Raproto: An Open Source Platform for Rapid Prototyping of Wearable Medical Devices

Amanda Watson
aawatson@seas.upenn.edu
University of Pennsylvania
Philadelphia, PA, USA

Hyonyoung Choi
hyonchoi@seas.upenn.edu
University of Pennsylvania
Philadelphia, PA, USA

Insup Lee
lee@seas.upenn.edu
University of Pennsylvania
Philadelphia, PA, USA

James Weimer
weimerj@seas.upenn.edu
University of Pennsylvania
Philadelphia, PA, USA

ABSTRACT

For researchers, especially in remote health monitoring and ubiquitous computing, it is common to expend a significant amount of time and effort to develop data collection systems that can be used outside of the lab or clinic. These systems tend to be customized and highly specific to the task at hand; thus, they are not general enough to support other tasks. In this paper, we present Raproto, an open-source, easy-to-use rapid prototyping platform that facilitates data collection and visualization from sensors on commercially available, off-the-shelf smartwatches. The Raproto platform consists of three components: the smartwatch, communication protocol, and server. These components support the collection, transmission, storage, analysis, and visualization of data. We evaluate our platform on limiting factors including the smartwatch battery life, data loss during transmission, and data latency. Overall, we find that the smartwatch can last for over 24 hours on a single charge, has little to no data loss, and less than a second of data latency per transmission.

CCS CONCEPTS

• **Human-centered computing** → **Mobile devices**.

KEYWORDS

healthcare, open source, wearable technology, smartwatch

ACM Reference Format:
Amanda Watson, Hyonyoung Choi, Insup Lee, and James Weimer. 2021. Raproto: An Open Source Platform for Rapid Prototyping of Wearable Medical Devices. In *Medical Cyber Physical Systems and Internet of Medical Things (MCPS '21), May 18, 2021, Nashville, TN, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3446913.3460315

1 INTRODUCTION

The increasing use of smart medical devices and wearable technology in healthcare has driven the development of remote health monitoring, improving patient outcomes and reducing the overall cost of care. Due to its success, the remote health monitoring market is expected to grow to more than $4.1 billion by 2028 [1]. However, significant time, effort, and money must be expended to implement remote health monitoring systems, as custom monitoring devices are needed, and platforms to aggregate, store, and analyze the data from the devices must be developed. Unlike the costly, custom devices developed for remote health monitoring, smart devices are inexpensive and already have many of the sensors used for personal remote health monitoring. Smart devices also have a pre-existing code base and easy to use user interface that can be leveraged for custom app development. While these devices are not clinically tested, they provide researchers with a more cost-effective method to develop rapid prototypes in their studies.

Commercially available smart devices provide computing ability, storage, connectivity, and a wide variety of sensors, housed in a portable frame with a familiar and easy-to-use interface. Smartwatches are commercially available smart wearable devices that are comfortable to wear and have become popular among the general population. In fact, by 2026, it is expected that 157.2 million smartwatches will be sold worldwide [7]. Their high adoption rate highlights their user-friendliness, comfort when worn, and useful functionalities. These standalone wearable devices provide continuous data collection, remote access and control, and communication with users, allowing them to be a productive research tool. Further, smartwatches do not have to be worn on the wrist, increasing their usefulness for data collection.

In recent years, there has been increasing interest in utilizing smartwatches for research on body sensing [9]. Limited work has been done to create easy-to-use, open-source platforms to promote rapid prototyping with smartwatches. WaDa [6] is a standalone Android-based smartwatch app that facilitates data collection from the sensors on a smartwatch. While the WaDa smartwatch and desktop app are freely available for research and academic purposes, it is not open source; thus, it cannot be customized to specific tasks. ROAMM [3] is a Tizen-based smartwatch framework for online assessment and mobility monitoring leveraging a smartwatch application and a remotely-connected server. While this framework is not open-source, the code can be requested under limited circumstances. This work aims to develop a fully open-source platform for use in remote health monitoring through the combination of a smartwatch-based application and a remote server. While we focus

MCPS '21, May 18, 2021, Nashville, TN, USA

ACM ISBN 978-1-4503-8327-1/21/05…$15.00
https://doi.org/10.1145/3446913.3460315

on the smartwatch, other smart devices can be connected to our platform.

We summarize our contributions as follows:

- We present Raproto, an open-source extensible platform for rapid prototyping of wearable medical devices. The Raproto system is made up of a smartwatch application, communication protocol, and server for the collection and storage of data. In this paper, we focus on creating a Tizen-based smartwatch application, but other smart devices, including smartphones and non-Tizen smartwatches, can be used.
- Our experimental results show that our system is very customizable in terms of battery, data loss, and data latency and the trade-offs between them. We observe that a smartwatch with the Raproto application running can last for more than 24 hours on a single charge, has little to no data loss, and experiences less than one second of data latency.

The remainder of the paper is structured as follows. First, we describe the components of our Raproto platform: smartwatch, communication protocol, and server. Second, we evaluate our platform based on the limiting factors: battery life, data loss, and data latency. Third, we discuss the future directions for the Raproto platform. Finally, we wrap up with the conclusion.

2 RAPROTO PLATFORM

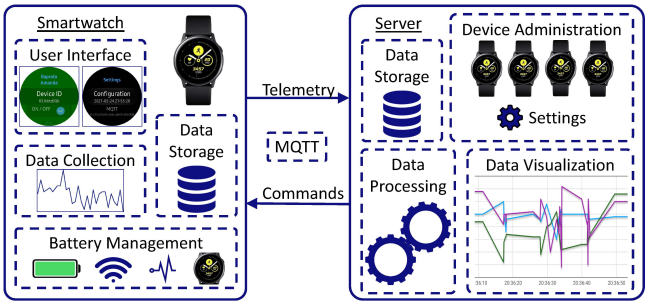

Figure 1: Raproto Platform

The Raproto platform is made up of three components: the smartwatch application, communication protocol, and the server. The smartwatch application facilitates the collection and storage of sensor data. The communication protocol enables the transfer of the sensor data collected on the smartwatch to the remote server and transfers commands from the remote server to the smartwatch. The remote server supports the configuration of the smartwatches, data storage, processing, and visualization. The Raproto platform and its components are shown in Figure 1.

2.1 Smartwatch Application

The Raproto application runs directly on the smartwatch without the need for a companion smartphone application. Raproto's independence makes it more convenient and less constrained during use than apps that require a companion application through a connected smartphone. The application records sensor data, stores and transmits that data, and features an easy-to-navigate UI for setup. The application was developed on the Samsung Galaxy Active smartwatches. These smartwatches run on the Tizen operating system, which provides APIs for managing and interacting with sensors and communication protocols. While our application targets the Tizen-based Samsung Galaxy Active, it can be adapted for other smartwatches such as those running Android WearOS. The smartwatch application repository is available at [8]. In practice, installation and setup from source takes approximately 30 minutes for users regardless of smartwatch programming experience.

(a) Configuration **(b) Main Screen** **(c) Data Collection**

Figure 2: User Interface of the Smartwatch Application

User Interface. The user interface, shown in Figure 2 serves two purposes: configuration and starting or stopping data collection. The Raproto application requires an initial configuration to set up the data collection. The user can accomplish the configuration task by opening the settings menu and pressing the configuration button as shown in Figure 2a. Once pressed, it loads the configuration settings from the remote server. In most cases, this includes the device name, sensor configurations, battery management settings, and data transmission settings. Following the configuration step, the user can begin collecting data. Data collection is started and stopped using a toggle switch as shown in Figure, 2b. Once started, the application turns the background green as shown in Figure 2c to inform the user that data is being collected. The application will continue to collect data in the background if the application is closed or switched to another screen. The only explicit method to stop data collection is to turn it off via the toggle switch. If an error occurs and data is no longer be collected, the screen background turns red to alert the user. The user can then restart the application and resume collecting data.

Watch Configuration. The Raproto application is designed to support a wide range of monitoring applications – each possibly having different requirements for data collection, battery life, and latency. To allow for on-the-fly customization to specific applications, the Raproto smartwatch application is configured via a JSON string to control Wi-Fi usage, sensor selection, and transmission rates. By pressing the configuration button (Figure 2a) Raproto will connect to the remote server over Wi-Fi using MQTT middleware [2] (see Section 2.2). Once connected, when the watch publishes on a customizable configuration-request topic (default: v1/devices/me/attributes/request/1) the server will respond by publishing on the customizable configuration-received topic (default: v1/devices/me/attributes/response/+). The default publish and subscribe topics are chosen to be consistent with the default settings of Thingsboard [10] (as described in Section 2.3). We note that

the Raproto watch application runs independently of Thingsboard, only borrowing the Thingsboard API for seamless integration (if desired).

Data Collection. The Raproto application collects data from the available sensors on the Samsung Galaxy Active. Currently, it supports the accelerometer, gyroscope, gravity sensor, heart rate monitor, photoplethysmograph, and battery level sensor. The application is customizable and any combination of sensors may be chosen. The sampling frequency of each sensor is also configurable. These customizations are chosen remotely and sent to the smartwatch during watch configuration. This allows the smartwatch to be updated whenever it is required without the need to return the smartwatch to the developer. The collected sensor data is loaded into a JSON format consistent with Thingsboard [10] and either immediately transmitted to the server or stored until it can be transmitted to the server.

Data Storage. Collected sensor data is not always immediately transmitted to the remote server. When this occurs, the data needs to be stored so that it can be transmitted at a later time without any loss of data. Our application provides 40 megabytes of buffer storage. Once a wireless connection is established, data is packaged and sent in 10 KB messages. The size of the messages as well as the size of the buffer storage can be customized depending on the needs of the application. Providing data storage allows our platform to function seamlessly in environments where wireless connections are not readily available or reliable.

Battery Management. Smartwatch battery life has improved in recent years, but in research studies, battery life is still a limiting factor [4]. The largest drains on the battery are the display, volume of sensor data, and Wi-Fi radio settings.

The largest consumer of smartwatch battery is the display as for many smartwatches; it remains on even while in sleep mode [5]. While our application cannot directly control all of the settings that will extend the smartwatch's battery life, we provide the user with instructions for a one-time configuration for maximal battery life. This configuration takes less than five minutes to set up which is nominal given the extended battery life it provides. For example, we ask the user to adjust the display settings so the watch face is not always on and to set the watch face to a static black and white screen.

The volume of sensor data is affected by the number of sensors collecting data and their sampling rates. As more sensors are added, and as sampling rates are increased, the volume of sensor data also grows. In research studies, it is common to collect data from all sensors simultaneously at high sampling rates to ensure high-quality data is collected. Since the data required by these studies, we acknowledge that battery drain occurs due to this and mitigate the effects this causes in data transmission.

The settings of the Wi-Fi radio can provide battery savings by controlling how often the radio transmits data and when it turns on and off. The smartwatch application transmits data on a configurable schedule; the default is every 60 seconds but this should be adjusted to ensure we do not fill the buffer to avoid data loss. The more time that can be spent between data transmission events promotes more battery savings. To enhance these battery life savings, Raproto can also control the Wi-Fi connection by turning it on and off based on the transmission settings. This compounds the battery savings described above as the further apart transmission event occur means that the Wi-Fi radio will be off for longer periods of time. This is not feasible with all Wi-Fi protocols, so by default, we set the Wi-Fi to always remain on.

2.2 Communication Protocol

MQTT [2] is a publish/subscribe messaging protocol designed to connect remote devices with a small footprint and minimal network bandwidth consumption. It allows for telemetry to be sent from the smartwatches to the server and commands to be sent from the server to the smartwatches. MQTT supports three levels of quality of service- Level 0 is the lowest-overhead method where the smartwatch will send telemetry without any acknowledgment that the server has received it. Level 1 guarantees that the server receives the telemetry by sending an acknowledgment back to the smartwatch. If the acknowledgment is lost, then the smartwatch will resend its telemetry until it receives an acknowledgment. This can create duplicate telemetry received by the server, but since the data is sent with a timestamp, duplicates will be easy to identify. Level 2 guarantees the telemetry will be received exactly one time by completing a "handshake" to confirm that the telemetry has been sent and that the acknowledgment has been received. Our system defaults to Level 1, but both other levels are supported.

2.3 Remote Server

The server provides a platform to register new devices, remotely configure device settings, store data, process data, and customize visualizations to suit the task at hand. For the purposes of this work, we leveraged an open-source internet of things platform, Thingsboard [10]; but custom-built applications or other internet of things platforms can be used as well if they support communication via the MQTT protocol descibed in the previous subsection. Thingsboard is available through a web portal allowing researchers to easily access it through their internet browser.

Device Administration. The device administration module provides the ability to configure and communicate with all active Raproto smartwatch applications. This module provides researchers with a convenient and fast way to reconfigure smartwatches remotely via a web portal without the need to collect the physical watches. This is accomplished through a bidirectional communication in which the server sends configuration parameters to the smartwatch and receives sensor data from the smartwatches. In Thingsboard, this is done in the device administration portal as shown in Figure 3 by completing the following steps. First, add a new watch to the system. Second, configure the smartwatch attributes and data collection parameters as suits the deployment of Raproto. Third, press update configuration on the smartwatch running the Raproto application. Lastly, check that the smartwatch connection and configuration are successful by viewing the latest telemetry. While the device administration portal is mainly for the configuration of the smartwatches, it is a great resource when debugging connection issues.

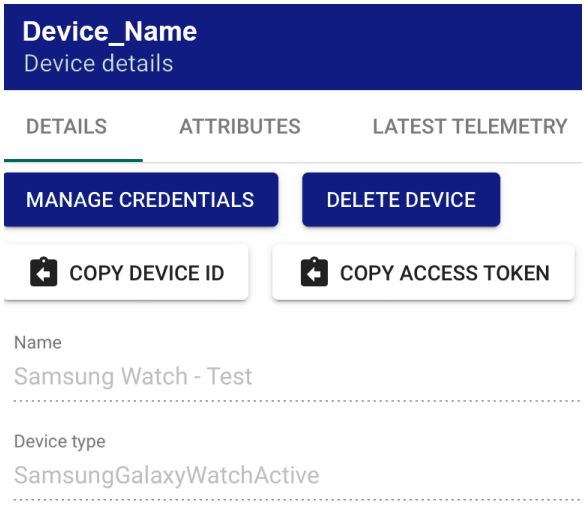

Figure 3: Device Administration Portal

Data Storage. Once the data is transmitted from the smartwatch to the server, it is stored in a database from which it can be processed, analyzed, and visualized. Specifically, we use TimescaleDB to store our time series data. It is an open-source database optimized for time-series data to provide fast storage of new entries and quick processing of complex analysis. To interact directly with the database, SQL queries can be run. The database is not directly accessible for end-users. To interact with the data they must use dashboards that Thingsboard provides.

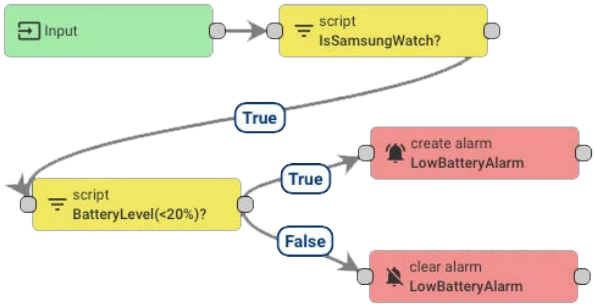

Figure 4: Device Processing Rule Chain

Data Processing. The remote server facilitates basic data processing enabling meaningful analysis and actions based on the data as it is received. We choose to run this on the remote server instead of the smartwatch as the server has more computing power, and battery life is not a concern. Thingsboard supports data filtering, enrichment, analytics, transformation, and rule chains. Rule chains provide the ability to transform and normalize the data stored in the database. An example of a rule chain that checks and alerts the user of a low battery for a smartwatch running the Raproto application is shown in Figure 4. It also allows for alarms and notifications to be sent to the user based on conditions over the incoming telemetry data, including sensor data, attribute updates, device inactivity, and user actions on the smartwatch.

Data Visualization. An essential component of the remote server is the ability for a user to visualize the data that has been collected and processed. Within these visualizations, the user should be able to view live streaming and static data, receive notifications about alerts, and manipulate the displayed data. While there are many available visualization platforms, we will discuss Thingsboard. In Thingsboard, dashboards are created and customized to display time-series sensor data. A single dashboard can contain many widgets which are pre-programmed to display data in a particular fashion. For example, in Figure 5 there are two different types of widgets, two digital scales on the left and a time series plot on the right. This is only a small example of the pre-programmed widgets available in Thingsboard. These widgets make displayed data easier to view and interpret at a glance, enhancing the dashboard's usefulness. The dashboards are web-based and can be shared and viewed by users other than the developer.

3 EVALUATION

We evaluate Raproto based on three metrics: battery life, data loss, and data latency. Battery life is one biggest limiting factor of using smartwatches in research studies. Data loss is important to take into account depending on the nature of the study at hand. Data latency is important in any system that provides feedback to the user from the server.

3.1 Battery Life

As previously discussed, smartwatch battery life is a limiting factor in research studies. The most significant drains on the battery life comes from the display, volume of sensor data, volume of radio traffic, and Wi-Fi radio usage [5]. Here we evaluate how these characteristics affect battery life and how effective our mitigation strategies are at preventing battery drain.

Display. We provide instructions for setting up the smartwatch to conserve as much battery as possible. This is a one-time, easy-to-do, and quick configuration process that mitigates battery drain from properties of the smartwatch that do not concern our application. The instructions will vary on various models of smartwatches, but the overall premises will remain the same such as allowing the screen to sleep. Overall, we see a gain of approximately 4.8 hours of battery life with our quick and easy configuration process.

Table 1: Expected Battery Life with Accelerometers at Various Sampling Rates

Sampling Rate (Milliseconds)	Expected Battery Life (Hours)
20	28.58
10	18.18
5	12.5
1	7.14

Sensor Data Volume. We discuss the effects a large volume of data has on battery life. We evaluate two parameters: sampling rate and

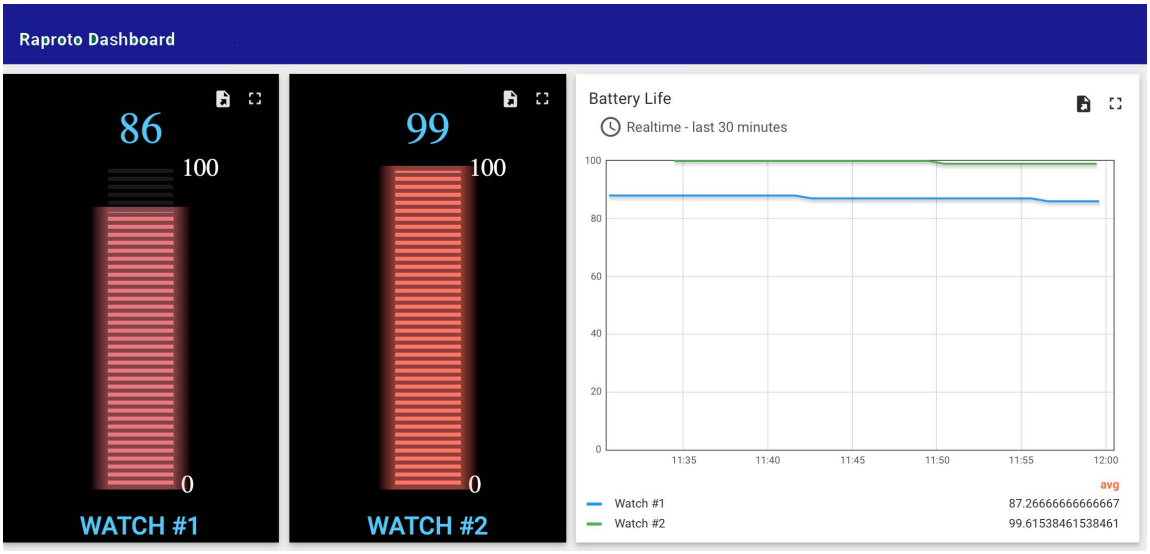

Figure 5: Dashboard

Table 2: Sensor Combinations Expected Battery Life

Accel	Gyro	Gravity	HRM	PPG	Battery Life
x					28.6 hrs
	x				28.6 hrs
		x			28.6 hrs
			x		33.3 hrs
				x	28.6 hrs
x	x				28.6 hrs
x	x	x			25.0 hrs
x	x	x	x		22.2 hrs
x	x	x	x	x	22.2 hrs

number of sensors. First we show that as sampling rate increases, battery life increase by only collecting accelerometer data in Table 1. Second, we show the impact that collecting data from multiple sensors has on the battery life in Table 2. We also show the impact each individual sensor has on the battery life. Overall we see that as the volume of sensor data increases, the battery life decreases.

Wi-Fi Radio Settings. We provide a Wi-Fi configuration that conserves power at the cost of some data latency. The Raproto power savings Wi-Fi configuration turns the Wi-Fi radio off while not transmitting data to the remote server. We observed this configuration saves approximately 4.8 hours of battery life. This is significant in studies where the goal is to collect data over long periods of time where near-instantaneous data transmission is not required.

3.2 Data Loss

MQTT has three levels of service providing different guarantees on data transmission. In this experiment, we on vary the MQTT level of service. We evaluate each level of service for data points lost and duplicated data entries recorded in the database. We calculate the number of data points lost by comparing the timestamps to the sampling frequency of the sensor. We check for duplicates by checking for duplicate timestamps. We display these results in Table 3 and explain them in the following paragraphs.

Table 3: Data Loss and Duplication

MQTT Service Level	Total	Lost	Duplicated
0	8,965	29	0
1	3,879	0	5
2	47,550	0	0

The lowest level of MQTT service, 0 does not guarantee that the data has been received by the server. It sends the datapoint at hand and then moves on to the next without verifying the first data point was received. Level 0 showed 29 data points lost among the total of 8,965 data points transmitted account for 0.32% of the data sent. We observed no duplicate data points, which is expected as this level of service does not resend data that could have been lost.

Our system defaults to MQTT service level 1, in which the smartwatch is assured the data has been received by the server after it is sent. Because it is only concern is that the data is received on the server end, data can be sent multiple times and thus duplicated in the database. In the battery experiments above, we used this default setting so we leverage that data here. We verified that there was no data loss among all 3,879 data points collected. We calculated the number of duplicated entries to be five 5 data points or 0.12% of the total data received.

The highest level of MQTT service guarantees that data is received only once by the server. This should take care of the data loss and duplication seen in the previous tests. We verified that level 2 showed no data loss or duplicate entries over 47,550 datapoints.

It should be noted that our data loss experiment was carried out is an environment that is at a low risk for data loss. Each level of service is suited to a different type of environment. For example, in

a medical setting where data loss is higher due to being near strong electromagnetic equipment, level 2 should be used to ensure data is received and not frequently duplicated. In low-risk transmission settings and when every data point collected is not critical, level 0 will extend the battery life allowing for longer studies. In studies where every data point is critical and speed is of importance, level 1 should be utilized.

3.3 Data Latency

The time accrued between data collection and the receipt of data by the server is influenced by the frequency at which Raproto transmits data and the transit time between the smartwatch and the server. The frequency at which we transmit data is a configurable parameter that can be tuned to meet the needs of the specific application. If we transmit data as is collected, we see a latency of less than one second between when the data is transmitted and the response is received. MQTT level of service also affects the data latency as the higher the level of service, the more time is needed to guarantee a data point has been received only once. For studies that require very low latency, service level 0 provides the least amount of transit time but there is a trade off with data loss.

4 EXTENSIBILITY

There are many future directions for the Raproto System to be explored. In this section, we discuss supporting alternate operating systems, making our Raproto Application available on app stores, and adding supporting smartwatches with cellular radios to allow our system to transmit data in environments without Wi-Fi.

4.1 Alternate OS Support:

Samsung Tizen-based smartwatches are not the only smartwatches available on the market. The other smartwatches provide different functionalities such as new sensors and different wireless radios. To make Raproto more accessible, we leave it up to future work to build applications for other smartwatches such as those that run Android Wear OS. While the native applications must be rebuilt to suit the new environment, the functionality and structure of the current Tizen-based Raproto application can be reused. Once the new applications are built, they should be tested and verified on the new smartwatches to ensure the new hardware is working and compatible with Raproto. From the server-side, the setup to add new sensors is quick and easy.

4.2 Application Store Availability

The Raproto application currently must be sideloaded via Tizen studio. While we provide instructions for user on how to accomplish this, and most users can fully set up their watch in approximately 30 minutes, it is not user-friendly for non-developers as users must download Tizen Studio and be comfortable enough using a terminal to connect their smartwatch to their computer. To simplify this process, especially for non-developers, we plan to make Raproto available for download on the Tizen App Store. As Raproto applications based in other operating systems are developed, we plan to put them on their respective app stores as well.

4.3 Cellular Enabled Smartwatches

Due to advances in smartwatch technology, some smartwatches now have cellular radios integrated directly. The problem that arises is that cellular radios are power-hungry. This causes a more rapid battery drain than seen with the smartwatches in our study. However, cellular radios can enable data transmission in environments where Wi-Fi cannot be connected. When adding this functionality, a trade-off between battery life and data availability can be evaluated and optimized for specific projects. Another way of reducing the battery drain is to choose the lowest power radio available to transmit data so the cellular radio will only be used when necessary.

5 CONCLUSION

In this paper, we presented Raproto, an open-source, easy-to-use rapid prototyping platform that facilitates data collection from sensors on commercially available off-the-shelf smartwatches. This platform provides researchers, especially in remote health monitoring and ubiquitous computing, an quick, simple to use, and customizable solution for developing data collection systems. We evaluated our platform and observed that a smartwatch with the Raproto application running lasted for over 24 hours on a single charge, has almost no data loss, and experienced less that one second of data latency.

ACKNOWLEDGMENTS

This research was supported by the U.S. National Science Foundation under grant 1915398 and the National Institute of Health under grants R01-EB029363, R01-EB029767, and R43-MH121205. The project was funded, in part, under a grant from the Pennsylvania Department of Health.

REFERENCES

[1] Grandview [n.d.]. Remote Patient Monitoring System Market Growth & Trends. https://www.grandviewresearch.com/press-release/global-remote-patient-monitoring-devices-market. Pubished: January, 2021.
[2] U. Hunkeler, H. L. Truong, and A. Stanford-Clark. 2008. MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks. In *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*. 791–798. https://doi.org/10.1109/COMSWA.2008.4554519
[3] Matin Kheirkhahan, Sanjay Nair, Anis Davoudi, Parisa Rashidi, Amal A Wanigatunga, Duane B Corbett, Tonatiuh Mendoza, Todd M Manini, and Sanjay Ranka. 2019. A smartwatch-based framework for real-time and online assessment and mobility monitoring. *Journal of biomedical informatics* 89 (2019), 29–40.
[4] Christine E King and Majid Sarrafzadeh. 2018. A survey of smartwatches in remote health monitoring. *Journal of healthcare informatics research* (2018), 1–24.
[5] Xing Liu and Feng Qian. 2016. Poster: measuring and optimizing android smartwatch energy consumption. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking (MobiCom)*. 421–423.
[6] Md Abu Sayeed Mondol, Ifat A Emi, Sirat Samyoun, M Arif Imtiazur Rahman, and John A Stankovic. 2018. WaDa: An Android Smart Watch App for Sensor Data Collection. In *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*. 404–407.
[7] Mordor [n.d.]. Smartwatch Market - Growth, Trends, COVID-19 Impact, and Forecasts (2021 - 2026). https://www.mordorintelligence.com/industry-reports/smartwatch-market.
[8] Raproto [n.d.]. Raproto-Tizen. https://github.com/weimerj/Raproto-Tizen.
[9] Nour Takiddeen and Imran Zualkernan. 2019. Smartwatches as IoT edge devices: A framework and survey. In *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE, 216–222.
[10] Thingsboard [n.d.]. ThingsBoard: Open-source IoT Platform. https://thingsboard.io.