Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Student Research Project Nr. 2428

Mobilelogging: Assessing Smartphone Sensors for Monitoring Sleep Behaviour

Sven Andre Mayer

Course of Study: Computer Science

Examiner: Prof. Dr. Albecht Schmidt

Supervisor: M.Sc. Alireza Sahami, Dr. Niels Henze

Commenced: 2013-01-01

Completed: 2013-09-30

CR-Classification: I.5.2

Abstract

This work deals with mobile devices, more specifically with Android smartphones. Smartphones have more and more accurate sensor data. This we can use to make us a more accurate picture of our trade. Here, the techniques are demonstrated on the present state of the art show. Using an example is shown how this works in detail data recording. It is a user study carried out over 7 days. The goal is to determine whether the data can be analysed sleep behaviour by the smartphone. To control a commercial objective sleep measurement device is used.

Contents

1 Introduction

Smart phones turn more and more into personal computing devices with a wide variety of applications that support users during everyday life. The ubiquity of the mobile phones allows people to use them in any context. The proliferation of mobile devices in everyday life leads to an increasing amount of information about users' personal contexts that can be obtained automatically and made available to third-party applications. In this project we aim to develop an application for Android devices, which logs all possible context information, can be retrieved. To achieve the goal, the potential context information is recognized in the first step. Then an application is developed to log the information and store it in a central database. To fill this central database it is necessary to collect data from different people with different age, gender and jobs to have a wide range of different moving behavior. This will happen in a user study. Every user will be record his context over one week. All data in our central database will give us an idea of the user's behavior and about his day cycle. The aspect we want to look at the data is the night sleep. Night sleep will here define as a sleep longer than 5 hours between 10 p.m. and 10 a.m. We are all people and people need to sleep, so in every day cycle there has to be a sleep time. The next aim it will be to extract the sleep time out of the user's data. But the question is: are there data the truth? To answer this question we need have something like a "truth recorder". This has to be a device we can give to all study members to record their precise sleep time, start and end while they are recording their data with the mobile device. With the precise sleep data and the mobile device context data we can analyze differences, anomalies and similarities in the night sleep. This will be the main goal for this work, to make clear if there is a signified similarity between humans sleep and their mobile device contexts. Another interesting information we want to get is with context information increases the accuracy of the difference in the data.

2 Context Data on Android Devices

Now we want to give an overview out data logging on an Android device. Android provides two options ready to pick off phone context. One is the Sensor-Class and the other is the BroadcastReceiver-Class. The difference between these classes is very simple. Sensor-Class check for changed values in a defined time interval and send if it is changed. BroadcastReceiver-Class only sends values if the value has changed.

2.1 Sensor-Class

The Sensor-Class is available since API level 3. Before API level 3, the Sensors-Class was stored in "android.hardware.SensorManager". Here we will describe to setup a Sensor since API Level 3. Table 2.1 shows all possible sensors Android provide in the Sensor-Class.

To use the Sensor-Class you class need to implements "android.hardware.SensorEvent-Listener". This implementation requires two methods "void onAccuracyChanged(Sensor sensor, int accuracy)" and "void onSensorChanged(SensorEvent event)". Both methods will be called by the AndroidOS. onAccuracyChanged will be called when the accuracy of a sensor has changed. onSensorChanged will be called when sensor values have changed.

2.1.1 Register a Sensor

If we have implements the "SensorEventListener" we can register one or more Sensors. Listing 2.1 shows the code to register a "Sensor.TYPE_ACCELEROMETER" sensor. Table 2.1 shows us all sensors we can register here. With the third parameter of the method "registerListener" its possible we set an update rate of the sensor. The are four delay time settings shown in table 2.2. Through the Java source files we have found that each delay times are assigned to a specific value. The table 2.3 shows the theoretical times. In the Java source file the delay time is given in µs. For "SENSOR_DELAY_FASTEST" i need to say that a 0 delay time is not possible, so what it mean that you got the lowest delay time your hardware sensor in your phone supports. With low delay times appears a problem. The problem is that the sequence of events is not necessarily sorted chronologically. This means that must be specially checked whether the sequence is correct.

```
1 SensorManager sensorManager = (SensorManager) this.getApplicationContext().
      getSystemService(Context.SENSOR_SERVICE);
2 Sensor sensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER
      );
```

```
3 sensorManager.registerListener(this, sensor, SensorManager.
      SENSOR_DELAY_NORMAL);
```

Listing 2.1: Registration of a sensor

If we register two or more sensors in one class its necessary to check which one called "onSensorChanged" like we did in listing 2.2.

```
1  @Override
2  public void onSensorChanged(SensorEvent event)
3  {
4      switch (event.sensor.getType ()){
5          case Sensor.TYPE_ACCELEROMETER:
6              doAccelerometer(event.timestamp, event.values);
7              break;
8          case Sensor.TYPE_MAGNETIC_FIELD:
9              doMagneticField(event.timestamp, event.values);
10             break;
11     }
12 }
```

Listing 2.2: How the onSensorChanged function is build

There are two other things to pay attention to the use of onSensorChanged. The parameter "event" of type "SensorEvent" has two fields that are important: "timestamp" and "values".

First we have the "values" variable, its a float array. Different sensors have different lengths of the array and various units, see table 2.4.

Second event offers the value "timestamp". But be care full its not a timestamp we would expect. The value is not given in ms an not since 01.01.1970. The value is the time in nanosecond at which the event happened and counts from the start up time of the phone. This means that 0ns is the start time of the phone. Listing **??** shows how we have a calculation to come from timestamp start up time to milliseconds till 1970. It is not perfect because "new Date()" and "SystemClock.uptimeMillis()" is called after the event happens but it is the best of what you can do. Through my recherche can say, "timestamp" is counted in nanoseconds since the system was booted. This clock stops when the system enters deep sleep (CPU off, display dark, device waiting for external input), but is not affected by clock scaling, idle, or other power saving mechanisms. So you need a similar clock if you wand to calculate something with the event.timestamp, we can use System.nanoTime() for nanosecond calculation and for milisecoind calulation SystemClock.uptimeMillis(). For my calculation in listing 2.3 i also could use System.nanoTime() but the the Android specification said NanoTime() "cannot be used as a very exact system time expression." [1] So we decided to use uptimeMillis() because our result value is also milliseconds. This method calculates always a time for approximately two ms before nanoTime result is.

Value	Title	Description
int	TYPE_ACCELEROMETER	A constant describing an accelerometer sensor type.
int	TYPE_ALL	A constant describing all sensor types.
int	TYPE_AMBIENT_TEMPERATURE	A constant describing an ambient temperature sensor type
int	TYPE_GRAVITY	A constant describing a gravity sensor type.
int	TYPE_GYROSCOPE	A constant describing a gyroscope sensor type
int	TYPE_LIGHT	A constant describing a light sensor type.
int	TYPE_LINEAR_ACCELERATION	A constant describing a linear acceleration sensor type.
int	TYPE_MAGNETIC_FIELD	A constant describing a magnetic field sensor type.
int	TYPE_ORIENTATION	This constant was deprecated in API level 8. use SensorManager.getOrientation() instead.
int	TYPE_PRESSURE	A constant describing a pressure sensor type.
int	TYPE_PROXIMITY	A constant describing a proximity sensor type.
int	TYPE_RELATIVE_HUMIDITY	A constant describing a relative humidity sensor type.
int	TYPE_ROTATION_VECTOR	A constant describing a rotation vector sensor type.
int	TYPE_TEMPERATURE	This constant was deprecated in API level 14. use Sensor.TYPE_AMBIENT_TEMPERATURE instead.

Table 2.1: Constants at the Sensor Class [2]

```
1 long timeInMillis = (new Date()).getTime() + (event.timestamp - System.nanoTime
      ()) / 1000000L;
```

Listing 2.3: Sensor start up nano time to milisec since 1970

Generally also SystemClock.uptimeMillis() has the same properties as System.nanoTime(). There is only a tiny difference, but we will never be noticed. Nano-time and milli-time is both stored in a long value. Because nano-time is much more accurate the value will reach his maximum faster. After reaching the maximum Long will flip around and runs again to the maximum. This phenomenon is reached after about 292.5 million years, so we must be no thoughts.

2.2 BroadcastReceiver-Class

BroadcastReceiver works since API level 1. There are a wide range of events that you can retrieve via a broadcast receiver. The Table 2.5 shows all possible values you can get with the BroadcastReceiver-Class. In fact, not all are really useful. There was

Value	Title	Description
int	SENSOR_DELAY_FASTEST	get sensor data as fast as possible
int	SENSOR_DELAY_GAME	rate suitable for games
int	SENSOR_DELAY_NORMAL	rate (default) suitable for screen orientation changes
int	SENSOR_DELAY_UI	rate suitable for the user interface

Table 2.2: Constants for delay time at the SensorManager Class [2]

Title	µs	ms	Hz
SENSOR_DELAY_FASTEST	0	0	0
SENSOR_DELAY_GAME	20000	20	50
SENSOR_DELAY_NORMAL	66667	66	15
SENSOR_DELAY_UI	200000	200	5

Table 2.3: Sensor delay times with real values [2]

Value	Length	Si units	Description
TYPE_ACCELEROMETER	3	m/s^2	0,1,2 = x, y, z-axis
TYPE_AMBIENT_TEMPERATURE	1	$°C$	
TYPE_GRAVITY	3	m/s^2	0,1,2 = x, y, z-axis
TYPE_GYROSCOPE	3	radians/s	0,1,2 = x, y, z-axis
TYPE_LIGHT	1	lux	
TYPE_LINEAR_ACCELERATION	3	m/s^2	0,1,2 = x, y, z-axis
TYPE_MAGNETIC_FIELD	3	µT	0,1,2 = x, y, z-axis
TYPE_ORIENTATION	3	Degree	deprecated in API level 8
TYPE_PRESSURE	1	hPa	
TYPE_PROXIMITY	1	cm	
TYPE_RELATIVE_HUMIDITY	1	%	
TYPE_ROTATION_VECTOR	3 or 4		
TYPE_TEMPERATURE	1	$°C$	deprecated in API level 14

Table 2.4: Constants at the Sensor Class [2]

even canceled an event from the list, with the Introduction of API level 14 ACTION_ PACKAGE_INSTALL was removed.

Table 2.5: Constants at the Intent Class for BroadcastReceiver [3]

Length	**Description**
ACTION_AIRPLANE_ MODE_CHANGED	The user has switched the phone into or out of Airplane Mode.
ACTION_BATTERY_ CHANGED	This is a sticky broadcast containing the charging state, level, and other information about the battery.
ACTION_BATTERY_LOW	Indicates low battery condition on the device.
ACTION_BATTERY_OKAY	Indicates the battery is now okay after being low.
ACTION_BOOT_ COMPLETED	This is broadcast once, after the system has finished booting.
ACTION_CAMERA_ BUTTON	The "Camera Button" was pressed.
ACTION_CONFIGURATION _CHANGED	The current device Configuration (orientation, locale, etc) has changed.
ACTION_DEVICE_ STORAGE_LOW	A sticky broadcast that indicates low memory condition on the device This is a protected intent that can only be sent by the system.
ACTION_DEVICE_ STORAGE_OK	Indicates low memory condition on the device no longer exists This is a protected intent that can only be sent by the system.
ACTION_DOCK_EVENT	A sticky broadcast for changes in the physical docking state of the device.
ACTION_DREAMING_ STARTED	Sent after the system starts dreaming.
ACTION_DREAMING_ STOPPED	Sent after the system stops dreaming.
ACTION_EXTERNAL_ APPLICATIONS_AVAILABLE	Resources for a set of packages (which were previously unavailable) are currently available since the media on which they exist is available.
ACTION_EXTERNAL_ APPLICATIONS_ UNAVAILABLE	Resources for a set of packages are currently unavailable since the media on which they exist is unavailable.
ACTION_GTALK_SERVICE_ CONNECTED	A GTalk connection has been established.
ACTION_GTALK_SERVICE_ DISCONNECTED	A GTalk connection has been disconnected.
ACTION_HEADSET_PLUG	Wired Headset plugged in or unplugged.

Continued on next page

Table 2.5 – continued from previous page

Length	Description
ACTION_INPUT_METHOD_ CHANGED	An input method has been changed.
ACTION_LOCALE_ CHANGED	The current device's locale has changed.
ACTION_MANAGE_ PACKAGE_STORAGE	Indicates low memory condition notification acknowledged by user and package management should be started.
ACTION_MEDIA_BAD_ REMOVAL	External media was removed from SD card slot, but mount point was not unmounted.
ACTION_MEDIA_BUTTON	The "Media Button" was pressed.
ACTION_MEDIA_ CHECKING	External media is present, and being disk-checked The path to the mount point for the checking media is contained in the Intent. mData field.
ACTION_MEDIA_EJECT	User has expressed the desire to remove the external storage media.
ACTION_MEDIA_ MOUNTED	External media is present and mounted at its mount point.
ACTION_MEDIA_ REMOVED	External media has been removed.
ACTION_MEDIA_ SCANNER_FINISHED	The media scanner has finished scanning a directory.
ACTION_MEDIA_ SCANNER_SCAN_FILE	Request the media scanner to scan a file and add it to the media database.
ACTION_MEDIA_ SCANNER_STARTED	The media scanner has started scanning a directory.
ACTION_MEDIA_SHARED	External media is unmounted because it is being shared via USB mass storage.
ACTION_MEDIA_ UNMOUNTABLE	External media is present but cannot be mounted.
ACTION_MEDIA_ UNMOUNTED	External media is present, but not mounted at its mount point.
ACTION_MY_PACKAGE_ REPLACED	A new version of your application has been installed over an existing one.
ACTION_NEW_ OUTGOING_CALL	An outgoing call is about to be placed.
ACTION_PACKAGE_ADDED	A new application package has been installed on the device.
ACTION_PACKAGE_ CHANGED	An existing application package has been changed (e.g.

Continued on next page

Table 2.5 – continued from previous page

Length	Description
ACTION_PACKAGE_DATA_ CLEARED	The user has cleared the data of a package.
ACTION_PACKAGE_ FIRST_LAUNCH	Sent to the installer package of an application when that application is first launched (that is the first time it is moved out of the stopped state).
ACTION_PACKAGE_ FULLY_REMOVED	An existing application package has been completely removed from the device.
ACTION_PACKAGE_ NEEDS_VERIFICATION	Sent to the system package verifier when a package needs to be verified.
ACTION_PACKAGE_ REMOVED	An existing application package has been removed from the device.
ACTION_PACKAGE_ REPLACED	A new version of an application package has been installed, replacing an existing version that was previously installed.
ACTION_PACKAGE_ RESTARTED	The user has restarted a package, and all of its processes have been killed.
ACTION_PACKAGE_ VERIFIED	Sent to the system package verifier when a package is verified.
ACTION_PROVIDER_ CHANGED	Some content providers have parts of their namespace where they publish new events or items that the user may be especially interested in.
ACTION_REBOOT	Have the device reboot.
ACTION_SCREEN_OFF	Sent after the screen turns off.
ACTION_SCREEN_ON	Sent after the screen turns on.
ACTION_SHUTDOWN	Device is shutting down.
ACTION_TIMEZONE_ CHANGED	The timezone has changed.
ACTION_TIME_CHANGED	The time was set.
ACTION_TIME_TICK	The current time has changed.
ACTION_UID_REMOVED	A user ID has been removed from the system.
ACTION_USER_PRESENT	Sent when the user is present after device wakes up (e.g when the keyguard is gone).

2.2.1 Register a BroadcastReceiver

To produce a particular broadcast receiver, we have to create a new class that is derived from the "BroadcastReceiver" types. Listing 2.4 shows the basic skeleton of a class that is derived from "broadcast receiver". The onReceive function is called by the system when the result previously defined occurs. In this function, the code must be placed to be executed when the event occurs. To really take advantage of a "Broadcast Receiver"

derived class, it must be registered with the system. The listing 2.5 shows a simple way a "Broadcast Receiver" to log on to system. In this example, "ACTION_SCREEN_ON" and "ACTION_SCREEN_OFF" are used as a filter. In this case, all the filters listed in table 2.5 can be used. If one of the events which have been specified as the filter occurs, the associated respective class is activated, and called onReceive. There is one more thing that needs to be done to get an event from the system, an entry into the "manifest.xml" file. "<Receiver android:name="PAKETNAME.MyBroadcastReceiver"></ receiver> "must be inserted into the the manifest. The Listing 2.6 shows the exact place where the tag has to be inserted

```
1  public class MyBroadcastReceiver extends BroadcastReceiver {
2
3      public MyBroadcastReceiver(){
4          super();
5      }
6
7      @Override
8      public void onReceive(Context context, Intent intent) {
9          //some stuff we want to do at receiving
10     }
11 }
```

Listing 2.4: basic skeleton of a class that is derived from "broadcast receiver"

```
1  private void addReceiver (Context context){
2      IntentFilter filter = new IntentFilter(Intent.ACTION_SCREEN_ON);
3      filter.addAction(Intent.ACTION_SCREEN_OFF);
4      //here you can add more Intents for BroadcastReceiver
5      MyBroadcastReceiver receiver = new MyBroadcastReceiver();
6      context.registerReceiver(receiver, filter);
7  }
```

Listing 2.5: Add a BroadcastReceiver to the System

Listing 2.6: Add MyBroadcastReceiver-Class to the Android-Manifest

```
1  <manifest ...>
2      ...
3      <application ...>
4          ...
5          <receiver android:name="PAKETNAME.MyBroadcastReceiver">
6          </receiver>
7          ...
8      </application>
9      ...
10 </manifest>
```

3 Sleep Tracking with Commercial Devices

Out there are a huge number of devices we could use. But the range of expenditure is also huge. The minimum of expenditure is to let the user write down his sleep time and the maximum of expenditure is to use a sleep laboratory. Therefore, we must analyze various techniques on the effort, the accuracy and quality of the output data.

For me a very important requirement is that the participants will not be disturbed in their daily routine. The interaction with the sleep detection must be very low. The reason: every interaction changes the behavior. To determine the relationship between sleep and smart phone use, a non-altered behavior are based.

3.1 Sleep Laboratory

This is a medical facility to be examined carefully the sagging hold. This goes far beyond the needs of my lap recording sleep. In addition, the daily behavior of the subjects is changed because they no longer allowed to sleep at home.

The price for this is unprofitable high. For this reason, and the reason of changed behavior, ruled out this possibility from the outset.

3.2 FitBit

A great advantage to Fitbit is the Fitbit API. They can be accessed with .Net, Java and PHP. This API works with OAuth authentication. Thus, it is imperative that the server used dominates this. The output format can be selected here, either JSON or XML. Furthermore, we will describe how to read certain data with PHP.[4] To record the sleeping behavior, this device must be attached to the arm and be put in sleep mode.

The price of a Fitbit is with 99 € at the bottom of the possible devices.

3.3 Withings Pulse

Withings Pulse has a slightly less large API. It works very similar to the Fitbit API, they also used a OAuth authentication. But it works only with PHP and it gives only the JSON output format. Thus we have the same requirements as for a FitBit here the use of PHP. [7] To record the sleeping behavior, this device must be attached to the arm and be put in sleep mode.

The price for one of these devices is € 99. And so it is with fitbit on the same.

3.4 Jawbone

Jawbone works like Withings Pulses with JSON. It does not work on OAuth, but with HTTP request. This method is not as safe for the end user. But you can use any programming language to understand the HTTP request. [6] Since I always wear this device on the arm where we do not have to entrench it at night. In addition, no sleep mode must be activated here.

Unfortunately, the price is more expensive than the competition, there are € 129 for a device.

3.5 User Inclusion

Here the idea is that each participant independently monitors its sleep phase. With this method two problems go hand in hand. One is the act of disturbing the sleep, because the test person must make sure to write down the correct time. And second, that the record is inaccurate.

The cost of this method are of course at 0 €.

3.6 The decision

First, we want to exclude the method can not find a park your app tables, sleep laboratory. If the goal is to monitor sleep habits, then we can not change this in advance. This issue we have with the self-monitoring method. The user must determine when it actually goes to sleep, but sleep normal occurs after a few minutes. Thus, the exact sleep time is not recorded. Instead, we only have a rough idea when the person wants to sleep.

Thus, the presented above devices remain. For this project, the safety of non-significant, but applies to other projects to consider this: While jawbone has a great possibility to use, but is very insecure in the transmission. This mean that the major negative point of jaw bone are the cost. FitBit and Withings pulses are equal in price and user-on. Both need to be fix on your arm at night and must also be put in the sleep mode. But both are € 30 cheaper than the Jawbone device.

The remaining decision is based on my personal feeling. Since my stay calculations to sleep recognition of the individual devices hidden, we have to believe this blind. Since we want to use the values for a larger purpose, it is in my opinion an advantage if we do not give the calculations completely out of hand. We achieve this so that we with the type of device. By activating the sleep mode, the calculation will start. So we have created a small control over the correctness of calculation. So we have the choice narrowed down to two, FitBit and Withings Pulses.

This final decision is based on the functionality and the functionality. Simply put, FitBit can be used more versatile. That is now in the course of a study, each participant will receive a Fitbit.

4 Implementation of an Android App

The app we are created for Andorid devices with minimum version number is 2.2.3. Since 2.0 there is a wide range of sensors to obtain data. Here is a list of the context we recorded:

- Accelerometer
- Air plain mode on/off
- App use
- Magnetic field
- Screen on/off
- Charger plugged
- Battery state changed
- Ambient volume
- Ambient light
- Proximity over Display
- Wireless service

For the server communication we need to have a unique id on each device. It will be calculated on each device by there one out of unique device information. This is necessary to map the data to the appropriate member.

We record all this sensor data by a service, so its hidden from the user and running all the time. One sensor is stored in one folder, for each record minute one file. Each file is a comma-separated values file, each line is one data set. In a line it comes a time stamp first and then separated the values. This is necessary to keep the volume to outline at the huge data.

For each sensor, we have the same life cycle, this is shown in figure 4.1. The cycle starts with the create a new file in which the data should be stored. At the beginning of an additional 60-second timer is initialized and started. Then waits for sensor data, and the expiration of the timer. If sensor data arrive, they are stored sequentially in the file. If the timer sends an interrupt, jumps back to the beginning, creates a new file and the timer is restarted.

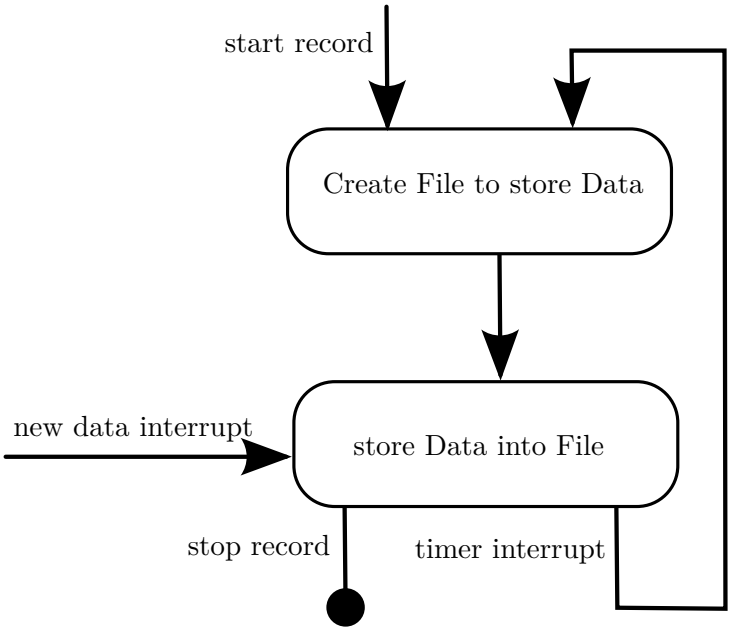


Figure 4.1: The cycle of one sensor

4.1 App Server Communication

Every 60 sec the app is trying to send the files to the server. With each file it send the unique id to now witch user sends the file. The file is sent via HTTP. If the smartphone is not online or the connection interrupts the app tries to send the files 60 sec later again. On the server, the data will only be accepted and stored in a similar folder structure. On the server is just one php script, see listing 4.1.

Listing 4.1: PHP code on the server

```
1  <?php
2  $uniqueid = $_POST['uniqueid'];
3  $eventname = $_POST['eventname'];
4  if (!empty($uniqueid) && !empty($eventname)){
5      $destination = "./uploads/".$uniqueid."/".$eventname."/";
6      if (!is_dir ($destination)){
7          if (!mkdir($destination, 0775, true)) {
8           echo('Dir wasnt create, please try again!');
9          }
10     }
11     $destination += basename( $_FILES['data']['name']);
12     if(move_uploaded_file($_FILES['data']['tmp_name'], ↩
           $destination))
```

```
13     {
14         echo "The file ".basename( $_FILES['data']['name']) ." ↩
               has been uploaded";
15         exit(200);
16     } else {
17         echo "There was an error uploading the file , please try↩
               again!";
18         exit(400);
19     }
20 } else {
21     echo "Not all required vars set.";
22     exit(406);
23 }
24 ?>
```

4.2 Problems

Over the time there came a big problem up, its about gaps in the recorded data. A service has the great advantage that it works all the time in the background. But the time has turned out that that's not quite. Some options are stressful for the system in these cases the service is paused. So we try a lot of different operations and parameters to use to charge the minimum to System. We were able to minimize the gaps in Recorded data, but unfortunately not completely prevent. The gap size depends on 2 factors from the computing power of smartphones and the impact of smartphones by users. Since we already knew about the problem in the run with the gaps modest, our biggest was hopeful that there will be sufficient data are useful.

5 User Study

To get a confirmation whether it is possible to get acceptable values through our app we made for the user study. Each user should run the app on his mobile phone and at the same time use a commercial device to record data. The commercial device used to check the data recorded by the app. The study focused mainly on the question: we can identify the sleep time using the mobile phone usage. For this reason, we set the duration of the study on seven full days, so we can evaluate seven nights of each participant. For a first analysis we selected six people, three of whom were female and three male. Not one of them was a computer scientist, we have to assume that these people use their mobile phones more often than not one of the lancer has to do with this topic. Our hope was to get a large spectrum of different behaviors despite the small number of participants.

5.1 Questionnaire

To verify that we have a wide range of different people, all people at any one have to answer some questions. The answers are shown in Table 5.1.

5.2 Progress

During the study, we have noticed that we have problems. Why we sent 3 times an update to the participating users, more in section 5.3.1. Because we thought we had solved the problem, which unfortunately was not the case. The last version we had then record 7 days. During this time the participants were four times Fitbit forget to Activate. This was clear from the beginning that this error be made.

Otherwise, the study went through without problems until the end. Everyone has to get a briefing in Fitbit and the app. At the app there is nothing to do, so it certainly have no problems. And the use of Fitbit has also worked without further between ask.

It was only after the end of the second problem is noticed. What then was not revisable, more in section 5.3.2.

5.3 Problems

5.3.1 Gap Problem

From the beginning of the study, we knew that there will be gaps in the record. The two factors for gaps are too little computing capacity and large user loads. These two

	P1	P2	P3	P4	P5	P6
Age?	24	25	24	21	48	24
gender?	f	m	m	f	f	m
Job?	Student	Electronics technician for automation technology	Student	justice professional employees	Physical therapist	Merchant in wholesale and foreign trade
Use your cell phone as an alarm clock?	Yes	No	Yes	Yes	Yes	Yes
Imagine an alarm clock on the weekend?	Yes	Yes	No	No	No	No
How do you make your alarm clock?	daily	daily	daily	daily	daily	daily
How intensively you use their phone every day?	Intensive	Intensive	Normal	Intensive	Normal	Intensive
How many hours do you sleep on average?	6,5	7,5	7,8	7	7	7
My sleep was ever monitored?	No	No	No	No	No	No
How did you sleep professionally monitored?	No	No	No	No	No	No
If you have trouble sleeping?	Yes	Yes	No	No	No	No
You only use their cell phone?	Yes	Yes	Yes	Yes	Yes	Yes
If your sleep monitored by experts?	No	No	No	No	No	No
You know what 'Fitbit?						
Have you already used Fitbit?	No	No	No	No	No	No
Are you in possession of a Fitbit device?	No	No	No	No	No	No

Table 5.1: The results of the questionnaire

factors result, gaps that arise are more likely if the user is using his smart phone. So, if a gap in the recording has arisen that it is more likely that the user has not slept.

5.3.2 Recording Time of Fitbit

Fitbit has about 14 days of battery life, some have more, some less. But that's not the big problem. Fitbit can charge you on any USB port without having to install it on the PC, Fitbit. It works right out of the box. The big problem is the storage capacity. Fitbit can hold data about 7 days until it begins to overwrite the old data. And not only that if you want to synchronize data, you have to have Fitbit installed on the PC. So that it results in that it is not easily possible to make a study that is longer than 7 days. Unfortunately, I did not at the beginning. Thus I have some data that got lost from the Fitbit devices my users.

6 Data Analysis

6.1 Download Fitbit Data

To download the FitBit data of each participant, we have written me an API interface in php. The Listing 6.1 shows the PHP source code to an HTML file to enter start and end date, this period is needed to obtain the correct data. This data is then forwarded to the fitbit.php file. As you can see in the Listing 6.2 data are processed there. As you can see in Listing 6.2, the two variables $conskey and $conssec initialize with multiple X, these are unique to each FitBit communication. How do you get these unique values is described in Section 6.1.1 "Register Fitbit App". The variable $apiCall specifies what information the Fitbit server to load. In this example we downloaded the sleep data of a user. But it can do much more data to be downloaded, more details follow in Section 6.1.2

In order to establish communication with Fitbit, the app must be active performances there.

6.1.1 Register Fitbit App

In order to establish communication with Fitbit is a registration with Fitbit needed. This is done via the following URL: https://dev.fitbit.com/apps. 8 values must be set. Application Name, Description, Application Website, Organization, Organization Website, Application Type, Callback URL and Default Access Type must be set. Be careful when enter the "Callback URL", this is very important because if authentication was successful Fitbit forwards the data out there on. When set of "Default Access Type" we used "Read-Only" and my "Application Type" is Browser. The rest of the details that need to be made, have no effect on the interface.

6.1.2 Retrieving Collection Data

The API call is a string, the different variables are separated by a slash, the correct order play an important role. Each call has this structure: /<api-version>/user/<user-id>/<moreinformation>.<response-format>. Until now there is only the API version 1 thus <api-version> can be occupied only by 1. <user-id> can be assigned ID or with a with a specific user "-". In the latter case, the call end user must log in and can bring its own data into the app. This is what I did.

The data can always be downloaded in two different formats, XML, or JSON. This can be determined with the use of <response-format>. Here you can use the ending ".json" for JSON and ".xml" for the XML output.

Then there's the wild card <moreinformation>, this can be combined for a variety of other keywords. For the most important GET-functions see Table 6.1. The full specification is available at [5].

Listing 6.1: index.php: The interface to determine the period

```
<?php
    $date = date("Y-m-d", time());
    $content = '
    <!DOCTYPE html>
    <html lang="de">
    <header>
       <title>Get FitBit Data</title>
       <script src="./includes/datetimepicker.js"></script>
    </header>
    <body>
    <table width="100%">
       <form  action="fitbit.php" method="post" >
       <tr>
          <td>
             <p><b>Start Date:</b>
             <input type="text" name="date1" id="date1" size↩
                ="20" maxlength="32" value="'.$date.'">
             <img src="./includes/cal.gif" onclick="javascript↩
                :NewCssCal(\'date1\', \'yyyyMMdd\')" style="↩
                cursor:pointer"/></p>
          </td>
       </tr>
       <tr>
          <td>
             <p><b>End Date:</b>
             <input type="text" name="date2" id="date2" size↩
                ="20" maxlength="32" value="'.$date.'">
             <img src="./includes/cal.gif" onclick="javascript↩
                :NewCssCal(\'date2\', \'yyyyMMdd\')" style="↩
                cursor:pointer"/></p>
          </td>
       </tr>
       <tr>
          <td>
             <input type="submit" name="web" value="Get Data">
          </td>
       </tr>
       </form>
```

Info	<moreinformation>
Profile	
Get-User-Info	user/<user-id>/profile
Body	
Get-Body-Measurements	body/date/<date>
Get Body Weight	body/log/weight/date/<date>
	body/log/weight/date/<base-date>/<period>
	body/log/weight/date/<base-date>
Get-Body-Fat	body/log/fat/date/<date>
	body/log/fat/date/<base-date>/<period>
	body/log/fat/date/<base-date>/<end-date>
Get-Body-Weight-Goal	body/log/weight/goal
Get-Body-Fat-Goal	body/log/fat/goal
Activities	
Get-Activities	activities/date/<date>
Get-Activity-Daily-Goals	activities/goals/daily
Get-Activity-Weekly-Goals	activities/goals/weekly
Foods	
Get-Foods	foods/log/date/<date>
Get-Water	foods/log/water/date/<date>
Get-Food-Goals	foods/log/goal
Sleep	
Get-Sleep	sleep/date/<date>
Heart	
Get-Heart-Rate	heart/date/<date>
BP	
Get-Blood-Pressure	bp/date/<date>
Glucose	
API-Get-Glucose	glucose/date/<date>

Table 6.1: The most important GET-functions for the wildcard <moreinformation>

```
33      </table>
34      </body>
35      </html>';
36      echo $content;
37  ?>
```

Listing 6.2: fitbit.php: Communication to the Fitbit API

```
1   <?php
2
3
4       $content = "";
5       //pecl install oauth-0.99.9
6
7       // Base URL
8       // API call path to get temporary credentials (request ↩
            token and secret):
9       $baseUrl = 'http://api.fitbit.com';
10
11      // Request token path
12      // Base path of URL where the user will authorize this ↩
            application:
13      $req_url = $baseUrl . '/oauth/request_token';
14      // Authorization path
15      // API call path to get token credentials (access token ↩
            and secret):
16      $authurl = $baseUrl . '/oauth/authorize';
17      // Access token path
18      $acc_url = $baseUrl . '/oauth/access_token';
19      // Consumer key
20      $conskey = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX';
21      // Consumer secret
22      $conssec = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX';
23      // Fitbit API call (get activities for specified date)
24
25      // Start session to store the information between calls
26      session_start();
27      // In state=1 the next request should include an ↩
            oauth_token.
28      // If it doesn't go back to 0
29
30
31      if (isset($_POST['date1']) and isset($_POST['date2']))
```

```
32      {
33          $_SESSION['date_from'] = $_POST["date1"];
34          $_SESSION['date_to'] = $_POST["date2"];
35      }
36
37      if (isset($_SESSION['date_from']) && isset($_SESSION['↩
            date_to'])){
38          if ( !isset($_GET['oauth_token']) && $_SESSION['state'↩
                ]==1 )
39          {
40              $_SESSION['state'] = 0;
41          }
42          try
43          {
44              // Create OAuth object
45              $oauth = new OAuth($conskey, $conssec, ↩
                    OAUTH_SIG_METHOD_HMACSHA1, ↩
                    OAUTH_AUTH_TYPE_AUTHORIZATION);
46              // Enable ouath debug (should be disabled in ↩
                    production)
47              $oauth->enableDebug();
48              if ( $_SESSION['state'] == 0 )
49              {
50                  // Getting request token. Callback URL is the ↩
                        Absolute URL to which the server provder will ↩
                        redirect the User back when the obtaining user↩
                         authorization step is completed.
51                  $request_token_info = $oauth->getRequestToken(↩
                        $req_url, $callbackUrl);
52                  // Storing key and state in a session.
53                  $_SESSION['secret'] = $request_token_info['↩
                        oauth_token_secret'];
54                  $_SESSION['state'] = 1;
55                  // Redirect to the authorization.
56                  header('Location: '.$authurl.'?oauth_token='.↩
                        $request_token_info['oauth_token']);
57                  exit;
58              }
59              else if ( $_SESSION['state']==1 )
60              {
61                  // Authorized. Getting access token and secret
62                  $oauth->setToken($_GET['oauth_token'], $_SESSION[↩
                        'secret']);
63                  $access_token_info = $oauth->getAccessToken(↩
```

```
                $acc_url);
64              // Storing key and state in a session.
65              $_SESSION['token'] = $access_token_info['↩
                    oauth_token'];
66              $_SESSION['secret'] = $access_token_info['↩
                    oauth_token_secret'];
67          }
68
69          $oauth->setToken($_SESSION['token'],$_SESSION['↩
                secret']);
70          $allData = array();
71
72          while (strtotime($_SESSION['date_from']) <= ↩
                strtotime($_SESSION['date_to'])) {
73              $apiCall = "http://api.fitbit.com/1/user/-/sleep/↩
                    date/".$_SESSION['date_from'].".json";
74              $oauth->fetch($apiCall);
75              $response = $oauth->getLastResponse();
76              $jsonData = json_decode($response, true);
77              $Data = array('date' => $_SESSION['date_from'],
78                            'data' => $jsonData);
79              array_push($allData, $Data);
80              $_SESSION['date_from'] = date ("Y-m-d", strtotime↩
                    ("+1 day", strtotime($_SESSION['date_from'])))↩
                    ;
81          }
82          $content = json_encode($allData);
83      }
84      catch( OAuthException $E )
85      {
86          print_r($E);
87      }
88
89      header('Content-Type: x-type/octtype');
90      header('Content-Length: ' . strlen($content));
91      header('Content-Disposition: attachment; filename="test↩
            .txt"');
92
93      print $content;
94  }
95
96  else
97  {
98      die ("Error");
```

```
99      }
100
101
102 ?>
```

6.2 Download Phone Data

The download of this data was no problem, we were able to download all data via FTP. Here was working up the data from the far greater part of the work.

6.3 Preparing the Data

For the analysis it is necessary to divide the data into sections of 24 hours. Since each sensor individually available in a file and includes just a minute, it was necessary to wrote an extra analysis tool. We wanted to create output as csv files include each 24 hours. To be included telephone and data Fitbit data. The whole should run automatically, for any number of users and any number of sensors. Since the app has to be already developed in JAVA, I would not take the new language and thus the analystool is written in JAVA.

We will now explain how a record of all user is analyzed. It starts with the function call analysisToFile, see listing 6.3. We must specify the parameter f. Here, a file must be specified, this must be the folder where the data is located. Thus, it is of type 'File' but linked to a folder. In the loop in this function, each user is then individually processed sequentially. Read, revise and then save. For each record, the function analysis, listing 6.4, is triggered. The parameter to this function is the folder which contains the data of individual users. This function determines how the data are processed. The data from a sensor are transmitted to the extractOneSensor function.

The extractOneSensor function is the only function with exciting content so we will describe this in more detail. See listing 6.5 for this function. First of all, this function calculates the correct times to the result values. This is the code of the bigger part but unspectacular. Much more interesting is the line 16, analysisOneMin(). Here it is decided what happens to the value. What information we have rated as Important. This is then shown in listing 6.6 on.

```
1 public void analysisToFile(File f) {
2       File[] list = f.listFiles();
3       for (int i = 0; i < list.length; i++) {
4           if (list[i].isDirectory())
5           {
6               Log.d(TAG, list[i].getAbsolutePath());
7               AnalysisOneUser a = new AnalysisOneUser(Log);
8               AllDaysOfOneUser d = a.analysis(list[i]);
9               writeCSV writer = new writeCSV(Log, list[i].getAbsolutePath());
```

```
10                writer.writeData(d);
11            }
12        }
13    }
```

Listing 6.3: This function opens the files

```
1  public AllDaysOfOneUser analysis(File pohneDir)
2  {
3          AllDaysOfOneUser allData = new AllDaysOfOneUser(pohneDir.getName());
4          File[] phoneSensor = pohneDir.listFiles();
5          if (phoneSensor.length > 0)
6          {
7              for (int j = 0; j < phoneSensor.length; j++) {
8                  if (phoneSensor[j].isDirectory())
9                  {
10                     try
11                     {
12                         STATE state = STATE.valueOf(phoneSensor[j].getName());
13                         if (STATE.accelerometer.equals(state) ||
14                                 STATE.light.equals(state) ||
15                                 STATE.magneticField.equals(state) ||
16                                 STATE.noise.equals(state) ||
17                                 STATE.proximity.equals(state))
18                         {
19                             allData.addData(state, extractOneSensor(state, phoneSensor[j]));
20                         }
21                         else if (STATE.batteryChanged.equals(state))
22                         {
23                             allData.addData(state, extractOneBlooeanSensor(phoneSensor[j])
                                    );
24                         }
25                         else if (STATE.app.equals(state))
26                         {
27                             allData.addData(state, extractApp(phoneSensor[j]));
28                         }
29                         else
30                         {
31                             allData.addData(state, extractOneBlooeanSensor(phoneSensor[j])
                                    );
32                         }
33                     } catch (IllegalArgumentException e){
34                         Log.d(TAG, "Fail to parse dir:" + phoneSensor[j].getAbsolutePath()
                                );
```

```
35                  }
36              }
37              else if (phoneSensor[j].getName().equals("fitbit.txt") ||
38                      phoneSensor[j].getName().equals("fitbit.json"))
39              {
40                  FitbitParser p = new FitbitParser(Log);
41                  allData.addData(STATE.fitbit, p.parse(phoneSensor[j]));
42              }
43              else
44              {
45                  Log.d(TAG, "Fail to parse file:" + phoneSensor[j].getAbsolutePath());
46              }
47          }
48      }
49      return allData;
50  }
```

Listing 6.4: This function pare the different data types

```
1   private List<AnalysisData> extractOneSensor(STATE name, File dir) {
2       List<AnalysisData> data = new ArrayList<AnalysisData>();
3       File[] file = dir.listFiles();
4       Log.d(TAG, name + " has " + file.length + " Files");
5       for (int i = 0; i < file.length; i++) {
6           if (file[i].getName().endsWith(".csv")) {
7
8               Long timeStamp = Long.valueOf(file[i].getName().substring(0, file[i].
                    getName().indexOf(".csv")));
9               long startTimeStamp = timeStamp - (timeStamp % (60*1000));
10
11
12              extractFile ext = new extractFile(Log, dir.getPath() + "\\" + file[i].
                    getName(), Long.valueOf(file[i].getName().substring(0, file[i].getName
                    ().indexOf(".csv"))));
13              List<EventObj> l = ext.getValues();
14              if (l != null && l.size() > 0)
15              {
16                  double value = analysisOneMin(name, l);
17                  data.add(new AnalysisData(startTimeStamp, value));
18              }
19          }
20      }
21      return data;
22  }
```

Listing 6.5: This function extract one sensor

```
1  public double analysisOneMin(STATE event, List<EventObj> l){
2      if (STATE.accelerometer.equals(event))
3      {
4          return standartAbweichung (l);
5      }
6      else if (STATE.light.equals(event))
7      {
8          return maximum (l);
9      }
10     else if (STATE.magneticField.equals(event))
11     {
12         return standartAbweichung (l);
13     }
14
15     else if (STATE.proximity.equals(event))
16     {
17         return maximum (l);
18     }
19     else if (STATE.noise.equals(event))
20     {
21         return maximum (l);
22     }
23     return Double.MIN_VALUE;
24 }
```

Listing 6.6: This function analysis one minute of one sensor

6.4 Data Analysis

As defined by the already problem in this project, we will just represent the feasibility of the Idea here. Therefore, we will present the data at only one example of example. In figure 6.1 a record is illustrated. This time exactly 24 quarter hours of a subject. Starting at 12 am and ends at 12 am the following day. Thus, the graph shows a full night. All values are normalized based on the clarity to 1. This means that all values in a sensor is divided by the maximum of the same sensor.

What we can see is that is a lot of activity during the day. Towards evening the activity are clear. At about 11:30 you see received is also the Fitbit data now, this is the sleep mode was activated. However, there is still something crucial by the Fitbit sensor. This even goes down to 1 high which means that the person is still awake. Also, the light sensor tells us that the light was identified shortly after 12 clock. Then it will

go through various hours away very quietly in the rashes. Therefore, we would that event the "light off", identified as beginning of sleep. The sleep lasts for several hours then without interruption. Up to about 9 clock we can see again a clear movement to be detected in the morning. The move takes about 10 minutes. From 5 minutes of exercise we would be identify a security. Minor phases can be scored as a short-term sleep disruption. Thus would be clearly identified on the growing point. Similarly, we see that at this time the light sensor to swing back. This theory is confirmed by the accounts of the Fitbit signal. Then from the first activity decreases again about an hour. In this cell phone is obviously not worn on the body. Since we do not take our phone with the shower, we think that this will be quite normal. Then will see the same activities as strong at the beginning of the diagram.

As we just stated, it is very possible to find out the sleep time using the phone data. Certainly, we have made a very large division of the data. However, one must keep in mind that we have served two sensors alone. On the other charts I have analyzed, has issued the lord for me these two sensors usually are enough for an acceptable to good result. This finding makes the burden on the battery and CPU can be significantly reduced. As well as the resulting data flow. With 3 sensors,we decided for the maximum. During acceleration and magnetic field sensor but we were looking for the standard deviation. So we could map the three axes to a value. And was able to circumvent the problem of the acceleration of gravity. Which permanently with about $9.81 m/s^2$ acting on the sensor. But this does not refer to a axle but distributed on all three axes.

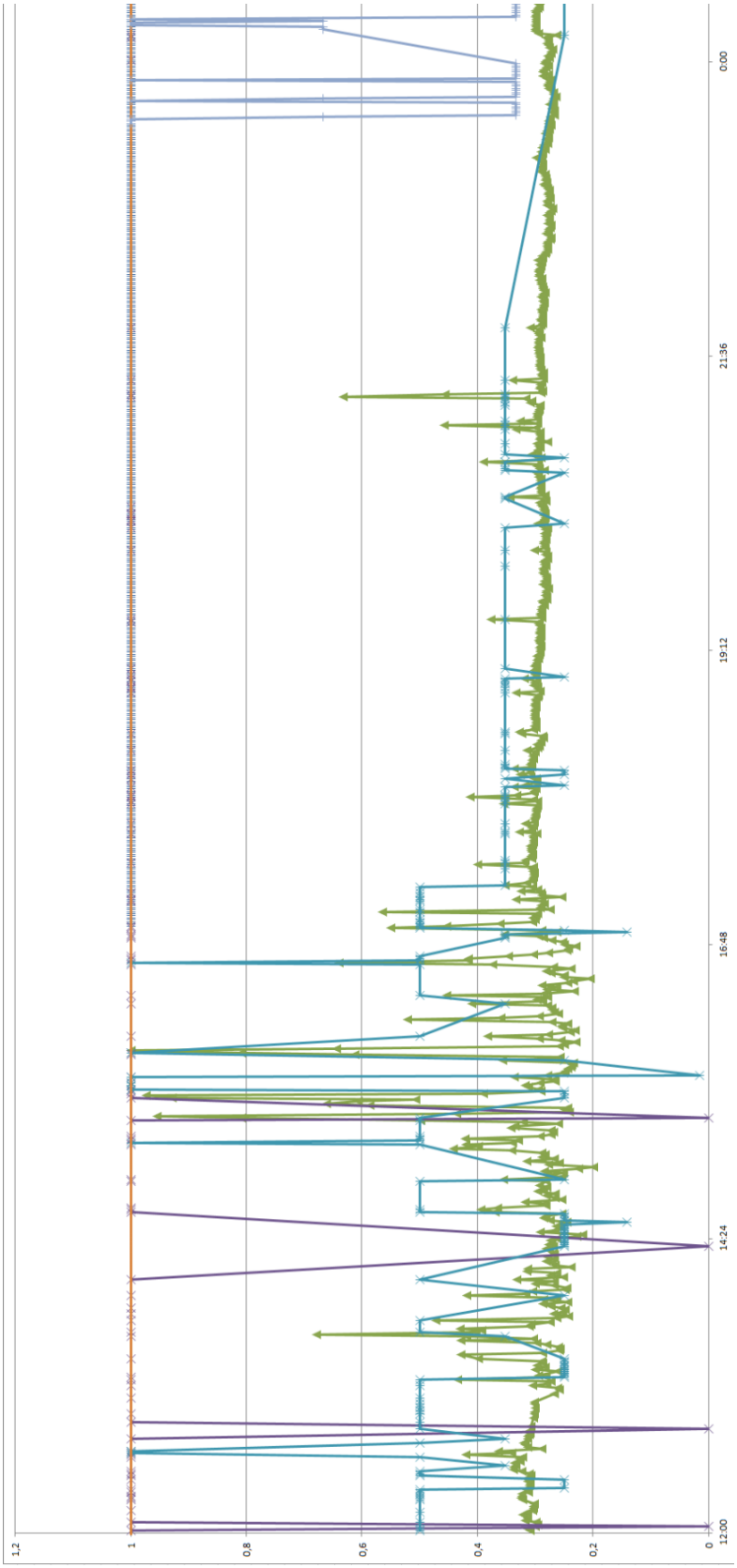
6.4.1 More Data

If one has more available sensors as there is some evidence certain sensors may be used.

The sensor network gives us a little more improvement to make a statement if someone is sleeping. Since most people have wireless home and sleep mostly home, tantamount increases the chance that someone is sleeping.

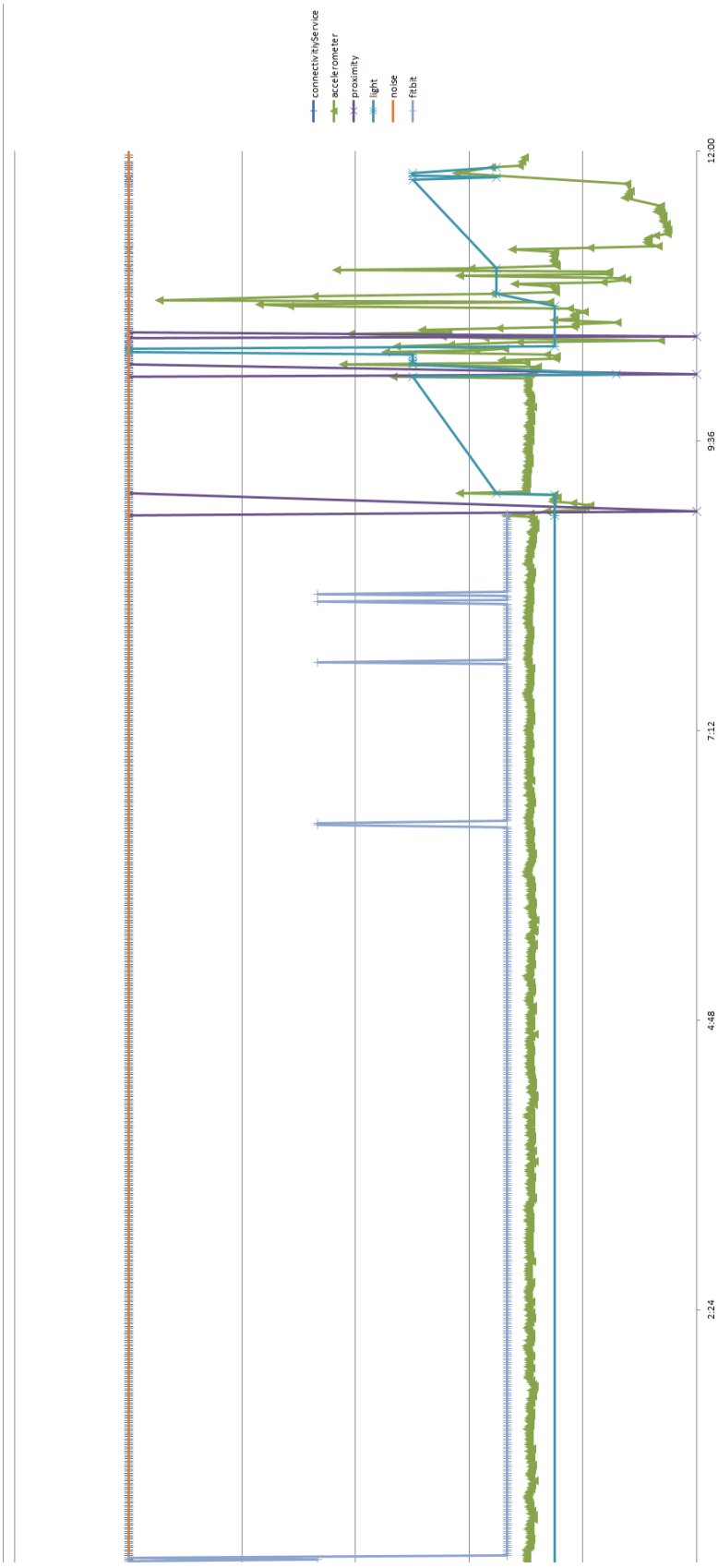
As already described, the signal can be picked up if the phone is charging. As a Plugged phone is not very mobile this is usually done during sleep. So here increases the chance that the person sleeps when it is plugged.

This behaves the same way with the airplane mode. To Vedas not disturbed at night activate a part of the people's airplane mode. So here increases the chance that sleeps the person. That can be taken even further. Because this also applies to mobile phones switched off completely. This is usefully made over several hours if you do not want to be bothered or just want to sleep in peace.



(a)

Figure 6.1: Recording over 24 hours of a mobile phone


connectivityService
accelerometer
proximity
light
noise
fitbit
12:00
9:36
7:12
4:48
2:24

(b)

7 Conclusion

Finally we can say that in this area is still much work to make. Through the research we have done, we have not really found much information. Information is visible in the maximum result of some products. This, however, give out no information about their calculations, understandably. So we am assuming that here a lot of potential is far from exhausted. For we find out when someone is sleeping only scratches the surface of what these data tell us everything.

We also have the GPS sensor for privacy reasons completely left out. This would increase the possibilities of the use of the data by a multiple. But here is a caution to permanently get GPS signal is almost impossible. Through GPS, the battery life to about 1/5th.

If somebody try's to make a similar project with more success. It's necessary to fix this bit gap problem at the recording time.

Bibliography

[1] andorid.com system nanotime. http://developer.android.com/reference/java/lang/System.html.

[2] Android 4.2 r1. http://grepcode.com/snapshot/repository.grepcode.com/java/ext/com.google.android/android/4.2.2_r1/.

[3] android.com intent. http://developer.android.com/reference/android/content/Intent.h.

[4] Fitbit api. https://wiki.fitbit.com/display/API/Fitbit+API;jsessionid=59FE43DB386815A07B23FC002B7CF45F.

[5] Fitbit resource access api. https://wiki.fitbit.com/display/API/Fitbit+Resource+Access+API.

[6] Jawbone up api. http://eric-blue.com/projects/up-api/.

[7] Withings api. http://www.withings.com/de/api.

Decleration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature