DroidEnergy - Detecting and Alerting Users of Power Supply Failures

Gonçalo Grazina
goncalo.grazina@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2016

Abstract
In a society where the number and variety of in-house technological devices continues to increase, some of these devices can not tolerate long power supply failures. For instance, a power loss of more than a couple of hours may spoil the food in the refrigerator, or the video surveillance system may become unusable. Currently, existing solutions either provide only a limited outages monitoring (on or off), or are specific products connected to a cloud server that can be scheduled to activate/deactivate outlets and provide a limited set of configurations. The goal of this work is to develop a system that is able to detect power supply failures, their duration, and alert the user of such events according to a flexible range of configurations. The proposed solution, called DroidEnergy, is a system composed by a smartphone application (along with a server for back-office purposes) that is plugged into an electric outlet, and is able to detect outages and alert users. DroidEnergy has several easy to configure features, e.g. alerting the user via e-mail or SMS, can be configured remotely or locally, creating a highly configurable system; we describe the solution design options, as well as the challenges to overcome in order to validate the proposed solution.

Keywords: Outage detection, Android, Home Automation, User alert

1. Introduction
Home automation has been in the vanguard of the research subjects for several years. It provides mechanisms to control various aspects of a house, with the objective of providing comfort to users and help them make energy efficient choices when using their own devices [1] or house embedded devices. With the continuous increase of in-house electronic devices, the power consumption is becoming an important issue and home automation gives an important contribution to this matter. Several contributions to this issue have been made through investigation of energy-aware techniques. Such contributions include energy monitoring systems [2, 3, 4] and context-aware power management [5, 6]. Besides providing efficient ways to manage power consumption, Home Automation Systems (HAS) should also be able to deal with power cuts, in order to maintain the availability of the devices when a power outage occurs. The solution to this problem is hard to achieve, since most of the infrastructures would have to be adapted to be able to incorporate programmable technology and power backup systems. Furthermore, most of the existing solutions only monitor the occurrence of power outages, in order to present statistical information to the user and are unable to alert the user of these electrical events and are constrained to in-house use. We propose DroidEnergy, a system composed by an Android application capable of detecting power supply failures, monitoring the frequency and duration of outages and alerting the user of such events and a web server which offers a tool for remote access and configuration of the system. DroidEnergy takes advantage of the power information when connected to a wall plug to identify the current charging state and of the fact that most houses only have one electric board. The system can also be configured directly on the smartphone or via a web server, giving the user the possibility of remote configuration. Furthermore, upon outage or reestablishment detection it is able to alert the user of the event according to a pre-specified time. Finally, the outage duration and frequency is monitored to provide statistical information to the user. This document is structured as follows. In Section 2 we present literature that is relevant to our work. We also discuss the existing solutions and perform a comparison between them and proposed solution according to the objectives defined previously. Next, in Section 3 we introduce the architecture of the proposed solution. In Section 4 we present the implementation of the
solution and describe the evaluation methods that will be used to measure the performance and quality of the proposed solution. In Section 5 we describe the methodologies and results of the evaluation. Finally, in Section 6 we present the conclusions drawn from the research work and implementation of our solution.

2. Background and Related Work

In recent years the number of devices in our homes has increased significantly. With this increase, the users want agile ways of operating, connecting and controlling the state of the devices. In 1984, the National Association of Home Builders (NAHB) introduced the concept of a “smart house” where a system would control automatically or manually various functions in the house, such as turning on/off a light based on daylight, therefore offering interoperability between devices [1]. The concept was later materialized into Home Automation Systems (HAS). HAS connects the “smart devices” in a house with the objective of providing comfort to users in a ubiquitous way.

With the functionality increase offered by (HAS) and consequent escalation of energy consumption, researchers started focusing more in energy efficient solutions. One interesting fact is that (HAS) are not only the cause of energy consumption increase as they are also a solution. In 2002, Banerjee et. al inquired a user about the performance of his solar panel installation to which the user responded that he monitored the reading manually [7]. Later on, technologies such as WattsUpMeter and Android@Home as well as services such as Google PowerMeter or Microsoft Hohm began the process of providing users access to raw data. Monitoring systems, capable of analyzing the raw data, can have a high impact in lowering the energy consumption by creating profiles and presenting it to the user in a human readable format. Dobson [8] and Chetty’s [9] research has shown that granting the user visibility into the energy consumption can reduce the electricity usage in 5-20% [10]. These monitoring systems, based in data collected from smart meters, can be further integrated with Advanced Metering Infrastructure (AMI) in order to create a power system capable of automatically diagnose, monitor and repair a smart grid [11]. Nezhad et. al [4] propose SmartD, a easy to use and to extend dashboard to visualize and analyze data gathered by smart meters. Based on GNS middleware [12] it can be integrated with almost every existing smart meter to provide real-time readings. By being capable of analyzing the gathered data according to different contexts, it provides the user with visualization of different power consumption profiles according to temporal aggregations and consumer segments. Being a smart meter "aggregation" system, it requires the installation of smart meters throughout the house which can prove to be expensive. More recently Apple launched an iOS application is serves as a gateway for smart devices such as wireless light bulbs. Although the amount of research work in this subject is high, houses that use (AMI) are still only a few. The three main factors that cause this low “acceptance” are the costs of installation, learning of user’s preferences that do not fully fulfill their needs and the wide range of customization that research solutions offer [13].

Klugman et. al propose GridWatch, one of the most similar solutions to our work. Their solution was developed based on the idea that by using a everyday device, such as a smartphone, users can have a better understanding of the power conditions and energy usage without the need of utility companies [14]. In addition, the solution does not require the installation of smart meters, lowering the complexity and cost of implementation. Using the sensors of the smartphone to analyze the power states while the smartphone is charging, their solution can retrieve information of power outages, such as GPS location of the outage and duration time. The results of the evaluation show that the prototype can detect power outages with high accuracy. The main constraints in using GridWatch are that it does not allow remote access or alerts users when an outage occurs. Another limitation is that it required a large community to fully function as expected, since the main purpose is to present information based on a large set of retrieved data.

Patel et al. presents an approach that uses a single sensor that monitors the electrical noise made by electrical applicants in order to recognize a variety of events [15]. Apart from the sensor this approach only requires a computer to collect and analyze the incoming electrical noise. This approach was influenced by PowerLine Positioning System [16], where the detection of electrical events is based on the electrical noise in the power line made by the switching of electrical devices. Although this solution is able to detect several electrical events, it requires special equipment. Furthermore the system must be handled locally and can not provide the results remotely.

Banerjee et. al proposes a monitoring station which makes energy recommendations in order to lower consumption based on the energy profile of the house. The recommendation system can advise users of when to use high-power applications or how to minimize the energy waste [7]. Although the solution is capable of monitoring energy consumption and detected power outages, the main goal is to provide energy efficient suggestions to the user. It also requires additional equipment, increase the installation complexity.
Although the primary focus of energy management research is monitoring, controlling the electronic devices is also an important step towards power efficiency. Some research works propose systems that are able to turn off power outlets when the maximum power load is reached \[17, 18\], but they do not offer real time control over the devices. Han et. al propose a remote-controllable and energy-saving room architecture that provides the user with a way to control the state of the power outlets using a hand held device \[19\]. This approach ensures that every user is able to adapt the configurations to fit his preferences. Their results show that the development of such systems is viable and can be applied to every room in a home, although it can prove complex to implement.

3. Architecture

This section presents a global view of DroidEnergy architecture. Figure 1 shows a global view of the proposed solution. The smartphone running DroidEnergy can be plugged to any outlet in the house. In the initial configuration the user will be prompted to add the house devices to monitor, the phone number and e-mail address as well as the alerts he wants to receive. After this step, to start the monitoring, the user has to press the "start" button in the main screen. Any additional changes to the initial configuration can be made locally or remotely (by enabling this option in the main screen or via SMS).

Since the smartphone connects directly to the outlet, we can take advantage of the fact that most houses have a single switchboard to connect all the electrical miscellaneous, therefore if an outage occurs it most likely affects all the devices. The smartphone should also be connected to the local internet router and have a cellular internet connection. We recommend these two internet connections in order to keep the availability of the system, since the router will be affected by the outage, while the cellular connection usually are not affected. In case the user only wants to be alerted via SMS, the internet connection will not be required to alert him.

Figure 2 shows the software modules of the application. The application is divided in 5 main modules: (1) Activities; (2) Broadcast Receivers; (3) Alerts; (4) Data Storage and (5) Web Server. When the application detects that an outage occurred (via the ACBroadcast receiver) it starts the monitoring phase. This phase consists in checking if any alert should be sent to the user, according to its configurations, e. g. the user wants to be alerted when the refrigerator is out of power for one hour. This module also sends periodic messages to the main screen activity to update the outage duration time information. When the outage ends it uses the duration of the outage and devices affected (if any) and stores it persistently.

When the system detects that an alert should be sent, it redirects the request to the corresponding alert manager (SMS or e-mail). If a cellular internet connection is not available, the alerts are stored and sent when an internet connection is reestablished. When the outage ends, the system can also alert the user, if it chooses to.

All system configurations can be made directly on the phone (via the settings activity) or via a web server (which can be accessed remotely). The web server fetches the pages from the assets folder and the specific information, such as devices, from the SQLite database. The user can also view statistical information in the web server, which is gathered during the outage, as stated previously. The configurations are stored according to their type. Location, device and user information are stored in SQLite tables. The assets folder has the HTML pages and JavaScript files of the web server. Finally, the Shared Preferences have mostly boolean values such as "is remote access enabled", to minimize the call to the database.
4. Implementation

While it is still not possible for an application to learn the user needs based on his smartphone information, we believe that with a simple initial configuration we can achieve the autonomy a smart appliance should have. The application was developed for API 22 (Lollipop), although it has backward compatibility until API 16 (JellyBean), since around 50% of Android devices run versions between API 16 and 19 (including). This detail also enables the use of older Android phones that users may have at home. In the following sections we describe the implementation of the application features, necessary to achieve the requirements.

4.1. Outage detection

DroidEnergy requires the smartphone to be connected to an electrical outlet to monitor electrical events (outage and energy supply reestablishment). Although smartphones are able to charge via USB, we chose to monitor electrical events only when the smartphone is connected directly to an electrical outlet, since intermediate devices may deceive the application into thinking an outage is occurring when in fact, it is not (e.g., some laptops are unable to charge via USB when they are powered off).

Android exposes battery information via system broadcast messages. These messages give information regarding various events such as charging state, charging type or battery percentage. We implement two broadcast receivers that react to battery changes. (i) UpdateBatteryReceiver and (ii) EnergyReceiver.

4.2. UpdateBatteryReceiver

The UpdateBatteryReceiver is the receiver used to verify the current charging state of the device. To enable this feature, we register it with an intent filter for the action ACTION_BATTERY_CHANGED. This action is sent every time a battery change is detected and the frequency depends of the hardware. When the receiver is called, we extract the information that lets the receiver know if the phone is charging and what type of charge (USB or AC), using the intent that was sent to the receiver.

We read the intent extra that returns the integer number associated with the types of power connections. We then check if this number is equal to the number of the BATTERY_PLUGGED_AC and if it is, the device is connected to an AC charger. We only use this type of power source since devices charging via USB can stop charging when the device they are connected to is turned off. This information is then used by the Energy Receiver to know if the system should start to monitor an outage or if an outage has ended.

4.3. EnergyReceiver

The EnergyReceiver is the application component that reacts to charger connect and disconnect changes in the smartphone. It is registered for two different actions: (i) ACTION_POWER_CONNECTED and (ii) ACTION_POWER_DISCONNECTED. The first one is the action broadcast by the smartphone when the user connects the device to a USB or AC charger.

When the receiver is called, it checks which action was sent. If the action was ACTION_POWER_DISCONNECTED, the receiver reads the previous charging state, which was determined by the UpdateBatteryReceiver. If the previous state was charging via the AC charger it means an outage was detected. To monitor this event, the receiver launches a service called Outage Service by invoking the method startService(new Intent(context, OutageService.class)). When the service is launched, it creates a runnable. Since the runnable code will always be the same, there is no need to create the runnables every time the receiver is called, or to use threads. Instead we define the runnable code and send it to a Handler each time an outage is detected. The run() method of the runnable updates an event list in the beginning of every run. This structure is a HashMap that contains the two possible outage events. The update on the list is made by reading the preferences set by the user and sorting it by the least alert time. This structure is created by a static class called Constants, which is responsible for saving and providing objects that are equal, independently of the application context or activity. After the update, the method for each event, time pair in the list checks, verifies if the current outage time exceeds the corresponding time. The actions when this occurs differ for each event:

Alert on outage - a simple alert is sent to the user with the current outage time.

Check devices - since the device’s information is stored in the database, to check each device we use an AsyncTask. As the name suggest, the AsyncTask is an asynchronous task that runs on a background thread, therefore removing the burden of manipulating threads and handlers or running tasks that could cause excessive work in the UI thread. On the doInBackground method of the AsyncTask, a request to the database is sent with the current outage time which returns an array list with the affected devices. The result of the method is the message to be sent in the alert.

Every ten seconds the method sends an update message to the main activity if it is running. This message carries the current outage time and type of event which will then be processed by the activity to update the main screen. When the action AC-
**TION_POWER_DISCONNECTED** is received, the outage ended. If the user chose to receive a reestablishment alert, the receiver launches the *ReestablishmentService* which is similar to the previous one. The only difference is that when the reestablishment alerts is sent the service stops itself.

### 4.4. Alerts

DroidEnergy may be used as a simple outage monitor or it can be used as an outage alert system. In the second case, the user may receive alerts via Short Message Service (SMS), e-mail or both. SMS alerts require a SIM card while e-mails require an internet connection or a SIM card (for cellular data service). Since cellular data service may be charged by the service provider, DroidEnergy only sends e-mails via cellular if no WiFi connection is available. Since the local internet connection may be unavailable during an outage, if a cellular internet connection is also unavailable, the system stores the message and re-sends it when an internet connection becomes available.

The alerts can be of three different events: 1 Outage - informs the user that an outage was detected; 2 Reestablishment - informs the user that an outage has ended; 3 Devices - informs the user which device is without power for longer than configured. To be sent automatically, the e-mail alert are sent using the JavaMail API. While it is beneficial since it does not require user interaction, the user must enable the *Access for less secure apps* in Gmail settings. To send a SMS, the system first checks if the cellular network is available using the *TelephonyManager* and in case it is, sends the SMS using the SMS Manager.

### 4.5. Web Server

The web server is implemented using *NanoHTTPD* which is an open source lightweight HTTP Java server with HTTP 1.1 and SSL support. To build our own server we created a new class that extends from the *NanoHTTPD* class and defined two constructors, one for a port server and another for a host:port server. Since smartphones have the option to configure a static IP, we only use the port server. Nevertheless we created the other constructor so that we can implement a host:port server if needed. The extended class described above is responsible for handling the HTTP requests by overriding the *serve(*HTTPSession*)* method. The *serve* parameter is a session which allows us to get the parameters of the request and parse the body of the message. The first step to implement the server is defining the possible paths between HTTP pages and requests to the JS and CSS resources via static strings. The second step is to construct an URI matcher to determine the request made to the web server. To do so we implement a class that uses a *Pattern* class. This java class is a representation of a regular expression that can be compiled to return a *Matcher* that can verify if the input string the compiled pattern.

The third aspect to notice is that although there is a difference between requesting a resource via the web server and requesting the same resource directly from the device, the server should access the resource in a similar way. This reduces the burden of implementing two different functions that have the same output. Therefore we developed a set of interfaces which are used by the web server to access the application resources as a user would do when interacting with the phone. The interfaces are the following:

- **PagesInterface** - methods to retrieve HTML pages and device information
- **SettingsInterface** - methods to retrieve and save individual settings of the preference menu
- **JSONInterface** - methods to retrieve bulk data, such as a set of locations, devices or preferences
- **CSSInterface** - methods to retrieve the CSS files
- **JSInterface** - methods to retrieve the JS files

Each interface is extended by a class that implements the methods. When the server receives an *HTTPSession* request, it retrieves the URI. It then passes the URI to the *URIMatcher* that verifies if the it matches any of the paths previously defined. When it detects a match it returns a string indicating the matched path. The returned string is then used in a switch statement to determine the requested page or resource. If no match is found the server replies with a "Not found (404)" code. In the other case the content of the message is sent to the interface responsible for the request. The content is parsed and the operation takes place. In case there is an error during the execution of the operation, the server replies with an error message to the client.

An Android service is an application component that can be used to perform long operations on the background. Since the user can activate the web server for an undetermined amount of time, we deploy the server as an unbound service. A bound service is useful if we want to communicate between the service and the activity it is bound too, which is not our case, so we opted by the use of an unbound service. When the user activates the server, via SMS or directly on the phone, the application launches the service by executing the *startService* method with an *Intent* which includes the service class name. The life cycle of an unbound service is
different from the Android activity life cycle. When a service is started the first method that executes is the `onCreate()` followed by the `onStartCommand()`. After the second method executes the service keeps running until it is stopped by itself or by the client. Before being completely shut down, the `onDestroy()` is called. Although being unbound, a service always runs on the UI thread. This may prove to be a problem if the service is executing too much work. In the `onCreate()` method we create an instance of the web server. On the `onStartCommand()` we create a new thread and start the server. This assures that the web server executes in a different thread and will not block the UI thread. When the user shuts down the server, the `onDestroy()` is called and the server is stopped before the service is destroyed.

To access the web server the user requires an account. The account can be created on the application or one can request an account via the web server. This option was chosen to provide some security on who accesses the server. In the application the user can view and manage all the accounts that have permission to access and view the web server content. Furthermore, the activation/deactivation of the server via SMS can only be made if the incoming SMS was sent by the phone number configured in the preferences menu.

The server presents all the configuration options present in the application with the exception of the account management, which only the administrator has access to. The server also features as charts and statistical information. The charts are build using the outage information stored in the device and Google Charts. This is a free tool that is easy to use and offers interaction which is particularly useful when changing, for instance, date ranges. The information to build the charts is requested when the page loads by a JQuery function which sends a HTTP GET request to the server and receives a string representation of a JSON object. This object is then parsed and the charts are built in addition each chart has a range slider so the user can zoom in and out.

5. Results
In order to evaluate the proposed solution according to the motivation and solution requirements, we defined a set of performance and usability tests. These tests provide information that is useful to validate and identify implementation problems and design options which are important to create the best solution possible. In this chapter we present the performance tests in section 5.1 and usability tests in section 5.2.

For the performance and usability tests we used two different devices with different versions the Android OS. The first one is a white-branded Android phone running Android KitKat (version 4.4.2) that has no retail price, although devices with similar hardware specifications cost around €35 in 2014. The second phone is a Motorola Galaxy Nexus 6 running Android Marshmallow (version 6.0.1) whose cost was around €600 when it was release, also in 2014. Each device has a SIM card with a mobile plan which has 1 GB of mobile data and unlimited SMS.

For the performance and user tests we defined the following scenarios.

1. Scenario A - Device A is configured with a time of 1 minute and Device B with a time of 3 minutes. The user selected the SMS alert. An outage occurs and its duration time exceeds the time set for Device A and the user is notified.

2. Scenario B - The user changes the time for Device B to 1 minute. The user selected the e-mail and SMS alerts. An outage occurs and its duration time exceeds the time set for both devices. The system notifies the user for both devices and using both types of notifications.

3. Scenario C - Before leaving the house, the user selected the "Activate WebServer via SMS". After he leaves, he tries to access the web server but he forgot to enable the server. He sends and SMS to the phone running DroidEnergy to enable the server. It then enables the "alert on reestablishment". An outage occurs and after a while ends, sending the alert to the user.

The possible scenarios are far more vast when compared to the scenarios we present, however we only present some examples that we believe can validate and improve the features of the solution. Ideally the timers set for the devices in the scenarios would be in hours but since these timers do not influence the overall performance of the tests, we reduced them to minimal values.

5.1. Performance tests
When developing mobile applications, the performance and resource management is one of the most important factors to take into account. While the performance is more dependent on the hardware itself, the resource management done in the applications can still make an impact. This management can be done with help of tools that retrieve fine-grained information and make them human readable, so the programmer can focus more on how to solve the problems.

Android Studio is the official IDE for Android which offers all the tools needed to develop and optimize applications. The best way to optimize an application is to run it while connected to the IDE and use the provided tools to check the resource consumption in real-time and therefore, dis-
cover more easily potential problems or optimization opportunities. Since one of the objectives of DroidEnergy is to monitor outages, we need to ensure that the application is able to run during long periods of time. For this to happen, there is a need to test the features of the application during the development phase, in order to minimize eventual problems in the code and have better performance.

In this sections we present the performance tests that were made to verify the resource consumption of DroidEnergy. In Section 5.1.1 we present the results of the RAM usage testing and in Section 5.1.2 we present the tests and results of the battery profiling tests.

5.1.1 RAM usage
Android offers a way to determine the amount of RAM an application can use (called threshold) before the system starts to reject additional memory allocation. This threshold varies according to the RAM of the device and can be retrieved to check if the RAM usage of the application is near the limit. To retrieve the threshold for each of the devices we made two slight modifications to the application, so it would log this information after the `onCreate()` method. Table 1 show the results of the memory information.

<table>
<thead>
<tr>
<th>Devices</th>
<th>White-brand</th>
<th>Nexus 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available 1</td>
<td>133</td>
<td>2186</td>
</tr>
<tr>
<td>Heap size 1</td>
<td>96</td>
<td>256</td>
</tr>
<tr>
<td>Threshold 1</td>
<td>56</td>
<td>144</td>
</tr>
</tbody>
</table>

1 *In MB*

These values are used to determine if the RAM usage of the application is in the accepted value range and that memory problems will not occur during the application life cycle. The second modification was to remove the need to connect an AC charger prior to start the monitoring. This modification was made to be able to simulate an outage while keeping the device connected to Android Studio via USB. Therefore we are able to visualize and retrieve information from the Android Monitor during an outage.

The procedure to profile the RAM usage is defined as follow. We connected the device via USB to Android Studio and gathered the RAM usage during the executing of the three scenarios described previously. In the beginning of the test we made the configured all the necessary details for the scenarios, such as the e-mail configuration and the web server accounts. We then performed each scenario separately and with a period of approximately two minutes between them. The reason why we chose not to do the scenarios consequently is because we wanted to check if any Garbage collections (GC) were triggered by each of the scenarios. Android documentation points the number of GC as an indicator of excessive allocation of temporary objects and therefore as an indicator of a potential memory usage problem. Therefore, besides measuring the peek RAM usage in the scenarios, we also counted the number of GC.

The results show that the Nexus only had only one GC during the three scenarios while in the other phone there were two. After the second scenario both devices issued a GC. This is probably related with the end of the outage service, since it is destroyed when the outage ends. The peek RAM usage was on average 30.90 MB for the Nexus and 9.72 MB for the other phone. If we compare the obtained results with the Heap Size for each device (refer to Table 1) we can see that the measure values correspond to approximately 12% of the heap size in the Nexus and 10% in the other case. From these tests we can conclude that the applications used approximately the same proportion of RAM in both devices and the values represent a small percentage of the total available memory for applications.

5.1.2 Battery usage
Battery usage is also a problem that is not easily resolved. While smartphone hardware components keep increasing every year in terms of performance, the battery life and cycle duration do not follow the hardware trend. When developing a mobile application the performance is probably the most important aspect but the battery duration also has a high importance to the user. In this work we use two smartphone with a noticeable difference in performance as well in battery duration. Since we developed the application with backward compatibility for older phones, most of the devices still running this version will not have a great battery duration. For this reason it is important to confirm that DroidEnergy uses the minimum battery as possible. This will result in a longer monitoring time during an outage.

Google provides two tools that are able to extract and analyze the battery information of the smartphone. The BatteryStats is part of the Android framework and is responsible for retrieving the smartphone power usage and the BatteryHistorian analyzes the extracted data and presents it in an HTML page. Unfortunately these tools only work in Android 5 or above, so we were only able to use these tools to determine the application power.
usage in the Nexus. To determine the power usage of DroidEnergy we defined an hour long scenario. We chose this time interval so that the RAM and CPU usage could stabilized between events. Since these resources have a direct relation with battery consumption and the events in the scenario will not occur at the same time, the estimation is more accurate. Furthermore, a measured time in a fixed interval of one hour can be easily used to estimate battery consumption in different time intervals. The steps of the additional scenario are the following:

- Enable the mobile internet connection
- Configured the e-mail and SMS settings
- Enable the web server
- Add the house area "bedroom" with the devices (<device A>, 20 minutes >, <device B>, 30 minutes >)
- Add the house area "kitchen" with the devices (<device C>, 40 minutes >, <device D>, 20 minutes >)
- Choose to receive the device alert
- Start the monitoring
- An outage occurs
- After one hour, the outage ends

Android smartphones gather battery information between full charges. In order to have a reliable battery usage reading, the battery stats must be cleared. To do so we connected the Nexus 6 to a computer and removed all battery stats using the Android Debug Bridge (adb). The adb is a command-line tool that enables the communication between a computer and an Android emulator or a Android-powered device. After connecting the device to the computer we can confirm that the device are detect by calling the \textit{adb devices} in the command-line.

After confirming that the device is detected we reset the battery statistics gathered by the phone. After the reset we disconnected the devices from the computer and started the testing. After completing the scenario we reconnected the phone to the computer and extracted the battery information gathered. A small part of this information can be viewed in the \textit{Apps} \textendash{} > \textit{Name of the App} \textendash{} > \textit{Battery}. The full report can be viewed by launching the battery historian server. In this server we can upload the bug report retrieved from the phone. The server analyzes the data and builds a chart where we can see many characteristics such as the CPU time, battery level and temperature. The server also has another section where we can select the application and check the individual statistics. From these statistics we retrieved that the estimated power consumption of DroidEnergy is 0.14% of the battery capacity of the Nexus. If we take into account that the application was used over a period of one hour, the estimated value of 0.14% battery per hour is a promising value. As an example, in a case where only our solution was draining power from the battery, it would take approximately 39 days and 3 hours for a full discharge. Taking into account that most outages last for a period of few hours, the application would be able to run trough almost every outage. Although we are not able to use the same method to retrieve the battery information of the other device, the CPU usage in the Nexus indicates that our application does not use much CPU time since the higher CPU usage was 2%. Another important aspect to notice is that the application also does not require constant rendering of the screens, which is a common source of considerable battery drain. In the case of the other phone, during the RAM usage testing we were able to observe the CPU usage. Naturally it was higher than in the Nexus but the highest registered value was 20% during one second. We also registered two other CPU usage peaks but they were less than 10%, also during one second. These values are high for an application but they occurred only three times during one hour of testing and during very short periods of time (one second maximum). We can not make a direct relation between the observations and the how much battery the application would drain in the cheaper phone, but we can have an insight on how much work intensive the application is and therefore, determine that the power usage of the application would not be high.

5.2. Usability tests

Usability tests are an important part of validating a platform. User experience provide an accurate way to determine the aspects to improve by testing the application in a real environment. Prior to testing the application with the users, we elaborated a survey to validate the idea behind the prototype. The questions present in the survey are more focused in determining if the users find the idea and its features useful and interesting.

5.2.1 Validation Survey

From all the participants in the survey, we first choose the ones that qualified to participate in the rest of the study using two questions. From the 53 subjects that answered the survey, 92.5% of the responses were affirmative to the question "Do you own a smartphone?". Since part of this work is prototyping a smart phone application, the remaining 7.5% were excluded.

The last question to filter the appropriate subjects was "Would you consider using a system that detects power outages and alerts you even if you are not at home?". 85.7% of the subjects responded

8
affirmatively, ending up with a total of 42 test subjects to continue the survey. We asked one final question to the 14.3% which did not find the application useful to determine why the subjects did not find the idea appealing. The justification that most participants gave was that they do not find it useful due to the rareness of the outages.

The second part of the survey has the objective of determining which features the test subjects find more useful. To do so we asked the users to rate from a scale of 1 to 5 the following four questions according to their usefulness, where 1 means Not useful, 3 means no opinion and 5 Very useful.

1. Would you find useful that the app could monitor different home devices?
2. Would you find useful accessing the app configurations even if you are not at home?
3. Would you find useful to receive outage alerts via SMS?
4. Would you find useful to be able to configure almost every aspect of the app?

For the first question the users were informed of the context of the application, where they were told that the monitoring of devices is to associate a time which will be used to determine if the user should be alert for that device, during an outage. 95.2% of the participants find it useful to monitor different devices. Regarding question number 2, again most of the participants found the feature of accessing the application from outside the house at least useful.

Regarding question 3, since the objective of this work is to alert users of power outage, we chose to question the participants if they would find useful to be alert for such events. As we can see 54.5% of the participants find this feature useful, which is worse than expected. Similar to the previous question, we feel that the participants answered from the context of non-smart houses or houses where there are not services or devices that need continuous power supply. Nevertheless the majority of the participants find this feature useful. Finally, in question 4 we inquired the participants about the customization one can make in the application. Similar to the previous questions, the majority of the participants (80.9%) find the customization useful.

As described before, the results of this initial survey show that the main features we included in the application are useful and are suitable in the context of this work. As a summary of the answers given in the four questions we asked, from a total of 168 given answers, there was a total of 134 (79.8%) positive answers and only 6% of negative answers.

5.2.2 Testing with users
The second phase of usability tests is testing with users. This is important to evaluate aspects of the application such as how the information is presented or if it is easy to use. We selected a group of ten people where five of them are experienced users and the other half is not. An experienced user is a person that interacts with smartphone applications everyday and an inexperienced user is a person which does not have much knowledge with application interaction, although four out of five participants own a smartphone.

In these tests we aim to learn if both types of users can easily adapt to the application after using it a couple of times. To do this we measure the time it took for the participants to complete each scenario and after completing all three scenarios they answered a quick survey. Besides being able to determine if the users can learn to interact with the application easily, we can also retrieve from the survey aspects that need to be corrected.

Before the beginning of the tests we set up the testing area. We used the Nexus 6 for testing to improve the user experience during the scenarios. We connected it to the AC charger and the charger to a power outlet with a on/off switch in order to simulate the outages. Additionally we equipped the smartphone with a SIM card and configured the application with the e-mail credentials needed to send the e-mail alerts.

After setting up the testing area, we made an introduction of the objective of the work to each user, as well as an explanation of the application and its features. This explanation was focused in what each feature does without giving any hints on how to do it, so we would not influence the testing. Furthermore, the application was previously configured in terms of e-mail account and web server settings since these configurations are not needed to validate the application and therefore, we can minimize the difficulty of completing the scenarios.

For each scenario described previously we handed out a script containing the actions that each user should perform during the scenarios. The scrips do not teach the user on how to use the application, instead they describe all the actions that need to be performed in order to complete each of the scenarios.

After the initial interpretation of the scenarios, each user completed each one of them individually. Figures 3 and 4 show the measured time that each user took to completed each scenario. The measured time presented in the figure does not include the time that users had to wait to an alert to be sent. For instance, in Scenario 1 the user had to wait one minute for the alert for device A to be sent, but since this time is independent of the user which
is testing the application, they were subtracted from the measured time.

As we can see in Figure 3 the experienced users took an average of 3 minutes and 16 seconds to complete scenario 1. If we compare to scenario 2, which is very similar to the first one, we see a reduction of approximately 27% (one minute) in completion time. When comparing the average time between scenarios 1 and 3 the time was almost equal, differing in only two seconds. These two scenarios tested different features and the results show that the users took almost the same time to adapt to the execution of new objectives, although the third scenario was harder than the first.

Regarding the inexperienced users, on average each user took 5 minutes and 54 seconds to complete scenario 1, 3 minutes and 23 seconds for scenario 2 and 4 minutes and 43 seconds for scenario 3, as we can see in Figure 4. Comparing the first and seconds scenarios, we see a noticeable difference, where the users took almost 45% less time completing scenario 2. We observed another noticeable difference between scenarios 1 and 3, where the third took almost 20% less time. This difference is justified by the adaptation effort users with low or no experience have in using smartphone applications. Although they take almost 45% more time to complete the first scenario compared to the experienced users, they reduce the time in approximately 15% in the other two scenarios. One interesting observation is that User 2 of the inexperienced group took only 3 minutes and 1 second to complete scenario 3, which is less than the average for the experienced group in the same scenario.

After the test phase each user was asked to fill a survey regarding their experience using the application and how difficult they found the scenarios to be. Figures 5 and 6 show the results of the scenario difficulty evaluation for the experienced and inexperienced users. As we can see in the green bar in figure 5 the experienced users found on average all the scenarios easy, being the second scenario the easiest. We can directly relate these results with the time spent by the users in each scenario. The users found scenario 2 as the easiest and took less time completing the scenario. Regarding scenarios 1 and 3 the time spent was almost the same and as we can see, both scenarios were rated with an average of 4. Analyzing figure 6 we can see the inexperienced users also found that scenario 2 was very easy but found the other two more difficult when comparing to the experienced users. This is an expected difference, since users with less contact with these technologies take more time to adapt to new scenarios, but inexperienced showed throughout the tests that they were able to rapidly learn the application with only a few guidelines.

The second part of the survey has the objective of knowing the general opinion about the system after completing the tests. All the users that participated in the tests had a positive experience using the system since there were no negative or neutral responses. Regarding the final survey question three of the users felt that the information was not clear and justified their choice for the time it took to find the path to complete the first scenario. However after the initial adaptation period, they lowered the completion time in the remaining scenarios for values near the rest of the participants.
5.2.3 Summary

In the previous sections we describe the methodologies to evaluate the proposed solution. These methodologies are divided in performance and usability tests. In terms of performance, we measured the RAM and CPU usage as well as the battery power drain. Regarding the CPU the occurrence of the peeks was sparse and they had a duration of only a few seconds, so we determined that the CPU usage was not able to significantly influence the battery duration. The RAM usage results show that the average usage throughout the test was 12% for the Nexus and 10% of the heap size for the other phone. The last performance test consisted in profiling the battery usage using BatteryStats and BatteryHistorian. The results show that the estimated power consumption is 0.14% which is very low. This value can be justified by the RAM and CPU measures and the fact that the application does not need constant rendering of the views. The second part of the evaluation are the usability tests which we divided in two section, (i) Validation survey and (ii) User testing. In the first one we inquired a group of 42 people that qualified for the test we presented the idea behind this work an asked them a few questions. Overall the answers were very positive. The user testing was made with a group of experienced users and another of inexperienced users. During this tests we measure both the time to complete each scenario and the overall reaction and difficulty while using the system. The experienced users had a time reduction of 27% from the first to the second scenarios while the inexperienced users had a noticeable reduction of 47%. The inexperienced users took on average 37% more time completing the scenarios although they improved throughout the tests. Finally, in terms of satisfaction while using the system, the majority of the users found the experience very good.

6. Conclusions

Developing systems that are able to detect power outages without constant user interaction is still a world to explore, although in recent years attention to this problem has been growing. In spite most existing researches are related with monitoring systems that help users to have a better energy efficient profile, solutions that deal with power outages are emerging. This may have to do with the fact that smart houses are yet to be deployed in commercial market and most of the equipment that create a smart house are still expensive. Nevertheless, this problematic still represents an important flaw which needs to be mitigated. We present an overview of the various techniques and approaches that exist to deal with power outages, as well as solutions that focus on some of the objectives we defined for our solution. We address this problematic by presenting a system that is capable of detecting power supply failures and can alert the user, independently from his location. The solution was designed to have a low price and able to be deployed in a unmodified Android smartphone, since most existing systems require additional equipment to properly function. We evaluated the system in terms of performance and usability. In the first evaluation we evaluate the system performance in terms of RAM usage and battery duration and had results that point to a low battery drain per hour. The usability tests revealed that the majority of the inquired users would use such system. It also revealed that the users which tested the system adapted very easily and were able to reduce significantly their time of scenario completion.

References


