UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA TELEMÁTICA

Design and implementation of a traffic control framework in Firefox OS

AUTOR: FRANCISCO GONZÁLEZ GARCÍA
TUTOR: DR. JOSÉ IGNACIO MORENO NOVELLA

Marzo 2014

Título: *DESIGN AND IMPLEMENTATION OF A TRAFFIC CONTROL FRAMEWORK IN FIREFOX OS*

Autor: *FRANCISCO GONZÁLEZ GARCÍA*

Tutor: *DR. JOSÉ IGNACIO MORENO NOVELLA*

La defensa del presente Trabajo Fin de Grado se realizó el día 4 de marzo de 2014; siendo calificada por el siguiente tribunal:

Presidente: *ALBERTO GARCÍA MARTÍNEZ*

Secretario: *JOSÉ JOAQUÍN ESCUDERO GARZAS*

Vocal: *ALBERTO CORTÉS MARTÍN*

Habiendo obtenido la siguiente calificación:

Calificación:

Presidente **Secretario** **Vocal**

Acknowledgements

I would like to thank my advisor Dr. José Ignacio Moreno for his guidance and advice during my internship, for helping me whenever I have needed him.

I would like to express my sincere gratitude and respect to my supervisor Dr. Nico Bayer for giving me the opportunity to work with him. He is an example to follow for me because of his attitude, expertise and dedication to work. I would have been lost without his ideas and advice.

Thanks to Roman Szczepanski for his valuable technical support and for helping me to shape my skills with computers. His willingness to give his time has been very much appreciated. I would also like to thank Hans Einsiedler and the other members of his team for their help and assistance during these six months.

I will forever be grateful to all of you for giving me the opportunity to work here and culminate my studies in such a special way.

Thanks to my fellow mates Arancha, Miguel and Dani because this internship would have never been the same without them. Our "kicker" matches and time together are some of the sweetest memories that I take with me.

Thanks to my "Primo" Alberto for being a real friend, he is absolutely one of a kind. Thanks to my friend Josema for finding always time for visiting me no matter where I am. Thanks to my friend Jose for welcoming me every time I come back in town.

Thanks to all the great people I met during my stay in Finland, especially my "tutor" Ayman and my "brother" Yifei. Thanks to the Demola team for giving me the chance to take part into an amazing project where I could learn a lot.

Thanks to my friends and relatives, to all of you who have supported me from the beginning and remained with me until the end.

Lastly, and most importantly, I wish to thank my family who have been always there for me, helped me to find my path in life and supported me in every decision I made. Thanks to my father Paco for instilling in me the passion for learning, for teaching me to always look for the answer to every question. Thanks to my mother Rosi for her patience and endless support, for teaching me to always aim high. Thanks to my little sister Gloria for spoiling me when I come back home and for standing my jokes. To them I dedicate this thesis.

Alku aina hankala, lopussa kiitos seisoo.
(All's well, that ends well.)

— Finnish proverb

⋆⋆⋆

A mis padres y hermana: Paco, Rosi y Gloria.

Abstract

Today's smartphones include a rich feature-set as well as various wireless interfaces that provide extra services rather than just voice communication or messaging, as it occurred with traditional mobile phones. Additionally, the widespread use of mobile devices using Third Generation (3G) and Long Term Evolution (LTE) networks has led to the development of various applications (apps) that take advantage of the always-on Internet connectivity provided by these networks (e.g. instant messaging and social network services). Unlike traditional Internet apps (e.g. web surfing and file transfer), the emerging apps that rely on always-on connectivity are often constantly running in the background to receive messages and status updates. This behavior causes that apps continuously generate short app signaling messages such as keep-alive and ping requests to maintain the always-on connectivity.

Although the traffic volume of keep-alive messages is not large, frequent short messages can incur a large amount of related signaling traffic in the mobile network. In 3G or LTE networks, the User Equipment (UE) and the Radio Access Network (RAN) keep the Radio Resource Control (RRC) states. The UE stays in Connected mode when it transmits or receives data during active periods and stays in Idle mode during inactive periods. To send even a small data packet, the UE changes the state to the Connected mode prior to transmission. This radio state change generates a lot of network signaling messages, resulting in a rapid increase in traffic loading. Large amounts of network signaling traffic leads to two major problems: rapid drainage of the mobile device's battery and a signaling traffic surge in the mobile network.

Since the air interface is a spare resource and the traffic for mobile end devices will grow enormously, it is important that the wireless resources are used in the most efficient way. However, this is not true for current networks as there is not alignment between devices, apps and the network.

This document proposes a traffic control framework which acts as an interface between the apps and the network and allows the network operator to aggregate packets prior to transmission. The aggregated packets are sent out at once after a configurable amount of time which means for instance that resources on the wireless link have to be reserved only once for a number of app signaling packets and not for each packet separately. By this the packet transmission will be bursty which will improve network efficiency as the amount of signaling messages is minimized. In addition, battery runtime is improved as lower signaling overhead will reduce the activity time and energy consumption within devices.

Resumen

Hoy en día los smartphones incorporan un amplio conjunto de utilidades, así como varias interfaces inalámbricas que proporcionan servicios adicionales a los ofrecidos por los teléfonos móviles convencionales. Por otra parte, el uso generalizado de las redes 3G y LTE ha originado el desarrollo de numerosas aplicaciones que aprovechan las ventajas que ofrecen dichas redes, un ejemplo son las aplicaciones de redes sociales. Estas aplicaciones, a diferencia de otras como la navegación web o la descarga de archivos, están constantemente ejecutándose en segundo plano y recibiendo notificaciones de actualización de estado. Este comportamiento propicia el intercambio de pequeños mensajes de señalización para mantener la conexión, tales como mensajes "'keep alive" o "ping requests".

A pesar de que el volumen de estos mensajes no es elevado, su constante intercambio puede ocasionar una gran cantidad de tráfico de señalización en la red. En las redes 3G o LTE, el equipo de usuario (UE) y la red de acceso radio terrestre (RAN) mantienen los estados RRC. El equipo de usuario permanece en el estado activo cuando transmite o recibe datos y retorna al estado de reposo durante los periodos inactivos. El envío de un pequeño paquete de datos supone la transición desde el estado de reposo al estado activo. Este comportamiento genera muchos mensajes de señalización e implica un rápido incremento en el tráfico de la red. Este incremento del tráfico de señalización ocasiona dos grandes problemas: la sobrecarga de la red y un impacto negativo en el consumo de batería de los dispositivos móviles.

Es de vital importancia que se haga un uso eficiente de los recursos de red, ya que el aire, en este caso el canal de comunicación, es un medio compartido. Además, se espera que el tráfico generado por los dispositivos móviles crezca enormemente en los próximos años. Las redes móviles actuales no son utilizadas de un modo eficiente debido a la falta de interacción entre la red, los dispositivos móviles y las

aplicaciones.

Este documento presenta una plataforma de control de tráfico que actúa como interfaz entre las aplicaciones y la red, permitiendo al operador de red agregar los paquetes antes de su transmisión. Esto permite, por ejemplo, que los recursos de red sean reservados sólo una vez para la ráfaga de paquetes y no para cada paquete individualmente, lo cual minimiza la cantidad de mensajes de señalización. Esta propuesta no sólo ayuda a mejorar la eficiencia de la red, sino que además optimiza el uso de la batería, ya que una disminución del tráfico de señalización provoca una reducción del tiempo de actividad y consumo de energía de los dispositivos móviles.

Contents

Abstract **ix**

Resumen **xi**

Contents **xiv**

List of Acronyms **xv**

List of Figures **xix**

List of Tables **xxi**

Listings **xxiii**

1 Introduction **1**

1.1 Motivation 2

1.2 Objectives and project phases 4

1.3 Document structure 5

2 State of the Art **7**

2.1 The smartphone challenge 8

2.2 Firefox OS 10

2.3 SoftToken Protocol 13

2.4 UMTS 16

2.5 WLAN 19

2.6 JSON 21

2.7 Node.js 22

2.8 MongoDB 23

2.9 Firefox OS-based Geeksphone Keon 25

3 Development **27**
3.1 Traffic control framework 28
3.1.1 Queuing in the Linux network stack 28
3.1.2 Packet aggregation approach 29
3.2 Implementation 31
3.2.1 Overview 31
3.2.2 Firefox OS Network Control app 33
3.2.3 Control daemon 35
3.2.4 Master daemon 37
3.2.5 Server and database 38
3.2.6 Integration of the packet aggregation module 39
3.2.7 Integration of command-line utilities 43

4 Evaluation **47**
4.1 Key Performance Indicators 47
4.1.1 Battery runtime 48
4.1.2 Network efficiency 48
4.1.3 Service quality 49
4.1.4 Comparison 52
4.2 Service quality measurements 52
4.2.1 Measurement environment 53
4.2.2 Measurement results 55
4.2.3 Conclusion 58

5 Outlook and conclusions **59**
5.1 Summary 59
5.2 Conclusions 62

A Budget **63**
A.1 Project phases 63
A.2 Material expenses 66
A.3 Human resources expenses 66
A.4 Total expenses 67

References **69**

List of Acronyms

AES Advanced Encryption Standard

AP Access Point

API Application Programming Interface

CCMP Counter Mode CBC-MAC Protocol

CN Core Network

CPITM Context- and Policy based Interface and Traffic Manager

CSS Cascading Style Sheets

DSL Digital Subscriber Line

FIFO First In, First Out

FTP File Transfer Protocol

GSM Global System for Mobile Communications

GUI Graphic User Interface

HAL Hardware Abstraction Layer

HSDPA High Speed Downlink Packet Access

HSPA High Speed Packet Access

HSUPA High Speed Uplink Packet Access

HTB Hierarchical Token Bucket

HTML5 HyperText Markup Language, version 5

HTTP Hypertext Transfer Protocol

IEEE Institute of Electrical and Electronics Engineers

ISM Industrial, Scientific and Medical

JSON JavaScript Object Notation

KPI Key Performance Indicator

LAMP Linux-Apache-MySQL-PHP

LAN Local Area Network

LTE Long Term Evolution

MAC Media Access Control

MOS Mean Opinion Score

NFC Near Field Communication

NIC Network Interface Controller

OEM Original Equipment Manufacturer

OFDM Orthogonal Frequency-Division Multiplexing

OpenEPC Open Evolved Packet Core

QoE Quality of Experience

QoS Quality of Service

RAN Radio Access Network

RDBMS Relational Database Management System

RNC Radio Network Controller

RRC Radio Resource Control

SIM Subscriber Identity Module

SKB Socket Kernel Buffer

TC Traffic Class

UE User Equipment

UMTS Universal Mobile Telecommunications System

WEP Wired Equivalent Privacy

WLAN Wireless Local Area Network

WPA2 Wi-Fi Protected Access 2

W-CDMA Wideband Code Division Multiple Access

XML eXtensible Markup Language

2G Second Generation

3G Third Generation

3GPP 3rd Generation Partnership Project

List of Figures

1.1 *Overall objective: Improving network efficiency.* 4

2.1 *Data and network signaling volume (Source: Nokia Siemens Networks Smart Labs 2011).* 9
2.2 *Network signaling and battery consumption in HSPA (Source: Alcatel-Lucent 2012).* 10
2.3 *Firefox OS logo.* 10
2.4 *Firefox OS architecture (Source: Mozilla Developer Network 2013).* . . 12
2.5 *SoftToken architecture.* 14
2.6 *HSPA state transitions (Source: Nokia 2010).* 18
2.7 *Wi-Fi logo.* 19
2.8 *Graphical representation of Wi-Fi channels in the 2.4 Ghz band (Source: Wikipedia 2014).* 20
2.9 *Object representation.* 21
2.10 *Array representation.* 21
2.11 *Node.js logo.* 22
2.12 *Apache vs. Node.js. Concurrency benchmark (Source: http://code.google.com/p/node-js-vs-apache-php-benchmark/wiki/Tests 2010).* . 22
2.13 *MongoDB logo.* 23
2.14 *Keon – Front.* 25
2.15 *Keon – Back.* 25

3.1 *Simplified overview of the transmit path of the Linux network stack (Source: D. Siemon, Linux Journal 2013).* 29
3.2 *Irregular transmission of app data or signaling packets.* 29
3.3 *Example of packet aggregation.* 30
3.4 *Architecture of the new transmit path.* 30

3.5 *Simplified architecture of the traffic control framework.* 31
3.6 *Shell interface.* 33
3.7 *Control daemon running.* 34
3.8 *Control daemon stopped.* 34
3.9 *FTP download via Wi-Fi.* 34
3.10 *Firefox OS userspace process with the master daemon (Adapted from Mozilla Developer Network 2013).* 37
3.11 *Traffic control framework and SoftToken module.* 42
3.12 *Capture on wlan0 interface without SoftToken.* 42
3.13 *Capture on wlan0 interface with SoftToken.* 43
3.14 *Wireshark capture on rmnet0 interface.* 45

4.1 *Measurement setup.* 53
4.2 *Website access time vs. aggregation delay.* 55
4.3 *Web browsing performance gain vs. aggregation delay.* 56
4.4 *FTP download time vs. aggregation delay.* 57
4.5 *FTP file transfer performance gain vs. aggregation delay.* 57

5.1 *Architecture of the CPITM.* 61

A.1 *Gantt diagram.* 65

List of Tables

2.1 *Amount of network signaling messages required for different state transitions in High Speed Packet Access (HSPA) (Data from Signals Research Group 2010).* 18

3.1 *List of implemented commands.* 33

4.1 *Web browsing measurement results.* 50
4.2 *Mapping functions between Mean Opinion Score (MOS) and number N of stalling events of length L.* 51

A.1 *Material expenses.* 66
A.2 *Human resources expenses.* 67
A.3 *Total expenses.* 67

Listings

3.1 Command message 36
3.2 Response message 37
3.3 Server output 38
3.4 Database search 39
3.5 SoftToken configuration example for wlan0 41
3.6 SoftToken configuration example for rmnet0 41
3.7 tc usage 44
3.8 iperf usage 45

Chapter 1

Introduction

"I don't believe there is anything in the whole earth that you can't learn in Berlin except the German language."

— *Mark Twain*

This document describes the final project prepared as a result of the work developed during an internship under the Erasmus Placement Programme. The work was carried out at Deutsche Telekom Innovation Laboratories (T-Labs) in Berlin from August 2013 to January 2014, within the Seamless Network Control department and under the supervision of Dr. Nico Bayer.

Telekom Innovation Laboratories (T-Labs) is the central research and innovation unit of Deutsche Telekom. T-Labs is also an associated scientific institute of the Technische Universität (TU) Berlin that is organized under private law. Around 360 Telekom experts and scientists from various disciplines from all over the world work in Berlin and at other sites in Darmstadt (Germany), Bonn (Germany), Beer Sheva (Israel) and Los Altos (USA) on developing novel services and solutions for the customers of Telekom. Another method of exploring results is the founding of new companies (start-ups). T-Labs is an internationally recognized research and development center for new information and communication technologies.

No regulatory framework that applies to the work developed has been found. Therefore, there is no reference to any technical or legal restriction in this document.

1.1 Motivation

The future of communication will be mobile and wireless. While some end devices such as TVs, set-top boxes, desktop PCs, etc. will be connected by wires, most of the end devices will get access via broadband wireless and mobile air interfaces. Since the air interface is a spare resource and the traffic for mobile end devices will grow enormously [1], it is important to use the wireless resources in the most efficient way. However, this is not the case nowadays [2, 3]. The reason is that there is only little interaction between the network and the mobile device as well as the applications (apps) and services.

Smartphones become more and more intelligent. Their feature-set is constantly increasing. For instance, current smartphones have various sensors (e.g. proximity, accelerometer, light, gyroscope, etc.), high resolution cameras at front and back, a GPS module, a compass, a microphone and a high resolution display. With the growth of smartphones features and the massive development of applications the mobile devices have turned into an all-round support system. While the early mobile devices were specialized for voice communication and messaging, current devices exploit the mentioned feature-set to be used for instance as a navigation system, wind sensor, variometer, TV, remote control, scanner, etc. Additionally, various wireless interfaces such as Global System for Mobile Communications (GSM), 3G, LTE, Wi-Fi, Bluetooth and Near Field Communication (NFC) can be found in a mobile device which have been introduced over time to be able to satisfy the ever growing bandwidth demand and to offer new services.

As can be seen there is a very strong interaction with the apps and the mobile device. However, the interaction between end devices and the network is very low and the wireless interfaces are considered as a bit-pipe only without taking into account the characteristics of each network technology.

Unlike traditional apps such as web browsing or file transfer, smart apps rely on the always-on Internet connectivity provided by 3G and LTE networks. They are often constantly running in the background to receive status updates. This behavior forces the device to continuously generate short app signaling messages such as keep-alive and ping requests to maintain the always-on connectivity. Although the traffic volume of keep-alive messages is not large, frequent short messages can incur a large amount of related signaling traffic in the mobile network.

In 3G or LTE networks, the UE and the RAN keep the RRC states. The UE

stays in Connected mode when it transmits or receives data during active periods and stays in Idle mode during inactive periods. To send even a small app data packet, the UE changes the state to the Connected mode prior to transmission. This radio state change generates a lot of network signaling messages, resulting in a rapid increase in signaling traffic loading.

Large amounts of signaling traffic lead to two major problems: rapid drainage of the mobile device's battery and a signaling traffic surge in the mobile network. All that leads to the fact that wireless resources are not used in the most efficient way, meaning that especially the wireless resource but also the energy resource usage on the mobile device as well as the delivered service quality is not optimal.

It is important to understand that there is a difference between application signaling and radio access (network) signaling. The app signaling are control messages exchanged between an app and a server for example. These messages are hidden in IP packets whereas the radio access signaling are control messages exchanged between the mobile device and the base station or the mobile core network. App signaling messages are transparent to the network and are treated as data packets. One of the biggest problems is the fact that in the majority of situations, the network does not have information about the app traffic and its requirements. This means that the network is not aware about the content of the packets and cannot differentiate between app signaling messages and app data packets. Signaling packets are often small IP packets and are sent out from time to time. Data packets are sent out normally in a row and can be aggregated so that the radio access signaling has to be established only for these aggregated IP packets. Since smartphones include numerous apps, a large number of app signaling messages are exchanged and this leads to signaling storms of radio signaling messages.

This document proposes a traffic control framework which acts as an interface between the apps and the network. The approach is based on the aggregation of data packets prior to transmission. The aggregated packets are sent out at once after a configurable amount of time which means for instance that resources on the wireless link have to be reserved only once for a number of app signaling packets and not for each packet separately. By this the packet transmission will be bursty which will improve network efficiency as the amount of signaling messages is minimized. In addition, battery runtime is improved as lower signaling overhead will reduce the activity time and energy consumption within devices.

1.2 Objectives and project phases

This project constitutes the first part of a larger project which is divided in two parts and whose scope is meant to last one year. The project presented in this document provides all the support and tools necessary to perform the second part and continue the research within network efficiency. Therefore, the overall objective here is to implement a traffic control framework which enables the network operator to study and control the traffic generated by apps. Since smartphones currently perform uncoordinated packet transmissions which trigger unnecessary network signaling, the approach followed in this project is based on the aggregation of outgoing packets in order to have coordinated transmissions and thus reduce the network signaling. If the traffic generated by apps can be aggregated, the amount of network signaling messages can be reduced. This improves the battery runtime without harming the delivered service quality. This Win-Win-Win situation is represented in Figure 1.1.

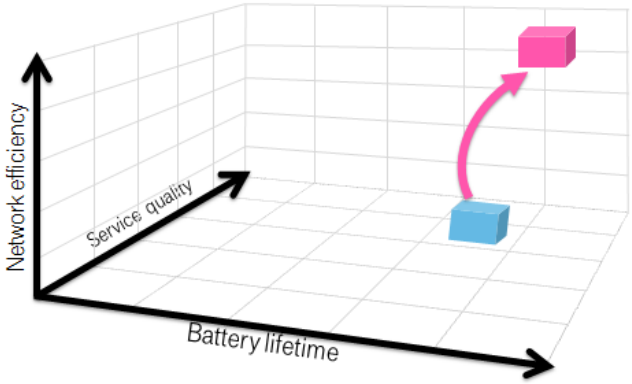

Figure 1.1: *Overall objective: Improving network efficiency.*

Additionally, this project aims to provide an Application Programming Interface (API) for developers. The framework proposed is interfacing to the hardware and to the application layer, which allows app developers to interact with these traffic control mechanisms in order to deliver services in the most efficient way providing the best quality service.

The project execution was divided in three phases which are briefly described below and explained in detail in Appendix A:

- Phase 1: Initial setup, design and implementation of the traffic control framework.

  - Setup of the development environment.
  - Design and implementation of the traffic control framework.

- Phase 2: Implementation of a custom version of Firefox OS.
  - Configuration of the kernel to enable Quality of Service (QoS) support.
  - Integration of a token-based traffic control module based on SoftToken protocol.
  - Integration of several command-line utilities.

- Phase 3: Evaluation and documentation.
  - Execution and interpretation of performance investigations for different types of services and QoS configurations of SoftToken vs. standard Wi-Fi and 3G.
  - Documentation.

The Firefox OS-based Geeksphone Keon has been used as development and testing device in this project.

1.3 Document structure

In this document, the concept and the implementation of a traffic control framework are presented. The work is done in the context of the Firefox OS framework. The rest of the document is organized as follows. Chapter 2 discusses the problem statement and summarizes the most relevant technologies involved in this project. Chapter 3 presents the traffic management approach and the traffic control framework, including details about the implementation, the functional architecture and the different components. Chapter 4 introduces some of the Key Performance Indicators (KPIs) to be used for evaluation and presents results of service quality laboratory measurements, while Chapter 5 concludes the document by summarizing the project.

Chapter 2

State of the Art

"I'd rather attempt to do something great and fail than to attempt to do nothing and succeed."

— *Robert H. Schuller*

The inefficient use of the wireless resources and the power resources of the mobile device is caused by the missing interaction between mobile devices, services and the network. This chapter introduces the problem statement in detail. Thereafter, a review of the most relevant technologies involved in this project is included.

The first technology addressed is Firefox OS, the operating system over which the whole project is based. Secondly, SoftToken protocol is introduced, which is the technology used to perform traffic management. The overall idea is presented together with the functional architecture and components which will be actively mentioned throughout the rest of the document. The review continues with wireless network technologies: Universal Mobile Telecommunications System (UMTS) and Wireless Local Area Network (WLAN). Thereafter, a description of JavaScript Object Notation (JSON), Node.js and MongoDB is presented. Finally, the chapter concludes with a review of the Firefox OS-based Geeksphone Keon, device used in this project for development and testing.

2.1 The smartphone challenge

At first, operators were pleased when they found that smart devices were, on average, generating only about one-tenth the data traffic that laptops were. Later, once smartphones became more popular and started to be used heavily in concentrated areas, customers in the U.S. and Europe began to experience a decrease in the quality of their voice and data services. After analyzing the network traffic it was found that smartphones were the origin of the problem. This result was caused by the existing difference between the behavior of laptops and smartphones in the network.

Initially, operators had optimized their wireless networks for browsing by keeping the data channel active as long as possible. This was done because it was presumed that laptop users would be the most frequent users of high-speed mobile data. This approach appeared to be ideal, avoiding repeated set-up delays every time the device reconnects – but prolonged time spent in the active data transmission mode consumes significant amounts of battery power and this constituted an important drawback, considering smartphones and their power limitations.

In comparison to smartphones, laptops generate a small amount of network signaling traffic because they tend to connect to the network and keep the connection active as long as possible. But smartphones, driven by popular applications such as social networking, email, online gaming and news reports that require constant updates, are constantly making and breaking connections, all of which generates a large amount of network signaling traffic. Handset manufacturers responded to this problem by introducing proprietary features to prolong battery life – but the same power-saving features ended up being a significant root cause of increased network signaling load. One of the proprietary software introduced was the Fast Dormancy mechanism.

In the 3rd Generation Partnership Project (3GPP) standard, the network determines what level of activity, or state, the handset is in. On the contrary, with Fast Dormancy, handset manufacturers enable the handset to determine its own state by forcing the network to release its data connection after downloading a piece of data, then the handset disconnects from the network and returns to the Idle[1] state.

The Figure 2.1 (Nokia Siemens Networks Smart Labs 2011) shows the data

[1]See Section 2.4 for information about the different activity states in UMTS.

growth (orange line) and network signaling growth (blue line) in a live network in Western Europe between December 2009 and July 2010 (Cell_PCH is not active). During the period, the data volume grew 65% and the network signaling volume grew 177%. The significantly higher network signaling growth rate is due to the high number of handsets with Fast Dormancy active in the network.

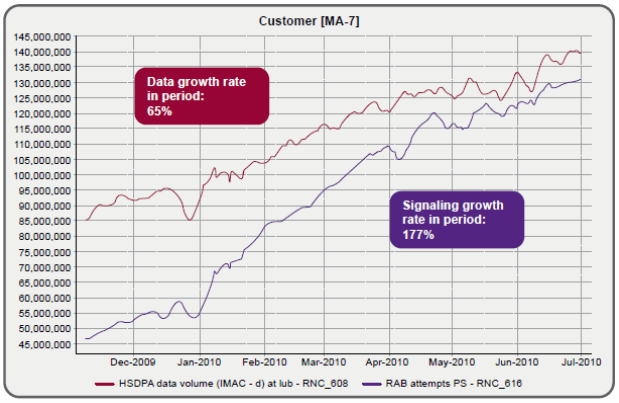

Figure 2.1: *Data and network signaling volume (Source: Nokia Siemens Networks Smart Labs 2011).*

Since handsets are no longer held for long periods of time in the Active state, the battery life is improved. The drawback is that if a smartphone wants to send more data, it must initiate a new connection from the Idle state and this greatly increases network signaling traffic. This network signaling increase leads to radio signaling storms which result in a lower number of served devices within an area.

State transitions are based on timers and buffer thresholds, the procedure takes time and causes network signaling. Furthermore, this behavior is strengthened by the introduction of Fast Dormancy mechanism. Handset manufacturers aim to save battery by putting the device in Idle state as much as possible. On the other hand, the mobile network operator aims to reduce network signaling by reducing state transitions, i.e., after a transmission, the device should stay in Cell_DCH state as long as possible.

The Figure 2.2 (Alcatel-Lucent 2012) shows how RRC states and state transitions influence end-user experience (i.e. packet delay and link capacity), energy consumption on the device and network signaling.

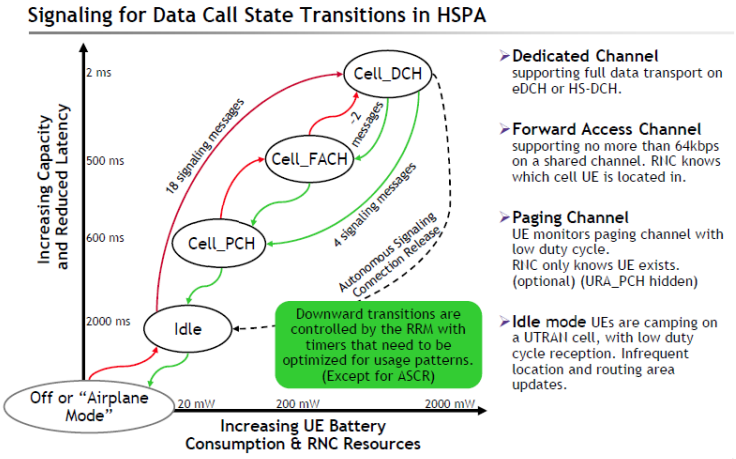

Figure 2.2: *Network signaling and battery consumption in HSPA (Source: Alcatel-Lucent 2012).*

2.2 Firefox OS

Firefox OS (also referred to as "FxOS", "Boot to Gecko" or by its codename "B2G") is Mozilla's open source mobile operating system based on Linux and Mozilla's Gecko technology [4]. Firefox OS is a mobile operating system that is free from proprietary technology and contains several architectural details which make it a powerful testbed and prototypical platform.

One of the main characteristics which make Firefox OS differ from the rest of mobile platforms is the fact that the entire user interface and applications are web apps with enhanced access to the mobile device's hardware and services.

Figure 2.3: *Firefox OS logo.*

Firefox OS is in itself – as depicted in Figure 2.4 (Mozilla Developer Network 2013) – a modular operating system, composed of three main layers (Gecko, Gaia and Gonk). In particular the latter separation from the Linux kernel might look familiar to those who know the architecture of the Android OS. This is a consequence of the design decision of the Firefox OS project team to share certain modules (i.e. the mobile Linux kernel) with the Android project. This makes it very easy to port the operating system to new terminals (end devices)

by simply using the same Linux kernel. In theory, all devices running Android are candidates for a relatively easy porting to Firefox OS since they use the same kernel and thus the same device drivers.

Gecko

Gecko is the cross-platform layout engine designed to support open standards: HyperText Markup Language, version 5 (HTML5) [5], Cascading Style Sheets (CSS) [6], and JavaScript [7]. Its function is to display web content and render it on user's screen or print it. Gecko includes also a complementary set of browser components but it does not package all of these components alongside other interface modules in a coherent, user-friendly application (including menus, toolbars, etc.), such as Firefox. This is what makes Gecko differ from a web browser.

Gaia

Gaia consists of a web app running on top of the Firefox OS software stack. Gaia forms the user interface of Firefox OS and it implements all the graphic services and applications (e.g. lock screen, home screen, standard applications, etc.). Gaia is implemented entirely in HTML5, CSS and JavaScript. It interfaces to the underlying operating system through open web APIs [8], which are implemented by the Gecko layer.

Gonk

Gonk is the lower level operating system of the Firefox OS platform. It consists of a Linux kernel and userspace Hardware Abstraction Layer (HAL). Only some parts of the HAL, such as GPS or camera, are shared with the Android project. The kernel and several of the user space libraries are common open-source projects (e.g. Linux, libusb, bluez, etc.). Gonk is a porting target of Gecko and this implies that developers can expose interfaces to Gecko that can not be exposed on other operating systems, since the Firefox OS project has full control over Gonk. For example, Gecko has direct access to the full telephony stack and display frame buffer on Gonk, but does not have this access on any other operating system.

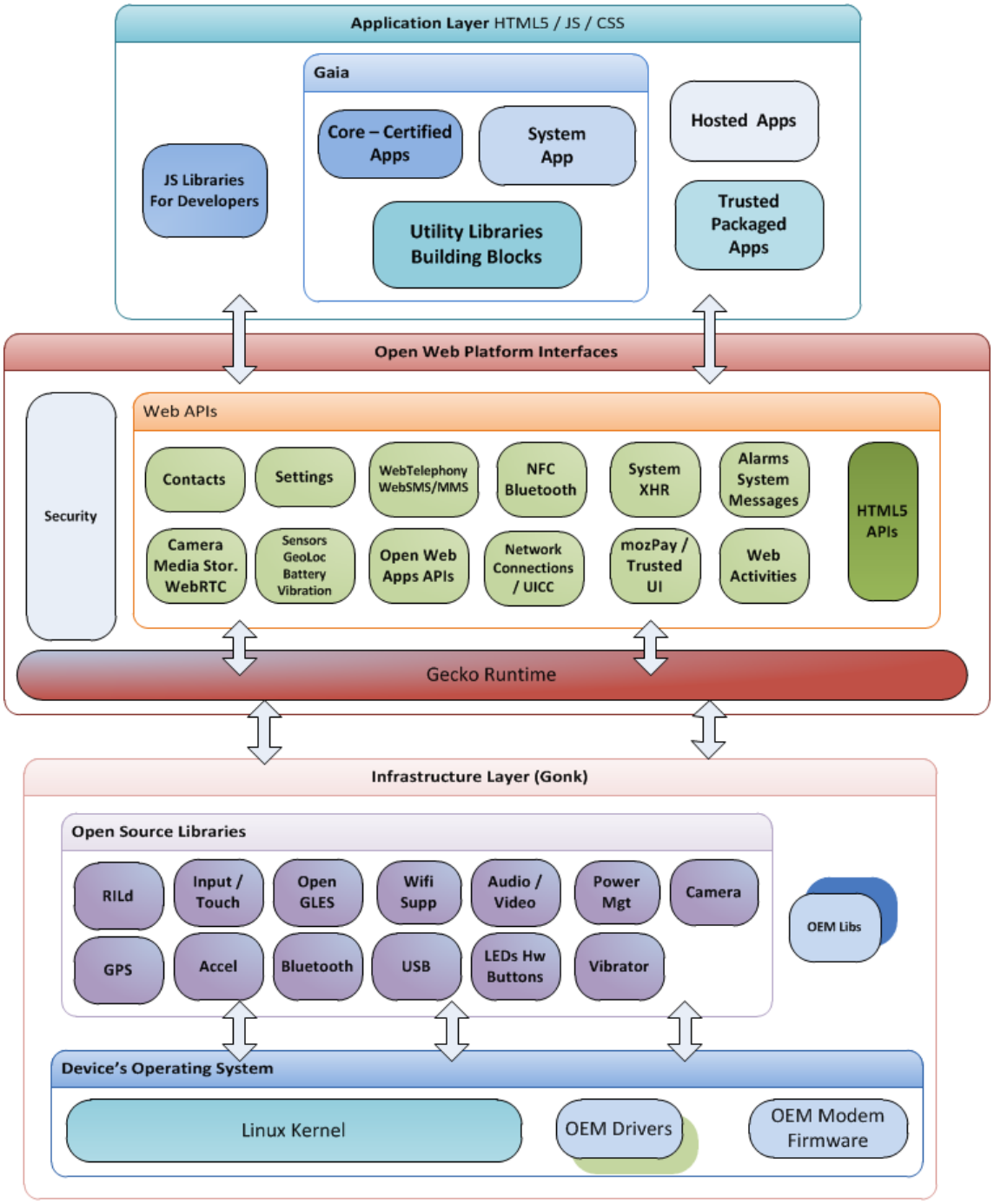

Figure 2.4: *Firefox OS architecture (Source: Mozilla Developer Network 2013).*

2.3 SoftToken Protocol

SoftToken is a Media Access Control (MAC)-layer extension for WLANs which aims to provide coordinated transmissions and avoiding collisions by using a token-passing mechanism. SoftToken protocol was implemented by Karl Thiel from T-Systems and developed by T-Labs [9].

There are two types of nodes in SoftToken, master and slaves. The master is the node responsible for coordinating transmissions of the other wireless nodes (slaves). The master node can be the Access Point (AP), in case of an infrastructure network, or any station, in the case of an ad-hoc network.

Once SoftToken is established, the master sends token request messages to slaves based on a given transmission schedule (i.e. round robin). The token request message signals the resource allocation and determines the amount and type of the data that can be transmitted at a time by a slave before it replies with a token response message. SoftToken maintains four separated queues at each slave and the outgoing packets are queued in their corresponding queues. Each queue can be associated to a different Traffic Class (TC). This scheduling mechanism permits traffic differentiation and provides QoS accordingly to the traffic class.

After completing its transmission, the slave returns the token by sending a token response message. In the token response message, the slave reports how much data it has transmitted (expressed in bytes or packets) and the amount of data remaining to be transmitted. After receiving a token response message, the master continues with the schedule by sending a token request message to the next slave or serves its own request.

SoftToken is implemented as a Linux kernel module. The user space includes the SoftToken Controller while the kernel space implementation consists of three modules: the SoftToken Client, the SoftToken Coordinator and the Scheduler. The SoftToken implementation is designed to work either in master or client (slave) mode. The Figure 2.5 represents the SoftToken architecture.

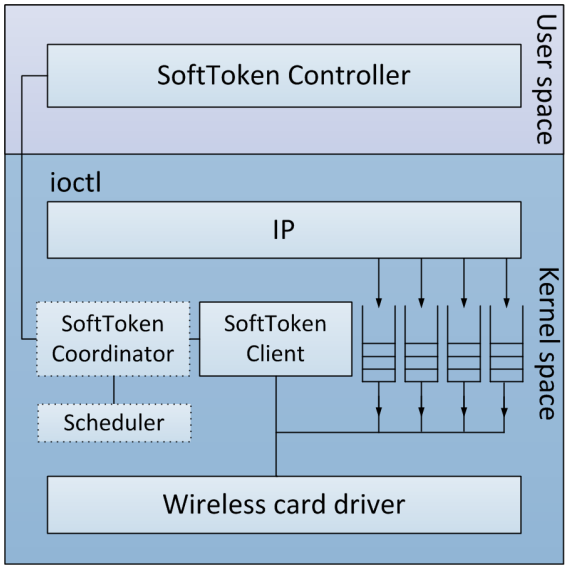

Figure 2.5: *SoftToken architecture.*

SoftToken Controller

The user space includes the SoftToken Controller module, which allows configuring and monitoring the SoftToken kernel module. It provides similar functionality to the *iwconfig* tool for wireless cards. Some of the functions of the controller are:

- Setting the SoftToken wireless interface.
- Getting the SoftToken interface and the scheduler configuration (only in master mode).
- Adding and removing slaves (only in master mode).
- Allocating bytes or packets to a queue (only in master mode).
- Assigning the traffic class of a queue (only in master mode).
- Setting the time the scheduler waits for scheduling the next node (only in master mode).
- Displaying the MAC address, IP address, enqueued packets and bytes, and the mode of the node running SoftToken.

SoftToken Client

The SoftToken Client module uses the Netfilter framework [10] to control packet traversal within the system.

When the SoftToken kernel module is inserted and configured to work as a slave (SoftToken Client), it registers the SoftToken device to the kernel, initializes certain node attributes such as the default resource allocation setting, and plugs into the Netfilter hook. Once the wireless interface is configured through the SoftToken Controller, SoftToken Client is ready to operate. In the slave mode, a node continuously waits for incoming or outgoing packets. SoftToken only interferes with packets destined to the configured SoftToken interface. Incoming packets are examined by the slave to check if they contain a SoftToken message from the master, else they are handled over the upper layer. In case of an outgoing packet, the TC field of its IP header is examined in order to enqueue it in the corresponding queue with the same TC value. Otherwise, the packet is enqueued in the default queue (queue 0).

SoftToken Coordinator

The SoftToken Coordinator module is mainly responsible for token-passing mechanism. This module maintains and manages a list of the slaves (i.e., slaves are deleted and added through this module). When a slave is added to the list of neighbors of the master, the master includes the slave to the schedule to send SoftToken management request. In order to transmit data, the master needs to schedule itself as well like the rest of slaves.

The SoftToken Coordinator also keeps track of resource allocations in order to determine the transmission schedule and reliability mechanisms to handle token losses. When the SoftToken module is inserted in master mode, in addition to the client functionality, it invokes the reliability mechanisms by initializing the retransmission timer and registering a call-back function for time-out handling.

Each token transmission starts the retransmission timer. If the SoftToken management response arrives on time, the master updates the statistics for the transmitted and enqueued packets for the corresponding slave. If the timer expires and no token response has been received from the slave, the token is assumed to be lost. On a timeout, a new token is generated and the schedule is continued. There may be the situation when the SoftToken management response arrives after the master has assumed that the token has been lost, in this case the master will ignore the

response and continue with the schedule.

SoftToken Scheduler

The SoftToken Scheduler determines the transmission schedule based on the type of traffic. The scheduler is configured manually in the master using the commands provided by the SoftToken Controller module. The scheduler configuration consists of:

- The number of the queue.
- The type of the resource allocation assigned to the queue, which can be: packets or bytes.
- The value of the resource allocation assigned to the queue.
- The traffic class assigned to the queue.

The configuration of the scheduler is signaled by the master to the slaves using the SoftToken management requests. In order to avoid conflicts between the different queues the scheduler also ensures that at a given time only one queue releases packets towards the wireless interface.

2.4 UMTS

UMTS is a term for the third generation radio technologies developed within 3GPP. UMTS is commonly known as 3G.

UMTS is the European vision of 3G mobile communication systems. It represents an evolution in terms of services and data speeds from today's Second Generation (2G) mobile networks. It adds performance over GSM by exploiting a wider radio band which provides high speed data channels.

UMTS uses Wideband Code Division Multiple Access (W-CDMA) radio access technology. W-CDMA is a spread-spectrum modulation technique which uses channels whose bandwidth is much greater than that of the data to be transferred. The modulation technique encodes each channel in such a way that a decoder, knowing the code, can pick out the wanted signal from other signals using the same band, which simply appear as so much noise.

Later, HSPA was introduced, giving substantially greater bit rates and improving packet-switched applications. HSPA is the combination of High Speed Uplink Packet Access (HSUPA) and High Speed Downlink Packet Access (HSDPA) [11].

UMTS specifies a complete network system, which includes the UEs, the RAN, the Core Network (CN) and the authentication of users via Subscriber Identity Module (SIM) cards. Any type of mobile handset can be counted as UE such as smartphones or USB data sticks. The connection between UE and CN is established by the RAN. It also implements the radio access technology. Within the RAN, the base station and the Radio Network Controller (RNC) are responsible for radio resource control, packet scheduling, handover control, etc. The RNC is the controller for a set of base stations that are connected to it. The CN is the backbone network which forwards the user data to external networks such as the Internet or the public switched telephone network. It also provides support for other additional functions such as billing, authentication, or location management.

In UMTS networks, the radio resources in the RAN between base station and UE are controlled and managed with the RRC protocol [12]. It offers services such as broadcast of network information, maintenance of a connection between the UE and RAN, establishment of point-to-point radio bearers for data transmission, QoS control, and reporting and cell selection management. In particular, RRC also participates in the coordination of other resource management operations such as channel measurements and handovers.

The protocol is divided into different parts: services for upper layers, communication with lower layers, protocol states, RRC procedures, and error control. All the RRC procedures rely on protocol states which are defined to trigger certain protocol actions for different situations. Typically, there are five RRC states characterizing a connection between UE and base station: Idle, URA_PCH, CELL_PCH, CELL_DCH, and CELL_FACH. The rest of this subsection concentrates on three of them and refers to them as Idle, DCH, and FACH.

If the UE is switched on and no connection to the mobile network is established, the UE is in Idle state. In contrast, if the UE wants to send data, radio resources are allocated by the base station for the handset and the UE will go into FACH or DCH state. A corresponding channel for data transmission is assigned to the UE. The FACH and the DCH state can be distinguished in that way that in DCH state a high-power dedicated channel for high speed transmission is allocated whereas in FACH state a shared access channel for general sporadic data transmission is used.

The possible transitions between the different states are defined by the network operator and the RRC protocol stack. Typically, the following state transitions are included: Idle → FACH, FACH → DCH to switch from lower radio resource utilization and low UE energy consumption to another state using more resources and energy, and DCH → FACH, FACH → Idle, DCH → Idle to switch to lower resource usage and energy consumption. The transitions are triggered by user activity and radio link control buffer level. A transition from DCH to FACH usually occurs when the buffer is empty and a threshold for a release timer is exceeded. The reverse direction is done if the buffer level exceeds a certain threshold value for a predefined time period. The UE goes into Idle state if the RNC detects overload in the network or no data was sent by the UE for a certain time.

The Figure 2.6 (Nokia 2010) shows the time delay and terminal energy consumption for the different states.

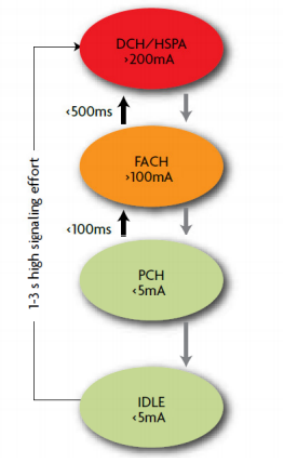

Figure 2.6: *HSPA state transitions (Source: Nokia 2010).*

The Table shows the amount of network signaling messages required for three different state transitions.

Transition	**Amount of messages**
Idle → DCH	~24-28
DCH → PCH	7
PCH → FACH	2

Table 2.1: *Amount of network signaling messages required for different state transitions in HSPA (Data from Signals Research Group 2010).*

2.5 WLAN

A WLAN links two or more devices using some wireless distribution method (typically spread-spectrum or Orthogonal Frequency-Division Multiplexing (OFDM) radio), and usually providing a connection through an AP to the Internet. Of the WLAN solutions that are available the Institute of Electrical and Electronics Engineers (IEEE) 802.11 standard, often termed Wi-Fi has become the de-facto standard.

Figure 2.7: *Wi-Fi logo.*

Wi-Fi is widely used as it gives users the ability to move around within a local coverage area and still be connected to the network. Furthermore, Wi-Fi provides cheap and convenient deployment of Local Area Networks (LANs). Wireless connectivity is now well established and virtually all new devices such as smartphones, tablets, some digital cameras, laptops, etc. contain a Wi-Fi capability.

IEEE 802.11 [13] is a set of MAC and physical layer (PHY) specifications for implementing WLAN computer communication in the 2.4, 3.6, 5 and 60 GHz frequency bands. They are created and maintained by the IEEE LAN/MAN Standards Committee (IEEE 802). The standards that are most widely known are the network bearer standards, 802.11a, 802.11b, 802.11g and 802.11n:

- 802.11a – Wireless network bearer operating in the 5 GHz ISM band with data rate up to 54 Mbps.
- 802.11b – Wireless network bearer operating in the 2.4 GHz ISM band with data rates up to 11 Mbps.
- 802.11g – Wireless network bearer operating in 2.4 GHz ISM band with data rates up to 54 Mbps.
- 802.11n – Wireless network bearer operating in the 2.4 and 5 GHz ISM bands with data rates up to 600 Mbps.

802.11b, 802.11g, and 802.11n-2.4 utilize the 2.4 – 2.5 GHz spectrum, one of the Industrial, Scientific and Medical (ISM) bands. 802.11a and 802.11n use the more heavily regulated 4.915 – 5.825 GHz band. These are commonly referred to as the "2.4 GHz and 5 GHz bands". Each spectrum is sub-divided into channels with a center frequency and bandwidth.

The 2.4 GHz band, as depicted in Figure 2.8 (Wikipedia 2014), is divided into 14 channels spaced 5 MHz apart, beginning with channel 1 which is centered on 2.412 GHz. The latter channels have additional restrictions or are unavailable for use in some regulatory domains.

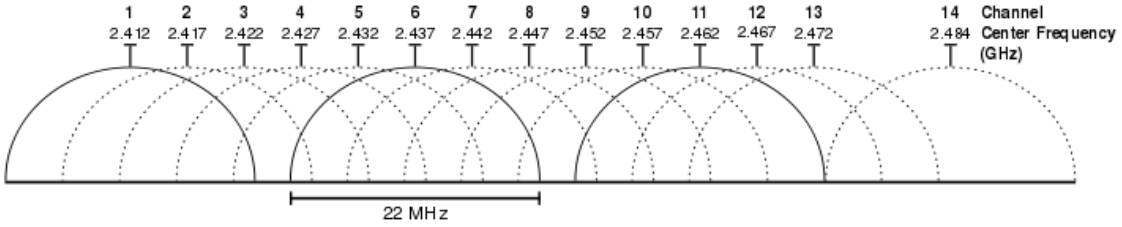

Figure 2.8: *Graphical representation of Wi-Fi channels in the 2.4 Ghz band (Source: Wikipedia 2014).*

Wi-Fi has adopted various encryption technologies over the time. The earlier protocols, such as Wired Equivalent Privacy (WEP) were proved easy to break. Nowadays Wi-Fi Protected Access 2 (WPA2) is the security protocol which is used to secure most wireless computer networks. This protocol introduces new features to overcome the vulnerabilities of previous protocols, such as Counter Mode CBC-MAC Protocol (CCMP), an enhanced data cryptographic encapsulation mechanism and a new encryption mode with strong security based on Advanced Encryption Standard (AES).

There are two types of WLAN networks that can be formed: infrastructure networks and ad-hoc networks. The infrastructure networks are aimed at office areas or to provide a "hotspot". A backbone wired network is required and is connected to a server. The wireless network is then split up into a number of cells, each one of them serviced by a base station or AP which acts as a controller for the cell. The range of the AP may be between 30 and 300 meters and it depends on the environment and the location of the AP. The ad-hoc networks are a decentralized type of wireless networks which do not rely on a pre existing infrastructure, such as routers in wired networks or APs in managed (infrastructure) wireless networks. Each node participates in routing by forwarding data for other nodes, so the determination of which nodes forward data is made dynamically on the basis of network connectivity. This type of network may be needed for instance when several people need to share data or to access a printer without the need for having to use wire connections.

2.6 JSON

JSON is an open standard format that uses human-readable text to transmit data objects consisting of attribute-value pairs [14]. It is used primarily to transmit data between a server and web application, as an alternative to eXtensible Markup Language (XML). The JSON format was originally specified by Douglas Crockford, and is described in RFC 4627 and ECMA-404.

Although originally derived from the JavaScript scripting language, JSON is a language-independent data format, and code for parsing and generating JSON data is readily available in a large variety of programming languages.

JSON's design goals were for it to be minimal, portable, textual, and a subset of JavaScript.

In general, the main advantages of JSON are:

- It is easy for humans to read and write and for machines to parse and generate.
- It is completely language independent.
- It is data-oriented and can be mapped more easily to object-oriented systems.

JSON is built on two structures:

- A collection of name/value pairs (e.g. object or struct).
- An ordered list of values (e.g. array or list).

The Figure 2.9 and Figure 2.10 show the representation of the two previous structures in JSON.

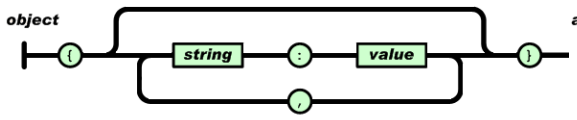

Figure 2.9: *Object representation.*

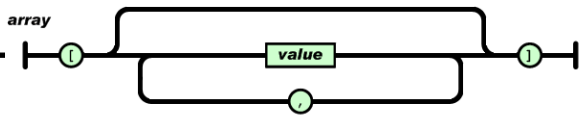

Figure 2.10: *Array representation.*

Since these are universal data structures for most programming languages, it makes sense that a data format that is interchangeable with programming languages is also based on these structures.

2.7 Node.js

Node.js is a software platform that is used to build scalable network (especially server-side) applications. Node.js utilizes JavaScript as its scripting language, and achieves high throughput via non-blocking I/O and a single-threaded event loop [15].

Node.js is a packaged compilation of Google's V8 JavaScript engine, the platform abstraction layer, and a core library, which is itself primarily written in JavaScript.

Figure 2.11: *Node.js logo.*

Node.js contains a built-in Hypertext Transfer Protocol (HTTP) server library, making it possible to run a web server without the use of external software, such as Apache or Lighttpd, and allowing more control of how the web server works.

In a typical Linux-Apache-MySQL-PHP (LAMP) server stack, an underlying Apache or Nginx web server exists, with PHP running on top of it. Each new connection to the server spawns a new thread, but creating new threads is costly. This works well with few connections, but as the number of users increases, it is very easy to quickly lose performance, being adding more servers the only way to support a large number of users [16]. It simply does not scale well. The Figure 2.12 shows how the performance is significantly reduced after 400 simultaneous connections.

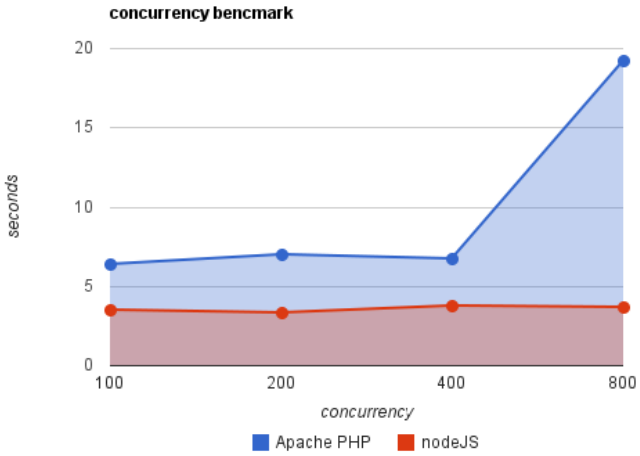

Figure 2.12: *Apache vs. Node.js. Concurrency benchmark (Source: http://code.google.com/p/node-js-vs-apache-php-benchmark/wiki/Tests 2010).*

In Node.js there is no Apache to listen for incoming connections and return HTTP status codes (it is up to the developer to handle the core server architecture, this can be done by using different modules provided).

JavaScript is an event-based language, which means that anything that happens on the server triggers a non-blocking event. Each new connection fires an event (e.g. data being received from an upload form or requesting data from the database). In practice, this means that Node.js will never lock up and can support tens of thousands of concurrent users.

With most server side scripting languages, the program has to wait whilst each function completes before going on to the next. With Node.js, the developer specifies functions that should be run on completion of something else, while the rest of the app moves on.

A Node.js application is implemented on a single thread. Node.js will create a thread on the background if there is a blocking operation (e.g. I/O) in the application, but it will not be done systematically for every connection as Apache would do. In theory, Node.js can handle as many connections as the maximum number of sockets supported by the system. In a UNIX system, this number is approximately 65.000 connections. However, in practice, this number depends on many factors, such as the amount of information that the application is sending to the clients. An application with a normal activity could handle around 25.000 clients without experiencing a decrease in the performance.

One of the drawbacks of Node.js is that it can only use one CPU, as it uses a single thread. However, there are options to bypass this limitation (e.g. by starting several Node.js instances on the server and add a load balancer in front of them).

2.8 MongoDB

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling [17].

Figure 2.13: *MongoDB logo.*

The data is stored in in the form of documents, which are JSON-like field and value pairs. MongoDB stores all documents in collections. A collection is a group of related documents that have a set of shared common indexes. Collections are analogous to a

table in relational databases.

MongoDB provides rich semantics for reading and manipulating data in addition to basic queries. For instance, it can return counts of the number of documents that match a query, or return the number of distinct values for a field. The number of possible search combinations is large and provides the user with flexibility for dealing with the data.

MongoDB focuses on: flexibility, power, speed, and ease of use:

Flexibility

MongoDB uses a modified version of the JSON format (BSON, or "Binary JSON") which is a binary representation of JSON with additional type information that allows the user to perform quick data searches. JSON provides a rich data model that seamlessly maps to native programming language types, and the dynamic schema makes it easier to evolve the data model.

Power

MongoDB provides a lot of the features of a traditional Relational Database Management System (RDBMS) (e.g. secondary indexes, dynamic queries, sorting, rich updates, upserts, and easy aggregation), with the flexibility and scaling capability that the non-relational model allows.

Speed/Scaling

MongoDB keeps related data together in documents, this way queries are much faster than in relational databases where data is separated into multiple tables and then needs to be joined later.

MongoDB also makes it easy to scale out the database. Autosharding allows the user to scale the cluster linearly by adding more machines.

Ease of use

MongoDB is easy to install, configure, maintain, and use. To this end, MongoDB provides few configuration options, and instead tries to automatically do the "right

thing" whenever possible. This means that developers can focus on developing their apps rather than spending a lot of time in tuning database configurations.

2.9 Firefox OS-based Geeksphone Keon

The Firefox OS-based Geeksphone Keon [18] is a short and compact developer preview device that has been created to be a testing bed for the nascent Firefox OS, itself in deep development.

Figure 2.14: *Keon – Front.*

Figure 2.15: *Keon – Back.*

The characteristics of this handset are listed below:

- CPU Qualcomm Snapdragon S1 7225AB 1Ghz.
- UMTS 2100/1900/850 (3G HSPA).
- GSM 850/900/1800/1900 (2G EDGE).
- Screen 3.5" HVGA Multitouch.
- Camera 3 MP.
- 4 GB (ROM) and 512 MB (RAM).
- MicroSD, Wifi N, Bluetooth 2.1 EDR, Radio FM, Light & Prox. Sensor, G-Sensor, GPS, MicroUSB.
- Battery 1580 mAh.

Chapter 3

Development

"The size of your success is measured by the strength of your desire; the size of your dream; and how you handle disappointment along the way."

— *Robert Kiyosaki*

This chapter introduces the concept and implementation of the traffic control framework. First, the packet aggregation approach is presented which is the mechanism used in this project to provide traffic management. Therafter, the chapter continues with an overview of the traffic control framework including implementation details and presenting the different components involved. In order to show the functionality of the framework several examples of messages and configurations are included. Furthermore, this chapter contains all the information regarding the extra features that have been added to the custom Firefox OS version.

The traffic control framework acts as an interface between the app traffic and the network. By this, the network operator gets control over the app traffic and is enabled to configure the packet transmission in order to achieve a better network resource usage by reducing the network signaling. The traffic control framework does not only provide benefits for the operator in terms of improved network efficiency but also improves battery runtime without harming the service quality.

3.1 Traffic control framework

This section introduces the concept of the traffic control framework. First, a review of queuing mechanisms in the Linux network stack is presented. Thereafter, the packet aggregation approach is introduced. This is the mechanism used in this project to perform traffic management with the goal to improve network efficiency by reducing network signaling.

3.1.1 Queuing in the Linux network stack

The driver queue lies between the IP stack and the Network Interface Controller (NIC). This queue typically is implemented as a First In, First Out (FIFO) ring buffer [19] which collects entering packets and dequeues them as quickly as the hardware can accept them. The driver queue treats all packets equally and has no capabilities for distinguishing between packets of different flows. This design keeps the NIC driver software simple and fast.

The driver queue does not contain packet data. Instead it consists of descriptors which point to other data structures called Socket Kernel Buffers (SKBs) [20] which hold the packet data and are used throughout the kernel.

The IP stack queues complete IP packets and is the input source for the driver queue. The packets may be generated locally or received on one NIC to be routed out another when the device is functioning as an IP router. Packets added to the driver queue by the IP stack are dequeued by the hardware driver and sent across a data bus to the NIC hardware for transmission. The Figure 3.1 (D. Siemon, Linux Journal 2013) shows the transmit path of the Linux network stack.

The driver queue gives the IP stack a location to queue data asynchronously from the operation of the hardware. This is done to ensure that whenever the system has data to transmit, the data is available to the NIC for immediate transmission. An alternative design would be for the NIC to ask the IP stack for data whenever the physical medium is ready to transmit; the drawback is that responding to this request cannot be instantaneous, so this design wastes valuable transmission opportunities resulting in lower throughput. The opposite approach would be for the IP stack to wait after a packet is created until the hardware is ready to transmit. This is also not ideal because the IP stack cannot move on to other work.

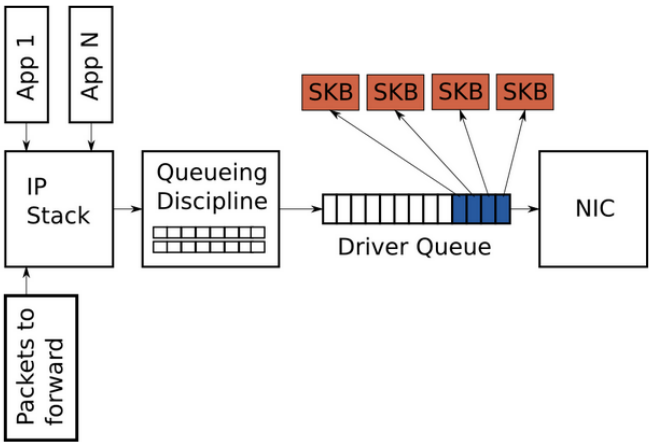

Figure 3.1: *Simplified overview of the transmit path of the Linux network stack (Source: D. Siemon, Linux Journal 2013).*

3.1.2 Packet aggregation approach

The reason why small app signaling messages (e.g. keep-alive messages, status updates, etc.) cause signaling storms is that these messages, coming from different apps, enter the transmission queue in irregular intervals, see Figure 3.2. This irregular arrival also means that these packets are transmitted via the wireless interface in irregular intervals which might cause an increased amount of network signaling messages in the control plane, e.g. because of transition state of the wireless interface changes from packet to packet [3] and because resources need to be reserved for a small amount of data.

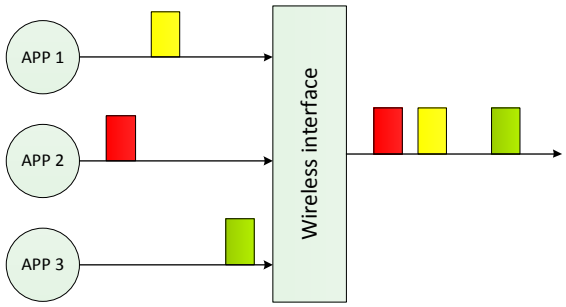

Figure 3.2: *Irregular transmission of app data or signaling packets.*

Different solutions to overcome this problem can be implemented. This document focuses on aggregation mechanisms meaning that data packets are queued for a configurable amount of time before they are forwarded to the wireless interface

for transmission. The aggregated packets are then sent out at once which means for instance that resources on the wireless link have to be reserved only once for a number of app signaling packets and not for each packet separately. By this the packet transmission will be bursty which will improve network efficiency as the amount of network signaling messages in the wireless network is minimized. Reduction of signaling overhead, namely minimizing the amount of radio resources consumed by signaling will implicitly result in a higher number of served devices within an area. In addition, lower signaling overhead will reduce the activity time and energy consumption within devices. An example is shown in Figure 3.3.

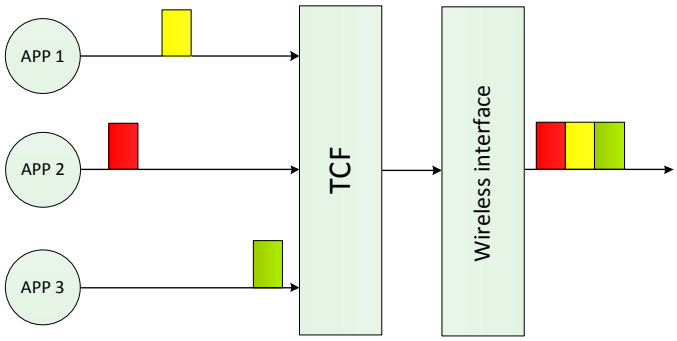

Figure 3.3: *Example of packet aggregation.*

However, it should be noted that the introduced aggregation delay (maximum amount of time that a packet is allowed to be delayed) also has influence on the service quality. For the best performance a trade-off between aggregation delay and service quality is required.

The Figure 3.4 depicts the mentioned approach which is the one used in this project.

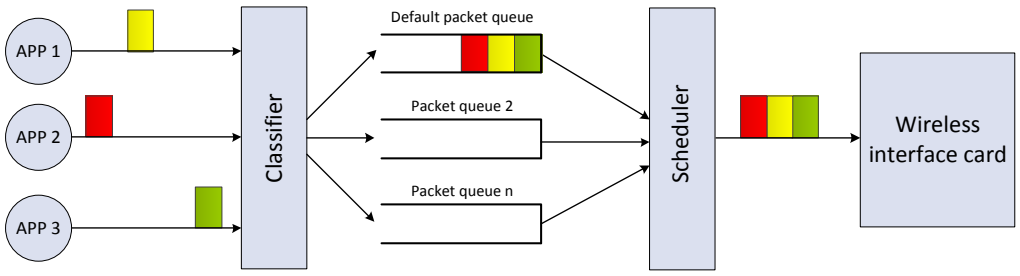

Figure 3.4: *Architecture of the new transmit path.*

3.2 Implementation

This section presents the traffic control framework from the implementation point of view and explains in detail the different components involved.

3.2.1 Overview

The framework - as depicted in Figure 3.5 - consists of six different components: Network Control app (Firefox OS app), control daemon, master daemon, SoftToken module, remote server and database. The first four components are implemented on the Firefox OS smartphone, while the server and the database are located on a remote machine. The framework runs on top of a custom kernel which includes several extras, being QoS support the most relevant one. Details about additional tools included in the custom Firefox OS version are explained in Subsection 3.2.7.

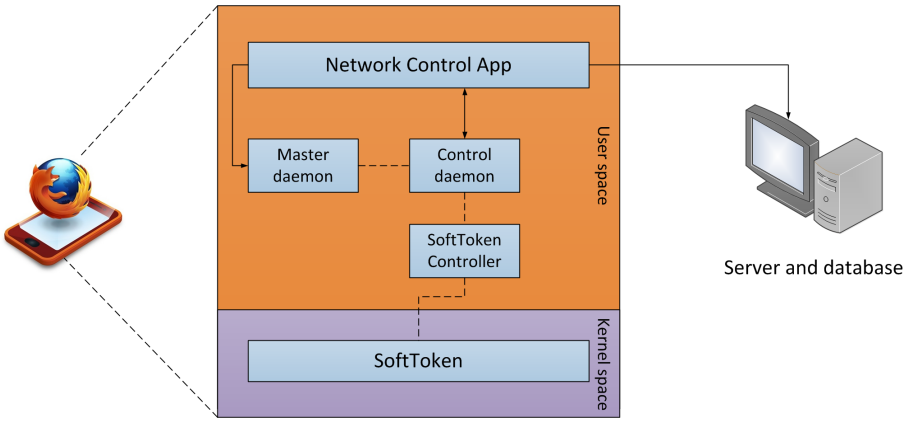

Figure 3.5: *Simplified architecture of the traffic control framework.*

The Network Control app provides a Graphic User Interface (GUI) to interact with the user. The GUI includes a commmand shell which enables the user to run implemented commands (e.g. requesting traffic statistics, starting or stopping packet aggregation, etc.).

The WebAPI provides basic information about the current active network connection (e.g. the rate or the number of transmitted and received bytes on a wireless interface). However, this information is not enough since the main goal of the framework is to enable packet control. Hence, information about packets is crucial. The need to fetch traffic information such as the number of transmitted and received

packets led to the implementation of a control daemon. The control daemon is located together with the rest of binaries of the UNIX operating system. The control daemon is connected to the Network Control app via a websocket [21] and runs in the background of the smartphone continuously. When the user sends the proper command, the daemon requests the operating system for traffic information (i.e. transmitted/received packets and transmitted/received bytes on the current active network interface[1]). Once the traffic information is retrieved, the control daemon computes extra statistics based on the previous information (i.e. average packet size and average inter-packet delay), builds a message containing all the data and sends it to the Network Control app and to the remote server for visualization and storage purposes. The communication link between the Network Control app and the remote server is done through a second websocket. In this project, all the traffic statistics are obtained by the control daemon. Details about what this data consists of and how it is retrieved are explained in Subsection 3.2.3.

Additionally, another websocket connects the master daemon to the app. The master daemon is started at boot and allows the user to start and stop the control daemon execution from the app.

The server and database have been implemented exclusively for demo purposes. The server listens constantly for incoming messages containing traffic statistics and displays this information on the shell output. In addition, the server includes a HTTP client that allows reading the received data on the browser in real time. The server is connected to a database which stores the messages as they are being received. The database contains rich semantics for reading and manipulating data. By this, the user can study traffic information in the future. The data can be filtered in a simple way in order to find out extra interesting information (e.g. the number of transmitted packets between two time intervals, the time elapsed between two numbers of packets, etc.).

Finally, the SoftToken module allows to perform packet aggregation prior to transmission. SoftToken is implemented as a Linux kernel module which can be configured by the SoftToken Controller, included in the user space. The control daemon (by request of the Network Control app) interacts with the SoftToken Controller in order to start, configure and stop packet aggregation.

[1]The current implementation provides information about the Wi-Fi and the 3G interfaces.

3.2.2 Firefox OS Network Control app

Like most Firefox OS apps, the Network Control app is built in HTML5, CSS, and JavaScript.

Its main functionality is to provide the user with a command shell, but it also includes other tools (i.e. a chart that plots in real time the transmitted and received packets on the current active network interface).

It integrates a simple GUI with two main buttons: "Terminal" and "Chart".

The "Terminal" button displays a command shell interface where the implemented commands can be executed (see Figure 3.6). The Table 3.1 contains all the commands that have been implemented and can be executed.

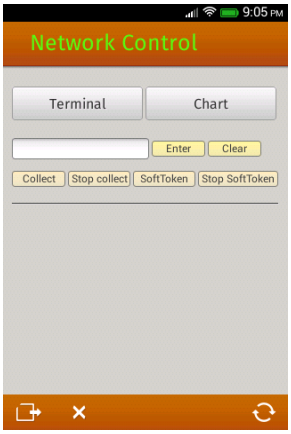

Figure 3.6: *Shell interface.*

To make using the app easier, a set of buttons has been added. The user can either type the command on the text box or press the corresponding button. Due to space restrictions on the smartphone screen only the most frequently used commands include a button.

Command	Description
ip	It is equivalent to run ifconfig wlan0.
date	It outputs the current system date and time.
httpdemo	It starts a file download via HTTP (for demos).
ftpdemo	It starts a file download via File Transfer Protocol (FTP) (for demos).
stat	It outputs traffic statistics of the current active interface.
collect	It collects and sends traffic statistics to the server.
stop	It stops the collection and transmission of traffic statistics.
pktaggr	It prompts several windows to start and configure packet aggregation.
stoppktaggr	It stops packet aggregation.

Table 3.1: *List of implemented commands.*

In order to work, the app needs to have a permanent connection with the control daemon. Two buttons on the footer allow the user to start or stop the execution of

the control daemon. This is done through the master daemon which is spawned by the *init* process (see Subsection 3.2.4). The color of the title indicates the status of such connection. When the title is coloured green the connection is established (see Figure 3.7), while red indicates a connection error (see Figure 3.8).

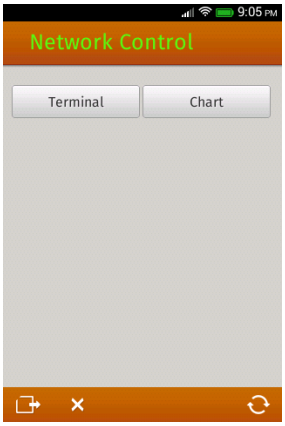

Figure 3.7: *Control daemon running.*

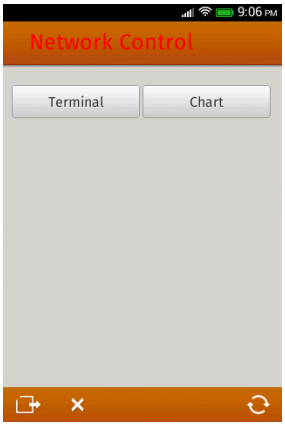

Figure 3.8: *Control daemon stopped.*

The "Chart" button allows the user to visualize a chart which represents in real time the transmitted and received packets on the current active network interface. The Figure 3.9 shows the transmitted packets (orange) and received packets (blue) for a test FTP download via Wi-Fi.

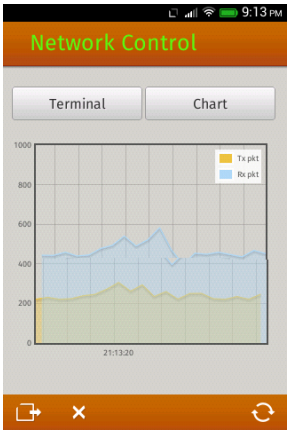

Figure 3.9: *FTP download via Wi-Fi.*

The Network Control app can be executed in the background while other apps are running. By this, it is possible to study with a quick look how different proto-

cols (e.g. HTTP, FTP), apps (e.g. VoIP clients, social networking apps) and QoS configurations affect the traffic. The Flot [22] JavaScript library for jQuery [23] has been used for the implementation of the chart.

Collecting statistics and sending them to the server

The *collect* button prompts two textboxes for setting two intervals. The first one indicates the amount of time to compute statistics. This is done since there may be situations when it would be suitable to delay or speed up the computation of statistics. The second interval indicates the amount of time to send the statistics to the server. This is necessary in order to avoid creating too much overload when requesting traffic information. For instance, if the control daemon would send the traffic statistics every second, it would be creating overload traffic implicitly. Hence, the larger this time interval is, the less overload the daemon will create when sending messages.

3.2.3 Control daemon

The control daemon is built entirely in C and two external libraries have been used for its implementation, these are: libwebsockets [24] and jansson [25]. The first one is a pure C library that allows implementing websockets in C. It is built to use minimal CPU and memory resources, and provide fast throughput in both directions. The second, jansson, is a C library for encoding, decoding and manipulating JSON data.

The main functionality of the control daemon is to communicate the Network Control app with the operating system and the remote server. In order to do this, two websockets have been implemented. The first one connects the app to the control daemon while the second websocket connects the app to the remote server. Even though the Websocket protocol allows setting SSL support, none of the websockets implement this feature, since it was not considered to be relevant for a demo version.

The initial implementation required that the smartphone is connected via USB to the PC in order to execute the daemon. This was done by starting a remote shell with *adb* [26]. Once the connection with the phone was established the daemon could be started as shown below:

```
daemon_netcontrol <server address | -d[efault]>
```

Where the first parameter indicates the IP address of the server that the control daemon connects to. The default option -d connects the control daemon to the default server located at T-Labs.

This was a provisional step, since the project was at a very early development stage. The implementation in its current version contains a modified *init* program that starts a second daemon (i.e. master daemon) at boot and allows the user to start or stop the control daemon execution from the app. This approach simplifies the daemon execution, since connecting the phone to the PC in order to run the full app is no longer needed.

The daemon acts as both server and client.

- **Server side**: The daemon listens on localhost for incoming connections from the app in order to communicate it with the operating system and fetching traffic statistics, as well as for executing other implemented commands.
- **Client side**: The daemon is connected via Wi-Fi to the remote server for sending useful data for visualization and storage purposes (i.e. traffic statistics).

Message format

The messages containing commands and responses are built accordingly to the JSON specification.

Below, two messages exchanged to communicate the control daemon with the server are shown. The Listing 3.1 shows a message sent by the app to the daemon in order to start collecting and sending statistics to the server. The Listing 3.2 shows a message sent by the daemon to the server containing traffic statistics and other information in response to *collect* command.

```
{
"type":"command", //type of message
"data":"collect"  //command
}
```

Listing 3.1: Command message

```
{
"type":"response-network-status", //type of message
"data": //traffic statistics
{
"txp": 136681, //transmitted packets
"rxp": 260941, //received packets
"txb": 10800875, //transmitted bytes
"rxb": 376285988, //received bytes
"avg_pkt_size": 79.02, //average packet size [B]
"avg_pkt_delay": 0.001 //average interpacket delay [s]
},
"time": 1385480728, //timestamp
"id": "88:f4:88:71:62:46" //device id [MAC address]
}
```

Listing 3.2: Response message

3.2.4 Master daemon

The master daemon is also built in C and is a simplified version of the control daemon. It is started at boot and connects to the app via a websocket. This daemon listens constantly for incoming connections from the app and is responsible for starting and stopping the execution of the control daemon at user request.

In Firefox OS, the init.b2g.rc file is responsible for providing machine-specific initialization instructions in Firefox OS. The modification of this file was necessary in order to start the master daemon execution at boot.

The Figure 3.10 shows the final Firefox OS userspace architecture with the master daemon included as one more process spawned by *init*.

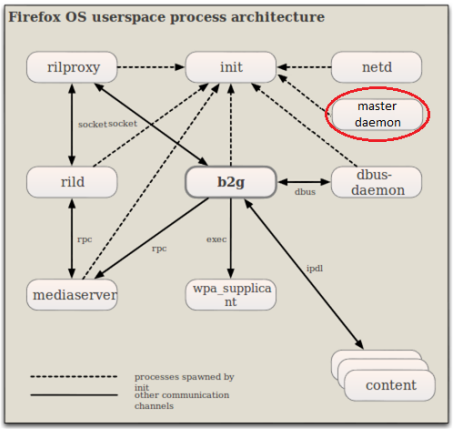

Figure 3.10: *Firefox OS userspace process with the master daemon (Adapted from Mozilla Developer Network 2013).*

3.2.5 Server and database

The server is built in Node.js, while the database is built in MongoDB. These two technologies provide several advantages and are easy to combine with each other since MongoDB provides JavaScript support.

They have been implemented for visualizing and storing traffic statistics. However, they are part of the demo version and will not be used in a commercial deployment.

The server listens constantly for incoming messages containing traffic statistics and stores them into the database. In the current implementation, the server is connected to the daemon via Wi-Fi. In order to keep the implementation simple and since the functionality of the server is merely displaying and storing traffic messages, no message is sent from the server to the daemon.

The Listing 3.3 shows the server output which includes the messages received during the app execution. The computing interval was set to 1 second and the interval for sending messages to the server was set to 3 seconds.

```
Fri Jan 17 2014 13:39:39 GMT+0100 (CET) Server is listening on port 3000
Fri Jan 17 2014 13:39:44 GMT+0100 (CET) Connection from origin 88:f4:88:71:62:46.
Fri Jan 17 2014 13:39:44 GMT+0100 (CET) Connection accepted.
>> Message: 1
txp [0]: 2879
rxp [0]: 7873
txb [0]: 161049
rxb [0]: 4419702
time [0]: 316209700
id/mac [0]:  88:f4:88:71:62:46
txp [1]: 2879
rxp [1]: 7873
txb [1]: 161049
rxb [1]: 4419702
time [1]: 316209701
id/mac [1]:  88:f4:88:71:62:46
txp [2]: 2879
rxp [2]: 7873
txb [2]: 161049
rxb [2]: 4419702
time [2]: 316209702
id/mac [2]:  88:f4:88:71:62:46
>> End of message: 1
>> Data saved @ /var/lib/mongodb
>> Message: 2
txp [0]: 2880
rxp [0]: 7874
txb [0]: 161776
rxb [0]: 4419782
```

```
time [0]: 316209703
id/mac [0]:  88:f4:88:71:62:46
txp [1]: 2880
rxp [1]: 7874
txb [1]: 161776
rxb [1]: 4419782
time [1]: 316209704
id/mac [1]:  88:f4:88:71:62:46
txp [2]: 2881
rxp [2]: 7875
txb [2]: 161782
rxb [2]: 4420025
time [2]: 316209705
id/mac [2]:  88:f4:88:71:62:46
>> End of message: 2
>> Data saved @ /var/lib/mongodb
```

Listing 3.3: Server output

As explained in Section 2.8, the user is provided with a wide set of search options that makes the data selection easy. An example of use is shown in Listing 3.4, where a search was made based on all the values of the attribute "txp" greater than 12.000 bytes. Thereafter, the information was filtered to display the "time" and "id/mac" attributes.

```
MongoDB shell version: 2.4.8
connecting to: netcontrolDB
> db.statistics.find(  { txp: { $gt: 12000 } } , { txb: 0, rxp: 0, rxb: 0, _id:0 } )
{ "txp" : 12047, "time" : 1386776642, "id/mac" : "88:f4:88:71:62:46" }
{ "txp" : 12234, "time" : 1386776644, "id/mac" : "88:f4:88:71:62:46" }
{ "txp" : 12443, "time" : 1386776644, "id/mac" : "88:f4:88:71:62:46" }
{ "txp" : 12692, "time" : 1386776645, "id/mac" : "88:f4:88:71:62:46" }
{ "txp" : 12783, "time" : 1386776646, "id/mac" : "88:f4:88:71:62:46" }
{ "txp" : 12783, "time" : 1386776647, "id/mac" : "88:f4:88:71:62:46" }
{ "txp" : 12784, "time" : 1386776649, "id/mac" : "88:f4:88:71:62:46" }
{ "txp" : 12784, "time" : 1386776649, "id/mac" : "88:f4:88:71:62:46" }
{ "txp" : 12785, "time" : 1386776650, "id/mac" : "88:f4:88:71:62:46" }
{ "txp" : 12785, "time" : 1386776651, "id/mac" : "88:f4:88:71:62:46" }
{ "txp" : 12785, "time" : 1386776652, "id/mac" : "88:f4:88:71:62:46" }
{ "txp" : 12786, "time" : 1386776654, "id/mac" : "88:f4:88:71:62:46" }
{ "txp" : 12786, "time" : 1386776654, "id/mac" : "88:f4:88:71:62:46" }
```

Listing 3.4: Database search

3.2.6 Integration of the packet aggregation module

The SoftToken module, presented in Section 2.3, is the mechanism used in this project to achieve packet aggregation.

SoftToken protocol has been found to be interesting for this project since it can be used to perform traffic management by queuing packets prior to transmission. In order to do that, the master and slave nodes need to be implemented and correctly configured in the same device (the Keon smartphone in this case).

Initially, the given SoftToken implementation was built to be loaded on a Linux machine running a kernel 2.6. Since the Keon smartphone run a newer kernel on a different platform (ARM), it was necessary to cross-compile SoftToken to make it compatible with the current architecture and kernel (3.0.8-GP+).

In order to make easier setting QoS configurations from the app, a shell script file was implemented. The SoftToken module is automatically loaded the first time the script is executed. The script accepts four parameters, the usage is listed below:

```
usage: softtokenctrl <iface> <queue> <pkts_queue> <sched_delay>
```

Where the parameters are:

- <iface>: The interface to set packet aggregation.
- <queue>: The queue to use (there are four queues in the current implementation).
- <pkts_queue>: The number of packets to be allocated in the selected queue.
- <sched_delay>: The scheduler delay or "aggregation delay" which defines the amount of time that the packets will stay enqueued before being transmitted.

To set the different parameters, the script makes use of the SoftToken Controller.

The basic idea is to select a wireless interface to manage. Thereafter, a specific queue can be selected, otherwise the packet aggregation will be done in the default queue (queue 0). Then the user can select the amount of packets to be allocated in the queue before they are transmitted over the selected wireless interface. The configurable amount of time that the packets will stay enqueued before being transmitted is set by the scheduler delay. These two last parameters are the most relevant ones to perform packet aggregation.

The Listing 3.5 shows a SoftToken configuration for the Wi-Fi interface (i.e. wlan0) of the Keon device, where the default queue allocates 5 packets and the scheduler delay is set to be 3 seconds. Notice that the MAC addresses are the same in both the master and the slave, since they are implemented on the same device.

```
Interface: wlan0
Queue Limit: 1500 packets
Token timeout: 2000 ms
Scheduler configuration:
queue: 0, 5 packets, tos: 0, tc: 0
queue: 1, 10 packets, tos: 1, tc: 1
queue: 2, 10 packets, tos: 2, tc: 2
queue: 3, 10 packets, tos: 3, tc: 3
Scheduler delay: 3000 ms after every turn
Interface wlan0 has 2 neighbors
mac address       ip address          packets                   bytes  comment
                                    0  1  2  3     0     1     2     3
-------------------------------------------------------------------------------
88:f4:88:71:62:46 192.168.0.17    0  0  0  0     0     0     0     0  MASTER
88:f4:88:71:62:46 0.0.0.0         0  0  0  0     0     0     0     0  slave
-------------------------------------------------------------------------------
```

Listing 3.5: SoftToken configuration example for wlan0

The Listing 3.6 shows another SoftToken configuration example but in this case the 3G interface (i.e. rmnet0) was used.

```
Interface: rmnet0
Queue Limit: 1500 packets
Token timeout: 2000 ms
Scheduler configuration:
queue: 0, 8 packets, tos: 0, tc: 0
queue: 1, 10 packets, tos: 1, tc: 1
queue: 2, 10 packets, tos: 2, tc: 2
queue: 3, 10 packets, tos: 3, tc: 3
Scheduler delay: 5000 ms after every turn
Interface rmnet0 has 2 neighbors
mac address       ip address          packets                   bytes  comment
                                    0  1  2  3     0     1     2     3
-------------------------------------------------------------------------------
9e:03:88:85:90:af 10.154.29.167   0  0  0  0     0     0     0     0  MASTER
9e:03:88:85:90:af 10.154.29.167   0  0  0  0     0     0     0     0  slave
-------------------------------------------------------------------------------
```

Listing 3.6: SoftToken configuration example for rmnet0

The wireless interface to configure (i.e. wlan0 for Wi-Fi or rmnet0 for 3G) must be enabled before running SoftToken.

The Figure 3.11 shows in a graphic way the interaction between the implemented traffic control framework and the SoftToken module.

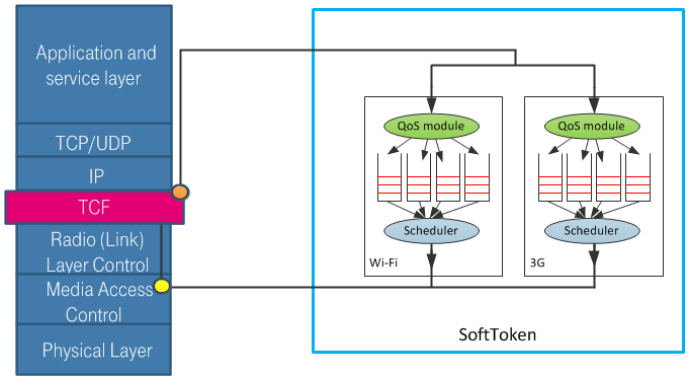

Figure 3.11: *Traffic control framework and SoftToken module.*

Packet aggregation example

In order to test the packet aggregation mechanism the command *netcat* was used to create a client-server connection. The server was set on a PC and the Keon acted as client which sent every second an UDP packet containing the current date.

The Figure 3.12 shows a traffic capture with the standard behavior (i.e. without SoftToken).

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
2	1.049563	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3	2.069744	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
4	3.119787	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
5	4.169352	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
6	5.190627	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
11	6.239429	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
16	7.259194	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
21	8.309070	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
22	9.359150	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
23	10.408720	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
24	11.428804	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
25	12.934876	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
26	13.765814	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
27	14.549489	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
28	15.602942	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
29	16.618248	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
30	17.671851	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
33	18.717851	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
36	19.737937	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
37	20.789563	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)

Figure 3.12: *Capture on wlan0 interface without SoftToken.*

As expected, the capture shows how UPD packets are received in the PC every second.

Thereafter, SoftToken module was inserted and configured with different delays. The Figure 3.13 shows the result.

No.	Time	Source	Destination	Protocol	Length	Info
3904	573.601082	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3905	573.601336	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3906	573.601668	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3908	576.620618	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3909	576.621011	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3910	576.621214	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3911	579.638235	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3912	579.638591	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3913	579.639028	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3914	582.659457	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3915	582.659993	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3916	582.660527	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3917	587.678965	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3918	587.679533	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3919	587.680267	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3920	587.680636	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3921	592.701081	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3922	592.701313	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3923	592.701522	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3924	592.702361	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)
3925	592.702573	172.16.62.136	172.16.62.93	ENTTEC	71	Unknown (0x54756520)

Figure 3.13: *Capture on wlan0 interface with SoftToken.*

UDP packets with sequence number in the range between 3904 and 3916 were aggregated and released after a delay of 3 seconds, then the delay was set to 4 seconds and finally to 5 seconds.

3.2.7 Integration of command-line utilities

Several command-line utilities have been cross-compiled for the ARM architecture and included in the custom Firefox OS version. This was needed mainly for two reasons. First, some of the programs included in the original Firefox OS version did not provide a full functionality (e.g. *wget* program did not include several options that would be required later to perform service quality measurements). Second, Linux offers several programs that are interesting and useful for this project but are not included in the original Firefox OS version. This subsection provides information about the most relevant programs that have been included: *tc*, *tcpdump*, *netcat* and *iperf*.

Traffic Control

tc command is part of the Linux *iproute2* package and consists of a set of tools that allow the user to provide QoS on a networked device. However, the integrated version contains limited functionality, for instance, it only provides support for Hierarchical Token Bucket (HTB) queues. Despite of its limitations, the *tc* command provides and interesting alternative and could be of great help for future research. This is the reason why it has been included in the custom Firefox OS version.

The usage is shown below:

```
Usage: tc [ OPTIONS ] OBJECT { COMMAND | help }
       tc [-force]
where  OBJECT := { qdisc | class | filter | action | monitor }
       OPTIONS := { -s[tatistics] | -d[etails] | -r[aw] | -p[retty] | -b[atch]
       [filename] }
```

Listing 3.7: tc usage

Wireless packet capturing

Capturing packets on the wireless interfaces of the smartphone is an interesting functionality for the purpose of this project and a way to do that is provided here. *tcpdump* is itself a powerful command-line packet analyzer. However, Wireshark provides extra functionalities integrated in an attractive GUI.

netcat and *tcpdump* can be combined in such a way that Wireshark can be executed on a PC to capture traffic on the smartphone. The commands for that purpose are shown below:

```
# adb shell "tcpdump -n -s 0 -w - | nc -l -p 11233"
  tcpdump: listening on wlan0, link-type EN10MB (Ethernet), capture size 65535 bytes
```

The interface to capture will be the one that is active at the moment, in this case it is wlan0. An example with rmnet0 is shown below:

```
# adb shell "tcpdump -n -s 0 -w - | nc -l -p 11233"
  tcpdump: WARNING: arptype 530 not supported by libpcap - falling back to cooked
  socket
  tcpdump: listening on rmnet0, link-type LINUX_SLL (Linux cooked), capture size
  65535 bytes
```

```
# adb forward tcp:11233 tcp:11233 && nc 127.0.0.1 11233 | wireshark -k -S -i -
```

After that, and as depicted in Figure 3.14, Wireshark will start and will be ready to start sniffing.

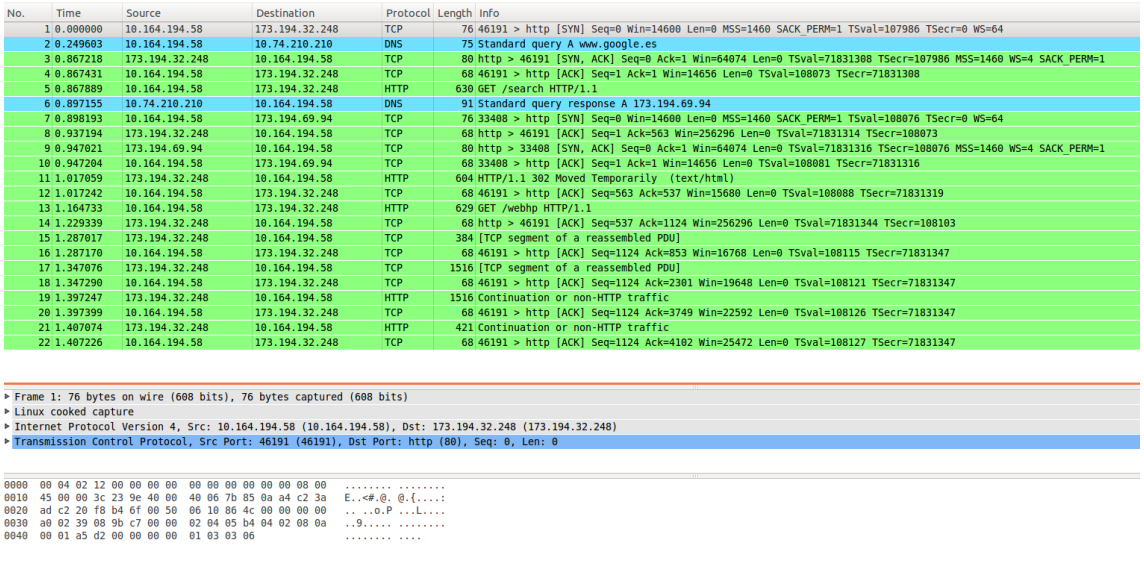

Figure 3.14: *Wireshark capture on rmnet0 interface.*

Network performance measurement

iperf is a tool for network performance measurement. It has a client and server functionality and can create TCP and UDP data streams, allowing the tunning of various parameters for testing a network. *iperf* reports badnwidth, delay jitter and datagram loss. All these characteristics make of *iperf* a very interesting tool for simulation and testing purposes.

The usage is shown below:

```
Usage: iperf [-s|-c host] [options]
       iperf [-h|--help] [-v|--version]

Client/Server:
  -f, --format    [kmKM]   format to report: Kbits, Mbits, KBytes, MBytes
  -i, --interval  #        seconds between periodic bandwidth reports
  -l, --len       #[KM]    length of buffer to read or write (default 8 KB)
  -m, --print_mss          print TCP maximum segment size (MTU - TCP/IP header)
  -o, --output    <filename> output the report or error message to this specified
                           file
```

```
  -p, --port      #         server port to listen on/connect to
  -u, --udp                 use UDP rather than TCP
  -w, --window    #[KM]     TCP window size (socket buffer size)
  -B, --bind      <host>    bind to <host>, an interface or multicast address
  -C, --compatibility       for use with older versions does not sent extra msgs
  -M, --mss       #         set TCP maximum segment size (MTU - 40 bytes)
  -N, --nodelay             set TCP no delay, disabling Nagle's Algorithm
  -V, --IPv6Version         Set the domain to IPv6

Server specific:
  -s, --server              run in server mode
  -U, --single_udp          run in single threaded UDP mode
  -D, --daemon              run the server as a daemon

Client specific:
  -b, --bandwidth #[KM]     for UDP, bandwidth to send at in bits/sec
                            (default 1 Mbit/sec, implies -u)
  -c, --client    <host>    run in client mode, connecting to <host>
  -d, --dualtest            Do a bidirectional test simultaneously
  -n, --num       #[KM]     number of bytes to transmit (instead of -t)
  -r, --tradeoff            Do a bidirectional test individually
  -t, --time      #         time in seconds to transmit for (default 10 secs)
  -F, --fileinput <name>    input the data to be transmitted from a file
  -I, --stdin               input the data to be transmitted from stdin
  -L, --listenport #        port to receive bidirectional tests back on
  -P, --parallel  #         number of parallel client threads to run
  -T, --ttl       #         time-to-live, for multicast (default 1)
  -Z, --linux-congestion <algo>  set TCP congestion control algorithm (Linux only)

Miscellaneous:
  -x, --reportexclude [CDMSV]   exclude C(connection) D(data) M(multicast)
                            S(settings) V(server) reports
  -y, --reportstyle C       report as a Comma-Separated Values
  -h, --help                print this message and quit
  -v, --version             print version information and quit

[KM] Indicates options that support a K or M suffix for kilo- or mega-

The TCP window size option can be set by the environment variable
TCP_WINDOW_SIZE. Most other options can be set by an environment variable
IPERF_<long option name>, such as IPERF_BANDWIDTH.
```

Listing 3.8: iperf usage

This program can have different utilities. For instance, VoIP can be simulated by running *iperf* in client mode and setting a packet size of 200 bytes and a packet interval of 100 ms.

Chapter 4

Evaluation

"Aim for the moon. If you miss, you may hit a star."

— *W. Clement Stone*

This chapter presents some of the KPIs that can be used to proof that the traffic control framework has a positive influence on service quality, network efficiency and battery runtime. Since performance measurements are not part of the project objectives, only some laboratory results of service quality measurements are included. These measurements allow to compare a standard device with a traffic control-enabled device under defined conditions[1].

For the best performance a trade-off between service quality, network efficiency and battery runtime is required. Hence, future research should take the given service quality results into account when performing measurements on network efficiency and battery runtime.

4.1 Key Performance Indicators

The purpose of these KPIs is to establish the theoretical basis to perform measurements in order to evaluate the influence of the traffic control framework on service quality, network efficiency and battery runtime.

[1] In collaboration with Dr. Nico Bayer and Roman Szczepanski.

4.1.1 Battery runtime

Battery runtime (B) depends on the battery capacity (C), the power consumption (P) of the device and the average amount of activity (A).

$$B = \frac{C}{P * A} \tag{4.1}$$

The power consumption of the device is the most relevant element here and it depends on many factors (e.g. hardware components, usage, communications interfaces, etc.).

However, power consumption of the device can be expressed in a generic way in terms of energy per information bit (λ_i) which is the most widely accepted metric for energy efficiency.

$$\lambda_i = \frac{E}{I} = \frac{P}{R} \quad in \left[\frac{J}{bit}\right] or \left[\frac{W}{bps}\right] \tag{4.2}$$

Where:

- E is the consumed energy in a given observation period T with consumed power P.
- I is the information volume with rate R.

4.1.2 Network efficiency

Network efficiency is expressed in terms of signaling overhead which is defined as the ratio between the amount of signaling information that is transmitted (e.g. for bearer activation or resource reservation) and the total amount of information that is transmitted (i.e. signaling plus payload).

Reduction of signaling overhead, namely minimizing the amount of radio resources consumed by signaling will implicitly result in a higher number of served devices within an area. In addition, lower signaling overhead will reduce the activity time and energy consumption within devices.

For the present measurements signaling overhead (O_S) is defined as follows:

$$O_S = 1 - PE = 1 - \frac{N_D}{N_D + N_S} \tag{4.3}$$

Where:

- N_D is the number of data bits.
- N_S is the number of signaling bits.

4.1.3 Service quality

Evaluation of the service quality is individual for each service and difficult to define. Relevant parameters which influence the service quality are recorded during the service usage (e.g. bandwidth, delay, error rate, jitter, etc.). Thereafter, based on these parameters as well as available Quality of Experience (QoE) models the service quality perceived by the customer is estimated.

The applications considered are the following: Web browsing, FTP file transfer, Youtube videostreaming and VoIP.

Web browsing

QoE model for web browsing is based on the MOS evaluation method score from 1 (bad) to 5 (excellent) [27].

$$MOS_{webbrowsing} = \frac{4}{ln\left[\frac{0.003T_{max}+0.12}{T_{max}}\right]}(ln(T) - ln(0.003T_{max} + 0.12)) + 5 \tag{4.4}$$

The time to download a website (T) is the most important parameter which influences service quality.

Since the user expectation varies depending in the context, three scenarios are defined which differ the maximal expected time in order to download and display a website (T_{max}).

Expectation level	T_{min}[s]	T_{max}[s]
6 s	0.67	12
15 s	0.79	38
60 s	2.16	155

Table 4.1: *Web browsing measurement results.*

Results of the laboratory measurements are presented in Subsection 4.2.2.

FTP file transfer

Service quality of a file download basically depends on the duration of the download. However, as it is a TCP-based application the throughput is the most interesting metric from a user point of view. The average aggregate TCP throughput is used as a performance metric, which is simply the sum of the throughput of every single TCP stream going through node j. The average aggregated throughput metric describes the amount of traffic going to/coming from the Internet [28]:

$$\overline{G'_j} = \sum_{i=1}^{u} G'_{j,i} \tag{4.5}$$

Where:

- u is the number of sessions received by node j.
- $G'_{j,i}$ is the is the average throughput of session i over that complete session duration.

Results of the laboratory measurements are presented in Subsection 4.2.2.

YouTube videostreaming

In the context of YouTube QoE for TCP-based video streaming, the number of stallings is considered as impairment. QoE model in terms of MOS is described by an exponential function [29]. The mapping functions between the number N of stalling events of length L are given in Table 4.2. The coefficient of determination (R^2_L) for the different fitting functions is close to 1.

length L	mapping function $f_L(N)$
1s	$f_1(N) = MOS_{youtube} = 3.26e^{-0.37N} + 1.65$
2s	$f_2(N) = MOS_{youtube} = 2.99e^{-0.69N} + 1.95$
3s	$f_3(N) = MOS_{youtube} = 2.99e^{-0.96N} + 2.01$
4s	$f_4(N) = MOS_{youtube} = 3.35e^{-0.89N} + 1.62$

Table 4.2: *Mapping functions between MOS and number N of stalling events of length L.*

VoIP

For VoIP services typically the so called R-Score model [30, 31] is used which expresses the quality on a scale between 0 and 100.

The following shows an example for the G.711 codec:

$$R = 94.2 - I_d - I_{ef} \tag{4.6}$$

I_d is the impairment caused by mouth-to-ear delay (δ_{netw} is the network delay):

$$I_d = \begin{cases} 0.024(60 + \delta_{netw}) + 0.11[(60 + \delta_{netw}) - 177.3] & \delta_{netw} < 117.3ms \\ 0.024(60 + \delta_{netw}) + 0.11 & \delta_{netw} \geq 117.3ms \end{cases} \tag{4.7}$$

I_{ef} is the equipment impairment factor that comprises the impairments caused by low bitrate codec and impairment caused by packet loss.

$$I_{ef} = 30 * ln(1 + 15(e_{trans} + (1 - e_{trans})e_{jitter})) - I_d - I_{ef} \tag{4.8}$$

Where e_{trans} is the loss due to transmission errors and e_{jitter} is the loss due to jitter:

$$e_{jitter} = \sum_{i=b+2}^{N} P(\delta_{netw,i} - \delta_{netw,b+1} > BG) \tag{4.9}$$

Where:

- N is the total number of voice packets in the stream.

- B is the size of the jitter buffer in packets.
- G is the packet interval.

4.1.4 Comparison

To express the performance gain of the proposed approach the previously mentioned KPIs measured for a scenario in which the traffic control framework is enabled (KPI_{TCF}) have to be compared with the same scenario in which these functions are disabled ($KPI_{standard}$).

In general, the gain (G) can then be expressed as follows:

$$G = \frac{KPI_{TCF}}{KPI_{standard}} \tag{4.10}$$

Where:

- For the energy consumption G represents the relative energy consumption influenced by the traffic control framework.
- For the network efficiency G represents the relative amount of signaling overhead influenced by the traffic control framework.
- For the service quality G represents the relative service quality influenced by the traffic control framework.

4.2 Service quality measurements

This section presents the results of two services: web browsing and FTP file transfer. The purpose of these measurements is to get an idea about the effect of SoftToken on the service quality.

The QoE is the overall acceptability of an application or service, as perceived subjectively by the end-user. It includes the complete end-to-end system effects (client, terminal, network, services infrastructure, etc.) and may be influenced by user expectations and context. However, this research will concentrate on the effect of SoftToken performance on the QoE perceived by the end-user.

4.2.1 Measurement environment

The measurement environment is presented in this subsection. First, the measurement setup is introduced. Thereafter, the tool used to perform the measurements is explained.

Measurement setup

The measurements were performed with the Firefox OS-based Keon device in a real Digital Subscriber Line (DSL) network located at T-Labs. The device was connected via public Wi-Fi and 3GPP networks to the Internet, as depicted in Figure 4.1. The SoftToken module was the mechanism used for traffic management. As traffic differentiation was not considered, only the default queue was used. The scheduling parameters for the queue are: packet release interval/aggregation delay (T) and maximum number of packets to be released per interval (N). The smartphone was connected to the PC via USB in order to establish a remote shell with *adb* and be able to run commands from the PC (e.g. load and set the SoftToken module, start downloading files with *wget*, etc.).

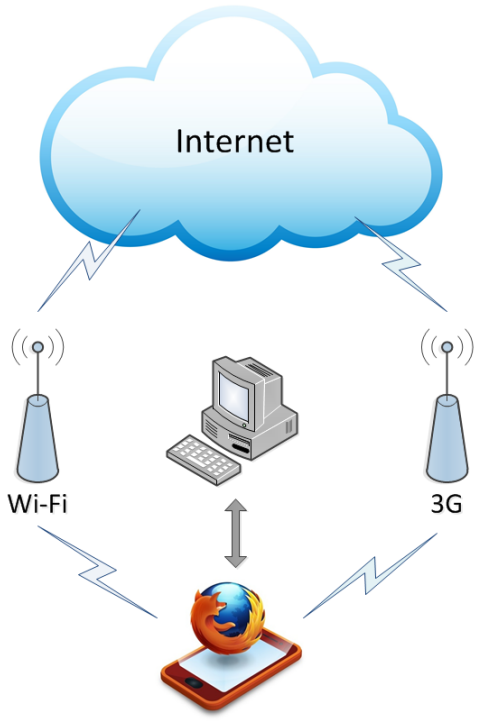

Figure 4.1: *Measurement setup.*

Measurement tools

For the present measurements the *wget* command was used. *wget* is a free command-line program for retrieving files using HTTP and FTP. Its features include recursive download, conversion of links for offline viewing of local HTML, and support for proxies.

In order to integrate *wget* into the Keon smartphone it was necessary to cross-compile it for the ARM architecture.

The following command was used to perform web browsing measurements:

```
time wget telekom.de --mirror -p --convert-links -P /data/local/test
```

- --mirror: turn on options suitable for mirroring.
- -p: download all files that are necessary to properly display a given HTML page.
- --convert-links: after the download, convert the links in document for local viewing.
- -P /data/local/test: save all the files and directories to the specified directory.

The command used to perform FTP file transfer measurements is shown below:

```
time wget -O /data/local/ftp_test_file http://vesta.informatik.rwth-aachen.de/ftp/
pub/comp/Linux/debian-cd/7.3.0/i386/jigdo-dvd/debian-7.3.0-i386-DVD-8.template
```

- -O: save the file to the given location path /data/local/ftp_test_file
- The second parameter is the URL of the file.

The reason why this specific file was chosen was its reduced size (6.3 MB) which allowed performing numerous measurements while still seeing the effect of SoftToken on the download time.

Additionally, the *time* command delivers the duration to download the whole website including all elements (e.g. images).

4.2.2 Measurement results

This subsection presents the measurement results of web browsing and FTP file transfer.

Web browsing

To measure service quality on web browsing different aggregation delays and maximum number of packets to be released per interval were set with SoftToken on both Wi-Fi and 3G interfaces.

The Figure 4.2 shows the time required to download a website. It is foreseeable that by increasing the aggregation delay the duration to download the website increases as well, this is very significant especially for $N = 1$. However, for $N = 100$ and $N = 1,500$ the degradation is not so important. It is also obvious that the Wi-Fi performance is in most of the times better compared to 3G. However, it should be noted that measurements have been done in a real network and that external influences (e.g. from other users) must be taken into account.

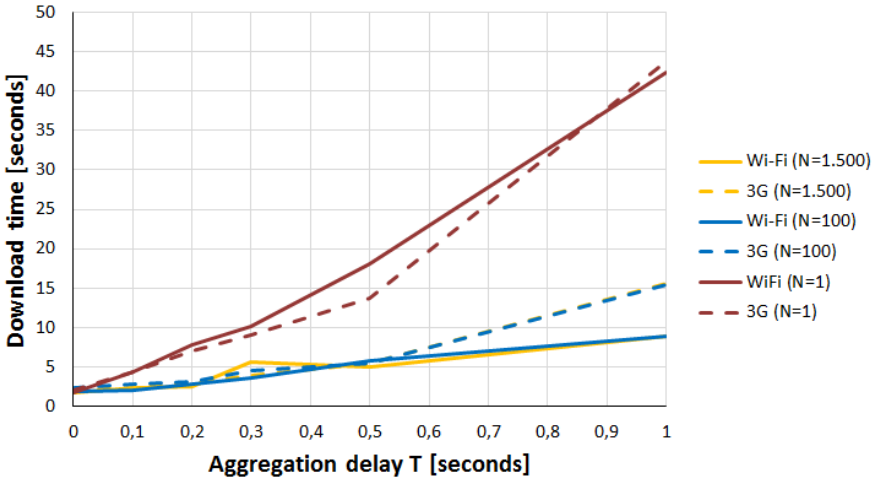

Figure 4.2: *Website access time vs. aggregation delay.*

The Figure 4.3 shows the performance gain for web browsing. The MOS for web browsing was calculated accordingly to the equation (4.4). Thereafter, the performance gain was obtained by comparing the scenario in which SoftToken was disabled (i.e. $T = 0$) with the same scenario in which SoftToken was enabled. The same conclusions can be extracted from this chart. As it is logical, the performance gain

decreases by increasing the aggregation delay. It can be seen in general that the performance is similar for small values of T.

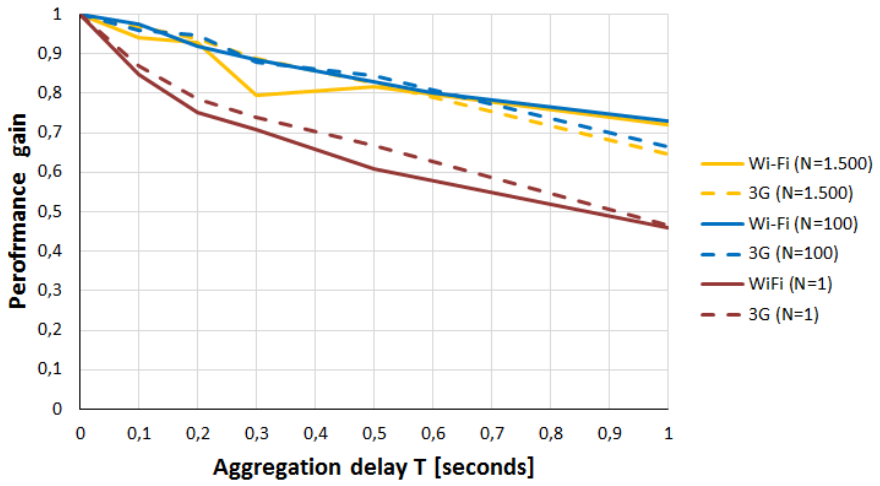

Figure 4.3: *Web browsing performance gain vs. aggregation delay.*

FTP file transfer

FTP file transfer was the second service evaluated. The same scenarios as for web browsing were recreated; that is, different aggregation delays and maximum number of packets to be released per interval were set with SoftToken on both Wi-Fi and 3G interfaces.

The Figure 4.4 shows the time required to download a file for different aggregation delays and maximum number of packets to be released per interval. The explanation of the results is almost equal to the explanation of the results of web browsing, in this case results for $N = 1$ have been neglected as the performance is too deficient when there is only one packet to be released per interval. It can be seen how 3G performance is very similar for $N = 100$ and $N = 1,500$ and it starts to decrease for values larger than $T = 0.3$. Wi-Fi performance is better compared to 3G but it starts to decrease notably for large values of T, especially for $N = 1,500$.

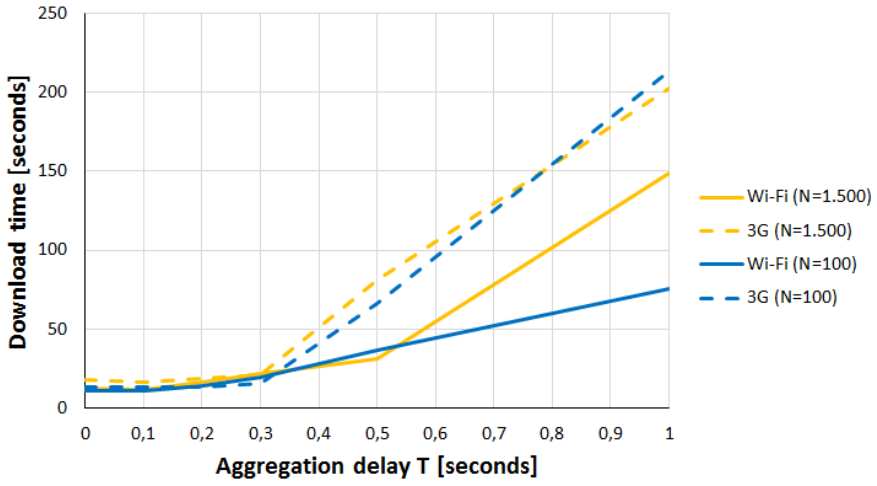

Figure 4.4: *FTP download time vs. aggregation delay.*

The Figure 4.5 shows the performance gain for FTP file transfer. The throughput is the metric used to measure the service quality. Thereafter, the performance gain was obtained by comparing the scenario in which SoftToken was disabled (i.e. $T = 0$) with the same scenario in which SoftToken was enabled. The chart suggests that for small values of T, 3G performance gain is better than Wi-Fi as it is less affected by the aggregation delay. However, for values larger than $T = 0.4$ Wi-Fi performance gain is better than 3G. It can also be seen that there is no relevant difference regarding the different values of N.

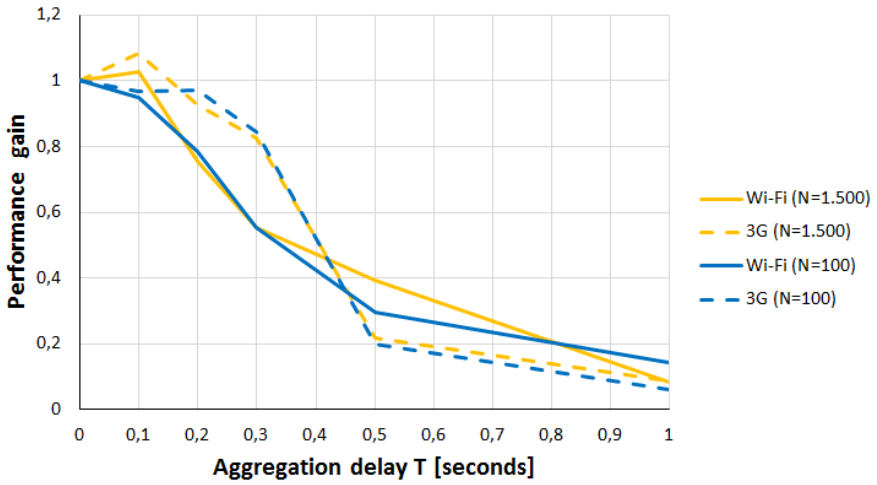

Figure 4.5: *FTP file transfer performance gain vs. aggregation delay.*

4.2.3 Conclusion

In general it can be seen that traffic can be aggregated without a negative influence on the service quality. Especially for very small values of T the performance is almost the same as without the proposed scheduling approach ($T = 0$). However, it is important to note that the QoE is perceived subjectively by the end-user. In comparison with 3G, Wi-Fi performance is better in the majority of situations.

After studying the obtained results, it seems reasonable to set $T = 0.3$ as a threshold for aggregation delay for a general scenario (i.e. without distinguishing between Wi-Fi and 3G). In general the performance gain for values below this threshold is acceptable, while for greater values the performance starts to decrease notably, especially for FTP file transfer. Regarding N, it seems that only small values (e.g. $N = 1$) affect the performance negatively, while the performance for larger values is similar.

Chapter 5

Outlook and conclusions

"Our best successes often come after our greatest disappointments."

— *Henry Ward Beecher*

This chapter concludes the document with a summary of the project, indicating the final status of the objectives. The traffic control framework proposed in this project is part of a Context- and Policy based Interface and Traffic Manager (CPITM) which uses the traffic control framework together with additional modules to provide an optimal solution to the smartphone challenge. The concept and basic functionality of the CPITM is introduced here. The review is completed by discussing the long term goal of the work on Firefox OS and the newly introduced CPITM. Thereafter the chapter is closed with a personal reflection about the experience of developing this project.

5.1 Summary

Smartphones have become indispensable in our lives. They are used not only for voice communication and messaging but also for additional services such as navigation system or TV. Since the air interface is a spare resource and the traffic for mobile end devices will grow enormously, it is important to use the wireless resources in the most efficient way. However, this is not what happens in current networks. The inefficient use of the wireless resources and the power resources of

the mobile device is caused by the missing interaction between mobile devices, apps and the network. Consequently poor network efficiency also affects other aspects, leading to delayed app responses, and subsequently to perceivable worsened user experience.

Additionally, the behavior pattern of emerging apps (e.g. instant messaging and social network services) requires the exchange of numerous messages and status updates. This attempt to keep the always-on connectivity leads to an increment of network signaling traffic in the mobile network. However, this is not the only cause of network signaling traffic increment, since the radio state transitions necessary to interact with the network generate a lot of network signaling messages as well. This leads to two major problems: rapid drainage of the mobile device's battery and a network signaling traffic surge in the mobile network. All that leads to the fact that the wireless resource but also the energy resource usage on the mobile device as well as the delivered service quality is not optimal.

The traffic control framework proposed overcomes these problems and creates a Win-Win-Win situation with the goal to deliver services with the best quality while providing a high battery runtime and high network resource usage efficiency. The approach is based on the aggregation of data packets prior to transmission. By this the packet transmission will be bursty which will improve network efficiency as the amount of network signaling messages is minimized. In addition, battery runtime is improved as lower network signaling overhead will reduce the activity time and energy consumption within devices. The main objective of the work developed in this project was to evaluate this approach and proof that the traffic control framework can be implemented on a Firefox OS-based device. Hence, as this document demonstrates the objective has been successfully accomplished.

The traffic control framework is part of a CPITM [32]. The basic idea of the CPITM is to estimate the current device usage based on input (context information) coming from different sources in order to perform interface and traffic management through the traffic control framework presented in this document. This is done for the current situation in such a way that wireless resource and power usage is minimized and service quality is enhanced. Therefore, different policies are used which tell how to behave in which usage scenario. These policies can either be local or remote. Local policies are pre-installed on the device or can be configured by the user while remote policies are pushed to the device from the network. Remote policies can dynamically be pushed to the device in order to adapt to changing conditions. For instance the

policy manager within the network can take network based context information (e.g. base station load) into account when creating a new policy which is then pushed to the device. By this the network is able to control to which access technology a mobile device will connect for instance to offload overloaded base stations. The policy manager is able to push generic policies which are valid for a large number of customers as well as individual policies which are only valid for a specific customer. The Figure 5.1 shows the architecture of the CPITM.

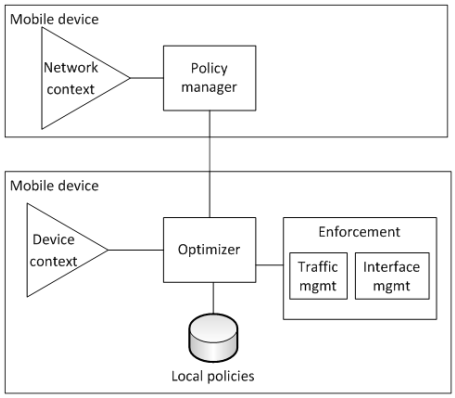

Figure 5.1: *Architecture of the CPITM.*

This project has made use of 3G networks. However, it should be noted that the benefits provided by the traffic control framework are not limited only to this type of networks. LTE and 2G networks could experience a network efficiency improvement with an intelligent traffic and interface management as well, since all these networks are treated as a bit-pipe with "infinite bandwidth".

The research within network efficiency will continue taking the work presented here as the basis to proof that the traffic control framework has a positive influence on service quality, network efficiency and battery runtime. For that reason this document provides some of the KPIs that can be used for evaluation together with results of service quality measurements. For the best performance a trade-off between service quality, network efficiency and battery runtime is required. Since network efficiency and battery runtime measurements are yet to be performed, future research should focus on them. Additionally, service quality measurements of other services such as VoIP, videostreaming or instant messaging could also be performed in order to extend the service quality measurements provided in this document. Thereafter, it would be possible to evaluate completely the proposed traffic control framework approach.

The long term goal of the work on Firefox OS and the newly introduced CPITM is of course an enhanced performance of the mobile device in all aspects (battery runtime, user experience, and network performance). Firefox OS is a relatively young and still small player in the mobile ecosystem; so the immediate impact might be limited. However, due to the similarity and close relationship to Android, the approaches described here can easily be ported to the big players in the market. And due to the modular approach and the relatively low level in the operating system stack, Original Equipment Manufacturers (OEMs) could make use of the proposed solution in Android even without using the full power and beauty of the approach.

5.2 Conclusions

The experience of carrying out this project has been an unique opportunity for me to learn and grow as an engineer and as a person. I feel greatly satisfied when looking at the final outcome, product of the hard work developed during the past six months. This experience has not only provided me with professional training, but also with valuable skills, such as programming, version control management, effective designing and working methodologies.

At the end of the internship the traffic control framework was successfully implemented. All the objectives set at the beginning of the project were accomplished. However, certain obstacles had to be overcome in order to complete key phases of the project. First, since the implementation is based on Firefox OS, a brand-new operating system itself in deep development at the same time as the execution of this project, it was not possible to count on extensive documentation or stable and reliable development tools. Second, Mozilla's WebAPI (e.g. Network Stats API) exposes only a limited API to apps and it could not be used to provide essential information required for the purpose of this project (i.e. statistics about outgoing and incoming packets). It is for this reason that the implementation of a daemon to obtain such information was needed. However, this could only be achieved thanks to the open nature of Firefox OS which allowed implementing the desired functionality at a level that is not accessible in other operating systems.

All these issues together with daily engineering challenges were effectively faced and they made of this project a perfect training for future research within the network efficiency as well as for my professional life.

Appendix A

Budget

"It is in your moments of decision that your destiny is shaped."

— *Tony Robbins*

In this appendix, a detailed view of the project phases is presented together with an estimation of the budget for the execution of this project. The total cost of this project is computed taking into account both material and human resources expenses based on the project phases discussed in Section 1.2 and detailed below. The amounts are expressed in euros.

A.1 Project phases

The tasks performed in this project could be grouped in three global activities, which derive in other secondary tasks:

1. Initial setup, design and implementation of the traffic control framework.
   (a) Setup of the development environment.
      i. Clonning the B2G repository, where the Firefox OS project is allocated, for building the emulator and the future Firefox OS version.
      ii. Installation of packages and libraries required for compiling and building Firefox OS.

iii. Installation of development tools for the Keon device.

iv. Installation of a ARM cross-compiler and external libraries needed for the implementation of the traffic control framework.

v. Setup of the Open Evolved Packet Core (OpenEPC) [33] environment as testbed for future research.

(b) Design and implementation of the traffic control framework.

i. Design and implementation of the Firefox OS Network Control application.

ii. Design and implementation of the network control and master daemons.

iii. Design and implementation of the remote server and database.

2. Implementation of a custom version of Firefox OS.

(a) Configuration of the kernel to enable QoS support.

(b) Modification of the *init* program.

(c) Integration of a token-based traffic control module based on the SoftToken protocol.

(d) Integration of command-line utilities: *tc*, *tcpdump*, *netcat*, *iperf* and *wget*.

3. Evaluation and documentation.

(a) Execution, documentation and interpretation of performance investigations for different types of services and QoS configurations of SoftToken vs. standard Wi-Fi and 3G.

(b) Documentation.

The duration of each task is represented in Figure A.1 which illustrates the project schedule.

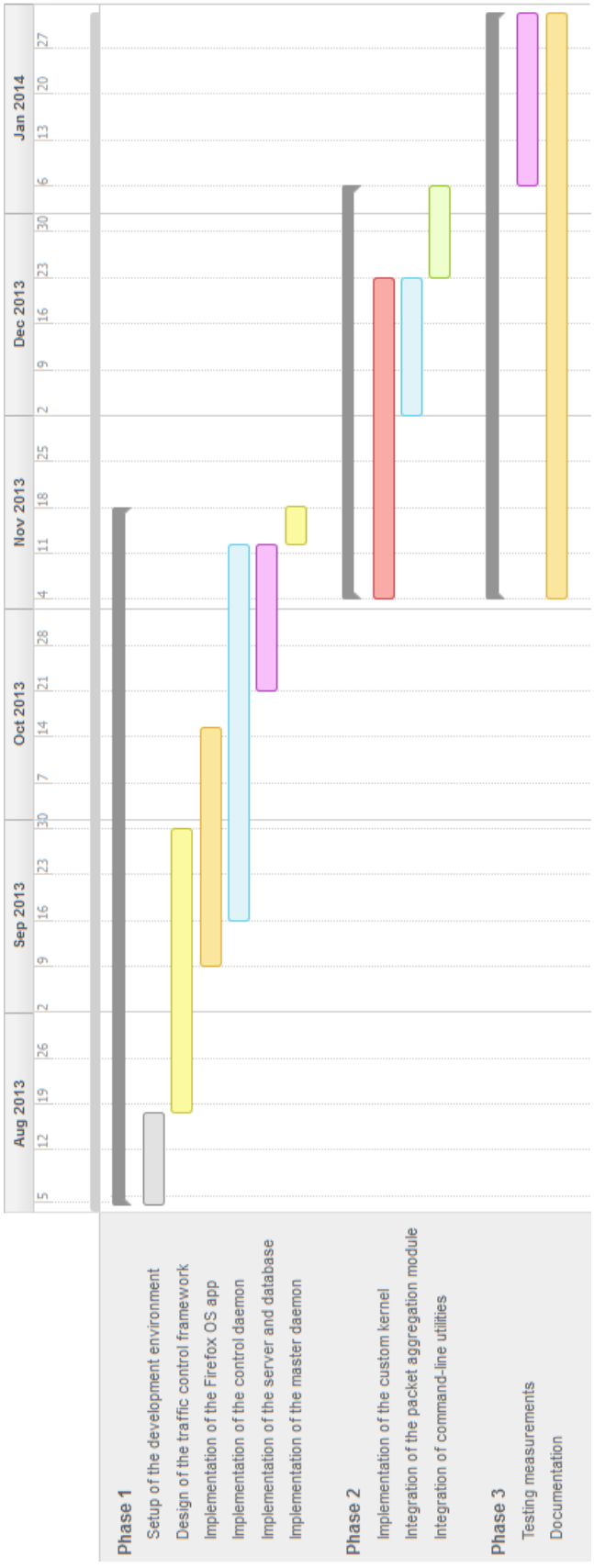

Figure A.1: *Gantt diagram.*

A.2 Material expenses

Material expenses only account for the material needed for the development, that is, one desktop PC, a desktop monitor, two laptops, a smartphone and a wireless router. Since the provided machines belong to T-Labs and they will be reused after this project, it would be misleading adding the whole cost to this project budget. A life of two years (24 months) is considered for the smartphone and the router, and three years (36 months) for the rest. Since the project lasted six months, it has been taken into account the amortization of the material to calculate the real cost.

The Table A.1 comprises the computed material expenses. The equipment used those six months specifically for this project is considered. Software costs are not taken into account because all software used is free; the only software with cost is Windows 7 and Microsoft Office, and the cost of such tools is already included into the laptop cost. The total amount includes a VAT of 20%.

Concept	Units	Cost/Unit	Amortized (%)	Total
Fujitsu-Siemens Lifebook S710	1	500	16.67	83.33
Lenovo G580 20150	1	600	16.67	100
Fujitsu-Siemens Display B24W-5	1	150	16.67	25
Acer Aspire R3700	1	300	16.67	50
TP-LINK Wireless N Router	1	50	25	12.5
Geeksphone Keon	1	90	25	22.5
Total				293.33

Table A.1: *Material expenses.*

A.3 Human resources expenses

The Table A.2 shows an estimation of the resulting human resources cost, considering a working day of 8 hours and 22 working days per month. In addition to the engineer responsible for carrying out the project, supervision an assistance tasks required the collaboration of a second engineer and a senior engineer. Working periods of 6 months, 1 month and 0.5 month are considered respectively.

Category	Months	Cost/Hour	Total
Engineer	6	24.75	26,136
Engineer	1	24.75	4,356
Senior Engineer	0.5	37.33	3,285.04
Total			33,777.04

Table A.2: *Human resources expenses.*

A.4 Total expenses

The table A.3 shows the final budget of the project, by adding the material expenses and the human resources expenses. As before, amounts are expressed in euros.

Concept	Total
Material expenses	293.33
Human resources expenses	33,777.04
Total	34,070.37

Table A.3: *Total expenses.*

The total budget of this project amounts to 34,070.37€.

References

[1] Cisco, "Global Mobile Data Traffic Forecast Update 2012–2017," http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html

[2] Nokia Siemens Networks Smart Labs, "Understanding Smartphone Behavior in the Network," White Paper, 2011.

[3] Y. Choi; Cha-hyun Yoon; Young-sik Kim; S.W. Heo et al., "The Impact of Application Signaling Traffic on Public Land Mobile Networks," *Communications Magazine, IEEE (Volume:52, Issue: 1)*, 2014.

[4] Firefox OS website, January, 2014, https://developer.mozilla.org/en-US/Firefox_OS

[5] The HTML5 Specification website, November, 2013, http://www.w3.org/TR/html5/

[6] The CSS Specification website, November, 2013, http://www.w3.org/TR/CSS/

[7] Ecma-262 Edition 5.1, The ECMAScript Language Specification website, November, 2013, http://www.ecma-international.org/ecma-262/5.1/

[8] Mozilla WebAPI website, November, 2013, https://developer.mozilla.org/en-US/docs/WebAPI

[9] L. Eznarriaga, P. Lozano, C. Senguly, N. Bayer and J. I. Moreno, "Experiences with SoftToken: a Token-based MAC Protocol for WLANs," *Int. J. Commun. Syst. DOI: 10.1002/dac.2679*, 2013.

[10] Netfilter website, November, 2013, http://www.netfilter.org/

[11] H. Holma and A. Toskala, *HSDPA/HSUPA for UMTS: High Speed Radio Access for Mobile Communications*, John Wiley & Sons Ltd., 2006, pp. 21-30.

[12] 3GPP, "Radio Resource Control (RRC) protocol specification," 3GPP specification of release 6 - TS 25.331, v6.2.0, Dec. 2008.

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

[14] JSON, December, 2013, http://en.wikipedia.org/wiki/JSON

[15] Node.js, December, 2013, http://en.wikipedia.org/wiki/Nodejs

[16] R. Muñoz, "Introducción a Node.js," http://www.rmunoz.net/introduccion-a-node-js.html

[17] MongoDB website, December, 2013, http://www.mongodb.org/

[18] Geeksphone Keon website, December, 2013, http://shop.geeksphone.com/en/phones/1-keon.html

[19] Circular buffer, December, 2013, http://en.wikipedia.org/wiki/Circular_buffer

[20] Socket Kernel Buffers, December, 2013, http://vger.kernel.org/~davem/skb.html

[21] The WebSocket Protocol, November, 2013, http://en.wikipedia.org/wiki/WebSocket

[22] Flot website, October, 2013, http://www.flotcharts.org/

[23] jQuery website, October, 2013, http://jquery.com/

[24] libwebsockets website, October, 2013, http://libwebsockets.org/

[25] jansson website, October, 2013, http://www.digip.org/jansson/

[26] Android Debud Bridge (adb) website, November, 2013, http://developer.android.com/tools/help/adb.html

[27] ITU-T Recommendation G.1030, "Quality of service and performance – Generic and user-related aspects".

[28] N. Bayer, B. Xu, V. Rakocevic, J. Habermann, "Application-aware scheduling for VoIP in Wireless Mesh Networks," *Computer Networks (Volume:54, Issue: 2)*, Feb. 2010, pp. 257-277.

[29] T. Hoßfeld, et al., "Quantification of YouTube QoE via Crowdsourcing," ISM, 2011.

[30] R. G. Cole, et al., "Voice over IP performance monitoring," SIGCOMM, 2001.

[31] ITU-T Recommendation G.107, "The E-Model, a computational model for use in transmission planning".

[32] H. J. Einsiedler, N. Bayer, K. Hänsge, R. Szczepanski, M. Kurze, T. Rettig, F. González García, A. Roos, S. Berg and J. I. Moreno, "Efficient Transmission of Smartphone Application Traffic in Wireless Access Networks," *The 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, Oxford, UK, April 2014.

[33] Open Evolved Packet Core (OpenEPC) website, November, 2013, http://www.openepc.net/index.html