



Bachelor's Thesis

Accessing and transferring sensor data on Wearables in real-time

Zugriff und Übertragung von Sensor Daten auf Wearables in Echtzeit

by

Stephan Schultz

Potsdam, July 2016

Supervisor

Prof. Dr. Christoph Meinel,
Philipp Berger, Patrick Hennig

Internet-Technologies and Systems Group

Disclaimer

I certify that the material contained in this dissertation is my own work and does not contain significant portions of unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation.

Hiermit versichere ich, dass diese Arbeit selbständig verfasst wurde und dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt wurden. Diese Aussage trifft auch für alle Implementierungen und Dokumentationen im Rahmen dieses Projektes zu.

Potsdam, July 19, 2016

(Stephan Schultze)

Kurzfassung

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Transate abstract

An academic abstract typically outlines four elements relevant to the completed work:

- The research focus (i.e. statement of the problem(s)/research issue(s) addressed)
- The research methods used (experimental research, case studies, questionnaires, etc.)
- The results/findings of the research
- The main conclusions and recommendations

Abstract

Wearable devices such as smartwatches or activity trackers with embedded sensors are capable of exchanging data with other connected devices. This data will often be transferred to the manufacturer or processed directly on a connected smartphone in order to provide user feedback based on the analyzed data. Almost every wearable device offers third-party developers a way to gain (at least partial) access to the gathered sensor data, allowing custom applications to process them.

In the course of this work, the availability of APIs¹ on different wearables and how likely they can be used to transfer and process sensor data in real-time has been evaluated, as well as the suitability of current devices running proprietary operating systems created by Apple, Google, Jawbone and Microsoft. In addition, an in-depth look at possible implementations for real-time processing of sensor data from devices running Android Wear is included.

¹Application Program Interfaces, provided by the manufacturer

Contents

1. Introduction	1
1.1. Motivation	1
1.2. Project Scope	2
2. Related Work	3
2.1. Project Abacus	3
2.2. Papers	3
3. Devices	5
3.1. Apple Watch	5
3.2. UP by Jawbone	5
3.3. Microsoft Band	6
3.4. Android Wear	6
4. Concept	7
4.1. Apps	7
4.1.1. Wearable App	7
4.1.2. Mobile App	7
5. Implementation	8
5.1. Accessing Data	9
5.1.1. Checking Availability	9
5.1.2. Monitoring data changes	10
5.1.3. Monitoring lifecycle changes	12
5.2. Persisting Data	13
5.3. Serializing Data	14
5.4. Transferring Data	16
5.4.1. General Approach	16
5.4.2. The Wearable Data Layer	18
5.4.3. The Message API	19
5.4.4. Sending Messages	19
5.4.5. Receiving Messages	21
5.5. Putting it all together	21

6. Evaluation	22
6.1. Benchmarks	22
7. Future Work	25
7.1. Open Research Questions	25
7.2. Extensions	25
8. Conclusion	27
Bibliography	29
A. Appendix	31

1. Introduction

When speaking about mobile devices and wearables, it might not be exactly clear which kind of hardware I'm talking about.

Mobile devices are smartphones or tablets in this context. Most of the currently available mobile devices offer capabilities that could only be expected from desktop PCs a few years ago.

Wearables can be described as technology gadgets that you can wear on your body, e.g. around your wrist. Usually they are loaded with sensors and can be connected to an other mobile device via Bluetooth. The most commonly used wearables are smartwatches and fitness trackers, but the technology can also live in clothing or jewellery.

1.1. Motivation

The data produced by sensors on wearables can be valuable for multiple use-cases, for example:

- **Activity feedback:** A big selling point for wearables is the ability to track fitness activities in order to improve the health of the person wearing them. Of course this tracking works by analysing sensor data, e.g. to count the steps or track the heart rate.
- **Event triggers:** Many wearables lack large user interfaces - in order to interact with them, one can use gestures. A smartwatch can turn on its screen if you raise your wrist, but in the background a gesture detection system requires access to the device orientation and acceleration.

While some wearables can work as stand-alone devices, they usually depend on connected mobile devices to some extend. That is because the available space for hardware components is very limited, leading to a lower battery and CPU² power. However, the device sensors on wearables can produce a large amount of data every second. This data may needs to be pre-processed, analysed or

²Central processing unit

persisted - all of which requires a lot of processing power and will drain the device battery. Thus, heavy work loads like these should be done on a connected mobile device and presuppose a performant way to transfer data.

1.2. Project Scope

The goal of my Bachelor project “Passwords are Obsolete - Seamless authentication using wearables and mobile devices.” was to authenticate a user by utilizing only data from devices that the user already owns. I opted for training classifiers to analyse how a user performs certain activities and to distinguish between behaviour patterns. These classifiers take raw sensor data or extracted features and perform different machine learning methods.

This unsupervised learning approach required that a wearable can exchange data with a mobile device in a performant yet battery friendly way while keeping the delay in an acceptable range.

2. Related Work

2.1. Project Abacus

- What's different

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

2.2. Papers

- Combine abstracts

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tel-

lus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

3. Devices

I had to build prototypes for research purposes and needed to decide which wearables I intend to work with. Although I wanted to support the widest range of devices that I could, I had to ditch some in order to be able to iterate fast. In the following section, I will list the advantages and disadvantages of different wearables.

3.1. Apple Watch

While the currently available Apple Watches all provide sufficient hardware and enough sensors, the software doesn't allow 3rd party developers to take full advantage of this. With WatchOS 2, Apple restricted apps running on the watch to only get access to sensor data while it's visible to the user. For my project however, I needed a way to access sensor data from a background service, which simply isn't possible with the existing APIs. Apple announced that in WatchOS 3 (which isn't available yet), this restriction will be eliminated.

3.2. UP by Jawbone

- 2h API delay
- Can't provide required data frequency

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3.3. Microsoft Band

- Awesome, but less users than competition
- Limited to SDK functionality

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3.4. Android Wear

Android Wear is a platform for smartwatches that many devices from different manufacturers build upon. Although it's customized to match the conditions of a watch, it's still a full Android OS without any limitations. Because of Android's open nature, it's possible to use everything that the devices offer without any software restrictions.

Unlike the Apple Watch, Android Wear devices are able to connect to devices that don't belong to the same ecosystem, which increases the number of potential users.

4. Concept

As mentioned in 1, data produced by sensors on wearables needs to be transferred to mobile devices in order to be processed. All Android Wear devices are connected via Bluetooth, we will use this connection to exchange the data with the mobile device.

4.1. Apps

We need an app on every device that we want to use.

4.1.1. Wearable App

To access the sensor data, an app needs to be deployed on the wearable device. It is responsible of:

- asdf
- asdfasdf

4.1.2. Mobile App

5. Implementation

To showcase and benchmark my work, I created an Android app that visualizes sensor data from the device it runs on and also from connected Android Wear devices. The app is called Sensor Data Logger (1) and can be downloaded for free from the Google Play Store³.



Figure 1: Sensor Data Logger App

Code samples in the following sections are snippets from this project and can be seen in context in the GitHub repository⁴.

³<https://play.google.com/store/apps/details?id=net.steppschuh.sensordatalogger>

⁴<https://github.com/Steppschuh/Sensor-Data-Logger>

5.1. Accessing Data

Android provides the `SensorManager`[1] system service class in order to grant applications access to the device sensors. The supported sensors can be divided into three categories:

- **Environmental sensors** (thermometers, barometers and photometers)
- **Motion sensors** (accelerometers, gyroscopes and gravity sensors)
- **Position sensors** (magnetometers and orientation sensors)

Not all sensors are hardware components, the so called “virtual-” or “synthetic sensors” derive their data from one or more hardware-based sensors. Examples for these virtual sensors are the linear acceleration sensor, which computes its data based on the accelerometer and the gravity sensor.

All sensors can be accessed through the Android sensor framework, which provides classes and interfaces that can be used to figure out which sensors are available on the current device, which capabilities they have and what data they produce.

5.1.1. Checking Availability

While most devices have an accelerometer and a magnetometer, only a few have a thermometer. The availability of sensors can’t be guaranteed, it’s good practice to check this at runtime:

```
1 // get a SensorManager instance
2 SensorManager sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
3
4 // get a list of available sensors
5 List<Sensor> deviceSensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
6
7 // check if an accelerometer is available
8 Sensor accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
9 if (accelerometer != null) {
10     // use accelerometer
11 } else {
12     // perform error handling
13 }
```


If a sensor is available, public methods from the `Sensor`[2] class can be used to get detailed information about it. The name, vendor, version, data range and reporting delay are useful properties, especially because one device can have multiple sensors of the same type.

5.1.2. Monitoring data changes

In order to get access to the actual data, a `SensorEventListener`[3] needs to be registered at the `SensorManager` instance. The `SensorEventListener` is an interface which exposes two callback methods:

```
1 // create a new SensorEventListener
2 SensorEventListener listener = new SensorEventListener() {
3     @Override
4     public void onSensorChanged(SensorEvent event) {
5         // sensor reported new data
6     }
7
8     @Override
9     public void onAccuracyChanged(Sensor sensor, int accuracy) {
10        // sensor accuracy changed
11    }
12 };
13
14 // specify a reporting delay for the sensor
15 int delay = SensorManager.SENSOR_DELAY_NORMAL;
16
17 // register the listener for the accelerometer
18 sensorManager.registerListener(listener, accelerometer, delay);
```

The `onSensorChanged` method will be called every time the sensor updates its values. The passed `SensorEvent`[4] holds the sensor, a timestamp, the accuracy and an array of floats containing the actual values.

If the sensor accuracy changes, which often happens when using location sensors, the `onAccuracyChanged` method will be called. It can be useful to care about these accuracies, the location obtained from the Cell-ID or Wi-Fi might be more accurate than the latest GPS coordinates for example. In other cases sensors might need a few seconds to calibrate, like the magnetometer.

When registering a `SensorEventListener`, a delay in microseconds is also passed to the `SensorManager`. It's worth to notice that this value is more like a suggestion, as other applications and the system can alter it. Because the reporting delay can impact the battery life of the device, some manufacturers will lower the reporting interval when the device is idle or the display is turned off.

5.1.3. Monitoring lifecycle changes

Once a `SensorEventListener` is registered, the system will keep the requested sensor active and continue to report data, even if the user leaves the application. Hence, one should always unregister listeners as soon as possible in order to prevent battery drain:

```
1 public class SensorActivity extends Activity implements SensorEventListener {
2     private SensorManager sensorManager;
3     private Sensor accelerometer;
4
5     @Override
6     public final void onCreate(Bundle savedInstanceState) {
7         super.onCreate(savedInstanceState);
8         setContentView(R.layout.main);
9         sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
10        accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
11    }
12
13    @Override
14    protected void onResume() {
15        super.onResume();
16        sensorManager.registerListener(this, accelerometer, SensorManager.SENSOR_DELAY_NORMAL);
17    }
18
19    @Override
20    protected void onPause() {
21        super.onPause();
22        sensorManager.unregisterListener(this);
23    }
24
25    @Override
26    public final void onSensorChanged(SensorEvent event) {
27        StringBuilder log = new StringBuilder("Acceleration:");
28        log.append(" X: ").append(String.valueOf(event.values[0]));
29        log.append(" Y: ").append(String.valueOf(event.values[1]));
30        log.append(" Z: ").append(String.valueOf(event.values[2]));
31        System.out.println(log.toString());
32    }
33
34    @Override
35    public final void onAccuracyChanged(Sensor sensor, int accuracy) {
36        // sensor accuracy changed
37    }
38 }
```

This very basic example activity would print accelerometer data to the console once deployed. Keep in mind that it lacks exception handling, as mentioned in 5.1.1. I presume a basic understanding of how the Android activity[5] lifecycle works.

[Explain](#)

5.2. Persisting Data

Once a sensor is reporting data, a new `SensorEvent` will be passed to the callback every few milliseconds. The `values` float array will contain new data, however the system won't allocate a new object for every update in order to improve performance. Handling these values without knowing about its object identity might cause confusion, as they will be overwritten with every update. To prevent this, a new float array can be used to hold the event data:

```
1  @Override
2  public void onSensorChanged(SensorEvent event) {
3      // create a new float array with the same size
4      float[] values = new float[event.values.length];
5
6      // copy data from event values
7      System.arraycopy(event.values, 0, values, 0, event.values.length);
8
9      // persist values in some way
10     persistValues(values);
11 }
```

For my project, I needed to look back at sensor events from the past few seconds in order to detect patterns and to extract features. I created some helper classes[6] that allowed me to wrap sensor event data in POJOs⁵, this way they could be persisted in a batch-like structure.

There is a `Data`[7] class which can wrap the values of a `SensorEvent`, its source and a timestamp. This is necessary because the `SensorEvent` holds references to objects that aren't required multiple times and because there's no public constructor available.

⁵Plain old Java objects

The `DataBatch[8]` class holds and manages a list of `Data` objects. It has a customizable capacity, one can add or remove `Data` and it will automatically remove old `Data` if the capacity has been reached. It also provides some convenience methods, for example to get `Data` from within a given time range. The following code would fill up a `DataBatch` with event data for each requested sensor type:

```
1 private Map<Integer, DataBatch> sensorDataBatches = new HashMap<>();
2
3 @Override
4 public void onSensorChanged(SensorEvent event) {
5     float[] values = new float[event.values.length];
6     System.arraycopy(event.values, 0, values, 0, event.values.length);
7
8     // create a new Data object
9     Data data = new Data(values);
10
11     // get a previously initialized DataBatch
12     DataBatch dataBatch = getDataBatch(event.sensor.getType());
13
14     // add the new data
15     dataBatch.addData(data);
16 }
17
18 public DataBatch getDataBatch(int sensorType) {
19     DataBatch dataBatch = sensorDataBatches.get(sensorType);
20     if (dataBatch == null) {
21         dataBatch = new DataBatch(sensorType);
22         sensorDataBatches.put(sensorType, dataBatch);
23     }
24     return dataBatch;
25 }
```

5.3. Serializing Data

At some point, we have to convert the persisted data into byte arrays. We need to serialize objects in order to transfer it to another device or to write it into a file.

The most straightforward solution is using JSON⁶, which is a common data-interchange format. It is easy to read for humans and easy to parse for software, which is why we decided to use this format. Fortunately, POJOs can be directly mapped to JSON name-value pairs. Existing libraries like gson⁷ or jackson⁸ are very well known and provide interfaces that make JSON handling very uncomplicated.

The `DataRequestResponse`[9] class for example holds a list of `DataBatches` and is the responsible for exchanging sensor data with the connected mobile device. For convenience, all classes that we transfer implement methods that can be used for JSON serialization and deserialization.

```
1  @JsonIgnore
2  public String toJson() {
3      String jsonData = null;
4      try {
5          ObjectMapper mapper = new ObjectMapper();
6          mapper.enable(SerializationFeature.INDENT_OUTPUT);
7          mapper.disable(SerializationFeature.FAIL_ON_EMPTY_BEANS);
8          jsonData = mapper.writeValueAsString(this);
9      } catch (Exception ex) {
10         ex.printStackTrace();
11     }
12     return jsonData;
13 }
14
15 public static DataRequestResponse fromJson(String json) {
16     DataRequestResponse dataRequestResponse = null;
17     try {
18         ObjectMapper mapper = new ObjectMapper();
19         mapper.disable(SerializationFeature.FAIL_ON_EMPTY_BEANS);
20         dataRequestResponse = mapper.readValue(json, DataRequestResponse.class);
21     } catch (Exception ex) {
22         ex.printStackTrace();
23     }
24     return dataRequestResponse;
25 }
```

This code block is part of the `DataRequestResponse` class. The `toJson()`

⁶JavaScript Object Notation

⁷<https://github.com/google/gson>

⁸<https://github.com/FasterXML/jackson>

method writes the current object state into a JSON string, while `fromJson(String json)` creates a new `DataRequestResponse` object from a given JSON string. The JSON string can be converted to a byte array, which can be transferred as described in section 5.4. It is crucial that the same character encoding is used, we stick with UTF-8.

5.4. Transferring Data

Actually using the sensor event data requires transferring it to a device with sufficient processing power.

5.4.1. General Approach

In a world where the IoT⁹ is a big topic, data is usually uploaded to the cloud and processed on powerful servers. Although one could do that from a mobile device, this approach would produce a huge amount of traffic and ultimately lead to privacy concerns.

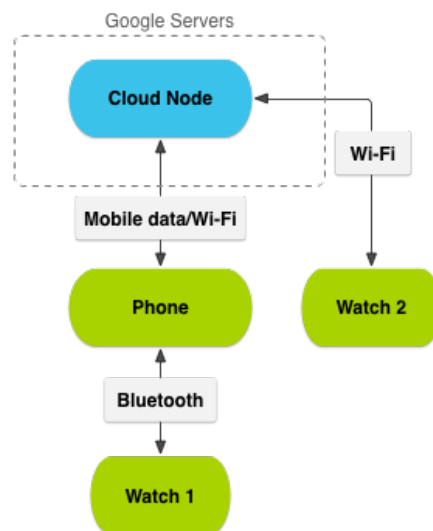


Figure 2: Sample network with mobile and wearable nodes¹⁰

⁹Internet of Things

¹⁰<https://developer.android.com/training/wearables/data-layer/>

The setup that this work is about could be seen as a peer to peer network between a mobile device and multiple wearables. Because the devices are connected with each other, there's no need to detour data through the internet. By default, wearables are connected via Bluetooth with a mobile device. On the Android platform, this connection can be accessed through the `WearableApi`[10].

5.4.2. The Wearable Data Layer

The Data Layer API is part of the Google Play Services. It contains the only APIs that should be used to set up a communication channel between a mobile device and mobile devices. Custom, low-level socket implementations are not recommended.

As shown in figure 2, the Data Layer API can handle multiple nodes at once. We are particularly interested in the Bluetooth channel, but we don't have to care about the connection because this is handled by the Play Services for us. The API can be reached using an instance of the `GoogleApiClient`[11]. Because some Play Services may not be available on every device, the `GoogleApiClient` needs to be setup first:

```
1 GoogleApiClient googleApiClient = new GoogleApiClient.Builder(context)
2   .addConnectionCallbacks(new ConnectionCallbacks() {
3       @Override
4       public void onConnected(Bundle connectionHint) {
5           // start using the Data Layer API
6       }
7       @Override
8       public void onConnectionSuspended(int cause) {
9           // something interrupted the connection
10      }
11  })
12  .addOnConnectionFailedListener(new OnConnectionFailedListener() {
13      @Override
14      public void onConnectionFailed(ConnectionResult result) {
15          // API might not be available
16      }
17  })
18  .addApi(Wearable.API)
19  .build();
```

This code block would initialize a `GoogleApiClient` instance. However, `addApi(Wearable.API)` would cause a call to `onConnectionFailed` if the device it runs on doesn't have the Android Wear app¹¹ installed. This app is required because it handles the connection and synchronization of wearables. For more graceful error handling, `addApiIfAvailable(Wearable.API)` might be the

¹¹<https://play.google.com/store/apps/details?id=com.google.android.wearable.app>

more appropriate solution.

5.4.3. The Message API

There are multiple ways of exchanging data between nodes using the Data Layer API. While the `DataApi`[12] can be used to synchronize larger binary blobs (`Assets`[13]) accross the wearable network, the `MessageApi`[14] is more suitable for exchanging smaller amounts of data. A message consists of the following items:

- **Path:** A string that uniquely identifies the message action.
- **Payload:** An optional byte array.

Payloads are not required by default because messages are a one-way communication mechanism, often used to only trigger RPCs¹². We use this to pass a serialized `DataRequest`[15] or `DataRequestResponse`[9].

5.4.4. Sending Messages

In order to send a message to a connected wearable, the `Node`[16] representation of that device is required. We can use the `NodeApi`[17] to query connected nodes:

```
1 // a list that holds available nodes
2 private List<Node> nearbyNodes;
3
4 private void updateNearbyNodes() {
5     Wearable.NodeApi.getConnectedNodes(googleApiClient)
6         .setResultCallback(new ResultCallback<NodeApi.GetConnectedNodesResult>() {
7             @Override
8             public void onResult(NodeApi.GetConnectedNodesResult nodes) {
9                 nearbyNodes = new ArrayList<Node>();
10
11                 // add all nearby nodes to the list
12                 for (Node node: nodes.getNodes()) {
13                     if (node.isNearby()) {
14                         nearbyNodes.add(node);
15                     }
16                 }
17             }
18         })
19 }
```

¹²Remote procedure calls

```
16     }
17     }
18     });
19 }
```

The `nearbyNodes` list now contains all currently connected wearables. We can get a display name and an id for each node, which we need to select our message target. For simplicity, the following code block sends a message to all nearby nodes:

```
1 private void startRequestingSensorData () {
2     // send a request message to all nodes
3     sendMessageToNearbyNodes("/start_requesting_sensor_data", null);
4 }
5
6 private void sendMessageToNearbyNodes(String path, byte[] payload) {
7     for (Node node: nearbyNodes) {
8         sendMessageToNode(node.getId(), path, payload);
9     }
10 }
11
12 private void sendMessageToNode(String nodeId, String path, byte[] payload) {
13     Wearable.MessageApi.sendMessage(googleApiClient, nodeId, path, payload)
14         .setResultCallback(new ResultCallback() {
15             @Override
16             public void onResult(SendMessageResult sendMessageResult) {
17                 if (!sendMessageResult.getStatus().isSuccess()) {
18                     // perform exception handling
19                 }
20             }
21         });
22 }
```

Note that we can pass `null` as a payload if no data is required. Instead of `null`, a serialized `DataRequest` object could be passed. The receiving nodes could deserialize it to find out which sensors are requested or at which interval they should report updates.

5.4.5. Receiving Messages

Apps running on the wearables need to implement the `MessageListener`^[18] interface in order to get notified about incoming messages. These listeners have to be registered using the `MessageApi.addListener()` function.

```
1 @Override
2 public void onMessageReceived(MessageEvent messageEvent) {
3     if (messageEvent.getPath().equals("/start_requesting_sensor_data")) {
4         // get the message payload
5         byte[] payload = messageEvent.getData();
6
7         // process the request
8         startTransferringSensorData();
9     }
10 }
```

Obviously, message paths should be static and final constants defined in a shared module that the mobile and wearable app package have access to. In our implementation, the payload would be a serialized `DataRequest`.

Explain ChannelApi

5.5. Putting it all together

asdf

Change section title

6. Evaluation

6.1. Benchmarks

- Transferred data per second
- Delay
- Serialization overhead
- Battery impact
- Different devices / API versions

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla

vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetur.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor

lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetur.

7. Future Work

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In erat mauris, faucibus quis pharetra sit amet, pretium ac libero. Etiam vehicula eleifend bibendum. Morbi gravida metus ut sapien condimentum sodales mollis augue sodales. Vestibulum quis quam at sem placerat aliquet. Curabitur a felis at sapien ullamcorper fermentum. Mauris molestie arcu et lectus iaculis sit amet eleifend eros posuere. Fusce nec porta orci.

7.1. Open Research Questions

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In erat mauris, faucibus quis pharetra sit amet, pretium ac libero. Etiam vehicula eleifend bibendum. Morbi gravida metus ut sapien condimentum sodales mollis augue sodales. Vestibulum quis quam at sem placerat aliquet. Curabitur a felis at sapien ullamcorper fermentum. Mauris molestie arcu et lectus iaculis sit amet eleifend eros posuere. Fusce nec porta orci.

Integer vitae neque odio, a sollicitudin lorem. Aenean orci mauris, tristique luctus fermentum eu, feugiat vel massa. Fusce sem sem, egestas nec vulputate vel, pretium sit amet mi. Fusce ut nisl id risus facilisis euismod. Curabitur et elementum purus. Duis tincidunt fringilla eleifend. Morbi id lorem eu ante adipiscing feugiat. Sed congue erat in enim eleifend dignissim at in nisl. Donec tortor mauris, mollis vel pretium vitae, lacinia nec sapien. Donec erat neque, ullamcorper tincidunt iaculis sit amet, pharetra bibendum ipsum. Nunc mattis risus ac ante consequat nec pulvinar neque molestie. Etiam interdum nunc at metus lacinia non varius erat dignissim. Integer elementum, felis id facilisis vulputate, ipsum tellus venenatis dui, at blandit nibh massa in dolor. Cras a ultricies sapien. Vivamus adipiscing feugiat pharetra.

7.2. Extensions

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In erat mauris, faucibus quis pharetra sit amet, pretium ac libero. Etiam vehicula eleifend bibendum.

dum. Morbi gravida metus ut sapien condimentum sodales mollis augue sodales. Vestibulum quis quam at sem placerat aliquet. Curabitur a felis at sapien ullamcorper fermentum. Mauris molestie arcu et lectus iaculis sit amet eleifend eros posuere. Fusce nec porta orci.

Integer vitae neque odio, a sollicitudin lorem. Aenean orci mauris, tristique luctus fermentum eu, feugiat vel massa. Fusce sem sem, egestas nec vulputate vel, pretium sit amet mi. Fusce ut nisl id risus facilisis euismod. Curabitur et elementum purus. Duis tincidunt fringilla eleifend. Morbi id lorem eu ante adipiscing feugiat. Sed congue erat in enim eleifend dignissim at in nisl. Donec tortor mauris, mollis vel pretium vitae, lacinia nec sapien. Donec erat neque, ullamcorper tincidunt iaculis sit amet, pharetra bibendum ipsum. Nunc mattis risus ac ante consequat nec pulvinar neque molestie. Etiam interdum nunc at metus lacinia non varius erat dignissim. Integer elementum, felis id facilisis vulputate, ipsum tellus venenatis dui, at blandit nibh massa in dolor. Cras a ultricies sapien. Vivamus adipiscing feugiat pharetra.

8. Conclusion

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In erat mauris, faucibus quis pharetra sit amet, pretium ac libero. Etiam vehicula eleifend bibendum. Morbi gravida metus ut sapien condimentum sodales mollis augue sodales. Vestibulum quis quam at sem placerat aliquet. Curabitur a felis at sapien ullamcorper fermentum. Mauris molestie arcu et lectus iaculis sit amet eleifend eros posuere. Fusce nec porta orci.

Integer vitae neque odio, a sollicitudin lorem. Aenean orci mauris, tristique luctus fermentum eu, feugiat vel massa. Fusce sem sem, egestas nec vulputate vel, pretium sit amet mi. Fusce ut nisl id risus facilisis euismod. Curabitur et elementum purus. Duis tincidunt fringilla eleifend. Morbi id lorem eu ante adipiscing feugiat. Sed congue erat in enim eleifend dignissim at in nisl. Donec tortor mauris, mollis vel pretium vitae, lacinia nec sapien. Donec erat neque, ullamcorper tincidunt iaculis sit amet, pharetra bibendum ipsum. Nunc mattis risus ac ante consequat nec pulvinar neque molestie. Etiam interdum nunc at metus lacinia non varius erat dignissim. Integer elementum, felis id facilisis vulputate, ipsum tellus venenatis dui, at blandit nibh massa in dolor. Cras a ultricies sapien. Vivamus adipiscing feugiat pharetra. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In erat mauris, faucibus quis pharetra sit amet, pretium ac libero. Etiam vehicula eleifend bibendum. Morbi gravida metus ut sapien condimentum sodales mollis augue sodales. Vestibulum quis quam at sem placerat aliquet. Curabitur a felis at sapien ullamcorper fermentum. Mauris molestie arcu et lectus iaculis sit amet eleifend eros posuere. Fusce nec porta orci.

Integer vitae neque odio, a sollicitudin lorem. Aenean orci mauris, tristique luctus fermentum eu, feugiat vel massa. Fusce sem sem, egestas nec vulputate vel, pretium sit amet mi. Fusce ut nisl id risus facilisis euismod. Curabitur et elementum purus. Duis tincidunt fringilla eleifend. Morbi id lorem eu ante adipiscing feugiat. Sed congue erat in enim eleifend dignissim at in nisl. Donec tortor mauris, mollis vel pretium vitae, lacinia nec sapien. Donec erat neque, ullamcorper tincidunt iaculis sit amet, pharetra bibendum ipsum. Nunc mattis risus ac ante consequat nec pulvinar neque molestie. Etiam interdum nunc at

metus lacinia non varius erat dignissim. Integer elementum, felis id facilisis vulputate, ipsum tellus venenatis dui, at blandit nibh massa in dolor. Cras a ultricies sapien. Vivamus adipiscing feugiat pharetra.

References

- [1] Google Developers: *Sensor manager*.
<https://developer.android.com/reference/android/hardware/SensorManager.html>.
- [2] Google Developers: *Sensor*.
<https://developer.android.com/reference/android/hardware/Sensor.html>.
- [3] Google Developers: *Sensor event listener*.
<https://developer.android.com/reference/android/hardware/SensorEventListener.html>.
- [4] Google Developers: *Sensor event*.
<https://developer.android.com/reference/android/hardware/SensorEvent.html>.
- [5] Google Developers: *Activity*.
<https://developer.android.com/reference/android/app/Activity.html>.
- [6] Stephan Schultz: *Sensor data logger: Data package*.
<https://github.com/Steppschuh/Sensor-Data-Logger/tree/master/Android%20Source/datalogger/src/main/java/net/steppschuh/datalogger/data>.
- [7] Stephan Schultz: *Sensor data logger: Data*.
<https://github.com/Steppschuh/Sensor-Data-Logger/blob/master/Android%20Source/datalogger/src/main/java/net/steppschuh/datalogger/data/Data.java>.
- [8] Stephan Schultz: *Sensor data logger: Data batch*.
<https://github.com/Steppschuh/Sensor-Data-Logger/blob/master/Android%20Source/datalogger/src/main/java/net/steppschuh/datalogger/data/DataBatch.java>.
- [9] Stephan Schultz: *Sensor data logger: Data request response*.
<https://github.com/Steppschuh/Sensor-Data-Logger/blob/master/Android%20Source/datalogger/src/main/java/net/steppschuh/datalogger/data/request/DataRequestResponse.java>.





- [10] Google Developers: *Wearable api*.
<https://developers.google.com/android/reference/com/google/android/gms/wearable/Wearable>.
- [11] Google Developers: *Google api client*.
<https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient>.
- [12] Google Developers: *Data api*.
<https://developers.google.com/android/reference/com/google/android/gms/wearable/DataApi.html>.
- [13] Google Developers: *Asset*.
<https://developers.google.com/android/reference/com/google/android/gms/wearable/Asset>.
- [14] Google Developers: *Message api*.
<https://developers.google.com/android/reference/com/google/android/gms/wearable/MessageApi>.
- [15] Stephan Schultz: *Sensor data logger: Data request*.
<https://github.com/Steppschuh/Sensor-Data-Logger/blob/master/Android%20Source/datalogger/src/main/java/net/steppschuh/datalogger/data/request/DataRequest.java>.
- [16] Google Developers: *Node*.
<https://developers.google.com/android/reference/com/google/android/gms/wearable/Node>.
- [17] Google Developers: *Node api*.
<https://developers.google.com/android/reference/com/google/android/gms/wearable/NodeApi>.
- [18] Google Developers: *Message listener*.
<https://developers.google.com/android/reference/com/google/android/gms/wearable/MessageApi.MessageListener>.

A. Appendix

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In erat mauris, faucibus quis pharetra sit amet, pretium ac libero. Etiam vehicula eleifend bibendum. Morbi gravida metus ut sapien condimentum sodales mollis augue sodales. Vestibulum quis quam at sem placerat aliquet. Curabitur a felis at sapien ullamcorper fermentum. Mauris molestie arcu et lectus iaculis sit amet eleifend eros posuere. Fusce nec porta orci.

Integer vitae neque odio, a sollicitudin lorem. Aenean orci mauris, tristique luctus fermentum eu, feugiat vel massa. Fusce sem sem, egestas nec vulputate vel, pretium sit amet mi. Fusce ut nisl id risus facilisis euismod. Curabitur et elementum purus. Duis tincidunt fringilla eleifend. Morbi id lorem eu ante adipiscing feugiat. Sed congue erat in enim eleifend dignissim at in nisl. Donec tortor mauris, mollis vel pretium vitae, lacinia nec sapien. Donec erat neque, ullamcorper tincidunt iaculis sit amet, pharetra bibendum ipsum. Nunc mattis risus ac ante consequat nec pulvinar neque molestie. Etiam interdum nunc at metus lacinia non varius erat dignissim. Integer elementum, felis id facilisis vulputate, ipsum tellus venenatis dui, at blandit nibh massa in dolor. Cras a ultricies sapien. Vivamus adipiscing feugiat pharetra.

Notes

 Transate abstract	iv
 Explain	13
 Explain ChannelApi	21
 Change section title	21