* of algorithms and models on edge upon improved software or changed health conditions. Below, we describe each stage in detail and qualitatively analyze the manpower required for each part, and reduction from our design.

1) Installation: Manually installing operating systems and software (e.g. Docker for containerized code, software for LED control and WiFi configuration, etc.), or setting up the environment for an algorithm (especially a deep learning model) for embedded systems, are all laborious. We estimate that it takes an expert familiar with embedded systems 10 minutes to set up a sensor node. Instead of repeating such manual effort for every single sensor node, we duplicate them based on a *"prep"* master image with all the needed packages using a flash cloning machine, which costs only 3 minutes to duplicate 7 units in a batch. When the number of sensor nodes scales up, the time for packaging one unit is amortized from 10 minutes to $\frac{3}{7}$ minutes, a 23-fold reduction.

2) Connectivity: After installation, we need to configure the network settings of sensor devices so they can find the IP of edge server and establish pub/sub connections for data transmission. We reduce 5 times on configuration time by *i)* Using an automated WiFi set up app for the embedded systems of sensor devices to avoid manual IP configuration; *ii)* Dynamic Edge Server Connection where sensor nodes find edge server IP automatically and dynamically by querying a backend database.

WiFi Setup. Setting up WiFi connection for embedded systems (e.g., Raspberry Pi's) in the deployment time is cumbersome, because they typically come without input/output devices that allow for direct interaction. To ease WiFi setup, the *"prep"* image is configured to open a WiFi portal, so that we can easily use a mobile app to connect to the WiFi portal and send over the WiFi identifier and password to embedded systems of sensor nodes. The edge server is usually a laptop or desktop with a GUI and keyboard, and only one edge server is required per home, thus the effort to configure network on the edge server is negligible.

We estimate manually connecting each sensor node takes 5 minutes, and with our utility it takes 1 minute, thus reducing setup time on 10 nodes from 50 minutes to 10 minutes (5-fold reduction).

Dynamic Edge Server Connection. To establish pub sub connection between the sensor nodes and edge server, the only information needed is the IP address of the broker, which is located in the edge server. Manually configuring IP addresses is labor-intensive and impractical due to frequent IP address changes, while static IP maintenance is difficult to scale. We eliminate such manual effort with a dynamic IP discovery mechanism. To achieve that, we create a lookup table in the cloud relational database to associate the edge server of each home with its MAC address, which is generally considered to be static. To establish pub sub connection, the sensor nodes only need to query the relational database to get IP address of the broker in the corresponding home. It takes 1 minute to add an entry in the cloud relational database to associate the edge

server of one home with its MAC address. In addition, we configure the home ID of the home where the sensor is deployed in the *"prep"* image for record, which takes 1 minute per home.

We estimate manually entering the IP address for each sensor node takes 1 minute, thus 10 minutes for 10 nodes. However, when the IP address changes, we have to be physically present in that home to enter the new IP address; the travel to and back from that home can take minutes or even hours, a prohibitively high cost that we avoid by automatic discovery. By combining the easy setup utility and dynamic edge server connection, 10 sensor nodes in one home can be connected within 12 minutes (10 minutes for setting up WiFi connections for 10 nodes, 1 minute for configuring home ID in "prep" image, 1 minute for recording edge server MAC address in relational database). Thus the reduction easily approaches two orders of magnitude.

3) OTA deployment and Update: Deploying code (e.g., algorithms and models) across tens to hundreds of sensors and edge devices is labor-intensive. The workload is compounded by constant code update on sensors and edge upon bug fixes, software improvements, or changing health conditions. We estimate it takes 2-3 minutes for an expert to manually deploy or update code on a sensor node (including remote login, downloading and running the code). A mere 10 such updates on 10 nodes would easily take away 3-5 hours from an expert who could focus on more productive work.

To alleviate such manual efforts, we build on enterprise DMS (e.g., AWS Greengrass) which has a dashboard to easily deploy and update code on many sensor nodes and edge servers via a few clicks. Deployment or one OTA update on 10 sensor nodes can be finished in 1 minute (30-fold reduction).

Self-Registration However, for a sensor device to be managed by a DMS, it needs to register and obtain its unique ID from the DMS. Only after registration can we use the DMS to deploy and update their software in the field (e.g., deploying new models). We estimate it can take up to 3 minutes for an expert to register a sensor device on the DMS.

We develop an automatic self-registration script, where upon the first boot, the script generates a unique ID for the edge device, registers the device to the DMS, and creates a record in our relational database (including its unique ID and the home ID) for subsequent tracking with the home ID stored in the *"prep"* image beforehand. Then the self-registration of dozens nodes in one home can be done in parallel automatically on first boot without any manual efforts, instead of 3 minutes per unit sequentially by manual efforts.

With our self-registration on boot and OTA update with enterprise DMS, we can deploy and update code for 10 times on many sensor nodes in parallel within 11 minutes (1 minute to deploy code, 10 minutes to update code with DMS for 10 times) (30-fold reduction).

D. Maintenance

Our system has multiple embedded devices and multi-hop network transport. Many problems (e.g. network interruption, accidental unplugging of sensors, hardware failures) may occur in a home environment, resulting in critical data loss. Continuous monitoring of system status and timely recovery on faults and errors are mandatory for seamless operation of the infrastructure. Although software such as AWS Greengrass and PM2 provides dashboard for remote monitoring of device status and watchdogs to restart programs if they exit abnormally, they do not provide detailed per-device performance metrics (e.g. data transfer rate per hop) which are significant for gaining more insights of system performance and troubleshooting. Furthermore, when such application level restart cannot bring the system back, more action is required. Many issues caused by buggy hardware, software (e.g., some Pi nodes have glitches in network adaptors and/or stacks) can be solved by rebooting the device. But manual watching for such problems is impossible and timely rebooting must be automated.

Thus, we design watchdogs combined with LED chipsets for automatic monitoring and recovery to detect, notify (with different LED lighting modes) and remedy on *i) network issues* and *ii) hardware issues* in time to ensure seamless operation of the infrastructure.

Network Issues. We have a local database on the edge server to temporarily store data when Internet is disconnected. However, there are network issues on sensor nodes that can cause data loss before data is transferred to the edge server, requiring extra efforts to operate correctly.

Considering network status on sensor nodes is not stable and can cause critical data loss within sensor, it is important to monitor if abnormal network issues (e.g. network outage, low data transfer rates) happen. Thus, we identify two key metrics to monitor sensor nodes: memory usage and publisher queue size over time, which can reflect data transfer rates and reveal abnormal network issues occurring (see details in Section IV).

Due to the resiliency of MQTT pub sub transfers, we only address network issues if they persist for an extended period of time or cause problems that require urgent intervention (e.g. memory is almost exhausted due to low data transfer rates). We monitor the network connection, notify users with an LED light pattern (blue light) and reboot the device to recover if the network outage lasts longer than a threshold. For low data transfer rates, we monitor the publisher queue size and reboot the device when publisher queue reaches the maximum limit (configured corresponding to memory limit) and data transfer rate remains low within a time window.

Hardware Failures. Multiple hardware failures (sensors being unplugged, power supply failure, file system corruption and edge server going offline) can happen on edge, requiring mandatory human operations. We integrate the embedded system with an LED chipset whose lighting modes can indicate the sensor states, aiding non-expert users and the research team to conduct troubleshoot and remedy on errors without physical travel. The states and corresponding recoveries are *i)* Green: the sensor device is running normally, no operation required; *ii)* Red: the embedded system does not detect the connected sensor and the sensor needs to be re-plugged. *iii)* Flashing: edge server going offline. We indicate different reasons of edge server going offline with different flashing colors for details: *iii-i)* Yellow flashing: the sensor node can not discover the edge server MAC address, requiring to record edge server MAC in relational database. *iii-ii)* Blue flashing: the sensor node can not discover the edge server IP address, requiring to connect

edge server to home WiFi or reboot it if it is unresponsive. It is possible that edge servers (e.g. laptops or desktops) require rebooting after long time continuous running. *iii-iii)* Red flashing: the sensor node can not connect to the broker on the edge server, requiring to reboot the edge server. *iv)* LED is off: file system corruption, requiring OS image reflash for recovery. *v)* A power light off on sensor nodes indicates a power failure and requires the power supply to be re-plugged.

More events can be supported by additional lighting patterns of the LEDs. Compared to having to read logs to analyze the problem, which is cumbersome, and feasible only by the research team, LED feedback offers an easy, direct visual means for a few most common problems, allowing even non-expert users to help the research team troubleshoot remotely (e.g., via phone calls).

Combined with our monitoring (metrics and LED feedback) and watchdog mechanism, we eliminate the operational time required for seamlessly recovery from errors (e.g. 5 minutes reading logs to identify and act on each error). We present more description on unexpected issues on sensor nodes in Section IV.

E. Data-agnostic pipeline

The utilization of heterogeneous sensing devices and modalities is essential to meet comprehensive research needs and effectively monitor home health. However, it is non-trivial to integrate into the infrastructure the heterogeneous sensing modalities, which are typically of varying data rates. While data transmission over the network independent of the data type can be easily achieved with pub sub transportation, extra effort is needed to deal with storage of data from heterogeneous sensing modalities to facilitate subsequent data analysis. We design *interfaces for data types* to minimize the required coding effort.

Interface for data types. When adding a new sensor modality, the developer will need to update code for streaming data from the sensor (according to respective libraries), and passing the opaque, tagged data in the format they wish to store in the database. Two categories of metadata are defined to accommodate the new data type: *i) database type and name*, which instructs the pipeline the type of the database (i.e. time-series, relational) and name of database to store data within; *ii)* tags (i.e., topics) to store each respective field (e.g. "*baseband_timestamp*" indicates two tags to store baseband data field and timestamp field of when the data is generated) appropriately within the database. We use the database tags as topics in pub sub transportation, so only microservices subscribing on those topics for respective databases receive the corresponding data.

With our data-agnostic pipeline, only 4-5 LOC are needed to parse the data from the sensor (2-3 LOC with appropriate libraries) and pass it to our API (2 LOC, with one defining metadata and the other calling the API).

To summarize, Proteus minimizes the manual efforts imposed on small research teams to deploy and operate home-based data collection infrastructure. To set up, the team only needs a couple steps. These include: 1) configuring the home ID in a "prep" master image, registering the home ID and MAC address of the edge server in the cloud; 2) setting up sensor nodes and edge servers using the "prep" image, and connecting the sensor nodes (using a mobile app) and edge servers to the home WiFi network.

To incorporate a new modality, one uses an existing or develops a new driver to stream data from the sensor to the embedded system, then defines a data type interface for the MQTT pub/sub transport and database storage on the edge and cloud. These manageability features automate lots of manual labor, saving huge amounts of efforts for small research teams in the long run.

For home residents, the only effort required is simple actions (e.g., power off then on to reboot) to address issues based on the LED light patterns, or notify the research team of the patterns, all straightforward for non-experts. Although some manual intervention by the team may be indispensable, we empower non-expert residents to resolve most common faults (e.g., network outages, memory leak thus "dead" sensor nodes) with simple actions. This relieves researchers from the mundane yet constant burden of resolving frequent problems in home environments with error-prone embedded systems.

IV. Experience and Lessons Learned

We present experiences and lessons during the design, development, and pilot deployment of the infrastructure, and hope they will gain attention from the community and inspire future research on continuous and longitudinal home health monitoring data collection.

Data loss within sensor nodes before network delivery. Initially, we anticipated no data loss with MQTT that guarantees once and exactly once data delivery through the network at QOS level 2, based on a four way handshake protocol [23]. Our observations were found to contradicting to what was initially hypothesized. Such contradicting phenomenon happen when data publishing fails due to network issues, during which the data will be kept in the publisher queue in the sensor node memory, waiting to be re-published until the publishing (i.e., four-way handshake) finishes successfully. While new data keeps coming in, data piles up in the queue. Once the maximum queue size is reached, new data cannot be appended in the queue and are dropped. Thus data loss still happens inside the node before data is sent over the network. Without any limit on the queue size, in extreme cases (e.g., extended network outage), we observed the node could run out of memory. A size limit on the publisher queue is needed to avoid it consuming all the memory. Our lesson is that data loss must be considered end-to-end, not only over networks.

To prevent data loss in such case, we design two distinct solutions on the sensor node and edge server based on their respective characteristics. On the edge server, the publisher queue accumulates primarily due to poor network connectivity to the cloud. Thus, we design edge database as backup. In contrast, the primary cause of queue buildup on sensor nodes is the low data transfer rate due to imperfect hardware and software, and the only solution is to restart the Pi node. Detailed solutions for each are provided below respectively.

On the edge server, we have a local database for temporary storage. When the data upload to the cloud fails, data will be stored on the local database, and resent the new coming data when the network resumes. We use the publisher queue size to indicate the status of the network transfer from the edge server to the cloud. The smaller the real time publisher queue size, the better the network transmission status. Let Q denote the maximum publisher queue size. The real time publisher queue size is q. We empirically set two thresholds x=0.3 and y=0.8, meeting the inequality expression $0 < x < y < 1$. If q reaches $Q \times y$, we write new data to local database, while keeping sending data already in the queue until q drops under $Q \times x$. Thus the queue size will not continue to grow once the higher threshold is reached. If $q < Q \times x$, we stop writing new data to the local database, and append them only to the queue for publishing. This continues as long as q fluctuates between the two thresholds. This design ensures queue size never exceeds $Q \times y$, and once reaching $Q \times y$, it will continue to drain until to a lower mark $Q \times x$.

On sensor nodes, we observed persistent low data transfer rate caused by some glitches or defects in the Pi node network adaptor and stack. When this happens, data drains much slower than it comes in, and eventually the queue will be filled up, and the only solution is to reboot the Pi node. We set up a watchdog that monitors the data transfer rate when the publisher queue is full. If the data transfer rate is consistently less than a threshold in a 10-min time window, the watchdog reboots the sensor node. If the data rate is high enough, the queue will drain gradually thus no need for rebooting. During rebooting (about 40 seconds), data may be lost briefly before it is again inserted into the queue.

Through our methods, we minimize data loss within sensor nodes and edge servers before network delivery.

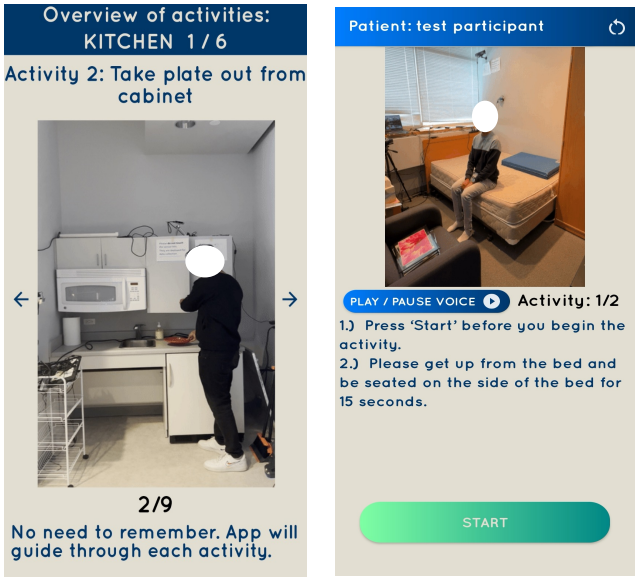
Unexpected Issues on Sensor Nodes. We observe some issues on sensor nodes especially with Raspberry Pi's. We hope that these experiences will save efforts for future researchers dealing with commodity hardware. *i)* When the WiFi router is disconnected for a period of time and then restored, the Raspberry Pi cannot automatically reconnect to the WiFi. The OS "believes" that the network is unreachable after a period of disconnection. We have to reboot the operating system so it can reconnect to the network. *ii)* Raspberry Pis' hardware quality can be really inconsistent. Some are good, Some can have questionable hardware. Watchdogs are necessary to ensure their operations are normal. *iii)* Network throughput on a Raspberry Pi can be erratic. It sometimes gets very low and requires a reboot to recover. The low rate can have several causes such as high CPU/memory load, network congestion, low disk space, file system corruption, etc. Glitches or defects in hardware and operating systems are also possible causes. It is important to prepare for such unexpected faults happening where normally we would not expect, and have extra mechanisms (e.g., watchdogs) for recovery.

Out of Order Data Write into Database. On the cloud, we initially used a thread pool to write sensor data into InfluxDB database. Each thread is free to take and write any data received by the cloud broker and transferred by Redis. We found that that data written into the database can be out of order. With 2 million data frames, about 7% in a test were out of order. In the case of time series health monitoring, such out of order data frames can cause incorrect health interpretation and must be reordered. What's more, the reading and analysis may happen many times. Thus lot resorting overhead is paid reading large amount of data, and it is better to ensure correct order in writing. As we dig deeper, we found that more than one parallel thread might be writing data from the same sensor, causing out of order writing. We modified the thread scheduling such that at most one thread handles the writing of data from each sensor. Experiments showed this solved the problem. We also found that the data received by the broker could be out of order, but the probability was really low (among 5.8 million frames, only 16 out of order, less than 0.001%). Thus, we consider this harmless and negligible.

Instructions for ADL data collection in a lab. Conducting self-assessment at home can significantly reduce the need for patients or elderly individuals to frequently visit the hospital. One effective way to gauge a patient's health status is by evaluating their performance in various Activities of Daily Living (ADLs), which can provide insights into their muscle strength, body coordination, and more. Activities of Daily Living reflect the body strength, coordination and can provide insights on the health status of people. As a use case of Proteus, we have designed an IRB protocol for patients to conduct a series of guided activities in a simulated home environment. Given the physical, cognitive challenges patient may face, we have developed a mobile app (Figure 2) to guide them through ADLs, while reducing cognitive strain. We work with users with different health conditions to get their feedback on cognitive differences and design the app to include: *i)* Room-by-Room Guidance: Activities are organized on a room-by-room basis. Users are given instructions for all activities within one room before being guided to the next, minimizing the need to move back and forth between rooms. *ii)* Overview Page: Each room has an overview page that provides a brief outline of the activities to be completed there, giving users a general idea of what to expect. *iii)* Detailed Instructions: For each activity, the app displays written descriptions on the screen, complemented by video and audio instructions to enhance understanding and ease of use. This app is intended as a proof of concept for self guided data collection, and we plan to generalize and further extend it to accommodate other use cases for data collection of different populations, in both simulated and real home environments.

V. EVALUATION

We implement our infrastructure's cloud components on a HIPPA-compliant data center using standard virtual machines running Red Hat Enterprise Linux 8.4.1. For each sensor node, we use a commodity Raspberry Pi 3B+ connecting a Ultra-Wide Band (UWB) radio sensor (5.8 Mbps data rate of baseband frames) and a LED chipset. We take gaming laptops (Intel Core i7, 16GB RAM, 1 GPU, 512 GB hard drive) as edge servers. We deploy one MQTT broker on the edge server to aggregate data from all sensors per home, and another in cloud for all homes. We leverage AWS Greengrass to control



(a) Overview page of all activities in kitchen.

(b) Instruction of one activity in kitchen.

Fig. 2: Screenshots of the APP. For each activity, we provide written descriptions on the screen, complemented by illustrative video and audio instructions so participants can easily follow. For the video instructions, we pre-record videos as examples of the corresponding activities to guide participants and reduce potential cognitive stress.

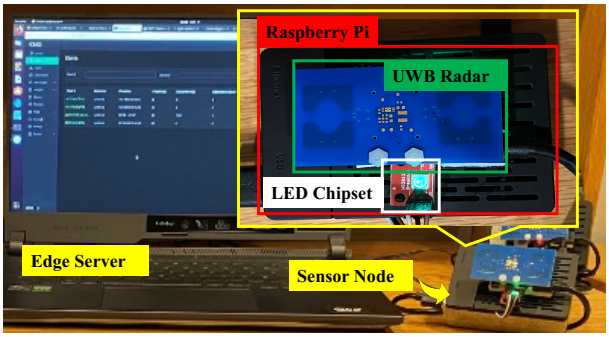


Fig. 3: Each home has a gaming laptop based edge server and multiple sensor nodes each consisting of a Raspberry pi, UWB sensor and LED chipset. The Raspberry Pi reads data from the UWB sensor and publishes it to an edge server using MQTT pub-sub. The LED chipset indicates the sensor states through different lighting modes, helping both non-expert users and the research team to troubleshoot and resolve issues remotely, without the need for physical presence. The edge server collects data from all sensor nodes and then forwards it to the cloud server by MQTT.

the sensor/edge devices and push containerized updates to them from a cloud dashboard. We deploy InfluxDB as our time-series database because of its fast read (427 Mbps, 74 times faster than sensor data rate) and write (348 Mbps, 60 times faster than sensor data rate) speeds in our preliminary tests on the laptop. We configure our sensor nodes and edge server (Raspberry Pi's and laptops) with automatic bootstrap and connectivity to minimize human efforts.

We summarize the comparison between Proteus and existing home-based health monitoring infrastructures and IOT platforms in Table I. We are the first to introduce a manageability focused infrastructure that minimizes manual efforts in deployment and operation. This enables small research teams to efficiently manage longitudinal home-based health monitoring systems. Furthermore, we conduct quantitative evaluations to assess the end-to-end loss, latency, and reduction in manufacturing time, providing concrete evidence of the robustness and efficiency of our infrastructure.

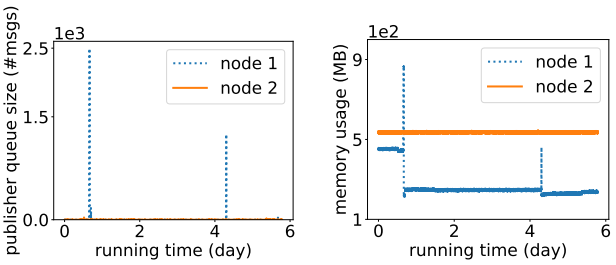
A. Automated and Continuous Deployment

25-fold reduction in manufacturing time from 420 minutes to 17 minutes. Originally, setting up 7 sensor nodes costs 420 minutes (60 minutes per unit in total, a rough estimate of the effort, including 10 minutes for OS and software installation, 5 minutes WiFi setup, 30 minutes dynamic edge server connection, 3 minutes deploy code on edge, 12 minutes update code on edge. Empirically, home WiFi breaks 1/day, thus 30 edge server connections/month. Code updates 1/week, thus 4/month). Through configuration and batch OS flash image cloning, automatic WiFi setup and connectivity, automated self-bootstrapping and OTA update, we reduce 420 minutes to 17 minutes (itemized as OS image cloning 3 minutes, WiFi setup 7 minutes, dynamic edge server connection 2 minutes, code deployment 1 minute and code update 4 minutes per month for 7 nodes in total). This saves significant time and makes it manageable by a small research team to deploy dozens of homes in a matter of hours. We have successfully worked with other research teams, sending them "prep" images and scripts to install and configure all necessary software. With their feedback, we further improve the deployment script and instructions so that they could set up the infrastructure (including sensor nodes and edge servers) without additional help from the development team, demonstrating the effectiveness of our autonomous deployment design.

B. Maintenance

Monitoring Network Issues. We monitor network status based on the two key metrics (i.e., memory usage and publisher queue size). To understand how effective these metrics can indicate data transfer status and abnormal network events occurring on sensor nodes, we examine developer logs on sensor nodes in experiments of the following setting. 7 real nodes run concurrently and continuously for 6 days, sending data at an aggregate rate of 40.6 Mbps. Each sensor node publishes 4 MQTT messages per second. Each message is 0.18 MB, including 20 baseband frames. Considering that Raspberry Pi has 900 MB of RAM, among them about 300 MB is used for the OS and software (e.g. AWS Greengrass) and 150 MB will be used for the data collection program except for the publisher queue, we set the maximum publisher queue size to 2490 number of messages (#msgs), corresponding to 448 MB of data. In Figure 4, we show the publisher queue size and memory usage over time on two of the sensor nodes. We use Figure 4 as examples to illustrate parameter patterns for normal and abnormal network conditions.

From Figure 4 publisher queue size and memory usage on sensor node 2 are relatively low and stable over time (queue size less than 5 #msgs most of the time, never greater than 35 #msgs). We check the developer logs on sensor node 2, and find that it has been running well with no abnormal events. However, there are two different events on sensor node 1. In the first event, the publisher queue size and memory usage



(a) publisher queue size (b) memory usage

Fig. 4: Publisher queue size and memory usage over time on two sensor nodes. Sensor node 2 is running stably with publisher queue size less than 35 #msgs (most are less than 5 #msgs) and memory usage around 500 MB. Sensor node 1 experiences two different abnormal events and is recovered by the watchdog, corresponding to two spikes in the queue size and memory usage.

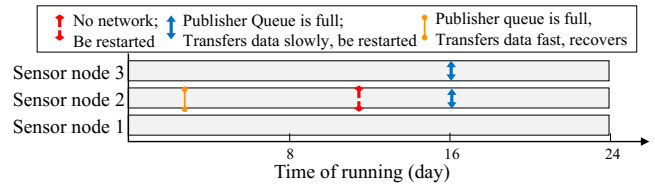


Fig. 5: Abnormal events occur on sensor nodes during a 24-day run.

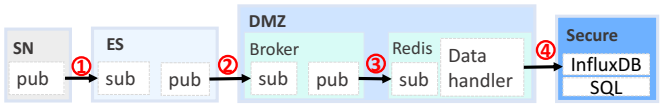


Fig. 6: The hop-by-hop data flow from the sensor node to the cloud database.

both hit limits. In the second event, the publisher queue size increases by 1200 #msgs, while the memory usage increases by 216 MB, which corresponds to the size of 5 minutes of data. By examining the logs on sensor node 1, we find that the publisher queue size reaches the maximum limit after 16 hours of running and the data transfer rate in the next 10 minutes is about 0.57 Mbps, indicating poor data transfer performance (one-tenth of the sensor data rate). The sensor node recovers after being rebooted by the watchdog. Then a network outage occurs after 103 hours and persists for 5 minutes, resulting in 5 minutes of data remained in the queue, until the sensor node is rebooted again by the watchdog. These explain the phenomena we observe in Figure 4. We observe similar spikes in queue sizes and memory usage for other sensor nodes as well.

These observations suggest that *i)* Publisher queue size and memory usage trends indicate the status and performance of sensor devices. Small and stable publisher queue sizes and memory usage indicate healthy performance. Sudden increases in publisher queue size and memory usage reflect poor network connection status and data transfer rates. *ii)* Our watchdog effectively recovers the sensor nodes from network problems. With our watchdog, only 0.08% of data is lost due to unexpected issues on two nodes running for 6 days. Without the watchdog, sensor nodes could crash (and indeed happened), resulting in hours to days of data loss, depending on when the developers or users find and resolve the problem. Furthermore, we save the effort of reading logs for troubleshooting and recovery on errors by our watchdog.

By monitoring CPU usage, we find the CPU load on sensor nodes is mainly between 12% and 15% and it is always below 35%, which shows that the sensor nodes can afford the required amount of computation.

Automatic Recovery. During infrastructure testing, abnormal events occur on several sensor nodes, but our watchdogs are able to recover them from errors and ensure continuous operation. In Figure 5 we present the events on three of the sensor nodes through 24 days of running. Sensor node 1 runs smoothly with no issues, while on sensor node 2, the publisher queue is full after 2.5 days briefly (no reboot or issue) and the node is rebooted after 11.5 days due to a network disconnection. After 16 days, node 2 is rebooted due to having publisher queue full for over 10 minutes. Meanwhile, sensor node 3 encounters the same event and is rebooted automatically. Our results confirm that multiple issues happen in continuous operating systems in the field, and that our watchdog can recover the nodes and prevent data loss (0.03% data loss on edge in 24 days).

During running, we also observe 2 times of file system corruptions on sensor nodes and 2 times of edge server going offline because it has been running for months without being restarted. Due to our monitoring with LED, we can be notified of such hardware failures and remedy timely to prevent data loss.

C. Easy integration of new modality (25× reduction in LOC)

We demonstrate the adaptability of our pipeline to support new data modalities. For time-series data like UWB baseband data and Neulog sensor data, we pack their data into MQTT messages for data transfer and store them in corresponding measurements in InfluxDB on cloud. For example, for Activities of Daily Living (ADL) recognition, the sensor nodes collect baseband data (containing all original information for developing better ADL recognition algorithms) from UWB sensors and package them with timestamps into MQTT messages to send under the topic "*BasebandData_timestamp/Node_ID*". On the cloud, a measurement named "*BasebandData_timestamp/Node_ID*" in InfluxDB should be created, including "*BasebandData*" and "*timestamp*" fields so that the data handler can read messages and write timestamps and baseband data into the database correctly. With our system, we can implement it within 4-5 LOC (2-3 LOC parse data from sensor, and 2 LOC pass it to our API) instead of hundreds of LOC :15 LOC to create a publisher or subscriber, 2 LOC to publish and subscribe to topics of interest, 20 LOC on the subscriber side to parse messages and decide what to do with them (e.g. push to the next hop or store in edge database), 55 LOC to parse data and convert it to the appropriate format before writing to the database, 15 LOC to write data into database in multi-threading. If programming languages used on the edge and cloud are different, the LOC count can easily double.

D. Overall Performance

We evaluate the overall performance of the infrastructure in terms of the data loss and latency with different number of sensor nodes running concurrently for an extended period of

TABLE II: Data Loss Rate Hop by Hop

# of SN	R	ES	Sub	Red	DB	EDB	T
4	23.2	0.05%	0	0	0.06%	8%	15D
7	40.6	0.27%	0	0	0.26%	12%	6D
10	58.0	0.5%	0.17%	0	0.03%	3.8%	14D
16	92.8	0	0	0	0.10%	1.4%	2H

time. Each sensor node sends 1 MQTT message per second, including 80 UWB base-band data frames, where the message data size is 0.72MB. In addition, we run multiple threads simultaneously on a gaming laptop to simulate multiple sensor node connections to stress test the system for data loss and latency. We estimate the data loss and latency from one hop to the next to get a insight of the system performance. The data flow and 4 hops of the infrastructure are shown in Figure 6. We estimate the data loss and latency hop-by-hop between the publishers on sensor nodes (SN), subscriber on the edge server (ES), subscriber of broker on cloud (Sub), subscriber of Redis on cloud (Red) and InfluxDB (DB) on cloud. We measure the loss and latency with different data rates (R) in Mbps, numbers of sensor nodes, and continuous running times (T, in the units of days denoted by "D" and hours "H"). We summarize the test results of data loss in Table II, including the ratio of data temporarily stored in the edge database as a percentage of all data (EDB). Results for 4, 7, and 10 nodes are from real sensor nodes, and the others are node connections from multithreaded simulations.

Results in Table II show that Proteus can support data collection with 32 sensor nodes sending data at 185.6 Mbps aggregate rate, with negligible data loss (0.05% end-to-end), sufficient for the needs of home health monitoring. In the test with 10 real sensor nodes running for 2 weeks, we observe 0.7% end-to-end data loss. Given the resilient pipeline and watchdogs, there are negligible data loss between SN and ES during rebooting of sensor nodes. In our preliminary experiment using 3 sensor nodes for 40 days, we observe no data loss and no errors from sensor nodes to the edge server. Results show that Proteus is reasonably stable and promising for continuous, longitudinal data collection.

EDB in Table II demonstrates the fraction of data backup on edge server at up to 12% upon network failures, which proves such backup storage is necessary. We also conduct offline tests (e.g., manually disconnecting the network of the edge server for 2 hours) and observe no data loss because of edge storage.

Figure 7 shows the average hop-by-hop latency with real and simulated sensor nodes. In the 15-day run test with 4 real sensor nodes, the average latency from sensor node to each hop of the infrastructure is 0.2, 0.23, 0.24, 0.53 seconds respectively. With more sensor nodes and higher data rates, we observe the end-to-end latency slightly increases due to higher data transfer and storage requirements. But even with 16 simulated sensor nodes to connect to the system and send data at 92.8 Mbps, we observe 3.65 seconds end-to-end latency, still sufficient for home health monitoring (e.g., detecting falls within seconds).

We observe that it is frequently desirable to change the baseband data frame rate thus data rate of sensors for different monitoring purposes. Consider activities of daily living (ADL) recognition, where the velocity distributions from Doppler maps are usually utilized. The frame rate per second (FPS) determines the maximum unambiguous radial velocity V_{max}. If the actual velocity exceeds V_{max}, the "aliasing" effect [24] will distort the velocity estimations in the Doppler map. Thus a higher FPS is needed for a larger V_{max}.

However, a higher FPS results in higher per sensor thus aggregate data rates, and may hit the cap of the WiFi upload bandwidth (usually in the range of 1-100 Mbps for most homes). We observe that 11 sensor nodes with an aggregate data rate of 144.1 Mbps (FPS 180, 13.1 Mbps data rate per sensor) hitting the cap, resulting in 3% loss within sensors.

To address this problem, we add compression on sensors, and decompression on the edge server. We use Zlib libary [25], which can cut down the payload size thus bandwidth to 22%, thus quadrupling the number of sensors. To evaluate the potential increase of CPU load and latency on sensors and edge server with data compression/decompression, we configure the sensor node to collect data at 180FPS, 13.1 Mbps to run w/o compression. Given the UWB radio band at 7.25$\sim$10.2 GHz, the maximum unambiguous velocity is 3 m/s. We measure the data loss within sensors (Lo-S), CPU load on sensor nodes (CPU-S) and edge server (CPU-E), end-to-end latency (La-E2E) and end-to-end data loss (Lo-E2E, including loss within sensors) and present the results in Table III ("(w)" denotes tests with compression, "(o)" denotes tests without compression).

With 12 sensor nodes running together at an aggregate data rate of 157.2 Mbps, there is 10.8% of data loss within the sensor, 32% end-to-end loss and 55.1 seconds end-to-end latency without compression due to limited bandwidth. With data compression, we observe the data loss within sensor of 0.6% ($18\times$ reduction), end-to-end loss of 0.34% ($76\times$ reduction) and latency of 0.34 seconds ($162\times$ reduction), with only 3.1% increase in CPU load on the edge server. Additionally, we stress tests the system by simulating 32 sensor node connections on laptops. To ensure consistency, we set the same memory limit for each simulated node as the real nodes. Due to the fact that the simulated nodes are running on laptops, we do not measure the CPU load as it would not provide a meaningful reference value. With 32 nodes collecting data at 180 FPS and an aggregate data rate of 419.2 Mbps, we observe zero data loss within sensor nodes, end-to-end data loss of 0.13% and latency of 0.17 seconds when deploying data compression before transmission. This demonstrates the scalability of our infrastructure to accommodate more sensor nodes in a home with high data rates. Given the fact that the number of sensors is capped by WiFi upload bandwidth, we further boost the number of sensors with compression by $4\times$.

TABLE III: Comparison of data transmission w/o compression

# of SN	Lo-S	CPU-S	CPU-E	La-E2E	Lo-E2E	T
12 (o)	10.8%	23%	6.1%	55.1	32%	3H
12 (w)	0.6%	22%	9.2%	0.34	0.42%	3H
32 (o)	58%	—	10.2%	43.3	60.7%	2H
32 (w)	0	—	6.8%	0.17	0.13%	2H

We also measure the fraction of out of order write into the database. After changing the scheduling such that at any time, at most one thread is writing data for one sensor node,

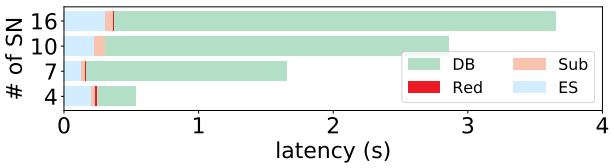


Fig. 7: Average Latency Hop by Hop. With 16 simulated sensor node connections sending data at 92.8 Mbps, the end-to-end latency is 3.65 seconds, slightly higher but still enough for home health monitoring (e.g., fall detection).

we observe that the out of order ratio dropped from 7% to 0% in a test writing 5.8 million data frames to the database.

VI. Discussion

We discuss multiple functions to be supported in future and limitations of our current infrastructure.

Data Privacy. We protect data during transportation (e.g., TLS encryption, edge gateway for access control) and storage (database on a secure, private cloud behind DMZ). To further protect privacy that could be exposed by analysing data, inspection of data content before it leaves home is needed. We are working with experts on system and machine learning privacy to develop needed solutions.

Analytics Support. We have yet to deploy real analytic models on the edge server. With AWS greengrass, deploying and updating containerized models only requires a few mouse clicks on the backend dashboard. We have pushed full docker images to both Pi's and edge server successfully many times thus the capability already exists.

System Visibility. We enable system visibility in two aspects: 1) visuals: indicating device status with different lighting patterns of LED; 2) code instrumentation: developer logs including key metrics. At present, we do daily metrics analysis offline on sensor nodes for performance reviewing. In our experiences, daily checks are mostly efficient because of all the watchdog mechanisms we have. In the future, we will integrate real-time transmission of the metrics into the data-agnostic pipeline to enable real-time visualization and monitoring of performance.

Auto-scaling on Cloud Workload. Our work involves auto-scaling on the cloud based on workload. There are existing services that we plan on leveraging that enable system's auto scalability when the load (either network, CPU, or both) increases to prevent increase in latency or data loss, such as: Amazon EKS [26] and AWS Elastic Beanstalk [27] services. This allows the system to scale on an as-needed basis.

Computing Resources on Sensor and Edge. Based on our current results in Tab III, CPU usage is less than 23% on sensor nodes and 10.2% on the edge server (with 32 simulated sensor nodes running concurrently), indicating that commodity hardware is sufficient for the load. In the future, we plan to deploy deep learning models on the edge and explore methods (e.g. model compression) to efficiently deploy and run these models on constrained edge resources.

Limitations. Our current work has limitations: *i) scale:* Although all our infrastructure is designed for scaling, the current test is limited to a dozen real sensors, about 30 simulated sensors. This is mainly constrained by hardware resources available to us - cloud, sensor and edge. We are in conversation with AWS Higher Education to obtain more resources to conduct much larger scale experiments. *ii) environment:* current evaluation is mainly done in the simulated home. We are in the process of deploying and testing the infrastructure in real homes which we believe will help uncover new issues and challenges , including participants of different health conditions. *iii) database type:* current system can handle time-series data and can be expanded for large record files (e.g., video, image files). While influxDB allows for storing events which can handle them, a record-based database (e.g., MongoDB) is more efficient for such data. Another cloud microservice handler can be added to support such data storage in MongoDB. *iv) expertise requirement:* We automate the management of the system so that only simply actions are required for non-expert home residents, and manufacturing effort is greatly (25-fold) reduced for researchers to manage the system. We do assume for researchers, some level of expertise (e.g. usage of DMS to check device status, update programs OTA, and etc.) is required to manage the system. We are planning home deployments to further test the system to understand what additional challenges it may pose, and will address them in future.

VII. Conclusion

In this paper, we present our experience in developing and pilot deploying an infrastructure for home-based health monitoring. We recognize the human efforts needed in the full cycle of deploying and operating such infrastructure. Our system cuts down on labor efforts by 25X and reduces code necessary for deployment by 25X. Our system ran end to end (from sensor on edge to database on cloud) and had 0.7% data loss over 14 days experiment. We share experiences and lessons hopefully valuable for other research teams, and we plan to make the infrastructure available to the community for shared data collection once it becomes mature.

References

[1] M. Liu, M. Elbadry, Y. Hua, Z. Xie, and F. Ye, "Proteus: Towards a manageability-focused home-based health monitoring infrastructure," in *Proceedings of the 14th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, 2023, pp. 1–6.

[2] H. Kwon, Z. Xie, M. Liu, and F. Ye, "Poster: Comparative study of transformer models on a large multivariate time series har dataset," in *2024 IEEE/ACM Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, 2024, pp. 193–194.

[3] A. Hornback, W. Shi, F. O. Giuste, Y. Zhu, A. M. Carpenter, C. Hilton, V. N. Bijanki, H. Stahl, G. S. Gottesman, C. Purnell *et al.*, "Development of a generalizable multi-site and multi-modality clinical data cloud infrastructure for pediatric patient care," in *Proceedings of the 13th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, 2022, pp. 1–10.

[4] I. Chouvarda, N. Y. Philip, P. Natsiavas, V. Kilintzis, D. Sobnath, R. Kayyali, J. Henriques, R. P. Paiva, A. Raptopoulos, O. Chetelat *et al.*, "Welcome—innovative integrated care platform using wearable sensing and smart cloud computing for copd patients with comorbidities," in *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE, 2014, pp. 3180–3183.

[5] P. Woznowski, X. Fafoutis, T. Song, S. Hannuna, M. Camplani, L. Tao, A. Paiement, E. Mellios, M. Haghighi, N. Zhu *et al.*, "A multi-modal sensor infrastructure for healthcare in a residential environment," in *IEEE ICCW 2015*. IEEE, 2015.

[6] K. Technologies, "Enterprise IoT platform with free plan — kaa," 2023. [Online]. Available: https://www.kaaiot.com/

[7] T. MathWorks, "IoT analytics - thingspeak internet of things," 2023. [Online]. Available: https://thingspeak.com/

[8] Y. Liu, G. Zhang, C. G. Tarolli, R. Hristov, S. Jensen-Roberts, E. M. Waddell, T. L. Myers, M. E. Pawlik, J. M. Soto, R. M. Wilson *et al.*, "Monitoring gait at home with radio waves in parkinson's disease: A marker of severity, progression, and medication response," *Science Translational Medicine*, vol. 14, no. 663, p. eadc9669, 2022.

[9] L. Zhang, D. Zheng, M. Yuan, F. Han, Z. Wu, M. Liu, and X.-Y. Li, "Multisense: Cross-labelling and learning human activities using multimodal sensing data," *ACM Trans. Sen. Netw.*, vol. 19, no. 3, apr 2023. [Online]. Available: https://doi.org/10.1145/3578267

[10] M. Liu, L. Zhang, D. Zheng, and X. Li, "Collaborative deep sensing by dynamically fusing multiple models," in *2021 7th International Conference on Big Data Computing and Communications (BigCom)*, 2021, pp. 316–323.

[11] P. N. Dawadi, D. J. Cook, and M. Schmitter-Edgecombe, "Automated cognitive health assessment using smart home monitoring of complex tasks," *IEEE transactions on systems, man, and cybernetics: systems*, vol. 43, no. 6, pp. 1302–1313, 2013.

[12] M. Elbadry, M. Liu, Y. Hua, Z. Xie, and F. Ye, "Poster: Towards robust, extensible, and scalable home sensing data collection," in *Proceedings of the 8th ACM/IEEE International Conference on Connected Health: Applications, Systems and Engineering Technologies*, 2023, pp. 192–193.

[13] L. Zhang, D. Zheng, Z. Wu, M. Liu, M. Yuan, F. Han, and X.-Y. Li, "Poster: Cross labelling and learning unknown activities among multimodal sensing data," in *The 25th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3300061.3343407

[14] M. Hämäläinen, L. Mucchi, S. Caputo, L. Biotti, L. Ciani, D. Marabissi, and G. Patrizi, "Ultra-wideband radar-based indoor activity monitoring for elderly care," *Sensors*, vol. 21, no. 9, p. 3158, 2021.

[15] H.-K. Ra, A. Salekin, H. J. Yoon, J. Kim, S. Nirjon, D. J. Stone, S. Kim, J.-M. Lee, S. H. Son, and J. A. Stankovic, "Asthmaguide: an asthma monitoring and advice ecosystem," in *2016 IEEE Wireless Health (WH)*. IEEE, 2016, pp. 1–8.

[16] H. Choi, A. Lor, M. Megonegal, X. Ji, A. Watson, J. Weimer, and I. Lee, "Vitalcore: Analytics and support dashboard for medical device integration," in *IEEE/ACM CHASE 2021*. IEEE, 2021, pp. 82–86.

[17] A. Ravulavaru, *Enterprise Internet of Things Handbook: Build end-to-end IoT solutions using popular IoT platforms*. Packt Publishing Ltd, 2018.

[18] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "MQTT-S—a publish/subscribe protocol for wireless sensor networks," in *COMSWARE'08*. IEEE, 2008, pp. 791–798.

[19] R. Ltd, "Redis," 2023. [Online]. Available: https://redis.io

[20] Y. Hu and W. Li, "Spatial data infrastructures," *arXiv preprint arXiv:1707.03969*, 2017.

[21] A. Co, "Advantech co-creating the future of the iot world," 1983-2023. [Online]. Available: https://www.advantech.com/en-us

[22] S. Sadhu, D. Solanki, N. Constant, V. Ravichandran, G. Cay, M. J. Saikia, U. Akbar, and K. Mankodiya, "Towards a telehealth infrastructure supported by machine learning on edge/fog for parkinson's movement screening," *Smart Health*, p. 100351, 2022.

[23] E. T. Inc, "Introduction to MQTT QoS 0, 1, 2—EMQ," 2013-2023. [Online]. Available: https://www.emqx.com/en/blog/introduction-to-mqtt-qos

[24] F. Serafino, C. Lugni, J. C. Nieto Borge, and F. Soldovieri, "A simple strategy to mitigate the aliasing effect in X-band marine radar data: Numerical results for a 2d case," *Sensors*, vol. 11, no. 1, pp. 1009–1027, 2011.

[25] G. Roelofs, "Zlib home site," 1996-2023. [Online]. Available: http://www.zlib.net/

[26] Microsoft, "Managed Kubbernets Service (AKS) — Microsoft Azure," 2023. [Online]. Available: https://azure.microsoft.com/en-us/products/kubernetes-service

[27] A. W. Services, "Website and Web APP Deployment - AWS Elastic Beanstalk - AWS," 2023. [Online]. Available: https://aws.amazon.com/elasticbeanstalk/?nc1=h_ls