An Android Wear OS Framework for Sensor Data and Network Interfaces

Bashkim Berzati, Andre Ippisch, Kalman Graffi
Technology of Social Networks Lab
University of Düsseldorf, Germany
{bashkim.berzati, ippisch, graffi}@hhu.de

***Abstract*—Wearable devices like smartwatches conquer the market and simplify our everyday life. Mostly, these devices are connected to our mobile phones, helping us to make communication easier and faster. However, wearable devices feature plenty of sensors and network interfaces that are mostly unused. Thus, the question arises on how to use this potential to improve the functionality and user experience of handset applications.**

This paper presents a framework for the Android operating system which enables us to retrieve information from the wearable device and store and use it on the handheld device. The primary focus lies on sensor and network interfaces, and as a result, this information can be considered for further computations. We offer the framework as an Android library which can be included in all Android projects.

We evaluate the framework by conducting network performance tests as well as tests regarding CPU, memory and battery usage, and achieve promising results. Concluding, we can say that the wearable technology offers lots of opportunities for present and future Android projects. Due to the different types of framework services, we achieve the goal of providing a broad base of information that can be utilized by all developers.

***Index Terms*—Wearables, Smartphones, Android, Wear OS, Sensor, Network Interfaces, Opportunistic Networks**

I. Introduction

Today, there are many wearable devices on the market which simplify our everyday life. This technology includes devices like activity trackers, smart glasses, and smartwatches. Mostly, these devices are connected to our smartphone, helping us to make communication in our public life easier and faster.

Wearable devices feature an abundance of sensors and network interfaces that mostly remain unchallenged. This is due to the habits of an average user which are mainly characterized by getting notifications of incoming messages or receiving weather news. Thus, the question arises how to use a wearable device's hidden potential, mainly to improve the functionality and user experience of handset applications. Furthermore, this research will try to explore the possibilities and limits current wearable devices bring to be able to assess their capabilities.

To this end, the idea came up to design a solution that makes it possible to use corresponding wearable functions efficiently. The design should enable developers and users to access wearable interfaces and sensors remotely via mobile phone. The so far unchallenged wearable should thus become a new source of information from which we can obtain additional data. Using this additional data raises the possibility to develop new functions and improve existing ones on the mobile phone.

We motivate the realization of this idea by the challenges in our network application *opptain* [4]. *opptain* is an application for Android that uses Opportunistic Networking [7] for local data exchange. *opptain* manages its data exchange between clients via Wi-Fi. If a network node is a hotspot in the network, it can only accept incoming requests but not scan for available participants at the same time; and if two hotspots meet, they will not see each other. A problem is the Wi-Fi chip on Android devices, which can only either be hotspot or client at the same time. With the help of a second Wi-Fi module, the one in the wearable device, we can solve this problem. In general, the framework creates a solid base of information generated by our wearable. We can use this information for similar problems without being limited to Wi-Fi or specific topics like Opportunistic Networking. A common field of research is the use of the wearables' body sensors [2]. These can be used to monitor vital signs of patients and automatically call for help in emergencies.

The contribution of this work is a library that can be integrated into present and future Android projects. The intuitive usage of this framework allows future developers to focus on realizing their idea rather than dealing with implementation difficulties to gain desired data. The implementation allows the user to access network interfaces and sensor and control element functions of the wearable and to transfer corresponding data to the mobile application. Through the various services that can be accessed, we create a broad information base on the side of the mobile phone which can be involved in future computation. This additional knowledge enables the user to make better decisions that finally improve the usability and functionality of the mobile application itself. Considering that most customer complains address functionality problems or functionality request [6], the need for our framework is emphasized.

To determine how reliable the library can be accessed, we analyze the network connecting the devices. We developed tests to show that the network withstands high loads of traffic and guarantees that the desired functions can be accessed at any time.

The remainder of this paper is structured as follows. Section II discusses fundamental knowledge. In Section III, we provide the idea as well as the demands and design of the framework, and in Section IV, we focus on the implementation in detail. Section V will take a look at the performance of the created network and how participation affects the devices. In Section VI, we give a conclusion based on the results gained and give an outlook on future work.

II. Fundamentals

To motivate our work, the first part of this section explains Opportunistic Networks in more detail. Essential for this work is the second part, in which we will define the term *The Android Wear OS Network*. Its purpose is to illustrate the structure that our mobile and wearable devices create. Furthermore, we explain which functionality the network provides using the Google Play Services.

A. Opportunistic Networks

Basic networks as we know them are mostly wired structures where a node can communicate with every other node connected to this network, no matter how far the distance between them. The topology of these networks is mostly static and hierarchical. In contrast to this idea, there are networks called Opportunistic Networks (OppNets) [7]. An OppNet is an ad-hoc network of wirelessly connected nodes in which data is shared on opportunistic contacts. The communication range is limited, and nodes can only directly connect to other nodes nearby. Thus, the topology of the network is highly flexible. This results in a possible lack of communication routes between nodes participating in a network and OppNets are therefore characterized as delay tolerant networks.

B. The Android Wear OS Network

The Android Wear OS Network describes the structure that the handset and Wear devices forge. Each of these devices is a node, and the communication link between these nodes are both Wi-Fi and Bluetooth completing the network. We will now explain how a device can take part in the network and how its task is determined.

The Google Play Service is a background service made up of a client library and a collection of different APIs which can be used by application developers. Popular features of the client library are for example *Google Maps* and *Google+*. The Google Play Service is provided by the Google Play Store and updates do not depend on carrier or OEM system image updates. This allows Google Inc. to distribute updates to their users fast, helping developers to use the latest versions of libraries and APIs for their projects. One of the APIs brought by the Google Play Service is the *WearableAPI*. Each application using this API will enter the *Wear Network*. Therefore this step is also called *Accessing the Wearable Data Layer*.

TABLE I
WEARABLE-API-COLLECTION

API-Type	Function
DataAPI	For nodes to synchronize data with all nodes
MessageAPI	For nodes to exchange messages with other nodes
ChannelAPI	For nodes to exchange large data items
NodeAPI	To learn about local or connected nodes
CapabilityAPI	To learn about capabilities provided by nodes on the Wear Network

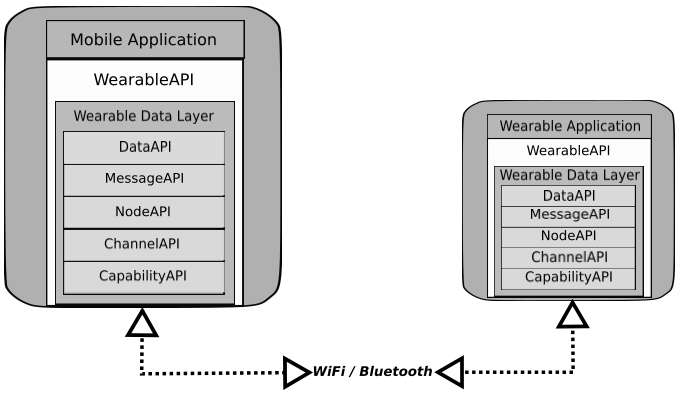


Fig. 1. Wearable-API inside the device

The *WearableAPI* is a collection of APIs that can be used for an application. Table I provides the names of the included APIs and a short description of their action performed, while Figure 1 shows the structure built up in the device. By using these APIs in our application, we determine which role our device will take in the network. Figure 2 shows a network of two wearable devices and one phone. Let's suppose that each of these devices uses the *DataAPI*. If we save a new contact in our address book and this type of information is approved to be shared, it will automatically appear on both of the user's Wear devices.

The *Cloud Node* is a service hosted by Google to allow the clients to use multiple Wear devices simultaneously. It is created regarding the used application, and the phone bounded to our Wear devices since wearable devices can be associated with a maximum of one device at a time. By entering the *Wear Network*, each device will establish a connection to this virtual node within our network directly or indirectly. The precondition for using the benefits of the hosted Cloud Node is access to the Internet. The cloud keeps track of each event within our network, like synchronization or disconnect events, and will inform all Wear devices not directly connected to the phone. It will also report to the phone about any events happening on our Wear devices that are not directly connected to it via Bluetooth. This virtual node increases the size of our *Wear Network* from a small local one to a potentially global one.

Summing up, we can see that the *Wear Network* can contain one phone and multiple Wear devices. Each network is characterized by the phone and the application. The Wear device has to be paired with the phone to be associated with the specific network. By implementing one of the APIs within

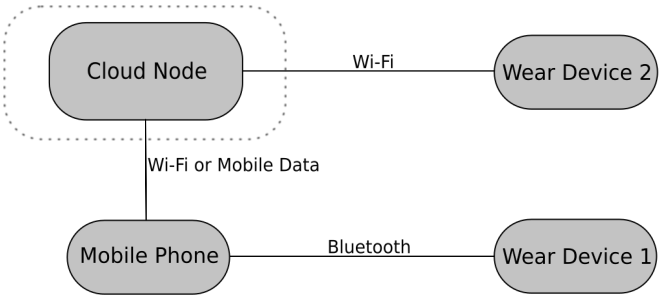


Fig. 2. WearNetwork-Example

the WearableAPI, we can access the network and determine the task each device will fulfill in the future.

III. Demands and Design

In this section, we want to reveal the reasons and goals that led to the development of the Wear OS framework. These explained objectives will shape a design for the framework whose advantages concerning user experience and functionality will be analyzed in the following. At the end of the section, we use the results of this section to present the implementation of the framework.

A. Demands

The background that motivated the development of this framework is our network application *opptain* [4]. This application builds an Opportunistic Network based on the possibilities Android offers. Installing this application will allow the device to enter this network and to start sharing data with other network participants without the need for an Internet connection. Since wearable devices are designed to perform as an extension of the mobile phone regarding improving utility, we question ourselves how wearable devices can bring these benefits to *opptain* or any other application that is based on the Android operating system.

Before any improvements can be made, we need to explore possibilities and limits that Wear devices face. The target is to collect as much information as possible from the Wear device and redirect it to the corresponding phone. This is comparable to the work of [3] in which software-, hardware-, and human-based data is collected from Android phones. Our collected information focuses on network and sensor data from Wear devices explicitly, but we also include control elements like the Wear device's touchscreen events and offer the possibility to manipulate the Wear device's network interfaces.

The result should be two applications, one running on the phone, the other on the Wear device. The hierarchy between the two devices is split into master and slave, where the phone takes the rank of the master and each Wear device bound to this phone is assigned as a slave. The mobile application will only request services on the Wear device. Furthermore, it should also work on devices running on a lower Android OS API and only need a few to no permissions. The Wear application, on the other hand, is responsible for preparing and providing the requested services and information to the mobile phone. This means that the central part of the framework is located in the Wear application. Finally, the mobile device should be holding a broad spectrum of information with the Wear device as the origin. It is then capable of analyzing and using the gained data for increased functionality.

B. Design

This section will demonstrate a design that will fulfill the demands set above by listing structures which will be used for developing both applications. Furthermore, we will specify all services provided by the Wear application that can be requested through the mobile device.

1) Wearable Application: The primary task of the wearable application is to collect information specified by the request of the mobile phone. This information collection needs to be transported reliably; therefore we will use the DataAPI. To listen for incoming request events, we will implement the MessageAPI. The idea of the Wear application is to provide a large amount of information. The type of information is listed in the following and can be divided into three groups called *Network*, *Sensor* and *Control Element*:

Network: This element of the Wear device can inform about nearby Wi-Fi networks, connect to and disconnect from specific networks and open a tethering hotspot. It can furthermore provide information about

- the current network Status
- available and bound Bluetooth hubs
- recorded audio samples.

Sensor: Each sensor shown in Table II can be requested by the phone. Additionally to these sensors, the phone can also request the GPS sensor of the wearable for updates of the current location if the device contains the necessary hardware.

Control Element: The phone can activate a control element listener on the wearable that will report to the phone if there is any interaction with the touchscreen of the Wear device.

2) Mobile Application: The mobile application will be the control center of the whole project. Since it has to request services on the wearable application, it will have a user interface, unlike the Wear application. The UI will have two tasks where the first one is to provide interaction points where a user can decide what kind of request should be sent. The second task is to display the results of the computed data received from the Wear device. The request will be forwarded with the MessageAPI. Incoming data that answers transmitted requests will be received with the help of the reliable DataAPI.

C. Advantages

To demonstrate a positive effect of the group network, we will discuss a challenge in *opptain*. Each device which is part of the *opptain* network can either be a client searching for tethering hotspots or be the hotspot itself to accept incoming client connections. Since current mobile devices only have

one Wi-Fi module, each device can only be in one state at the same time. If a device is currently transmitting a larger amount of data while being in the hotspot mode, it is not possible for the phone to scan for other hotspots. This could lead to missing a potential network member providing urgently required data. However, if using a wearable device, the task to scan for nodes can be forwarded to the Wear device. The Wear device can inform the phone about any events and suitable other hotspots, making it possible to reschedule the current transmission and switch to client mode again. Since we also provide the function to establish a complete Wi-Fi connection, our wearable could fetch the data on its own and transmit it to the phone afterward.

The possibilities introduced by sensor and control element information are quite similar. For instance, there might be an application that asks the user to reboot the system. This message could be forwarded to the wearable and be answered there. Interacting with the touchscreen would imply a yes, shaking it would imply later. There are many similar examples, and they all affect the user experience positively since we do not have to pull out the phone and unlock it for a simple confirmation.

To avoid misunderstandings, we will explain a little more detailed why we apply the recording of audio samples in the group network. Voice or sound is a collection of different frequencies, which encode the message that needs to be transmitted [1]. Since modern devices have speakers and microphones, we can provide this type of communication, but in our case, sound or ultrasound is the medium that transports meta information between the nodes in the network.

In this section, we listed the idea, the demands, and the design of our framework. Next, we will present details about the implementation of these components.

IV. Implementation

This section provides an overview of the structure of our framework by listing the components that our project is made of. Each of these components is further explained in its corresponding subsection. At the end of this section, we will have an implementation of our framework in the form of applications and a library that can be used by Android applications. We evaluate the library afterward.

A. Structure

The project consists of the three following components:

Library: This module contains all provided functionality that can be requested by the mobile application.

Wear Application: The Wear application listens for incoming service requests by the corresponding mobile phone and is responsible for starting the demanded service.

Mobile Application: The mobile application provides a graphical user interface (GUI) that allows the user to remote control the connected Wear devices and display the results of all desired and requested services.

TABLE II
Supported Sensors

Supported Sensor Type	Sensor Category
Heart Rate	Body Sensors
Acceleration, Accelerometer, Gravity, Gyroscope, Rotation, Step Counter	Motion Sensors
Game Rotation, Magnetic Field, Proximity	Position Sensors
Humidity, Light, Pressure, Temperature	Environment Sensors

B. Library

The library contains services that can be requested by the mobile device. We will take a look at each service by explaining the way it works and the data it generates.

1) SensorDataCollectorService: This service enables us to listen for sensor events. Our service creates an instance of the *SensorManager* which is connected to the sensor system service of Android. We can specify the sensors we want to keep an eye on by activating them in the mobile application.

2) BluetoothScanner: The *BluetoothScanner* is responsible for returning available and bound Bluetooth hubs surrounding the Wear device. The BluetoothScanner creates an instance of the *BluetoothAdapter* that is connected to all system services provided by Android regarding Bluetooth connectivity.

3) WifiScanner: The task of the *WifiScanner* is to report on all available wireless networks around the wearable device if demanded. Therefore, we create an instance of the *WifiManager* that allows us to access all provided Android system services that concern the Wi-Fi connectivity status of our wearable device.

4) WifiConnectionManager: The *WifiConnectionManager* is designed to manage all connectivity issues of the wearable device. It allows to connect to a local Wi-Fi network and disconnect from a current one and provides a function that returns the connection status of the Wear device. This is done by the *WifiManager* and *ConnectivityManager* class provided by Android.

5) WifiHotspotManager: The *WifiHotspotManager* is designed to turn our wearable device into a tethering hotspot. With this functionality, it is possible that the mobile and wearable device together form a network node that can be access point and client at the same time. This feature is implemented using the *WifiManager* of Android.

6) AudioRecorderManager: The *AudioRecorderManager* is designed to start recordings using the microphone of the wearable and to return these audio samples to the requesting device. The implementation is done with the *AudioRecorder* class provided by Android. With the possibility of using sound and ultrasound, we offer *opptain* and other applications another signal way to form and improve network topologies.

7) LocationScanner: The *LocationScanner* is used to determine the current position of the Wear device. Again, Android brings a system service that provides all functionality concerning the location management of the device. To access it, we create an instance of the *LocationManager* class.

8) ControlElementListener: This service is included to inform the mobile application user about touch events on the Wear device. The Android OS equips each screen with so-called *OnTouchListeners* which are used for this task.

C. Wearable Application

The wearable application is responsible for entering the *Wear Network*. It provides a reference to the communication channel that connects the device to the network. Furthermore, the wearable application serves as the connection between the mobile application and the library. Each request will be sent to the Wear application. Depending on the message received, the class satisfying the demand will be chosen from the library.

D. Mobile Application

The mobile application is, like the Wear application, responsible for entering and providing a reference to the *Wear Network*. Its duty is divided into sending requests and displaying received data recording the request. The process of requesting and receiving data is initiated through buttons deployed on the UI of the application. For example, one button will start the *LocationScanner* class on the wearable resulting in a *DataItem* with the location synchronized across the network. This information will be received and displayed on the mobile UI.

Our code, including an example application, is available at `https://github.com/hhucn/ansWEARs`. In this section, we presented the implementation of the library, the Wear applications, and the mobile application. Next, we present the evaluation of these components.

V. Evaluation

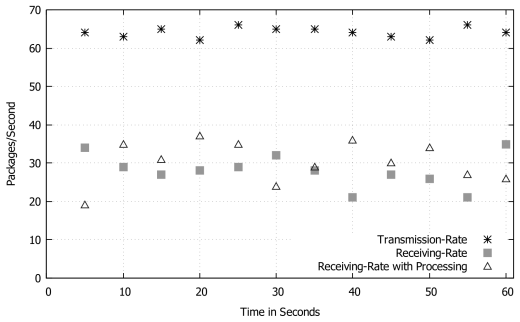
In this section, we want to present the tests designed for the applications developed in this paper and evaluate the results based on the collected data. First of all, we will introduce the test devices that we used for the evaluation. The following paragraph will explore the limitations and possibilities of the infrastructure connecting our devices and explain our test environment in detail. In a final step, we will illustrate in what way the applications will require the devices' hardware regarding the CPU, memory and battery capacities.

A. Test Devices

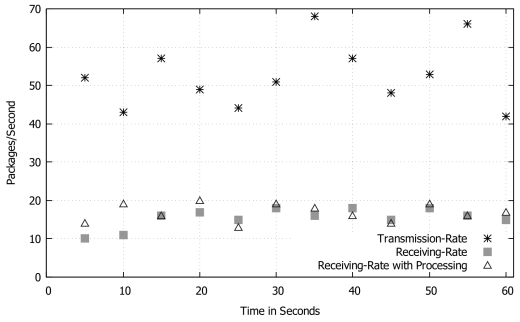
TABLE III
Devices used for Testing

Name	Model	Type	Android-Version
Samsung Galaxy S6	SM-G920F	Smartphone	6.0.1
Huawei Y3	Y360-U61	Smartphone	4.4.2
Sony SmartWatch 3	SWR50	Smartwatch	6.0.1

When choosing our test devices, we need to keep our demands regarding the Wear and mobile application in mind. Since the mobile application should contain as little functionality as possible, it will ensure that it will work correctly on many Android OS smartphones. The Wear device, on the other hand, must contain as much functionality as possible to fulfill our goal of building a broad base of information. Table III lists an exemplary selection of devices. The Sony SmartWatch was equipped with Android Wear OS 1.0. The framework has also been successfully tested on the latest Android 8.0 and Android Wear OS 2.0 devices.



(a) Synchronize 28 Byte Items



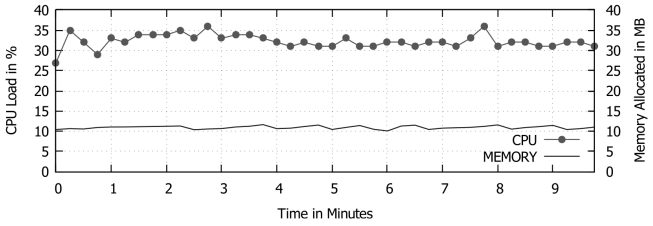
(b) Synchronize 10028 Byte Items

Fig. 3. Synchronize Items

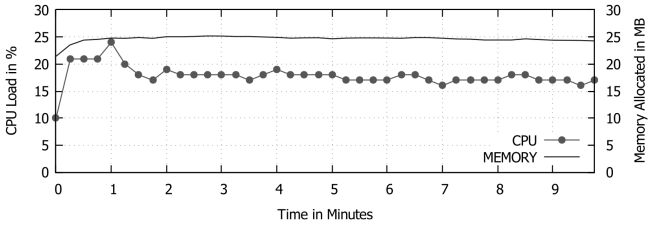
B. Network Performance

To test the performance of our network, we first need to create an event with a high frequency that occurs in a periodic time interval. To simulate an event like this, we make use of the sensors provided by our device illustrated in Table II. Each sensor event initiates the transmission of a data item, so slowly raising the number of active sensors facilitates determining how much packages can be transmitted every second. Surprisingly, the very first sensor of type *Accelerometer* exceeded the limitation that seems to be at approximately 30 packages per second. This raises the suspicion that the further processing of received data can slow down the receiving rate of the mobile device. To eliminate this suspicion, we implemented a receiver type that dropped all items right after a successful transmission. The average results from this test, illustrated in Figure 3a, show nearly no differences. In fact, the test with processing was slightly faster. This leads to the conclusion that the network is the limiting factor. Recapping the results from this test, we had an average of 28 packages per second with a package size of 28 bytes. This results in a bandwidth of 0.77 KB per second.

Since we cannot increase the number of transmissions per second, we had to increase the data item size and monitor the bandwidth again. The next experiment enlarges the data

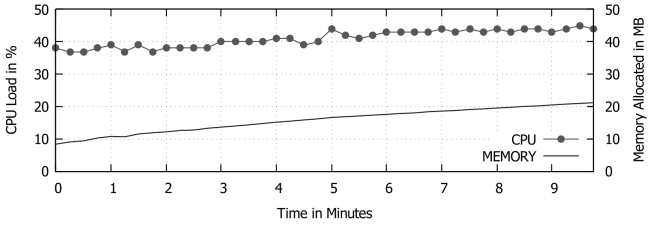


(a) Huawei Y3

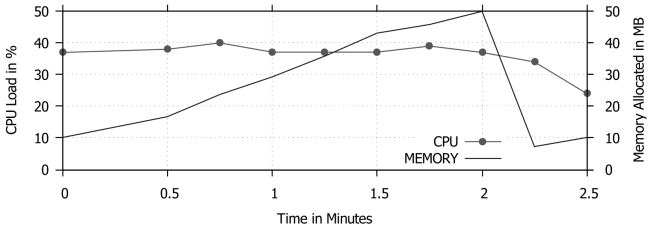


(b) Samsung Galaxy S6

Fig. 4. CPU and Memory Usage of the smartphones



(a) Sony SmartWatch3



(b) Extended CPU and Memory Usage of the smartwatch

Fig. 5. CPU and Memory Usage of the smartwatch

item size from 28 bytes to 10028 bytes. Figure 3b displays the average results of this test. Again, we distinguished between processing and not processing the received items. This time, we have an average of 16 packages per second with a bandwidth of 156.69 KB per second. Increasing the size of the data items resulted in a more than 200 times larger bandwidth, up to 220 KB per second.

From this, we can conclude that it is important to use fewer data items with a large size to reduce overhead and improve bandwidth.

C. CPU and Memory Usage

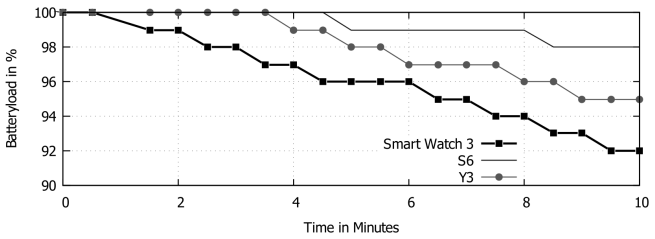
To monitor how much of the devices' hardware is used to run the applications, we again synchronize sensor data between them. This time, we protocol the CPU and memory usage on each device for ten minutes while synchronizing data. Figure 4a shows the test results for the Huawei Y3 and Figure 4b shows the results for the Samsung Galaxy S6. The Y3 has an average CPU load of about 32%, the S6 an average load of about 17%. The memory allocated by the applications is approximately 10 MB for the Y3 and 25 MB for the S6. These differences are caused by the hardware, the one in the S6 being superior to the Y3. Also interesting is the resource usage of the wearable device, as demonstrated in Figure 5a. While the CPU usage is generally around 40% during the whole test, the memory usage increases continuously. This behavior reveals a significant fact that should be kept in mind when developing for Android Wear OS. The DataAPI represents a reliable method to transport data. However, since our transmission rate is way higher than the capacity of the network to transport these packages (see Figure3a), the Wear device is forced to store the remaining synchronization events. Another negative aspect resulting from this can be the transmission of outdated sensor data. Regarding the use case for further processing, it could be essential that the sensor values are nearly live. To demonstrate how devastating it is not to keep a balance in transmission rate and package size, we reran the same test increasing the payload by 10 KB. The result can be observed in Figure 5b. This time we were only able to keep the test alive for about two minutes before no free space was available anymore.

From this, we can conclude that it is essential to balance transmission rate and package size.

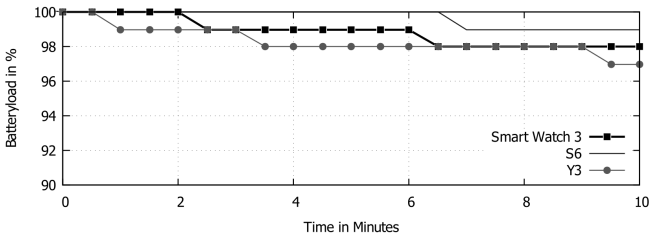
D. Battery Life

To reveal how the hardware usage influences the battery of the running device, we additionally recorded the battery consumption while logging the CPU and memory usage as described above. The results can be observed in Figure 6a. To compare this to the state in which there is no synchronization between the devices, we ran the applications doing nothing else than keeping the devices' screens unlocked for ten minutes, which is visualized in Figure 6b.

During the test run with the extended data items, we noticed that the CPU load of the SmartWatch 3 was not influenced, despite to the substantial relative growth of the package payload. This leads to the hypothesis that the primary battery consumer is not the process of computing and storing data into packages but the Bluetooth negotiation between the devices itself. Therefore, we created a new test scenario, in which the item size was increased slightly to 100 bytes to guarantee the same duration of testing without any aborts due to missing free space. Since the results regarding the battery status were precisely the same at the end of the test, despite increasing the package size and the bandwidth by the factor of three, the beforementioned hypothesis is strengthened. Reviewing this test, we could have stopped transmitting after only four minutes, since we had already forwarded the same amount of data as previously during the whole test duration displayed in Figure 6a. This would have resulted in consumption of only 3% of the device's battery instead of the prior usage of 8%.



(a) while synchronizing



(b) while not synchronizing

Fig. 6. Battery usage

E. Section Conclusion

We can conclude that current wearables are powerful and reliable data suppliers if used correctly. Therefore each connection between the devices should be used as efficiently as possible since the negotiation between devices is the leading energy consumer. By bundling information into one package, we ensure that the capacity provided by the network is used optimally, resulting in fewer connections to transport the same data. This increases bandwidth and saves battery. Our test cases consist of the most hardware intensive functions provided by the framework. Therefore the results demonstrate an upper bound regarding the hardware consumption. Table IV lists all services that are provided by the Wear application connected to their package consumption. In this context, the word *Stream* implies that once the request is fulfilled, updates need to be disabled again.

VI. Conclusion

In this paper, we have successfully created a library which allows developers and users to request and control all functions of their wearables via the mobile phone and to use additional information to improve the usability and user-friendliness of the corresponding mobile application.

For demonstration purposes, we created two applications, one running on the mobile device and one on the Wear device. The communication between the devices is defined in a way that enables the mobile application to request services on the Wear application. The Wear application, on the other hand, is responsible for answering this request by providing the desired information. The service itself, generating the information, is part of the implemented library.

Due to the different types of services, we achieved the goal of building a stable and extensive base of information that can be transmitted to the mobile application. This base of information can be used to influence and improve the functionality and user experience of mobile applications.

TABLE IV
DataItems required answering a Request

Service	DataItems	Stream
Location, Satellites, Control Elements, WiFi Networks, WiFi Status, Audio	1	No
Bluetooth	2	No
Sensors	0-60/s	Yes
Accelerometer	∼60/s	Yes

Also, we provide functions to manipulate the wireless status of our Wear device actively. This allows the developer to control the way in which data can be received.

The library is currently integrated into multiple projects which use Wi-Fi and Bluetooth communication. With the additional information provided by the library, it is now possible for these applications to prevent network transmission deadlocks.

Summing up, we want to remark that the wearable technology as a part of mobile communication is currently in the early stages. Nevertheless, the wearable devices already now provide a vast amount of opportunities that can be accessed by developers of the Android community. The release of *Android Wear OS 2.0* in June 2017 reveals the trend of future Wear devices. These will be equipped with faster hardware, an increasing number of interfaces, and the ability to operate more independently of mobile devices. We believe that wearable devices are going to be an essential part of the mobile communication future. This is the reason why integrating them into current projects is an exciting and gainful decision.

For Future Work, we will work on the transmission efficiency and more ways to exploit the network functions of wearable devices. For example, Bluetooth can be used to create an ad-hoc network of smart devices for local data transmission without the Wi-Fi infrastructure mode. Also, the storage on the wearable device can be used for Opportunistic Networking [5]. As the last point, we want to pick up our idea to use ultrasound for the transmission of metadata in these networks.

References

[1] D. Arp, E. Quiring, C. Wressnegger, and K. Rieck, "Privacy threats through ultrasonic side channels on mobile devices."

[2] P. Bonato, "Wearable sensors and systems," *IEEE Engineering in Medicine and Biology Magazine*, vol. 29, no. 3, pp. 25–36, May 2010.

[3] D. Ferreira, V. Kostakos, and A. K. Dey, "Aware: Mobile context instrumentation framework," *Frontiers in ICT*, vol. 2, p. 6, 2015. [Online]. Available: https://www.frontiersin.org/article/10.3389/fict.2015.00006

[4] A. Ippisch and K. Graffi, "Infrastructure mode based opportunistic networks on android devices," in *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, 2017, pp. 454–461.

[5] A. Ippisch, T. Küper, and K. Graffi, "Time and Space in Android-based Opportunistic Networks," in *Proceedings of the IEEE International Conference on Advanced Information Networking and Applications*, 2018.

[6] S. Mujahid, G. Sierra, R. Abdalkareem, E. Shihab, and W. Shang, "An empirical study of android wear user complaints," *Empirical Software Engineering*, Mar 2018. [Online]. Available: https://doi.org/10.1007/s10664-018-9615-8

[7] L. Pelusi, A. Passarella, and M. Conti, "Opportunistic networking: data forwarding in disconnected mobile ad hoc networks," *IEEE Communications Magazine*, vol. 44, no. 11, pp. 134–141, November 2006.