Aalto University
School of Science
Degree Programme of Computer Science and Engineering

Jussi-Pekka Erkkilä

Web and Native Technologies in Mobile Application Development

Master's Thesis
Espoo, Feb 19, 2013

Supervisor:	Professor Jukka K. Nurminen, Aalto University
Instructor:	Antti Saarinen M.Sc. (Tech.)

Aalto University
School of Science
Degree Programme of Computer Science and Engineering

ABSTRACT OF MASTER'S THESIS

Author:	Jussi-Pekka Erkkilä		
Title: Web and Native Technologies in Mobile Application Development			
Date:	Feb 19, 2013	**Pages:**	97
Professorship:	Data Communication Software	**Code:**	T-110
Supervisor:	Professor Jukka K. Nurminen		
Instructor:	Antti Saarinen M.Sc. (Tech.)		

In recent years, the mobile application development has became an increasingly important area in the software industry. However, there are multiple different and incompatible mobile platforms and ecosystems on the market. The issue for application developers is supporting all the different devices and platforms.

New web programming technologies, such as CSS3 and HTML5, bring new opportunities for the mobile application development. Since practically every mobile device includes a web browser, web technologies provide a way to cover almost all modern mobile devices by writing a single application. This thesis reviews the current status of HTML5 and other web technologies, and compares them to the native, platform-specific development technologies. The main focus of the thesis is to evaluate the advantages and disadvantages of the web technologies in mobile application development, and how the web technologies affect on resource usage, performance and user experience in mobile applications.

As a conclusion we state that whereas the native technologies provide an optimal user experience and performance, the web technologies provide fast and flexible way to produce cross-platform mobile applications. The web technologies already provide a competitive alternative to the native technologies. However, the best technology for implementing a mobile application depends on several factors, such as business objectives, target audience and technical requirements.

Keywords: mobile, web, application development, Android, power consumption, memory usage, performance, user experience

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan tutkinto-ohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Jussi-Pekka Erkkilä		
Työn nimi: Web- ja natiiviteknologiat mobiilisovellusten kehityksessä			
Päiväys:	19. helmikuuta 2013	**Sivumäärä:**	97
Professuuri:	Tietoliikenneohjelmistot	**Koodi:**	T-110
Valvoja:	Professori Jukka K. Nurminen		
Ohjaaja:	Diplomi-insinööri Antti Saarinen		

Viime vuosina mobiilisovellusten ohjelmistokehitys on noussut merkittävään rooliin ohjelmistoteollisuudessa. Markkinoilla on kuitenkin useita keskenään yhteensopimattomia mobiililaitteita ja -alustoja. Useiden eri laitteiden ja järjestelmien tukeminen on ohjelmistokehittäjille merkittävä haaste.

Uudet web-ohjelmointiin tarkoitetut tekniikat, kuten CSS3 ja HTML5, avaavat uusia mahdollisuuksia mobiilisovellusten kehittäjille. Koska käytännössä kaikki modernit mobiililaitteet sisältävät Internet-selaimen, web-tekniikoita hyödyntäen voidaan kehittää mobiilisovelluksia, jotka toimivat lähes kaikissa mobiililaitteissa. Tämä diplomityö käsittelee HTML5:n ja sen ympärille rakentuvien web-tekniikoiden käyttöä mobiilisovellusten ohjelmistokehityksessä. Diplomityön päätavoite on arvioida web-tekniikoiden tarjoamia hyötyjä ja haittoja verrattuna perinteisiin alustakohtaisiin ohjelmistokehitystekniikoihin, sekä sitä, miten web-tekniikoiden käyttäminen sovelluskehityksessä vaikuttaa mobiililaitteiden resurssien käyttöön, suorituskykyyn sekä käyttökokemukseen.

Johtopäätöksenä esitetään, että perinteiset laite- ja alustakohtaiset tekniikat tarjoavat parhaan käyttökokemuksen ja suorituskyvyn mobiilisovelluksille. Sen sijaan web-tekniikat tarjoavat nopean ja joustavan tavan tuottaa alustariippumattomia mobiilisovelluksia. Yhteenvetona todetaan, että web-tekniikat tarjoavat kilpailukykyisen vaihtoehdon mobiilisovellusten tuottamiseen, mutta paras tekniikka yksittäisen sovelluksen toteuttamiseen riippuu tapauskohtaisesti useasta eri tekijästä, kuten sovelluksen kohdeyleisöstä, teknisistä vaatimuksista ja kaupallisista tavoitteista.

Asiasanat:	mobiili, web, sovelluskehitys, Android, virrankuluts, muistin käyttö, suorituskyky, käyttökokemus
Kieli:	Englanti

Acknowledgements

I would like to thank my instructor Antti Saarinen and my long-time mentor Matti Saarinen from Whatamap.com Ltd., for all the instructions and guidance during this thesis and my studies as whole. Additionally, I am thankful for my supervisor Jukka Nurminen for taking time to supervise my thesis.

Also, I want to thank all my friends, colleagues and people from Aalto University department of Computer Science, who have provided me with comments, feedback and ideas during this project.

Last, but not least I want to thank my wife Jenni and our sweet children Elsa and Aarni for their supportive role on background, despite all the long days and nights I have spent with my studies.

Espoo, Feb 19, 2013

Jussi-Pekka Erkkilä

Abbreviations and Acronyms

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CPU	Central Processing Unit
CSS	Cascading Style Sheet
CSS3	Third revision of CSS.
DOM	Document Object Model
GPU	Graphics Processing Unit
HTML	Hypertext Markup Language
HTML5	Fifth revision of HTML.
JS	JavaScript
JSON	JavaScript Object Notation
RAM	Random Access Memory
UI	User Interface
SDK	Software Development Kit
VM	Virtual Machine
WWW	World Wide Web
W3C	World Wide Web Consortium
XML	eXtensible Markup Language

Contents

Abbreviations and Acronyms **v**

1 Introduction **1**

1.1 Diversity of Mobile Platforms and Devices 2

1.2 Objectives and research questions 3

1.3 Thesis structure 3

2 Related work **5**

3 Overview of Web and Mobile Technologies **7**

3.1 Web Technologies 7

3.1.1 Structure of web applications 9

3.1.2 JavaScript 9

3.1.3 AJAX, JSON 10

3.1.4 HTML5 and related APIs 12

3.2 Mobile Platforms 13

3.2.1 Android 13

3.2.2 Apple iOS 16

3.2.3 Other alternatives 17

4 Web as a Mobile Application Platform **19**

4.1 Advantages of Web 20

4.2 Capabilities and constraints 21

4.2.1 I/O and hardware access 22

4.2.2 Communications 24

4.2.3 Data storage and offline content 26

4.3 Support for HTML5 in the mobile 27

4.4 Hybrid solutions 30

4.5 Security considerations 31

5 Web vs. Native - Experimental testing 33
5.1 Methods and goals 33
5.2 Testing environment 35
5.2.1 Hardware 35
5.2.2 Test applications 37
5.2.3 Automated use case 40
5.3 User Experience test 40
5.3.1 Implementation 40
5.3.2 Results 41
5.3.3 Evaluation 42
5.4 Performance measurement 42
5.4.1 Implementation 42
5.4.2 Results 44
5.4.3 Evaluation 46
5.5 Memory usage 47
5.5.1 Implementation 47
5.5.2 Results 48
5.5.3 Evaluation 50
5.6 Energy consumption 51
5.6.1 Implementation 51
5.6.2 Results 53
5.6.3 Evaluation 56

6 Discussion and Analysis 57
6.1 Sum up and analysis of results 57
6.1.1 User Experience 58
6.1.2 Performance 58
6.1.3 Memory usage 59
6.1.4 Power consumption 60
6.2 Pros and Cons 61
6.2.1 Native technologies 63
6.2.2 Web technologies 63
6.2.3 Hybrid solutions 64

7 Conclusions 65

A Web application source code 73

B Android application source code 78

C Example input data from service 84

D Power consumption graphs **85**

Chapter 1

Introduction

Along with the breakthrough of smartphones, mobile application development has became increasingly significant area in software industry. The most popular marketplaces for mobile applications – Apple App Store and Google Play – combine more than one million third-party applications, whereas number of application downloads per year is counted in billions. For many brands, it is more a rule than an exception to have an own mobile application in common marketplaces.

Usually the mobile applications are written and built using proprietary tools and APIs (Application Programming Interfaces) provided by mobile platform vendors. In this thesis we call these *native applications.* In practice, a native application is an application that is built and designed for a specific mobile platform or device – or a set of devices. A native application can not be installed and run on any other platform, expect the one that it is built and designed for.

An alternative to native applications are *web applications.* Web applications are applications that are run inside web browser. Instead of the proprietary and the platform specific components, web applications utilize open and standardized web technologies for implementing functionality and providing user interface and interaction.

Also, it is possible to combine web and native applications so that a significant part of an application is implemented using the web technologies and packaged inside a native mobile application using web view components. All these three different approaches include some advantages and disadvantages, which we are going to discuss in this thesis.

1.1 Diversity of Mobile Platforms and Devices

In contrast to personal computers, for mobile devices no *de facto* standard operating system exists. Of course there are few platforms – such as Google Android and Apple iOS – with significant market shares, but to cover 90% of all mobile users, one needs to write an application for four different platforms (Android, iOS, Symbian and Blackberry) [11]. With a high probability, these numbers will change rapidly in the near future. The market share of Symbian has been falling down for a while, and on the other hand Windows Phone is expected to gain more market share shortly. And still there are other platforms, such as MeeGo and Samsung's Bada, with millions of devices sold.

Moreover, it is not only a number of the platforms but also a number of different software versions and physically different devices that cause headache for software developers. Hundreds of Android devices are on market, with wide variety of hardware specifications. To cover all Android devices, an application should be compatible with about 10 different Android API versions and be able to adapt on dozens of different screen resolutions. [33]

Of course, other mobile platforms are not that fragmented. Apple iOS, for instance, is much more homogeneous platform used only in a couple of different mobile devices. However, to develop and maintain an application that covers most mobile devices, requires remarkable amount of work and a wide skill set. And that does not mean only a couple of programming languages, but also custom development tools, software development kits (SDKs) and proprietary APIs that the developers should handle – separately for each mobile platform.

Only software component that is relatively compatible through every platform and mobile device is a web browser. All modern smartphones and mobile devices are shipped with a web browser that is capable to interpret JavaScript code and view graphical UI components written using HTML and CSS. Naturally, there are many drawbacks and issues in writing an application by using web technologies but the new features of HTML5 and CSS3 are closing the gap between web applications and native applications in mobile devices. In this thesis we focus more closely on the web technologies and whether they can solve the problems with fragmentation of the mobile platforms and the devices.

1.2 Objectives and research questions

The main goal of this thesis is to evaluate the capabilities and drawbacks of the web technologies compared to the native technologies. The question is not, whether the web technologies can be used to build mobile applications. They can, plenty of real-life examples exist.

However, we know that the web technologies are not capable for anything. There are limitations for example in UI-construction, performance and hardware access. We are going to evaluate, how significant these drawbacks are compared to advantages, that web technologies provide us. Here are the main questions that we are finding answers to.

1. What are the main advantages of the web technologies and what opportunities they provide?

2. How utilizing the web technologies affects on mobile applications in sense of resource usage, performance and usability?

3. What are the most critical limitations of the web technologies and how can we address these limitations.

In addition, this thesis provides a comprehensive overview of current status of HTML5, JavaScript and other relevant web technologies. We review the HTML5 support in the latest mobile browsers. Also we discuss about the differences between the native and the web technologies from both architectural and design perspective.

The purpose of this thesis is not to compare different mobile platforms to each other. Instead, we are comparing the web technologies to the native technologies independent of mobile platform. In examples and the research part, we will focus primarily on Android. However, the results are evaluated and discussed in a wider perspective, not only from Android platform's point of view.

1.3 Thesis structure

This thesis is structured as follows. After the introduction, we review the related works that have been done on this area. After that, we evaluate the current status of the web technologies and the mobile platforms – how mobile applications actually work from software technical point of view, and how we define the web technologies.

The fourth chapter describes, how the web can actually be used as a mobile application platform, and what are the constraints and the opportunities of the web technologies. Whereas the fourth chapter is mostly a literature study, the fifth chapter investigates certain properties of mobile applications in practice. The sixth chapters analyzes and discusses the results and findings of the fourth and the fifth chapter. The final chapter concludes the whole thesis.

Chapter 2

Related work

There have been a lot of discussion and debate around HTML5 and mobile platforms lately. Bloggers and developers have been arguing about pros and cons of HTML5 and native development technologies. The general consensus is, that native applications work smoother and provide better look and feel, whereas development and deployment of web applications is fast and low-cost [16, 22, 23, 29].

Charland and Leroux (2011) compare native and web technologies in mobile application development [6]. Their main concerns are performance and user experience, and they also measure the performance of JavaScript using Google's V8 benchmark suite[1] and SunSpider JavaScript benchmark[2]. They state, that performance of web technologies has not yet attained the level of native technologies, but the gap is closing.

Mikkonen and Taivalsaari (2011) suggest that "use of open Web applications will eventually surpass the use of custom native applications on mobile devices" [25]. They list several strengths of the web, such as platform-independent software code and instant worldwide deployment, to support their conjecture. They also emphasize the role of HTML5 and open standards in the future.

There are also pessimistic views. Savage (2012) argues that HTML5 has not met the high expectations, especially in mobile development [38]. He argues that web technologies do not solve the fragmentation problem, because of incompatibility of the mobile browsers. Also, he emphasizes the low performance of mobile browsers, and lack of 3D mobile games using WebGL. While Savage has good points, he ignores the fact that many popular mobile applications already utilize web technologies[3].

[1]http://v8.googlecode.com/svn/data/benchmarks/current/run.html
[2]http://www.webkit.org/perf/sunspider/sunspider.html
[3]http://phonegap.com/app

Mahemoff (2011) argues that web applications are catching up native applications in performance and features [22]. He also provides results on how graphics rendering and code execution performance have improved in the web applications lately. Execution performance was measured using Google's V8 benchmark suite whereas graphics rendering was measured by tracking frames-per-second (FPS) values of different graphical demos running on HTML5 canvas.

In contrast, Spaceports.io published a study that compares graphics rendering in mobile and desktop browsers [40]. They argue, that graphics rendering in mobile browsers is tens or even hundreds of times slower than in desktop browsers. As a test suite they used an open-source project PerfMarks[4]. The tests were run using two different iOS and Android devices, and MacBook Pro laptop with 2.5GHz Intel Core i7 CPU and 16GB of RAM. However, the study covers only hardware accelerated CSS transforms. Hence, it is practically a comparison of GPU speeds between mobile devices and a high-end PC.

In addition, generic programming language comparisons exist as well. The Computer Language Benchmarks Game[5] provides a comprehensive comparison of memory usage and performance of different languages. The benchmarks were run on x86 computers running Ubuntu Linux 12.04 using several test programs. Every test case was run as as child-process of a Python script. GTop system information library was used for measuring CPU load, and memory usage was read every 0.2 seconds. Whereas C outperforms JavaScript clearly both in performance and memory usage, the study shows that Google Chrome's V8 JavaScript engine performed better than other common scripting languages such as Perl, PHP or Python. Also, in memory usage JavaScript was more conservative than Java.

Whereas the studies indicate that performance of JavaScript and web technologies has improved lately, there are few experimental studies specific to mobile platforms – especially when it comes to comparison of web and native technologies. Moreover, in addition to performance, in mobile field there are many other relevant factors, such as usability and resource consumption. These are the issues that are considered in this thesis.

[4]https://github.com/sibblingz/PerfMarks

[5]http://shootout.alioth.debian.org/

Chapter 3

Overview of Web and Mobile Technologies

In this chapter we present an overview of the web technologies and the existing mobile platforms. This is not really a focus of the thesis, but a necessary part to understand the differences between the concepts of web application and native application.

We focus on software stack of mobile platforms and the environments where native applications and web applications are run. Also, we review the programming languages, APIs and tools that are used to produce applications on different platforms.

3.1 Web Technologies

Since the beginning of 1990s, the World Wide Web has evolved from a simple text document sharing platform to an enormous content distribution network and application platform [26]. This change has been evolutionary and more spontaneous than controlled. New technologies and solutions have been developed to meet needs and requirements of users.

During the 1990s, many technologies were introduced to provide rich and interactive contents to web pages. Many of the technologies, such as RealPlayer, Shockwave and Apple QuickTime, were browser plugins to enable playing multimedia files inside web browsers. Web pages began to appear more like multimedia presentations than ordinary text pages [41]. Along with introduction of DHTML (Dynamic HTML) [24], developers were able to create interactive web pages with ability to dynamically update the contents of pages. DHTML is combination of CSS (Cascading Style Sheet), DOM (Document Object Model) and JavaScript - all of which are still among the most important technologies for building web applications.

Also, server-side scripting became increasingly popular as the web evolved to its current form. Technologies such as CGI and PHP made possible to easily invoke program code on the server-side, according to web client requests. In practice, this enabled many basic features of current web applications, such as "logging in" to web pages and storing users' data on web services. Web pages were not longer static but dynamically generated documents depending on request parameters, browser cookies and other variables.

Hence, the web technologies can roughly be divided in two groups: client-side technologies that are run inside browser or the client application, and server-side technologies that are run as a service on remote server and can be called from client. In practice, the difference is fundamental from architectural point of view. [34]

In this thesis, we focus primarily on client-side technologies. The client-side technologies are run in mobile devices and can be compared to the native mobile technologies. The server-side technologies are used in remote services and can be called from mobile client – no matter which technology have been used to build the mobile application. Server-side technologies may also have important role in mobile application development, for example in outsourcing the computing to a cloud service. However, the technologies that have been used for implementing the remote services, are not relevant from client application's point of view.

In web application development, there are three relevant languages that we focus on: HTML – especially HTML5, JavaScript and CSS. HTML is a markup language for defining the UI layout and creating the visual elements of web application. CSS (Cascading Stylesheet) can be used for styling the HTML elements and web pages. JavaScript is a programming language for implementing functionality and interaction between the user and the application. Also, fundamental parts of web development are concepts such as DOM (Document Object Model), AJAX (Asynchronous JavaScript and XML), JSON (JavaScript Object Notation) and XML (eXtensible Markup Language). These are used mainly for dynamic contents update and passing the data between remote services and end-user applications. All of the previously mentioned technologies are important part of concept of Web 2.0. [30]

Also, there is a bunch of various mobile web frameworks, such as jQuery Mobile[1] and Sencha Touch[2]. These are application-level extensions and libraries that focus on accelerating the web application development, providing the cross-browser support and improving the usability. However, these are

[1]http://jquerymobile.com

[2]http://www.sencha.com/products/touch

out of the scope of this thesis.

3.1.1 Structure of web applications

In web applications, user interface is usually written in HTML and styled either by using the specific HTML attributes or CSS. HTML is an open standard for describing the structure and contents of web pages. The essential features of HTML are hyperlinks and tags for describing the structure of web pages, such as headers, paragraphs, images and body text. XHTML refers to specific version of HTML where the document structure fills both HTML and XML definitions.

CSS is a style sheet language that can be used for styling the documents written in markup languages. CSS is commonly used on websites along with HTML, and it is well supported in most graphical web browsers. Basically, CSS can be used to describe the look and the formatting of visible HTML elements, but the new standard CSS3 enables also animated styling such as transforms, rotations and sliding of HTML elements.

DOM (Document Object Model) is a cross-platform, language- and browser-independent interface that allows dynamic updating and modifying of the markup language documents [46]. From a web developer's point of view, DOM enables programmatic modification of HTML pages without reloading the page. The DOM API is implemented in the most web browsers, and can be called with JavaScript.

3.1.2 JavaScript

JavaScript is a dynamic, weakly typed scripting language with object-oriented capabilities. JavaScript was initially developed by Netscape in the middle of 1990s [27]. In this thesis we examine JavaScript primarily as a client-side web development language, although it can be used for other purposes as well.

The usual way for writing JavaScript code is to embed the code inside web pages or in separate js-files which can be included on a HTML page. JavaScript can be used to manipulate the web page's user interface and contents, and create interactivity for web pages without any communication between a web client and a server [7]. When a website is loaded, the browser parses, interprets and executes the JavaScript code. Currently, almost all web browsers include a native support for JavaScript.

Along with the introduction of Web 2.0 and technologies such as AJAX, DOM and HTML5, JavaScript has became a serious application programming language. If the web will evolve to a main application platform in the future, JavaScript may also evolve to one of the most important programming language in the software industry. JavaScript has already gained growing attention in recent years. Currently, most web applications are written in JavaScript, and many of those may include thousands of lines of code [42]. Increasing number the of advanced JavaScript libraries are developed rapidly and JavaScript virtual machine technology is one of the key topics on today's browser wars. [6, 27]

One existing problem with JavaScript is performance. Being a high-level, interpretable language, it is natural that JavaScript code is slower to execute than corresponding code written in low-level languages such as C or Assembly. Also, JavaScript applications are run in a restricted sandbox inside a web browser. However, the performance of the JavaScript engines has improved significantly lately [27]. In chapter 5, we experiment the performance of the JavaScript applications compared to native mobile applications.

3.1.3 AJAX, JSON

AJAX (Asynchronus JavaScript and XML) is a term used to refer a set of technologies that can be used to make websites look and feel more like dynamic applications instead of static pages. In classic concept, all the interaction between a web server and a client was made by accessing to a specific URL and loading the web page. In contrast, AJAX can be used to access remote web services asynchronously on background. The web page is not reloaded but only parts of it are updated once the response is received.

According to Garret (2005) [10] definition, AJAX consists of the following technologies:

- HTML and CSS for presentation
- Dynamic displaying and interaction with DOM (Document Object Model)
- XML for data interchange
- XMLHttpRequest for asynchronous HTTP requests and data retrieval
- JavaScript for binding everything together

XML is not actually the only format for data interchange. The technology itself does not place any restrictions on which format the data should be. However, the XML have been $defacto$ standard for a long time. Recently JSON (JavaScript Object Notation) has became a considerable alternative.

JSON objects are less redundant as well as more efficient to parse and generate [44]. Also, JSON suits well to be used with JavaScript because JSON objects can be easily assigned to JavaScript arrays.

Below we present a graph that visualizes the differences of an AJAX call and a classic HTTP request.

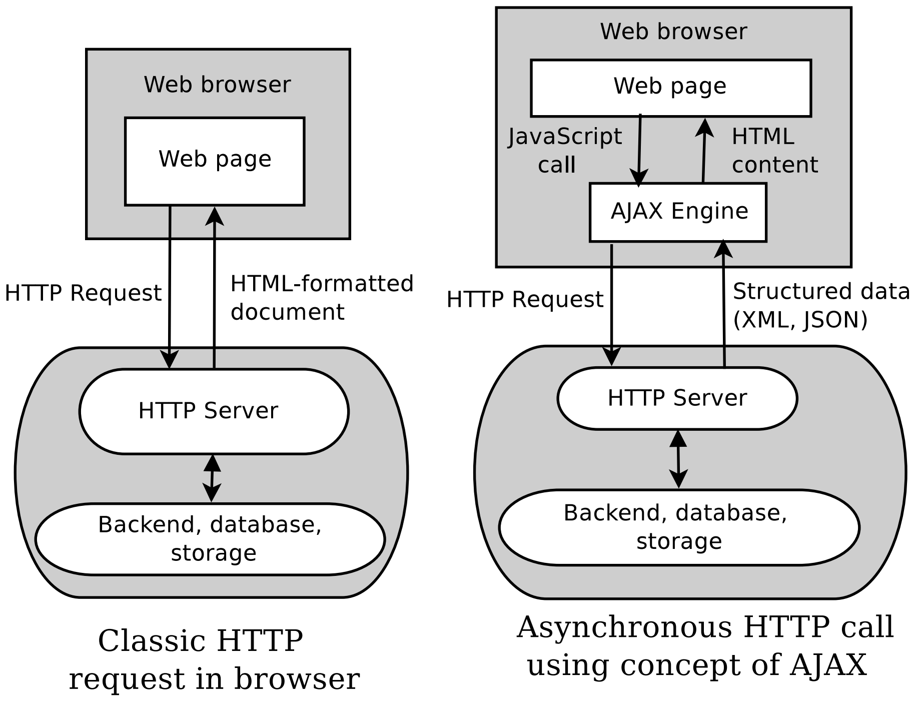

Web browser
Web page
HTTP Request
HTML-formatted document
HTTP Server
Backend, database, storage
Classic HTTP request in browser
Web browser
Web page
JavaScript call
HTML content
AJAX Engine
HTTP Request
Structured data (XML, JSON)
HTTP Server
Backend, database, storage
Asynchronous HTTP call using concept of AJAX

Figure 3.1: Normal web page request compared to asynchronous AJAX call.

Here the AJAX engine is responsible of sending an HTTP request and receiving the response. Once the response is received, a specific callback-function is invoked where the response is processed and the DOM is manipulated accordingly. Compared to the classic web model, AJAX call does not improve the latency of data transfers as every AJAX call still requires a new TCP connection and HTTP handshake.

Updating the contents of web application asynchronously improves the user experience, since the operation is non-blocking and user can meanwhile use the application normally. Also, by using AJAX the amount of data transferred between web application and remote services can be optimized. In overall, AJAX has a significant role in bridging the gap between native applications and web applications, especially in sense of user experience and interactivity. [19]

3.1.4 HTML5 and related APIs

HTML5 is a fifth revision of the HTML standard [48]. As its predecessors, HTML5 is a markup language for describing and structuring the web pages. The new features of HTML5 turn the web to a highly potential application platform. In contrast to proprietary web technologies such as Adobe Flash or Microsoft Silverlight, HTML5 is an open standard and will eventually be natively supported by the most web browsers without external plugins.

The new features of HTML5 include both new syntactical features - such as *video* and *canvas* elements and also APIs (Application Programming Interfaces) which can be called with JavaScript. These APIs include immediate mode 2D drawing on canvas elements as well as timer-based callbacks and offline applications with application cache. [49]

The standardization of HTML5 is not yet finished. The specification work is being done by W3C HTML Working Group and WHATWG. Both working groups have published own specifications, but the contents of these are quite similar. However, the official standardization is expected to take still years which may cause problems since the software vendors are writing mutually incompatible implementations. [25]

In addition, along with HTML5 many other related JavaScript APIs have been specified as well. These APIs are not part of W3C HTML5 specification, but those are compatible with upcoming HTML5 standard and may be referred as HTML5 APIs in informal contexts.

Many of the APIs are still W3C's internal drafts whereas part of them are stabilized and implemented in most web browsers. The details are outside of this thesis' context, but we list here the W3C working groups that are relevant from this thesis' point of view and briefly describe their objectives and main features.

- **W3C Device APIs Working Group.** W3C Device APIs Working Group is specifying the APIs that enable web applications to interact with the device services such as calendar, camera and battery state. APIs include Battery API, Network Information API, Vibration API, media capture API and several others. [51]

- **W3C Geolocation Working Group.** W3C Geolocation Working Group defines an interface for reading the sensor data of the device in order to provide real-time location information for location-aware web applications. APIs specified are Geolocation API that is already available in most browsers, and DeviceOrientation Event Specification that provides access to compass and accelerometer data. [52]

- **W3C Web Applications Working Group.** WebApps Working Group is developing the standard APIs for client-side development to enable richer web applications. The work include File APIs for accessing to a local file system, Web Sockets API for two-way communication between a remote host and a client, Web Worker API for thread-like operations and many others. [50]

- **Khronos Group WebGL Working Group.** WebGL Working Group specifies a web API that enables interactive, GPU accelerated 2D and 3D graphics rendering using HTML5 canvas elements. [17]

- **W3C Web Real-Time Communications Working Group.** Web Real-Time Communications Working Group defines a real-time peer-to-peer communication channel between web browsers. [55]

Together with related web standards, the HTML5 will extend the possibilities of the web as an application platform in a completely new way. Web browsers are expected to support interactive graphics rendering, web sockets, video, audio, offline contents and many other features, even 3D acceleration. Hence, this makes possible to port almost any native application to a web browser platform. [42]

Despite the lack of official standards and final specifications, HTML5, WebGL and many other APIs are relatively well supported by latest versions of the major web browsers. We will focus more closely on the HTML5 support in the mobile devices at section 4.3.

3.2 Mobile Platforms

This section describes the common mobile platforms from application developer's point of view. As one of our goals is to compare the mobile platforms to the web, it is useful to understand the principles and the basic structure of mobile application platforms, and how end-user applications are implemented.

We are keeping in practice and focusing primarily on two of the main mobile platforms, Apple iOS and Google Android. Also we briefly overview the alternative platforms at the end of this section.

3.2.1 Android

Android is an open-source software stack and operating system for mobile devices. It is developed by Open Handset Alliance which is a consortium of many high technology companies, led by Google. [3]. During the recent years, Android has bypassed Symbian as the dominant smartphone platform. As mentioned earlier, currently around 60% of all smartphones sold globally are running on Android [11].

The Android platform is a complete software stack and operating system that runs on Linux kernel. It consists of several layers, including low level libraries, Android Runtime and application framework. The low level libraries are native libraries part of Linux kernel, written mainly in C/C++. The application framework provides a Java interface to the libraries. An overview of Android software stack is visualized in Figure 3.2.

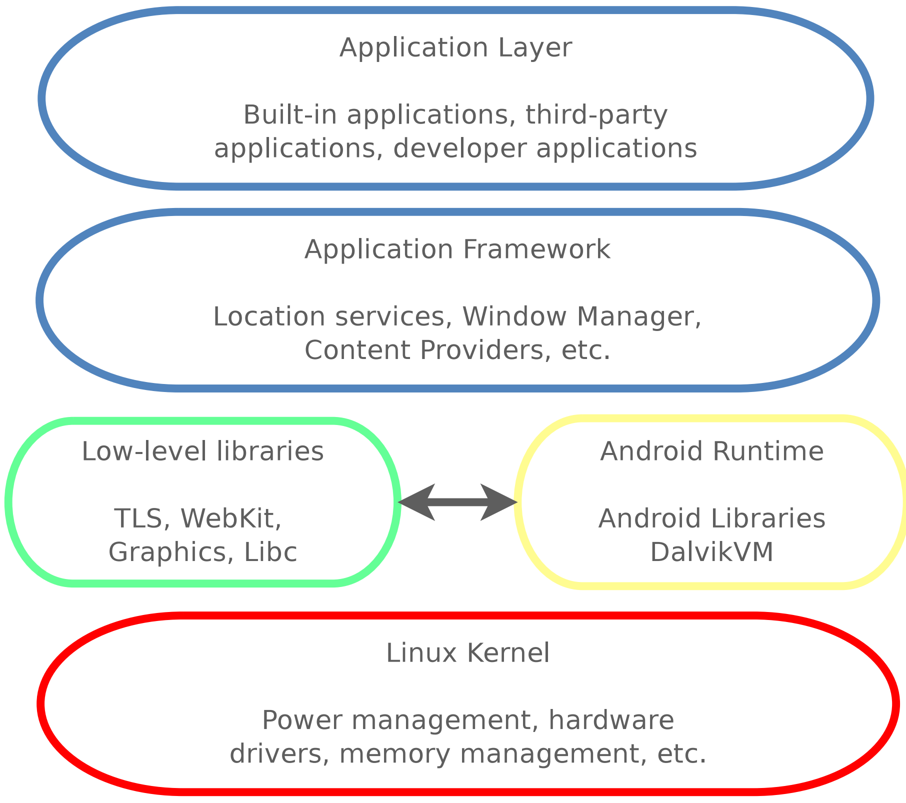

Application Layer
Built-in applications, third-party applications, developer applications
Application Framework
Location services, Window Manager, Content Providers, etc.
Low-level libraries
TLS, WebKit, Graphics, Libc
Android Runtime
Android Libraries DalvikVM
Linux Kernel
Power management, hardware drivers, memory management, etc.

Figure 3.2: Android software stack

All end-user applications are written in Java and they utilize the application framework. Applications are run inside Dalvik Virtual Machine (commonly referred as DalvikVM). DalvikVM is a register based virtual machine and it executes Dalvik Executable (.dex) files that are optimized for low memory footprint [9]. Every application is run on its own instance of DalvikVM and multiple instances can be run simultaneously [37]. These features of DalvikVM enable efficient application level multi-tasking. Also it is good to notice, that Android applications running inside DalvikVM are not actually *native* in traditional sense, since they are running inside a virtual machine. However, from application developer's point of view, these can be referred as native applications. Below we present a table about the relevant information of the Android platform.

Devices	Hundreds of mobile devices, including many Samsung, HTC and Google smartphones and several tablets.
Versions	More than ten releases, a few major versions that cover most of all Android devices.
Market share	64.1% of devices sold in Q2 / 2012. [11]
Development languages and tools	Java, Android SDK. Development tools available for multiple platforms. Applications may be executed either on real devices or emulator.
Application distribution	Through Google Play or as idenpendent APK packages (requires allowance for 3rd party software installation).

Table 3.1: Android platform properties from software developer's point of view.

In this thesis, we do the most of experimental research using Android device Samsung Galaxy S. The reason for this is that Android is the most popular smartphone platform in the world and it is relatively open. Hence, measuring the certain properties – such as memory footprint and battery consumption – is straightforward compared to the alternatives that tend to be relatively closed platforms.

3.2.2 Apple iOS

Apple iOS (formerly known as iPhone OS) is an operating system initially published for the first version of iPhone in 2007. Currently iOS is used as platform for Apple's iPad and iPod devices as well. The latest published version of iOS is 6.0. After Android, iOS is the most popular mobile device platform with about 19% market share in Q2 / 2012. [11]

Compared to Android, iOS is quite restricted platform from application developer's point of view [4]. Certain functionalities are disabled either because of architectural limitations or Apple's application policy. However, these policies have resulted in better security as we will see in section 4.5.

Apple provides application development tools for iOS, including Xcode (graphical IDE for application development), SDK, Interface Builder for UI design and implementation, as well as simulator for testing and running applications without real devices. The primary development language is Objective-C, which is an extension to C. Thus native C can be used as well. [12]

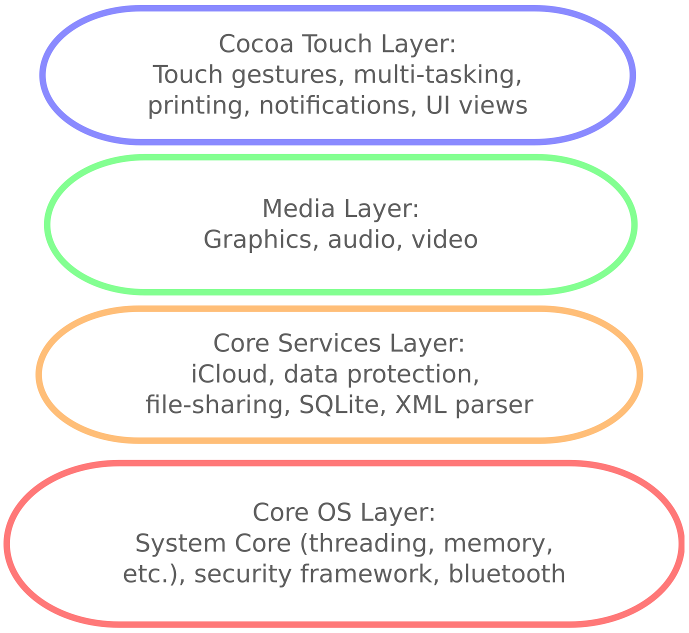

Cocoa Touch Layer:
Touch gestures, multi-tasking, printing, notifications, UI views
Media Layer:
Graphics, audio, video
Core Services Layer:
iCloud, data protection, file-sharing, SQLite, XML parser
Core OS Layer:
System Core (threading, memory, etc.), security framework, bluetooth

Figure 3.3: iOS architecture layers and some of their features.

Architecture of iOS consists of four layers: Core OS, Core Services, Media and Cocoa Touch. The higher-level frameworks provide high-level abstractions of certain technologies and functionalities that are implemented on lower level [5]. However, all the layers are accessible from application code with certain limitations. [4, 5]

Installing uncertified third-party applications on iOS devices is disabled because of Apple's application policy. Hence, the only way to develop and publish applications on iOS is joining iOS Developer Program and submitting the applications on Apple App Store. Apple has defined rather strict requirements for the applications that are published in App Store.

Devices	Apple's iPhone, iPad and iPod touch, including different hardware versions
Versions	Six major versions, several minor updates. No official statistics, approximately 95% of devices run iOS 5 or iOS 6 [39].
Market share	18.8% of devices sold in Q2 / 2012. [11]
Development languages and tools	Obective-C, iOS. Tools available only for OS X. Applications may be executed either on real devices or emulator.
Application distribution	Only via AppStore, reviewed and approved by Apple.

Table 3.2: Apple iOS from software developer's point of view.

3.2.3 Other alternatives

Since Android and iPhone combine roughly around 80% of smartphone market share, other alternatives exist as well. These include Windows Phone, Symbian, Samsung Bada, RIM's Blackberry OS and a few others. Most notably, both Blackberry OS and Symbian possess market share of around 6%. However, as Nokia is migrating to Windows Phone as its primary smartphone platform, numbers of Symbian are expected to fall in near future as well as Windows Phone is expected to increase its market share.

We do not go in details or market analysis here, but we present a simplified table listing the basic information of the alternative mobile platforms.

Point to note here is that Firefox OS, Open webOS (formerly HP webOS) and MeeGo support the web technologies as an official application development technology. Also, all of them are relatively open Linux-based mobile platforms. However, the future views of the projects are still unclear. Nokia

Platform	**Developer**	**Development technologies**
Windows Phone	Microsoft	.NET
Symbian	Nokia / Accenture	C++ / Qt, Java, Python, widgets using Web technologies
Bada	Samsung	C++
MeeGo	Nokia	C++ / Qt, QML, Python, Web technologies
Blackberry OS	RIM	Java (J2ME)
Open webOS	HP, Community	Web technologies, C/C++
Firefox OS	Mozilla, community	Web technologies (JavaScript, HTML5)

Table 3.3: Alternative software platforms for smartphones.

abandoned development of MeeGo, but the project was ultimately continued by a new start-up company Jolla Mobile. Hewlett-Packard, on the other hand, announced that they will publish the webOS under open-source license and they plan to contribute to the project in the future as well [13]. Firefox OS is not officially published or distributed on any device. The project is under development, but is has been demonstrated on several Android-compatible devices.

Chapter 4

Web as a Mobile Application Platform

In the world of desktop computers, the web has already gained a strong position as an application platform. Back to 1990s, users had to download or buy their applications or games, and install those on their computers. However, since the broadband connections became common for end-users in the beginning of 2000s, the web has evolved at a rapid pace. Currently many applications – especially those related to social media – are published as a web services. This is actually what both end-users and application distributors prefer.

Web is user-friendly. Whereas installing applications on a computer might be difficult task for non-technical users, everyone can browse the web. On the other hand, service providers can efficiently gather data about users of online applications and that way optimize and target advertisement better, and generate more revenue.

Google already provides a full office suite as a web application. Web is full of smallish games and entertainment services, online-TV services, social media services and many application like services. Actually, there are quite few applications that users really need to install their computer any more – in addition to web browsers and plugins. Heavy 3D-games are among these, as well as applications for syncing with mobile phones and digital cameras, watching DVDs and manipulating images.

The reason is that all these are closely related to computer hardware - such as USB connection, DVD drive or graphics adapter. Web applications are – for really good reasons – executed inside a restricted environment of a web browser. Hence, web applications can not access to the USB-connected devices or DVD-drive. However, even this may change in the future, since most mobile devices may synchronize the data with cloud services through mobile broadband connections. In addition, WebGL and HTML5 will enable hardware accelerated graphics in web browsers.

In the world of mobile applications, things are still different. Usual way for performing a specific task wit a mobile device is to download and install the corresponding application from a store, such as Google Play or App Store. The application distribution via application stores is strictly controlled by the vendors. Companies such as Apple and Microsoft want to remain in control, which kind of applications can be distributed on their platform. For instance, Microsoft has specified strict policies for the functionality and the appearance of certified Windows Phone applications [31].

Naturally, web applications for mobile devices exist as well. However, the web as an environment draws a few constraints for mobile applications, both in usability and functionality. In this chapter, we investigate the capabilities and constraints of the web as an application platform; what is possible, what is not and how can we address the existing issues. Also, we review the HTML5 support on mobile devices and evaluate the compatibility of the web browsers. Also, when talking about the web, we naturally need to pay attention to security concerns that are discussed in the last section of this chapter.

4.1 Advantages of Web

Web applications can be run practically on any device that includes a modern and graphical web browser. This is the main reason for the potential and the increasing popularity of the web as a platform. As we have pointed out in previous chapters, the fragmentation of mobile devices, platforms and platform versions are a significant problem for software developers. Writing an application that runs on every platform on the market requires a vast amount of resources, let alone that the mobile platforms are all the time under continuous development. New platforms and versions are published in a rapid pace. Web browsers, however, are quite well backwards compatible. Thus, it is likely that applications written using web technologies will work flawlessly on upcoming mobile devices as well.

A small mobile software company may support only one or two mobile platforms to achieve sufficient target group. Nevertheless, other parties, such as public sector or IT departments, may be required to provide a support for a much larger device base. On these situations, the web may be the most reasonable way to achieve the goal. Google's Vice President for Engineering, Vic Gundotra has said that "even Google is not rich enough to support all different mobile platforms" [6]. Hence, for a small business, web technologies are excellent way to cut development costs and save the resources.

Moreover, web technologies fit well for current trends of software industry. Application lifecycles are short, development is rapid and requirements are changing often [42]. Being a lightweight and dynamic language, JavaScript is well capable for this concept. There is no need for bloat SDKs or development tools: all that one needs to start development is a text editor and a web browser. Building a user interface with HTML is fast and flexible and adding functionality with JavaScript is straightforward. Changing and updating web applications is also fast and lightweight process, compared to native applications that need to be recompiled and submitted to application stores.

The web provides advantages for end-users as well. Compared to native ones, web applications are always up-to-date. There is no need to download and install updates. In a fact, there is no need for installations at all. Web applications are not wasting storage space of mobile device: they are always available through Internet connection independent of time, location and device. All that user needs to run web application is Internet connection and a device with capable web browser.

In an ideal world, developer only needs to write a web application once, test it on one web browser and publish on web, and the application should run flawlessly on every platform. However, this is not the reality. Browsers have certain differences that may cause pain for a developer. For example, small differences in interpretation of style sheet may totally break down the user interface on one browser, while it still may work on another. Also, different devices have different screen resolutions and capabilities, such as support for zooming or multi-touch events. These are the issues that we discuss in the following sections.

4.2 Capabilities and constraints

The capabilities and possibilities of the web as an application platform can not be exactly specified. The web itself is nothing more but HTTP protocol and set of rules and standards to define how the hypertext documents should be interpreted and presented to end-user. However, it depends on a browser, how web applications actually appear and work.

Here, we define the capabilities as features and functionalities that are working and usable in practice at least in one environment and one use case. The constraints may be either platform- or browser-specific or general. For instance, web applications can not invoke low-level system calls or launch native applications – expect via possible security vulnerabilities. This is a general constraint. On a few platforms and browsers, web applications are able to catch and handle multi-touch events. However, all devices and platforms do not support multi-touch at all, which is a platform specific constraint. On the other hand, some platforms may include a support for multi-touch but the corresponding web browser may not deliver the multi-touch events to web applications, but catches and handles the events itself.

In addition to technical restrictions – such as lack of full HTML5 support on web browsers – web applications also have some practical drawbacks. Generally, in order to use web applications, Internet connection is required. Thus, availability of web applications depends on Internet connection which may be a problem outside the local operator coverage area (high roaming costs in foreign countries), in the extreme conditions (no network available) or in the large public events, when the network is congested. Also, the primary way to monetize mobile applications is selling those in application stores. In case of web applications, this is not an option as e.g. Google Play and Apple App Store only accept native applications in their catalog. To address these issues, web applications can be packaged and distributed as native applications. However, this in turn requires manual and platform-specific work by developers. Application producers may also find alternative ways for monetizing, such as advertising.

4.2.1 I/O and hardware access

Web applications are run inside web browser's sandbox. Thus, the input events are handled by the web browser and delivered to web application only if the web browser does not catch the event itself. For example, web application can not listen the hardware buttons of Android devices. Functionality of these buttons is defined on lower level of operating system and the click events are never delivered to the web applications.

Moreover, by default web applications have no access to hardware sensors such as GPS, NFC, compass or camera. Also, the mobile device services such as calendar, messages and contacts are unavailable. Animations and page transitions written in JavaScript can not utilize GPU for rendering. However, these are among the problems that HTML5 and related APIs are going to address. As mentioned earlier in this thesis, W3C's working groups are specifying the APIs that enable access to almost any interface in mobile devices. Part of these APIs are already implemented and widely used whereas many of them are still at working draft state. Here we list the common services and functionalities of mobile devices and availability of those in web applications. All API specifications mentioned below are published by W3C working groups expect WebGL which is developed by Khronos Group [17].

Device or service	**Availability**
Accelerometer, compass	Specified in DeviceOrientation Event Specification. First public draft is available.
Ambient light sensor	Public working draft available.
Bluetooth	No specification available.
Camera, microphone	Access specified in API for Media Capture and Streams, which is not yet widely implemented.
Contacts, Calendar	Public working drafts available as Contacts API and Calendar API.
Dialer	No specification available. However, some browsers support “tel:” protocol in hyperlinks for opening the dialer.
Graphical Processing Unit (GPU)	Hardware accelerated graphics provided by WebGL, few early implementations are available in latest browsers.
Touch and multi-touch displays	W3C’s Candidate Recommendation available for touch events [57]. Implemented and supported on several platforms.
Near Field Communications (NFC)	No specification available.
Temperature, humidity, pressure	Internal drafts published by Device APIs Working Group.
Positioning	Obtaining the real-time location of device is specified in Geolocation API. Feature is widely implemented and used.

Table 4.1: Availability of the common device sensors in web applications.

It is worth noticing that usually web applications can not access straight the raw sensor data, even though the hardware would be accessible through web APIs. For example, DeviceOrientation Event API provides high level information about device orientation, motion and acceleration – not the raw data of compass, gyroscope or accelerometers [53]. However, accessing the high-level web APIs is easy and straightforward for web developers. In following example we read device position using Geolocation API.

```
var success = function(e) {
  alert("You are at " + e.coords.latitude + ", " + e.
     coords.longitude);
};
var error = function(e) {
  alert("Unable to retrieve position!");
};
if(navigator.geolocation) {
  navigator.getPosition(success, error, {
    enableHighAccuracy: true
  });
}
```

Listing 4.1: Getting position through JavaScript Geolocation API

In the listing above, we define callback functions “success” and “error”. These are invoked depending on whether the position was obtained successfully or not. If the browser does not support Geolocation API, variable “navigator.geolocation” is undefined. The parameter “enableHighAccuracy” indicates for the API that we want a location with high accuracy, which means usually obtaining the data using GPS. Otherwise the device may provide data using some other method – such as mobile cell positioning – that might be faster but also less accurate.

Many of the previously mentioned APIs are still in specification stage. Standardization and implementation processes may take years, although some experimental implementations may be available.

4.2.2 Communications

Along with HTML5, new APIs have been defined for improving client-server communications as well as for enabling peer-to-peer communication between web browsers. These API definitions are WebRTC [54] and WebSockets [56].

The purpose of WebRTC is to enable real-time communications - such as video and voice calls - directly from browser to browser over web. WebRTC API is being specified by W3C WebRTC Working Group, whereas IETF RTCWEB group is responsible of defining the protocols that together with API will enable the real-time web communications.

On the other hand, WebSockets enable a real-time bi-directional communication channel between a web client and a server. The WebSockets API is specified by W3C Web Application Working Group, whereas the protocol – which is independent of HTTP – is specified in IETF RFC 6455 [15].

Compared to traditional HTTP or AJAX calls, where a client makes the request and waits the response from server, an open WebSocket is a full-

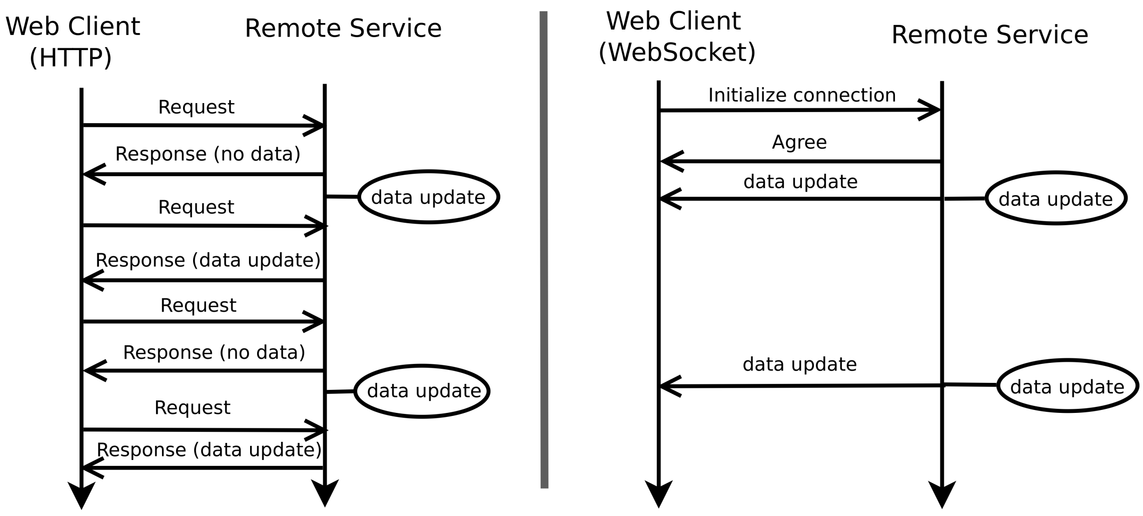


Figure 4.1: On left side, periodic HTTP requests for data updates. On right side, bi-directional WebSocket for delivering updates with low delay.

duplex communications channel with no need for polling data. This is a remarkable difference to the traditional request-response convention as illustrated in figure 4.1. This approach will both decrease the latency and reduce the amount of network traffic. Low redundancy implies that less computing is needed for handling and parsing the data. The lower latency results as better usability in the client application due to faster interaction between client and service [32]. Because of lack of HTTP headers, WebSockets may provide more than 400:1 reduction in unnecessary traffic as well as 3:1 reduction in latency [20].

Using WebSockets in web application is easy through JavaScript API.

```
var socket = new WebSocket("ws://<address>:<port>");
socket.onopen = function(e) {
  // socket opened
  this.send("Hello server, cheers from client!");
};
socket.onmessage = function(e) {
  alert("data sent by server: " + e.data);
};
socket.onclose = function(e) {
  // socket closed
};
```

Listing 4.2: Initializing a WebSocket connection using JavaScript API

As an alternative technology, long HTTP polling allows emulation of server-sent events from a server to a client. The client requests information from the server as normal HTTP requests. However, if the server does not have the information, it does not send the response. Instead, it holds the request and waits until the requested information becomes available. This technology, also known as Comet, utilizes AJAX to emulate server push mechanism. Long polling may achieve almost as low latency as WebSockets [35], but it requires that client actively maintains an active HTTP connection to the server. However, long polling does not require support for HTML5 and WebSockets by the server nor the client.

4.2.3 Data storage and offline content

Traditionally, the only way to store data on a web client have been cookies. Cookies are stored in a web browser and are sent to a remote service inside HTTP headers. Cookies are accessible both from the server and the client, and they are still very common way to save session state in web service.

However, HTML5 opens new possibilities in this area as well. The WebStorage interface defines two attributes: Local Storage and Session Storage. Much like cookies, WebStorage objects are simple key-value pairs with usual methods for getting and setting value for key and removing a key [45].

WebStorage objects are very similar to cookies, but there are a few remarkable differences as well. WebStorage objects are accessible only from web client and the objects are not delivered over every HTTP request. Whereas size of cookie object is defined to be at most 4096 kilobytes [14], most browsers support storage objects with size of five megabytes [45]. The real numbers depend of course on browser-specific implementations, but commonly WebStorage objects may be much larger than cookies.

As mentioned before, there are two types of WebStorage objects. Whereas Session Storage is window- and session-specific temporary storage, Local Storage is a long-standing data store. Basically, data stored in a Local Storage object is written on device mass memory and expected to stay available until it is explicitly removed. Session storage, on the other hand, is emptied as soon as the browser window is closed.

HTML5 enables also offline web applications and use of Web SQL database for saving structured information in web client. However, the development of Web SQL Database specification is not active anymore, although the draft versions have been implemented in many web browsers. The reason for stopping the specification work was that all the vendors implemented the same SQLite backend, whereas W3C needed multiple independent implementations to continue with the specification work [47].

4.3 Support for HTML5 in the mobile

In this section we review the current status of HTML5 support in the mainstream mobile platforms and browsers. It is worth noticing that this an area that changes constantly. New versions of mobile platforms and web browsers are being published all the time. Hence, the information presented in this section might be outdated. To obtain the latest information it is recommended to use some existing web service to check the up-to-date status.

The data was gathered from The HTML5 Test[1] which evaluates every web browser that accesses to the page. The service awards points from every supported feature of HTML5 and related API specifications. The maximum number of points is 500 plus bonus points. Bonus points are awarded from supported data formats, such as audio and video codecs, that are not part of HTML5 specification.

The tests were run with the following hardware and the software set-up:

- Samsung Galaxy S running Android 2.3.3 Gingerbeard. The HTML5 test run with the default browser, Opera Mobile 12.00 and Firefox Mobile 16.
- Nokia N9 running Meego Harmattan PR1.3. The HTML5 test run with Nokia Browser 8.5.0.
- Apple iPhone 4 running iOS 6.0. The HTML5 test run with Mobile Safari.
- Nokia Lumia 800 running Windows Phone 7.5. The HTML5 test run with Internet Explorer Mobile 9.0.

The HTML5 test does not cover all features such as WebRTC or Device APIs. In practice, these are not yet supported in any mobile platforms, although some experimental implementations exist in the latest desktop browsers.

The results are presented at Table 3.1. The table shows overall scores for each platform, and the support for most features of HTML5 and the related APIs. The results clearly show us the recent evolution: the recent browsers, Opera Mobile 12, Firefox 16 and Mobile Safari iOS 6.0, include a significantly better support for the advanced web technologies, than the older browsers. The only difference is IE Mobile 9 in Nokia Lumia 800, which is the most restricted browser, despite the fact that it is newer than Android 2.3.

[1]http://www.html5test.com

As comparison, on desktop computers the most popular browser Google Chrome 448 points (version 23), whereas Firefox 17 achieves 392 points. The evolution of HTML5 support in mobile browsers is illustrated in figure 4.2. Both data and the graph has been received from html5test.com.

[1] In Opera Mobile 12, WebSocket API is supported but not enabled by default.
[2] Firefox 16 has support for WebWorkers but not for SharedWorkers.

Feature	Android 2.3.3	MeeGo PR1.3	Opera Mobile 12	Firefox 16	iOS 6.0	WP 7.5
Scores + bonus	189+1	286+12	369+11	372+9	360+9	138+1
HTML5 Canvas	OK	OK	OK	OK	OK	OK
Audio and video	OK	OK	OK	OK	OK	OK
Offline applications	OK	OK	OK	OK	OK	-
Geolocation	OK	OK	OK	OK	OK	OK
WebGL	-	-	OK	OK	-	-
WebSocket	-	OK	OK[1]	OK	OK	-
FileReader API	-	-	OK	OK	OK	-
FileSystem API	-	-	-	-	-	-
WebStorage API	OK	OK	OK	OK	OK	OK
IndexedDB API	-	-	-	OK	-	-
Workers	-	OK	OK	Partial[2]	OK	-
WebCam	-	-	OK	OK	-	

Table 4.2: The result table of HTML5 test.

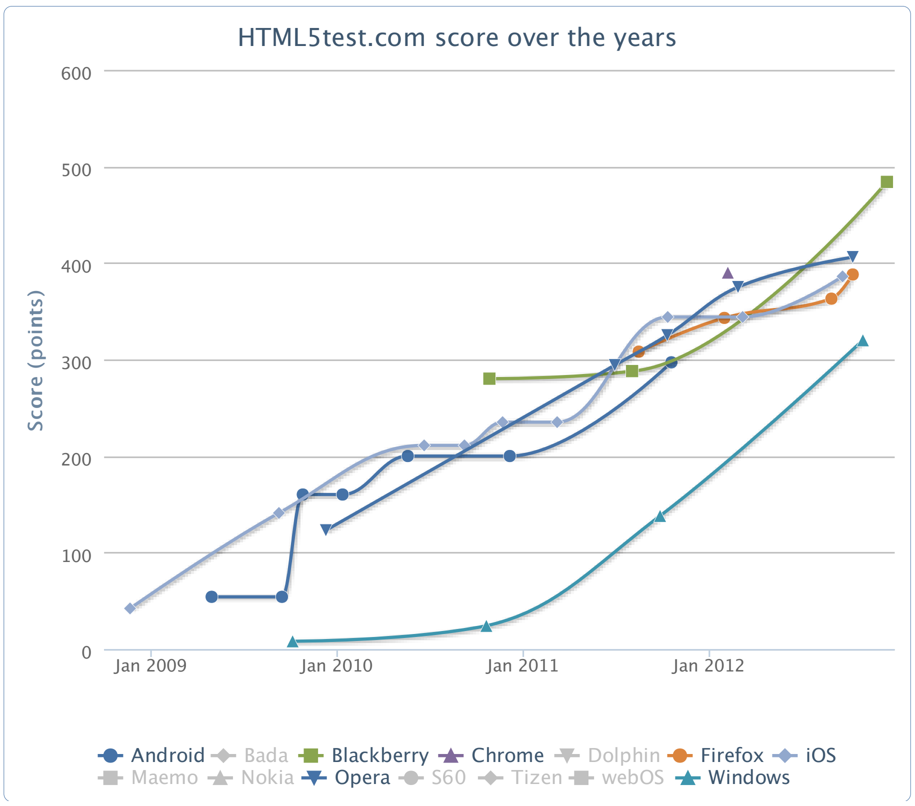


Figure 4.2: Development of HTML5 support on mobile browsers.

4.4 Hybrid solutions

In all situations and use cases, online web applications are not a suitable solution. The reason for this may be either technological limitations or practical needs. Application may require access to a certain service or hardware sensor – such as NFC – that is not accessible through the JavaScript APIs. In addition, users may want to have the application installed permanently on their mobile devices and to be able to run application offline. HTML5 provides a way to do this, but the application is installed inside the browser, not as standalone applications which can be installed and removed as usual. Also, producers may want to advertise and monetize their applications by submitting them to App Store or corresponding marketplace. All these scenarios require that applications are distributed in a native, platform-specific format.

However, the developers may still utilize the advantages of web technologies in application development. The common platforms, such as iOS, Android and Windows Phone, allow embedding so called "web views" inside native applications. Web view is a component of a native application that is able to display HTML pages and interpret JavaScript code within the limits of the platform. Hence, by building a native application with a single, fullscreen web view, one is able to write a native application by using primarily web technologies. In short, a hybrid application is a web applications packaged inside a native application. Creating a web view is straightforward, the following example is for Android:

```
WebView myWebView = (WebView)findViewById(R.id.webview);
myWebView.getSettings().setJavaScriptEnabled(true);
myWebView.loadUrl("file:///android_assets/index.html");
```

There are also frameworks available for accelerating the hybrid application development on mobile platforms. As an advanced and the most common example, PhoneGap[1], which is also distributed as Apache Cordova[2]. PhoneGap is an open-source project and it aims to enable native application development using web technologies in most mobile platforms. PhoneGap utilizes the web views, but it also allows developers to easily write native plugins for web applications. These plugins allow web application to execute and communicate with the native code written by developers.

Hence, PhoneGap combines the advantages of web and native technologies. An application can be packaged and distributed as a native application,

[1]http://www.phonegap.com
[2]http://incubator.apache.org/cordova/

it may include the functionalities available only in native applications, but it still can be implemented mostly by using web technologies.

However, the hybrid approach requires building and publishing applications separately for every platform. Also, all the minor and deprecated mobile platforms may not be supported. Nevertheless, the hybrid solutions provide a compromise solution that may be a reasonable option in many cases.

4.5 Security considerations

When evaluating the web technologies, security must always be considered. Especially in the case of HTML5 and new APIs that enable access to mobile device sensors and even filesystem. Also, the new communication mechanisms – WebRTC and WebSockets – are potential targets of attackers in the near future.

In the mobile world, most malware and other threats are targeted against Android [8]. Moreover, many attacks already exist for web applications using Android WebView [21]. This is natural, as Android is the most popular smartphone platform – and much more open than main competitors Apple iOS and Windows Phone. Also, distributing malware on Android is much easier than on iOS, since Apple does not allow third-party applications to be installed on iOS devices. However, web applications may potentially make a difference here. Web applications are not only accessible from every device but the same code also runs on every device. This is why the security of web applications is really important factor now and in the future.

One problem with HTML5 and related API specifications is lack of official standards. Many of the API specifications are still at draft stage. Hence, the proper security features and models may not be specified and validated, although many of the APIs are already implemented in the latest web browsers. Moreover, the existing security features may not have been audited and confirmed to be trustworthy.

The valid and secure specifications are only part of the solution. The specifications must also be implemented properly. Thus, web browser vendors are in important role when implementing APIs and web browser sandboxing. Since web applications have potentially access to private information of user – such as location, filesystem and even contacts – it is important that user is aware and in control of which device interfaces web applications can access [1].

Among HTML5 and new web APIs, the most critical new features from security perspective are the new communication and data storage technologies. For instance, communication technologies WebRTC and WebSockets, utilize totally new protocols, other than HTTP. WebStorage, on the other hand, enables new ways to store persistent data on web client's memory. Careful implementation is required from all different parties: browser vendors, web service developers and web application developers.

Chapter 5

Web vs. Native - Experimental testing

To answer our actual research questions, we conducted a set of experimental studies. The main purpose of our studies was to find out how the web technologies differ from the native technologies as an application development technology. We studied following properties of both native and web applications:

- Resource usage on mobile devices, including energy consumption and memory footprint
- Raw computing speed
- User experience

This chapter is structured as follows. The section 5.1 presents the research methods, the purpose and the motivation for each separate study. The section 5.2 describes the testing environment including the hardware and the software used in each test case. The following sections describe the implementation and present the results and the evaluation of each research phase. Further discussion and analysis takes place in chapter 6.

5.1 Methods and goals

As mentioned, the research part of this thesis consists of a set of experimental studies. The experimental tests were run on real devices using real applications. The ultimate goal of the study was to find out, how well web applications perform on mobile devices compared to native alternatives. At first we state, that to be a considerable alternative to native applications, web applications should fill at least the following requirements.

1. Common technical requirements must be filled, such as being able to read the device sensors, display graphics and interact to touch events.
2. Performance level of web applications should not be considerable worse than on native applications. That is, web applications should function and run smoothly.
3. User experience must be sufficient, so that users are willing to use web applications.
4. Web applications must not ruin the device performance, such as slow down the background tasks or cause memory leaks.
5. Since the battery life is one of the key factors in the mobile field, web applications should not significantly increase the energy consumption of mobile devices.

In the previous chapters, we reviewed technical capabilities of the web technologies. As pointed out, HTML5 and related API specifications extend the capabilities of web applications significantly. Moreover, hybrid solutions, such as PhoneGap[1] can be used to fill technical requirements if necessary. Thus, the requirement 1. is not a problem.

Measuring the performance of web applications is not simple. Mobile applications do not usually need to solve heavy computational problems. Instead, for mobile users it is relevant that applications work smoothly and nicely, the response times are short and interaction is fluent. However, measuring the response times of touch events, or fluency of the interaction on web applications is not easy. User interaction, such as touch events, are handled and processed primarily by web browser and other lower-level components. Thus, touch events are invoked on a web application with a delay, which may depend on low-level system architecture. Thus, measuring the responsiveness of web applications in different mobile platforms is difficult.

Hence, to evaluate performance of web technologies we implemented two separate studies: a user experience study to evaluate smoothness and overall fluency of web applications, and a performance test to measure the raw computing speed. Both were experimental tests on real devices. In the case of user experience test, the data was gathered as user feedback, whereas the computing performance was measured using a specific application.

[1]http://www.phonegap.com/

The resource usage tests were also run on real hardware and software with a special power meter gathering the data. The purpose was to find out how the web and native solutions consume the device resources: memory and battery. The exact results are available as megabytes (MB) and milliwatts (mW)

5.2 Testing environment

In this section we present the testing environment we used in the research part of this thesis. The environment covers the devices, system versions and settings, as well as the applications we implemented for the measurements.

5.2.1 Hardware

All the tests, except the user experience study, were executed using four different devices described in the table 5.2.1.

Device	**Model number**	**Android version**
Samsung Galaxy S	GT-I9000	2.3.3 Gingerbread
Motorola Milestone 2	MotoA953	2.3.4 Gingerbread
Sony Xperia J	ST26i	4.0.4 Ice Cream Sandwich
Samsung Galaxy S III	GT-I9300	4.1.1 Jelly Bean

Table 5.1: Overview of the test devices

During the tests, the devices did not include SIM card. GSM, Bluetooth and all automatic updates were disabled. Internet connectivity was obtained via WiFi. Only exception was power consumption study, which was implemented separately for WiFi and cellular network, since the data transfer technology has a significant effect on power consumption. Also, factory reset was done for each device before the studies to make sure that environment was clean and conditions were similar for each test case. Each device was running an official Android version either shipped with the device or provided as an update by the manufacturer.

User experience test was different, since the study was conducted by allowing participants to evaluate a native application and a web application on two equal devices. Hence, for user experience test we used Samsung Galaxy S and Google Nexus S (manufactured by Samsung). These devices are very similar with practically same hardware specifications and software versions.

Component	**Model specification**
CPU	1 GHz ARM Cortex A8
GPU	200 MHz PowerVR SGX 540
Memory	512 MB RAM
Display	4.0” 800x480 pixels (233 ppi)

Table 5.2: Samsung Galaxy S GT-I9000

Component	**Model specification**
CPU	1 GHz ARM Cortex A8
GPU	200 MHz PowerVR SGX 530
Memory	512 MB RAM
Display	3.7” 480x854 pixels (265 ppi)

Table 5.3: Motorola Milestone 2

Component	**Model specification**
CPU	1 GHz ARM Cortex A5
GPU	Adreno 200
Memory	512 MB RAM
Display	4.0” 480x854 pixels (245 ppi)

Table 5.4: Sony Xperia J

Component	**Model specification**
CPU	Quad-core 1.4 GHz ARM Cortex-A9
GPU	Mali-400MP
Memory	1024 MB RAM
Display	4.8” 720x1280 pixels (306 ppi)

Table 5.5: Samsung Galaxy S III GT-I9300

5.2.2 Test applications

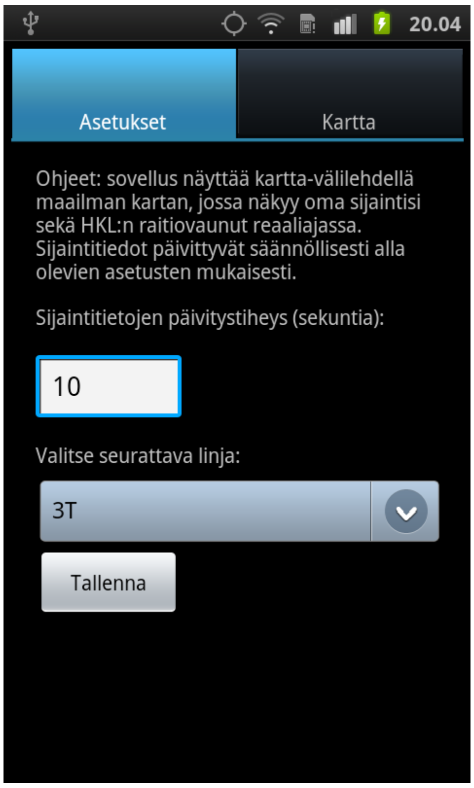
To evaluate the properties mentioned in the previous section, we built two separate test applications: a native Android application and a web application. The both applications implement similar functionalities and features, although there are differences in the appearance and the underlying implementation. For instance, in the web application the native Android UI components are replaced with corresponding HTML elements.

Both applications implement the following features:

- Displaying a world map
- Allowing the user to move around the map and zoom in and out
- Displaying the real-time location of the user on the map.
- Periodically calling a remote service to fetch and parse location data, and displaying the data on map.
- Settings dialog, comprising of ability to change update interval and filter the data that is fetched.

In practice, the applications implement a real-time tracking of trams in Helsinki. The trams are shown as points on a interactive map, which the user is able to move around and zoom in and out. The applications are making periodical HTTP requests to a web service that returns the real-time data in XML format. The data is provided by Helsinki Region Transport’s HSL Live API [43] – although the applications receive the data through a custom proxy. The applications also include a “settings”-view where the user can define the update interval for the tracking and choose a line to track.

In the native application the maps are displayed using Google Maps Android API. This is the standard and common way to implement a map view on Android applications. On the other hand, in the web application the map view is implemented using Leaflet[2] which is a mobile-friendly, open-source JavaScript library for displaying interactive maps. The reason for these choices was that both are common and popular way to implement an interactive map functionality for the corresponding platform. There are differences in the low-level implementations, but in our test cases, the solutions provide an equal functionality. The main language of the applications is Finnish, since all of the participants in the user experience study were native Finns.



Ohjeet: sovellus näyttää kartta-välilehdellä maailman kartan, jossa näkyy oma sijaintisi sekä HKL:n raitiovaunut reaaliajassa. Sijaintitiedot päivittyvät säännöllisesti alla olevien asetusten mukaisesti.

Sijaintitietojen päivitystiheys (sekuntia):

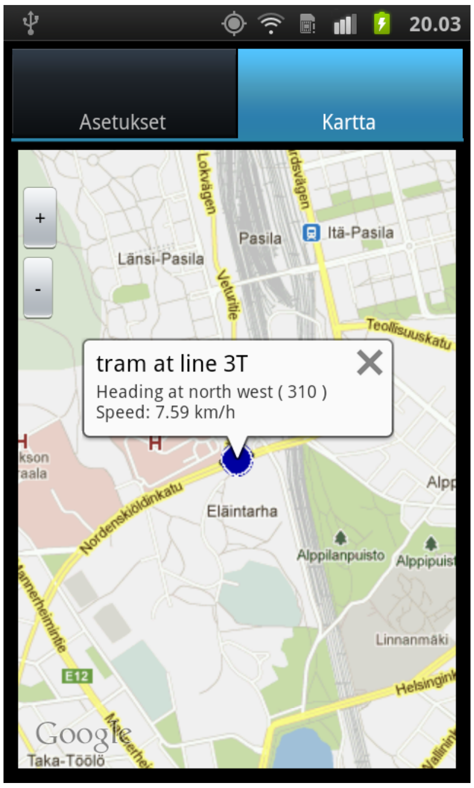
10

Valitse seurattava linja:

3T

Tallenna

Figure 5.1: Settings view of the native application.



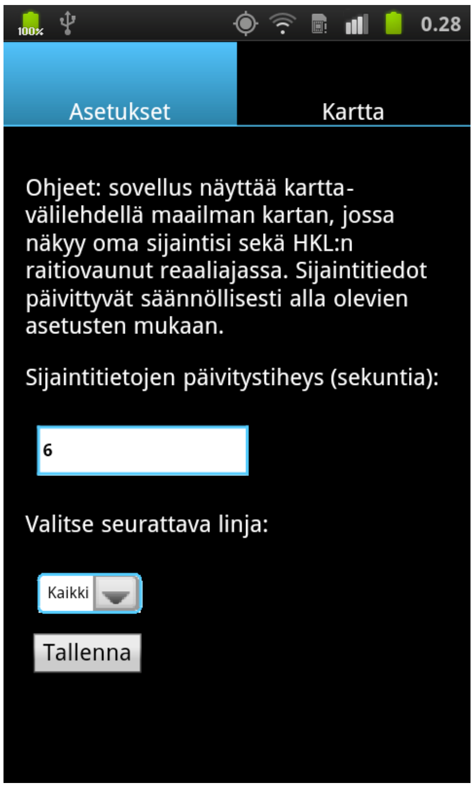
tram at line 3T
Heading at north west (310)
Speed: 7.59 km/h

Figure 5.2: Map view of the native application.

To make the appearance and the functionality of the applications as similar as possible, multi-touch zooming was disabled on the native application, since the feature was not supported in the web application either. Instead, zoom-in and zoom-out buttons were added to top left of the map area. Also, the layouts of both applications were designed and implemented so that they remind each other.

[2]http://leaflet.cloudmade.com

The web application is implemented using the common web technologies: HTML, CSS and JavaScript. Retrieving the location data from the remote service was implemented with asynchronous HTTP requests. From the features of HTML5, the web application utilizes Geolocation API for the positioning and Web Storage API for saving the settings. Also, the Leaflet map library utilizes several recent features of CSS3 and HTML5 such as animated zoom, transitions and hardware accelerated graphics rendering [2]. However, all of these are not supported in Android version 2.3.


Asetukset
Kartta

Ohjeet: sovellus näyttää kartta-välilehdellä maailman kartan, jossa näkyy oma sijaintisi sekä HKL:n raitiovaunut reaaliajassa. Sijaintitiedot päivittyvät säännöllisesti alla olevien asetusten mukaan.

Sijaintitietojen päivitystiheys (sekuntia): 6

Valitse seurattava linja: Kaikki

Tallenna

Figure 5.3: Settings view of the web application.

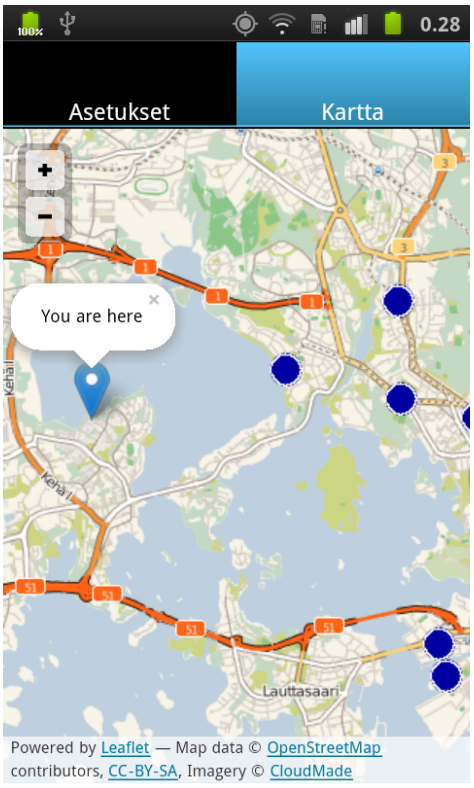

0.28
Asetukset
Kartta
You are here
Lauttasaari
Powered by Leaflet — Map data © OpenStreetMap contributors, CC-BY-SA, Imagery © CloudMade

Figure 5.4: Map view of the web application.

The web application can be executed either as a pure web application inside a browser, or as a hybrid application utilizing a web view component. When executed as a hybrid application, the application typically utilizes the platform's default browser engine to run the application code. Hence, in following tests, results for Android Browser can also be applied for the corresponding hybrid application.

5.2.3 Automated use case

Since energy consumption and memory usage were measured by collecting data on real-time while the test application was running on the device, it is important that circumstances and use cases are similar during each test case. To obtain reliable and comparable data, we had to make sure that during every use case, same number of map tiles are downloaded and stored in the memory, same amount of animations are displayed, graphics rendered and so on.

Hence, we implemented a simple automated use case, in which the map was controlled automatically through API of map view using pre-written coordinates and zoom levels. This use case was repeated every time the memory usage and energy consumption was measured. Same automated use case was implemented in both web application and native application. Running the automated use case takes about 40 seconds and it moves the map view around Helsinki district, zooming in and out couple of times.

5.3 User Experience test

To measure the overall usability and the user experience of our web application compared to the native one, we conducted a user experience study. The idea was to let a group of volunteers experiment the web application and the native application described in section 5.2.2. After the experimenting, the participants filled a form where they were asked to grade the certain features of the applications, including fluency, functionality and responsiveness.

5.3.1 Implementation

In the user experience test, two separate mobile devices were used. These were Samsung Galaxy S and Google Nexus S. Both devices were running Android 2.3 Gingerbread. To confirm the comparability of the devices, we experimented at the beginning that the native application was running on both devices equally fast without a notable difference in functionality, usability or performance.

The native application was installed and run in Samsung Galaxy S, whereas the web application was running on Google Nexus S. To make the applications appear as identical as possible, the web application was compiled to a hybrid application using PhoneGap and Android WebView.

In the test event, the participants were handed the devices with applications already running. They were instructed to test the both applications for couple of minutes, experiment how they work and compare the applications to each other. After this, the participants were asked to evaluate the following properties of the applications with a grade from 1 to 5.

1. Overall usability
2. Responsiveness
3. Speed and fluency
4. Overall functionality

The devices were marked with “A” (Google Nexus S running the web application) and “B” (Samsung Galaxy S running the native application). The participants evaluated the applications independently. However, in the grading, they were instructed to pay attention to differences of the two applications. Participants were also asked, which one of the applications they would prefer in their own device.

Totally 15 volunteers participated in the study. The average age of the participants was 23 years and 73% of the participants were men. 66% of the participants reported using a smartphone on a daily basis.

5.3.2 Results

The average grades for each property are presented in Figure 5.5.

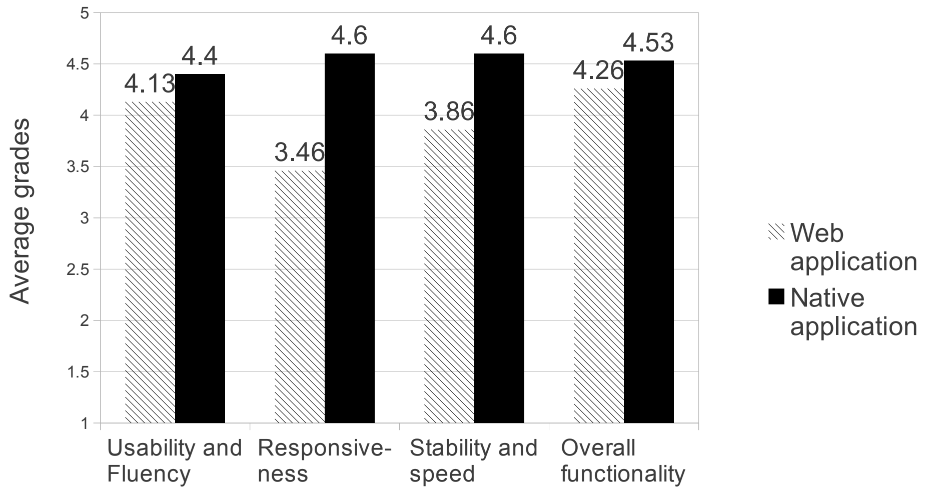


Figure 5.5: The results of the user experience study.

73% of the participants stated that application “B” (the native application) was better than application “A” (the web application). 13% preferred the web application whereas 13% could not find notable difference between the applications.

5.3.3 Evaluation

The number of participants was relatively small, but the results are clear and quite expected. The primary problems with the web application were the slow responsiveness to the user interaction, and the fluency and the speed of the user interface. In usability and overall functionality the web application performed quite well. That is, the web application is usable and provides the required functionality. However, the native application works more fluently and provides a better user experience.

To provide more reliable information, the user experience study should be organized using larger number of participants as well as several applications, use cases and mobile platforms. Nevertheless, Android is the most popular mobile platform, and Android 2.3 is still the most common Android version. Moreover, our test applications cover typical features of the mobile applications: interactivity, 2D graphics and the common UI elements. Hence, we suggest that figure 5.5 indicates how users experience the differences between web applications and native mobile applications in general.

5.4 Performance measurement

Whereas the user experience study was conducted to evaluate the UI performance, fluency and responsiveness, we also implemented a simple benchmarking to evaluate the raw computing performance. This may not be very relevant detail in an average mobile application, but illustrates the differences in performance between the native code and JavaScript.

5.4.1 Implementation

For this study we did not use our test applications that were described in section 5.2.2. Instead, we implemented a very simple application for benchmarking the pure computing speed, both for the native Android platform and the web. The application includes only a button that triggers an algorithm which generates a set of random strings. These random strings are used for MD5 hashing. After the function has finished, the application displays the total time consumed for the calculations. Essentially, this study evaluates the raw CPU speed and the implementation of virtual machine that executes the code.

```
function create_random_string(len) {
  var text = "";
  var chars = "0123456789abcdefghijklmnopqrstuvwxyz
  ABCDEFGHIJKLMNOPQRSTUVWXYZ+_-.,<>'/*-";
  for( var i=0; i < len; i++ )
  text += chars.charAt(Math.floor(Math.random() * chars.
     length));
  return text;
}

function benchmark() {
  d = new Date();
  n = d.getTime();
  for(i = 0; i<10000; i++) {
    hash = md5(create_random_string(12),null,1);
  }
  d = new Date();
  m = d.getTime();
  alert("time: " + (m-n));
}
```

Listing 5.1: The web implementation of the CPU benchmarking application in JavaScript

The benchmark-function of the web application generates 10 000 random strings with length of 12 characters. For each of them, a MD5 hash is calculated. After this, the total time of the process is displayed in milliseconds. Naturally, the web application also includes a button that invokes the benchmark-function. Since JavaScript does not include a native MD5-implementation, a freely available open-source implementation was used[3].

The relevant part of the code of the native test environment is presented in Listing 4.2.

```
btn.setOnClickListener(new View.OnClickListener() {
  public void onClick(View v) {
    long start = System.currentTimeMillis();
    String rnd = null;
    for(int i = 0; i<10000; i++) {
      rnd = createRandomString(12);
      byte[] bytesOfMessage = uuid.getBytes();
      try {
        MessageDigest md = MessageDigest.getInstance("
           MD5");
```

[3]http://pajhome.org.uk/crypt/md5/

```
            byte[] digest = md.digest(bytesOfMessage);
        } catch(NoSuchAlgorithmException e) { }
    }
    long end = System.currentTimeMillis();
    Toast.makeText(ctx, "Time: " + (end-start), Toast.
        LENGTH_SHORT).show();
  }
});
```

Listing 5.2: The native implementation of CPU benchmarking application in Java

The practical implementation of the measurement was quite straightforward. The test was executed ten times on each platform and the execution times were written down.

The benchmark tests were run in all of our four test devices. We evaluated both the native code and JavaScript code running in web browser. In order to obtain comprehensive results, the web benchmark was run on the following browsers: Chrome Mobile 18, Firefox Mobile 17, Opera Mobile 12 and the default Android Browser. Chrome is available only for Android 4.0 and later.

Also, to obtain supportive results we also ran SunSpider benchmark for each web browser on every device. SunSpider is a JavaScript benchmarking tool which measures the performance of JavaScript engines – not the browser APIs or operating with DOM [36]. The purpose of this test was to confirm that our MD5-hashing test is valid and correlates with another benchmark. SunSpider is actually much more comprehensive web browser benchmarking tool than our simple script, but the main focus of this test was to execute same algorithm both in native code and as a web application.

5.4.2 Results

The results are presented as bar charts below. Each chart covers one device. As excepted, in average the native application was faster than web applications. However, there seems to be major differences between the browsers. While on Android 2.3 devices the default browser is the slowest by far, Firefox is even faster than the native application. On the other hand, on newer Android devices the native application outperforms all web browsers. SunSpider results seem to correlate relatively well with MD5-hashing results. Naturally, since we observe the execution times of a certain task, the lower is better.

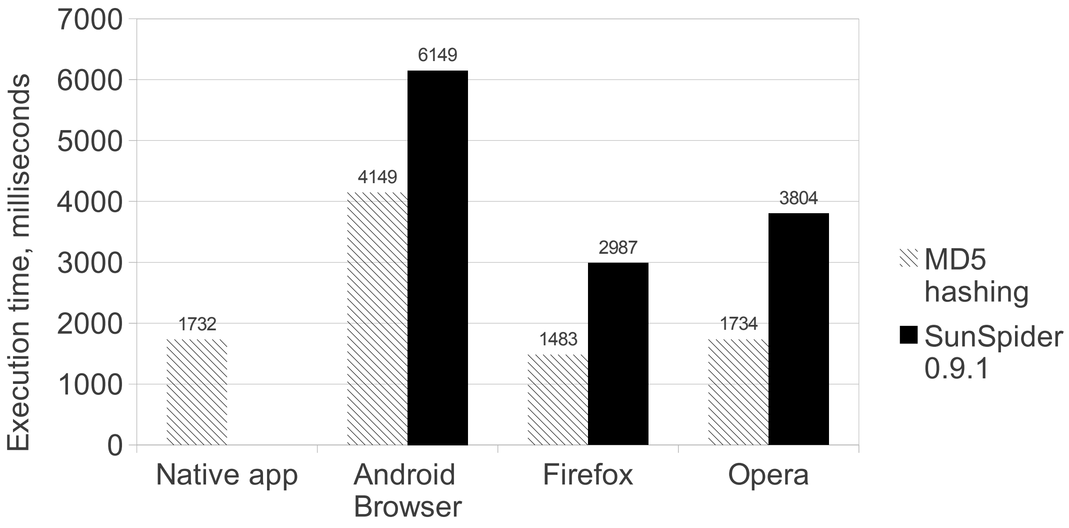


Figure 5.6: Performance test results: Samsung Galaxy S, Android 2.3.3

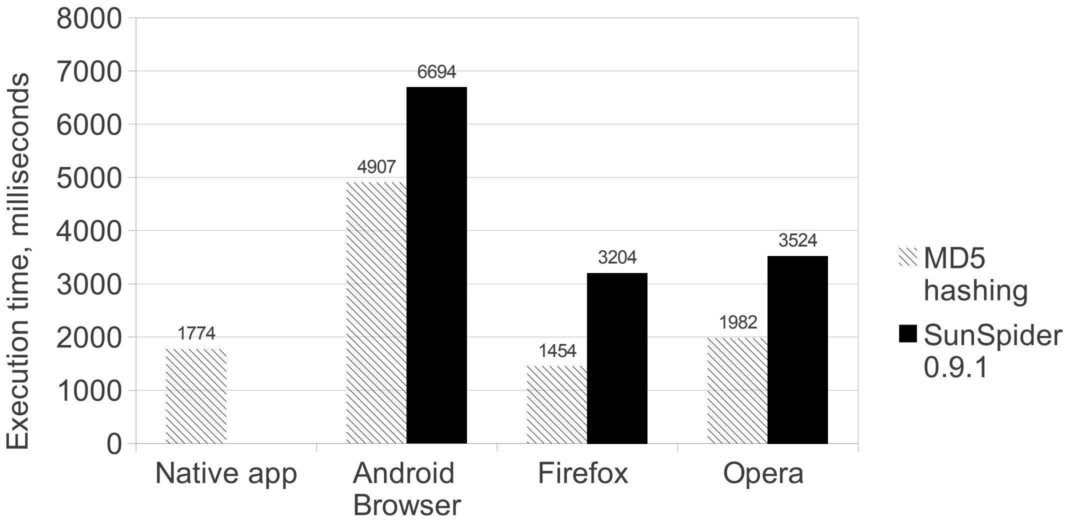


Figure 5.7: Performance test results: Motorola Milestone 2, Android 2.3.4

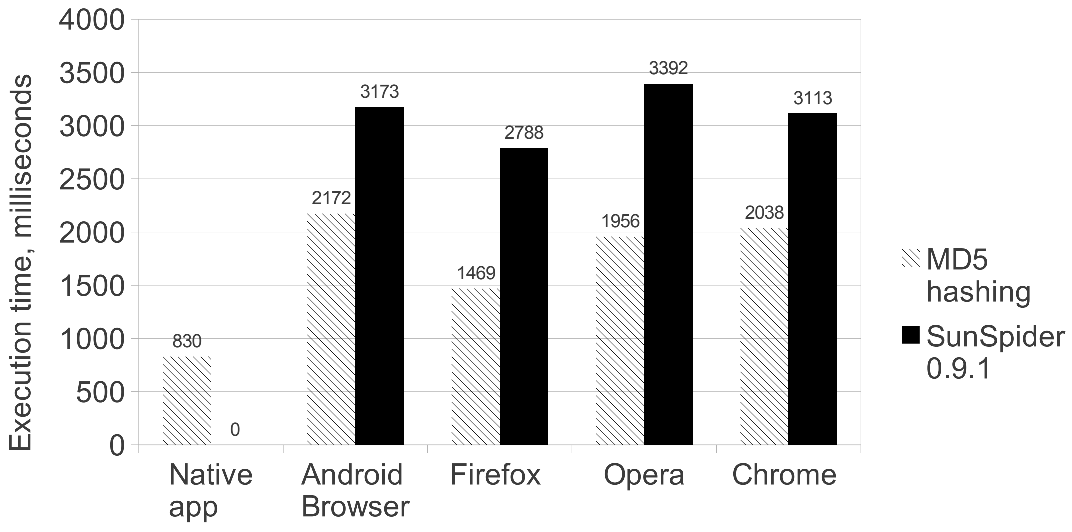


Figure 5.8: Performance test results: Sony Xperia J, Android 4.0.4

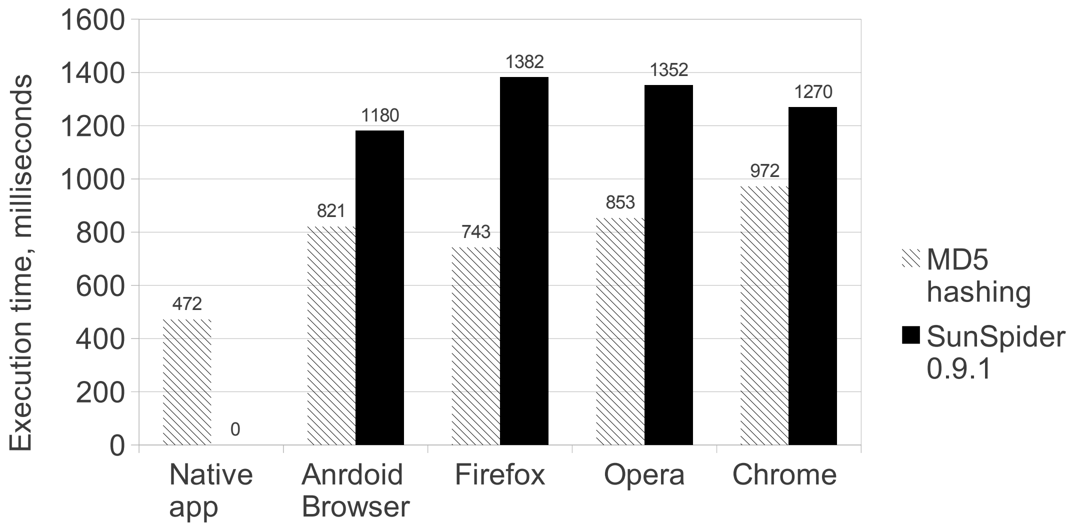


Figure 5.9: Performance test results: Samsung Galaxy S III, Android 4.1.1

5.4.3 Evaluation

Essentially, this study measures the performance of JavaScript engines compared to performance of DalvikVM and native Android applications. The results were quite much expected: interpreting JavaScript code has been much slower than running native code, but lately the performance of the JavaScript engines have been improved significantly. On the late Android versions, the default browser has closed the gap to other web browsers. The results of both MD5-hashing and SunSpider benchmark support this assumption, and they also correlate relatively well.

However, running an application inside Android's DalvikVM does not provide optimal performance either. For obtaining the best possible performance for a device running Android, applications should be written in pure C, compiled to binary code and executed directly under the Linux kernel [18].

It is worth noticing that the implementations of MD5 hashing algorithm may not be identical on the web application and the native Android application. However, we assume that *MessageDigest* class of Android SDK, provided by package *java.security* which was used on the native application, is well implemented and optimized as it is part of Android APIs. Hence, we assume that the native implementation is not at least considerably worse than the web implementation.

5.5 Memory usage

In this section we compare the memory footprint of a native Android application and corresponding web application. In this part of the study we use the test applications described in section 5.2.2.

5.5.1 Implementation

At first, it is important to notice that every Android process has two different memory heaps: native heap and VM heap. In practice, the difference is that native heap is allocated using C's *malloc*() function, whereas VM heap is handled by Dalvik VM and allocated with Java's *new* keywords.

Moreover, every application has so called "shared memory" and "private dirty" memory. Parts of the memory is shared across the running applications. In practice, this area of memory may include common classes and routines that every application needs for running, but that are usually not written or modified. On the other hand, "private dirty" is the amount of memory that can not be paged to disk and that is not shared with any other process. In practice, "private dirty" is the amount of memory that will be freed once the application is terminated[4]. From our point of view, "private dirty" is the relevant value.

[4]http://stackoverflow.com/a/2299813

For tracking the memory usage, we used our test devices, applications and Android SDK tools. Each mobile device was attached to a computer that was reading the memory information with command "adb shell dumpsys pid" where "pid" is id of the process. Before the study, the devices were rebooted to make sure that the memory was cleared and all the user-land applications were terminated. After this, the application to be monitored was launched and the memory tracking was started. Automated use case was run for each application, and snapshots of memory usage were taken every three seconds with a following command line script (Bash):

```
for i in {1..15} ; do adb shell dumpsys meminfo PID >>
   FILE_TO_BE_WRITTEN; sleep 3 ; done
```

The same test was organized separately on both for the native application, and the web application running on multiple web browsers. Since the web application is always executed inside a web browser, we measured the memory footprints of browser processes.

5.5.2 Results

The results for each device are illustrated in the tables below. Graphs illustrate the total size of "private dirty" memory of each process. The memory usage was monitored since the application was started until the automated test case was finished. Total length of each test case was 42 seconds. During each test case, 15 memory snapshots were taken.

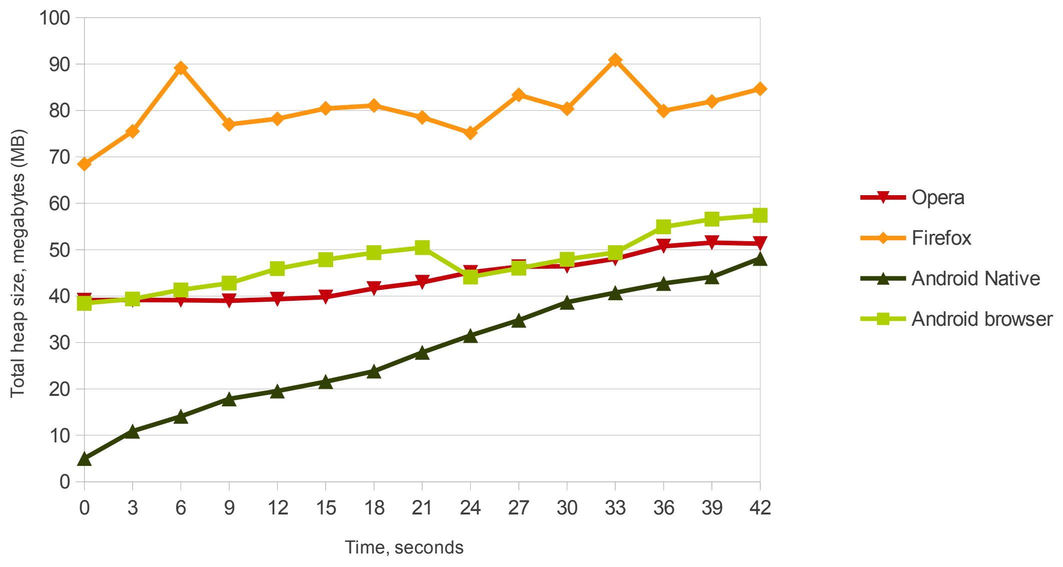


Figure 5.10: Memory usage: Samsung Galaxy S, Android 2.3.3

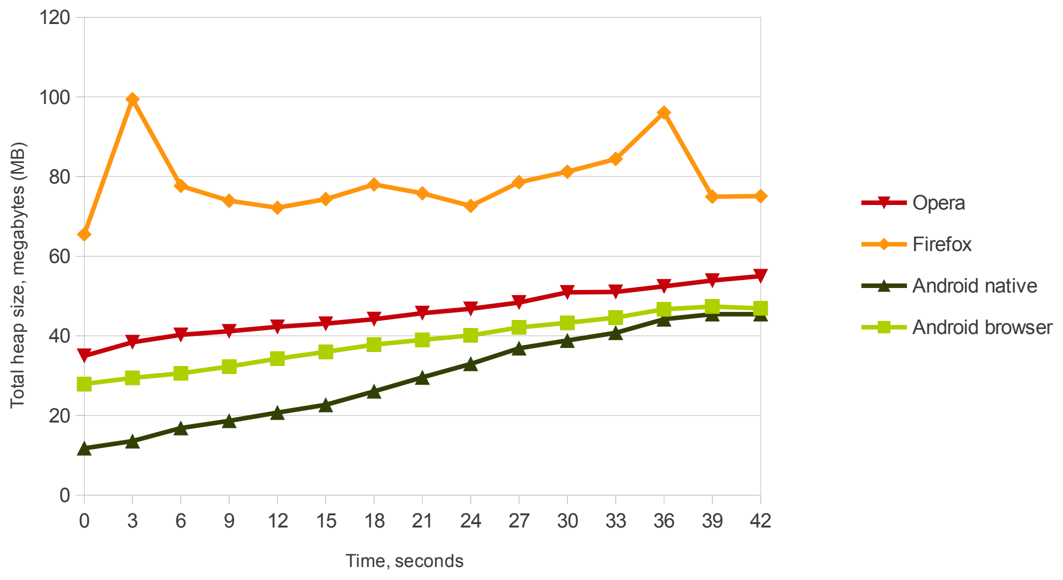


Figure 5.11: Memory usage: Motorola Milestone 2, Android 2.3.4

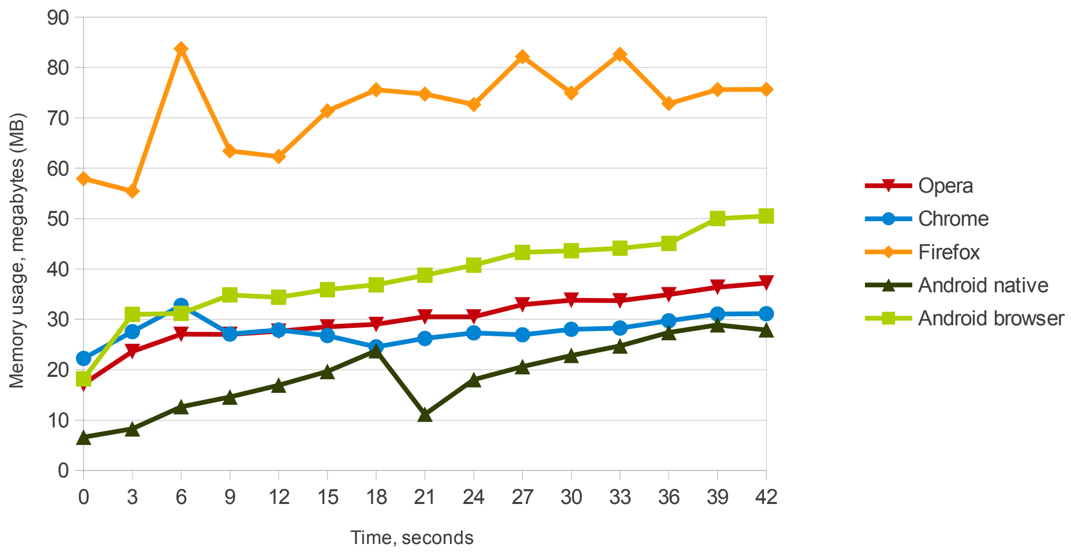


Figure 5.12: Memory usage: Sony Xperia J, Android 4.0.4

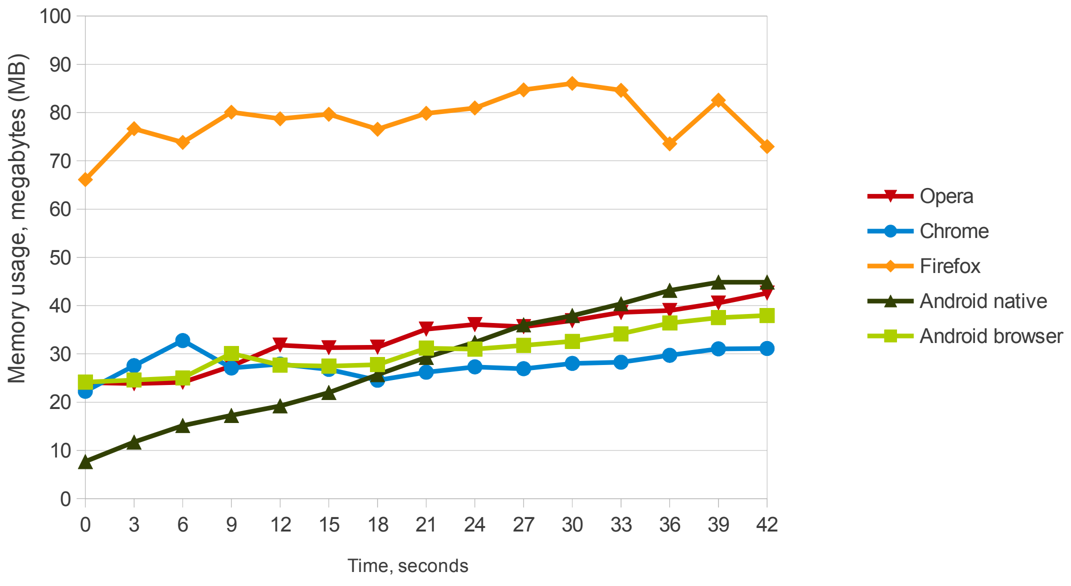


Figure 5.13: Memory usage: Samsung Galaxy S III, Android 4.1.1

5.5.3 Evaluation

It is easy to notice that on every device and every platform, the average memory usage of the native application is lower than any browser. However, we need to remember that we are comparing the memory footprint of the native application to memory footprint of web browsers – not only the web application.

Current browsers are complicated and heavy systems that may require tens of megabytes memory for just loading and starting. Hence, we can not draw a conclusion that web applications consume much more memory than native application. Actually, it seems that the higher is memory footprint in the applications, the lower is the difference between the native application and web browsers. Only exception is Firefox, which has almost twice larger memory footprint than rest of the browsers.

Overall, we need to take into account that we measured the memory consumption using only a one type of application. It seems that loading and processing hundreds of bitmap tiles actually requires quite large amount of memory. Thus, web browsers and Dalvik VM may act in a completely different way when running a different kind of applications.

5.6 Energy consumption

Since battery life is an important factor on mobile devices, we arranged a study to evaluate and compare the energy consumption on web and native applications.

5.6.1 Implementation

The energy consumption was measured using a specific power monitor by Monsoon Solutions Inc. [28] The power monitor was connected to the terminals of the device battery, labeled as positive (+) and negative (-) by copper tape and the DC leads.

After this, the power monitor was connected to a computer and turned on. The power consumption of the device was measured using Power Tool by Monsoon Solutions Inc. Once the hardware setup was ready, we started to record the power usage and ran several test cases on the mobile device. In these test cases we used our test applications described in section 5.2.2. Hence, the test application displayed an interactive map with overlay data that was periodically updated over a network connection.

During the power measurements, the test applications were run separately in the following environments:

- Motorola Milestone 2 and Samsung Galaxy S (Android 2.3): native application, Android default browser, Opera Mobile 12, Firefox Mobile 16

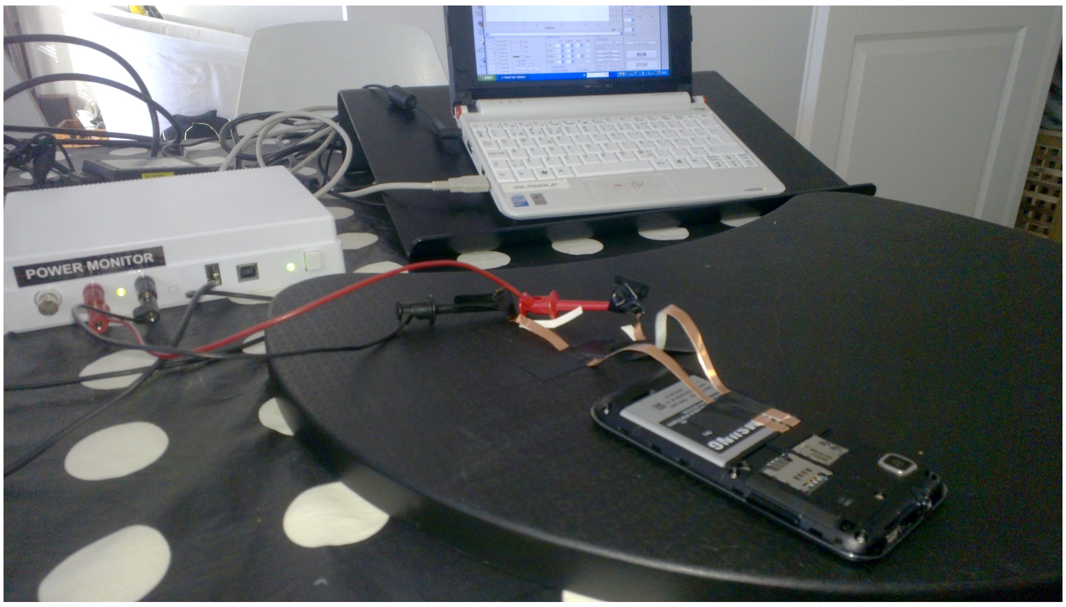


Figure 5.14: Power monitor connected to Samsung Galaxy S battery.

- Sony Xperia J (Android 4.0) and Samsung Galaxy S III (Android 4.1): native application, Android default browser, Chrome Mobile 18, Opera Mobile 12, Firefox Mobile 17

On each environment, the power usage of applications was measured both in idle state and during the automated use case described in section 5.2.3. The duration of the measurements were between 20 (idle) and 45 seconds (running). Each test case was run at least twice to verify the reliability of the results.

The devices were connected to network via WiFi and cellular network (UMTS). Since the type of network connection significantly affects the power consumption, the results are presented and analyzed separately. When WiFi connection was used, cellular network was disabled and vice versa.

The applications utilized GPS for positioning. Automatic screen brightness adjustment was disabled, as well as other power saving-saving features. In the each test case, the update interval of the data obtained from a remote service, was set to 10 seconds. To provide an equal starting point for each test case, the mobile device was rebooted before every power measurement.

5.6.2 Results

Average results, with usage and idle state combined, are presented as bar charts in the figures 5.15 and 5.16.

Exact results are presented in numerical form in the tables 5.6, 5.7, 5.9 and 5.8. The graphs of raw measurement data can be found in the appendix list of this thesis.

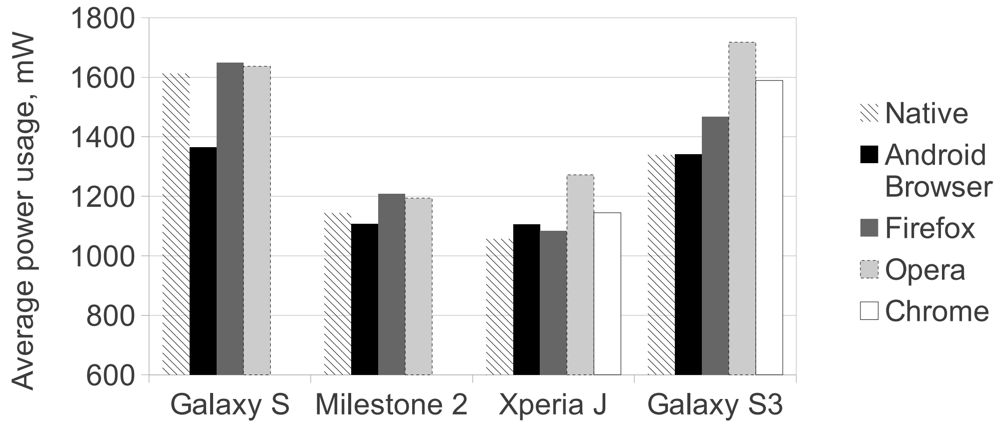


Figure 5.15: Differences of power usage across the different environments using WiFi as Internet connection.

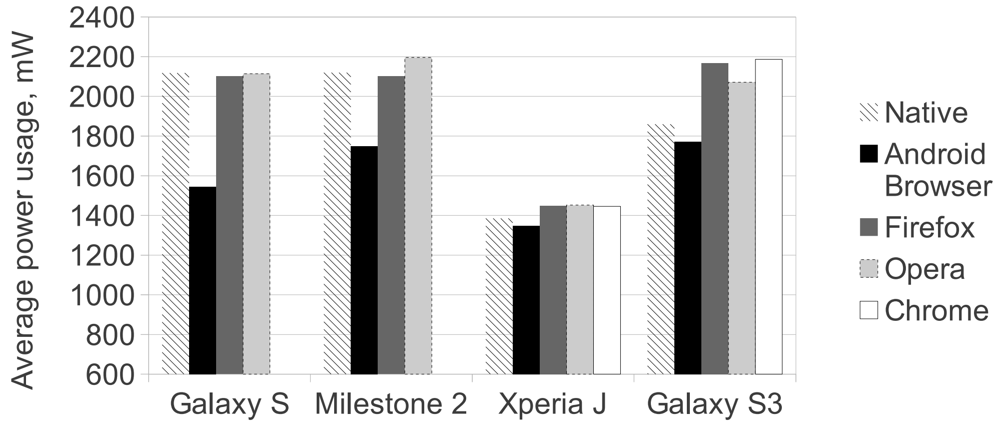


Figure 5.16: Differences of power usage across the different environments using cellular network as Internet connection.

Environment	**Network**	**Idle**	**Running**	**Average**
Native application	WiFi	1151mW	2076mW	1613mW
	3G	1485mW	2751mW	2118mW
Android Browser	WiFi	1119mW	1611mW	1365mW
	3G	1032mW	2059mW	1545mW
Firefox	WiFi	1258mW	2040mW	1649mW
	3G	1661mW	2541mW	2101mW
Opera	WiFi	1512mW	1766mW	1639mW
	3G	1914mW	2315mW	2114mW

Table 5.6: The results of the power measurement test on Samsung Galaxy S, Android 2.3.3

Environment	**Network**	**Idle**	**Running**	**Average**
Native application	WiFi	716mW	1573mW	1144mW
	3G	1592mW	2648mW	2120mW
Android Browser	WiFi	840mW	1376mW	1108mW
	3G	1269mW	2229mW	1749mW
Firefox	WiFi	834mW	1584mW	1209mW
	3G	1626mW	2575mW	2101mW
Opera	WiFi	913mW	1481mW	1197mW
	3G	2018mW	2375mW	2196mW

Table 5.7: The results of the power measurement test on Motorola Milestone 2, Android 2.3.4

Environment	**Network**	**Idle**	**Running**	**Average**
Native application	WiFi	848mW	1267mW	1057mW
	3G	1195mW	1575mW	1385mW
Android Browser	WiFi	836mW	1375mW	1105mW
	3G	1070mW	1628mW	1349mW
Firefox	WiFi	861mW	1308mW	1084mW
	3G	1261mW	1640mW	1450mW
Opera	WiFi	1138mW	1407mW	1272mW
	3G	1273mW	1632mW	1452mW
Chrome	WiFi	867mW	1424mW	1145mW
	3G	1143mW	1749mW	1446mW

Table 5.8: The results of the power measurement test on Sony Xperia J, Android 4.0.4

Environment	**Network**	**Idle**	**Running**	**Average**
Native application	WiFi	1103mW	1575mW	1339mW
	3G	1552mW	2169mW	1860mW
Android Browser	WiFi	1110mW	1573mW	1341mW
	3G	1551mW	1992mW	1771mW
Firefox	WiFi	1139mW	1795mW	1467mW
	3G	1636mW	2698mW	2167mW
Opera	WiFi	1324mW	2112mW	1718mW
	3G	1657mW	2485mW	2071mW
Chrome	WiFi	1096mW	2083mW	1589mW
	3G	1515mW	2841mW	2178mW

Table 5.9: The results of the power measurement test on Samsung Galaxy S III, Android 4.1.1

5.6.3 Evaluation

We assume that the results correlate quite well with the reality. Results across the different platforms and devices support each other. Also, the results were obtained using both cellular network and WiFi. However, as well as on our other test cases, the main problem is that we evaluate only a one type of mobile application. Thus, we can not draw too hasty generalizations.

However, the overall trend is rather clear. On Android 2.3 the default browser seems to be clearly the most energy-efficient platform for running mobile applications. On newer devices and Android versions the advantage of Android Browsers seems to be much lower. However, on average it still performs better than any other browsers and does not lose for the Dalvik VM either.

Again, we need to take into account the differences between the native application and the web application. The implementations are not exactly identical. They use different libraries and mechanisms for displaying the maps and controlling the user interface. Also, the map tiles are served from different servers. The native application downloads the tiles from Google's map service, whereas the web application uses CloudMade[5] tile servers. This may cause differences in latency and average file sizes, which in turn, may influence the network usage and the energy consumption.

[5]http://cloudmade.com/

Chapter 6

Discussion and Analysis

This chapter briefly sums up and evaluates the results and findings of the thesis. Also, the advantages and the disadvantages of the web and the native technologies are evaluated and compared to each other.

6.1 Sum up and analysis of results

At first, it is important to notice that our study covers only Android platform – no other common mobile platforms, such as iOS or Windows Phone at all. Thus, there may be differences across the platforms and the devices. However, since Android is the most popular mobile platform, we suggest that the results are relevant and correlate with the overall trend. Hence, we propose that in general the following findings are true.

- The native technologies provide better user experience. That is, native applications work faster and more fluently, resulting in better user interaction and responsiveness.
- Energy consumption on web applications depends on the browser environment the application is executed. In case of Android, it seems that web applications running on Android default browser provides the lowest power consumption on average. However, difference to native environment is rather small.
- The total memory usage in web applications is usually higher than in native applications, since the whole web browser need to be started to run web applications. However, once the web browser is started, running web applications does not increase the total memory footprint that much. Thus, if the web browser is running anyway, it may be

even more memory-efficient to run web application instead of native application.

- The native technologies provide a higher computing performance. This is natural, since JavaScript is a dynamic and weakly-typed language, and native code is usually compiled and and optimized for the corresponding platform.

6.1.1 User Experience

The user experience study indicates the flaws of web applications in practice. As noticed by many users and bloggers [16, 22], native applications tend to work smoother and faster than web applications. This is natural and logic, and opposite result would have been a surprise. Although the native application performed clearly better than the web application in the user experience test, the difference was relatively small. As expected, the native application took advantage over the web application in responsiveness, speed and fluency.

While the web application performed relatively well in user experience test, the test applications did not include many technically demanding or computationally heavy features. Thus, the advantage for native technologies could be much more significant for different kind of applications. Hence, we conclude that with careful implementation certain type of web applications may provide sufficient user experience and performance. However, developers need to pay attention especially in responsiveness and fluency of the user interface.

6.1.2 Performance

In contrast to the user experience test, results of the performance measurements are much more complicated. While the performance of Android 2.3 default browser is very poor compared to any other browser or platform, the late web browsers Opera 12 and Firefox 17 are able to match to the performance of the native benchmark on Android 2.3. On the other hand, on Android 4.0 and Android 4.1 platforms, the native benchmark clearly outperforms all the web browsers. Also, on the newer Android versions, the default browser has caught up the other web browsers. Hence, during the last two years the performance of JavaScript engines has improved, but so is the performance of Android's Dalvik VM.

This can be easily seen when comparing Galaxy S (Android 2.3) and Xperia J (Android 4.0) to each other. Samsung Galaxy S includes 1GHz

Cortex A8 processor, while Xperia J includes 1GHz Cortex A5. Processor of Galaxy S should be slightly faster, but according to the benchmark results on Firefox and Opera, the devices are quite close to each other. However, in the native benchmark Xperia J is even twice as fast as Galaxy S. This implicates, that performance of Dalvik VM has improved a lot.

From our test devices, Samsung Galaxy S III is in its own class. Galaxy S III includes a 1.4GHz quad-core processor and even the performance of web benchmark matches to performance of native benchmark on any other device. However, it does not utilizes all the cores during the benchmark. When monitoring CPU usage, it executes all the tests using just a single core. Nevertheless, it is still two times faster than any other device. Also, Galaxy S III runs Android 4.1, which may affect to the difference.

One interesting detail is the success of Firefox, which was clearly the fastest browser in our MD5-hashing test. According to SunSpider, which evaluates many different properties of the JavaScript engine, in cryptographic operations Firefox is not any faster than other browsers. However, SunSpider results show that the only part where Firefox dominates is string processing. In our MD5-hashing test, every browser generates 10,000 of random strings with length of 12 characters. Since Firefox was very fast in string processing, this must be the part where it makes the difference to other browsers.

6.1.3 Memory usage

As explained in the section 5.5.1, memory management of Android is quite complicated and monitoring the amount of memory consumed by an application is not straightforward. Every application has two different memory heaps - a native heap and a Dalvik heap - as well as private memory and shared memory separately. We monitored the size of “private dirty” memory which essentially represents the amount of memory that will be freed, once the application is terminated.

Also, when evaluating amount of memory consumed by the web application, we needed to monitor memory usage of the browsers, since the web application is run inside a web browser. When a web browser is started, a lot of data is loaded to memory, including browser plugins and runtime environment. Thus, it is natural that initially the web application allocates a large amount of memory.

Generally, the results show that starting the native application consumes much less memory than starting a web browser and navigating to the web application. However, once the automated use case is being executed, memory usage of the native application is increasing more rapidly than memory usage of the web application. On Samsung Galaxy S III, three of the four web browsers resulted in lower memory footprint than the native application. On the other devices, the native application resulted in the lowest power consumption, but the difference to web browsers was relatively small at the end. Firefox is a special case: it produced almost a twice larger memory footprint that the other browsers of the native application.

It seems that while the web application needs more resources to be loaded and started, the native application consumes more memory when application is being used. One possible reason for this might be garbage collection. For the native application, garbage collection is implemented in Dalvik VM. On the other hand, on the web application garbage collection is implemented both in JavaScript and lower level, such as Dalvik VM or native code. This may restrict the memory usage of the application during the execution.

For sure, running a lightweight native application results in a much lower memory consumption than opening a heavy web browser and running a web application. However, if the web browser is running anyway on background, opening a web application may increase the total memory usage of the system less than opening a native application.

6.1.4 Power consumption

In the power consumption test we measured the total power usage of each device on idle state and during the automated test case, while both WiFi and cellular network were separately used as Internet connection.

A bit surprisingly, the default browser of Android performed the best on average. On the devices running Android 2.3, Android Browser was almost dominant, while on Xperia J and Galaxy S III the difference to the native application was very small. However, no other browsers were more energy-efficient than the native application.

There are a few of factors that can explain the differences in power consumption across browsers and platforms: CPU / GPU usage and network usage. Of course, different devices have different CPUs, screen sizes and other hardware, but we can omit these, since we focus on the differences between the web application and the native application – not the differences between the devices.

For instance, the default browser of Android 2.3 does not support GPU acceleration for graphics rendering. Thus, it does not utilize GPU at all, while the native application does. This may partially explain the success of Android Browser on the older Android platform. Another essential factor is CPU usage. During the automated test case, the Android 2.3 native application and Firefox utilized around 60–80% of CPU whereas Android Browser and Opera kept the CPU utilization around 30%.

On the newer platforms, Android 4.0 and Android 4.1, the native application and every browser supported GPU acceleration for CSS transforms. Also, the results between the browsers and the native applications are fairly even compared to Android 2.3. The only test case where the native application performed better than than the web application on Android Browser, was Xperia J using WiFi as Internet connection. On the other hand, when the device was connected to Internet via cellular network, Android Browser was the most energy-efficient platform also on Xperia J. On average, once the WiFi connection was switched to UMTS network, the power consumption of the native application increased higher percentage than the power consumption of the web application.

This is interesting, since a network monitor showed that the web application actually downloaded more data from network for during the automated test. There are minor differences between the devices – mainly because of different screen sizes and resolutions – but generally, the applications require between 3 to 4 megabytes (MB) of data during the 40 second test case. However, it seems that the web application downloads 10-20% more data than the native application – even though we exclude the data the web application needs to download during the startup.

However, when we compare the two map servers – Google's map server and CloudMade tile server that the web application uses – it seems that latency to CloudMade server is significantly lower. While 56 byte ICMP ping (round-trip time) to Google's map server is almost 40 milliseconds, round-trip time to CloudMade tile server is only 10-15 milliseconds. Hence, the native application needs to wait each response from the map service much longer than the web application. Thus, the native application requires the device to keep network connection active longer time.

6.2 Pros and Cons

In this section we evaluate the advantages and disadvantages of the different technologies and solutions in mobile application development. Table 6.1 illustrates the differences between the web and the native technologies. It also shows how the hybrid solutions compare to other technologies. The table illustrates that the advantages of the each technology depends heavily on the usage purpose of the application.

In the Table 6.1 the gray cells refer to a disadvantage, whereas the white cells refer to an advantage. The center cells indicate how the hybrid solutions compare to the web and the native solutions. There are couple of features in web applications which may be considered both as an advantage and a

Web technologies	**Hybrid**	**Native technologies**
Limited performance		Optimal performance
Lower user experience		Optimal user experience
Functionality limited to the features that are supported by web browsers and HTML5		Full access to device APIs and sensors
No deployment via application markets, need for custom monetizing strategies.		Easy monetizing and distribution via application markets.
Applications available online, network connection required		Applications available for offline use
Lower development costs		Higher development costs
Rapid development using the common web technologies		Platform specific development tools
Immediate world wide distribution over the web		Platform specific packaging and distribution
Applications can be distributed across all platforms with small or no modifications		Applications must be implemented separately for every platform
Applications always up-to-date and available as web services		Applications must be installed and updated manually for every device

Table 6.1: Strenghts and weaknesses of each application development technology.

disadvantage depending on the context. For instance, the fact that web applications are available as web services is an advantage, since the users do not need to manually download and install a certain application, and the application is available whenever needed on any device. On the other hand, this may also restrict the usage of the application, since the network connection is required. The application may be unavailable in extreme conditions when network connection is not available. Also, a temporary break in the network connections may bring the application down.

In the following subsections we analyze the advantages and disadvantages of each technology in more detail. Also, we evaluate, which technology provides the optimal solution in different use cases. Commonly, there is no correct answers, since every application has its own functional and non-functional requirements.

6.2.1 Native technologies

In the most platforms, native technologies are common and official way to produce applications. The mobile platform vendors, such as Apple, Google and Microsoft, recommend that the native technologies are used for mobile application development. As our results show, the native technologies provide the best performance, user experience and usability. In addition, the users are used to download and install the native applications from application markets. Hence, the application markets are the best way to advertise and distribute applications.

The problems with the native technologies are the fragmentation and incompatibility of the mobile platforms. Because applications must be implemented separately for every platforms, this may result in high development costs and limited coverage of mobile platforms.

However, the native technologies have several strengths. The native technologies are the best choice for the projects that require high performance, optimal user experience or hardware accelerated graphics. Moreover, if an application is meant to be available only for a certain platform, the native technologies provide the best result.

6.2.2 Web technologies

The main advantage of the web technologies is their compatibility across the platforms. Basically, web applications can be run on any platform that includes a modern web browser. This results in fast development and deployment of the applications practically on any platform.

However, web applications are limited both in the usability and the functionality. For instance, many web browsers are not able to handle the advanced functionalities, such as 3D-graphics, or provide access to certain device sensors. Web applications are used primarily as online services, which may be either an advantage or a disadvantage depending on the use case. Web applications can not be advertised or monetized via application markets. However, web applications are available anywhere and any time via network connections.

We recommend the web technologies to be used for applications that require high portability and fast online deployment. Web applications can also be used for viewing basic 2D-graphics and providing interactivity, as our test application shows. However, for high performance and optimal user experience, the alternative technologies should be considered.

6.2.3 Hybrid solutions

Hybrid solutions are somewhere between web and native. That is, hybrid applications may include parts or components implemented using the native technologies, whereas the main application may still be written using the web technologies. This approach is useful, when an application requires functionalities that are not available by using pure web technologies. A significant amount of the application code can still be recycled and utilized on multiple platforms, which reduces the amount of the work.

The hybrid approach can also be used for packaging a web application for a certain platform and distributing it through official channels, such as Apple's App Store. Nevertheless, the hybrid solutions always draw some kind of trade-off. For instance, native components of a hybrid application must be implemented separately for each platform. Hence, the more native components are used, the higher is the effort needed for the cross-platform deployment. Moreover, the performance and the usability of hybrid applications are still dependent on the capabilities of the web browser.

Chapter 7

Conclusions

It is clear, that the best technology for implementing a certain application depends on several factors: objectives, business model, target audience, technical requirements [29], non-functional requirements and budget. Overall, based on our studies and the other findings, we propose that the main advantages provided by the web technologies are fast development and deployment of the application, and compatibility across the platforms. On the other hand, the native technologies provide optimal performance, which in turn results in better user experience and interactivity. There are also minor differences between the web and the native technologies in the resource consumption, but their importance is rather minimal when it comes to business or user experience.

To obtain more comprehensive results, the experimental research should be extended to cover several platforms and different types of applications. However, the research results of this thesis cover the most common mobile platform and several different devices. Thus, the results are relevant and provide us a good indication on how web technologies compare to native technologies in average mobile application.

A potential success factor for mobile web applications might be the upcoming features of HTML5 that are not yet widely deployed, such as WebGL and WebRTC. However, their suitability and performance on mobile is still unclear. Overall, we suggest that once the performance and the technical capabilities of the web technologies are close to the native technologies, the web technologies provide a very useful set of tools for replacing the native technologies almost in any use case.

While evaluating the future of mobile applications and their evolution, there are several alternative paths. Either the web or the native technologies may dominate the markets, or they will be used along with each other depending on the situation. Also, hybrid applications have several strengths and they might become an important player in the mobile application field.

We suggest that the future of the web technologies in mobile application development looks promising. The web technologies already provide a viable alternative for the native technologies. Moreover, we expect that the breakthrough of the advanced web technologies, such as HTML5, will emerge the adaption of the web technologies. However, we do not believe that the native applications will disappear from the market. Since the large companies, including Apple, Google and Nokia, are rapidly building the ecosystems around their products, the native APIs and the platform specific features will become increasingly important to differentiate from the others. In addition, the users are used to find and download their applications from the platform specific application markets hosted by the companies. Hence, we expect that the usage of the web technologies will increase in the near future, but they will not completely supersede the native technologies.

The main challenge of web technologies is user experience. While web technologies may provide sufficient functionality and meet the technical requirements of most mobile applications, they are still struggling with slow and lagging user interfaces. This is a problem especially in mobile field, where users expect nice and smooth interaction and immediate response. Moreover, platform vendors may not be willing to address this problem, since they want to keep their SDKs and APIs as primary software development technology.

Bibliography

[1] Shruthi Adappa, Vikas Agarwal, Sunil Goyal, Ponnurangam Kumaraguru, and Sumit Mittal. User controllable security and privacy for mobile mashups. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, HotMobile '11, pages 35–40, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0649-2. doi: 10.1145/2184489.2184498. URL `http://doi.acm.org/10.1145/2184489.2184498`.

[2] Vladimir Agafonkin and CloudMade team. Leaflet Featuers. URL `http://leaflet.cloudmade.com/features.html`. [Online; received 3 Oct 2012].

[3] Android Open Source Project. Philosophy and Goals. URL `http://source.android.com/about/philosophy.html`. [Online; received 8 Sep 2012].

[4] Mohsen Anvaari and Slinger Jansen. Evaluating architectural openness in mobile software platforms. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ECSA '10, pages 85–92, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0179-4. doi: 10.1145/1842752.1842775. URL `http://doi.acm.org/10.1145/1842752.1842775`.

[5] Apple. About the iOS Technologies. URL `http://developer.apple.com/library/ios/#Documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html`. [Online; received 30 Sep 2012].

[6] Andre Charland and Brian Leroux. Mobile application development: web vs. native. *Commun. ACM*, 54(5):49–53, May 2011. ISSN 0001-0782. doi: 10.1145/1941487.1941504. URL `http://doi.acm.org/10.1145/1941487.1941504`.

[7] Mark Doernhoefer. JavaScript. *SIGSOFT Softw. Eng. Notes*, 31(4): 16–24, July 2006. ISSN 0163-5948. doi: 10.1145/1142958.1142972. URL `http://doi.acm.org/10.1145/1142958.1142972`.

[8] F-Secure. Mobile Threat Report, Q2 2012.

[9] N. Gandhewar and R. Sheikh. Google Android: An emerging software platform for mobile devices. *International Journal on Computer Science and Engineering(IJCSE)*, pages 12–17, 2010.

[10] Jesse James Garrett. Ajax: A new approach to web applications, 2005. URL `http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications`.

[11] Gartner Inc. Market Share: Mobile Devices, Worldwide, 2Q12. *Gartner Research Report*, 2012. URL `http://www.gartner.com/it/page.jsp?id=2120015`.

[12] M.H. Goadrich and M.P. Rogers. Smart smartphone development: iOS versus Android. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 607–612. ACM, 2011.

[13] Hewlett-Packard New release. HP to Contribute webOS to Open Source. URL `http://www.hp.com/hpinfo/newsroom/press/2011/111209xa.html`. [Online; received 18 Sep 2012].

[14] Internet Engineering Task Force (IETF). HTTP State Management Mechanism, RFC 2965, . URL `http://www.ietf.org/rfc/rfc2965.txt`.

[15] Internet Engineering Task Force (IETF). The WebSocket Protocol, RFC 6455, . URL `http://tools.ietf.org/html/rfc6455`.

[16] Adrian Jones. Native, Hybrid or Web Apps? URL `http://buildmobile.com/native-hybrid-or-web-apps/`. [Online; received 28 Nov 2012].

[17] Khronos Group. WebGL - OpenGL ES 2.0 for the Web. URL `http://www.khronos.org/webgl/`. [Online; received 1 Oct 2012].

[18] Sangchul Lee and Jae Wook Jeon. Evaluating performance of Android platform using native C for embedded systems. In *Control Automation and Systems (ICCAS), 2010 International Conference on*, pages 1160 –1163, oct. 2010.

[19] Zhijie Lin, Jiyi Wu, Qifei Zhang, and Hong Zhou. Research on Web Applications Using Ajax New Technologies. In *MultiMedia and Information Technology, 2008. MMIT '08. International Conference on*, pages 139 –142, dec. 2008. doi: 10.1109/MMIT.2008.107.

[20] Peter Lubbers and Frank Greco. HTML5 Web Sockets: A Quantum Leap in Scalability for the Web. `http://www.websocket.org/quantum.html`. [Online; received 25 Sep 2012].

[21] Tongbo Luo, Hao Hao, Wenliang Du, Yifei Wang, and Heng Yin. Attacks on WebView in the Android system. In *Proceedings of the 27th Annual Computer Securityleaflet html5 canvas Applications Conference*, ACSAC '11, pages 343–352, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0672-0. doi: 10.1145/2076732.2076781. URL `http://doi.acm.org/10.1145/2076732.2076781`.

[22] Michael Mahemoff. HTML5 vs Native: The mobile app debate. URL `http://www.html5rocks.com/en/mobile/nativedebate/`. [Online; received 28 Nov 2012].

[23] Ryan Matzner. Why Web Apps Will Crush Native Apps. URL `http://mashable.com/2012/09/12/web-vs-native-apps/`. [Online; received 28 Nov 2012].

[24] Microsoft Developer Network. HTML and DHTML Reference. URL `http://msdn.microsoft.com/en-us/library/ms533050.aspx`. [Online; received 4 Sep 2012].

[25] T. Mikkonen and A. Taivalsaari. Reports of the Web's Death Are Greatly Exaggerated. *Computer*, 44(5):30 –36, may 2011. ISSN 0018-9162. doi: 10.1109/MC.2011.127.

[26] T. Mikkonen and A. Taivalsaari. Apps vs. Open Web: The Battle of the Decade. In *Proceedings of the 2nd Workshop on Software Engineering for Mobile Application Development (MSE'2011, Santa Monica, California, USA*, pages 22–26, 2011.

[27] Tommi Mikkonen and Antero Taivalsaari. Using JavaScript as a real programming language. Technical report, Mountain View, CA, USA, 2007.

[28] Monsoon Solutions Inc. Power Monitor. URL `http://www.msoon.com/LabEquipment/PowerMonitor/`. [Online; received 18 Sep 2012].

[29] JT Mudge. Native App vs. Mobile Web App: A Quick Comparison. URL `http://sixrevisions.com/mobile/native-app-vs-mobile-web-app-comparison/`. [Online; received 28 Oct 2012].

[30] S. Murugesan. Understanding Web 2.0. *IT Professional*, 9(4):34 –41, july-aug. 2007. ISSN 1520-9202. doi: 10.1109/MITP.2007.78.

[31] Microsoft Developer Network. Application Certification Requirements for Windows Phone. URL `http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh184843(v=vs.92)`. [Online; received 4 Sep 2012].

[32] Gerard Nicolas, Karim Sbata, and Elie Najm. Websocket enabler: achieving IMS and web services end-to-end convergence. In *Proceedings of the 5th International Conference on Principles, Systems and Applications of IP Telecommunications*, IPTcomm '11, pages 3:1–3:3, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0975-2. doi: 10.1145/2124436.2124441. URL `http://doi.acm.org/10.1145/2124436.2124441`.

[33] OpenSignalMaps. Android Fragmentation Visualized: the many faces of a green robot. *OpenSignalMaps Signal Reports*, May 2012. URL `http://opensignalmaps.com/reports/fragmentation.php`. [Online; received 27 Jul 2012].

[34] Daniel Pavlic, Mile Pavlic, and Vladan Jovanovic. Future of Internet technologies. In *MIPRO, 2012 Proceedings of the 35th International Convention*, pages 1366 –1371, May 2012.

[35] V. Pimentel and B.G. Nickerson. Communicating and Displaying Real-Time Data with WebSocket. *Internet Computing, IEEE*, 16(4):45 –53, July-Aug. 2012. ISSN 1089-7801. doi: 10.1109/MIC.2012.64.

[36] The WebKit Open Source Project. SunSpider JavaScript Benchmark. URL `http://www.webkit.org/perf/sunspider/sunspider.html`. [Online; received 7 Oct 2012].

[37] A.K. Saha. A Developer's First Look At Android. *LinuxForYou, Issue*, 5(11):48–50, 2008.

[38] Ben Savage. Why HTML5 provided more tricks than treats in 2012, November 2012. URL `http://venturebeat.com/2012/11/24/html5-more-tricks-treats-2012/`. [Online; received 7 Dec 2012.

[39] David Smith. iOS Version Stats, December 2012. URL `http://david-smith.org/iosversionstats/`. [Online; received 11 Dec 2012.

[40] Spaceport.io. Spaceport PerfMarks Report II. Technical report, May 2012. URL `http://spaceport.io/spaceport_perfmarks_2_report_2012_5.pdf`. [Online; received 7 Dec 2012.

[41] A. Taivalsaari, T. Mikkonen, D. Ingalls, and K. Palacz. Web Browser as an Application Platform. In *Software Engineering and Advanced Applications, 2008. SEAA '08. 34th Euromicro Conference*, pages 293 –302, sept. 2008. doi: 10.1109/SEAA.2008.17.

[42] A. Taivalsaari, T. Mikkonen, M. Anttonen, and A. Salminen. The Death of Binary Software: End User Software Moves to the Web. In *Creating, Connecting and Collaborating through Computing (C5), 2011 Ninth International Conference on*, pages 17 –23, jan. 2011. doi: 10.1109/C5.2011.9.

[43] Helsinki Region Transports. Other APIs, Reittiopas. URL `http://developer.reittiopas.fi/pages/en/other-apis.php`. [Online; received 2 Oct 2012].

[44] Guanhua Wang. Improving Data Transmission in Web Applications via the Translation between XML and JSON. In *Communications and Mobile Computing (CMC), 2011 Third International Conference on*, pages 182 –185, april 2011. doi: 10.1109/CMC.2011.25.

[45] William West and S. Monisha Pulimood. Analysis of privacy and security in HTML5 web storage. *J. Comput. Sci. Coll.*, 27(3):80–87, January 2012. ISSN 1937-4771. URL `http://dl.acm.org/citation.cfm?id=2038772.2038791`.

[46] Word Wide Web Consortium W3C. Document Object Model (DOM). . URL `http://www.w3.org/DOM/`. [Online; received 4 Sep 2012].

[47] Word Wide Web Consortium W3C. Web SQL Database. . URL `http://www.w3.org/TR/webdatabase/`. [Online; received 11 Dec 2012].

[48] World Wide Web Consortium. HTML 4.01 Specification. URL `http://www.w3.org/TR/1999/REC-html401-19991224/`. [Online; received 15 Aug 2012].

[49] World Wide Web Consortium W3C. HTML5 differences from HTML4, . URL `http://www.w3.org/TR/html5-diff/`. [Online; received 15 Aug 2012].

[50] World Wide Web Consortium W3C. WEB APPLICATIONS (WEBAPPS) WORKING GROUP, . URL `http://www.w3.org/2008/webapps/`. [Online; received 19 Sep 2012].

[51] World Wide Web Consortium W3C. Device APIs Working Group. . URL `http://www.w3.org/2009/dap/`. [Online; received 19 Sep 2012].

[52] World Wide Web Consortium W3C. Geolocation Working Group, . URL `http://www.w3.org/2008/geolocation/`. [Online; received 27 Aug 2012].

[53] World Wide Web Consortium W3C. DeviceOrientation Event Specification. . URL `http://www.w3.org/TR/orientation-event/`. [Online; received 19 Sep 2012].

[54] World Wide Web Consortium W3C. WebRTC 1.0: Real-time Communication Between Browsers. . URL `http://www.w3.org/TR/webrtc/`. [Online; received 4 Sep 2012].

[55] World Wide Web Consortium W3C. Web Real-Time Communications Working Group. . URL `http://www.w3.org/2011/04/webrtc/`. [Online; received 4 Sep 2012].

[56] World Wide Web Consortium W3C. The WebSocket API. . URL `http://dev.w3.org/html5/websockets/`. [Online; received 4 Sep 2012].

[57] World Wide Web Consortium W3C. Touch Events version 1, W3C Candidate Recommendation. . URL `http://www.w3.org/TR/touch-events/`. [Online; received 28 Sep 2012].

Appendix A

Web application source code

Source code for our web application used in the test cases. Leaflet library [1] is required, as well as the necessary resources, such as image files.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Master's Thesis test app</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-
        scalable=no">
    <meta charset="UTF-8" />
    <link rel="stylesheet" href="leaflet.css" />
    <script src="leaflet.js"></script>
    <link rel="stylesheet" href="style.css"/>
    <script src="app_header.js"></script>
  </head>
  <body>
    <div id="bar" >
      <div id="btn1" class="active" onclick="show_settings();">
        <br/><br/>Asetukset
      </div>
      <div id="btn2" onclick="show_map();">
        <br/><br/>Kartta
      </div>
    </div>
    <br/><br/><br/>
    <div id="settings" style="display: block;">
      <p>
      Ohjeet: sovellus n&auml;ytt&auml;&auml; kartta-v&auml;lilehdell&auml; maailman kartan, jossa n&
          auml;kyy oma sijaintisi sek&auml; HKL:n raitiovaunut reaaliajassa. Sijaintitiedot p&auml;
          ivittyv&auml;t s&auml;&auml;nn&ouml;llisesti alla olevien asetusten mukaan.
      </p>
      <p>Sijaintitietojen p&auml;ivitystiheys (sekuntia):</p>
      <input type="number" value="10" id="updateInterval" style="color: #000; font-family: Verdana;
          font-weight: bold; font-size: 12px; background-color: #fff;" />
      <p>Valitse seurattava linja:</p>
      <select id="lines">
        <option value="Kaikki">Kaikki</option>
        <option value="3T">3T</option>
        <option value="3B">3B</option>
        <option value="4">4</option>
        <option value="7">7</option>
        <option value="10">10</option>
      </select>
      <br/><br/>
      <button id="save" onclick="save_settings();">Tallenna</button>
    </div>
    <div id="mapcontainer" style="display: none;">
      <div id="map"></div>
    </div>
    <script src="app.js"></script>
  </body>
```

[1] http://leaflet.cloudmade.com

```
</html>
```

Listing A.1: index.html: Web application HTML source code

```
var map = L.map('map');
L.tileLayer('http://{s}.tile.cloudmade.com/<set-your-id-here>/256/{z}/{x}/{y}.png', {
  maxZoom:18,
  attribution: 'Map data &copy; <a href="http://openstreetmap.org">OpenStreetMap</a> contributors, <a
        href="http://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>, Imagery Â© <a href="http
       ://cloudmade.com">CloudMade</a>'
}).addTo(map);

var busIcon = L.icon({
  iconUrl: 'bus.png',
  iconSize:     [24,24], // size of the icon
  iconAnchor:   [12,12], // point of the icon which will correspond to marker's location
  popupAnchor:  [0,-12] // point from which the popup should open relative to the iconAnchor
});

map.setView([60.195, 24.913], 13);
window.mylocation = new L.LayerGroup();
window.poigroup = new L.LayerGroup();
var xmlhttp;
xmlhttp=new XMLHttpRequest();

xmlhttp.onreadystatechange=function() {
  if (xmlhttp.readyState==4 && xmlhttp.status==200) {
    window.poigroup.clearLayers();
    map.removeLayer(window.poigroup);
    xmlDoc=xmlhttp.responseXML;
    x = xmlDoc.getElementsByTagName("vehicle");
    for(i = 0; i < x.length; i++) {
      type = x[i].childNodes[2].textContent;
      line = x[i].childNodes[4].textContent;
      lng = x[i].childNodes[6].textContent;
      lat = x[i].childNodes[8].textContent;
      heading = x[i].childNodes[10].textContent;
      heading_str = "";
      if(heading >= 337 || heading <= 22) heading_str = "north";
      else if(heading >= 293 && heading <= 336) heading_str = "north west";
      else if(heading >= 247 && heading <= 292) heading_str = "west";
      else if(heading >= 202 && heading < 247) heading_str = "south west";
      else if(heading >= 157 && heading < 202) heading_str = "south";
      else if(heading >= 112 && heading < 157) heading_str = "south east";
      else if(heading >= 67 && heading < 112) heading_str = "east";
      else heading_str = "north east";
      L.marker([lat,lng], {icon: busIcon}).addTo(window.poigroup)
      .bindPopup("Line: " + line + " (" + type + ") <br/> Heading " + heading_str).openPopup();
    }
    map.addLayer(window.poigroup);
  }
}

/* Read Web Storage and set values accordingly */
updateInterval = localStorage.getItem("updateInterval");
if(!updateInterval) {
  updateInterval = 10;
}
lines = localStorage.getItem("lines");
if(!lines) {
  lines = "Kaikki";
}

linesList = document.getElementById("lines");
i = 0;
for(i = 0; i< linesList.length; i++) {
  if(linesList.options[i].value == lines) {
    break;
  }
}

linesList.options[i].selected = true;
document.getElementById("updateInterval").value=updateInterval;

/* Async HTTP requests depending on settings */
if(lines != "Kaikki") {
  xmlhttp.open("GET","http://<api-url>/index.php?lines="+lines+"&"+Math.random(),true);
}
else {
  xmlhttp.open("GET","http://<api-url>/index.php?"+Math.random(),true);
}
xmlhttp.send();

window.interval = setInterval(function(){
  if(lines != "Kaikki") {
    xmlhttp.open("GET","http://<api-url>/index.php?lines="+lines+"&"+Math.random(),true);
  }
  else {
    xmlhttp.open("GET","http://<api-url>/index.php?"+Math.random(),true);
  }
  xmlhttp.send();

},updateInterval*1000);


function save_settings() {
  /* Stop timer and read settings */
  clearInterval(window.interval);
  var new_interval = document.getElementById("updateInterval").value;
  if(new_interval < 5) new_interval = 5;
  if(new_interval > 60) new_interval = 60; // force the update interval between 5-60 sec

  /* Save settings */
  localStorage.setItem("updateInterval", new_interval);
  localStorage.setItem("lines", document.getElementById("lines").value);

  /* Set up new update timer */
  window.interval = setInterval(function(){
    params = "";
    if(document.getElementById("lines").value != "Kaikki") {
      params = "?lines="+document.getElementById("lines").value + "&" + Math.random();
    } else {
      params = "?"+Math.random();
    }
    xmlhttp.open("GET","http://<api-url>/index.php"+params ,true);
      xmlhttp.send();

  }, new_interval*1000);
  alert("Asetukset tallennettu");
}

function show_settings() {
  document.getElementById("mapcontainer").setAttribute("style", "display: none;");
  document.getElementById("settings").setAttribute("style", "display: block;");
  document.getElementById("btn1").setAttribute("class", "active");
  document.getElementById("btn2").setAttribute("class", "");

}

function show_map() {
  document.getElementById("mapcontainer").setAttribute("style", "display: block;");
  document.getElementById("settings").setAttribute("style", "display: none;");
  document.getElementById("btn1").setAttribute("class", "");
  document.getElementById("btn2").setAttribute("class", "active");
  setTimeout("map.invalidateSize();", 300);
}
```

Listing A.2: app.js: web application JavaScript source code

```css
input, select   {
  height: 28px;
  font: 200 11px Arial, Helvetica, sans-serif;
  margin: 8px;
  padding-right: 0px 0px 6px 2px;
  border: 2px #5cf solid;
}
* { /* disable android default tap-highlight */
  -webkit-tap-highlight-color: rgba(0, 0, 0, 0);
}
body {
  padding: 0;
  margin: 0;
  background-color: #000;
}
html, body, #map {
  width: 100%;;
  height: 100%;
  padding-top:0px;

}
div#bar {
  color: white;
  z-index: 999; background: black; position: fixed;
  top: 0px; left: 0px; display: block; width: 100%; text-align: center; height: 58px;
  overflow: hidden;
  font-family: "Helvetica Neue", "Arial";
}
div#settings {
  color: white;
  background: black;
  font-family: "Helvetica Neue", "Arial";
  position: absolute;
  left: 0px;
  right: 0px;
  bottom: 0px;
  top: 59px;
  padding: 15px;
}

div#mapcontainer {
  position: absolute;
  left: 0px;
  right: 0px;
  bottom: 0px;
  top: 59px;
}
div.active {
  background: -webkit-gradient(linear, left top, left bottom, from(#52c3fc), to(#2c83a8));
}
div#btn1 {
  width: 50%; overflow: auto;border-bottom: 2px solid #5cf; float: left;
}
div#btn2 {
  width: 50%; overflow: auto; border-bottom: 2px solid #5cf;  float: right;
}
```

Listing A.3: style.css: web application stylesheet

Appendix B

Android application source code

This appendix includes only the program code, to compile and run the application, Android MapView Balloons[1] is required as well as the resource files.

```java
/* This file is a part of TramTracker for Android
 * Copyright (C) 2012, Jussi-Pekka Erkkilä
 * For copying, please apply the MIT license
 */
import android.util.Log;
import android.app.TabActivity;
import android.os.Bundle;
import android.widget.TabHost;
import android.content.res.Resources;
import android.content.Intent;

public class MainTabActivity extends TabActivity
{
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    Resources res = getResources(); // Resource object to get Drawables
    TabHost tabHost = getTabHost();
    TabHost.TabSpec spec;
    Intent intent;

    // Create an Intent to launch an Activity for the tab (to be reused)
    intent = new Intent().setClass(this, SettingsActivity.class);
    spec = tabHost.newTabSpec("hw").setIndicator("Asetukset")
      .setContent(intent);
    tabHost.addTab(spec);
    intent = new Intent().setClass(this, TramTrackerMap.class);
    spec = tabHost.newTabSpec("nwt").setIndicator("Kartta")
      .setContent(intent);
    tabHost.addTab(spec);
    tabHost.setCurrentTab(0); // load settings first
  }
}
```

Listing B.1: MainTabActivity.java: Android application main activity

[1]https://github.com/jgilfelt/android-mapviewballoons

```java
/* This file is a part of TramTracker for Android
 * Copyright (C) 2012, Jussi-Pekka Erkkilä
 * For copying, please apply the MIT license
 */
import java.util.List;
import java.util.Timer;
import java.util.TimerTask;
import android.os.Handler;

import android.util.Log;
import android.os.AsyncTask;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.view.View.OnTouchListener;
import android.view.View;
import android.view.MotionEvent;
import android.widget.Button;

import org.apache.http.client.HttpClient;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.HttpResponse;
import org.apache.http.HttpEntity;

import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.NamedNodeMap;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import android.os.Bundle;
import android.graphics.drawable.Drawable;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapView;
import com.google.android.maps.MyLocationOverlay;
import com.google.android.maps.MapController;
import com.google.android.maps.Overlay;
import com.google.android.maps.GeoPoint;
import com.google.android.maps.OverlayItem;


public class TramTrackerMap extends MapActivity {

  private MyLocationOverlay myLocationOverlay;
  private MapView mapView;
  private MapController mapController;
  private List<Overlay> mapOverlays;

  private SimpleItemizedOverlay itemizedOverlay;

  // Settings are loaded in these
  private String apiUrl;
  private String followRoute;
  private int updateInterval;
  private boolean useMock;
  public static final String PREFS_NAME = "TramTrackerPreferences";

  @Override
  protected boolean isRouteDisplayed() {
    return false;
  }

  protected void onResume() {
    super.onResume();
    Log.w("TramTrackerMap", "Resuming");
    this.getPrefs();
  }

  private void getPrefs() {
    apiUrl = "http://<api-url>/";
    SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
    updateInterval = settings.getInt("updateInterval", 10);
    followRoute = settings.getString("followRoute", "Kaikki");
    if(!followRoute.equals("Kaikki"))
      apiUrl = apiUrl + "?lines="+followRoute;
    Log.v("ASDFDASF", "ApiUrl is now " + apiUrl);

  }
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Init mapview component
    setContentView(R.layout.mapview);
    mapView = (MapView) findViewById(R.id.mapview);
    mapView.setBuiltInZoomControls(false);

    mapView.setOnTouchListener(new OnTouchListener() {
      public boolean onTouch(View v, MotionEvent event) {
        if(event.getPointerCount() > 1) {
          return true;
        }
        return false;
      }
    });
    final Button zoomin_button = (Button) findViewById(R.id.zoomin);
    zoomin_button.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        Log.v("MapViewController", "zoomin");
        mapController.zoomIn();
      }
    });

    final Button zoomout_button = (Button) findViewById(R.id.zoomout);
    zoomout_button.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        Log.v("MapViewController", "zoomotu");
        mapController.zoomOut();
      }
    });

    mapController = mapView.getController();
    initMap();
    this.getPrefs(); // Load settings

    // Setup map overlays, start asynchronous http calls
    mapOverlays = mapView.getOverlays();
    Drawable drawable = getResources().getDrawable(R.drawable.bus);
    itemizedOverlay = new SimpleItemizedOverlay(drawable, mapView);
    mapOverlays.add(itemizedOverlay);
    toCallAsynchronous();
  }

  private void initMap() {
    myLocationOverlay = new MyLocationOverlay(this, mapView);
    mapView.getOverlays().add(myLocationOverlay);
    myLocationOverlay.enableMyLocation();
    myLocationOverlay.runOnFirstFix(new Runnable() {
      public void run() {
        mapController.animateTo(myLocationOverlay.getMyLocation());
      }
    });
  }

  // Start automatically repeating non-blocking background task
  public void toCallAsynchronous() {
    TimerTask doAsynchronousTask;
    final Handler handler = new Handler();
    Timer timer = new Timer();
    doAsynchronousTask = new TimerTask() {
      @Override
      public void run() {
        handler.post(new Runnable() {
          public void run() {
            AsyncHttpRequest req = new AsyncHttpRequest();
            try {
              req.execute( apiUrl);
            } catch (Exception e) {
            }

          }
        });
      }
    };
    // Don't allow less than 5 sec update interval
    timer.schedule(doAsynchronousTask, 0,Math.max(updateInterval * 1000, 5000));
  }

  // Non-blockng http request
  private class AsyncHttpRequest extends AsyncTask<String, String, Document> {
    protected Document doInBackground(String... params) {
      // Init http client and make request to url
      HttpClient httpclient = new DefaultHttpClient();
      HttpGet httpget = new HttpGet(params[0]);
```

```java
      HttpResponse response;

      // Init DOM document
      DocumentBuilderFactory factory;
      DocumentBuilder builder;
      Document doc;
      try {
        factory = DocumentBuilderFactory.newInstance();
        builder = factory.newDocumentBuilder();
        response = httpclient.execute(httpget);

        // Get response content
        doc = builder.parse(response.getEntity().getContent());
        return doc;

      } catch (Exception e) { // dump error message
        return null;
      }
    }

    // Once we've got some response, the data is
    // parsed and markers placed on map.
    protected void onPostExecute(Document doc) {
      itemizedOverlay.clear();
      if(doc != null) {
        // Get XML elements
        NodeList vehicles = doc.getElementsByTagName("vehicle");
        OverlayItem overlayItem;
        GeoPoint point;
        Node tmpNode;
        double lat = 0, lng = 0;
        int heading = 0;
        String headingStr = "";
        NodeList vehicleData;
        String line = "", type ="", speed ="";
        for(int i = 0; i < vehicles.getLength(); i++) {
          vehicleData = vehicles.item(i).getChildNodes();
          line = "Unknown";
          lat = 0; lng = 0;
          type = "Tram";
          speed = "";

          for(int j = 0; j < vehicleData.getLength(); j++) {
            tmpNode = vehicleData.item(j);
            if(tmpNode.getNodeName().equals("lat"))
              lat = Double.parseDouble(tmpNode.getFirstChild().getNodeValue());
            if(tmpNode.getNodeName().equals("lng"))
              lng = Double.parseDouble(tmpNode.getFirstChild().getNodeValue());
            if(tmpNode.getNodeName().equals("line"))
              line = tmpNode.getFirstChild().getNodeValue();
            if(tmpNode.getNodeName().equals("line"))
              line = tmpNode.getFirstChild().getNodeValue();
            if(tmpNode.getNodeName().equals("type"))
              type = tmpNode.getFirstChild().getNodeValue();
            if(tmpNode.getNodeName().equals("heading"))
              heading = Integer.parseInt(tmpNode.getFirstChild().getNodeValue());
            if(tmpNode.getNodeName().equals("speed"))
              speed = tmpNode.getFirstChild().getNodeValue();
          }
          point = new GeoPoint((int)(lat*1E6),(int)(lng*1E6));
          if(heading >= 337 || heading <= 22) headingStr = "north";
          else if(heading >= 293 && heading <= 336) headingStr = "north west";
          else if(heading >= 247 && heading <= 292) headingStr = "west";
          else if(heading >= 202 && heading < 247) headingStr = "south west";
          else if(heading >= 157 && heading < 202) headingStr = "south";
          else if(heading >= 112 && heading < 157) headingStr = "south east";
          else if(heading >= 67 && heading < 112) headingStr = "east";
          else headingStr = "north east";
          if(speed.length() > 0)
            headingStr = headingStr + " ( " + heading + " )\nSpeed: " + speed + " km/h";

          overlayItem = new OverlayItem(point, type + " at line " + line,
              "Heading at " + headingStr);
          itemizedOverlay.addOverlay(overlayItem);
        }
        mapView.invalidate();
      }
    }
  }
}
```

Listing B.2: TramTrackerMap.java: Android application map view

```java
/* This file is a part of TramTracker for Android
 * Copyright (C) 2012, Jussi-Pekka Erkkilä
 * For copying, please apply the MIT license
 */
import android.os.Bundle;

import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.app.Activity;
import android.widget.TextView;
import android.widget.CheckBox;
import android.widget.Spinner;
import android.widget.ArrayAdapter;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.AdapterView;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.app.AlertDialog;
import android.app.AlertDialog.Builder;


import android.app.Activity;
public class SettingsActivity extends Activity {
  private Button saveButton;
  private EditText apiUrl;
  private EditText updateInterval;
  private CheckBox useMock;
  private String toFollow;

  public static final String PREFS_NAME = "TramTrackerPreferences";

  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Read settings widgets from the layout file
    setContentView(R.layout.settingsview);
    saveButton = (Button)findViewById(R.id.btn);
    updateInterval = (EditText)findViewById(R.id.update);


    // Read settings from the device storage
    SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
    updateInterval.setText(""+settings.getInt("updateInterval", 10));

    // Initialize "choose route to follow" -dropdown menu
    Spinner spinner = (Spinner) findViewById(R.id.trackline);
    ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource
      (this, R.array.linearray, android.R.layout.simple_spinner_item);
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    spinner.setAdapter(adapter);
    // Set spinner listener
    spinner.setOnItemSelectedListener(new OnItemSelectedListener() {
        public void onItemSelected(AdapterView<?> parent,
            View view, int pos, long id) {
          toFollow = parent.getItemAtPosition(pos).toString();
        }
        public void onNothingSelected(AdapterView parent) {
          toFollow = "Kaikki";
         }
    });

    // Set spinner value from phone's storage
    ArrayAdapter myAdap = (ArrayAdapter) spinner.getAdapter();
    int spinnerPosition = myAdap.getPosition(settings.getString("followRoute", "Kaikki"));
    spinner.setSelection(spinnerPosition);

    // Savebutton listener
    saveButton.setOnClickListener( new View.OnClickListener() {
        public void onClick(View view) {
          int interval;
          try {
            interval = Math.max(5, Integer.parseInt(updateInterval.getText().toString())); // Min
                 update interval forced to 5 secs

            // All objects are from android.context.Context
            SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
            SharedPreferences.Editor editor = settings.edit();
            editor.putInt("updateInterval", interval);
            editor.putString("followRoute", toFollow);
            editor.commit();
            AlertDialog.Builder dialog = new AlertDialog.Builder(SettingsActivity.this);
            dialog.setTitle("Settings saved");
            dialog.setMessage("The changes will take effect once you switch to the map view.");
            dialog.show();
          } catch (Exception e) {
            Log.v("SettingsActivity", "Saving the settings failed: " + e.toString());

          }
        }
    });

  }
}
```

Listing B.3: SettingsActivity.java: Android application settings view

Appendix C

Example input data from service

This is an example of XML data that the test applications receive and parse from the remote service.

```xml
<?xml version="1.0"?>
<locationdata>
  <vehicle>
    <id>RHKL00092</id>
    <type>tram</type>
    <line>10</line>
    <lng>24.906529</lng>
    <lat>60.193526</lat>
    <heading>305</heading>
  </vehicle>
  <vehicle>
    <id>RHKL00090</id>
    <type>tram</type>
    <line>3T</line>
    <lng>24.9349</lng>
    <lat>60.168629</lat>
    <heading>79</heading>
  </vehicle>
  <vehicle>
    <id>353567040316870</id>
    <type>bus</type>
    <line>4</line>
    <lng>24.91868</lng>
    <lat>60.187983</lat>
    <heading>331</heading>
    </vehicle><vehicle>
    <id>RHKL00095</id>
    <type>tram</type>
    <line>4</line>
    <lng>24.925151</lng>
    <lat>60.183213</lat>
    <heading>143</heading>
  </vehicle>
  <vehicle>
    <id>RHKL00210</id>
    <type>tram</type>
    <line>9</line>
    <lng>24.939487</lng>
    <lat>60.200497</lat>
    <heading>348</heading>
  </vehicle>
  <vehicle>
    <id>RHKL00213</id>
    <type>tram</type>
    <line>9</line>
    <lng>24.921471</lng>
    <lat>60.159597</lat>
    <heading>352</heading>
  </vehicle>
</locationdata>
```

Listing C.1: Example of XML data received by the client

Appendix D

Power consumption graphs

On every figure, blue graph illustrates the power consumption during use case, while black curve shows the power usage on idle state.

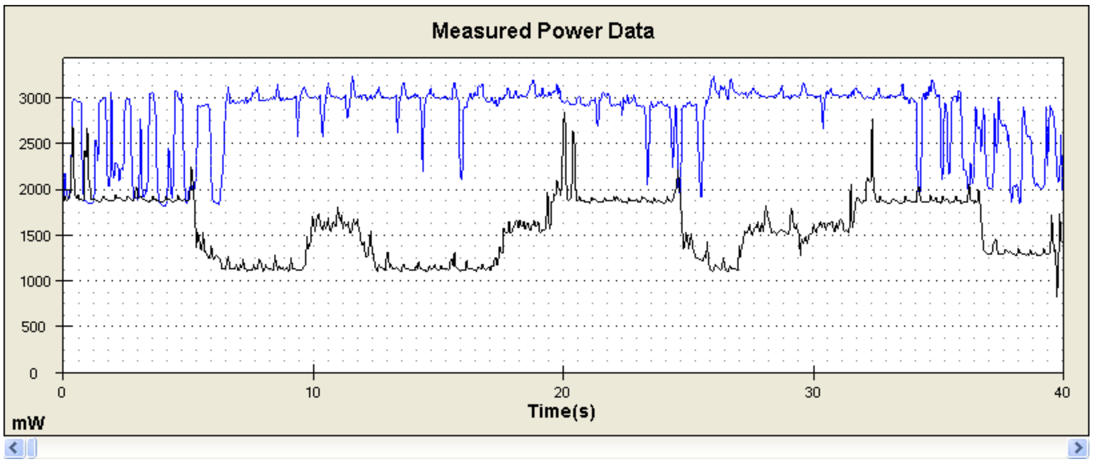


Figure D.1: Samsung Galaxy S: Native application / UMTS network

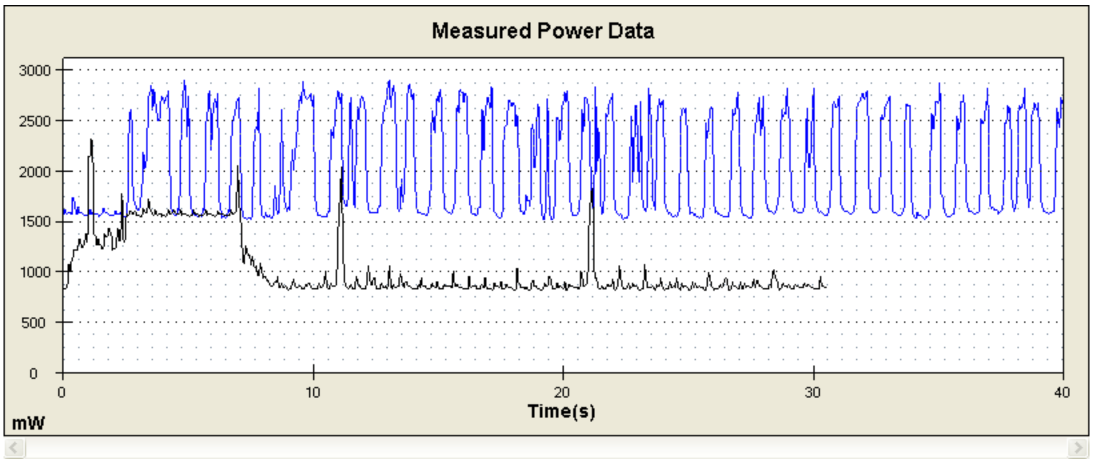


Figure D.2: Samsung Galaxy S: the default browser of Android 2.3 / UMTS network

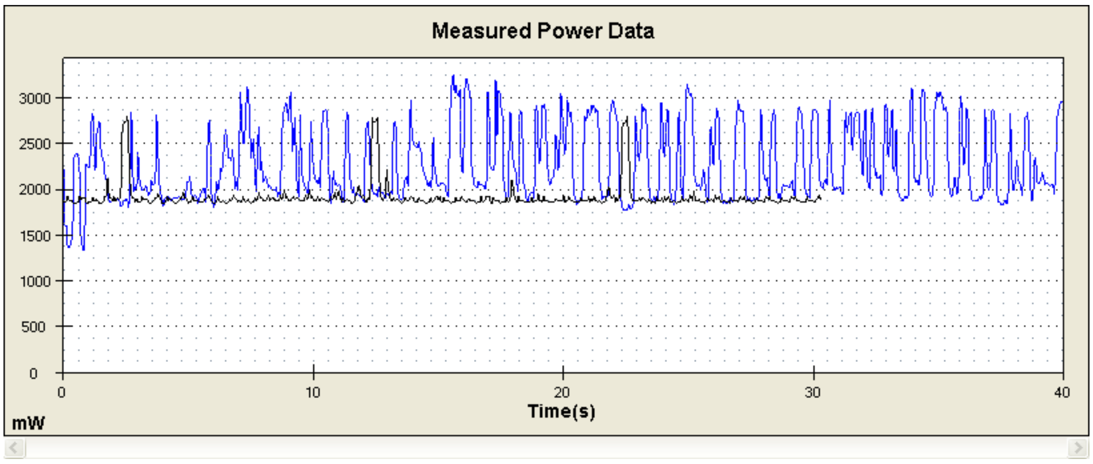


Figure D.3: Samsung Galaxy S: Opera 12.10 / UMTS network

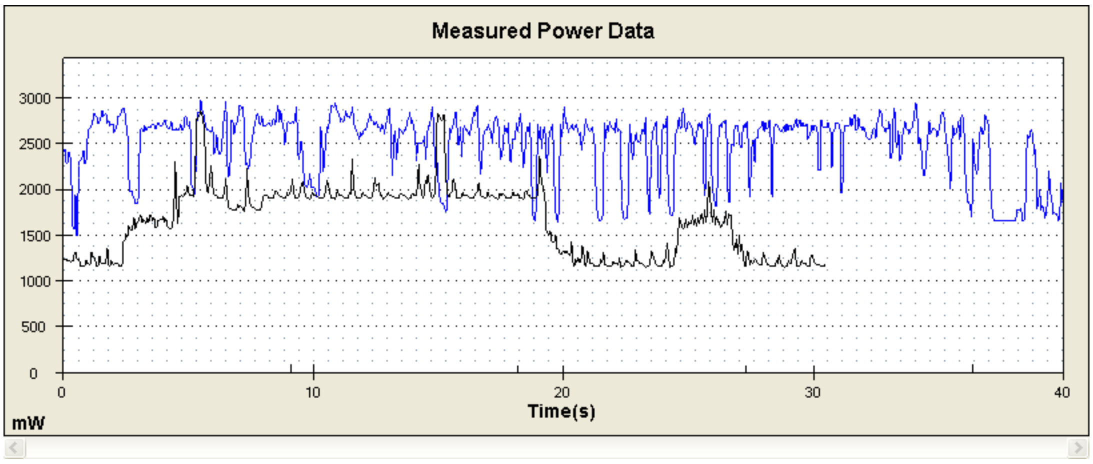


Figure D.4: Samsung Galaxy S: Firefox 17 / UMTS network

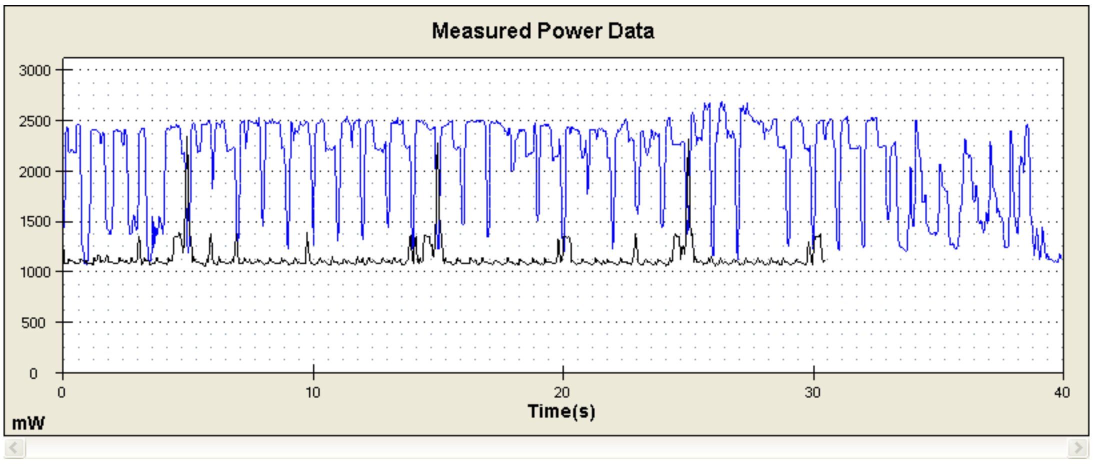


Figure D.5: Samsung Galaxy S: Native application / WiFi network

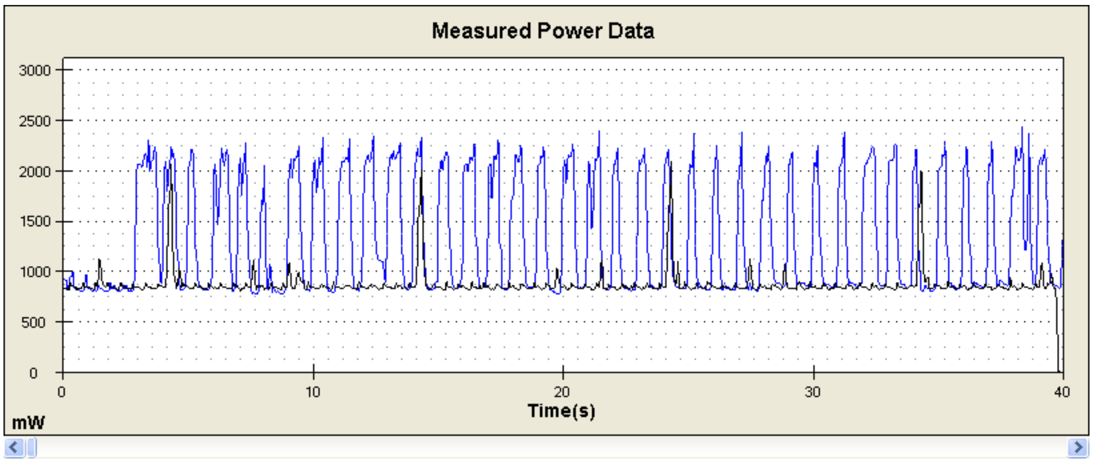


Figure D.6: Samsung Galaxy S: Android 2.3 Browser / WiFi network

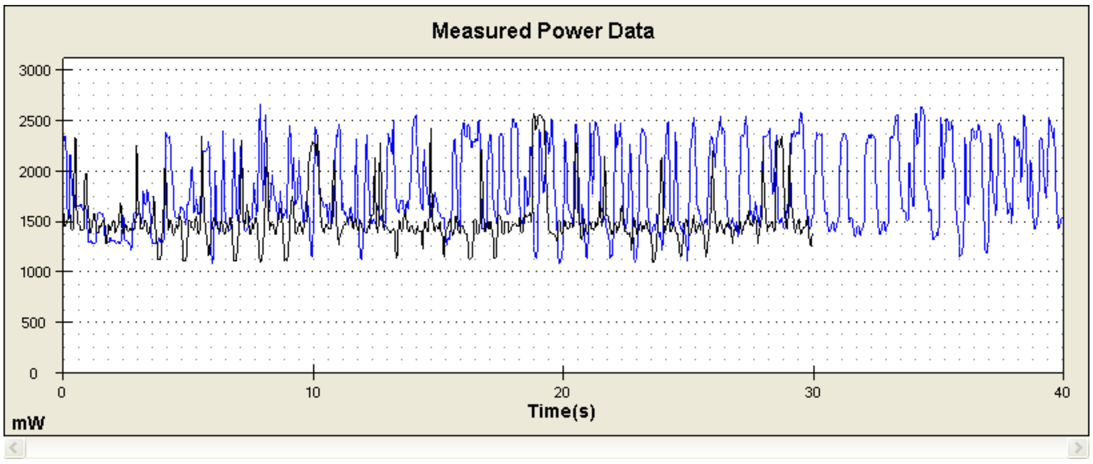


Figure D.7: Samsung Galaxy S: Opera 12.10 / WiFi network

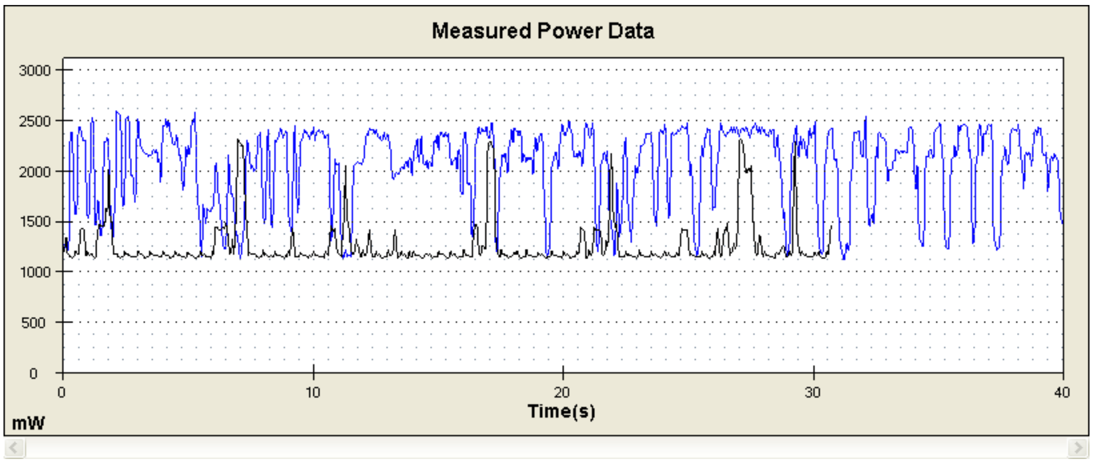


Figure D.8: Samsung Galaxy S: Firefox 17 / WiFi network

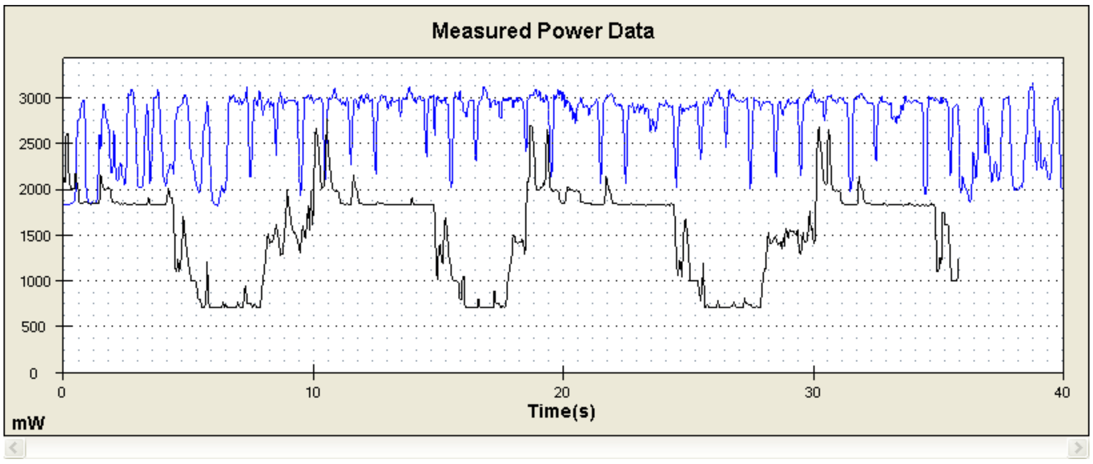


Figure D.9: Motorola Milestone 2: Native application / UMTS network

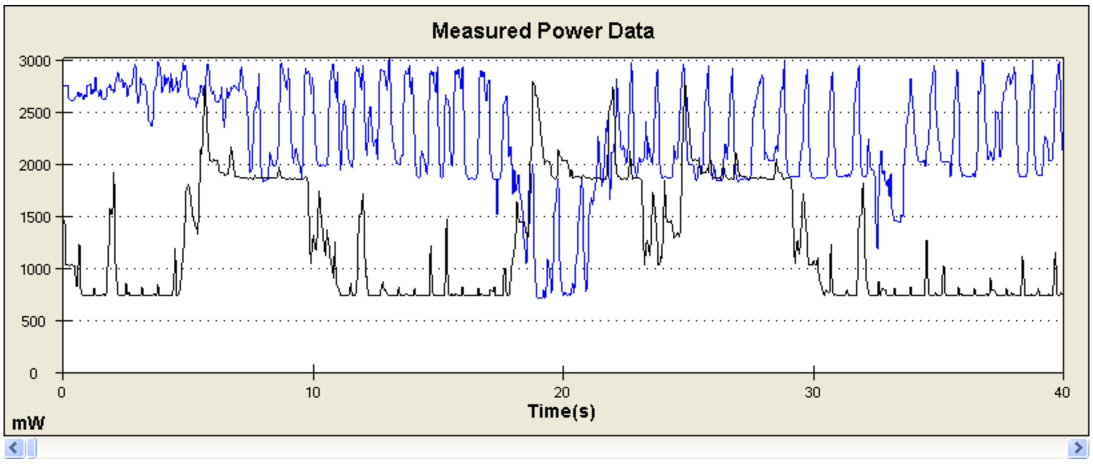


Figure D.10: Motorola Milestone 2: Android 2.3 Browser / UMTS network

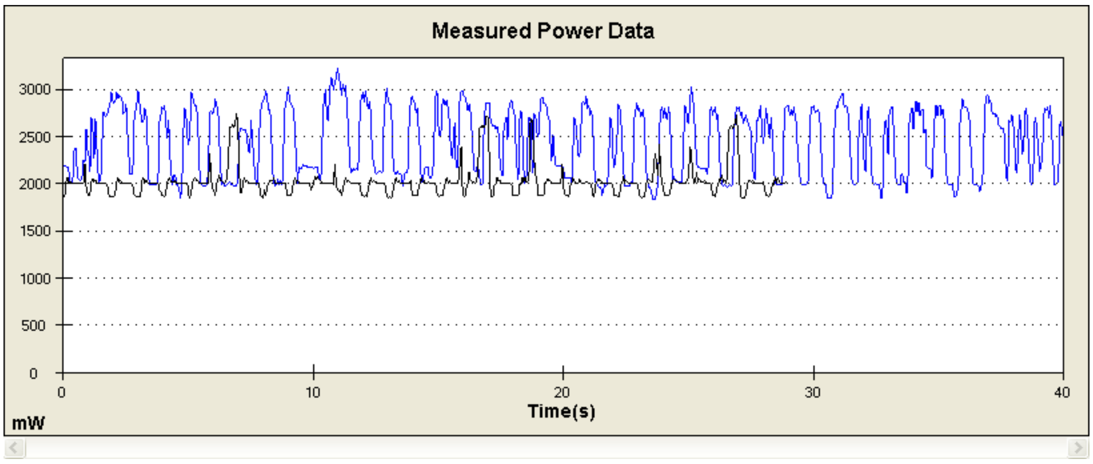


Figure D.11: Motorola Milestone 2: Opera 12.10 / UMTS network

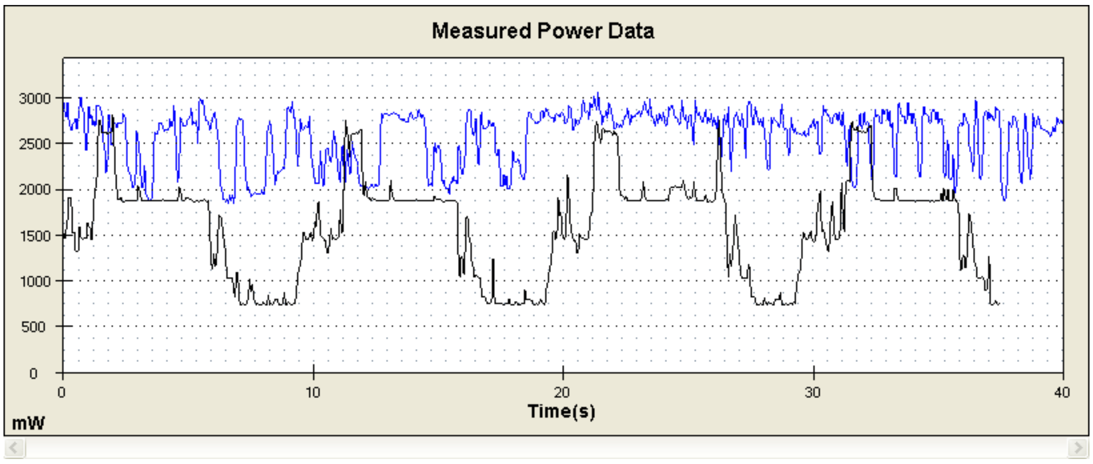


Figure D.12: Motorola Milestone 2: Firefox 17 / UMTS network

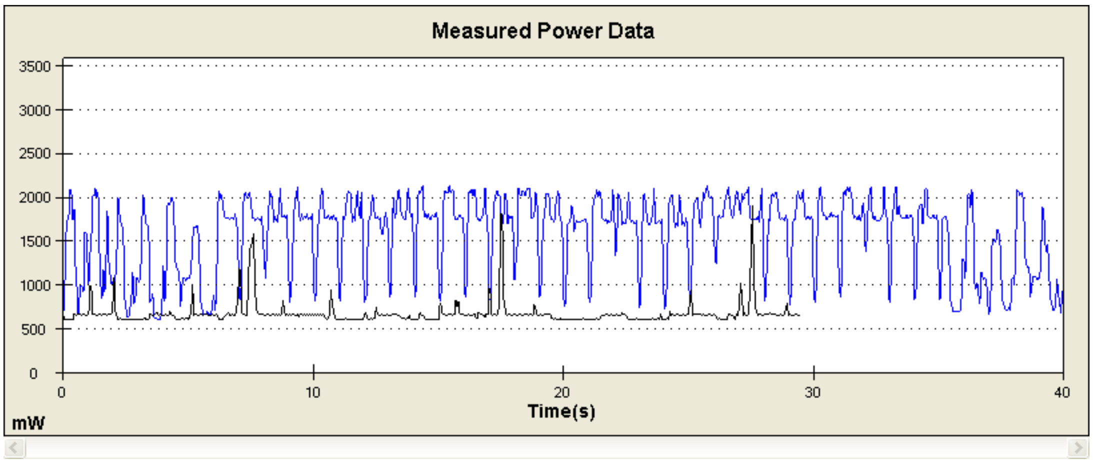


Figure D.13: Motorola Milestone 2: Native application / WiFi network

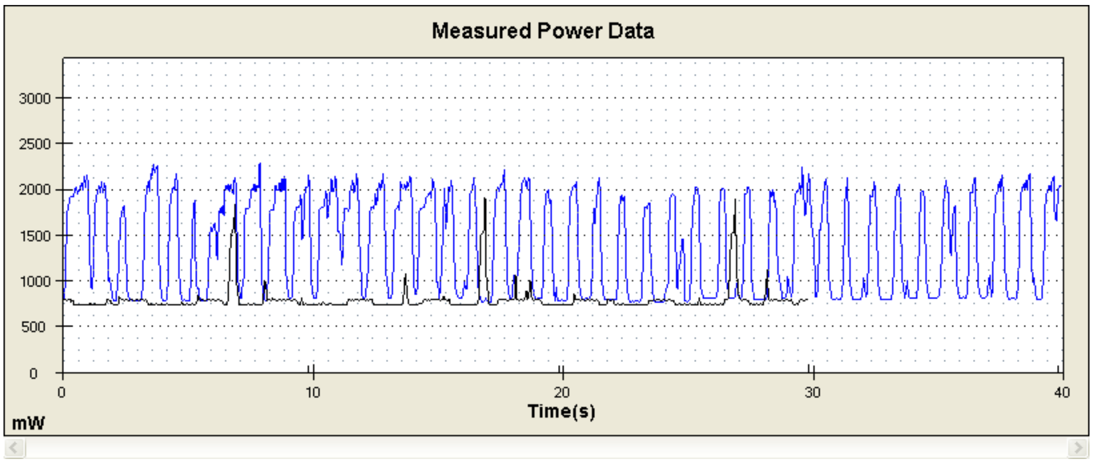


Figure D.14: Motorola Milestone 2: Android 2.3 Browser / WiFi network

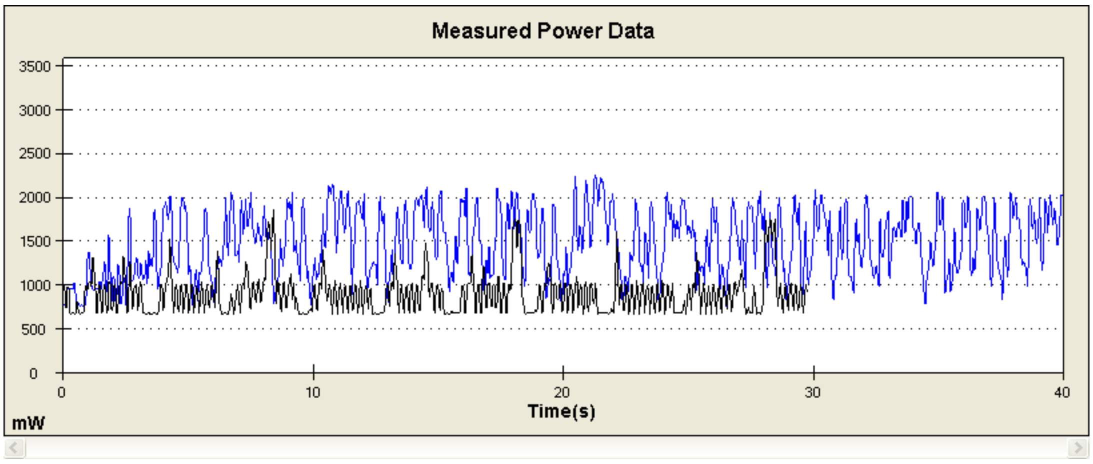


Figure D.15: Motorola Milestone 2: Opera 12.10 / WiFi network

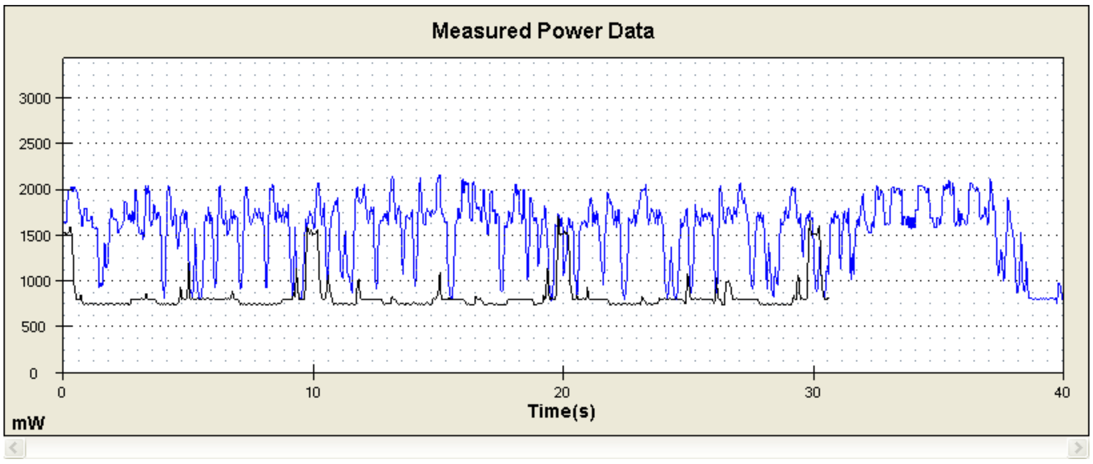


Figure D.16: Motorola Milestone 2: Firefox 17 / WiFi network

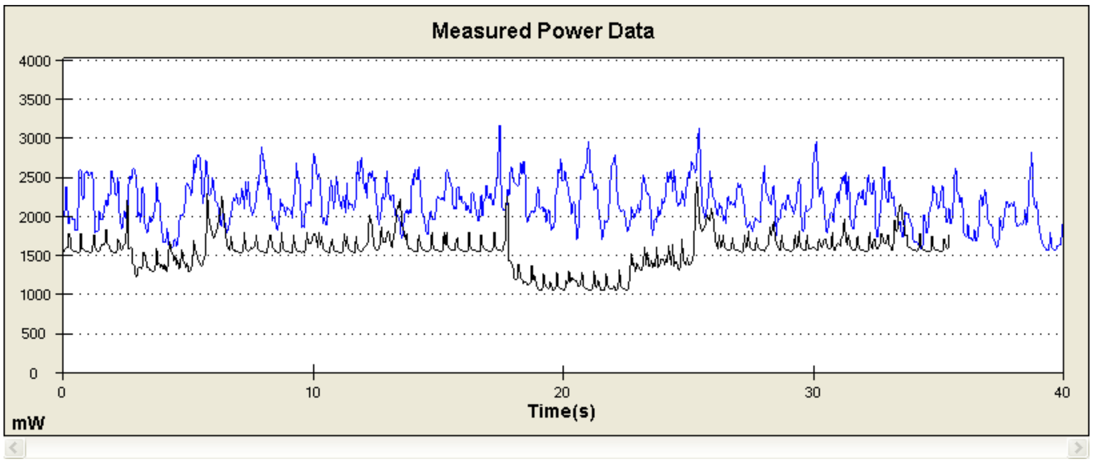


Figure D.17: Samsung Galaxy S III: Native application / UMTS network

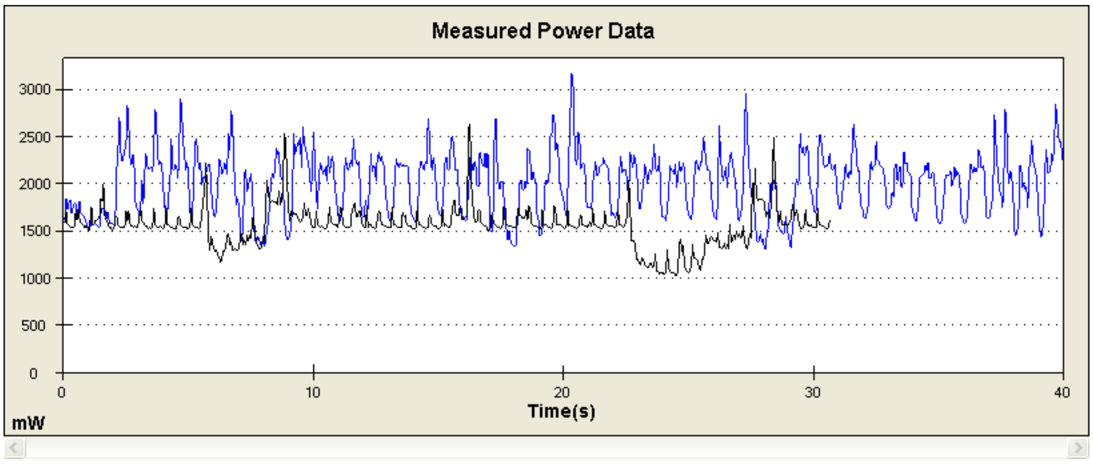


Figure D.18: Samsung Galaxy S III: Android 4.1 Browser / UMTS network

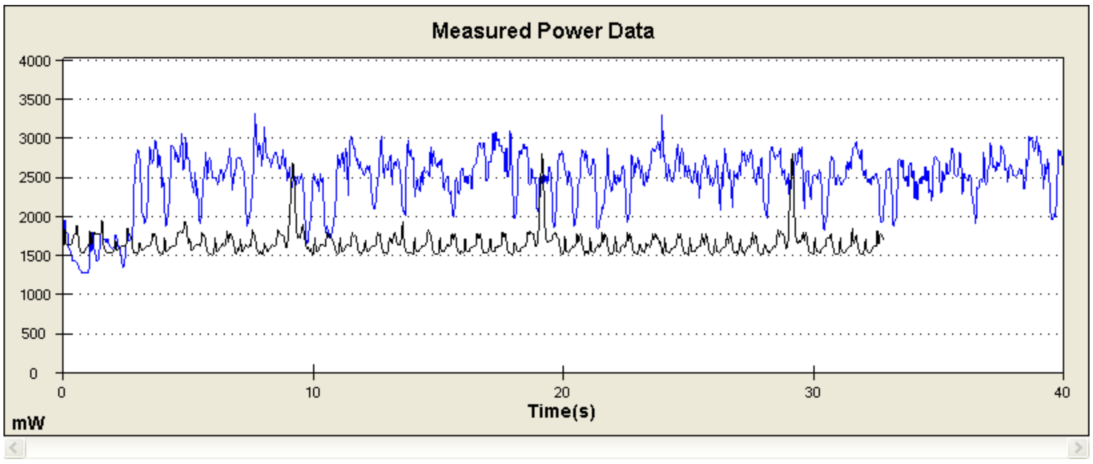


Figure D.19: Samsung Galaxy S III: Opera 12.10 / UMTS network

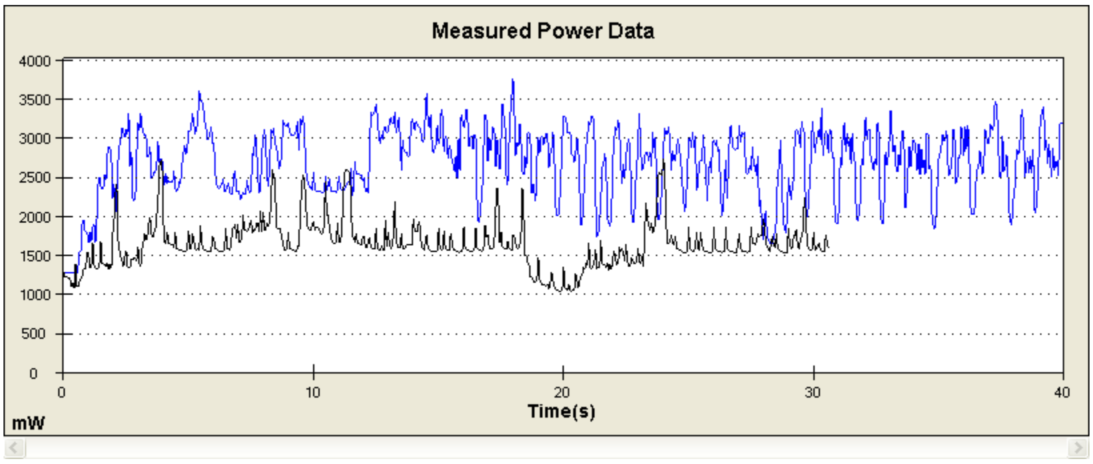


Figure D.20: Samsung Galaxy S III: Firefox 17 / UMTS network

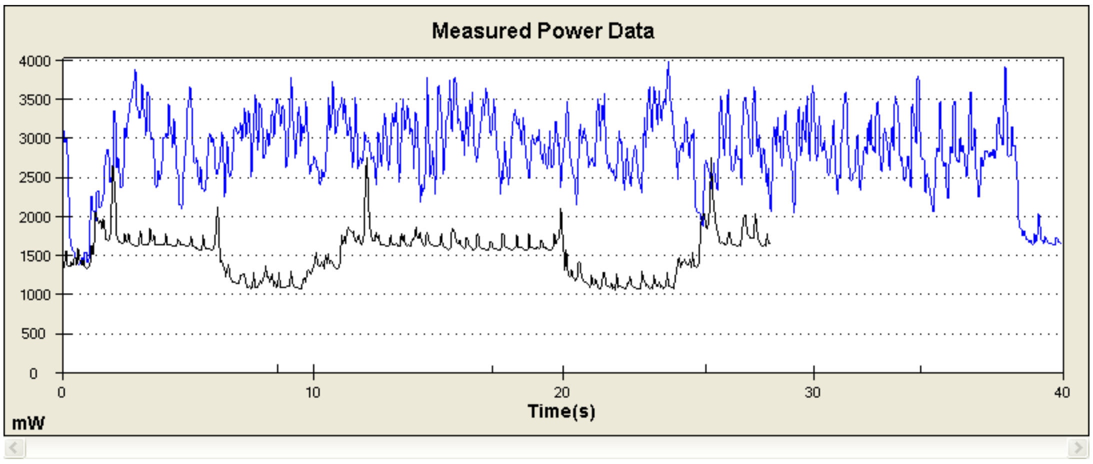


Figure D.21: Samsung Galaxy S III: Chrome 18 / UMTS network

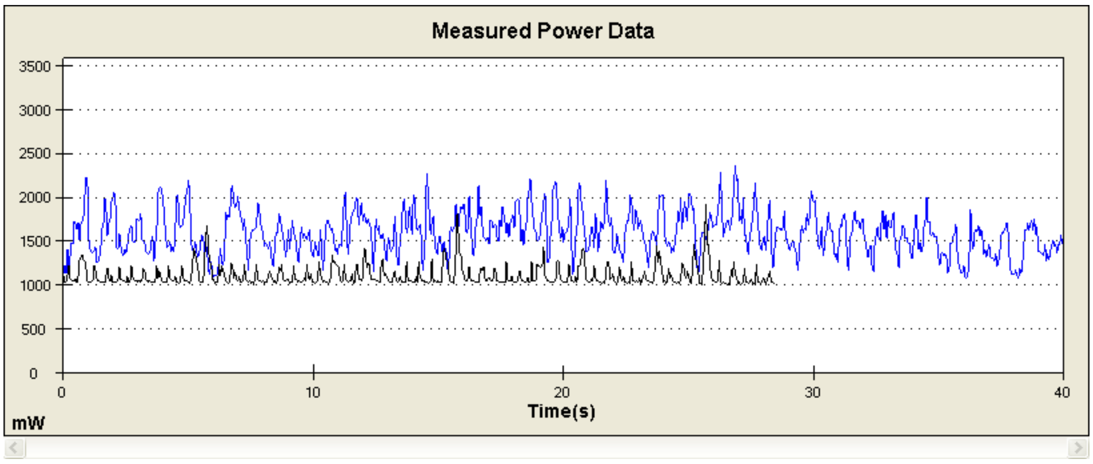


Figure D.22: Samsung Galaxy S III: Native application / WiFi network

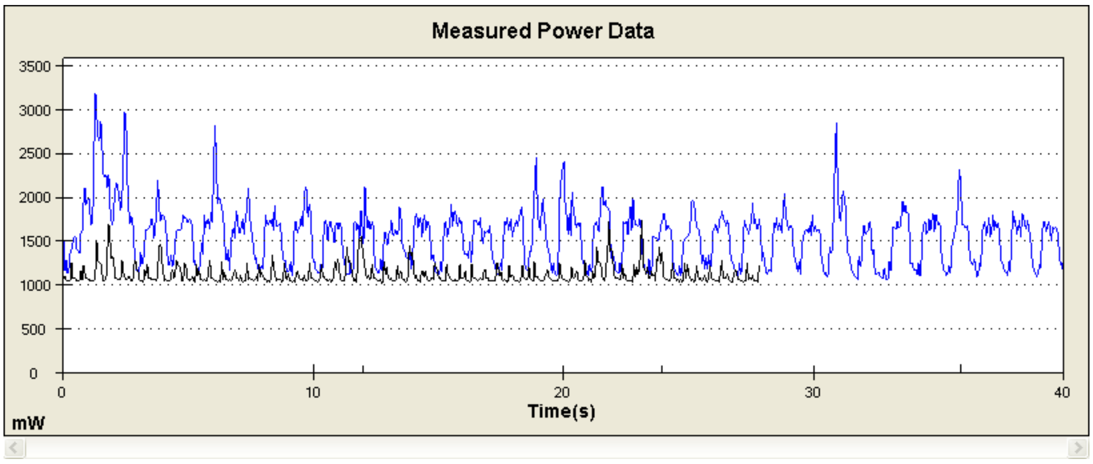


Figure D.23: Samsung Galaxy S III: Android 4.1 Browser / WiFi network

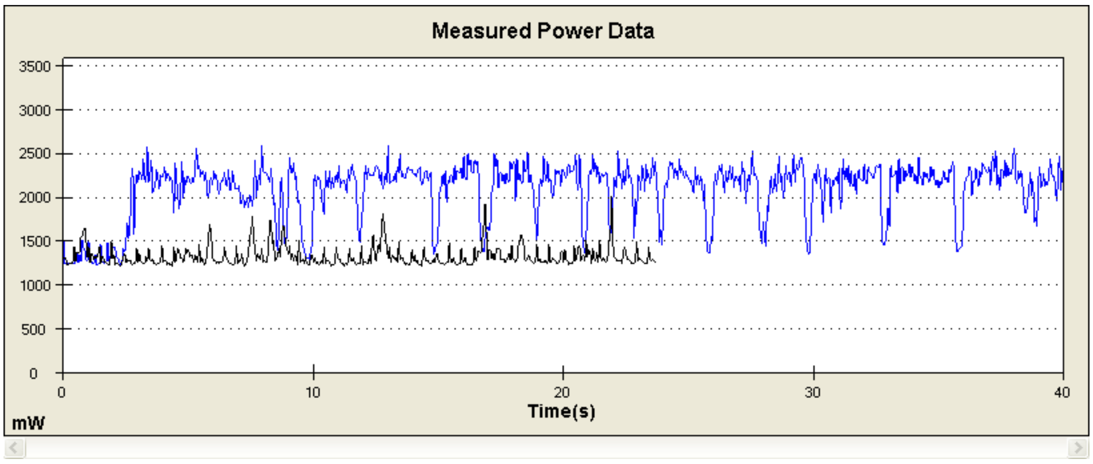


Figure D.24: Samsung Galaxy S III: Opera 12.10 / WiFi network

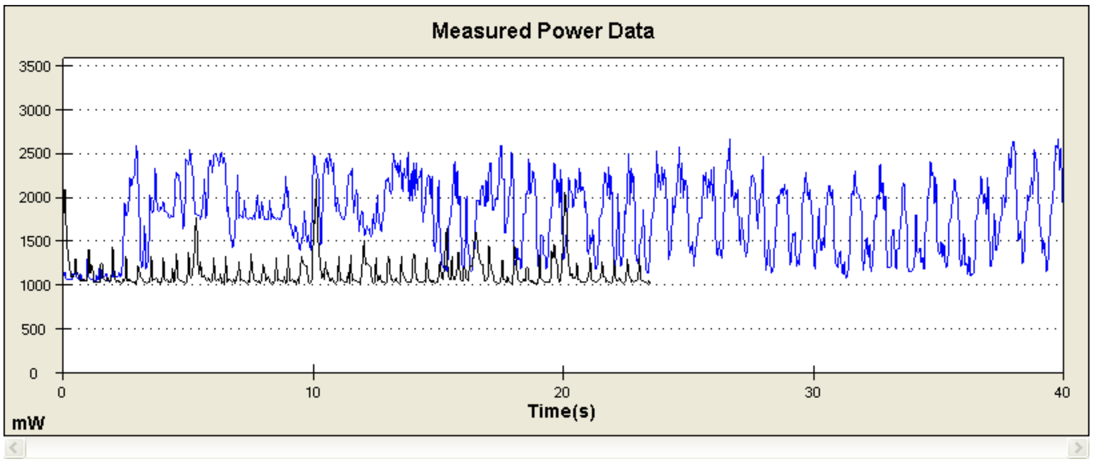


Figure D.25: Samsung Galaxy S III: Firefox 17 / WiFi network

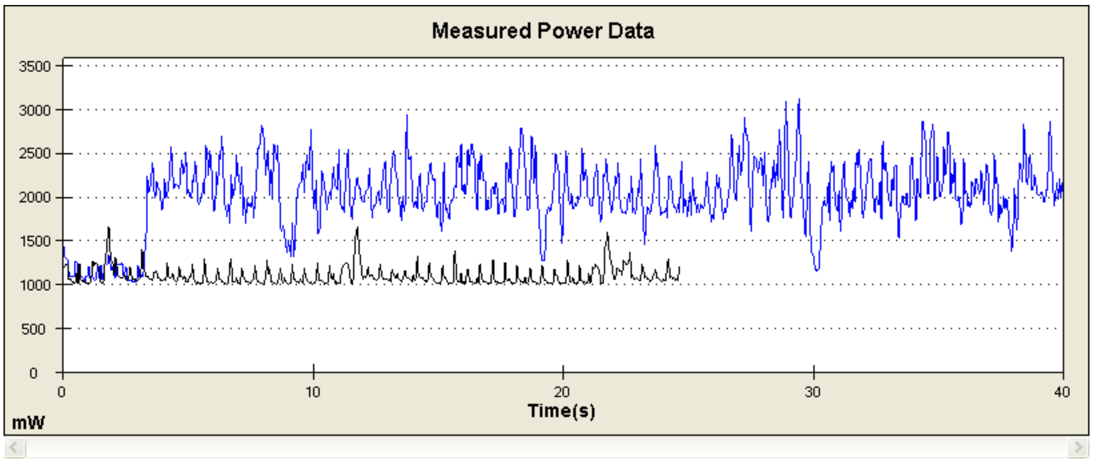


Figure D.26: Samsung Galaxy S III: Chrome 18 / WiFi network

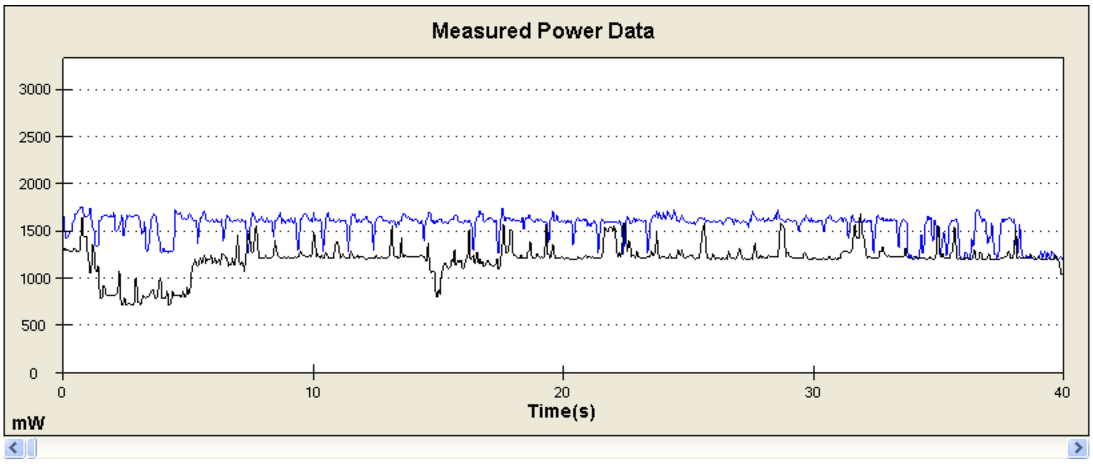


Figure D.27: Sony Xperia J: Native application / UMTS network

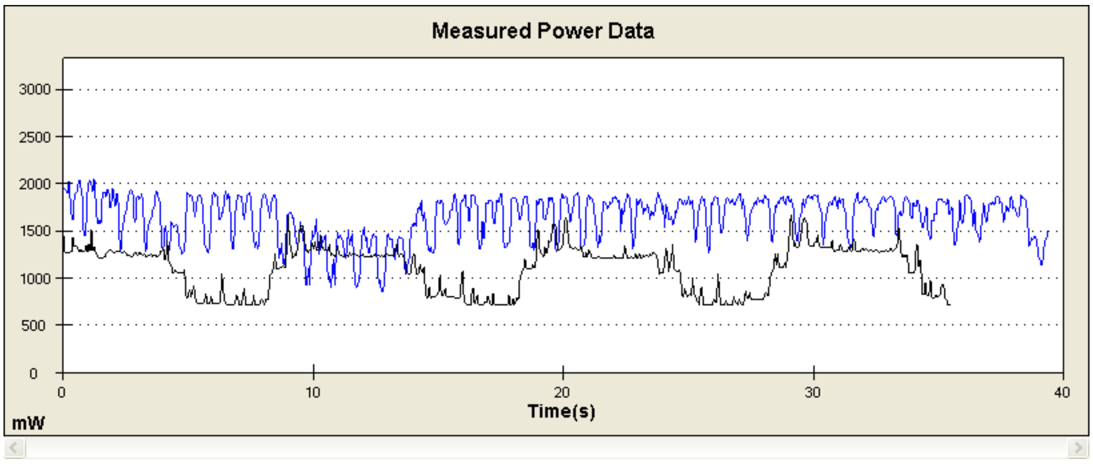


Figure D.28: Sony Xperia J: Android 4.0 Browser / UMTS network

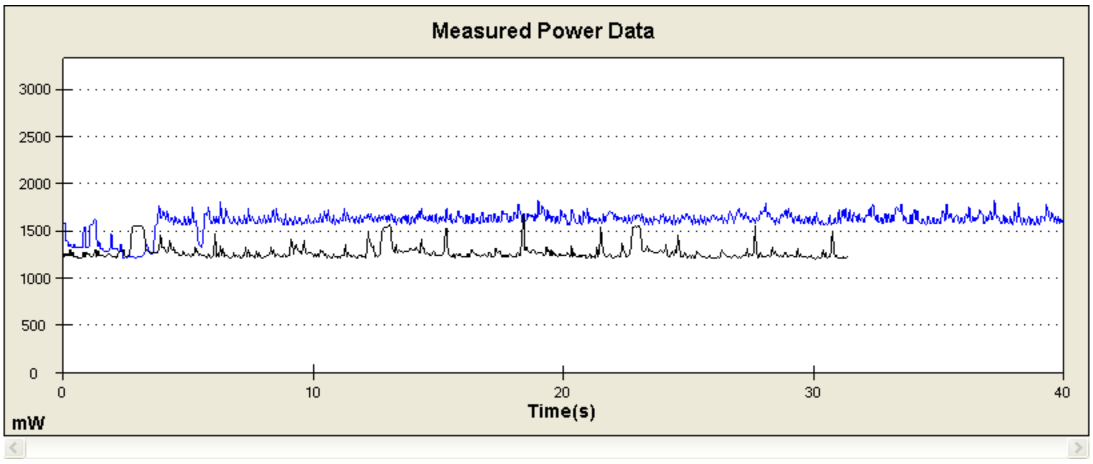


Figure D.29: Sony Xperia J: Opera 12.10 / UMTS network

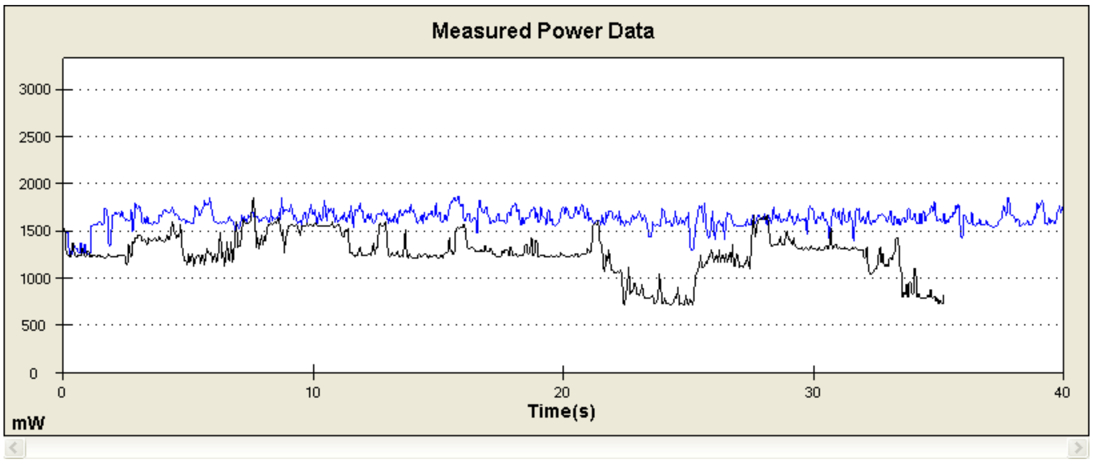


Figure D.30: Sony Xperia J: Firefox 17 / UMTS network

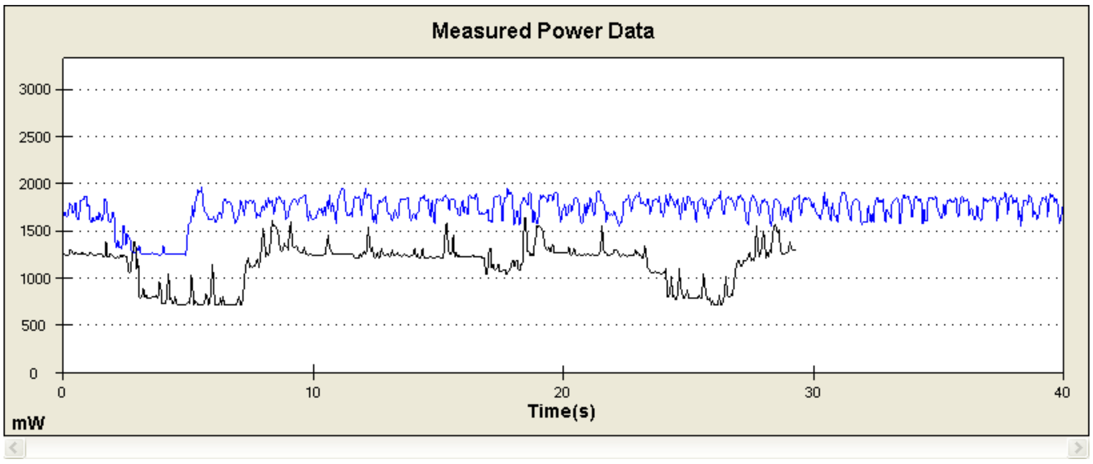


Figure D.31: Sony Xperia J: Chrome 18 / UMTS network

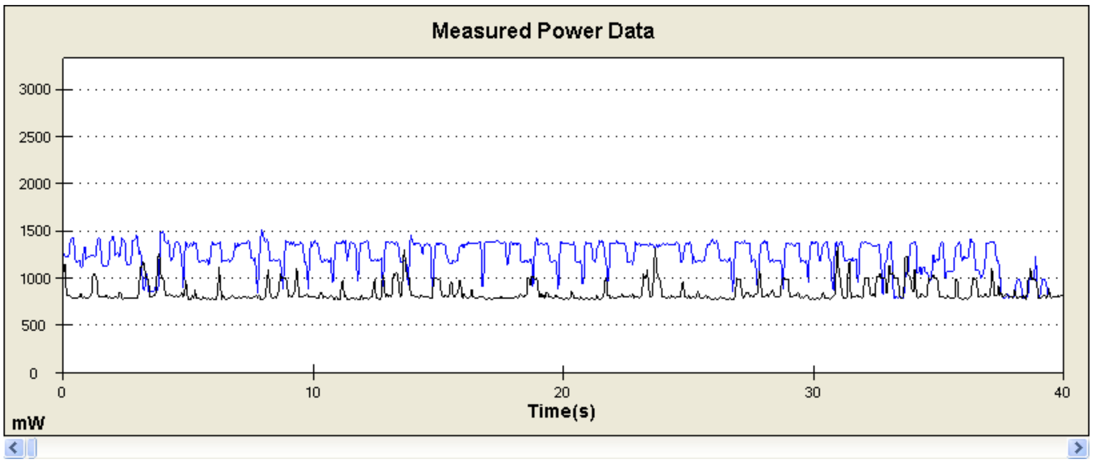


Figure D.32: Sony Xperia J: Native application / WiFi network

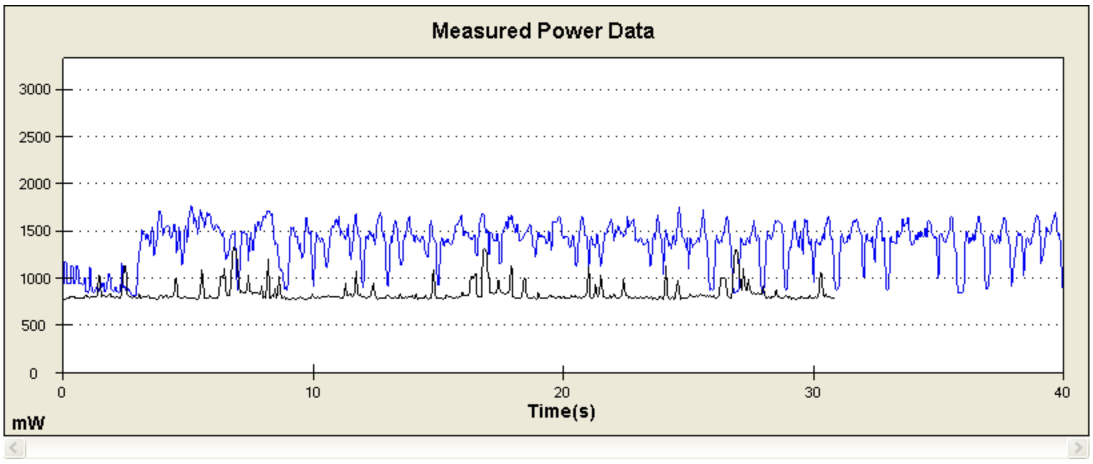


Figure D.33: Sony Xperia J: Android 4.0 Browser / WiFi network

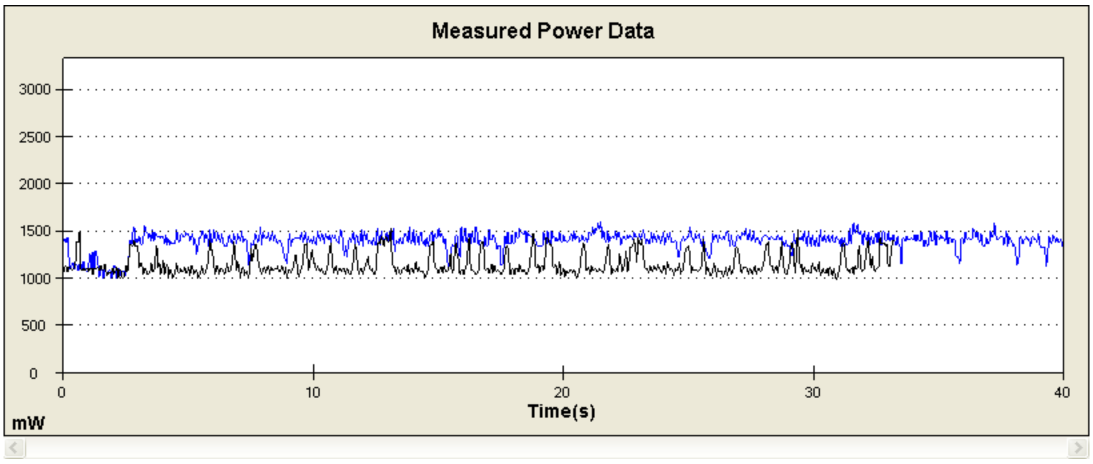


Figure D.34: Sony Xperia J: Opera 12.10 / WiFi network

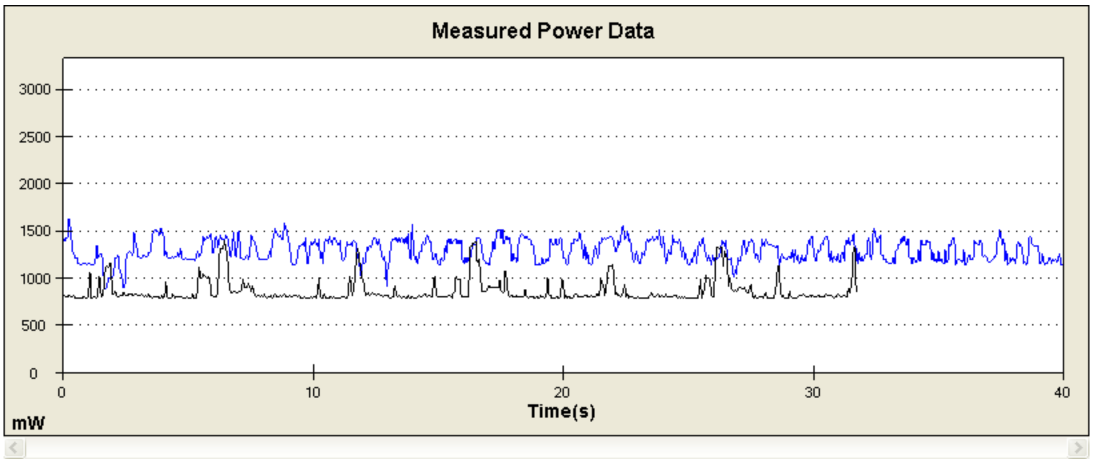


Figure D.35: Sony Xperia J: Firefox 17 / WiFi network

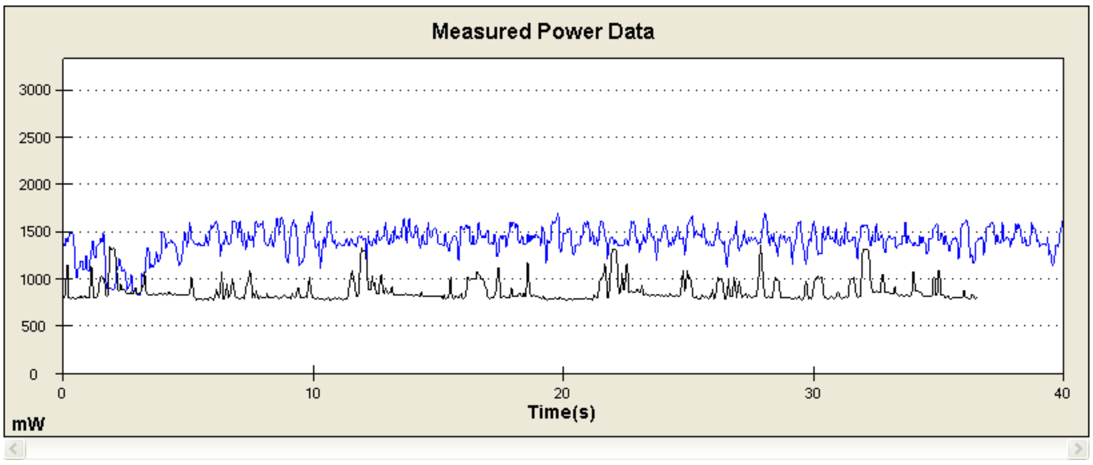


Figure D.36: Sony Xperia J: Chrome 18 / WiFi network