Sánchez, M. Isabel; Oliva, Antonio de la; Bernardos, Carlos J. (2016), Experimental analysis of connectivity management in mobile operating systems, *Computer Networks*, v. 94, pp.: 41-61.

DOI: https://doi.org/10.1016/j.comnet.2015.11.021

Experimental analysis of connectivity management in mobile operating systems

M. Isabel Sanchez[a,b], Antonio de la Oliva[b], Carlos J. Bernardos[b]

[a]*IMDEA Networks Institute*
[b]*Departamento de Ingeniería Telemática, Universidad Carlos III of Madrid*

Abstract

We are immerse in a world that becomes more and more mobile every day, with ubiquitous connectivity and increasing demand for mobile services. Current mobile terminals support several access technologies, enabling users to gain connectivity in a plethora of scenarios and favoring their mobility. However, the management of network connectivity using multiple interfaces is still starting to be deployed. The lack of smart connectivity management in multi-interface devices forces applications to be explicitly aware of the variations in the connectivity state (changes in active interface, simultaneous access from several interfaces, etc.). In this paper, we analyze the present state of the connection management and handover capabilities in the three major mobile operating systems (OSes): Android, iOS and Windows. To this aim, we conduct a thorough experimental study on the connectivity management of each operating system, including several versions of the OS on different mobile terminals, analyzing the differences and similarities between them. Moreover, in order to assess how mobility is handled and how this can affect the final user, we perform an exhaustive experimental analysis on application behavior in intra- and inter-technology handover. Based on this experience, we identify open issues in the smartphone connectivity management policies and implementations, highlighting easy to deploy yet unimplemented improvements, as well as potential integration of mobility protocols.

Keywords: Mobility, handover, smartphone, Android, Windows Phone, iOS, connection manager.

1. Introduction & Motivation

In the last years we have witnessed the market explosion of a new kind of handheld device, the smartphone. At this point in time it is clear that these devices are going to keep a steady market penetration and are going to be a major source of revenue and data traffic for network operators. According to [1], smartphones account for 88 percent of the growth of global mobile devices and connections in 2014, with 439 million net additions in this year. Even though they represent just the 29 percent of global handsets in use, they are responsible for 69 percent of the total handset traffic, which clearly shows the major adoption of this technology. The use of smartphones has also modified the traditional patterns of mobile data consumption. The typical smartphone user is craving for data-based services, imposing a high burden on the operators, which see their investments on network deployments pushed to the limits due to greater bandwidth requirements. Due to the shift in user profile and data service demand experienced in the recent years, smartphones have become a powerful tool in most people's daily life. In addition, the enhanced capabilities and fast upgrades of hardware in handheld devices have considerably increased their usage. These facts pave the way to advanced research and development that relies on the use of smartphones for carrying out innovative tasks, mostly related to health care or behavioral studies and using the smartphone as a measurement instrument [2]. Moreover, there is a trend towards specialized, almost personalized services, which could benefit from an accurate knowledge on the capabilities supported by smartphones and how they manage their resources.

The current offer in the market is very wide, in terms of manufacturers, hardware resources and operating systems, but do the majority of the devices behave similarly? Is there any difference in the management of their own resources? Is the access to an application homogenized across the different systems? Is the network connectivity management standardized in all the various devices? Answering these questions could back the actual research on mobile devices and their applications to make daily tasks easier, but also more demanding use cases, applied to different fields. However, mainly because these details remain closed by manufacturers, most of this information is still missing or disregarded in research, which focuses on measuring performance or developing applications to serve specific purposes.

Smartphone users have one common and defining characteristic, they move. The need for supporting data services on the move has shaped the design of the cellular network, which must deploy access and core networks able to redirect users' traffic to their current location. In addition to the obvious problem of designing such networks,

Email addresses: `mariaisabel.sanchez@imdea.org` (M. Isabel Sanchez), `aoliva@it.uc3m.es` (Antonio de la Oliva), `cjbc@it.uc3m.es` (Carlos J. Bernardos)

current smartphones support several wireless technologies, such as IEEE 802.11 [3], complementing the cellular connection. This heterogeneity brings new opportunities and challenges to the industry, since the additional technologies can be used to offload the traffic from the cellular network to local accesses, such as the broadband connection usually deployed at home.

The smartphone operating system provides a set of mobility-related functions from which an application can benefit in case it decides to handle mobility. These functionalities depend on the operating system (e.g., Android, iOS, Windows Phone 8) and include connectivity events such as network up or down events, and commands exposed to the application layer to extract information on connection availability. Usually the terminal connectivity is handled by a service widely known as the *Connection Manager*, in charge of deciding which is the best connection for the terminal in a specific moment, and the application has to deal with those decisions.

This work focuses on the analysis of the current state of the art on the mobility support at the Connection Manager in different terminals, providing a functional view of the differences between the major operating systems and the different improvements that can be done to optimize the mobile user experience. Our main contributions are:

- We analyze the default network connectivity management of the three currently most popular families of mobile OSes: Android, iOS and Windows Phone 8,[1] including iOS8 and Android Lollipop, in their latest versions supported to date. We test the same OS versions in different terminals, to avoid biased conclusions, derived from the performance of the terminal rather than from the OS behavior.

- We study how the smartphones running these different OSes perform inter- and intra-technology handover, considering the most widely used access networks: cellular and IEEE 802.11. For this study, we measure the handover latency in different scenarios and we evaluate the differences and similarities in the management and configuration of the networking parameters in each device.

- We evaluate how the handover performance affects the user experience by considering different applications, and whether they can survive to a change in connectivity: changes in IP address and changes in access technology.

- As a result of our experimentation, we identify the challenges and open issues that are present in current smartphones and discuss on potential improvements for the connectivity management that are feasible but not yet implemented and on the integration of connectivity management with current mobility protocols.

The rest of the paper is structured as follows: In Section 2 we present the related work available in the literature and compare it to our work. We present **some background of the implementation of the network management in the systems under study**, analyzing their main similarities and differences in Section 3. In Section 4 we present the experimental deployment, and in Section 5 we evaluate the initial attachment to the access network performed by the systems under study. In Section 6 we present a detailed analysis of the handover and how the connection manager of the three families of OSes handle the changes in connectivity, both inter- and intra-technology, making this analysis extensive to the management of this change in connectivity by some popular applications. Section 7 summarizes our main results and highlights our findings comparing all the studied systems. In Section 8 we identify open issues and improvements to tackle the shortcomings of the network management in mobile devices, in light of the data extracted from our experiments. We also discuss on the integration with mobility protocols. Finally, Section 9 concludes the article and presents guidelines for our future work.

2. Related work

A significant part of the previous works in the literature analyzes the energy consumption of smartphones, however, we focus on the connectivity management. We have compiled in Table 1 the previous works that address network performance in smartphones for a better comparison. Network connectivity has been addressed, but mostly in terms of application usage and traffic patterns. For instance, [4] conducted a thorough study of application popularity and usage, characterizing the patterns followed by different demographic groups of users and the traffic generated. Their study confirms the high diversity in smartphone usage, leading to the conclusion that the tools in use may provide acceptable performance in average, but it could be considerably enhanced by some specific knowledge on applications performance and usage. Similarly, [5] also uses a logging application installed in the smartphone of a group of users and presents a personalized optimization for Android smartphones, based on application usage patterns per user, showing that the default task manager can be enhanced to improve user experience. We argue that a similar approach to these two can also be extended to network connectivity management.

The use of mobile devices equipped with several network interfaces motivates the performance study in [6], which characterizes consistency and compares the WiFi and the cellular accesses worldwide in terms of download/upload speeds and latency. The first promising application of this

[1] Product names, logos, brands and other trademarks are registered and remain property of their respective holders (Google Inc., Apple Inc. and Microsoft Corporation). Their use throughout this paper aims only at describing the results and the work performed and in no way it indicates any relationship with the holders of such trademarks.

Table 1: Comparison with related work on smartphone networking and Connection Manager

Ref.	System	Scope	Main features studied	Contributions	Main conclusions
[4]	Android Windows Mobile	Logging application	User interactions, application use, network traffic, energy drain	Evaluate the diversity range across users and time and their impact on network and energy	Patterns on app usage and user interaction time; apply diversity in usage to predict energy drain
[5]	Android	Logging application	Application popularity and usage patterns	Application usage models, app-launching optimization, personalized optimization framework for task manager	User experience can be improved by knowing usage patterns and context-aware resource management
[6]	iOS Android	Speedtest application	Cellular and WiFi network performance	Temporal and geographical analysis, aggregate performance	Similar throughput performance but, iOS higher latency
[7]	Android	Logging application	Cell, WiFi usage, phone usage	Analysis on application, phone, WiFi and cellular traffic usage	On average, WiFi traffic is 30% of the total data consumption
[8]	Android	Logging application	Traffic volume in 3G and WiFi networks	Aggregated and per-user analysis	Total traffic via WiFi much larger than via 3G, most by a small number of users
[9]	Smartphones Laptops	Wireless traffic captured by a gateway router in campus network	Network traffic, TCP impact, comparison to laptops, app layer parameters	Study network performance and traffic, mainly focusing on TCP-related parameters	Akamai and Google servers are heavily used. Different receive window advertised by iOS and Android, but similar performance
[10]	iOS Android Windows Mobile	Customized application: *3GTest*	TCP performance, RTT, DNS lookup time as metrics for app and network performance	Measurement tool and methodology for app performance comparison, 3G network performance for various operators	Smartphones are often the web browsing performance bottleneck, rather than the network
[11] [12]	Windows PC	Customized connectivity management	Vertical handover, session continuity and handover decision triggers	Connection manager to detect changes between WLAN and WWAN, virtual connectivity manager	Reduce number of handovers by keeping active connection instead of switching to WWAN when RSSI is below a threshold
[13]	Android	Application for connectivity management	WiFi offloading, simultaneous usage of cellular and WiFi interfaces	App for WiFi offloading and content aggregation (*Enhanced Android Connection Manager*)	Different use cases for WiFi offloading: content aggregation, SIM authentication or flow optimization and segregation
[14]	Android iOS	Server logs, sniffed wireless traffic and media player source code	Video streaming performance comparison.	Thorough comparison of iOS and Android clients behavior on streaming	Media players follow different content request and buffer management approaches. Redundant traffic downloaded by iOS

diversity in network connectivity is WiFi offloading. However, [7] shows that in spite of the dense WiFi deployment, cellular data consumption is still dominating and analyzes the reasons behind that fact. A similar study is conducted in [8], but in this case, the authors conclude that the percentage of offloaded traffic is not negligible, being mostly exchanged at home APs. The differences between these two studies may lie on the geographical differences in their datasets. Yet again, it is proven that a unique solution for connectivity management cannot perform optimally, advocating for a kind of customizable solution per user or based on usage or mobility patterns. Chen et al. in [9] evaluate network performance of handheld devices by monitoring the traffic captured at a university campus. They also confirm the predominant presence of TCP and HTTP flows in the traces analyzed and focus their analysis on parameters such as slow start, the advertised receive window and characteristics related to the TCP flows. However, our analysis is centered on the network connectivity management and the performance in case of a handover. While the study in [9] is restricted to a WiFi connection, Huang et al. [10] evaluate network and application performance over 2G and 3G cellular accesses. They measure UDP and TCP throughput before examining the performance of two widely used applications, web browsing and video streaming, by comparing them with different combinations of smartphone and network operator.

Devices equipped with multiple interfaces typically rely on the entity of the connection manager mostly for the interface selection, and then provide networking information to the applications. In the literature we can find alternative designs for the connection manager. Zhang et al. [11] study mobility management between WLAN and WWAN and propose an architecture that relies on a connection manager and a virtual connectivity manager, which integrates end-to-end information to be used to optimize the handover. Their connection manager includes RSSI monitoring and network availability detection modules. This architecture is experimentally tested in [12], achieving promising results such as 2.1 seconds interruption from WLAN to cellular and seamless handover from cellular to WLAN, being able to select the best AP to associate with (in terms of available bandwidth).

To the best of our knowledge, only [13] tackles the shortcomings of the current Android Connection Manager by developing an application that enhances and extends its functionality. Their application plays with the possibilities offered by the usage of multiple interfaces and – by enabling simultaneous usage of cellular and WLAN interfaces – adds support for WiFi offloading, flow segregation and content aggregation. However, they do not provide an assessment on the Connection Manager itself neither analyze the overall behavior under different network scenarios and various conditions. Besides, [14] compares iOS and Android behavior in streaming, finding out that Android and iOS media players request data from the server differently and they also have different buffer management policies. As of the time of writing we are the first ones to provide a thorough analysis of the network management supported by an experimental evaluation of the inter- and intra-technology handover under a wide variety of configurations. In addition, we examine the network attachment executed by each device and we analyze the behavior of several applications in case of a handover, and how they can handle the change in the global connectivity of the terminal – most of the times unsuccessfully. The performance of an application does not only depend on the ability of the developer to handle network connectivity, but also the accessibility and flexibility offered by the operating system and the exposed API. Above all, we compare the three most popular families of operating systems worldwide[2] and state the differences and similarities among the connection manager of Android, iPhone and Windows Phone 8 devices, establishing the guidelines for further improvements in this unexplored feature.

3. Mobile terminal networking stack

Even though the developer community for the three families of OSes is considerably large, there is no official documentation on the networking stack and the network management of the system. The effort of the community is focused on the application layer, thereby the main interest of a developer is centered at checking whether Internet connection is available, rather than making an efficient usage of the networking resources and optimizing performance. Still, we identify the five most representative elements that define the network management in Android, iOS and Windows Phone 8, and we introduce them comparatively in this section.

3.1. Android

Android is an open source software stack released by Google, and publicly available under Apache or GNU General Public Licenses. It is based on Linux kernel 3.X (kernel 2.6 in versions up to Android 4.0). On top of the Linux kernel, you can find libraries [3], and the Android runtime. The Android runtime consists on a Dalvik virtual machine –where the applications run– and on core libraries, specific for Android devices and used by the applications. In Android Lollipop, the new Android Runtime (ART) replaces Dalvik by default. An application framework interacts between the lower layers and the applications, which are on top of the system architecture.

1. **Default interface**: the Android API provides tools for an application to configure its preference on a default interface. It is important to isolate this behavior from the terminal's own networking preferences. The default interface in Android is the cellular one, and simultaneous active connections over the cellular and the WLAN interfaces are not supported in versions previous to Lollipop. However, it is possible to modify the default Android behavior to use both interfaces at the same time [13]. In Android Lollipop, each interface has its own routing table and the cellular connection is kept for 30 seconds after switching to WiFi. The ongoing communications started over the cellular interface will remain there. In addition, the terminal will not connect to a WiFi AP that has no Internet connection.

2. **WLAN interface**: the hardware abstraction layer (HAL) contains the software modules that talk directly to the kernel wireless stack and drivers. The HAL is a user-space layer developed in C/C++ used by the application framework to interact with the *wpa_supplicant* module[4], which runs in the background and controls the wireless configuration.

3. **Network related events**: Android offers, by means of the Android Debug Bridge (ADB), the possibility of tracking system logs and monitor different system events. ADB is a command line tool to communicate from a computer to emulated or physical Android devices. For example, we can monitor the active network connection (WIFI or cellular) and its coarse-grained status (*disconnected*, *connecting*, *connected*, *disconnecting*, *suspended*, *unknown*). In the case of the WiFi connection, it is also possible to monitor the state of the *wpa_supplicant* module, which also reports changes on its status (*associated*, *associating*, *authenticating*, *completed*, *disconnected*, *dormant*, *four_way_handshake*, *group_handshake*, *inactive*, *interface_disabled*, *invalid*, *scanning*, *uninitialized*). This information is also made available to application developers by the API framework.

4. **Application Programming Interface (API)**: the Android API framework is published in API levels[5]. Each API level is an integer that identifies the API revision (current API level 21). Applications must support the API level for the specific Android version, and they usually are backward compatible. Network access is handled by the Android core libraries. As we have previously stated, the API provides applications with information of the network connection status and the HAL and kernel modules access to the network interfaces and are in charge of the (re)configuration. However, the network management still keeps a simplistic or conservative approach, as it is very limited in terms of optimization or using the two access network interfaces at the same time. Interestingly enough, the API provides constants and methods to check whether the signal strength has changed, to know when a scanning has been performed and information about surrounding APs is available, compute the difference in

[2] `http://www.businessinsider.com/android-is-utterly-dominant-in-europe-2013-7`

[3] Commonly found also in Linux systems. Note that not all the Linux libraries are fully supported by Android and some changes have also been introduced to the architecture, such as *wake locks* and power management.

[4] `http://w1.fi/hostap.git`

[5] Not necessarily a new Android version has to support a new API level, but commonly a new version upgrades the API too.

signal strength or change the state or configuration of the WiFi connection, so that applications could make a much wiser usage of the network connectivity[6].

5. **Flexibility and restrictions to the user**: Android, being Linux-based, offers a high degree of flexibility in configuration and networking support. In addition, the lax licensing has favored the distribution of customized versions of the firmware[7]. By default, Android users are not given root access, but root access can be configured, empowering the accessibility to all the terminal features. The root access and the Linux characteristics make the Android OS the most flexible and accessible of the systems under study. In this way, we have full access to the system logs record, we can monitor the status of the network interfaces and we can capture traffic with a network analyzer such as *tcpdump*[8].

3.2. iOS

iOSn is the operating system running on Apple mobile devices, being n the version number, currently the latest version is iOS8, released in September 2014. iOS is based on the open source OS *Darwin*, however, iOS remains as closed-source. It is built upon 4 abstraction layers, ordered from top to bottom: *i)* Cocoa Touch layer, *ii)* Media layer, *iii)* Core Services layer and *iv)* Core OS layer. The Core OS layer has access to the kernel, drivers and networking features as the interface to BSD sockets. However, developers are recommended to implement applications by using the framework in the highest level as possible, as the complexity of handling networking events or configuration is hidden by the OS.

1. **Default interface**: the cellular interface is assumed to provide always-on connectivity, but as soon as a wireless AP appears in range, the WiFi connection takes over the cellular connection.

2. **WLAN interface**: the WLAN interface is selected as the primary interface, over the cellular connection.

3. **Network related events**: the development platform does not offer access to the lower layers that talk directly to the hardware. However, the application can react to network connectivity changes, for instance, by means of the *SCNetworkReachability* API (available in the System Configuration framework in Core Services API) to diagnose the cause of the failure of a connection and determine availability of different connections.

4. **Application Programming Interface (API)**: iOS provides three different networking API layers: Foundation layer, Core Foundation layer (these two are specific to iOS) and POSIX layer (as in other UNIX systems). The three of them support common networking tasks, being recommended to use the highest level API that fulfills the developer's requirements. The networking API provided by iOS7 can be categorized into three main groups: BSD sockets, web access and Bonjour. The interested reader is referred to [15] for further details.

5. **Flexibility and restrictions to the user**: the iOS system is closed source thereby the user is not given much flexibility of configuration with respect to network preferences. Even though the operating system has been hacked and it can be *jailbroken* it does not change the networking behavior or configuration, but increases flexibility application-wise.

3.3. Windows Phone 8

Windows Phone 8 (WP8) is developed by Microsoft, and, as the OS version for PC, they changed completely what was established in previous releases. Therefore, due to a change in the architecture, applications designed for WP8 cannot run in previous OS versions or devices running an older version cannot upgrade to WP8. As part of the features that Microsoft tried to improve in this new version, they include the network stack, focusing on speeding up the connection and reducing power consumption.

1. **Default interface**: one of the main changes in the networking stack in Windows 8 is the prioritization of network connections. Despite having a priority, multiple interfaces can be connected simultaneously. By default, the cellular interface is the one that is actively connected, but in the moment that an already visited WiFi network becomes available, the phone will try to connect to it. However, the connection through one interface does not kill a previous connection in another interface. It is only after a short period of time that the cellular connection will be terminated, unless it is being used by an application. In addition to that, existing connections at the moment of the new attachment, are kept alive and only the new connections will use the new interface. WP8 even offers the applications the possibility to select the network interface to use.

2. **WLAN interface**: the wireless network stack builds on top of the hardware device (and its firmware) with the driver and the Wi-Fi service of the OS. With this new generation of their OS, Microsoft targets to optimize power consumption and connection delay, which worsens user experience. In order to do so, they have tried to integrate as much operations as possible into the hardware layer.

[6] The interested reader is referred to `http://developer.android.com/reference/packages.html` for a complete guide of the Android API framework.

[7] One of the most popular ones is CyanogenMod, which reports more than 12 million installs (`http://www.cyanogenmod.org/`).

[8] `http://www.tcpdump.org/`

3. **Network related events**: as mentioned, the access to the configuration of the WP8 device is more restricted to the user, providing no information on the current connections or the networking events, other than enabling or disabling a network interface and changing the priority of a WiFi network in the list of preferred networks.

4. **Application Programming Interface (API)**: WP8 exposes a set of APIs that comprises the ones for previous releases (WP7) and the ones that the manufacturer recommends for applications built from scratch for WP8. We are only going to mention here the ones recommended for this system, which are *.NET HttpClient* and *WinRT Sockets*, and the so-called "Native code" (*IXML HttpRequest2* and *WinSock*). Although there are several development frameworks, we can differentiate two parts: the one devoted to the web browsing (HTTP-related) and the one directly related to the network connections (sockets). Despite the restrained flexibility, WP8 offers its applications information about the network connectivity and when a change in connectivity occurs. An application can even set preferences on the use of one interface over the other. Moreover, the emulator provided by the development framework can simulate changes in the network connectivity to test the application under different scenarios.

5. **Flexibility and restrictions to the user**: WP8 tries to optimize mobile user experience and is designed considering the performance of current devices (touch-screen, portable devices, wireless connectivity). However, this optimization, achieved by implementing a more integrated system, speeds up some processes but no involves more freedom to the user in configuration nor accessibility. The manufacturer's claim to favor this design is that the user just wants to be connected, but does not care about how they get connected.

4. Experimental setup

This section describes the characteristics of the mobile terminals under test and provides an overview of the different experiments. We aim at studying the connection manager in several mobile devices running the most representative OSes – iOS, Windows Phone and Android. The characteristics of the smartphones in our experiments are collected in Table 2. We have included in our analysis the latest versions available of the OSes, including iOS8 and Android Lollipop. We have tested the same operating system version running in different terminals, so we can avoid dependency on the terminal rather than on the OS. For the sake of fairness we have avoided performing any modification to the terminals.

We deploy two IEEE 802.11 Access Points that provide Internet access. These Access Points are under our complete control to keep track of the behavior of the terminals attached to them and to be able to modify network parameters, such as the ESSID (Extended Service Set Identifier), the wireless channel in which the Access Point (AP) operates and the IP subnet managed by the access router.

We intend to characterize the Connection Manager on the systems under study, identifying the main strengths and weaknesses of network connectivity management and comparing their behavior. Our analysis focuses on understanding the following mechanisms:

1. **Initial attachment procedure to an 802.11 network**. Regarding this mechanism, we aim at understanding: *i)* how the attachment to a WLAN is carried out by every device, analyzing the differences among them, if any, and, *ii)* how the network selection algorithm works and what criteria are used to choose among the different candidate networks. This is explained in Section 5.1.

2. **Initial configuration of the protocol stack**. Once the mobile terminal has attached to a point of access, we aim at understanding the main steps and the protocols used to complete its networking stack configuration. Note that, if this procedure takes place entirely whenever there is a change in the point of attachment to the network, and not only as an initial configuration, it may enable potential optimizations for the handover process. This operation is explained in Section 5.2.

3. **Horizontal handover**. We examine the handover procedure between two IEEE 802.11 APs. We play with different network parameters to have a wide view of the performance for the different mobile terminals. Specifically, the current AP and the target one may have the same or different ESSID, operate in the same or different channels and manage the same or different IP subnets, in which case the handover would imply also a layer three reconfiguration. Through this analysis, we aim at knowing whether there are any dominant factors when the mobile device changes its point of attachment to the network and to what extent the different changes impact the configuration and connectivity management. The horizontal handover is explained in detail in Section 6.1.

4. **Vertical handover**. We evaluate the handover procedure when it involves a change in the access technology. We aim at understanding how the mobile devices handle the inter-technology handover, whether they can keep both technologies operative simultaneously and whether they handle the survival of ongoing connections. Characterizing the inter-technology handover is essential to design potential optimizations and flow mobility solutions. However, due to restrictions by the terminal and the network operator, we

Table 2: Main characteristics of the analyzed smartphones

	LG Nexus 4 E960	LG Nexus 5	iPhone 3GS	iPhone 4	iPhone 5	HTC 8S
OS Version	Android 4.2.2, 5.0	Android 5.0	iOS 6.0	iOS 7.0.4	iOS 8.1.2	Windows Phone 8.0
Chipset	Qualcomm Snapdragon S4 Pro APQ8064	Qualcomm MSM8974 Snapdragon 800	Samsung APL0298C05	Apple A4	Apple A6	Qualcomm Snapdragon S4 Plus MSM8227
CPU	Quad-core 1.5 GHz Krait	Quad-core 2.3GHz Krait 400	600 MHz Cortex-A8	1 GHz Cortex-A8	Dual-core 1.3GHz Swift	Dual-core 1GHz Krait
RAM	2GB	2GB	256MB	512 MB	1GB	512MB
WLAN	Atheros WCN3660 Murata SS2908001	Broadcom BCM4339 (5G WiFi combo)	Broadcom BCM4325	Broadcom BCM4329	Murata 339S0171 Broadcom BCM4334	Atheros WCN3660

were not able to obtain direct measurements from the cellular interface. The vertical handover results are presented in Section 6.2.

5. **Application behavior**. We study application survival to handover in the cases already explained. The objective is to know the perception of the user when an application is running and there is a change in connectivity. In addition, we get to know whether applications can handle an interruption due to horizontal or vertical handover seamlessly. These experiments are presented in Section 6.3.

We start our analysis in Section 5, with the evaluation of the initial attachment procedure to an IEEE 802.11 network.

5. IEEE 802.11 Initial attachment procedure

In order to test the default attachment to an IEEE 802.11 [3] WLAN, we deploy a wireless access point and run thirty experiments for each smartphone. In each experiment, which lasts for 60 seconds, we monitor the traffic by means of a network analyzer[9]. The monitor interface is located close to the access point, so we can likely capture all the frames involved in every exchange. Initially, the WLAN interface of the mobile terminal is down, so we do not miss any packet or reach misleading results because of having the device already connected to another network. We start our experiment by bringing up the interface and checking that the device actually connects to the AP under our control. This experiment describes the case in which the terminal finds an already known network and connects successfully. Note that to connect to a network for the very first time the user must identify and select manually the network to connect to.

Figure 1 presents in a diagram the different steps performed during the initial attachment to the IEEE 802.11 network. Note that this is a general diagram and it does not try to highlight the differences among the terminals, but to illustrate the common procedures followed by all of them. In the following, each of the steps in the initial attachment is explained in detail, highlighting the differences among terminals.

5.1. IEEE 802.11 attachment

Active scanning to broadcast address: when the WLAN interface of a mobile terminal goes up, it detects all the surrounding wireless networks available by initiating an active scanning procedure. The terminal sends Probe Request frames to a wildcard ESSID (and to every previously visited ESSID) in every channel sequentially. Neighboring access points will receive these Probe Requests and answer with Probe Response messages, indicating their capabilities and providing synchronization information. This active scanning phase differs slightly in the systems under study.

By monitoring the traffic in different channels we are able to measure the delay induced by the scanning procedure. The results show high variability in the amount of Probe Requests being broadcast as well as in the time interval between them, but we can identify different patterns:[10]

- The Android terminal scans approximately every 10 seconds in every channel, sending a number of consecutive Probe Requests inter-spaced approximately 15 ms.
- The Windows terminal spends approximately 6 seconds between consecutive scans in the same channel. The time lapse between consecutive Probe Requests is very variable, but shows two different dominant values: the terminal sends several requests every 6 ms and one after 60 ms to continue with 6 ms interval again.
- The iPhone terminal presents an interval of approximately 9 seconds between the scan in every channel and the delay between consecutive Probe Requests is around 20 ms. As the WP8 terminal, there are several requests spaced 20 ms (much higher interval than WP8) another one 700 ms after that, to continue sending every 20 ms again.

It is also interesting to evaluate the behavior of the terminals in channel 14, which is not allowed in Europe, where we are based. The Android and Windows phones do not list the networks operating in that frequency as available, but the iPhone does, regardless of the regulatory

[9] http://www.wireshark.org/

[10] Due to the high variability we just provide rough numbers.

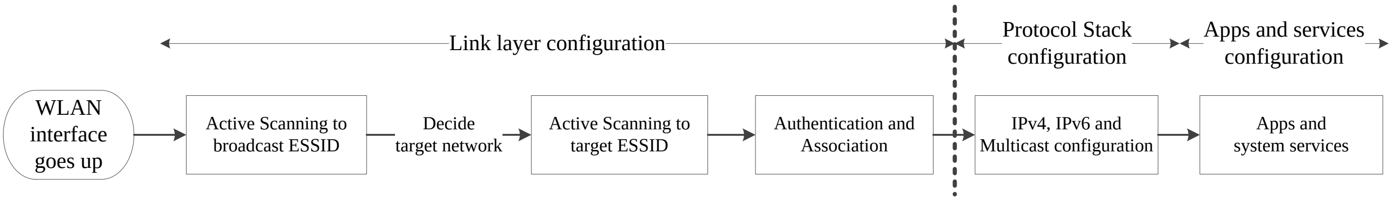
Link layer configuration
Protocol Stack configuration
Apps and services configuration
WLAN interface goes up
Active Scanning to broadcast ESSID
Decide target network
Active Scanning to target ESSID
Authentication and Association
IPv4, IPv6 and Multicast configuration
Apps and system services

Figure 1: Initial attachment to the WLAN.

domain. However, the three terminals send Probe Request frames in every channel including that out of the allowed frequency band (at a different interval than in the other channels, though). Noteworthy, Android and Windows devices do not give the user the opportunity to attach to these networks although they actively scan that channel.

The scanning policy followed by a terminal has several potential effects. First, the number of Probe messages sent during the scanning phase impacts directly the energy consumption. Second, a terminal can only obtain information regarding the signal level from the AP by the frames received, hence receiving more responses before deciding the target point of attachment can be beneficial. Last, the delay in the attachment to a WLAN AP is directly influenced by the time spent in the active scanning, and so is the handover delay if the terminal has to scan again before connecting to a new AP.

Target Network Decision: this step corresponds to the actual decision on the AP to connect to. It seems reasonable to expect the terminals to perform some kind of complex algorithm considering, for instance, the signal level received from the different APs. After the analysis, we have discovered through our extensive tests that all the terminals use a simple rule to decide where to connect to. If the AP used in the immediately previous connection is available, the terminals will connect to it, no matter its signal level. In case the last visited AP is not available, the terminal connects to the one previous to the last one, and so on. In the case secured and open networks are available, only the iPhone terminal shows a preference for secured networks if the immediately previous connection is not possible. **Note that we have confirmed this behaviour also with several wireless networks that are available at our laboratory, not only the two APs we deploy in our set of experiments.** It is worth to note that the Android phone includes an option in the WiFi settings to connect to a different WLAN or to the cellular network if the signal is weak. However, even if this option is enabled, the terminal follows the same approach and disregards signal strength information to connect to an AP. This way of choosing the target network leads to weak connections and poor performance.

Active scanning to selected ESSID: in this step, the terminal addresses a Probe Request message directly to the AP selected previously and indicates the target ESSID in the correspondent field in the frame.

Authentication and Association: these are the last two steps before being attached to a WLAN AP. Both of them are standard procedures and are equally executed in the different terminals. Security procedures are out of scope as the main focus in this work is on the Connection Manager.

Table 3 gathers the average and standard deviation of the delay during the initial attachment to a WLAN measured for the different OSes versions. We distinguish between link layer and network layer configuration, which in turn is measured for IPv4 and IPv6 configuration, in order to provide more information on the influence of both processes. The link layer delay is measured as the time from the interface going up until the terminal is associated to the AP. The scanning policies presented previously influence considerably these delays. The Android terminal clearly outperforms the rest of the systems. Although it sends a higher number of frames before attaching to the selected AP, these frames are more frequent, whereas the other systems have a longer interval and therefore, the association is delayed. However, Android Lollipop has clearly impaired performance in the connection for Nexus 5 devices (well-known issues are being reported by Nexus 5 users since updated to Lollipop). The WP8 delay doubles that of Android 4.2 and Nexus 4 Lollipop, although the scanning time in every channel is lower for Windows. There is no clear difference in performance between iOS6 and iOS7, so this process seems not to have suffered major modifications from one version to the other. Still, the latest version, iOS8 increases the delay, as it happened with the Android update. We have had access to a new iPhone 6 and been able to perform the same tests, finding no difference, on average, in our measurements with respect to the ones in Table 3, which correspond to an iPhone 5. It calls our attention the usage of *CTS-to-self* frames sent by Android Lollipop in Nexus 5 and iOS8 in iPhone 6, but not in previous terminals running the same operating system. Moreover, Nexus 5 just sends one frame right before the authentication frame. The process and the delay for IP configuration is explained in detail in the next subsection.

5.2. Protocol stack initial configuration

IPv4, IPv6 and Multicast configuration: we group these configuration steps because they are very similar for the evaluated systems. Table 3 shows the delay during the initial attachment due to the configuration of an IP address in the wireless interface of the mobile terminal, once it is associated to the AP (under the column "Network

Table 3: Initial attachment delay

OS Version	Link layer delay (s)	Network layer delay (s)		Total (s)	
		IPv4	IPv6	IPv4	IPv6
Android 4.2 Nexus 4	0.42 ±0.12	1.09 ±0.31	4.75 ±0.43	1.51 ±0.33	5.17 ±0.45
Android 5.0 Nexus 4	0.47 ±0.05	0.68 ±0.36	9.58 ±3.53	1.15 ±0.33	10.05 ±3.54
Android 5.0 Nexus 5	2.58 ±0.05	0.50 ±0.04	3.33 ±1.15	3.09 ±0.06	5.90 ±1.14
iOS6	1.91 ±0.08	0.13 ±0.01	1.16 ±0.06	2.05 ±0.07	3.08 ±0.14
iOS7	1.98 ±0.26	0.13 ±0.02	1.12 ±0.58	2.12 ±0.26	3.1 ±0.67
iOS8	2.28 ±0.47	0.25 ±0.11	0.87 ±0.25	2.54 ±0.45	3.25 ±0.51
WP8	1.08 ±0.28	1.1 ±0.12	1.28 ±0.15	2.18 ±0.30	2.36 ±0.32

layer delay"). In the case of IPv4, the different terminals follow the same mechanism and use DHCP (Dynamic Host Control Protocol) [16] to configure the IP address when they attach to the network. In our experiments, the DHCP server is located in a node belonging to the same network but different from the AP. The iPhone terminals are the fastest ones in this case because they start the process much earlier, while Android 4.2 (ICS) and WP8 present a very similar delay, close to one second over the one from iOS. Contrarily to the attachment to the WiFi AP, the IPv4 address configuration has been made faster in the new Android version. Another interesting difference is that the new iOS version, iOS8, uses gratuitous ARP in the IPv4 configuration when the wireless interface goes up, for both iPhone 5 and iPhone 6. Neither Android nor WP8 do that.

Regarding IPv6 configuration, the WP8 device tries to configure an IPv6 address by means of DHCPv6 [17], but as we have no DHCPv6 server in our network, all the terminals configure local and global IPv6 addresses following the SLAAC (Stateless Address Auto Configuration) procedure as specified by [18] and [19] for configuration and DAD (Duplicate Address Detection). First, the mobile node acquires a link local IPv6 address and joins the all-nodes and the solicited-node multicast addresses when the connection is established in the interface. Then, in order to perform the DAD the terminal sends a Neighbor Solicitation message to the solicited-node multicast address. As the source address of this message is the unspecified address, any other node will not respond to that message and will identify the tentative target address and know that address cannot be used. Whether the node itself receives its own Neighbor Solicitation depends on the particular implementation of the multicast loopback. In the case of iOS devices, the delivery to upper layers of their own multicast message is disabled, so, if no other node in the network has the same IPv6 address as the target one, no Neighbor Advertisement from any other node will be received and the mobile terminal will silently configure the interface with the target IPv6 address. However, the Android and Windows terminals send out a solicited Neighbor Advertisement with their own source address upon receiving their own Neighbor Solicitation messages to announce this configuration. The Android device unicasts the advertisement to the router, while the Windows device broadcasts it to all the nodes.

In light of the results in Table 3, the delay for the IPv6 configuration is comparable in the iOS and Windows Phone systems, but the Android device takes significantly more time, which delays any IPv6 connection. It is worth to mention that the three families of mobile OSes follow [20] considerations to protect privacy. According to SLAAC rules, the IPv6 address is configured from an interface identifier (EUI-64 identifier), and the second half of the global IPv6 address (without considering the 8-byte prefix announced by the router for configuration) is the same regardless of the location, so the device could be tracked. Therefore, IPv6 privacy extensions are defined so the network interfaces are configured with randomized strings, which change over time, instead of the interface identifier in order to complicate the activity correlation. RFC 7217 [21] provides the specification for the generation of these random interface identifiers while keeping IPv6 addresses stable in each visited subnet. Typically, the address derived from the EUI-64 identifier is kept, in addition to a temporary address built upon randomized identifiers. In our tests, we have observed that only the Android device configures an IPv6 address matching its EUI-64 identifier and a randomized one. The WP8 and the iOS devices configure the two IPv6 addresses from random strings. According to the RFC, "devices implementing this specification MUST provide a way for the end user to explicitly enable or disable the use of temporary addresses"; however, none of the systems are compliant with this statement. Table 3 shows a considerably higher delay for network layer configuration for the Android (Nexus 4) terminal. It starts the IP address configuration process approximately 1 second after the association, starting with the Router Solicitation message, and approximately 4 seconds after the association, it starts with the SLAAC, which leads to the highest delay among the terminals studied. Unfortunately, this process has been impaired significantly in the updated version.

Finally, the use of multicast for the interface configuration is slightly different in the three families of systems. For instance, Windows Phone makes use of LLMNR (Link Local Multicast Name Resolution) protocol [22] in addition to IGMP (Internet Group Management Protocol) [23] and MLDv2 (Multicast Listener Discovery) [24, 25] which are used in the iPhone signaling. The Android phones just make use of MLDv2 messages, as IGMP is not supported. This behavior is specific of some Android devices and constitutes a known issue in the community.[11]

Higher layer configuration: **As part of the process to gain Internet connectivity, all mobile OSes perform a technique called Network Connectivity Indicator, to detect captive portals. This protocol issues a DNS query to establish a TCP connection intended to send an HTTP GET method and retrieve a light-weighted web page, which is mainly void. This procedure serves to check the Internet connection at the device and prompt the users with a login form to introduce their credentials, if required by the WLAN administrator. The API in Android includes a way to access the information on whether the terminal is connected or connecting to the Internet, through which interface and allows registering to event notifications in case of network status changes. The connection manager can detect the status of the interface and if it is connected, and in addition, application developers have the option to try and reach a website from their application when it starts running to check that there is actual connectivity.** With regard to Apple's devices, it is worth highlighting a recent change performed in iOS7 compared to iOS6. The captive portal detection in iOS6 was performed issuing a query to the web page `http://www.apple.com/library/test/success.html`, while in iOS7 this web page check has been swapped by a randomized query to a URL selected in a list identified by Apple. Table 4 collects the service connections that are preceded by a DNS query when the different terminals attach to a WLAN. As soon as the mobile terminals gain Internet connectivity, all the terminals try to reach the central servers of their correspondent manufacturers to update their location, configure some services - time synchronization or push notification service – and re-initiate some connections – as GTalk, in the case of Android. IP addresses on the server side are expected to change, so the terminal connects by hostname, issuing DNS queries, rather than by IP address. Commonly, servers implement a load balancing scheme, so it is possible that the same query returns a different IP address for the same host name. For that reason, to identify the connections we track an IP address block, instead of a specific name or address. In addition we observe that many providers and applications use CDN (Content Distribution Network) nodes to offer their services, which complicates the identification of the service as the connections are hidden by the CDN. For instance, iPhone uses the Akamai network [26] for all Apple related services.

To sum up, the results show that the Android terminal associates to the AP much faster in the previous versions of the system, and the network layer configuration time is comparable to that of WP8 for IPv4, but it takes significantly longer than in the other terminals for IPv6 configuration and Duplicate Address Detection (DAD). However, it calls our attention that, as Linux does, the first DNS query sent by the Android phone is always a request for an IPv6 address (type AAAA), so IPv6 takes precedence over IPv4. On the contrary, the WP8 device issues the IPv4 query before the IPv6 one. Note that the differences in hardware, as reported in Table 2, should not be responsible for the differences in performance, specially in the case of the Android phone, which is the one with the slowest initial connection. In the case of the iOS terminals, they require much less time than the Windows and Android devices to configure an IPv4 address, but the association to the wireless AP takes longer (except for Nexus 5). The fast configuration in IPv4 makes possible to have comparable total times for iOS and Windows devices, while for IPv6 the Windows terminal is almost 1 second faster than iOS and almost 3 seconds faster than Android.

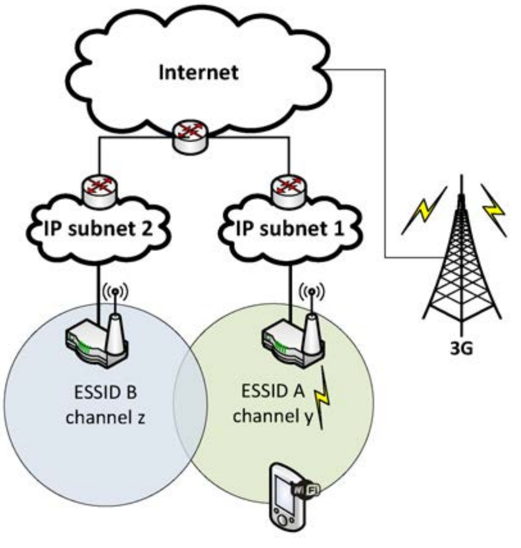

Figure 2: General scenario for handover tests.

6. Handover dissection

This section presents a thorough study of different handover scenarios and their effect on applications and user experience. For these experiments, we vary the configuration of two 802.11 APs in order to cover as many different scenarios as possible. The different variations in the setup are presented in Figure 2. The access points provide Internet access to the terminals attached to them and we play with different parameters of the network – ESSID, channel and IP subnet – to evaluate their influence when the handover process takes place from one AP to the other.

We do not only analyze the handover from an 802.11 AP to another, but also the inter-technology, or vertical

[11] Issue 51195: many devices have multicast disabled in the kernel. `http://code.google.com/p/android/issues/detail?id=51195`

Table 4: DNS queried for **initial configuration** of services on **WLAN interface start-up**

Service	Android	iOS	WP8
Initial connection to central servers	clients3.google.com (IPv4 and IPv6)	www.apple.com; (IPv4 and IPv6)	login.live.com clientconfig.passport.net
Network status indication	clients3.google.com/generate_204 www.google.com/blank.html	http://www.apple.com/library/test/success.html (iOS6) randomized (iOS7)	www.msftncsi.com
push notification service	mtalk.google.com	18-courier.push.apple.com (IPv4 and IPv6)	push.live.net
Software updates	play.googleapis.com android.clients.google.com (IPv4 and IPv6)	www.apple.com; (IPv4 and IPv6)	ctldl.windowsupdate.com crl.microsoft.com (IPv4 and IPv6)

handover, moving the connection from the cellular interface to the WLAN one and vice versa. Lastly, we evaluate the behavior of several applications when a handover takes place to check handover impact from the point of view of the user.

6.1. WLAN Horizontal Handover

6.1.1. Initial considerations

In this section we focus on the main core of the handover analysis, which is the intra-technology or horizontal handover, where the mobile node changes the point of attachment to the WLAN and connects to a different WLAN AP. Table 5 gathers the link layer delay measured for the different experiments. We have indicated the differences in ESSID and channel of operation between the two APs. Our analysis is centered on the link layer handover, so we do not change the IP subnet. As none of the three mobile OS families bases a handover decision on the received signal strength from the current AP or link quality degradation, we force a handover in our experiments by initiating it in the network or in the terminal side following three different approaches: *i)* the AP deauthenticates the station (since the mobile terminals do not react to signal strength or link quality degradation, we turn off the AP). This is presented in Table 5 as "disconnect AP" in light of the obtained results we differentiate two subcases; *ii)* the mobile user terminates the connection by manually omitting – "forgetting" – that network (referred to in Table 5 as "forget") or *iii)* the mobile user switches the connection directly to other network ("manual" in Table 5). The difference between the second and the third option is that, in the latter, the user explicitly indicates the target network to switch to.

Another preliminary consideration is that the cellular interface is the default interface in smartphones, since it provides always-on connectivity. Therefore, we also evaluate the influence of having this interface enabled or disabled in the case of a horizontal WLAN handover. We have noticed that the analyzed OS families always fall back to the cellular connection as soon as the current WLAN connection fails, even if there are other WiFi networks available. This change involves another variation of the IP address that provides global connectivity to the device, however, it does not necessarily worsen the handover latency and the interruption experienced by the user. On the contrary, the change to the cellular connection actually does not increase the handover delay and, as we will see in Section 6.3, improves the performance in case the handover implies a change of IP subnet, contributing to the survival of a running application, depending on the implementation. For the experiments presented in this section, we have confirmed that having the cellular data connection enabled or disabled makes no significant difference in the horizontal handover delay, so we do not distinguish these two cases in Table 5 for the sake of clarity.

6.1.2. Handover latency

The first issue that calls our attention is the considerable handover latency for the systems under study in every scenario. However, it is remarkable that this latency does not always translate into a complete loss of connectivity or killing a running application – see Section 6.3.

From the figures in Table 5 we can clearly see that the fastest way of handing over from one WiFi network to the other is to manually change the connection. All the terminals can change the connection in less than a second, but iOS devices are particularly fast, with times around 200 ms. In the case the handover is initiated by the user but just deciding to disconnect from the current AP, without directly connecting to the new one ("forgetting" the current connection) the handover latency increases to values around 1.5 seconds as the mobile terminal scans again in every channel. However, the Android terminal presents a stable delay as the handover delay remains around 0.9 s when the mobile terminal hands off between two APs from different ESSs (Extended Service Sets), regardless whether the handover is initiated by the terminal or the network. For the rest of the systems, instead, it varies significantly. It is remarkable the sticky client implementation in iOS8 wireless client, which tries to remain connected to the same AP regardless of network conditions or even user choices, especially when having to change to a different channel, which explains the 4.92 s delay in the "manual" handover and the 0.17 s for the "forget" handover, as the terminal cannot try to remain connected to the previous AP.

In the case of roaming between two APs within the same ESS, the only possibility is to hand off by disconnecting the AP, because the user cannot choose manually to which AP

Table 5: Layer 2 handover delay [s] for the different terminals

			Handover mechanism			
			Manual	**Forget**	**disconnect AP**	
					AP deauthenticates	**Connection lost**
Same ESSID	Same channel	A 4.2	N/A	N/A	0.92 ±0.13	5.54 ±0.09
		A 5.0 Nexus 4	N/A	N/A	7.20 ±2.54	21.09 ±6.39
		A 5.0 Nexus 5	N/A	N/A	9.12 ±1.72	2.96 ±1.06 (reassoc) 13.14 ±2.66
		iOS6	N/A	N/A	1.94 ±0.36	4.63 ±1.12 (reassoc)
		iOS7	N/A	N/A	2.48 ±0.29	4.59 ±0.59(reassoc)
		iOS8	N/A	N/A	2.44 ±0.81	11.23 ±3.33 (reassoc)
		WP8	N/A	N/A	0.24 ±0.1	
	Different channel	A 4.2	N/A	N/A	0.99 ±0.19	5.40 ±0.11
		Lollipop Nexus 4	N/A	N/A	3.25 ±0.26 (reassoc)	2.71 ±0.37 (reassoc)
		Lollipop Nexus 5	N/A	N/A	9.05 ±4.68	13.40 ±4.60
		iOS6	N/A	N/A	1.57 ±0.34	5.66 ±1.38 (reassoc)
		iOS7	N/A	N/A	2.38 ±0.32	4.88 ±0.97 (reassoc)
		iOS8	N/A	N/A	14.80 ±5.94	18.87 ±1.87
		WP8	N/A	N/A	0.4 ±0.097	
Different ESSID	Same channel	A 4.2	0.95 ±0.08	0.9 ±0.03	0.78 ±0.29	5.42 ±0.1
		A 5.0 Nexus 4	0.10 ±0.08	1.04 ±0.45	5.31 ±3.02	10.73 ±3.18
		A 5.0 Nexus 5	0.09 ±0.04	4.70 ±1.78	8.53 ±3.81	9.75 ±4.83
		iOS6	0.14 ±0.054	1.53 ±0.10	1.36 ±0.53	11.87 ±0.83
		iOS7	0.22 ±0.03	1.64 ±0.33	2.41 ±0.67	11.88 ±0.36
		iOS8	0.12 ±0.06	0.27 ±0.06	3.90 ±1.90	11.44 ±2.23
		WP8	0.87 ±0.34	1.52 ±0.62	10.14 ±1.83	
	Different channel	A 4.2	0.93 ±0.07	0.91 ±0.025	0.91 ±0.06	5.41 ±0.15
		Lollipop Nexus 4	0.27 ±0.28	3.29 ±0.31	9.07 ±1.65	15.09 ±3.39
		Lollipop Nexus 5	0.10 ±0.08	4.24 ±1.30	6.87 ±3.26	13.12 ±4.48
		iOS6	0.22 ±0.06	1.43 ±0.45	1.09 ±0.82	12.00 ±0.51
		iOS7	0.28 ±0.04	1.69 ±0.61	1.20 ±0.26	12.16 ±0.887
		iOS8	4.92 ±0.03	0.17 ±0.08	8.61 ±4.12	12.12 ±2.38
		WP8	0.73 ±0.05	0.75 ±0.05	10.41 ±0.065	

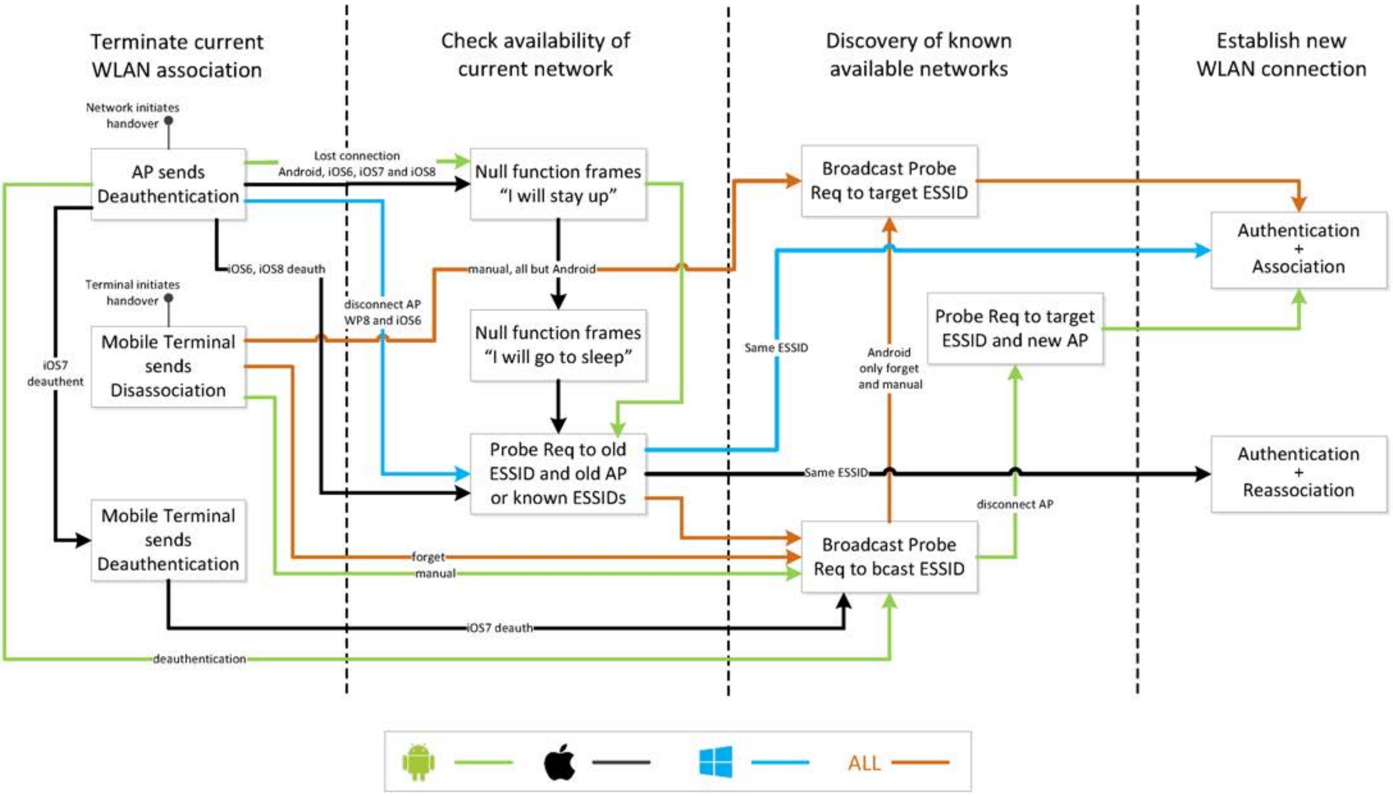
Terminate current WLAN association
Check availability of current network
Discovery of known available networks
Establish new WLAN connection
Network initiates handover
AP sends Deauthentication
Lost connection Android, iOS6, iOS7 and iOS8
Null function frames "I will stay up"
Broadcast Probe Req to target ESSID
Authentication + Association
iOS6, iOS8 deauth
manual, all but Android
Terminal initiates handover
disconnect AP WP8 and iOS6
Null function frames "I will go to sleep"
Same ESSID
Android only forget and manual
Probe Req to target ESSID and new AP
Mobile Terminal sends Disassociation
iOS7 deauthent
Probe Req to old ESSID and old AP or known ESSIDs
Same ESSID
Authentication + Reassociation
disconnect AP
Mobile Terminal sends Deauthentication
forget
manual
Broadcast Probe Req to bcast ESSID
iOS7 deauth
deauthentication
ALL

Figure 3: Flow diagram of a handover procedure for the different OS families.

in the ESS it connects. When the handover is triggered from the network side, the AP sends a deauthentication frame when it disconnects. In this case, we have identified two differentiated behaviors that repeat in our experiments, except for the WP8 terminal. In the first case, the mobile terminal receives the deauthentication from the AP and starts active scanning for a new AP to connect to. This case shows lower delays and the signalling is similar to the one described in Section 5 for the initial attachment. On the contrary, in the other case we have measured much higher delays until the mobile terminal associates to the new AP. The reason for this difference is that the mobile terminal does not get disconnected because it recognizes the deauthentication from the AP, but because the connection is lost (e.g. missed Beacons). The mobile terminal tries to reconnect to the lost AP by sending Null Function frames and Probe Requests. As the AP does not respond anymore, the mobile terminal needs to scan to look for other available APs. Once again, the Android terminal presents a similar delay irrespective of whether the former and the new AP are part of the same ESS (around 5 seconds). However, the iOS devices perform a re-association in approximately 5 seconds when roaming within the same ESS, whereas the delay raises to 12 seconds when the two APs are in different ESSs. We have traced the events that take place in the case with the higher delays in the wireless networking stack until the wireless driver. However, we cannot confirm if this is a buggy behavior of the implementation, but it clearly gives us some room for improvement in the connection management. The Windows Phone terminal does not present these differences in its behavior. Its handover process is more stable although moving within the same ESS clearly decreases the delay and reduces the scanning phase. It is worth to mention that, when roaming within the same ESS the iOS devices send a Reassociation Request frame instead of an Association Request to the new AP. The main difference between these two frames is that the Reassociation includes the BSSID (Basic Service Set Identifier) of the previous AP that the mobile terminal was connected to. This is because from iOS6 Apple has implemented support for 802.11r (amendment for Fast BSS transition). Since our APs do not implement 802.11r mechanisms, this feature reduces to a regular handover, although having it enabled, should influence significantly aspects like security or QoS.

If the handover is performed between two APs within the same ESS, this change should have an effect only at the link layer, being transparent to the IP layer. Therefore, the mobile terminal should not renew its DHCP lease until it is expired even though it changes from one AP to another inside the same ESS. This behavior is confirmed by the iPhone terminal, but both Android and Windows phones initiate a DHCP discover process when they connect to the new AP, regardless of being part of the same ESS.

In these experiments we do not consider changes in the IP layer, as we are focusing on the link layer delay. Nevertheless, it is worth pointing out that for the Android and Windows phones any kind of handover involves a reconfiguration in the IP layer (new DHCP and IPv6 configuration), even though the target network is managing the same IP subnet.

6.1.3. Link layer behavior

Figure 3 illustrates the signaling differences among the systems under study for the four handover cases we have analyzed. We indicate the steps common to all the systems with a brown line, and we differentiate the steps for each system with green for Android, blue for WP8 and black for iOS. In the following, we highlight the main differences among the three. The handover process starts either when the AP sends the deauthentication to the mobile terminal ("disconnect AP" case in Table 5) or the mobile terminal disassociates ("manual" and "forget" cases in Table 5). One of the main differences we can appreciate is that the Android terminal does not differentiate between the "manual" and "forget" handovers. Both trigger the active scanning with broadcast Probe Request frames to the wildcard ESSID; then, it follows a broadcast Probe Request to the target ESSID (when the AP sends the deauthentication to the Android terminal, this Probe Request frame is not broadcast) and finally authenticate and associate. For WP8, we notice it follows a shorter path than the other systems when the AP sends the deauthentication (and the ESSID does not change), which matches the low delay of the WP8 handover in this case. We notice that only the iOS7 device sends a Deauthentication frame when it receives the deauthentication from the AP. Finally, the two different cases for the "disconnect AP" handover are illustrated by the two different paths that part from the "AP sends Deauthentication" box. The case that incurs the highest delay is the path through the Null Function frames, while the direct path to the active scanning box also involves considerably lower delays (as abovementioned in Table 5). In this last case, the terminals may follow two different paths, sending Null Function frames or not, depending on whether they accept the deauthentication from the AP or think the connection has been lost.

The use of Null Function frames is a common practice and, as their usage is not defined by the standard, different implementations make different use of them. On the one hand, they are used for power saving, notifying the AP when the station is going to sleep mode and waking up. On the other hand, they are used during active scanning to get the AP buffering the frames addressed to the station while it is sensing other channels to avoid retransmissions. A third alternative is to use the Null Function frames as a keep alive for the connection. However, the flexibility provided by these frames is also applied for attacking wireless networks [27].

6.1.4. Upper layers behavior

It is worth mentioning that the handover delay increases significantly in the case of not having an application running. Moreover, although the behavior on the link layer is quite similar in the different devices for any of the scenarios, the behavior in the upper layers is very diverse.

We aim to characterize also the interruption at upper layers due to the handover. However, we have not been able to extract similar results to the link layer measurements, due to the variability in the behavior of the application running and the high delay to re-establish the flow with the main servers. The application behavior is not under our control, but several cases can be identified. In the case of Android, every handover case involves a complete reconfiguration also at the network layer, reissuing DHCP discovery and performing DAD. Therefore, the interruption of any traffic flow is considerably long. When the device gains connectivity again, the TCP connection is reset (RST flag set) and any connection of an application running at the moment of the handover has to be renewed, sometimes even connecting to different servers – e.g. due to load balancing at the server. On the other hand, in the case both APs handle different IP subnets, this re-issue of DHCP discoveries enables a faster re-configuration of the IP layer. Otherwise, even though the connection at link layer takes place successfully, no data will be delivered to or from the mobile terminal as it does not reconfigure its IP address according to the new network. **However, passing through the cellular interface during the transition between two WLAN APs enables the IP re-configuration, also within the same ESS. This is the case for iOS devices, which do not re-issue a DHCP discovery when there is a handover within the same ESS if the cellular interface is not enabled.**

6.2. 3G-WLAN Vertical Handover

One of the main differences among the operating systems under study is in the simultaneous usage of the cellular and 802.11 interfaces. iOS does not allow one interface to remain active when the other is getting started too, e.g. finishes the open connections and turns down the cellular interface when a known WLAN appears in range and tries to connect to it. On the other side, Android keeps the cellular connection on until the IP address is configured in the WLAN interface. Finally, Windows Phone allows the simultaneous connectivity through both interfaces, even keeps an active communication through the cellular interface although it gets attached to a WLAN AP, while applications started from that moment use the WLAN. This is an important advancement over its competitors, as for instance, it allows to keep a VoIP (Voice over IP) conversation over the cellular network while connecting to an 802.11 AP, ensuring that the call will not be interrupted. Android 5.0 includes a significant change in the network management. First of all, the simultaneous usage of cellular and WiFi interfaces is allowed. Therefore, the connections established through the cellular connection remain active through that interface even in the case the mobile terminal connects to a WLAN. If there are not active connections, the WLAN interface is still the preferred one, deactivating the IP connectivity in the cellular interface 30 seconds after the WiFi interface gets a connection. In order to introduce these changes, the development team has defined a routing table per interface, instead of a system-wide routing table, as in the previous Android versions. This has allowed to improve, as we show in 6.3 the performance of applications running when a handover occurs, for instance, in the case of an ongoing Skype call.

6.3. Application survival to handover

Following the different approaches for handover described in Section 6, we have studied the influence of these handover procedures in the behavior of several applications. The test procedure consists on starting the application under stable network conditions and perform a handover to analyze the effect of this change. The results of these experiments are presented in Tables 6 and 7 for intra-technology WLAN handover and inter-technology handover respectively. We have observed that the mobile devices under study fall back to the cellular interface as soon as the WLAN connection is lost, although other 802.11 networks are available and even attach to one of them immediately. Because of that, we have included another variant in our experiments apart from the different configurations of the WLAN, which is to have the cellular data connection in the mobile terminals enabled or disabled. However, regardless whether the cellular interface is on or off, the delay (presented in Section 6.1) is not affected but only has an influence on the survival of the application, which is more likely to overcome the handover when the cellular interface is on. We have identified different application behaviors:

a) The application interrupts and does not recover even when there is global connectivity unless you restart the application. This is indicated in the table by a black cell.

b) The application interrupts for a small lapse, continues working intermittently and finally stops. This is indicated by a dark grey cell.

c) The application interrupts for a small interval, but it continues working properly and go back to its normal operation with a noticeable but acceptable glitch for the user. This is indicated by a light grey cell.

d) The application tolerates the handover, which happens smoothly and the interruption is seamless for the user. This is indicated by a white cell.

We have chosen some of the most widely used applications **serving for different purposes, also to check if there are differences in the way they access to the network**. The applications we have evaluated are Skype, as a VoIP application (Android v3.2.0.6673, v5.1.0.57240 (Nexus4 Lollipop) and v5.1.0.58677 (Nexus 5 Lollipop); iOS6 v4.2.2601; iOS7 v4.17.0.123; iOS8 v5.11; WP8 v2.1.0.241); and three applications for radio streaming, which will be referred to in the following as Radio 1[12] (Android v1.08.16 (ICS), v4.1.2 (Nexus4 Lollipop) and v1.08.33 (Nexus 5 Lollipop); iOS6 v2.1.7216; iOS7 v2.1.9327; iOS8 v3.0.0; WP8 v2.1.0.0), Radio 2[13] (Android v1.0 and v2.1.3 (Nexus 4 and Nexus 5 Lollipop); iOS6 v2.4; iOS7 v3.0; iOS8 v3.5; WP8 v1.3) and Radio 3[14] (Android v2.2.2 and v2.2.6 (Nexus 4 and Nexus 5 Lollipop); iOS6 v2.1.1; iOS7 v2.2.1; iOS8 v2.2.2; WP8 v1.2). We choose three different radio stations to avoid biasing the conclusions, because some features may be implementation-dependent. We have also considered in our analysis Youtube, as a video streaming application (Android 4.2 v4.2.16 and v6.0.13 (Nexus 4 and Nexus 5 Lollipop); iOS6 v1.1.0; iOS7 v2.3.1.11214; iOS8 v10.09.11358); and Facebook, as the most relevant social network at the moment[15] (Android v3.3, v23.0.0.22.14 (Nexus 4 Lollipop) and v24.0.0.30.15 (Nexus 5 Lollipop); iOS6 v5.6; iOS7 v6.8; iOS8 v26.0; WP8 v4.1.0.0). **However, Youtube and Facebook are not included in the comparison of the results. Youtube has been excluded from the table because there is no difference in the behavior of this application in any of the handover combinations. As long as the buffer does not empty, the user will be able to watch the video without noticing that there is a handover in progress. However, the application may stop if the handover does not allow to keep filling the buffer to continue playing the video. Facebook does not appear in the table neither, as this application can always recover from the loss of connectivity, retrying to load the user profile (or the target page) in several attempts, either automatically triggered by the application or reloaded by the user.**

As recommended by the developer documentation of the three OS families, HTTP or HTTPS are the way to send or receive small pieces of information, and this is the way that all the Radio applications use for streaming their content. The only application in our experiment set that uses a different protocol is Skype.

In order to mitigate the interruption in network connectivity, the different mobile terminals analyzed fall back to the cellular connection as soon as the current WLAN connection fails, even if there are other known WiFi networks available. Regaining IP connectivity by the 3G, gives more time to the mobile terminal to scan in the WiFi channels and connect

[12] The application tested is the one by *Los 40* radio station.

[13] The application tested is the one by *Cadena 100* radio station.

[14] The application tested is the one by *RNE* radio station.

[15] http://www.dreamgrow.com/top-10-social-networking-sites-by-market-share-of-visits-may-2013/

Table 6: Survival to handover for applications of different nature in the three OS families. Intra-technology handover

			Skype				**Radio 1**				**Radio 2**				**Radio 3**			
			3G on		**3G off**		**3G on**		**3G off**		**3G on**		**3G off**		**3G on**		**3G off**	
			AP	user	AP	user	AP	user	AP	user	AP	user	AP	user	AP	user	AP	user
Same ESSID	Same IP subnet, same channel	A 4.2	■		■		□		□		■		■		■		■	
		A 5.0 Nexus 4	□		□		□		□		■		■		■		■	
		A 5.0 Nexus 5	□		□		□		□		■		■		■		■	
		iOS6	□		□		░		□		□		□		■		■	
		iOS7	□		■		□		□		□		□		□		□	
		iOS8	□		■		□		░		■		■		□		□	
		WP8	□		□		□		□		□		□		■		■	
	Same IP subnet, different channel	A 4.2	■		■		□		□		■		■		■		■	
		A 5.0 Nexus 4	□		□		□		□		■		■		■		■	
		A 5.0 Nexus 5	□		□		□		□		■		■		■		■	
		iOS6	□		□		□		■		□		□		■		■	
		iOS7	□		▓		░		□		□		□		□		□	
		iOS8	□		■		□		░		■		■		□		□	
		WP8	□		□		□		□		□		□		□		■	
	Diff IP subnet, same channel	A 4.2	■		■		□		□		■		■		■		■	
		A 5.0 Nexus 4	□		□		□		□		■		■		■		■	
		A 5.0 Nexus 5	□		□		□		□		■		■		■		■	
		iOS6	■		□		░		░		■		■		■		■	
		iOS7	░		■		■		■		░		░		□		□	
		iOS8	□		■		□		░		■		■		□		□	
		WP8	□		░		■		■		□		░		■		■	
	Diff IP subnet, different channel	A 4.2	■		■		□		□		■		■		■		■	
		A 5.0 Nexus 4	□		□		□		□		■		■		■		■	
		A 5.0 Nexus 5	□		□		□		□		■		■		■		■	
		iOS6	■		■		■		■		■		■		■		■	
		iOS7	□		■		□		■		■		■		□		□	
		iOS8	□		■		□		░		■		■		□		□	
		WP8	░		▓		■		■		░		░		■		■	
Different ESSID	Same IP subnet, same channel	A 4.2	■	■	■	■	□	□	□	□	■	■	■	■	■	■	■	■
		A 5.0 Nexus 4	□	□	□	□	□	□	□	□	■	■	■	■	■	■	■	■
		A 5.0 Nexus 5	□	□	□	□	□	□	□	□	■	■	■	■	■	■	■	■
		iOS6	□	■	□	□	□	□	□	■	□	□	■	■	■	■	■	■
		iOS7	□	□	░	░	□	░	□	■	□	□	□	□	□	□	□	□
		iOS8	□	□	■	■	□	□	░	░	■	■	■	■	□	□	□	□
		WP8	□	□	□	□	■	■	■	■	□	□	□	□	■	■	■	■
	Same IP subnet, different channel	A 4.2	■	■	■	■	□	□	□	□	■	■	■	■	■	■	■	■
		A 5.0 Nexus 4	□	□	□	□	□	□	□	□	■	■	■	■	■	■	■	■
		A 5.0 Nexus 5	□	□	□	□	□	□	□	□	■	■	■	■	■	■	■	■
		iOS6	□	□	□	□	□	□	□	■	□	□	■	■	■	■	■	■
		iOS7	□	□	■	░	□	░	□	■	□	□	□	□	□	□	□	□
		iOS8	□	□	■	■	□	□	░	░	■	■	■	■	□	□	□	□
		WP8	□	□	■	□	■	■	■	■	□	□	□	□	■	■	■	■
	Diff IP subnet, same channel	A 4.2	■	■	■	■	□	□	□	□	■	■	■	■	■	■	■	■
		A 5.0 Nexus 4	□	□	□	□	□	□	□	□	■	■	■	■	■	■	■	■
		A 5.0 Nexus 5	□	□	□	□	□	□	□	□	■	■	■	■	■	■	■	■
		iOS6	□	▓	■	■	□	■	□	■	■	■	■	■	■	■	■	■
		iOS7	□	□	■	░	□	□	□	■	□	□	□	□	□	□	□	□
		iOS8	□	□	■	■	□	□	░	░	■	■	■	■	□	□	□	□
		WP8	□	□	■	□	■	■	■	■	░	░	□	□	■	■	■	■
	Diff IP subnet, different channel	A 4.2	■	■	■	■	□	□	□	□	■	■	■	■	■	■	■	■
		A 5.0 Nexus 4	□	□	□	□	□	□	□	□	■	■	■	■	■	■	■	■
		A 5.0 Nexus 5	□	□	□	□	□	□	□	□	■	■	■	■	■	■	■	■
		iOS6	□	■	■	■	□	■	□	■	■	■	■	■	■	■	■	■
		iOS7	░	░	■	░	□	□	□	░	□	□	□	□	□	□	□	□
		iOS8	□	□	■	■	□	□	░	░	■	■	■	■	□	□	□	□
		WP8	▓	■	■	■	■	■	■	■	□	░	□	□	■	■	■	■

Table 7: Survival to handover for applications of different nature in the three OS families. Inter-technology handover

		Skype	**Radio 1**	**Radio 2**	**Radio 3**
3G → WiFi	**A 4.2**	■	□	■	■
	A 5.0 Nexus 4	□	□	■	■
	A 5.0 Nexus 5	□	□	■	■
	iOS6	□	□	□	□
	iOS7	□	□	□	□
	iOS8	□	□	□	□
	WP8	□	□	□	■
WiFi → 3G	**A 4.2**	■	□	■	■
	A 5.0 Nexus 4	▨	□	■	■
	A 5.0 Nexus 5	▨	□	■	■
	iOS6	■	□	□	■
	iOS7	▨	▨	□	□
	iOS8	□	□	□	□
	WP8	▦	■	□	■

Color guide for application behavior during handover in Table 6 and Table 7:
■ Application interrupts ▦ Annoying, finally interrupts ▨ Acceptable glitches □ Seamless handover

to another AP if there are any other networks available. However, the change in the technology involves a change in the IP address in use for currently ongoing connections, and this may impact open connections more than the link layer handover, depending on the implementation. There are applications that can survive these two changes in the current connection, while others are interrupted as soon as the current network interface loses connectivity. Note that the applications that easily survive are those that can benefit from buffering the content (Youtube) but note that buffering does not necessarily imply a seamless handover, as not all the applications that buffer the content can survive a handover (Radio 3) concluding that the survival of an application is implementation dependent. It does not depend neither on the OS nor the API offered to application developers, as in the same system different applications can overcome the handover interruption whereas others do not regain connectivity anymore – Radio 1 and Radio 3 in Windows Phone, respectively. In addition, in the case of the applications that hand-off successfully, even though the change of access technology back to 3G involves an additional change in the IP configuration, it helps improving the user experience making the interruption smoother.

In the light of the results in Tables 6 and 7 the first issue that calls our attention is the poor performance of the application Radio 3, which cannot survive the change of the point of attachment to the network. Secondly, we identify that Radio 2 performs significantly better in the Windows Phone 8 than in Android or iOS. A user of this application in the Windows terminal can continue listening to the radio station with a seamless handover or with a minimal interruption in every case, while an Android or iPhone user would stop being provided the service. However, while the Android Radio 2 application fails in every handover, under any circumstances, the iPhone Radio 2 application shows different behavior under different scenarios and for iOS6, iOS7 and iOS8: *i)* for iOS6, the application cannot handle a handover that involves a change in the IP subnet; a change in the point of attachment within the same ESS is supported but, changing to a different ESS this application only survives if the 3G interface is enabled *ii)* for iOS7, Radio 2 interrupts only when IP subnet and channel change within the same ESS; *iii)* for iOS6, iOS7 and iOS8, the application tolerates inter-technology handovers, although the application for iOS8 is the one that interrupts the most. It is also remarkable the improvement from iOS6 to iOS7 versions of the same application, especially for Radio 2 and Radio 3, as well as in the case of iOS8, where Radio 1 does not interrupt when 3G connection is on and Radio 3 can always maintain the connection. However, Radio 2 does not show an improvement, as it interrupts playing.

The issue of changing the IP address is not trivial. A similar behavior is observed for Radio 1: the Windows terminal can handle the change of AP unless it involves a change in the IP address of the target subnet. Similarly, the iPhone terminal starts experiencing trouble even when the target wireless network has the same ESSID, but a different IP. When the target AP operates in the same channel, as the scanning process takes less time, the application can recover, but that is the only case. When the ESSID is different between both networks, the application running on the iPhone terminal can only handle the handover when the action that triggers the change comes from the network, but not when the terminal decides to terminate the connection. This has been overcome in iOS7, only if the connection can fall back to the cellular interface, as waiting for having configured the new WLAN network connection adds too much delay. However, the Android terminal can manage the change in connectivity for Radio 1 without interruption in every case.

By observing the results in Table 6 we can see that the applications running in the Android 4.2 device present a more homogeneous behavior and mostly do not tolerate the handover, but for the application Radio 1. However, the Android update 5.0, has allowed to maintain an ongoing Skype call in several scenarios, while applications Radio 2 and Radio 3 still show the same poor performance and interrupt when handover occurs. It is important to note that although the application Radio 1 **seems to keep the transmission without interruption, the user actually hears the last packets before the handover twice. That is to say, the sound keeps being heard, but if, for instance, listening a song, a part of it will be heard twice before recovering the connection, with the subsequent impairment of user experience.**

Last, in the case of Skype, the application running on the Windows phone outperforms the other two versions. This result was expected, being a proprietary solution by the same manufacturer. Among the rest, the Android implementation offers the poorest performance, not being able to survive the handover in any case. It is worth pointing out that in the case of inter-technology handover from 3G to WiFi, Windows Phone 8 and iPhone terminals are able to keep the call active for two different reasons: the iOS device turns down the 3G interface when the WiFi network becomes available, then, the call just survives the change of connection and takes advantage of the higher bandwidth of the WiFi network to overcome packet losses during the interruption. On the other hand, the Windows Phone 8 device keeps the ongoing call on the 3G connection, even though the WiFi connection becomes ready and active to be used by other applications. We have confirmed that this only happens when the WiFi connection becomes available while there is an ongoing call, but if the call starts when the device is already connected to a WLAN, every call will use that connection. In any case, Skype is the most sensitive application among the ones we tested. We cannot claim that it does not handle a change in network connectivity, but it will interrupt an ongoing call in case the communication between the two endpoints is lost for more than 15 seconds.

7. Summary

In this section we present a summary of the main findings for the different families of OSes and highlight the differences and features that called our attention, categorizing them into five groups:

Simultaneous usage of network interfaces: Out of the three OS families under study, Windows Phone 8 and Android Lollipop allow simultaneous usage of cellular and WLAN interfaces. The Windows Phone 8 device keeps active connections over the cellular and the WLAN interfaces simultaneously. Android, only in its latest version (Lollipop) modified its policy to allow simultaneous connection through both interfaces. The cellular connection remains active for 30 seconds after ongoing sessions finish. iOS devices finish the open connections on the cellular interface when a connection to a WLAN is established.

Network selection: None of the systems under study perform a network selection algorithm to decide on the best network available to connect to. They rely on the network used in the last connection, if it is available. In addition, none of the systems uses information on link quality or performance to change point of attachment if needed. iOS8 WiFi client is particularly sticky, even not responding to user choices and remaining attached to the current AP if the signal is not lost.

Connection establishment: Android and Windows Phone 8 renew their IP address by reissuing a DHCP discovery every time they connect to a different AP, but iOS does not if the change of AP takes place within the same ESS. In the initial attachment to a WLAN, Android outperforms the other systems in the link layer attachment, but it is considerably slower in the IP configuration. The WP8 terminal has proven to be the fastest one, on average, for initial attachment to an already visited WLAN. It calls our attention the remarkable impairment of performance in the connection establishment to WiFi networks in Android Lollipop running on Nexus 5. The delay in the connection establishment for iOS8 is also slightly higher than in previous versions, but the difference is not as notorious as for Lollipop (on Nexus 5). Although the IP configuration has been made faster, the connection to the AP has been considerably damaged. It is also new with respect to previous versions, the use of Gratuitous ARP in iOS8 and the CTS-to-self frames sent before authentication frames by iOS8 (iPhone 6) and Lollipop (Nexus 5).

IPv6 configuration: The three OS families implement privacy extensions for SLAAC [20], configuring an IPv6 address that does not match their respective EUI-64 identifiers. Actually, this is not the only IPv6 address configured in the terminal interface, so applications should handle this and take into account that this kind of address will change over time, which may affect ongoing sessions. The first DNS query sent by the Android phone always requests an IPv6 address, so IPv6 takes precedence over IPv4. However, the IPv6 configuration takes a significantly longer period to be completed. On the contrary, WP8 issues first the IPv4 query but the delays for IPv4 and IPv6 configurations are comparable. The cellular networks available for our experiments do not offer IPv6 support for the moment. Although standardization bodies have provided guidelines for the migration to IPv6, to our knowledge, at the time of writing only some LTE networks in North America and Europe support IPv6 access.

Handover: Not having any application running increases delay in case of a handover. Android (except for Lollipop) presents a more regular behavior, having a handover latency of 0.9 s for most of the cases evaluated. WP8 and iOS devices present a more variable performance. Particularly, when a handover is initiated by a deauthentication from the AP (between different ESSIDs), the interruption in connectivity through the WLAN can take approximately 5 s, 10 s or 12 s on average for Android, WP8 and iOS systems respectively. However, when the handover happens within the same ESS it is completed in 0.24 s or 0.4 s by the WP8 phone, a result which outperforms the other two systems. When the handover is initiated by the terminal, the fastest handover (90 ms) is performed by the Nexus 5 Android Lollipop device if the user is manually indicating the target network, closely followed by iOS devices. Note that changing to a different channel, even if the user does it manually, increases delay considerably in iOS8, due to its sticky client implementation. However, Android and WP8 offer lower delays than iOS if the terminal needs to scan for available networks to decide on the new AP to connect to ("forget" case in our experimental results). Nevertheless, the new Android version offers a significantly higher delay (around 4 s) and the new iOS version (iOS8) overcomes the issues with the manual connection and lowers the delay to just 170 ms. Only the iOS devices and Android Lollipop perform a re-association when the handover takes place within the same ESS. This diminishes the handover delay for the Lollipop devices, but not for the iOS terminals. Although all the systems fall back to the cellular connection when they lose WLAN connectivity, this change does not increase the delay in the WiFi-WiFi handover. The change to the cellular network shades the interruption in the WiFi interface, allowing for time to perform the scanning and the association to the new AP. The remarkably bad performance of Android Lollipop and iOS8 deserve a special mention, especially in the case of changing to a different channel manually for iOS8 and as a general consideration for Android Lollipop.

Multicast and network traffic: Unsurprisingly, HTTP is the dominant traffic as it is also the recommended way to access remote content in the development documentation. It calls our attention the intensive use of IEEE 802.11 Null function frames. The use of these frames is not specified by the standard, but WiFi clients, especially in smartphones, send a significant amount of these frames regularly. The most extended use of Null Function frames is power management, so the station informs the AP when it goes to sleep or awakes. However, these frames are sent very regularly, and, specifically, when connection to the current AP is lost. In our handover experiments, we have detected that Android and iOS devices try to reach the AP, whose signal is lost, by sending numerous Null Function frames and Probe Requests, delaying the connection to a new point of attachment. It is also remarkable the number of DNS requests required every time that any connection is open. Regarding multicast support, IGMP is not supported in some Android devices, including Nexus 4 and Nexus 5, which we used in our experiments.

8. Open issues and future directions

The thorough assessment of the connection management in the three mobile operating system families under study highlights some flaws in current implementations. Our study reveals that the design of current mobile terminal OS and applications only takes into account the availability of Internet connection, but does not consider the presence of several access networks as a resource. In addition, current implementations do not optimize network access selection or handover and pay minimal attention to connection management, beyond identifying the interface being used and detecting Internet connection. To fill these gaps we identify some potential implementation changes that would be feasible, even easy to **implement**, and that would enhance user experience, reducing latency in the connection and the handover and improving efficiency in the handling of several interfaces.

8.1. Enhanced network selection

If several known WLANs are in range, all the systems analyzed connect to the one they were connected the last time. None of them takes into account the signal strength or link quality towards the different APs as a criterion to choose what network to connect to. If this were considered, handover after a short time could be avoided. Current drivers and firmware as the ones in smartphones have full access and capabilities to monitor certain key indicators of the performance or link quality. Keeping track of changes in these parameters and other decision policies are easily implementable in software tools like *wpa_supplicant*. Even further improvements, as supporting some of the recent IEEE 802.11 standard amendments, are already included in recent versions. The potential changes that we suggest as an example for access network selection are mainly related to the WLAN connection:

- **Connection to the best WLAN**: when the WLAN detects that already visited networks are available, the smartphone connects to the last visited one. Our suggestion is to connect to the network that provides the best link quality at that moment. Another variant of this selection is to keep track of the performance offered by the network in the previous connection – or keep an average measurement of historic connections – at that given location (similar approaches exist in the context of vehicular networks [28]). This choice would be customizable and applicable to different criteria, like security or delay instead of just throughput or signal quality when several of the networks offer similar characteristics.

- **Reduced scanning during handover**: since the mobile terminal keeps sending Probe Request frames after attaching to the WLAN AP, this information could be used to speed up the process of handover, shrinking the interval that the terminal spends scanning again after disassociating from the previous AP and connecting to the new one. Moreover, the mobile terminals keep sending Probe Requests frames to all the visited networks. As the terminal already has all the information about the user location and movement, the scanning could adapt to the user location, only scanning for nearby networks, reducing considerably the number of frames being sent regularly.

- **Early detection of low link quality**: similarly to prioritizing the connection to the AP that offers the highest signal quality, the link quality should be monitored, given the dynamic nature of wireless networks. This monitoring would allow a quick reaction when the link quality gets low, making the current connection likely to fail. Moreover, we could take advantage of the considerable amount of frames that are exchanged with the AP constantly, as we have checked that, apart from the active scanning, the three OS families that we have evaluated send 802.11 Null Function frames at all times for power management and signaling purposes. Depending on the terminal capabilities, the current open connections could be handed-off to the cellular interface and start trying to connect to a new WLAN in order to avoid interruptions before the current one fails.

- **Get information from network repositories**: standardization bodies have made an effort in the specification of different information repositories and distribution such as ANDSF [29], ANQP [30], ALTO [31] and MIIS [32] [33]. Their generalized deployment and the access to this information can help the mobile device to choose the best access network to connect to, increase performance in network-assisted handovers and contribute to more efficient network management.

8.2. Multi-interface management and integration of mobility protocols

The IP layer constitutes a reasonable level to offer inter-technology mobility support, being the most widespread network layer protocol and in use with different access technologies underneath. IP mobility has been a research topic for a long time and the different solutions designed to allow a user to freely roam across different points of attachment are a clear example of the evolution of the research on this topic, continuously adapting to the new requirements imposed by operators. Although there are hundreds of different solutions, none of them has been a clear market success and none is massively deployed. The current panorama on mobility management is somehow mixed, since mobility solutions have only been deployed within the cellular operator boundaries, e.g., a user can freely roam across the different access networks defined by 3GPP, but there is no solution for inter technology handover or IP mobility within non-3GPP technologies in the wide sense[16], due to the lack of support in the network and in the terminal. The lack of a common IP mobility solution implemented in the majority of smartphones and networks results in the inter-dependence between the mobile user experience and the smartphone mobile services exposed to the applications.

The network connectivity management in multi-interfaced devices can mainly follow two models, widely known as *weak host* and *strong host* models. The weak host model will accept any packet destined to one of its IP addresses, regardless of the interface where the packet is received. On the contrary, the strong host model will only accept the packet if the destination address matches that one of the interface which received it. Different operating systems decide to implement one or the other. For instance, Linux implements the weak host model, whereas Windows Vista and Windows 7 default to the strong host, although weak host model behavior is configurable. Such implementation decisions affect the performance of the devices, especially when different access technologies are available. We argue that a flow mobility solution [34, 35] may enhance the user experience when a handover takes place with an ongoing communication. Current smartphones, which have multiple interfaces and can connect to different access technologies, need a connection manager that enables this feature. For Internet access, the two main technologies currently used are cellular (UMTS, LTE) and IEEE 802.11. Nevertheless, the most common approach is to use only one of the interfaces at a time, missing the benefits that the usage of both interfaces simultaneously could provide. One of these benefits, currently being discussed by standardization bodies is 3G offloading. By offloading some of the flows at the mobile terminal over a congested cellular network to a WiFi network whenever it is possible not only is convenient for the user, who enjoys greater bandwidth with less delay and at a lower cost, but also for the network operator that frees resources to serve other users. The offloading can be selective depending on the application running or respond to user requirements.

From our experiments, the only OSes that make possible the simultaneous usage of 3G and WLAN interfaces are Windows Phone 8 and the latest version of Android (Lollipop). In this way, e.g., a Skype call can continue without interruption even though the mobile terminal attaches to a WLAN that just became available. The attachment to the new network is totally seamless for the ongoing call as it keeps going through the cellular interface, but the applications that can tolerate a hand off, or that start after it, will be bound to the WLAN interface. This, as we have reported in Section 6.3 improves considerably the performance in case of an inter-technology handover, which is seamless for the user. However, once the ongoing session has finished, the new connections will use the WiFi in-

[16] Some technologies have their own mobility support at link layer but the connections at the terminal will not be able to survive an IP address change.

terface mandatorily, without a choice, for instance, if the user needs to establish a longer session and they will be on the move or if they need to ensure session continuity. According to the IETF [36] there are different approaches for connectivity management in multi-interface devices: *i)* per-application, *ii)* centralized, system-wide or based on user input *iii)* stack-level solutions to specific problems. If flow mobility were enabled, applications could choose their default interface, mobility requirements if more than one access technology is available or specify minimum resources or QoS needs according to the network interface. Both hardware and software tools in current smartphones allow the implementation of this kind of policies, although it increases complexity both in the development of the applications and the connectivity management. In addition, the mobile OS has to deal with multiple applications running in parallel, which may or may not specify network connectivity requirements in the same way. Therefore, the system needs to provide a combined approach, with a default policy, system-wide, based on information available and at the same time offer the possibility of a more advanced connectivity management per-application, if it is specified. Moreover, the management of simultaneous connectivity opens issues as routing, default address selection [37] and the selection of parameters to be configured on a per-interface basis [36].

9. Conclusions

In this article we have studied a feature that commonly goes unnoticed, not existing much documentation about its operation: the network manager in smartphones. To that purpose, we have analyzed the connection procedure of the three most popular mobile OS families – Android, iOS and Windows Phone 8 – and we have studied the performance when a handover, both inter- and intra-technology takes place. We have also examined how the handover impacts the performance of some applications and affects user experience. Finally, we introduced some optimizations that are directly extracted from the conclusions gathered as a result of our experiments. The potential optimizations that we propose are the base for our future lines of research. The mobile terminal that presented the least network attachment delay and the most advanced features in terms of connectivity management is the Windows Phone 8 terminal. WP8 and Android Lollipop allow both the cellular and WiFi interfaces to be active at the same time. Unfortunately, WP8 is also the one that offers the most restricted access to the device's features and the least flexibility for potential modifications resulting from our research. Indeed, in terms of flexibility and room for modifications, the mobile terminal chosen for performing further improvements is the Android device. The open source licensing and the root access provide a convenient development environment to continue improving the connectivity management in current mobile devices. **As we have confirmed in this article, the three analyzed OS families access the network in a very similar way and perform also similarly when the point of attachment changes. Due to these similarities, the conclusions of the potential enhancements to the connection manager performed in Android, could be ported to the other two platforms, by adapting it to their software stack.**

Acknowledgment

This work has been partially supported by the European Community through the CROWD project, FP7-ICT-318115.

References

[1] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2014-2019, White paper, 2015. URL: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.pdf.

[2] P. Daponte, L. D. Vito, F. Picariello, M. Riccio, State of the art and future developments of measurement applications on smartphones, Measurement 46 (2013) 3291 – 3307.

[3] IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007) (2012) 1–2793.

[4] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, D. Estrin, Diversity in Smartphone Usage, in: Proceedings of the 8th international conference on Mobile systems, applications, and services, MobiSys '10, ACM, New York, NY, USA, 2010, pp. 179–194. URL: http://doi.acm.org/10.1145/1814433.1814453. doi:10.1145/1814433.1814453.

[5] W. Song, Y. Kim, H. Kim, J. Lim, J. Kim, Personalized Optimization for Android Smartphones, ACM Transanctions on Embedded Computing Systems 13 (2014) 60:1–60:25.

[6] J. Sommers, P. Barford, Cell vs. WiFi: On the Performance of Metro Area Mobile Connections, in: Proceedings of the 2012 ACM conference on Internet measurement conference, IMC '12, ACM, New York, NY, USA, 2012, pp. 301–314. URL: http://doi.acm.org/10.1145/2398776.2398808. doi:10.1145/2398776.2398808.

[7] S. Liu, A. Striegel, Casting Doubts on the Viability of WiFi Offloading, in: Proceedings of the 2012 ACM SIGCOMM workshop on Cellular networks: operations, challenges, and future design, CellNet '12, ACM, New York, NY, USA, 2012, pp. 25–30. URL: http://doi.acm.org/10.1145/2342468.2342475. doi:10.1145/2342468.2342475.

[8] K. Fukuda, K. Nagami, A Measurement of Mobile Traffic Offloading, in: Proceedings of the 14th international conference on Passive and Active Measurement, PAM'13, Springer-Verlag, Berlin, Heidelberg, 2013, pp. 73–82. doi:10.1007/978-3-642-36516-4_8, http://dx.doi.org/10.1007/978-3-642-36516-4_8.

[9] X. Chen, R. Jin, K. Suh, B. Wang, W. Wei, Network Performance of Smart Mobile Handhelds in a University Campus WiFi Network, in: Proceedings of the 2012 ACM conference on Internet measurement conference, IMC '12, ACM, New York, NY, USA, 2012, pp. 315–328. URL: http://doi.acm.org/10.1145/2398776.2398809. doi:10.1145/2398776.2398809.

[10] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, P. Bahl, Anatomizing Application Performance Differences on Smartphones, in: Proceedings of the 8th international conference on Mobile systems, applications, and services, MobiSys

'10, ACM, New York, NY, USA, 2010, pp. 165–178. URL: http://doi.acm.org/10.1145/1814433.1814452. doi:10.1145/1814433.1814452.

[11] Q. Zhang, C. Guo, Z. Guo, W. Zhu, Efficient mobility management for vertical handoff between WWAN and WLAN, IEEE Communications Magazine 41 (2003) 102–108.

[12] C. Guo, Z. Guo, Q. Zhang, W. Zhu, A seamless and proactive end-to-end mobility solution for roaming across heterogeneous wireless networks, IEEE Journal on Selected Areas in Communications 22 (2004) 834–848.

[13] G. Bollano, D. Panno, F. Ricciato, M. Turolla, N. Vaccaro, D. Ettorre, Enhanced Android Connection Manager: An Application-Based Solution to Manage Mobile Data Traffic, in: World Telecommunications Congress (WTC), 2012, 2012, pp. 1–6.

[14] Y. Liu, F. Li, L. Guo, B. Shen, S. Chen, A Comparative Study of Android and iOS for Accessing Internet Streaming Services, in: Proceedings of the 14th international conference on Passive and Active Measurement, PAM'13, Springer-Verlag, Berlin, Heidelberg, 2013, pp. 104–114. URL: http://dx.doi.org/10.1007/978-3-642-36516-4_11. doi:10.1007/978-3-642-36516-4_11.

[15] Apple Inc., iOS7 Developer Library, ???? [Online]. Available: https://developer.apple.com/library/ios/navigation.

[16] R. Droms, Dynamic Host Configuration Protocol, RFC 2131 (Draft Standard), 1997. URL: http://www.ietf.org/rfc/rfc2131.txt, updated by RFCs 3396, 4361, 5494, 6842.

[17] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, M. Carney, Dynamic Host Configuration Protocol for IPv6 (DHCPv6), RFC 3315 (Proposed Standard), 2003. URL: http://www.ietf.org/rfc/rfc3315.txt, updated by RFCs 4361, 5494, 6221, 6422, 6644, 7083, 7227, 7283.

[18] T. Narten, E. Nordmark, W. Simpson, H. Soliman, Neighbor Discovery for IP version 6 (IPv6), RFC 4861 (Draft Standard), 2007. URL: http://www.ietf.org/rfc/rfc4861.txt, updated by RFCs 5942, 6980, 7048.

[19] S. Thomson, T. Narten, T. Jinmei, IPv6 Stateless Address Autoconfiguration, RFC 4862 (Draft Standard), 2007. URL: http://www.ietf.org/rfc/rfc4862.txt.

[20] T. Narten, R. Draves, S. Krishnan, Privacy Extensions for Stateless Address Autoconfiguration in IPv6, RFC 4941 (Draft Standard), 2007. URL: http://www.ietf.org/rfc/rfc4941.txt.

[21] F. Gont, A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC), RFC 7217, 2014. URL: http://www.ietf.org/rfc/rfc7217.txt.

[22] B. Aboba, D. Thaler, L. Esibov, Link-local Multicast Name Resolution (LLMNR), RFC 4795 (Informational), 2007. URL: http://www.ietf.org/rfc/rfc4795.txt.

[23] B. Cain, S. Deering, I. Kouvelas, B. Fenner, A. Thyagarajan, Internet Group Management Protocol, Version 3, RFC 3376 (Proposed Standard), 2002. URL: http://www.ietf.org/rfc/rfc3376.txt, updated by RFC 4604.

[24] R. Vida, L. Costa, Multicast Listener Discovery Version 2 (MLDv2) for IPv6, RFC 3810 (Proposed Standard), 2004. URL: http://www.ietf.org/rfc/rfc3810.txt, updated by RFC 4604.

[25] H. Holbrook, B. Cain, B. Haberman, Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast, RFC 4604 (Proposed Standard), 2006. URL: http://www.ietf.org/rfc/rfc4604.txt.

[26] E. Nygren, R. K. Sitaraman, J. Sun, The akamai network: A platform for high-performance internet applications, SIGOPS Oper. Syst. Rev. 44 (2010) 2–19.

[27] W. Gu, Z. Yang, D. Xuan, W. Jia, C. Que, Null Data Frame: A Double-Edged Sword in IEEE 802.11 WLANs, IEEE Transactions on Parallel and Distributed Systems 21 (2010) 897–910.

[28] A. Balasubramanian, R. Mahajan, A. Venkataramani, B. N. Levine, J. Zahorjan, Interactive wifi connectivity for moving vehicles, ACM SIGCOMM Computer Communication Review 38 (2008) 427–438.

[29] G. T. 23.402, Architecture Enhancement for Non-3GPP Accesses (Release 12) v12.5.0, 2014.

[30] IEEE Standard for Information Technology-Telecommunications and information exchange between systems. Local and Metropolitan networks. Specific requirements. Part II: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Amendment 9: Interworking with External Networks, Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, IEEE Std 802.11w-2009, IEEE Std 802.11n-2009, IEEE Std 802.11p-2010, IEEE Std 802.11z-2010, and IEEE Std 802.11v-2011 (2011) 1–208.

[31] J. Seedorf, E. Burger, Application-Layer Traffic Optimization (ALTO) Problem Statement, RFC 5693 (Informational), 2009. URL: http://www.ietf.org/rfc/rfc5693.txt.

[32] IEEE Standard for Local and metropolitan area networks - Media Independent Handover Services, IEEE Std 802.21-2008 (2009).

[33] L. Sarakis, G. Kormentzas, F. Guirao, Seamless service provision for multi heterogeneous access, Wireless Communications, IEEE 16 (2009) 32–40.

[34] T. Melia, C. J. Bernardos, A. de la Oliva, F. Giust, M. Calderon, IP Flow Mobility in PMIPv6 Based Networks: Solution Design and Experimental Evaluation, Wireless Personal Communications 61 (2011) 603–627.

[35] A. De La Oliva, C. Bernardos, M. Calderon, T. Melia, J. Zuniga, IP flow mobility: smart traffic offload for future wireless networks, IEEE Communications Magazine 49 (2011) 124–132.

[36] M. Wasserman, P. Seite, Current Practices for Multiple-Interface Hosts, RFC 6419 (Informational), 2011. URL: http://www.ietf.org/rfc/rfc6419.txt.

[37] D. Thaler, R. Draves, A. Matsumoto, T. Chown, Default Address Selection for Internet Protocol Version 6 (IPv6), RFC 6724 (Proposed Standard), 2012. URL: http://www.ietf.org/rfc/rfc6724.txt.