

# WLAN-OPP: WLAN-based Opportunistic Networking Implementation on Android

Semester Thesis

Dimitrios Gkounis

April, 2012

**Advisors:** Bernhard Distl  
**Supervisor:** Prof. Dr. Bernhard Plattner

Computer Engineering and Networks Laboratory, ETH Zurich

### **Abstract**

The goal of this semester project was to develop an Android-based application that enables opportunistic communication for WiFi-capable smartphones that are at the same location. To do this, the WiFi AP functionality is used to interconnect multiple devices. To implement different connectivity scenarios among the mobile devices within the WLAN range, two Android-based applications, instead of one, were developed. Finally, it was important for us to profile the performance of such applications by measuring the execution time of different actions that use WLAN. Thus, measurements on Google Nexus One mobile phones were conducted.

### **Acknowledgments**

I would like to thank Professor Bernhard Plattner who gave me the opportunity to conduct a semester thesis at the Communications Systems Group of the ETH Zurich. I would also like to thank Bernhard Distl for his valuable help and for the fruitful discussions we had throughout this thesis. Many thanks to all my friends who support me all these years. Last, but not least, special thanks to my family which supports me mentally and financially, gives me courage and helps me make my dreams come true.

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Applications</b>	<b>5</b>
2.1	WLAN-Opp First Try . . . . .	5
2.2	WLAN-Opp FlexibleSTA . . . . .	8
<b>3</b>	<b>Measurements</b>	<b>10</b>
3.1	MeasureTethering . . . . .	10
3.1.1	Tethering Activation Time . . . . .	10
3.1.2	Tethering Deactivation Time . . . . .	11
3.2	MeasureWiFi . . . . .	12
3.2.1	WiFi Activation Time . . . . .	13
3.2.2	WiFi Deactivation Time . . . . .	13
3.3	MeasureScanning . . . . .	14
3.4	MeasureRescanning . . . . .	15
3.5	MeasureAutoConnection . . . . .	17
3.5.1	Connection and Disconnection Time . . . . .	17
3.6	MeasureConnectionLoss . . . . .	19
3.7	Discussion . . . . .	20
<b>4</b>	<b>Conclusion</b>	<b>22</b>
<b>A</b>	<b>Developer How-To</b>	<b>23</b>
A.1	Installation procedure of development software . . . . .	23
A.2	Rooting Google Nexus One phones . . . . .	23
	<b>Bibliography</b>	<b>24</b>

# List of Figures

2.1	Menu of the application . . . . .	5
2.2	Tethering is enabled . . . . .	5
2.3	Tethering is disabled . . . . .	5
2.4	Starting AP scan mode . . . . .	6
2.5	Receiving AP scan results . . . . .	6
2.6	AP scanning stopped . . . . .	6
2.7	List item pressed – no connection . . . . .	7
2.8	Connection established . . . . .	7
2.9	List item pressed – connected . . . . .	7
2.10	Connection loss . . . . .	7
2.11	Automatic connection . . . . .	8
2.12	Automatic disconnection . . . . .	8
2.13	Fast reconnection after an automatic disconnection . . . . .	9
2.14	Automatic disconnection from another PodNetAP . . . . .	9
2.15	Connection loss . . . . .	9
2.16	Fast reconnection after a connection loss . . . . .	9
3.1	Tethering activation time . . . . .	11
3.2	Tethering deactivation time . . . . .	12
3.3	WiFi activation time . . . . .	13
3.4	WiFi deactivation time . . . . .	14
3.5	AP scan time . . . . .	15
3.6	AP rescan time . . . . .	16
3.7	Connection time . . . . .	18
3.8	Disconnection time . . . . .	18
3.9	Connection loss time . . . . .	20
3.10	Measurements . . . . .	21

# List of Tables

3.1	Tethering activation time . . . . .	11
3.2	Tethering deactivation time . . . . .	12
3.3	WiFi activation time . . . . .	13
3.4	WiFi deactivation time . . . . .	14
3.5	AP scan time . . . . .	15
3.6	AP rescan time . . . . .	16
3.7	Connection time . . . . .	17
3.8	Disconnection time . . . . .	18
3.9	Connection loss time . . . . .	20
3.10	Measurements . . . . .	21

# Chapter 1

## Introduction

The use of smartphones has been increased the past years. People more and more use their smartphones in all aspects of their daily life. In addition to using them to access to the internet, read emails, connect to calendar etc., people use smartphones to exchange pictures, music, contacts, files and data of some common applications among their peers. Thus, the interest in opportunistic communication is ever increasing.

Connectivity among smartphones is possible through Bluetooth. But this protocol doesn't enable opportunistic networking due to its short range, limited bandwidth and pairing difficulties.

Furthermore, some standards have been adopted that enable WiFi connectivity among mobile phones. These are the WiFi Ad-Hoc and WiFi Direct [1]. But, the most popular smartphone platforms don't support WiFi Ad-Hoc in stock phones up to now [2]. Besides that, this standard spends a lot of the mobile device battery energy when it is used. As for WiFi Direct, it was just designed for enabling smartphone communication over WiFi. Due to its complex pairing procedure, this standard cannot effectively enable opportunistic communication, which happens in very dynamic environments. So, a new architecture has been proposed for this purpose. WiFi-Opp is a WiFi-based opportunistic networking architecture for smartphones [3]. WiFi-Opp uses the WiFi Access Point (AP) feature (Tethering, Personal Hotspot) of smartphones, which was originally designed to allow devices to share Internet access, to enable connectivity among devices that are at the same place, i.e. in range of WiFi.

This thesis is a part of the WiFi-Opp project (also called WLAN-Opp) of the Communications Systems Group (CSG) at ETH Zurich. This thesis takes the existing proof of concept code and develops different Android-based applications corresponding to different opportunistic networking scenarios [3].

This thesis is organized as follows: In Chapter 2, the applications developed for implementing multiple connectivity scenarios in opportunistic networking are described. In Chapter 3, the performance of some of the application components in terms of time to be executed is measured. The objects measured, the methods used for and the results of the measurements are presented. Finally, in chapter 4, a conclusion of this thesis is written.

# Chapter 2

## Applications

In this part of the thesis, the applications developed to enable opportunistic networking on Android will be explained in detail. The first application, WLAN-Opp First Try, was developed just for integrating all the modules necessary for enabling opportunistic networking on Android-based smartphones. This application is also an implementation of a simple connectivity case in opportunistic networks. The second one, WLAN-Opp FlexibleSTA, implements a particular connectivity scenario regarding opportunistic communication.

### 2.1 WLAN-Opp First Try

This application was developed as an attempt to use all the features needed for enabling opportunistic networking on Android-based mobile devices. The design of the application is similar as the WiFi-Opp Manual scenario on [3].

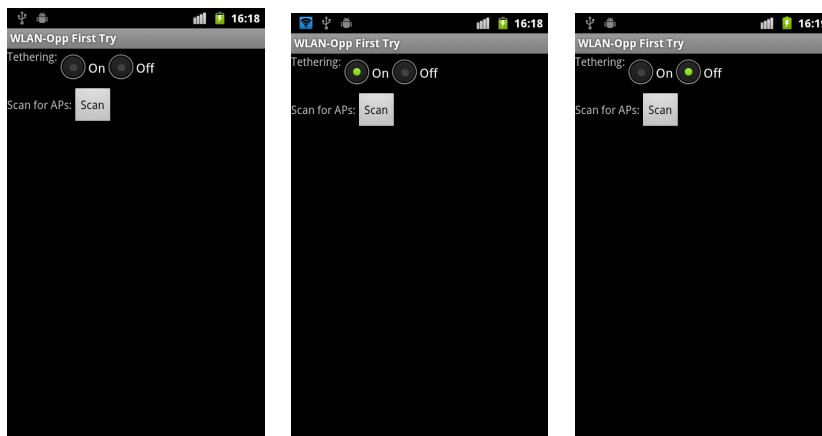


Figure 2.1: Menu of the application

Figure 2.2: Tethering is enabled

Figure 2.3: Tethering is disabled

When this application starts running, a screen appears so the user can choose between using the Tethering mode and using the WiFi mode (Fig. 2.1). If the user chooses to use the Tethering mode, i.e. to become a mobile Access Point



(AP) using WiFi, then he can enable it by using the appropriate radio button (enabling one, disables the other) (Fig. 2.2). When Tethering is enabled, an AP with PodNetAP as its SSID is created. The user can also disable this mode by the use of the same radio button (Fig. 2.3). If the user wants to use the WiFi mode, i.e. to scan for the available APs in range, then he has to press the Scan button.

If the user chooses to scan for APs, then a new screen appears. The WiFi gets enabled (Fig. 2.4), if it is not, and the screen fills up with a list of all available APs in range (Fig. 2.5). This list is scrollable, so the user can be provided with the information of all possible APs around. Each entry in the list is comprised by three features. At the top in each entry, the SSID of AP is displayed. Below that, the authentication, key management and encryption schemes supported by this AP appear. In the last line of each list item, a number that corresponds to the signal level in dBm of the AP is shown. All the APs in list are sorted based on the strongest signal level, i.e. APs with higher signal level are placed higher in the list than others. It has to be noticed that only the AP with the strongest signal level among multiple APs with the same SSID is displayed in the list.

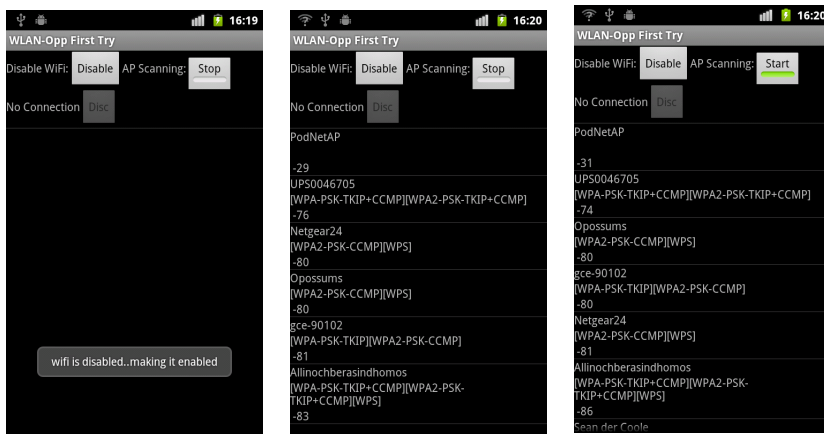


Figure 2.4: Starting AP scan mode      Figure 2.5: Receiving AP scan results      Figure 2.6: AP scanning stopped

As it can be understood by using the application, the mobile device receives results about the APs in range really fast, approximately two times per second. The user is given the possibility to stop receiving AP scan results by pressing the STOP button (Fig. 2.6). The list stops getting updated and thus the user can choose to connect to the AP he wants. It has to be mentioned that this button is a toggle button, so after it has been pressed, the button gets to the state START. If the user presses it again, the AP scanning starts again.

In order to connect to an AP, an item of the AP list has to be pushed on screen. When that happens, a dialog box appears on screen to ask the user if he wants to connect to this AP (Fig. 2.7). If the user chooses Yes, a connection attempt is made. If he chooses No, the application returns to its previous state. It has to be mentioned that the device, when using this application, can only connect to an open AP. By default, there is a display on screen (No Connection)

that states that the device is not connected to an AP. It is assumed that no preconfigured network exists on the device. When a connection to an AP has been established, this display changes showing the SSID of the AP to which the device is connected (e.g. Connected to PodNetAP) (Fig. 2.8). Then, if a list item is pushed, a different dialog box appears to state that the device is already connected to a specific AP (Fig. 2.9). This means that the application doesn't allow the user to connect to another AP while connected. If he wants to connect to a different AP, he has to be disconnected from the other one first.

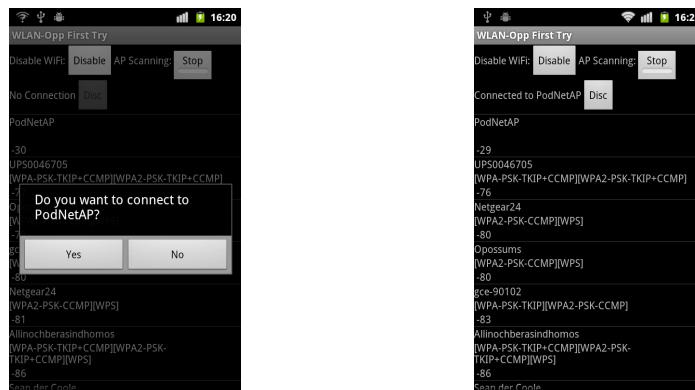


Figure 2.7: List item pressed – no connection      Figure 2.8: Connection established

When a connection to an AP has been established, a Disc button, i.e. for disconnecting from the AP, is also enabled next to the display regarding the connected AP. If a disconnection happens, this button becomes disabled and the display next to it turns to its default state. In case the device or the AP to which is connected gets out of each other's range or the AP gets disabled due to a failure, a small message appears on screen stating that the connection has been lost (Fig. 2.10).

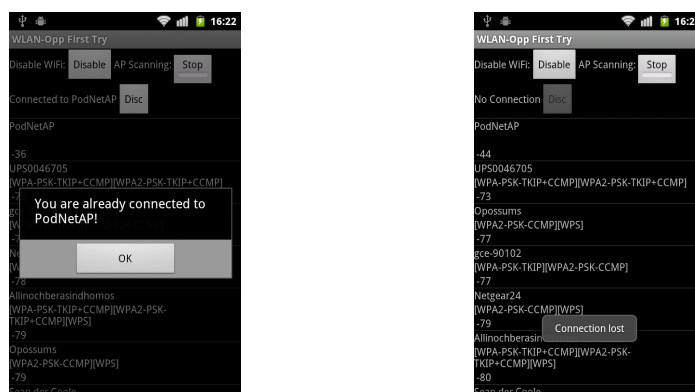


Figure 2.9: List item pressed – connected      Figure 2.10: Connection loss

The user has also the choice to abort the WiFi mode and to switch to the

first screen where as explained he can choose between the WiFi mode and the Tethering mode for his device. This is done by just pressing the Disable button whenever the user wants to, even if his device is already connected to an AP. It is considered in this implementation that the user will first disable the WiFi before trying to use Tethering. He could just press the Back button, resulting in returning in the first screen of the application, but he wouldn't be able to access to the Tethering mode as the WiFi would still be enabled.

It is also considered that when the device is in the Tethering mode, the user doesn't try to enter to the WiFi mode. So, it is assumed that the user disables the Tethering mode before trying to access to the WiFi mode.

## 2.2 WLAN-Opp FlexibleSTA

The implementation scenario was based on WiFi-Opp Flexible STA scenario that was presented on [3]. This application is similar to the WLAN-Opp First Try application but it has been developed taking only into account the WiFi mode of the previous application, i.e. the Tethering mode has not been integrated into this application.

In this application, the device that is connected to an AP (such device is called STA) is automatically disconnected from this AP after some time has passed. The time that the STA stays connected to an AP is defined as  $t_{con} \sim U[t_{con_{min}}, t_{con_{max}}]$ . A STA that stays connected to an AP for a long time is not able to connect to other APs available in range. So, this scenario is defined in order not to have "multiple independent clusters that cannot communicate although they are physically collocated" [3].

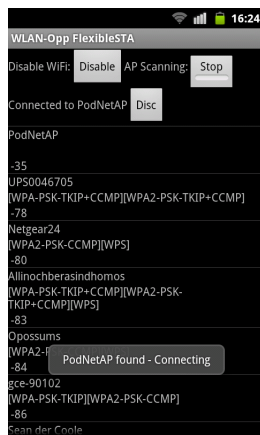


Figure 2.11: Automatic connection

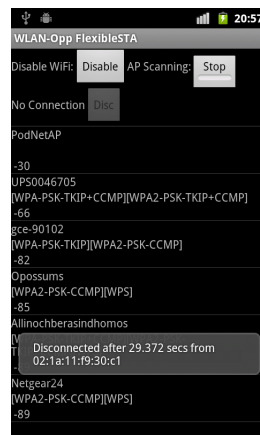


Figure 2.12: Automatic disconnection

When this application starts running, it automatically tries to connect to PodNetAP (the SSID used by the Tethering mode of the previous application) (Fig. 2.11). If such an open network doesn't exist in range, the user can also choose to connect to any other open network he likes by just pressing the corresponding item of the AP list displayed on screen, like in the previous application. In all cases, i.e. to whichever AP the STA is connected, the STA stays connected to this AP for just  $t_{con}$  (Fig. 2.12, 2.14). After this time has elapsed, if the

STA is connected to a PodNetAP, the STA gets disconnected from this PodNetAP and immediately tries to find another PodNetAP (same SSID, different BSSID) to connect to. If it finds one, a fast reconnection happens (Fig. 2.13). If it doesn't, the device just keeps scanning for the available APs in range. In case a connection loss is detected due to an AP (PodNetAP) failure or out of range communication, a fast reconnection to another PodNetAP is also attempted (Fig. 2.15, 2.16). Thus, the fast reconnection feature works only if the STA is previously connected to a PodNetAP. This feature is used to help the device quickly adjust to topological changes due to mobility [3].

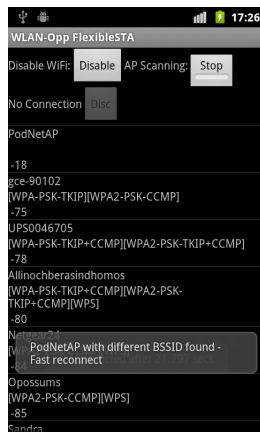


Figure 2.13: Fast reconnection after an automatic disconnection

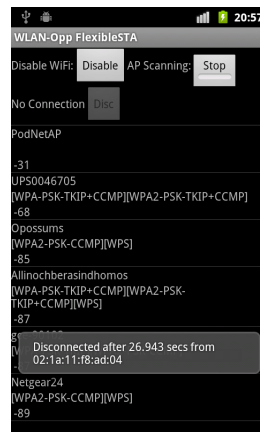


Figure 2.14: Automatic disconnection from another PodNetAP

For all the functionalities described above, small messages appear on screen as in the other application. Besides some additional functionalities that were described in this section, the application works and appears in the same way as the previous one.

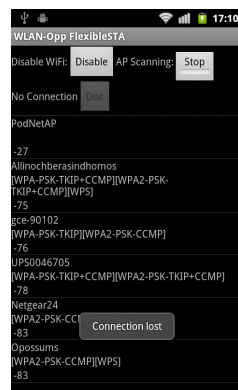


Figure 2.15: Connection loss



Figure 2.16: Fast reconnection after a connection loss

## Chapter 3

# Measurements

In this part of the thesis, the time it takes a defined set of WiFi-related actions of the aforementioned applications to be executed is measured. For each measurement, a small Android application was developed. All the measurements, except the last one, were made on four stock (unrooted) Google Nexus One phones running Android 2.3.6. In the following sections, a description of the applications developed and of the methods used for the measurements will be given.

### 3.1 MeasureTethering

This application was developed in order to measure the time it takes tethering to be enabled and disabled. For enabling, disabling and checking whether tethering is enabled or not, some custom made methods were used, `enableTethering()`, `disableTethering()` and `tetheringIsEnabled()` respectively, as there are no Android APIs for these purposes.

#### 3.1.1 Tethering Activation Time

To measure tethering activation time, tethering is first enabled by executing `enableTethering()` and this current time is stored. To find out when tethering is really enabled, `tetheringIsEnabled()` is executed every 10 milliseconds. When tethering gets enabled, the time difference between the current time and the time previously stored produces the tethering activation time. This value is then stored on a particular file (`TetheringON.txt`) inside a specific folder (`/measurements`) hosted at the SD card of the mobile device. After this measurement has been made, tethering gets disabled by executing `disableTethering()`. After a short period of time (15 seconds), tethering is enabled again and so on and so forth. This situation happens many times iteratively so as many samples to be collected for determining the tethering activation time of Google Nexus One phones.

The time interval (15 seconds) for re-enabling tethering was determined by some measurements made to choose the right time interval. If a shorter time interval is used, e.g. 5 or 10 seconds, tethering doesn't work properly and takes

it much more time to be enabled than in the case of using 15 seconds or more for the reactivation time interval.

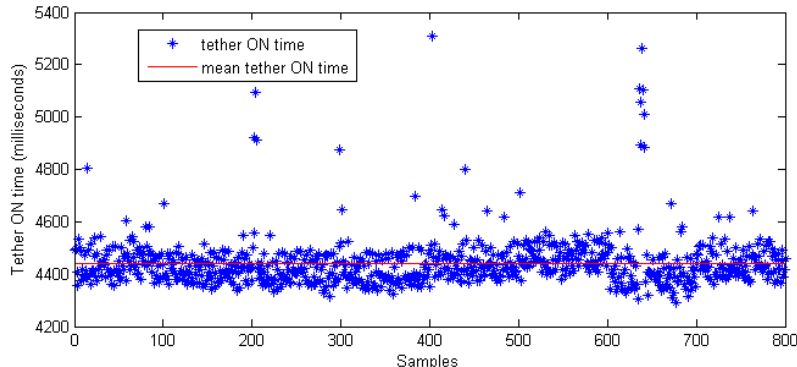


Figure 3.1: Tethering activation time

Figure 3.1 shows the tethering activation time of the collected samples. On the horizontal axis, we have the samples and on the vertical axis the tethering activation time, measured in milliseconds, that corresponds to each sample. The red line corresponds to the mean value of the tethering activation time, based on all collected samples.

Tethering Activation Time (milliseconds)	
Mean Value	4442.6
Standard Deviation	97.5316

Table 3.1: Tethering activation time

Based on Table 3.1, we can say that the tethering activation time is approximately 4.44 seconds.

### 3.1.2 Tethering Deactivation Time

To measure tethering deactivation time, a similar technique as when measuring tethering activation time is used. Tethering is first enabled by executing `enableTethering()`. After a short period of time while tethering has been enabled, tethering gets disabled by executing `disableTethering()` and the current time is stored. To measure when tethering is really disabled, `tetheringIsEnabled()` is executed every 1 millisecond. When tethering is disabled, the time difference between the current time and the time previously stored produces the tethering deactivation time. This value is then stored on a particular file (`TetheringOFF.txt`) in the measurements folder (`/measurements`) hosted at the SD card of the mobile device. After some time has passed, tethering gets enabled again and so on and so forth. This situation happens many times iteratively so as many samples to be collected for determining the tethering deactivation time of Google Nexus One phones.

Figure 3.2 shows the tethering deactivation time of the collected samples. On the horizontal axis, we have the number of the samples and on the vertical

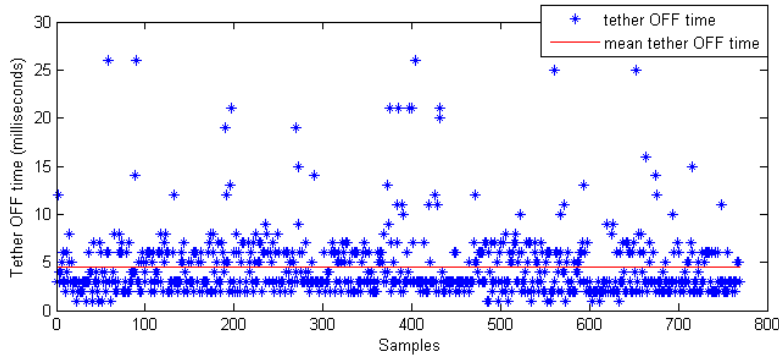


Figure 3.2: Tethering deactivation time

axis the tethering deactivation time, measured in milliseconds, that corresponds to each sample. The red line corresponds to the mean value of the tethering deactivation time, based on all collected samples.

Tethering Deactivation Time (milliseconds)	
Mean Value	4.4590
Standard Deviation	3.3814

Table 3.2: Tethering deactivation time

Based on Table 3.2 , we can conclude that the tethering deactivation time is approximately 4.46 milliseconds. It has to be noticed, based on Figure 3.2 , that some samples have unexpectedly high values. This is maybe because the Android OS executes sometimes multiple processes in parallel with Tethering. Thus, the mean value should be a little bit lower than this at Table 3.2. This is also why the standard deviation seen at Table 3.2 has so high value in comparison with the mean value.

In order to measure the tethering activation and deactivation time, two parts of code were used in this application. In other words, these measurements were made independently. At first, tethering activation time was measured and after this had finished, tethering deactivation time was. It was feasible to make these two kinds of measurements simultaneously but unexpected results received. This happened maybe because of the same reason explained before regarding the values on Table 3.2.

## 3.2 MeasureWiFi

This application was developed in order to measure the time it takes WiFi to be enabled and disabled. For enabling, disabling and checking whether WiFi is enabled or not, the public methods `setWifiEnabled(true)`, `setWifiEnabled(false)` and `isWifiEnabled()` of the `WifiManager` class of Android APIs were used respectively [4]. The techniques used for these measurements are pretty much the same as in the tethering case.

### 3.2.1 WiFi Activation Time

To measure WiFi activation time, WiFi is enabled by using `setWifiEnabled(true)` and this current time is stored. To find out when WiFi is really enabled, `isWifiEnabled()` is executed every 10 milliseconds. When WiFi gets enabled, the time difference between the current time and the time previously stored produces the WiFi activation time. This value is then stored on a particular file (`WiFiON.txt`) in the same folder as when using tethering. After this measurement has been made, WiFi gets disabled by using `setWifiEnabled(false)`. After a short period of time (15 seconds for the same reason as in tethering activation time case), WiFi is enabled again and so on and so forth. This situation happens many times iteratively so as many samples to be collected for determining the WiFi activation time of Google Nexus One phones.

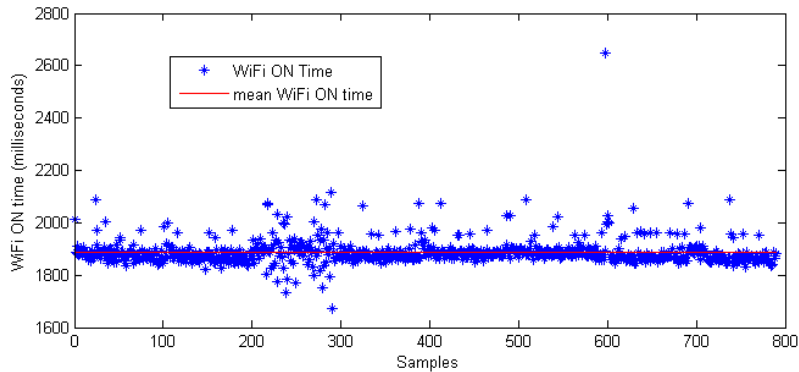


Figure 3.3: WiFi activation time

Figure 3.3 shows the WiFi activation time of the collected samples. On the horizontal axis, we have the number of the samples and on the vertical axis the WiFi activation time, measured in milliseconds, that corresponds to each sample. The red line corresponds to the mean value of the WiFi activation time, based on all collected samples.

WiFi Activation Time (milliseconds)	
Mean Value	1887.3
Standard Deviation	50.8064

Table 3.3: WiFi activation time

Based on Table 3.3, the WiFi activation time is approximately 1.89 seconds.

### 3.2.2 WiFi Deactivation Time

To measure WiFi deactivation time, a similar technique as when measuring WiFi activation time is used. WiFi is first enabled by using `setWifiEnabled(true)`. After a short period of time while WiFi has been enabled, WiFi gets disabled by using `setWifiEnabled(false)` and the current time is stored. To measure when WiFi is really disabled, `IsWifiEnabled()` is used every 1 millisecond. When WiFi is disabled, the time difference between the current time and the time previously



stored produces the WiFi deactivation time. This value is then stored on a particular file (WiFiOFF.txt) in a specific folder (/measurements) hosted at the SD card of the mobile device. After some time has passed, WiFi gets enabled again and so on and so forth. This situation happens many times iteratively so as many samples to be collected for determining the WiFi deactivation time of Google Nexus One phones.

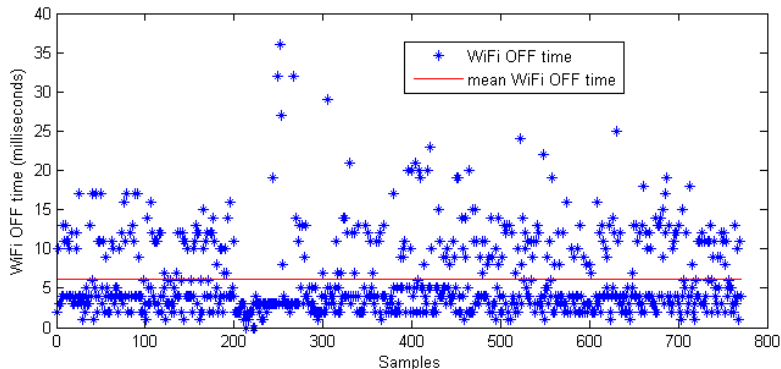


Figure 3.4: WiFi deactivation time

Figure 3.4 shows the WiFi deactivation time of the collected samples. On the horizontal axis, we have the number of the samples and on the vertical axis the WiFi deactivation time, measured in milliseconds, that corresponds to each sample. The red line corresponds to the mean value of the WiFi deactivation time, based on all collected samples.

WiFi Deactivation Time (milliseconds)	
Mean Value	6.2234
Standard Deviation	5.0382

Table 3.4: WiFi deactivation time

Based on Table 3.4, the WiFi deactivation time is approximately 6.2 milliseconds. The same conclusions drawn when measuring tethering deactivation time are also valid here for the very same reason.

This application was also consisted of two parts, like when measuring tethering activation and deactivation time. This also happened for the same reason as in that case.

### 3.3 MeasureScanning

This application was developed in order to measure the time it takes the device to receive results about the available Access Points (APs) in range. For this measurement, there should be no preconfigured networks in range so as no interruption in scanning APs happens.

The WiFi gets enabled and then a receiver (Broadcaster Receiver) is registered for getting the scan results when these are ready. When the receiver

is enabled, i.e. when the AP results are received, this current time is stored. When the receiver is enabled again, the time difference between this time and the time previously stored produces the scan time. This value is then stored on a particular file (Scanning.txt) in the common folder created for all types of measurements, which is hosted at the SD card of the mobile device. Also, the time that the scan results were received for second time is also recorded to be used for comparison with the time the receiver gets enabled another time. This happens many times and thus many samples are collected to be used for calculating the scan time of Google Nexus One phones.

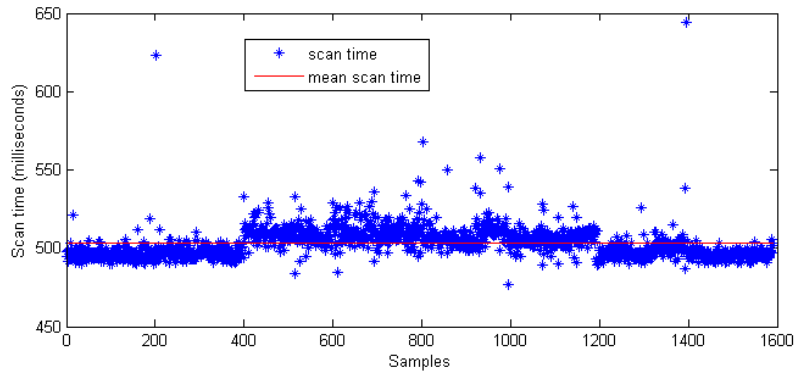


Figure 3.5: AP scan time

Figure 3.5 shows the scan time of the collected samples. On the horizontal axis, we have the number of the samples and on the vertical axis the scan time, measured in milliseconds, that corresponds to each sample. The red line corresponds to the mean value of the scan time, based on all collected samples.

Scan Time (milliseconds)	
Mean Value	503.1667
Standard Deviation	9.7853

Table 3.5: AP scan time

Based on Table 3.5, the AP scan time is approximately 503.2 milliseconds. By observing Figure 3.5, it is clear that the first and the last 400 samples collected have lower scan time than that of the 800 samples collected in between. A different Google Nexus One phone was used for this measurement for each 400 samples seen on Figure 3.5. Thus, this might be due to the fact that there are Nexus One phones with different WLAN chipsets around, but we cannot reliably identify them.

### 3.4 MeasureRescanning

As it was presented in previous chapter, there was a toggle button in those applications for starting and stopping the AP scan. This one application was created to measure the time for getting AP scan results for first time by the

device after the START AP scan button has just been pressed. This assumes that the button was previously in the state STOP AP scan. Like in the previous measurement, there should be no preconfigured networks in range so as no interruption in scanning APs happens.

The WiFi gets enabled. Then, a receiver (Broadcaster Receiver) is registered for getting the scan results when these are ready. This is the same as pressing the START AP scan button. The time that the receiver is registered (button is pressed) is logged. When the receiver is enabled, i.e. when the AP results are ready, this current time is compared with the previous logged time and their time difference is the rescan time. This value is then stored on a particular file (`Rescanning.txt`) in the same folder on the mobile device as in the previous measurements. The receiver then gets unregistered, which means that no other AP scan results will be received. After a small amount of time has elapsed, the receiver is registered again and so on and so forth. This happens many times iteratively and thus many samples are collected to be used for calculating the rescan time of Google Nexus One phones.

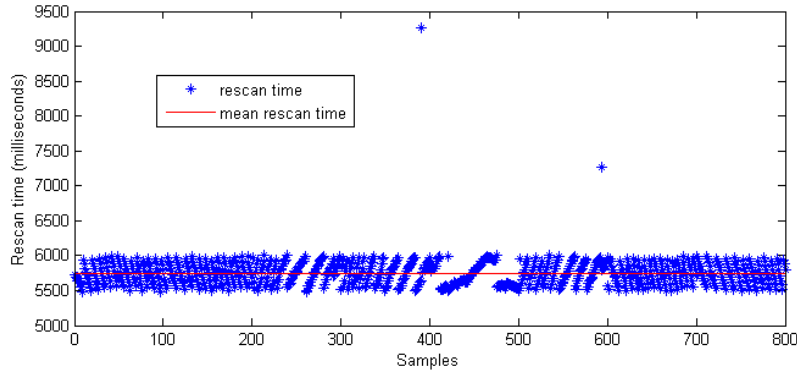


Figure 3.6: AP rescan time

Figure 3.6 shows the rescan time of the collected samples. On the horizontal axis, we have the number of the samples and on the vertical axis the rescan time, measured in milliseconds, that corresponds to each sample. The red line corresponds to the mean value of the rescan time, based on all collected samples.

Rescan Time (milliseconds)	
Mean Value	5749.9
Standard Deviation	200.4969

Table 3.6: AP rescan time

Based on Table 3.6, the AP rescan time is approximately 5.75 seconds. By observing Figure 3.6, it is clear that the first and the last 200 samples collected have the same pattern in comparison with the 400 samples in the middle. Thus, the same conclusion can be drawn like in the previous application. The rescan time either seems to start slow and speed up or to start fast and slow down in regular intervals.

## 3.5 MeasureAutoConnection

This application was developed for determining the time it takes the device to connect to an AP, after the AP scan results have been received. This application also measures the time it takes the device to detect a disconnection. This assumes that a disconnection from an AP happens on the device by users choice (by pressing the Disc button on the applications presented in previous chapter). Like in previous measurements, there should be no preconfigured networks in range because a particular AP will only be used for the measurements.

### 3.5.1 Connection and Disconnection Time

The WiFi is enabled and both a receiver for getting AP scan results and a receiver that detects changes in the connection state of the device are registered. When the AP scan receiver is enabled, i.e. when the results of the available APs in range have been received, the current time is logged and the device is forced to connect to a particular AP. To be specific, it connects to PodNetAP which is created by using the tethering mode on another mobile device. When the appropriate receiver detects that a connection has been established, the current time is logged and the time difference of this and the previously logged time is the connection time. This value is then stored on a particular file (AutoConnection.txt) in the measurements folder (/measurements) hosted at the SD card of the mobile device. The device stays connected for a short period of time to this AP and then gets automatically disconnected by the use of a particular method of the WifiManager class of Android (it is the same as pressing a Disc button) [4]. The time when this method is executed is recorded. After that, the receiver that detects connection state changes is enabled and a disconnection is detected. By comparing this time and the previously recorded time, the disconnection time is created. This value is then stored on a particular file (AutoDisconnection.txt) in the same folder as before. After some time has passed, the AP scan receiver gets enabled and the device is forced again to connect to PodNetAP and so on and so forth. It has to be noticed that the device is not allowed to connect to PodNetAP just after the disconnection happened, although it is allowed to in general. This is because the AP is not immediately removed from the list with the preconfigured networks on the device and thus a quick reconnection complicates the whole scenario and unexpected results are received. This scenario is repeated many times so as many samples are collected to be used for calculating the connection and disconnection time of Google Nexus One phones.

Figure 3.7 shows the connection time of the collected samples. On the horizontal axis, we have the number of the samples and on the vertical axis the connection time, measured in milliseconds, that corresponds to each sample. The red line corresponds to the mean value of the connection time, based on all collected samples.

Connection Time (milliseconds)	
Mean Value	573.304
Standard Deviation	24.2161

Table 3.7: Connection time

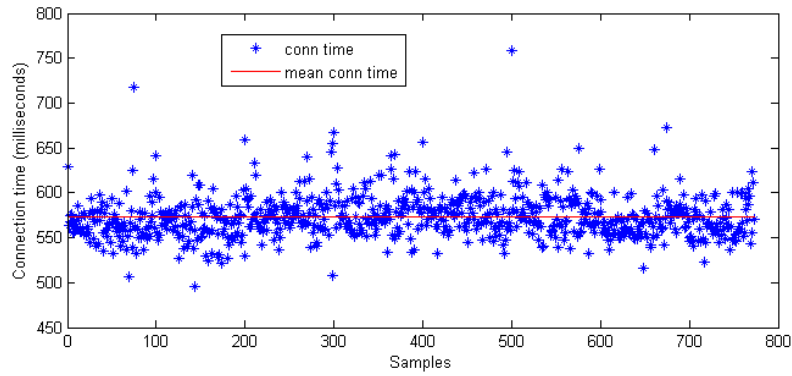


Figure 3.7: Connection time

Based on Table 3.7, the connection time is approximately 573 milliseconds.

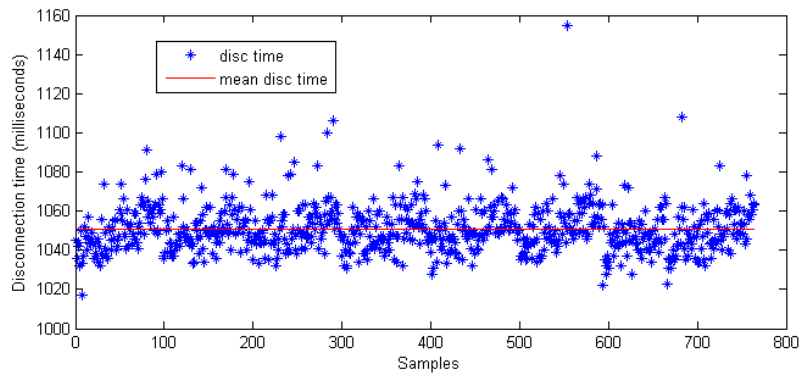


Figure 3.8: Disconnection time

Figure 3.8 shows the disconnection time of the collected samples. On the horizontal axis, we have the number of the samples and on the vertical axis the disconnection time, measured in milliseconds, that corresponds to each sample. The red line corresponds to the mean value of the disconnection time, based on all collected samples.

Disconnection Time (milliseconds)	
Mean Value	1050.8
Standard Deviation	12.1360

Table 3.8: Disconnection time

Based on Table 3.8, the disconnection time is approximately 1.05 seconds.

### 3.6 MeasureConnectionLoss

This measurement was the last one that was made. The purpose of this was to measure how much time it takes one device that is connected to an AP to detect that the connection is lost due to AP failure or because of the one of these two communicating parties went out of range. In this measurement, there weren't used stock (unrooted) Google Nexus One phones. Furthermore, there were just used two out of the four phones used in all previous measurements. The reason why this happened is because in general smartphones don't have the same system time, i.e. smartphones are not time synchronized to each other. To enable time synchronization among smartphones, a Network Time Protocol (NTP) client should be installed to them. By searching on Android Market, a particular application was found for this purpose. ClockSync can synchronize the system time of an Android-based phone with atomic time using NTP over 3G or WiFi connection. The problem about using this application was that the devices must be rooted for automatic synchronization. So, phones were rooted and CyanogenMod 7 (7.1.0-N1) was installed on them. Two devices were only used because there were only two SIM cards with 3G connection available. A 3G instead of a WiFi connection was used by each device for the synchronization. This happened due to the scenario of the application developed for the measurement. As in previous measurements, there should be no preconfigured networks in range because a particular AP (e.g. PodNetAP) will be used for the measurements. Thus, a 3G connection was used. In case a WiFi connection was used, then this measurement couldn't have happened due to the nature of the scenario.

To measure the time for the connection loss detection, tethering is enabled on the one phone and WiFi is enabled on the other. Also, a receiver which listens for connection state changes is registered on the second device. The WiFi enabled device (called STA from now on) is forced to connect to the AP (e.g. PodNetAP) provided by the tethering enabled device (called AP from now on). After the connection has been established and some time has passed, the tethering on the AP device is disabled. The time that the tethering is deactivated is logged and stored on a particular file (ConnectionLossAP.txt) in the measurements folder at the root of the SD card of the AP device. When the connection loss is detected by the receiver that detects connection changes on the STA device, this current time is also logged and stored on a particular file (ConnectionLossSTA.txt) in the measurements folder hosted at the SD card of the STA device. After a short period of time, tethering is enabled again and the whole scenario starts from the beginning. This scenario runs many times iteratively and many samples are collected. By further processing of the samples after the measurement has finished, i.e. by subtracting the corresponding logs on the two files, the connection loss time of Google Nexus One phones is found.

Figure 3.9 shows the connection loss time of the collected samples. On the horizontal axis, we have the number of the samples and on the vertical axis the connection loss time, measured in milliseconds, that corresponds to each sample. The red line corresponds to the mean value of the connection loss time, based on all collected samples.

Based on Table 3.9, the connection loss time is approximately 61 milliseconds.

By observing Figure 3.9, it is clear that many samples have negative values

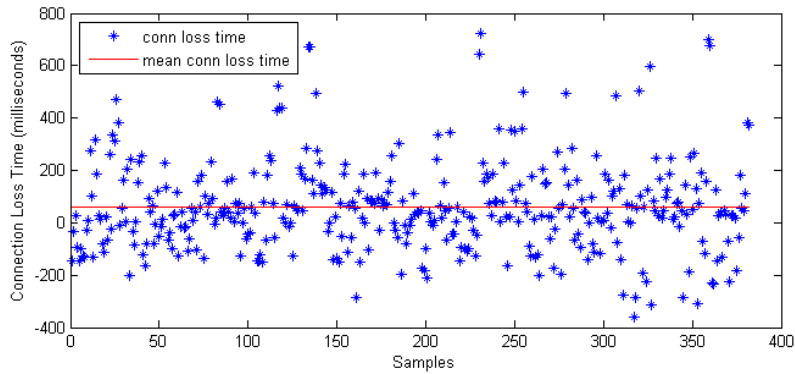


Figure 3.9: Connection loss time

Connection Loss Time (milliseconds)	
Mean Value	60.9791
Standard Deviation	174.0652

Table 3.9: Connection loss time

and some others have high values. This happens due to time synchronization loss between the two mobile devices that were used. Clock Sync application is able to automatically synchronize the devices with an NTP server every 30 seconds. This time interval is adequate to produce an average offset of  $\pm 200$  milliseconds on every device. That is why this measurement is not that accurate. To have as more realistic results as it could be, some measurements that had high values (positive or negative) were not taken into account. However, the standard deviation at Table 3.9 has still a large value in comparison with the mean value. Nevertheless, it is true that the connection loss time is too short as it can be observed during making this measurement by taking samples. Thus, these results can be considered as good qualitative results regarding the connection loss time on Google Nexus One phones.

### 3.7 Discussion

Based on the measurements which were described in the previous sections, a table (Table 3.10) which contains the mean values of the measurements and a bar chart (Fig. 3.10) which contains the mean values and error bars based on the standard deviation of the corresponding measurements are shown below. The error bars were added on the bar chart to indicate how reliable the mean values of the measurements are.

By the use of the bar chart (Fig. 3.10), some interesting conclusions can be derived. For example, the rescan time has very high value in comparison with the other WLAN-related actions which were measured. Also, the deactivation time of WiFi and Tethering is too short in comparison with the values of the other measurements. Furthermore, the value of Tethering activation time is more than twice as long as that of the WiFi activation time. In addition, the time it takes the device to detect a disconnection from a network (Disconnection

Measurement	Mean Value (milliseconds)
Tethering Activation Time	4442.6
Tethering Deactivation Time	4.4590
WiFi Activation Time	1887.3
WiFi Deactivation Time	6.2234
Scan Time	503.1667
Rescan Time	5749.9
Connection Time	573.304
Disconnection Time	1050.8
Connection Loss Time	60.9791

Table 3.10: Measurements

time) is higher than the time it takes a device to detect a connection to a network (Connection time).

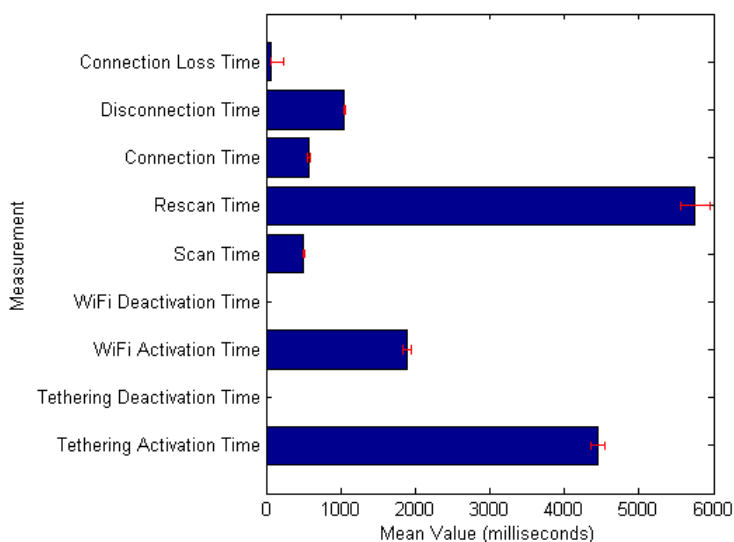


Figure 3.10: Measurements

In an opportunistic networking environment as in [3], many devices switch their WiFi state from WiFi mode to Tethering (AP) mode and the other way around and they connect and disconnect to multiple devices all the time. Observing Figure 3.10 and Table 3.10, it can be understood that the high values (comparing to the values of the other WiFi-related actions) of both the WiFi and Tethering activation time can consist a limitation in networking in a highly dynamic environment, in which devices change their location fast.



## Chapter 4

# Conclusion

In this thesis, applications which enable opportunistic networking on Android-based smartphones were developed. To make this possible, the mobile AP mode of smartphones was used. The design of the applications was based on a WiFi-based opportunistic networking architecture for smartphones which was proposed on [3]. Measurements on some WiFi-based functions of the applications were then conducted. It was important to measure the performance of these WiFi-related modules of the applications. For the measurements, Google Nexus One mobile devices were used. Based on the measurements, appropriate conclusions were derived. The factors which impact the application performance were identified. Also, while making measurements, it was noticed that Android-based smartphones are not time synchronized. There was a difficulty to synchronize them while making a particular measurement. This fact affected the results of the corresponding measurement.

# Appendix A

## Developer How-To

### A.1 Installation procedure of development software

- Download and install the Eclipse Classic development environment, Eclipse 3.6.2 (Helios) or greater, from: <http://www.eclipse.org/downloads/>
- Download and install the Java JDK 6, if your system doesn't have it already, from: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- Download and install the Android SDK from: <http://developer.android.com/sdk/index.html>
- Download, install and configure the ADT plugin for Eclipse from: <http://developer.android.com/sdk/eclipse-adt.html#installing>
- Add platforms and SDK components from: <http://developer.android.com/sdk/adding-components.html>

### A.2 Rooting Google Nexus One phones

In order to root Google Nexus One phones and install CyanogenMod, we follow the steps explained on: [http://wiki.cyanogenmod.com/wiki/Nexus\\_One:\\_Full\\_Update\\_Guide](http://wiki.cyanogenmod.com/wiki/Nexus_One:_Full_Update_Guide)

# Bibliography

- [1] WiFi Alliance. Wi-fi direct. *<http://www.wi-fi.org/discover-and-learn/wi-fi-direct>*.
- [2] Android issue 82: wifi - support ad hoc networking. *<http://code.google.com/p/android/issues/detail?id=82>*.
- [3] D. Schatzmann S. Trifunovic, B. Distl and F. Legendre. Wifi-opp: Ad-hoc-less opportunistic networking. *ACM Chants*, 2011.
- [4] Wifimanager. *<http://developer.android.com/reference/android/net/wifi/WifiManager.html>*.