Universidad Carlos III de Madrid

Escuela Politécnica Superior

Ingeniería de Telecomunicación

Multipath Traffic Bundling with Firefox OS

Autor: María Mora Martínez

Tutor: Dr. José Ignacio Moreno Novella

Acknowledgements

I want to thank my professor Dr. José Ignacio Moreno for his advice during my internship and his help in the completion of this thesis.

I would like to thank my supervisor in T-Labs, Dr. Nico Bayer, for his time and support. I can't put into works how grateful I am to have had the opportunity to work in such a great environment.

I also would like to thank Hans Einsiedler and the rest of his team, for their time and making this internship so great. Thank you, Roman Szczepanski, for helping me move on when I got stuck.

CONTENT

List of Figures

List of Tables

CHAPTER 1. INTRODUCTION

Connectivity is the foundation of a great mobile experience. While new generations like LTE of mobile technology can increase the connection speed, taking advantage of other technology available at the time can also improve the quality of service for the user. As more devices become connected, the challenge that emerged is how to provide secure and high speed connectivity with a good coverage of all areas. The amount of traffic keeps increasing, and even if UMTS is already much more advanced than GSM, limitations are still experienced, especially for big traffic flows or a high number of users. There are interference and isolation challenges that reduce the bandwidth and coverage of the technology. The frequency bands also get collapsed, and frequencies need to be reutilized, but there will be a point where the spectrum cannot allocate any more bands.

Other techniques to provide scalable solutions include the use of small cells. Small cells provide 3G connectivity in an area of coverage without modifications in the core network. Opposite to Wi-Fi, these devices use licensed radio spectrum, tying it to a single carrier company.

Proposals like Voice over Wi-Fi (VoWi-Fi) [1] are set to leverage LTE networks, increasing coverage with Wi-Fi hotspots. Security in communication must be maintained, so secure tunneling protocols are required, as public Wi-Fi networks are susceptible to hacking.

The motivation for the project at hand is to obtain the maximum efficiency of the technology available. The rapidly growing and highly competitive Wi-Fi market can be taken advantage of. Mobile phones usually provide two options to get an Internet connection, but only one of them is used at a time. The Wi-Fi connection is preferred, so if both are active, the cellular connection will be kept idle, with no traffic going through. By combining the throughput, when both interfaces are available, mobility, availability and also overall throughput could be improved.

Only on very specific situations the traffic is routed through both interfaces, as it is for example in the Download Booster [2] feature implemented on recent Samsung devices, where the bandwidth of both interfaces is combined to download files of more than 30MB. However, a similar process could be implemented to be used in any situation, if the user activates the feature.

A similar approach is taken by Alcatel-Lucent [3], in which the download traffic is transferred through the Wi-Fi connection, while the upload traffic (usually a smaller amount of data in mobile devices) is kept on the 3G/4G cellular channel.

The thesis is based on a project that started development during an internship of the Erasmus Placement program. The internship took place at the offices in Berlin of

Deutsche Telekom Innovation Laboratories (T-Labs) [4]. It lasted for six months, from October 2014 to April 2015. The work took place within the department of Seamless Network Control (SNC) under the supervision of Dr. Nico Bayer.

T-Labs is the central research and innovation unit of Deutsche Telekom. Working with other departments of the company, they develop innovative products, infrastructures and services for Telekom's growth areas. Apart from the research locations, T-Labs also maintains a network of Startups in Germany, Silicon Valley and Israel.

1.1. GOALS

The main goal is to design a setup in which both interfaces, Wi-Fi and cellular, can be used at the same time to access the Internet. To achieve this, the first step will be to investigate the way the operating system handles the network devices and change it so it allows both interfaces to be active simultaneously.

The target is to test two different approaches once the interfaces can work at the same time. As a first approach, a basic separation of traffic according to the session the packets belong to will be tried. From this we wish to figure out if using both interfaces is efficient and in which situations it makes sense to spread the traffic. Extracting information about the battery consumption would also be of interest.

Once we know how effective diversifying the traffic can be, a more intelligent approach will be sought, where traffic goes through either of them without differentiation, as if they were a single link.

A secondary goal will be to implement a simple application from which these new approaches can be easily controlled and tested.

Other secondary goals involve different tools and mechanisms in the phone that will need to be modified in order to achieve the main purpose.

Separating traffic in session blocks may not be efficient; some sessions can involve a high number of packets congesting the link and others very little, one of the links could have a bad signal thus providing low throughput. That is why it is needed to find a smart way to schedule traffic taking into account the environment. Designing this scheduling policy will be a final goal once the setup is designed and working, but could also be a future task for the next team, since time is limited.

1.2. PHASES

For the development of this project, different phases can be distinguished, each with a few milestones:

1. **Evaluation of the state of the art:**
- Previous research of tools and work from other teams and getting familiar with the Firefox OS environment.
- Update devices and implement changes designed by the previous team. Check everything is working as expected and modify if needed.
2. **Session-based scheduling:**
- Implement first setup, session-based scheduling. Take measurements of battery and time consumption.
- Evaluation of first setup.
3. **Packet-based scheduling:**
- Improve setup with more intelligent traffic scheduling. OpenVPN tests.
- Modify tools for bonding interfaces. Investigate if all tools work on the Firefox OS devices.
- Tests with bonding and traffic bundling solution.
- Test different scheduling policies for traffic bundling.
4. **Final evaluation and analysis.**

1.3. EQUIPMENT

The equipment and resources that will be needed for the development of this Project include:

- Lenovo G580 laptop.
- Acer aspire r3700 PC.
- Fujitsu-Siemens Lifebook S710 laptop.
- Fujitsu-Siemens Display B24W-5.
- 2 Mozilla Flame phone.
- 2 Geeksphone Keon phone.
- Powertool Ammeter Yocto-Amp.
- Subscription to the mobile network.
- Connectivity to a Wi-Fi access point.
- Ethernet connectivity.

1.4. DOCUMENT STRUCTURE

The current document describes the process of development and analysis of the setup for traffic bundling on a Firefox OS environment. Chapter 2 introduces the technology and features that were involved in the project. In Chapter 3, the development is described, with details of the implementation of the different features. Chapter 4 presents the tests that were carried out and analyzes the results obtained. Chapter 5 contains the summary of the document and a personal overview of the project.

CHAPTER 2. STATE OF THE ART

In this chapter, the problem to be addressed will be stated first, as well as the most relevant technology employed to tackle it. From mobile communications technology to basic Linux tools, they all have a relevant role in the development of the project.

As well as some common programming languages as JavaScript and HTML, more specific techniques and tools were used, which will be described and introduced in this section. JSON was used together with JavaScript to facilitate the data process. The technology provided by the devices, such as Wi-Fi and UMTS will also be explained. Other tools that will be described are NIC bonding, OpenVPN and packet scheduling. Finally, a bigger project in which this one was enclosed is mentioned, with an overview of its functionality.

The demand for faster Internet connection keeps increasing as years pass, and traffic grows exponentially. Developing new technology to supply for this is not always the best or fastest answer and other techniques using the available resources must be designed.

Firefox OS is the operating system that allows the developer to control deeper characteristics of the hardware of a device. Network operations involved in the project rely on more than just the operating system, which is merely an environment that allows to do the changes needed.

2.1. USAGE OF MOBILE PHONES

Global mobile traffic grew by 74% in 2015 [6], reaching 3.7 exabytes per month at the end of the year, and it is expected to keep growing at a very fast pace. Asia Pacific accounts for the largest part of this traffic.

The bigger contribution to this growth is the large number of new wireless devices that access the network. Over half a billion new mobile devices and connections were added in 2015. The number of smartphones increases, while laptops have experienced a reduction. But overall, the devices are getting smarter, requiring more intelligent networks. Smart devices generate as much as 14 times more traffic than non-smart devices.

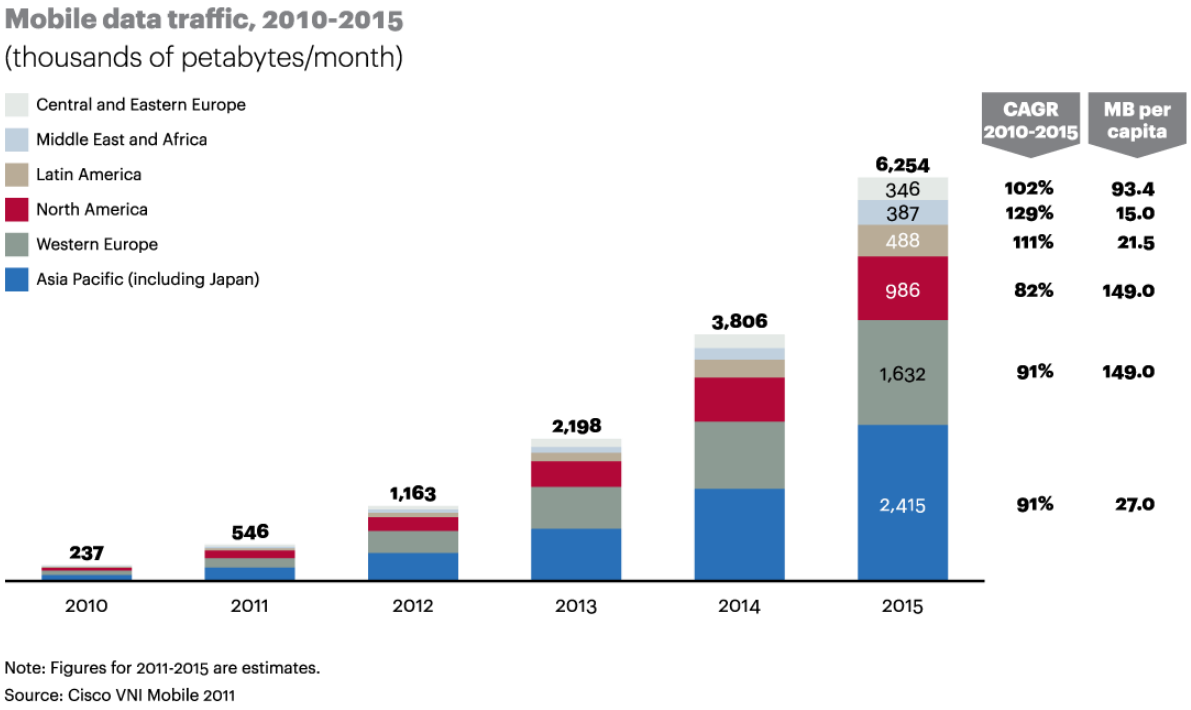

IMAGE 1. MOBILE DATA TRAFFIC BY AREAS [7]

The more traffic generated, the more base stations will be needed to service the customers. Offloading the traffic from the cellular network can be very important in order to avoid modifications in the network structure, which entails a higher cost. As the amount of data is increasing very fast, the network can become congested, deteriorating the quality of service (QoS).

As subscribers become increasingly active on social networks and run a growing number of applications at the same time, their smartphones can produce huge volumes of traffic and signaling as they interact with the network.

Mobile broadband penetration is set to increase simultaneously and as a result, the growth in signaling traffic will be much faster than the corresponding rise in data traffic. Nokia Siemens Networks estimates that the growth in signaling traffic will be up to 50% faster than the growth in data traffic over the next few years [8].

The Radio Network Controller (RNC) resides between the base station and 3G core network elements, protecting the core from the signaling generated by the radio access network for mobility management. In contrast, LTE uses a flat architecture that eliminates the RNC. As the core network is connected directly to the LTE base stations, it must handle all signaling traffic. The average signaling requirement per subscriber is up to 42% higher with LTE than with HSPA. It's clear that signaling has become a critical consideration when dimensioning packet core networks.

2.2. MOBILE DATA OFFLOADING

The ever-growing data consumption by mobile device users is straining mobile networks. Operators are attempting to address the challenge by upgrading their wireless WANs and deploying femtocells. However, in some scenarios, even these measures may not be adequate. With an increasing number of mobile devices featuring Wi-Fi capabilities and Wi-Fi access becoming more widely available in homes, enterprises and retail locations, Wi-Fi offload is emerging as an attractive option for network operators [9].

Cellular carriers in Europe are also investing to deploy machine-to-machine (M2M) communications for smart devices in industry, creating additional mobile traffic, but cellular networks do not have enough capacity to serve all this data. Due to low bandwidth or extra costs for the users, they might choose to avoid large transfers of data through the cellular network. An approach in which the request for the data is done over the cellular network and the actual transfer of the file or resource has been proposed, and could relieve the network from this strain, benefiting operators and at the same time users, since they will receive better quality and lower latency from a Wi-Fi network [10].

Operators would benefit most from seamless Wi-Fi offloading by applying it to data traffic that requires best effort and low quality of service (QoS). An ideal offload solution should provide users with a seamless experience while they use various applications on their devices. It should also make intelligent decisions about keeping data flows on preferred networks (e.g., keep some traffic, such as VoIP, on 3G/LTE even when Wi-Fi is available). It should optimize the resources on the mobile device such as battery life in addition to optimizing the user experience.

Solutions nowadays send all the traffic to Wi-Fi when available and only some check Internet connectivity. But other metrics are not taken into account. From the device perspective, the solutions view Wi-Fi as being available even if it may be unusable, which results in a poor user experience. Wi-Fi offload decisions based solely on signal strength may also be less than optimal in many situations. For example, a Wi-Fi network with "excellent" signal strength may be suffering backhaul congestion or may be blocked by a firewall. A good solution would be one that achieves efficient capacity management, improvements in user experience through seamless handovers, longer battery life by making intelligent decisions and support for simultaneous 3G/LTE and Wi-Fi access.

2.3. FIREFOX OS

Firefox OS is an open source mobile operating system based on Linux, open web standards and Mozilla's Gecko technology [11]. WebAPIs make it possible to access hardware capabilities. The entire user interface is a Web app, capable of displaying and launching other Web apps. This involves standard web technologies, with enhanced access to the mobile device's hardware and services.

Boot to Gecko (B2G) is the codename for Firefox OS, which is the branding and support services applied on top of B2G to create a final release product. Firefox OS is composed of three main modules or layers.

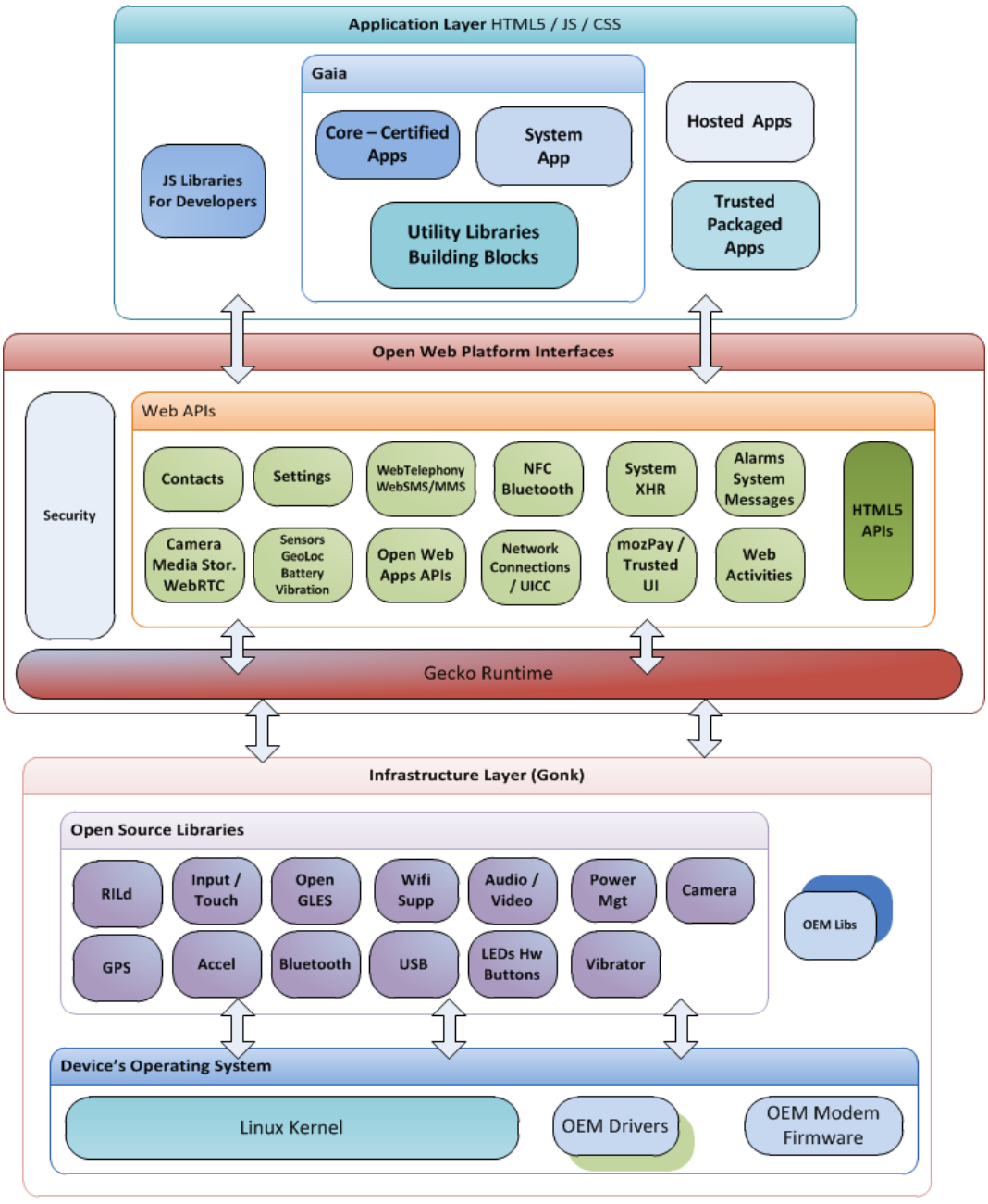

IMAGE 2. FIREFOX OS LAYER VIEW

Gaia is the user interface of the platform; it is in charge of anything that is drawn to the screen. It is implemented entirely using HTML, CSS and JavaScript. It interacts with the underlying operating system only through open Web APIs, implemented by the Gecko layer.

Gecko is the Firefox OS application runtime; it is the generic browser engine by the Mozilla project. It is located in the user space of the OS. It provides all the support for HTML, CSS and JavaScript, and makes sure all these APIs work well on every operating system it is supporting.

It includes a networking stack, graphics stack, layout engine, a JavaScript virtual machine and porting layers. Gecko's core is a very fast layout engine. It provides the foundation needed to display content on the screen. Gecko enables a pioneering new class of dynamic content that is more interactive and offers greater presentation control to web developers, using open and recommended Internet standards instead of proprietary APIs.

Finally, Gonk is the lower level operating system of the Firefox OS platform. It consists of a Linux kernel and userspace hardware abstraction layer (HAL). The kernel and several of the user space libraries are common open-source projects. Gonk is like a very simple Linux distribution, a porting target of Gecko, an adapter between the hardware and Gecko. Firefox

OS has full control over Gonk, which means that it is possible to expose interfaces that cannot be exposed on other operating systems. The kernel is based on the Android Open Source Project (AOSP).

It is interesting to see some details of the specific devices used during the project.

2.3.1. THE GEEKSPHONE KEON AND THE POWERTOOL

For the first part of the project, the Geeksphone Keon [12] was be used, since a power tool was available, allowing to measure the battery consumption of the phone.

IMAGE 3. GEEKSPHONE KEON DEVICE

The Keon is a developer-only device that is no longer being manufactured, but can be very useful for testing purposes.

It has a Qualcomm Snapdragon S1 7225AB CPU with 1GHz, supports up to 3G HSPA, with 4GB ROM and 512MB RAM. The two devices available were used in previous Firefox OS related projects.

The FxOS Powertool! is a very useful tool that provides information of battery consumption when using an ammeter connected to the phone. The ammeter is connected to a battery harness specific to each device model, since it has to fit in where the battery is located. It is connected via USB to a PC.

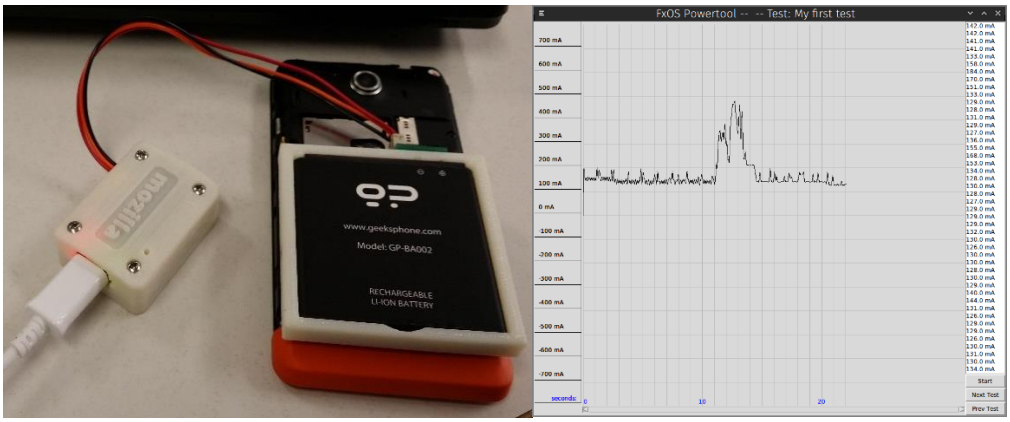

IMAGE 4. GEEKSPHONE KEON WITH AMMETER. FXOS POWERTOOL!

2.3.2. The Flame

Flame is the new official reference phone for development, testing and debugging of the operating system and its open web applications [13]. It was released in 2014 and started to ship to customers during that summer.

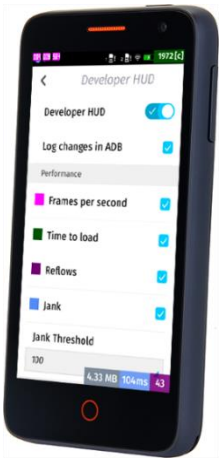

Image 5. Firefox Flame device

This device offers a series of hardware specifications that the Firefox OS software supports. This allows testing all the characteristics of the system, including NFC, frontal camera, dual-SIM, flash, software-configurable RAM, etc.

The main characteristics are the following:

- **Network**
  - GSM 850/900/1800/1900MHz
  - UMTS 850/900/1900/2100MHz
  - Wifi 802.11b/g/n
  - Bluetooth 3.0
- **Hardware**
  - Dual-SIM
  - NFC
  - Accelerometer
  - Proximity Sensor
  - GPS W / A-GPS support
  - Ambient Light Sensor
  - Qualcomm Snapdragon 200 MSM8210, 1.2GHZ Dual core processor
  - 256MB–1GB RAM(adjustable by developer)

Halfway through the project, two of these devices were acquired, providing much better and faster performance, making the experiments easier to carry out. Not all of the above specification is relevant to the project.

2.4. JSON

JSON stands for JavaScript Object Notation and it is a lightweight text format to interchange data [14]. It will be very useful in the project to transmit information easily between different structures.

Reading and writing it is easy for humans, but also for machines to interpret and generate it. It is independent from the programing language but uses conventions that are well known to programmers of C, Java, JavaScript, Perl, Python... Making it ideal for data exchange. It provides a syntax to serialize objects, arrays, numbers, strings, Booleans and null.

JSON is composed of two structures:

- A collection of name/value pairs. It is also known as object, record, struct, hash table...
- An ordered list of values. In most languages this is implemented by lists, sequences, vectors...

These are universal structures that are supported by all programming languages in one way or another. In JSON an object is a group of pairs of a name and a value, enclosed in braces "{ }". After each name there is a colon ":" separating it from its value, and between each pair there is a coma ",".

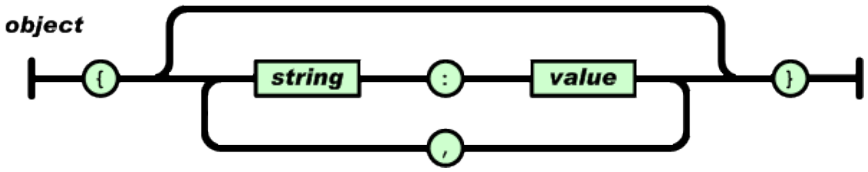

IMAGE 6. GRAPHIC REPRESENTATION OF A JSON OBJECT

There can also be arrays formed of multiple values enclosed by brackets "[ ]" and separated by comas. A value can be a chain of characters enclosed in quotation marks, a number, true or false or null, an object or an array.

JavaScript is the language that will use JSON in this project. It provides methods to deal with JSON objects [15], like JSON.parse() to analyze a string, and it can also transform the value, or JSON.stringify(), which returns a JSON string corresponding to the specified value.

2.5. UMTS

Universal Mobile Terrestrial System (UMTS) [16], the third generation of mobile telecommunications technology comprises several applications in wireless voice telephony, mobile Internet access, fixed wireless Internet access, video calls and mobile TV. 3G telecommunication networks support services that provide an information transfer rate of at least 200 kbps. Later 3G releases (3.5G and 3.75G) also provide mobile broadband access of several Mbps to smartphones and mobile modems. Each generation of cellular standards is characterized by new frequency bands, higher data rates and non-backward-compatible transmission technology.

In GSM data services are only possible over a circuit switched modem connection, with very low data rates. The first step towards an IP based packet switched solution was the evolution of GSM to GPRS, which use the same air interface and access method, TDMA (Time Division Multiple Access).

WCDMA (Wideband Code Division Multiple Access), was developed as a new access technology to reach higher data rates in UMTS. The access network emulates a circuit switched connection for real time services and a packet switched connection for datacom services. In UMTS the IP address is allocated to the UE when a datacom service is established and released when the service is released [17].

UMTS is the European system for 3G radio technologies, also used in Japan, China and other regions. Cell phones are usually UMTS and GSM hybrids. Several radio interfaces are offered, sharing the same infrastructure (W-CDMA, TD-SCDMA (China), HSPA+). CDMA2000 is the 3G system used in North America and South Korea. All these systems and radio interfaces are based on spread spectrum radio transmission technology.

The radio access specifications provide for Frequency Division Duplex (FDD) and Time Division Duplex (TDD) variants, and several chip rates are provided for in the TDD option, allowing UTRA technology to operate in a wide range of bands and co-exist with other radio access technologies.

UMTS includes the original W-CDMA scheme using paired or unpaired 5 MHz wide channels in globally agreed bandwidth around 2 GHz, though subsequently, further bandwidth has been allocated by the ITU on a regional basis. W-CDMA was specified in Release 99 and Release 4 of the specifications. High Speed Packet Access (HSPA) was introduced in Releases 5 (Downlink) and 6 (Uplink) giving substantially greater bit rates and improving packet-switched applications.

The radio access network connects to the core network, which is an evolution from the GSM core. 3GPP allows most services to be delivered over either 2G GERAN (GSM/EDGE) or 3G UTRAN. The core network is becoming progressively access-agnostic, allowing home base stations serving pica-cells to connect directly to the core network via subscribers' ASDL lines [18].

UMTS uses the same core network standard as GSM/EDGE. This allowed a simple migration for existing GSM operators. However, while much of the core infrastructure is shared with GSM, the cost of obtaining new spectrum licenses and overlaying UMTS at existing towers was high.

2.6. WLAN

Nowadays any smartphone is capable of connecting to the Internet via Wi-Fi. It is very important for our purposes of improving QoS in the FxOS devices, so it is necessary to understand its main features.

A wireless local area network (WLAN) is a wireless computer network that links two or more devices using a wireless distribution method (often spread-spectrum or OFDM radio) within a limited area such as a home, school, computer laboratory, or office building. This gives users the ability to move around within a local coverage area and still be connected to the network, and can provide a connection to the wider Internet. Most modern WLANs are based on IEEE 802.11 standards, marketed under the Wi-Fi brand name.

IEEE Std. 802.11 [21] is a set of media access control (MAC) and physical layer (PHY) specifications for the implementation of a wireless local area network (WLAN) computer communication in the 2.4, 3.6, 5 and 60 GHz frequency bands. It is required to appear to higher layers as a wired IEEE 802 LAN. The base version of the standard was released in 1997 and has had subsequent revisions. They provide the basis for wireless network products using the Wi-Fi brand. While each revision is officially revoked when it is incorporated in the latest version of the standard, the corporate world tends to market to the revisions because they concisely denote capabilities of their products. As a result, in the market place, each revision tends to become its own standard. The standards that are generally used and most known are:

- IEEE 802.11a: 54 Mbit/s, 5 GHz standard (1999, shipping products in 2001).
- IEEE 802.11b: Enhancements to 802.11 to support 5.5 and 11 Mbit/s in the 2.4GHz band (1999).
- IEEE 802.11g: 54 Mbit/s, 2.4 GHz standard (backwards compatible with b) (2003).
- IEEE 802.11n: Higher throughput improvements using MIMO (multiple input, multiple output antennas), operating in the 2.4GHz and 5GHz bands (September 2009).

Wi-Fi has adopted various encryption technologies over 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).

2.7. NIC BONDING

Network Interface Card (NIC) bonding is a method to gain higher bandwidth by combining multiple interfaces into a single logical one. All the bonded interfaces will appear as a single device, sharing even a MAC address. Linux has a special module to provide this functionality [22].

The way NIC bonding works, load balancing is only happening when there are different source MAC addresses, so for a single connection, all the traffic will be sent through the same link. Here is where load balancing algorithms come into action, providing, for example, round-robin scheduling [23].

There are different modes to perform the channel bonding and they can be divided in three categories:

- Failover-only. There is just one mode in this category: active-backup. In this mode, only one port is active until it fails, then the other one takes over the MAC and becomes active.
- Modes that require switch support.
    - balance-rr. Frames are delivered to the bonded channels using round-robin sequence. It is the only mode that sends packets across multiple interfaces that belong to the same TCP/IP connection. This mode and balance-xor work with any switch that supports EtherChannel or trunking.
    - 802.3ad. The official standard for link aggregation has a series of configurable options to balance traffic.
    - Balance-xor. The traffic will be balanced and hashed according to the receiver. It accepts incoming traffic from any active port.
- Generic modes.
    - broadcast. All the traffic is simply broadcasted on all interfaces.
    - balance-tlb. The incoming traffic uses only one interface and the outgoing traffic is load balanced changing the MAC address of the NIC. balance-tlb and balance-alb require that the device driver of the interfaces implement certain features such as support of ethtool and MAC modification while the device is active.
    - balance-alb. Both directions are load balanced changing the MAC address.

Bonding is a very important feature to achieve the goals of this project in order to be able to combine the traffic from multiple interfaces on the mobile phone.

2.8. OpenVPN

OpenVPN is a full featured secure network tunneling VPN software solution [25]. It offers a series of capabilities such as OpenVPN server and enterprise management. It supports many configurations, including secure and granular remote access to internal network or private cloud network resources. It is compatible with the majority of operating systems and it offers a web GUI to manage the server apart from the command line interface.

Image 7. OpenVPN logo

The Access Server is composed of three parts:

- OpenVPN Server. It is the component that does all the background work of routing, tunneling, encryption, authentication... There is a Web GUI to manage the underlying components of the VPN server.
- Admin Web Interface. It's an interface to easily manage the Access Server. Options such as layer 2 or layer 3 routing can be managed, user permissions, server network settings, authentication and web server certificates.
- Connect Client. It allows users to connect to the VPN through their web browser. Users can also download their configuration files to use on other clients.

OpenVPN allows to tunnel any IP subnetwork or virtual Ethernet adapter over a single UDP or TCP port and use all of the encryption, authentication and certification features of the OpenSSL library. It is designed to reduce the entry barrier for any business that needs to deploy a low-cost SSL-VPN solution.

Two different devices can be used to create tunnels. They are called tun and tap. A tun device is a virtual IP point-to-point device, while a tap device is a virtual Ethernet device. Tun and tap devices can be interconnected to create a complex routing topology.

There are different ports of OpenVPN for other operating systems. It avoids features that are not standardized well across different OSes, so porting it is quite straightforward. A port for Android above Ice Cream Sandwich is available.

2.9. PACKET SCHEDULING ON MULTIPATH ENVIRONMENTS

A network or packet scheduler is a program that manages the sequence of packets in the transmission and reception queues of the network interface controller. There are many different schedulers that implement a variety of scheduling algorithms.

Scheduling not only needs to be done according to the order of packets, the type of traffic and when to send each type. In a multipath environment it is also important to decide which interface will be used at each moment to send the packets. This is the scenario of interest for this project, as the goal is to share the traffic on two connections.

The main research in this area has been done regarding Multipath TCP (MPTCP). MPTCP is a transport layer protocol that allows network devices to transfer data over multiple concurrent paths.

Different approaches have been proposed to schedule traffic on an MPTCP environment where, as in TCP, the order of packets is extremely important. The method proposed in [26] delivers packets according to the estimated forward delay and throughput differences of the available paths. In [27] the most important type of schedulers for multipath transfers are compared. Minimizing packet delivery delay is what achieves the best performance but it is complex to accomplish. Depending on the scheduler, the capacity of the subflows and the propagation delay are important characteristics that need to be measured for scheduling decisions.

Other research involves video streaming over multipath [28][29]. Depending on the importance of the video packet and the dependence to other packets, the traffic is load-balanced and the paths selected, also taking into account the state of the network, for which the server must collect information. The results show that the solution proposed in [28] has a better performance than common scheduling algorithms.

Overall, taking into account the status of the network is important when choosing the path to send traffic and load-balance efficiently all the available links.

As mentioned before, Alcatel-Lucent announced in 2015 [3] its 'Wireless Unified Networks' strategy that has similarities with the project of Multipath Traffic Bundling with Firefox OS. They implement two capabilities to increase capacity and improve user experience. The downlink of Wi-Fi is combined with the uplink of cellular. This requires an OS software update to existing user devices and also updates in the network to blend both access networks.

2.10. DSL Community

The DSL Community Project [30] is being developed by Deutsche Telekom Laboratories to supply for the increasing demand for broadband. In their approach, the higher speeds are achieved by bundling the DSL connections of multiple neighbors when they are not using their capacity.

IMAGE 8. DSL COMMUNITY BASICS

This way, neighbors would be able to use other customers' idle bandwidth. For this, both neighbors need to have an intelligent router to make sharing possible [31]. The simultaneous use of multiple paths improves resource usage in the network and user experience, providing higher throughput and improved resilience to network failure. The patent application has already been filed and they have some initial prototypes running. The ideas was presented at the Mobile World Congress and the CeBIT.

IMAGE 9. DSL COMMUNITY PRESENTATION AT CEBIT

The Firefox OS bundling project could be interpreted as a part of this bigger project, since it uses the same basic ideas (combining traffic from multiple routes) and also some of the tools and techniques.

CHAPTER 3. DEVELOPMENT

In this chapter, the different steps of the development will be described, as well as the details of how the operating system and tools were modified to achieve the goals of the project.

An important feature is the possibility to enable both the Wi-Fi interface and the cellular data connection. This step is basic in order to fulfill our goal of combining both interfaces to obtain a higher throughput and better user experience. How this was achieved will be explained in detail in a later section.

Some tools and applications were downloaded or developed in order to control the features of the phone and exploit its full potential. Some typical Linux tools could be very useful in the development, so they had to be cross-compiled for the specific architecture of the phone.

As a first approach, all traffic corresponding to a session was routed through one of the interfaces. This way multiple sessions can be dealt with at the same time, using a different interface for each case. This environment was deployed and evaluated, with interesting results that will be discussed in chapter 4.

The second approach required more intelligent software to distribute all the traffic to both interfaces, independently of which session the packets belonged to. This meant new elements had to be introduced, such as a scheduler doing calculations to decide the routes, extra modules of the Linux kernel, and even a server with a public IP address.

3.1. SETUP

At a starting point, two Geeksphone Keon devices were available. The Keon is a Tier 1 device and was one of the first developer devices, which means it is fully compatible with FxOS.

The first step was to bring the device to the current state. The operating system was updated to the latest version released by Mozilla. At this point it was a prerelease version of B2G 2.2.0.

But before the new version could be installed, it had to be compiled. This had some minimum requirements for the machine in which it would be compiled. These are recommendations above the minimum, so the build was successful. Since this was done in a Linux machine, the following requirements applied:

- An installed 64 bit GNU/Linux distribution.
- At least 4 GB of RAM/swap space.
- At least 40 GB of available hard disk space.
- It was also recommended that the cache is set to 10GB so the build process would not saturate it.

The official repository from Mozilla, the B2G project, is stored in Github and can be retrieved and copied easily. The build process is going to create the four files needed to install the new FxOS version on the phones. To perform this build, some other tools were installed in the computer: autoconf 2.13, bison, bzip2, ccache, curl, flex, gawk, git, gcc / g++ / g++-multilib, java sdk (jdk), lzop, make, OpenGL shared libraries, patch, X11 headers, 32-bit ncurses, 32-bit zlib.

When the repository was copied and the tools installed, the software was configured for the Keon CPU architecture. This is simple, since the repository provides a configuration script that only needs a parameter specifying the architecture, in this case ‘keon’.

Up to this point, everything was done as described in the Mozilla Developer Network [32], however things changed somewhat since some new characteristics were needed for the purposes of the project. Before building the new version into the phone, some modifications were introduced in the source code, as well as some kernel options.

One of the most important modifications was the enabling of the cellular interface at the same time as the Wi-Fi interface for Internet access. The process to do this will be explained and discussed in the following section.

Once this finished, the code was compiled with another script. The whole operating system was built as it was a new version, but FxOS allows building only specific modules

of the system. When this was finished the new version was ready to be flashed into the device.

3.2. Enabling simultaneous use of cellular and Wi-Fi interfaces

To this date, generally mobile phones do not allow data connections through multiple interfaces except for very specific situations, such as download boosters, for example. This means that if a Wi-Fi connection is available all the traffic will be routed through it and the cellular route will be disabled. One of the advantages of Firefox OS is that it gives access to the source code, allowing changes.

The process in which this was done for this project is very rough and not elegant, but it is enough for testing purposes. It is done before building the operating system, by modifying the source code.

The first step was to locate the part of the code in which these interfaces are handled. This is in a file called 'RadioInterfaceLayer.js', part of the Gonk level of the system.

Once in this file, it was interesting the section triggered in the moment when the cellular data connection is disabled when a Wi-Fi connection is established. It required that this connection was kept active and its routes remain in the routing table of the device.

In this case:

```
if (networkInterface.enabled && wifi_active) {
    if (DEBUG) {
      this.debug("Disconnect data call when Wifi is
     connected.");
      }
    //networkInterface.disconnect();
    return;
}
```

The action line is deleted or commented to avoid the network interface being disabled as it would normally be.

The other situation is when Wi-Fi is already working and the network interface is enabled. In a normal course of events, this would do nothing until the Wi-Fi connection is lost. In order to enable both interfaces at the same time, the code is modified in this way:

```
if (wifi_active) {
     if (DEBUG) {
      this.debug("Don't connect data call when Wifi is
     connected.");
      }
    /*NEW*/ this.setupDataCallByType("default");
```

```
    return;
}
```

The new line enables the network interface. It is done like this because it is the same way it is done in other sections of the code.

These two changes allowed us to test the device using both interfaces at the same time. As it was said, it is not an elegant way of doing it; there are some errors in the functionality, and it would have to be improved in order to make a permanent change in the operating system behavior.

3.3. SESSION-BASED SCHEDULING

Once both interfaces can be enabled at the same time, the first approach consisted on routing different traffic sessions on different interfaces simultaneously. This means that if a website is loaded or a file downloaded, all the packets will be routed through the same interface, either Wi-Fi or cellular.

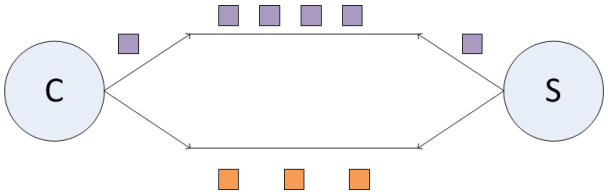

IMAGE 10. SESSION-BASED SCHEDULING WITH TWO INTERFACES

For these tests, the FxOS Powertool! was used in order to measure the battery usage during the process.

To perform this test, it was very relevant the MNI application. This app began to be developed by a previous team, but needed several modifications to fulfill the needs of this experiment.

Through a local socket, a visualization website could obtain the information about power consumption from the power tool. To obtain the bandwidth (BW) information, the website communicates via Wi-Fi with the boot daemon in the phone. Within the MNI app, another daemon is started, which takes care of downloading the files and adjusting the routing table in order to use the 3G and Wi-Fi interfaces, when required.

The main issue was to modify the routes so that both interfaces would be used. As a simple hack, two different files of the same size were used, to download each on a different interface. Then the destination address was introduced in the routing table, using as gateway each of the interfaces' IP address.

3.3.1. MNI APPLICATION

The method to easily take care of all the routing issues is by means of the application called My Network Interfaces (MNI). The layout is shown in the picture below. For this part of the project, the interesting parts were the access bundling section and the button that starts a daemon. At the top of the view there are two buttons that easily turn the two interfaces on and off, without having to access the settings of the device.

IMAGE 11. MAIN VIEW OF THE MNI APP AND DAEMON STARTING BUTTON

The interfaces are controlled with JSON commands passed to the daemon, then the view is changed to display the switch in interfaces.

```
var result = lock.set({
                    'wifi.enabled': true
                    });
                    result.onsuccess = function () {
                         $("#sw_label_1").text("on");
                         $("#state_wlan").text("WiFi is
enabled");
                    };
```

The daemons that have been mentioned are the ones listening for new commands and executing them. There is a boot daemon that starts running when the phone is switched on (at boot).

The boot daemon creates a socket on port 5000. When the MNI app is started, a connection is established to the socket and other tasks can take place.

```
var webSocketsServerPort = 5000;
server.listen(webSocketsServerPort, function() {
     console.log((new Date()) + "\nServer is listening on
port " + webSocketsServerPort);
});
…
if (request.origin.substr(0, 3) == "app") {

          console.log("Phone connected");

          if (!phone_connected) {
               phone_connected = true;
               connection_to_phone = request.accept('fxos-
protocol', request.origin);
```

```
                console.log((new  Date())  +  '  Connection
accepted phone.');
```

Now the “Start/Stop” button in the application can be pressed, so the second daemon is started, the app daemon. This daemon needs to be running for the app to recognize any of the other commands from the buttons. When the button is pressed the boot daemon takes care of launching the app daemon or stop its execution.

```
if (jsonrx["data"] == "execnode") {
          console.log("execnode received");
          var spawn = require('child_process').spawn;
          exec = spawn('node', ['/system/bin/app.js']);

          exec.stdout.on('data', function(data) {
               console.log('stdout: ' + data);});

               exec.stderr.on('data', function(data) {});

               exec.on('close', function(code) {
                    console.log('child process exited with
code ' + code);});

               var json = {
                    type : 'daemon',
                    data : "nodeexecuted"
                         };

               setTimeout(function() {

                    connection_to_phone.send(JSON.stringi
               fy(json));
                         }, 2000);
}
if (jsonrx["data"] == "killnode") {

     if (exec != null)
          exec.kill();

}
```

There is a similar code section to register a connection from the PC. This will allow to receive information on the PC regarding the download time. The boot daemon also registers the received and transmitted bytes.

When the app daemon is started, it opens a socket on port 3000. It is waiting for commands, which will be generated when a button is pressed. The two commands that were interesting at this point are the ones in the Access Bundling section.

The first one, labeled “Simple”, performs two downloads of 10MB files. The tool used to download the files was wget. Downloads were done using a single interface, either the one that is active if there is only one or via Wi-Fi if both are active.

```
var spawn = require('child_process').spawn;
var wget = spawn('wget',
['http://speedtest.reliableservers.com/10MBtest.bin', '-
O', '/dev/null']);
```

When each download finished it generated a message that would be used to register the download time.

```
wget.on('close', function(code) {
          console.log('child process exited with code ' +
code);
          var json = {
               type : 'responsesimple',
               data : "simplefinished1"
          };
connection.send(JSON.stringify(json));
});
```

The “Multipath” option required a few more operations, since the routing had to be modified. First, a script to change the routing had to be executed.

```
var   script   =   spawn('sh',   ['/system/bin/routing.sh',
'start']);
```

The script contained the following command, where IPdest is the IP address where the file to be downloaded with the cellular interface is stored. IPmovilegw and dev are the IP of the gateway and the name of the device (rmnet0) respectively.

```
start(){
ip ro add $IPdest via $IPmobilegw dev $dev
ip ro
}
```

Once the routes were set, two different files were downloaded. The second one corresponds to the IPdest used in the routing table. If the files didn’t have a different IP address it wouldn’t be possible to route them through different interfaces.

```
scriptwget = spawn2('wget',
['http://speedtest.reliableservers.com/10MBtest.bin', '-
O', '/dev/null']);
scriptwget2 = spawn3('wget',
['http://androidnetworktester.googlecode.com/files/10mb.t
xt', '-O', '/dev/null']);
```

When downloads finished the routing table was reset so it doesn't tamper other tests.

```
var script = spawn('sh', ['/system/bin/routing.sh',
'stop']);
```

3.3.2. VISUALIZATION WEBSITE

In order to easily visualize the values for download time and battery consumption a website that displays the progress was used. In this website, the bandwidth used can also be tracked. The graphs were created using Rickshaw [33], a JavaScript open source toolkit for interactive time series graphs.

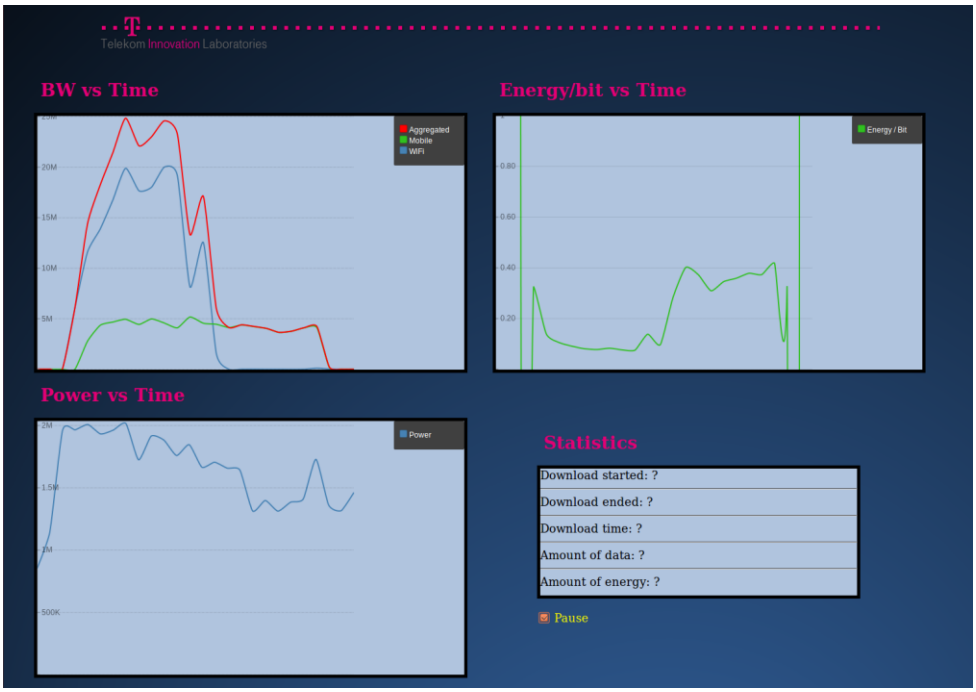

IMAGE 12. VIZUALIZATION WEBSITE

The website connects to the socket on port 5000 to the boot daemon, receiving the information to plot. For this to happen, it needs to be connected to the same Wi-Fi network as the phone. On local port 9000 it will connect with FxOS Powertool! to receive the battery information from the ammeter.

The data received from both sides is plotted in the graphs. Meanwhile, calculations to obtain the total data transmitted and power consumption are performed. When the download ends, the total values are displayed.

```
total_data = total_data + aggr;
total_power = total_power + power;
…
document.getElementById('dwtime').innerHTML = msg.data + "
seconds";
```

```
document.getElementById('dwenergy').innerHTML              =
(total_power / 1000)*(0.2/3600) + " mWh";
document.getElementById('dwdata').innerHTML = (total_data
/ 8000000)/2 + " MB";
```

3.4. PACKET-BASED SCHEDULING. FINAL ENVIRONMENT

Instead of systematically routing packets through one interface or the other according to the session they belong to, a more intelligent algorithm to separate them is needed. Packets don't need to traverse the network all in the same way; the important part is that they are in order in the end. So it was needed to find a way to separate individual packets and send them through different interfaces and being able to put them together again on the other end. This implies that there has to be a connection between client (phone) and server through both interfaces, and these two connections must be treated as only one. The setup described in [34] is the model that has been followed and adapted for this mobile scenario.

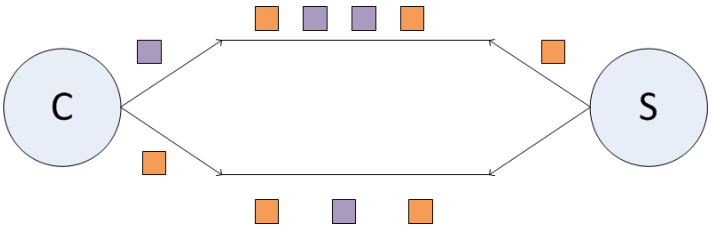

IMAGE 13. PACKET-BASED SCHEDULING WITH TWO INTERFACES

It was questioned whether the mobile phone would support the same technology employed in [34], so the first step was to test this.

One of the most important things is the bonding module of the kernel. Since the kernel of the phone is a Linux kernel, it should be possible to use this module, but it was not created with the compilation done so far. There is not much information about these kind of processes, but as Linux PCs have this module activated already, the compilation options were checked there, and added in the configuration of the phone's kernel.

```
CONFIG_BONDING = m
```

This simple change creates a new file with extension ".ko" in /system/lib/modules, where all modules configured in the kernel are stored. When the system is built and flashed again into the phone, the bonding module can be loaded and used, although there were some restrictions in functionality.

The idea was to create two tunnels, one for each interface, which would be combined in a bonding interface created with this module. The tunnels would be assigned to one of the interfaces, while the bonding interface would be designated as the gateway for all traffic.

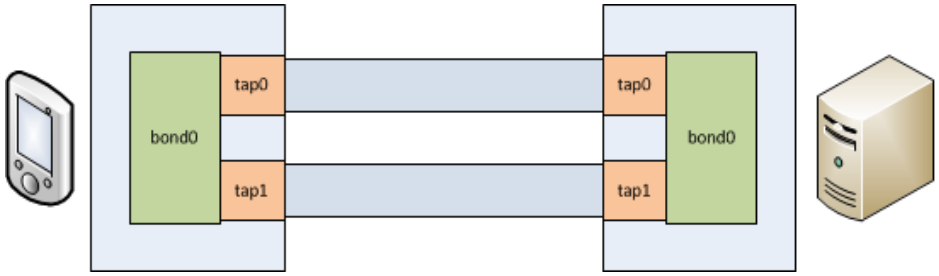

IMAGE 14. INTERNAL STRUCTURE OF PACKET-BASED SCHEDULING

However, these tunnels need to be created at both ends, with the right configuration so the connection is established correctly. This means a server is needed on the other side to combine again the traffic from those tunnels and deliver the packets to the final destination. As this is an ongoing project, the final details of this server have not yet been defined and it will be discussed in the final conclusions.

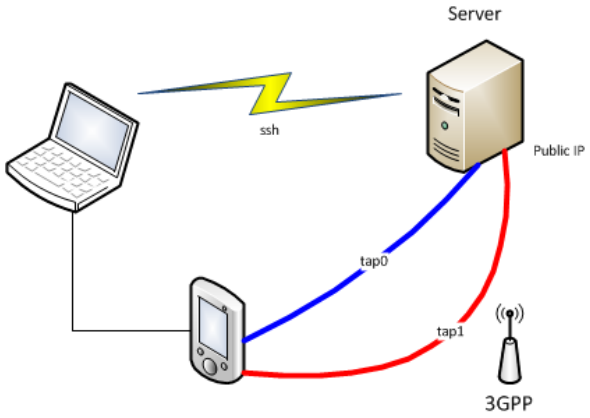

IMAGE 15. DESIRED MULTIPATH SETUP

3.4.1. BONDING TCP TUNNELS

In order to create this bonding interface to blend the traffic from Wi-Fi and data networks, it was needed to create a link through each interface that could be attached by the bonding module. Even when both interfaces could be active at the same time, by default only one of them would be used at a time (that is, Wi-Fi, since it is the preferred connection).

This is why tunnels were needed. These tunnels would connect to the bonding server and could be handled by the bonding module. The regular connection to the Wi-Fi access point or the mobile network could not be attached to a bonding interface since access to these points is not usually available.

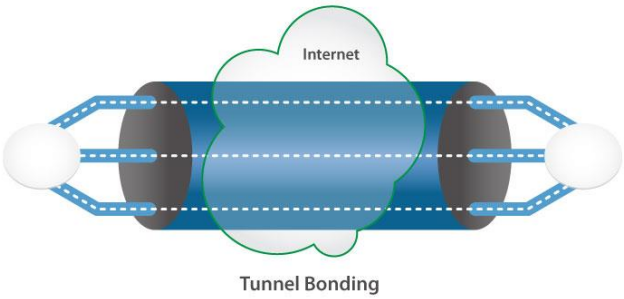

IMAGE 16. CONCEPT OF TUNNEL BONDING [35]

Before including a server with public IPs, it was needed to test the support that the phones have for tunnels and the bonding module. As explained before, the bonding module was already compiled.

A tool to create tunnels was needed. It was found that OpenVPN could provide the functionality needed and it was simple enough to use. It was chosen to use TCP tunnels because they allow to collect information and monitor the congestion window, different thresholds, etc. This would obviously add encapsulation overhead and extra acknowledgements of the packets, which made the process slower but provide useful information. But even if the tunnels were running TCP traffic, they could encapsulate any kind of traffic inside, not only TCP.

A version of the OpenVPN program for the phone's architecture, armv7, was installed correctly. As a first test, a Linux PC was used to communicate with the phone over Wi-Fi. From the test, it was obvious that the module version was more limited than the one available on the PC, not supporting all the working modes, but this was expected, since the kernel in the phone is a much older one, the one in the PC being kernel 3.11 and the one in the phone, 3.4.

At first, tun devices were used to create the tunnels, but the phone was not handling the MAC correctly, which meant the tunnels could not be bonded with ifenslave. So the tap devices were used instead and it was possible to create the tunnels and ping from the other end.

The tunnels can be created using configuration files for OpenVPN, but the phone does not support this functionality. Luckily they could also be created with command line operations the following way.

```
openvpn --mktun --dev $3
openvpn --remote $1 $2 tcp-client --dev $3 --ifconfig $4
255.255.255.0 --float --keepalive 10 120
```

Where $3 is the name of the device that we want to create, $1 is the destination IP, which is the public IP of the server, $2 is the remote port and $4 is the IP which will be assigned to the tunnel. The keepalive configuration works to keep the tunnel open when there is no traffic going through.

3.4.2. MNI App with new features

New functionality requires new features. This meant that the MNI app had to be modified to be able to test and manage the tunnels and bonding interface.

After modifying the app, it was possible to specify the local and remote IP addresses and ports to create the tunnels and create both by simply pressing one button; specify the local and remote IP addresses of the bonding interfaces and activate it by pressing the same button, also modifying the routes so the bonding interface is the default gateway for any connection; and download one file to test the connection. The bonding section has two expandable areas to modify the IPs or ports of the connection.

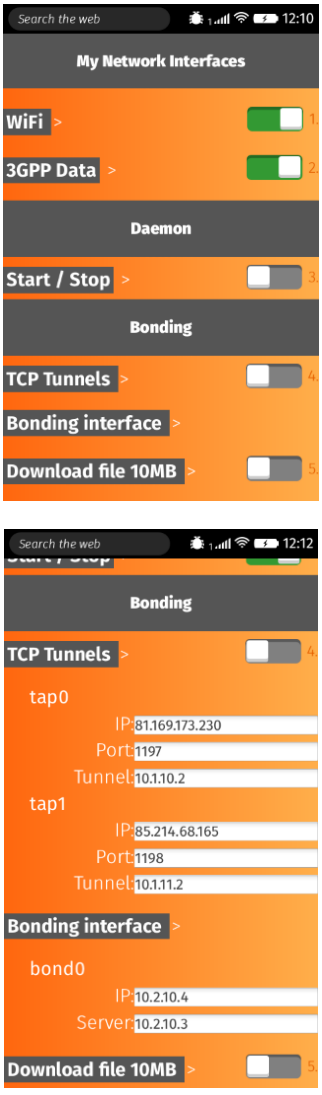

Image 17. Modified MNI App in collapsed and expanded view

This was all done is again using JSON commands. The IP addresses were read from the text boxes and sent to the app daemon, and once all the data was introduced, the button labeled with the number 4 could be pressed and the tunnels would be established to the server, also creating the bonding interface and setting the tunnels as slaves of bond0.

When the button is pressed, various JSON messages are generated, which trigger the different actions from the app daemon. The flag indicates whether the tunnels are already active, which indicates if by pressing the button they should be opened or closed.

```
if(status_tun == 0){
     status_tun = 1;
     var json_msg = {
               "type" : "command",
               "data" : "opentunnel",
               "ip0" : ip0,
               "ip1" : ip1,
               "port0" : port0,
               "port1" : port1,
               "tap0" : tap0,
               "tap1" : tap1
          };
          websocket.send(JSON.stringify(json_msg));
```

When the daemon receives the message, it collects all the data from the message and uses shell scripts to create the tunnels with OpenVPN.

```
if (jsonrx["data"] == "opentunnel") {
          //collect info for tunnels from html
               var ip0 = jsonrx["ip0"];
               var ip1 = jsonrx["ip1"];
               var port0 = jsonrx["port0"];
               var port1 = jsonrx["port1"];
               var tap0 = jsonrx["tap0"];
               var tap1 = jsonrx["tap1"];
               //create tunnels
               var                    spawn2                    =
require('child_process').spawn;
               var tunnel0 = spawn2('sh',
['system/bin/create_tunnel.sh', ip0, port0, 'tap0',
tap0]);
               var                    spawn3                    =
require('child_process').spawn;
               var tunnel1 = spawn3('sh',
['system/bin/create_tunnel.sh', ip1, port1, 'tap1',
tap1]);
          //add routes to table for the tunnels
               var spawn = require('child_process').spawn;
               var route = spawn('sh',
['system/bin/create_routes.sh', ip0, ip1]);
```

After the message to open the tunnels, a second message is sent so that the daemon creates the bonding interface.

```
var ipbond=document.getElementById("ipbond").value;
var ipserver=document.getElementById("ipserver").value;
```

```
            var json_msg2 = {
                 "type" : "command",
                 "data" : "openbond",
                 "ipbond" : ipbond,
                 "ipserver" : ipserver
            };
            websocket.send(JSON.stringify(json_msg2));
```

Again, the daemon receives the message and obtains the information about the bonding device that will be created.

```
if (jsonrx["data"] == "openbond") {
     var ipbond = jsonrx["ipbond"];
     var ipserver = jsonrx["ipserver"];
     var spawn = require('child_process').spawn;
     var bond = spawn('sh', ['/system/bin/create_bond.sh',
ipbond, ipserver]);
…
}
```

The script that is executed loads the bonding module with the static scheduling mode, which will be explained later, and creates the bonding device, attaching the tunnels with ifenslave. The scheduling is set to symmetric (11), because the phone cannot run the Python scheduler.

```
#load bonding module static mode
insmod /system/lib/modules/bonding.ko mode=static
#create bonding device
ifconfig bond0 $1 netmask 255.255.255.0 up
#load ifenslave sched
ifenslave -s 11
#attach tunnels to bonding dev
ifenslave bond0 tap0 tap1
#route traffic through bonding interface
ip ro change default via $2 dev bond0
```

Where $1 is the local IP assigned to bond0 and $2 is the remote IP assigned to their bonding device.

A visualization website was provided by the team in Darmstadt to see the progress of the traffic through the tunnels. It was based on the one used for the DSL Community project and it showed the view from the server side. This site also allowed to control which tunnel was used for the outgoing traffic from the server. This could also be changed on the phone side through a command line with ifenslave. The website allows to visualize other parameters, such as the congestion window of the TCP tunnel or the slow start threshold, besides other TCP session parameters.

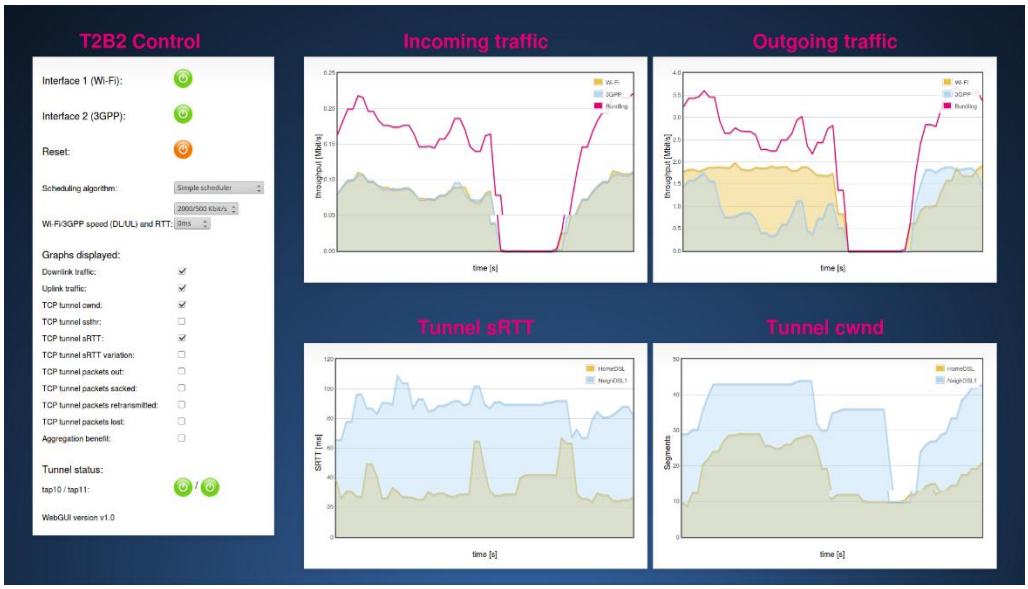

IMAGE 18. SERVER VISUALIZATION WEBSITE

3.5. MODIFIED TOOLS

The implementation of this idea simply requires some basic Linux tools, but some of them do not provide the whole functionality that is needed, so they were slightly modified.

The goal was to achieve the scenario depicted in the scheme below. The scheduler receives the traffic information and calculates the proportion in which packets should be balanced between the output interfaces. The TCP Probe module provides the data of congestion window and various thresholds to the scheduler to recalculate these values.

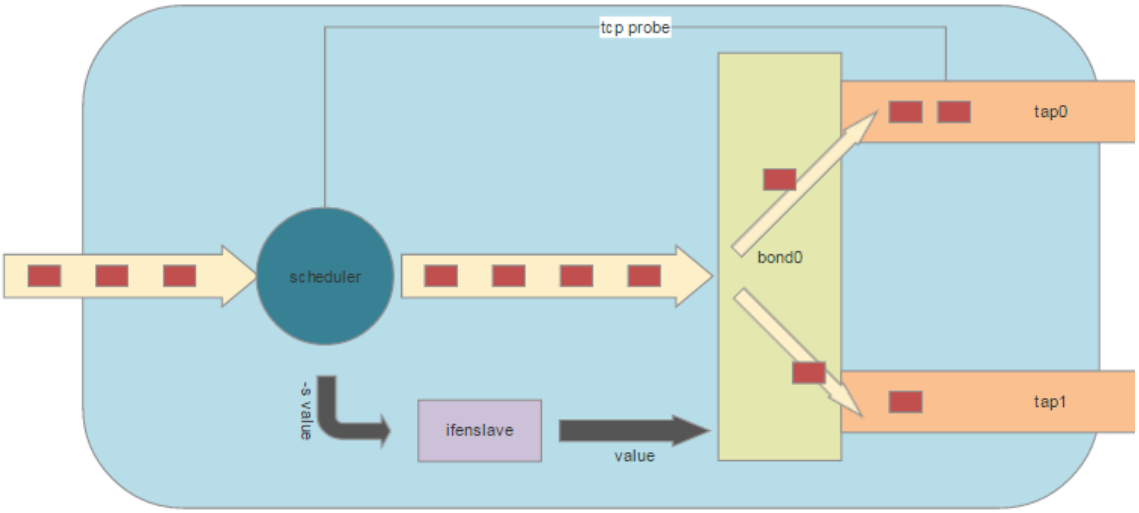

IMAGE 19. INTERACTION BETWEEN THE STRUCTURES FOR PACKET-BASED SCHEDULING

The scheduler was programmed in Python by the team in Darmstadt. It will execute an ifenslave command to apply the new scheduling values, so ifenslave had to be modified to be able to pass the value to the boding module. Also, the bonding module required some modifications of its own, to include a new scheduling mode that uses these values that were calculated.

3.5.1. IFENSLAVE

Ifenslave is a Linux tool that allows to attach and detach slave network devices to a bonding device [36]. This way, the host will see the bonding interface instead of all the slaves individually.

By a simple command, more than one interface can be attached to the bonding device:

```
ifenslave bond0 eth0 eth1
```

In this example, bond0 is the bonding device and would be the master interface. The two Ethernet devices eth0 and eth1 would be the slaves. In general, a simple Round-Robin scheduler will be used to send out the packets through the different slaves.

For the project, what is needed is exactly that, blending the traffic of the cellular interface and the Wi-Fi interface into one bonding device. But the Round-Robin scheduler might not be optimal for the mobile situation, since the bandwidth is not balanced between the two network devices. That is why some modifications were introduced in the source code.

A new usage mode was created, to allow control of the scheduling. This option was called with the parameter "-s" and specifying how to balance the traffic with two digits.

```
{"Packet",          0, 0, 's'}
```

The first digit defines how many packets to send through the first slave in the bonding device, and the second digit, through the second slave. So, for example, if the number was 12, it meant one packet would be sent through the first interface and two through the second; and they would continue to alternate until this number changes.

```
static int sched(char *master_ifname, int value){

     struct ifreq ifr;
     printf("hi    the    value    is    %d    %s\n",value,
master_ifname);
     memset(&ifr, 0, sizeof(ifr));
     strncpy(ifr.ifr_name, master_ifname, IFNAMSIZ);
     ifr.ifr_data = (void *)&value;
     if(ioctl(skfd, SIOCBONDSCHED, &ifr)<0)
          return -1;

     return 0;
}
```

The desired value for the scheduling is passed to the bonding module, as well as the name of the master interface. Basically, what ifenslave does in this mode is deliver the

information to the bonding module sending an interrupt signal, which will take care of using this value to organize the packets.

3.5.2. BONDING MODULE

As stated in the manual page of the bonding module [22], only three modes or transmit polices are available: Round robin, XOR or active-backup policy. Neither of these will balance the traffic according to congestion, available bandwidth and changes in the environment.

A new policy was introduced, called static scheduling. This way, packets will be sent through the interfaces according to the values calculated by the scheduler. A sequence is constructed in a vector, with the indexes of the interface to send each packet through.

New structures are created, as well as new functions. A vector of length 33 was used, initializing the number of packets for each interface to one (symmetric scenario). These variables will be updated when a new value is received from ifenslave.

```
static int sched_vector[33] = {0};
static int nr_pkt_first = 1 ;
static int nr_pkt_second = 1 ;
```

The vector is initialized with the values of packets for each interface in the function below. When packets should be sent through the first interface, the value will be zero, and for the second it will be one. If the first value is, for example, three, there will be three zeros in the vector, alternating with the value of the second interface. Let's say the second value is one; the resulting sequence contained in the vector will be: 00010001000100010001000100010.

```
static int sched_vec_init(size_t nr_pkt_st ,size_t
nr_pkt_nd) {
     int i;
     int cnt = 0;
     memset(sched_vector_cpy,0,32* sizeof(int));
     for (i=0 ; i<32 ; i++)
     {
          if (cnt < nr_pkt_st) sched_vector_cpy[i] = 0;
          else sched_vector_cpy[i] = 1;
     cnt++;
     if (cnt >= (nr_pkt_st + nr_pkt_nd)) cnt = 0;
     }
     return 0;
}
```

This initialization will be done also whenever a new value is received. This occurs when an interrupt happens of type SIOCBONDSCHED. In the previous section it was shown how the interrupt was generated by ifenslave.

When a signal of this type is received, the data is copied and, to obtain the values for each interface, the number passed from ifenslave is divided by ten to obtain the first digit, and the rest of the division gives the second digit. After that, the vector is updated with a new sequence.

```
case SIOCBONDSCHED:
          u_scinfo   =   (struct   ifsched   __user   *)ifr-
>ifr_data;
          if      (copy_from_user(&k_scinfo,      u_scinfo,
sizeof(ifsched)))
               return -EFAULT;
          nr_pkt_st = k_scinfo.sched_value / 10 ;
          nr_pkt_nd = k_scinfo.sched_value % 10 ;
          sched_vec_init(nr_pkt_st, nr_pkt_nd);
     for(i=0;i<32;i++)printk("%d",sched_vector_cpy[i]);
          printk("\n");
          sched_flag = 1;
          return 0;
```

The main function of the new scheduling policy simply takes the sequence and sends the packets according to it. Let's have a look at the key points.

When a packet is received at the bonding device, the function will be called. The slave to transmit is chosen using the scheduling vector and after that the counter is incremented for the next packet to send.

```
slave_no = sched_vector[sched_index];
     sched_index++;
```

Once the whole vector has been traversed, the counter is reset to keep sending packets with the same sequence.

```
if(sched_index == 32) {
     sched_index=0;
```

Finally, the packet is sent to the corresponding slave interface, provided it is up and active.

```
bond_for_each_slave_from(bond, slave, i, start_at) {
          if (IS_UP(slave->dev) &&
               (slave->link == BOND_LINK_UP) &&
               bond_is_active_slave(slave)) {
                res = bond_dev_queue_xmit(bond, skb, slave-
>dev);
                break;
          }
}
```

3.6. OBSTACLES

Throughout the development of the project, different obstacles and difficulties were encountered. Some of them only made the process slower, but others meant that a new approach had to be decided.

The Keon devices were a little limited in their hardware capabilities, which made them slower, increasing the time needed to perform the tests. The first stage of session-based scheduling was performed with these devices, and even though they were slow, the tests were performed successfully. But when the second stage of packet-based scheduling was tested, they showed greater limitations. A new version of Firefox OS was released during that time and the compilation process brought up many errors, which were complicated to overcome.

Luckily, the new Flame devices had already been ordered and arrived at the early stages of the second phase. The newer Firefox OS version was installed without any issues.

Another issue that came up was caused by the way the two interfaces were forced to be active at the same time. When Wi-Fi is active but cellular is not and then it is turned on, the routes are not added to the routing table, since Wi-Fi is the default link. In this case, Wi-Fi needs to be deactivated to force the cellular route to be added. Then it can be switched on again and both routes will stay in the table, Wi-Fi still being the default option. In order to make this functionality more effective, the source code of Firefox OS will have to be studied and modified even further.

In order to have the new kernel modules they had to be added to the OS compilation, but there is little to no literature about how to do it. The Mozilla B2G development blog was consulted, but in the end it was more of a trial and error process.

As has been mentioned, the scheduler was programmed in Python. Firefox OS does not include a Python interpreter by default, so one had to be found and compiled for the phone architecture. However, this was not enough, because the scheduler uses NumPy, which is a Python package for scientific computing. It was not possible to find a way to incorporate NumPy in to the phone, so the scheduler could only be used on the server side.

As it was decided, there would be a server to create the other end of the tunnels from the phone and merge the traffic. For this, it was necessary a public IP address on the server side where the OpenVPN clients can connect. But because of the proxy at the office, it was not possible to have a local server with a public IP. The public IP was assigned to it, but when tried to contact, it did not respond. This meant that a different server had to be used, so one in another office location in Germany that was already operative was used for this purpose as well.

Also related to bonding, the module modified by the Darmstadt team did not compile for the kernel version of the phone, so parts of the code had to be adapted, since it was an

older kernel. Some of the functions had changed slightly, as well as some structures that had different parameters. The solution was to use the original source code for kernel version 3.4 and add the new functions and variables for the static scheduling mode.
When the modified bonding tools were received from the other team, the master interface which should modify the scheduling sequence according to the values received was hardcoded into the function of ifenslave. This meant that, regardless of which interface was specified in the ifenslave command, “bond0” was always modified. The slave name is irrelevant, since the master device should use the slaves according to the values provided.

```
static int sched(int value){
     struct ifreq ifr;
     char *master_ifname ="bond0";
     char *slave_ifname = "eth5";
     memset(&ifr, 0, sizeof(ifr));
     strncpy(ifr.ifr_name, master_ifname, IFNAMSIZ);
     strncpy(ifr.ifr_slave, slave_ifname, IFNAMSIZ);
     ifr.ifr_data = (void *)&value;
     if(ioctl(skfd, SIOCBONDSCHED, &ifr)<0)
          return -1;
     return 0;
}
```

CHAPTER 4. MEASUREMENTS AND EVALUATION

In this chapter, the tests carried out during the project and the results obtained will be described and analyzed.

There were two distinct phases in the project according to the scheduling policy used. In the first part it was easy to take measurements and the results gave a good idea of the effectiveness of this approach.

For the second approach, the devices required a higher degree of modifications, which took much longer and, as it was in a very early stage of development, many details had to be tested to find the proper way to implement the desired functionality.

4.1. SESSION-BASED SCHEDULING

Through the MNI App, two files of the same size were downloaded. In order to perform the measurements of download time and energy consumption of the device, the setup presented in the image was used. The ammeter was connected to the Keon phone and to a PC. The configuration of the application was explained in the development section.

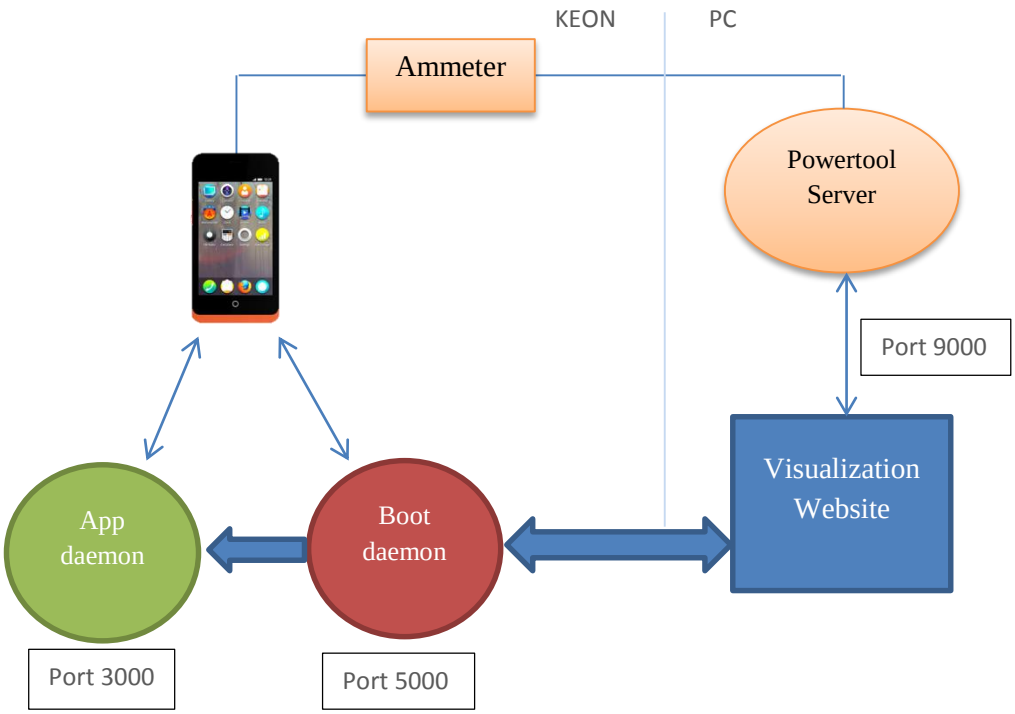

IMAGE 20. SETUP FOR SESSION-BASED SCHEDULING

The following table contains the values for download time (s) and energy consumption during the download (mWh) depending on the maximum bandwidth available in the Wi-Fi link. The values of energy consumption were measured in miliwatt by hour (mW·h) since it is a typical unit for batteries. Ten measurements were taken for each value of Wi-

Fi bandwidth and for each of the three cases. The bandwidth of the Wi-Fi link was modified from the local access point that was created on a PC for this test, using a script based on the Linux tool *tc*. In the first case, two files of 10MB size were downloaded via only Wi-Fi; in the second case, both downloads were downloaded through the 3G network; and in the last case, one of the downloads was performed via Wi-Fi and the other one via 3G network. As expected, the results when using only the 3G network did not depend on the Wi-Fi bandwidth, since the link was not being used.

	Wi-Fi		3G		Wi-Fi+3G	
Wi-Fi BW (Mbps)	T(s)	E(mWh)	T(s)	E(mWh)	T(s)	E(mWh)
1,5	117,43	309,8632	38,78	138,6591	59,79	210,0634
2	90,430	232,703	42,450	151,051	45,010	173,903
2,5	70,660	196,103	38,340	136,014	36,610	155,866
3	59,040	164,353	39,190	140,421	30,880	134,048
3,5	50,680	142,927	41,040	146,355	25,950	114,838
4	44,430	126,680	39,880	142,234	23,430	105,169
6	29,780	88,095	39,130	140,052	20,320	89,510
8	22,780	68,478	39,790	142,468	18,670	79,734
10	18,650	56,219	39,860	142,391	17,840	76,207
10,5	17,640	52,787	39,030	140,024	19,820	81,517
11	16,840	50,203	39,990	142,363	18,200	74,484
11,5	16,230	48,257	39,710	141,130	16,230	69,084
12	17,310	52,984	40,400	143,909	16,610	70,315
14	14,950	46,536	37,800	135,419	18,000	75,671

TABLE 1. DOWNLOAD TIME AND BATTERY CONSUMPTION

Comparing the three scenarios, we could see that using only the 3G link was most of the times worse in terms of download time than the other two cases, except for when the Wi-Fi bandwidth was very low. Since during the test the bandwidth of the 3G link ranged from 4 to 6 Mbps, when the Wi-Fi bandwidth became higher than this limit, the download was faster through the Wi-Fi link. When both technologies were used at the same time, downloading one of the files through each interface, the performance was improved, especially for the lower Wi-Fi bandwidth values. As this bandwidth increased, it would be faster to use only Wi-Fi when the time it takes to download one file via 3G becomes larger than the time it takes to download both via Wi-Fi (between 10 and 12Mbps).

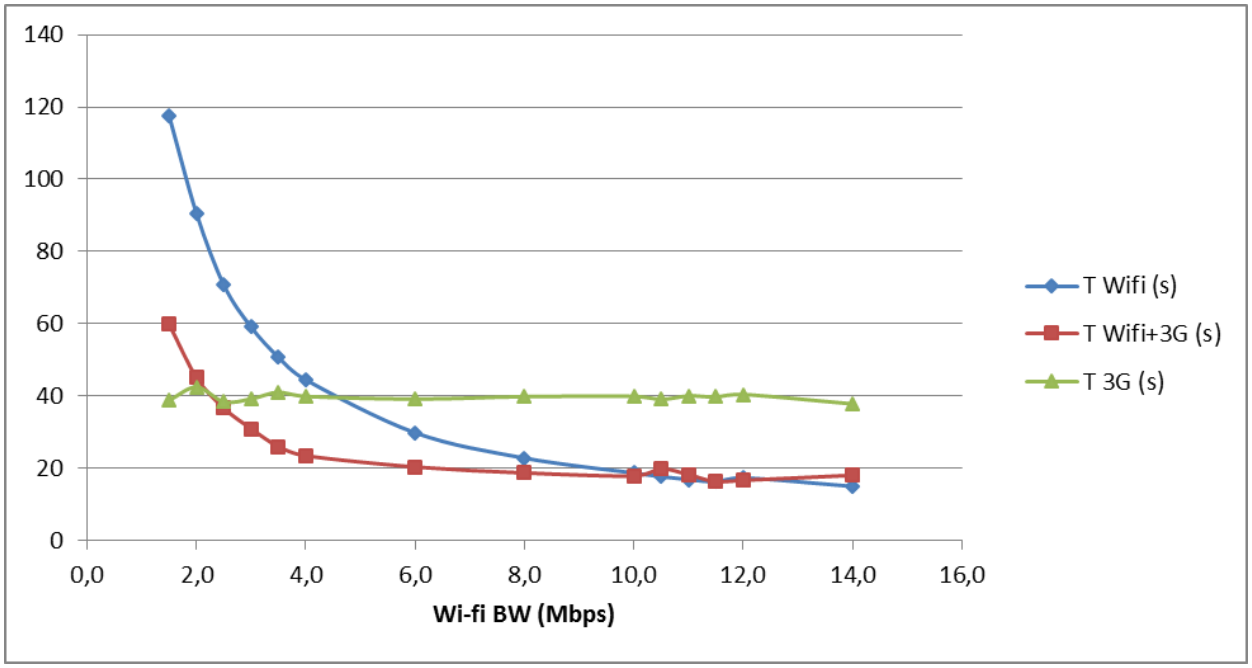

IMAGE 21. DOWNLOAD TIME OF THE THREE SCENARIOS

In terms of energy consumption, the results were quite similar as in the case of download time. For very low values of Wi-Fi bandwidth, the consumption was higher than with 3G, both for only Wi-Fi and using both links at the same time, since the energy consumed by Wi-Fi is quite high in these cases. When the Wi-Fi bandwidth is increased up to 3Mbps and higher, using only the 3G link is less energy efficient than the other scenarios. Comparing the use of only Wi-Fi with using both technologies, we can see that when the Wi-Fi bandwidth is higher than the 3G bandwidth (usually between 4 and 6Mbps) the Wi-Fi link consumes less energy than both links working at the same time.

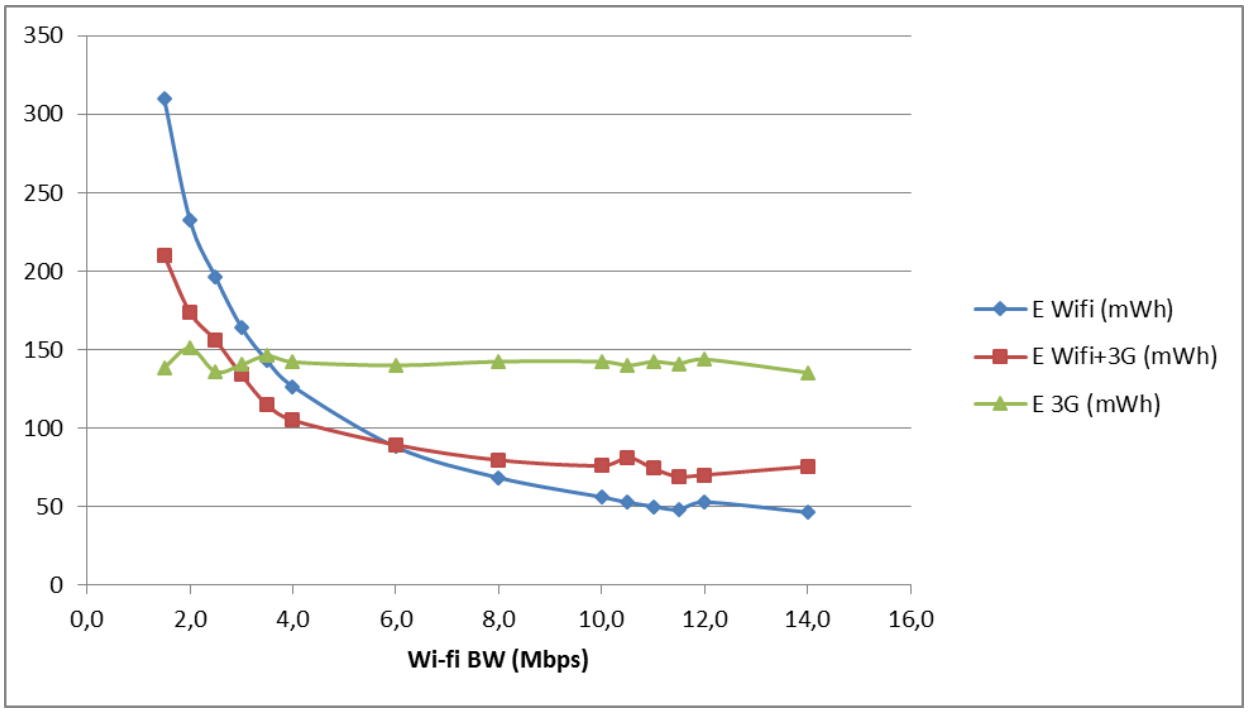

IMAGE 22. ENERGY CONSUMPTION OF THE THREE SCENARIOS

With these values we could obtain a measurement of the gain produced by using the Wi-Fi and the 3G links at the same time. When the time or energy used by the single link was higher than for both links, the gain was positive, which meant that it makes sense to use both links, otherwise the gain would be negative and it would be more efficient to use a single link.

	Wi-Fi vs Wi-Fi+3G		3G vs Wi-Fi+3G	
BW	T(s)	E(mWh)	T(s)	E(mWh)
1,5	57,64	99,800	-21,010	-71,404
2	45,42	58,799	-2,560	-22,852
2,5	34,05	40,237	1,730	-19,852
3	28,16	30,305	8,310	6,373
3,5	24,73	28,090	15,090	31,518
4	21,00	21,511	16,450	37,065
6	9,46	-1,415	18,810	50,542
8	4,11	-11,255	21,120	62,734
10	0,81	-19,988	22,020	66,184
10,5	-2,18	-28,730	19,210	58,507
11	-1,36	-24,281	21,790	67,880
11,5	0,00	-20,827	23,480	72,046
12	0,70	-17,331	23,790	73,594
14	-3,05	-29,134	19,800	59,749

TABLE 2. GAIN COMPARISON OF SESSION-BASED SCHEDULING

From the plot, we can see that when the Wi-Fi bandwidth was lower than 2.5Mbps, it was faster to use only the 3G network. Up to 10Mbps it was faster to use both links rather than just Wi-Fi.

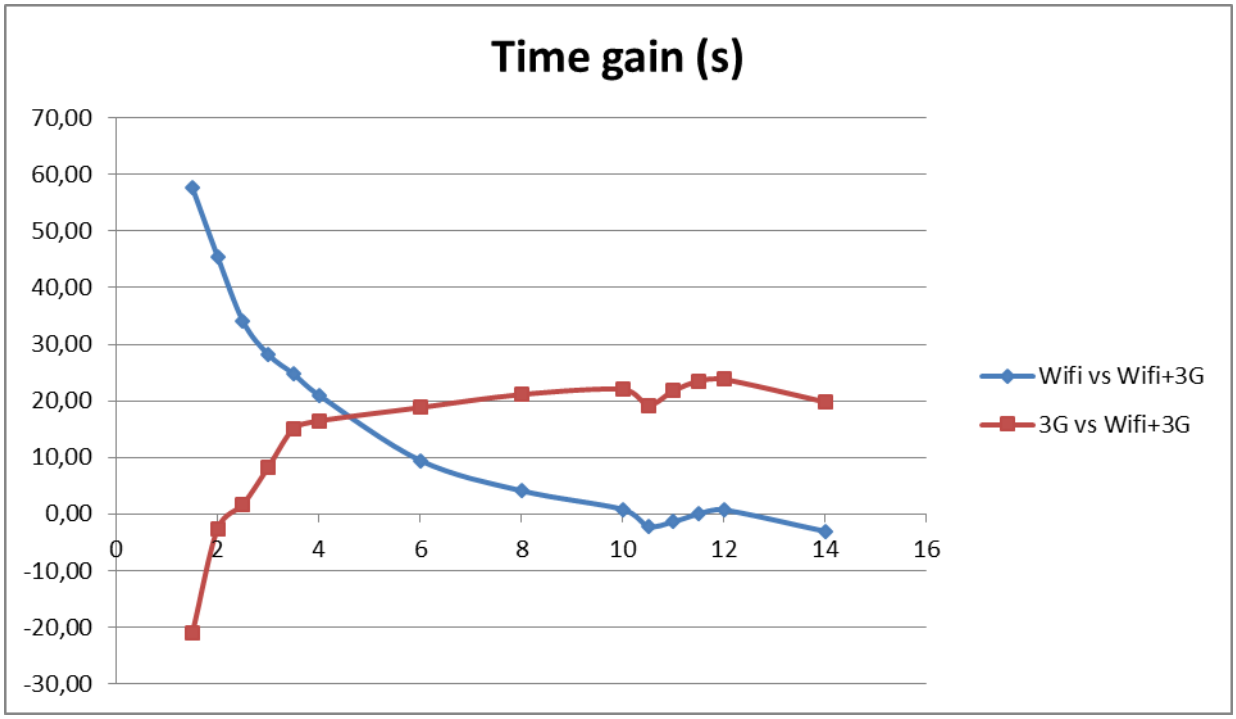

IMAGE 23. TIME GAIN COMPARISON OF THE SCENARIOS

As can be seen in the plot, the gain is negative when comparing the multipath scenario with the 3G network for low Wi-Fi bandwidth, so it was again better to use the 3G network instead of both links. As the bandwidth increased, the Wi-Fi link consumed less and less power and, from 6Mbps on, it was better to use only the Wi-Fi link.

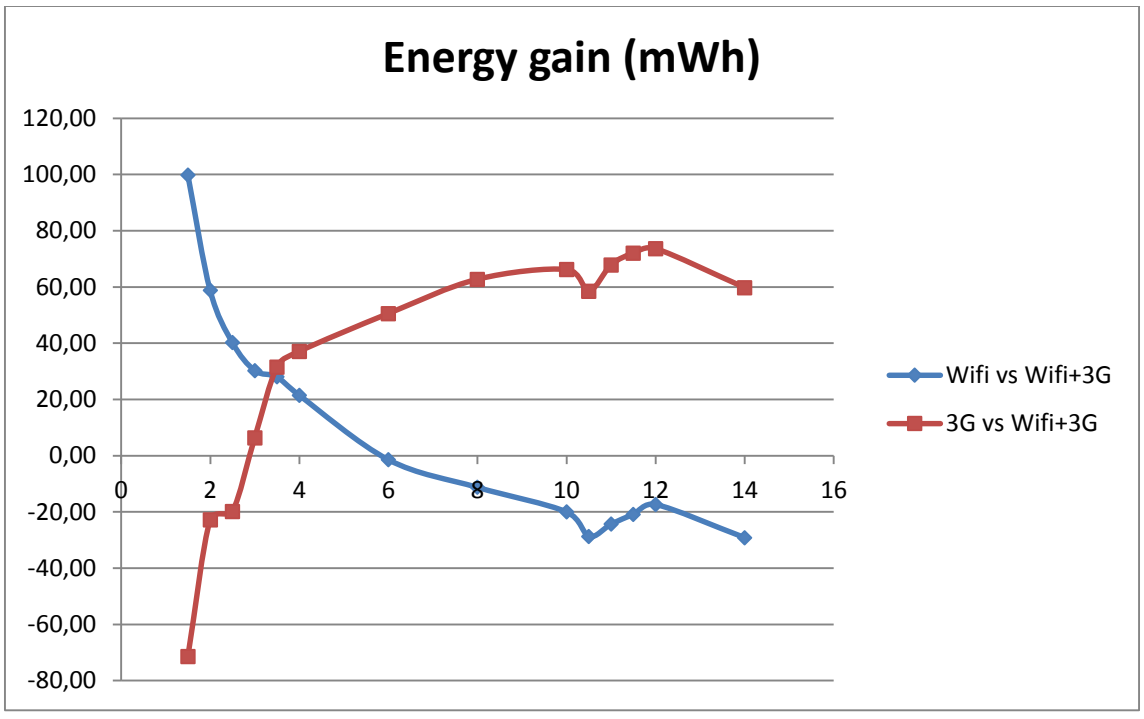

IMAGE 24. ENERGY GAIN COMPARISON OF THE SCENARIOS

Depending on the goal, the maximum Wi-Fi bandwidth for which it makes sense to combine it with the 3G network would be: 6Mbps according to energy consumed, or 10Mbps according to download time. With a Wi-Fi link of 2,5Mbps or slower, it is not efficient to use this connection, having better results using only the 3G network.

This means that this approach has a very small window of efficiency. If we wanted to distribute traffic among interfaces, a better distribution system is needed, and so a packet-based scheduling must be introduced.

4.2. PACKET-BASED SCHEDULING

This setup could not be tested as much as the first one, since it was in a very early stage and needed more development. But some measurements can give an idea of how useful and efficient this approach can be.

A general test was done with a speed meter application. From the visualization website mentioned in chapter 3, one of the tunnels was shut off to test the speed of a single link, one each time, and then both working together. As it was changed on the server and not on the phone, only the download value is relevant in these measurements. This way the command line does not need to be used, making it easy to use and present during an exhibition.

In this first image, the results from using only the Wi-Fi link can be seen. The download speed is 2Mbps, and it can also be seen that it shows the public IP of the server as the IP of the phone.

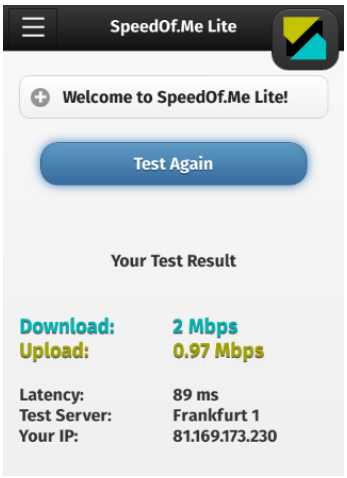

IMAGE 25. BUNDLING DOWNLOAD SPEED USING ONLY WI-FI

Then the Wi-Fi link was shut down and only the 3G link was used, giving a similar result of 1.95Mbps.

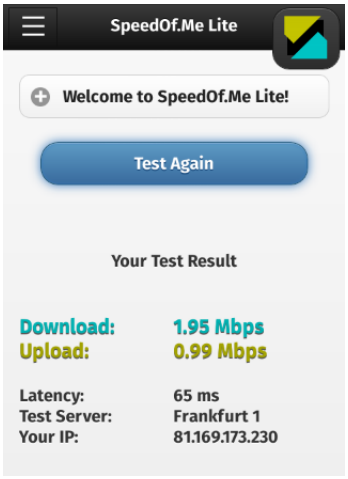

IMAGE 26. BUNDLING DOWNLOAD SPEED USING ONLY 3G

Finally, the two tunnels were activated, using a symmetric scheduling, which means basically that a round-robin scheduling was used. It can be seen that the speed was higher, 3.46Mbps, but not doubled. It was expected that the speed would be lower than the sum of the speed of the single links.

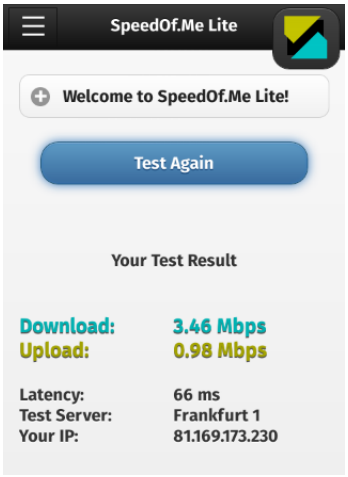

IMAGE 27. BUNDLING DOWNLOAD SPEED USING BOTH LINKS

This shows that both links have an advantage over a single link, but the trouble of having to send all the traffic to this server and then once it's merged continue its path to the requested service also has an impact on the results. In the next test it can be seen that using the Wi-Fi connection without the bundling is incredibly faster, but it must be taken into account that the bundling server was imposing a bandwidth limit of 2Mbps, so a real comparison would not be fair.

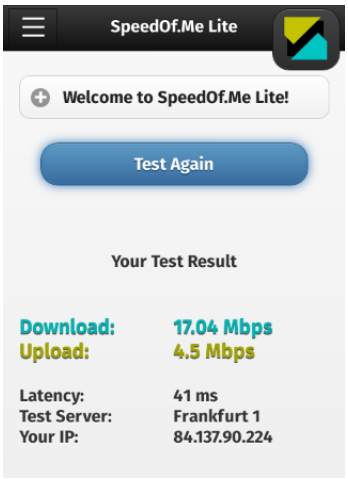

IMAGE 28. DOWNLOAD SPEED OVER A REGULAR WI-FI LINK

Next, using the MNI app and its button to download one 10MB file, the download time was measured using a bandwidth limit of 2Mbps. The values are collected in the next table.

Wi-Fi (time in seconds)	3G (s)	Bonding (s)
48.9	49.0	26.3
48.9	48.8	25.1
48.9	48.9	26.1
48.8	49.1	25.7

TABLE 3. BUNDLING DOWNLOAD TIME COMPARISON

The values were very consistent with the single link scenarios and vary a bit more using both, but it is visible that the overall result is a faster download.

In the following video [37] it is shown the traffic flow between the server and the phone in the visualization website. During the video, one of the links was shut down at each time to see how the speed is improved when using both. When the tunnels are up there is a green light showing they are working and the interfaces can be controlled, so the outgoing traffic uses only the interfaces selected. As was said, this is done only on the server side, downloading the files through the MNI app. The traffic going from the phone to the server is mainly acknowledgements for the packets received, and of course the HTTP communication to request the file.

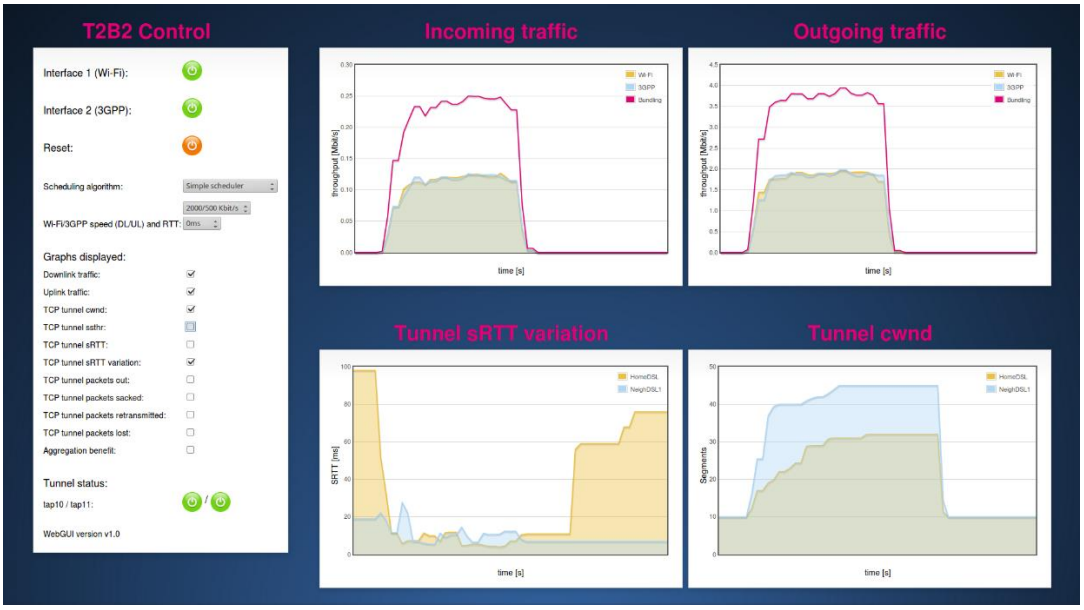

IMAGE 29. VIDEO DEMONSTRATION OF PACKET-BASED SCHEDULING [37]

4.2.1 Comparison of schedulers

It has been seen that using both links improves the speed over using only one, but it is necessary to find which scheduling policy obtains the best results and can adapt better to the environment, since bandwidth will most likely vary over time.

Some already existing policies were tested. The basic round-robin was the first one, changing the balance of each link to see which combination works better. For this, using ifenslave to select the weights, the scheduler policy was labelled according to them. For example, a round-robin policy with weight 3 on the Wi-Fi link and 1 for the cellular link is called "RR31". Other policies were also tested, like the cheapest pipe first (CPF) or different policies designed by the Darmstadt team modifying the standard ones. They can all be consulted in the code of the Python scheduler. Again the maximum bandwidth was limited to 2Mbps and a file of 10MB was downloaded to measure the time it took and the results are in the following table.

	Wi-Fi only	RR11	RR21	RR12	RR13	CPF
	49	25.7	36	32.7	37.5	48.2
	48.9	26.2	33.8	33.6	37.3	42.3
	48.1	26.4	34.8	32.2	38.6	47.9
	49.2	25.4	34	32.9	37.3	48.8
	49	25.4	33.5	32.7	37.1	35.5
	48.8	25.5	34.3	32.7	37.4	49
	48.1	25.6	34.5	32.3	37.1	48.9
	49.4	25.7	34.3	32.9	36.9	49.5
	49	25.5	34.1	32.8	37.2	48.9
	48.6	25.4	34.8	32.2	37.3	48.8
	47.6	27.1	33.8	33.2	37.5	48.7
	48.5	25.8	35.3	33.6	37.3	48.6
Average (s)	48.6833	25.8083	34.4333	32.8166	37.375	47.0916
Advanced1	Advanced5	Queue	Q2	Q3	Q3 (start at 5)	CPFadv2
31.1	35	41.6	36.8	33.7	29.2	29.6
31.2	34.8	44.7	37.1	34.1	29.7	25.3
32.2	39.2	39	39.5	35.4	27.9	30.2
31.3	39	43.3	37.1	34.1	29	31.1
28.6	34.5	36	36.9	36.9	29.2	28.1
32.1	39.2	39.8	36	34.5	30.5	25.7
30	39.3	39.3	38.7	36.7	32	27.5
31.5	39.8	42	34.6	34.6	28.2	26.3
29.7	33.6	35.4	39.3	34.1	29.7	28.1
30.9	37.1	34.3	35.4	36.1	29.9	28.8
29.6	36.6	39.2	37.1	36.2	31.9	28

29.5	37.4	35.2	35.2	37.9	29.3	27.9
30.64166667	37.125	39.15	36.975	35.3583	29.7083	28.05

TABLE 4. COMPARISON OF DIFFERENT SCHEDULING POLICIES WITH TRAFFIC BUNDLING

From the table, RR11 is the fastest one, but may only be because the other policies were not suited for this kind of environment. For example, the CPF uses the Wi-Fi link most of the time, since it is the "cheapest". The policy called "Advanced1" and others like "Queue" use high values for the weights, but not many packets are sent before the scheduler changes the values, which means it is not really useful since it is restarting the sequence too often. The policy called Queue3 (Q3) obtained better values when the weights were set to 5 at the start, and then it would adjust them from there, reducing the average in over 5 seconds. Finally, CPFadv2 didn't change the weights very often so the sequence could actually be traversed.

CHAPTER 5. CONCLUSIONS

5.1. SUMMARY

The demands for higher and more reliable download speed are steadily increasing. While the next hardware generation of mobile communications will help to meet those demands, software solutions have to be developed to complement the effort. The goal is to obtain a better throughput taking advantage of the technology already available. Offloading traffic of the already strained mobile networks is also a goal worth pursuing.

With this project, the intention was to improve the user mobile experience by offering better throughput and availability, and at the same time to relieve the network, distributing the traffic between the cellular link and a Wi-Fi link.

The platform chosen to tackle this issue was Firefox OS, an open source operating system based on Linux that provides a very deep access to the hardware and software characteristics of the device running it.

The first and very relevant step was finding the part of the source code in the operating system that needed to be changed in order to enable both Wi-Fi and cellular interfaces at the same time. The segments of code were identified and tweaked so the phone wouldn't shut the cellular connection off when Wi-Fi was available. For the case when Wi-Fi was active and the cellular interface was switched on, it needed to be activated and not kept idle regardless of the state of the Wi-Fi link. With this solution, the first goal of the project was completed.

A session-based approach was taken as a first setup to evaluate the efficiency of a multipath connection. It consisted on sending all the packets from one HTTP session through a single interface, while at the same time another session was sent through the other interface. A daemon was installed on the phone that would start running at boot (boot daemon), creating an HTTP socket that allows connection from inside the phone and from a PC through a visualization website. A simple mobile application allows controlling the process without the need of a command line interface. This app triggers the launch of a second daemon that reacts to the different buttons on the screen. This second daemon is the one that runs the scripts and commands to change routes or download files. The two options for downloads were either using a single link (Wi-Fi or 3G) or using both at the same time. The first option downloads two files of the same size using the regular path. The second option, where both interfaces work simultaneously, downloads again two files. These files have the same size as before, but each file is downloaded through a different interface. Through the boot daemon socket the information of download time can be seen on a visualization website that graphically

shows the progress of the download. As these tests were done with the Geeksphone Keon device and there was an ammeter available, measurements of battery consumption were possible too. The ammeter was connected via USB to the PC allowing the visualization website to access this information.

The results obtained from this setup show that this approach was not advantageous in all situations. When the Wi-Fi bandwidth was very poor, using both links improved the overall results, since the cellular bandwidth was quite stable. Downloading the two files through a weak Wi-Fi link consumed more battery and time. As the Wi-Fi bandwidth improved, using either both links or only Wi-Fi became very similar in terms of download time, therefore use of both links together does not pay off. In terms of battery, the consumption using both links became higher once the Wi-Fi link had a bandwidth over 6Mbps, but the difference was not significant. In the end, this approach was only efficient in very specific situations, where one of the links had a poor signal and the other one could offload the first by taking over half of the work. The first target setup defined at the beginning of the project was completed, although the results show a very restricted advantage.

There is another project within T-Labs that became of great interest at this point. The DSL Community project aims to share broadband home connections among clients. This means that one could use the extra bandwidth of a neighbor home when it is not being used. The research and tools developed for this project [20] could also be used in the mobile environment, since the basic idea is the same of combining multiple paths to serve a single user. This way, the traffic could be load-balanced between the interfaces without looking into which session the packets belong to.

Making use of the Linux bonding module, two interfaces could be combined in a single network device that will take care of distributing the traffic. But the bonding module on its own offers a limited amount of traffic scheduling policies. By modifying it to include a new mode, and with the aid of an also modified version of the Linux tool ifenslave, a wide range of policies can be designed for the scheduler. Ifenslave can now pass a value that contains two weights, one for each link, which indicate how many packets will be sent in a row through them. The bonding module will read this value in the new working mode, static scheduling, and send the packets through the links according to the weights. A scheduler programmed in Python is in charge of calculating the value following the scheduling policy chosen. Two tunnels were created to a server with a public IP, where the traffic is merged again. These tunnels encapsulate all traffic into TCP segments, which allows the kernel module TCP_probe to obtain information about congestion and TCP parameters, but it also means any kind of traffic can be sent through. Since the Firefox OS phone did not include a Python interpreter and none could be found that included the NumPy package, the scheduler had to run only on the server side, and the phone side was set to use a round-robin scheduling policy with the static mode. Testing showed that using both links improved the results to using a single link through the tunnel to the server, but it did not become faster than using a single link over the regular path. Through a

visualization website adapted from the one used in the DSL Community project, the bandwidth use can be seen in a chart. This website also allowed to see the information of congestion window, slow start threshold and other TCP parameters that will help define the scheduling policy most suited for this environment.

The goal of creating a setup similar to the one of DSL Community was accomplished with some drawbacks. The Python scheduler could not be implemented in the phone because of technical complications.

Designing a scheduling policy was one of the endpoints of this project, but due to lack of time it was not possible to develop enough tests to complete it.

5.3. PERSONAL VIEW

This internship has been one of the most valuable experiences in my life so far. I have learned not only new skills for my future career, but also about other cultures. I have become more independent in my work and in my life, enjoyed great teamwork and discovered ways to develop my potential.

Even if this were a small contribution to the efficiency of the network, if it is easy to implement, this mechanism would still be worth it. With more time, more and better tests of the packet-based scheduling could have been carried out. The bad relationship with mobile carriers resulted in Mozilla stating that they would not release any more devices through this channel, although they will continue with the development of the operating system for other devices [38]. This may be a setback for projects like this one, which rely on the openness of the system. But from a wider view, if this mechanism is to be deployed, in my opinion it shouldn't be limited to a single operating system, especially one that is not very popular among users.

The packet-based scheduling seems promising, but it also raises a lot of questions of how to actually implement it in a real environment and deploy it for all the users. It doesn't require the installation of new antennas, just the integration of a bonding server module, so installing new software into the network shouldn't represent too high a cost.

It would mean that all the user terminals that want to make use of this technology would need software modifications of the operating system.

It's interesting to think about which nodes would be modified. Do we need to include in the network structure these proxies to launch the other end of the tunnels as independent nodes or would they be encapsulated within the nodes of the cellular network? How far away from the client node would the merging of the traffic happen? One could think that the further away the longer two paths are used, keeping the throughput higher, and the moment the traffic is merged again it is reduced to that of a single link. But if all the traffic needs to travel to one of these servers and then be redirected to the desired content,

the longer the path to the merging server is, the longer it will take to serve the client. There are still a lot of issues to be resolved before users can enjoy this feature.

5.2. FUTURE STEPS

The first step for the next team to work on this project would most likely have to be thoroughly testing different scheduling policies, in order to design one that will adapt to the mobile communications environment. When a policy is found that yields better results than others, it will need to be optimized, so that the throughput is efficiently used and it improves the final user experience. The scheduler needs to be analyzed and investigate if a Python interpreter can be installed in the phone or if it would be better to use a different programming language, such as C.

The source code of Firefox OS should be modified further to fix the problems in the activation of both interfaces simultaneously, so that when the cellular connection is switched on after Wi-Fi the link is not kept idle.

A more advance issue would be to define how this mechanism would fit in the current communication networks and which elements need to be included or modified.

APPENDIX A: PLANNING AND RESOURCES

A.1 PLANNING

The phases of the project were divided according to the time available for development during the internship, while the last phases of analysis were scheduled for later.

The tasks can be differentiated as follows:

- Evaluation of the State of the Art. 2 weeks.
    - Research. 2 weeks.
    - Updates. Update the operating system and test changes in the source code. 1 week.
- Session-based scheduling.
    - Setup environment. 1 week.
    - Measurements. 2 weeks.
    - Evaluation of results. 1 week.
- Packet-based scheduling.
    - Test device support of tools. 2 week.
    - Setup bonding. 1 week.
    - Update devices. 4 days.
    - Modify tools. 4 weeks.
    - Setup full environment. 2 weeks.
    - Tests and measurements. 3 weeks.
    - Evaluation of results. 2 weeks.
- Analysis. 4 weeks.
- Documentation. 30 weeks.

In the next page these tasks have been represented in a Gantt diagram.

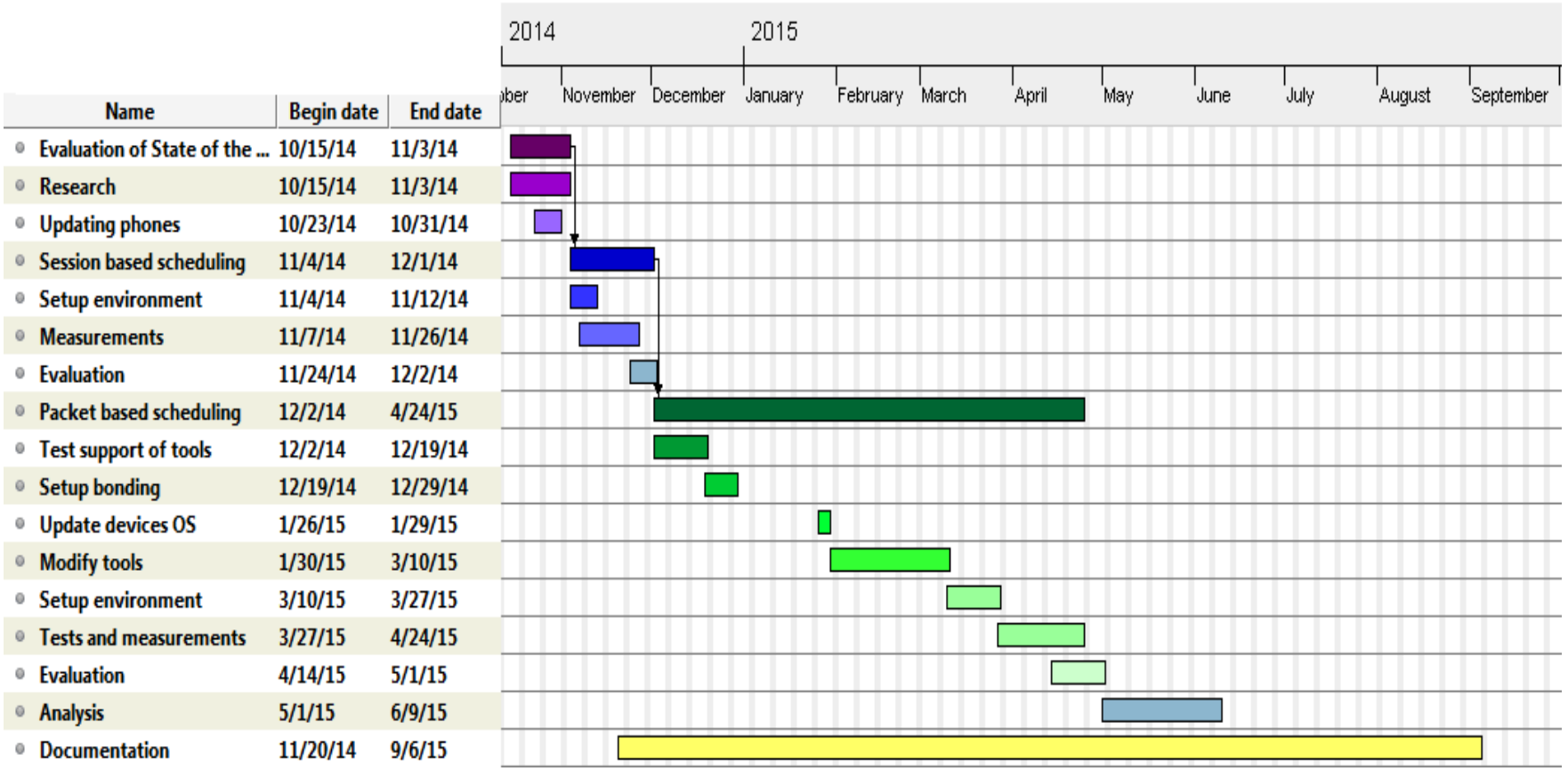

IMAGE 30. GANTT DIAGRAM OF THE PHASES OF THE PROJECT

A.2 Budget

This section contains the information of the costs that the project represented, both material and human resources.

Material resources

The costs of the material resources are considered taking into account the life expectancy of the devices, since they will be used for other projects after this one (or have already been used before). The laptops, displays and PCs are considered to have a life of three years. Mobile devices are considered to have a life of 1.5 years, because due to the tests performed on them for development they become very slow and limited faster than for personal use. The ammeter has a life of two years. The final cost is calculated with the use of six months in relation to the total life.

These costs refer only to the period of development in Berlin. It is not possible at this stage to evaluate the investment needed for deployment of this solution in a real environment until all the structural details are defined.

Item	Units	Cost/Unit (€)	Usage (%)	Total
Lenovo G580	1	800	16.67	133.36
Acer aspire r3700	1	300	16.67	50.01
Fujitsu-Siemens Lifebook S710	1	500	16.67	83.35
Fujitsu-Siemens Display B24W-5	1	150	16.67	25
Mozilla Flame	2	158.48	33.33	105.64
Geeksphone Keon	2	90	33.33	59.99
Powertool Ammeter Yocto-Amp	1	64.80	25	16.2
Mobile network access	6	10	100	60
Total				**533.55**

Table 5. Material expenses

Human resources

In the human resources department, the work day is considered to be of eight hours, five days per week and 22 days per month. The main engineer will work for six months, while the supervisor, a senior engineer will do the work of one month in total, providing guidance and support. A second engineer is available to assist with the problems during the project.

Category	Months	Salary/hour (€)	Total
Engineer	6	25	26,400
Senior engineer	1	40	7,040
Engineer	2	25	8,800
Total			**42,240**

Table 6. Human resources

TOTAL EXPENSES

The total expenses are contained in the next table, as a result of the material costs and the salaries of the different engineers involved in the development of the Project.

Resource	**Total**
Material	533.55
Human	42,240.00
Total	**42,773.55**

TABLE 7. TOTAL EXPENSES OF THE PROJECT

APPENDIX B: RESUMEN EN ESPAÑOL

Introducción

La conectividad es el fundamento de una Buena experiencia móvil. Mientras que las nuevas generaciones de comunicaciones móviles pueden aumentar la velocidad de conexión, beneficiarse de la tecnología ya existente mediante soluciones software puede suponer un complemento para mejorar la calidad de servicio. Mientras que aumenta el número de dispositivos conectados, el desafío se presenta en proporcionar una conexión segura y rápida con buena cobertura de todo el territorio.

La motivación para este proyecto es obtener la mayor eficiencia de la tecnología disponible hoy en día. Los dispositivos móviles generalmente disponen de dos modos de conectarse a Internet, Wi-Fi y red de datos, pero normalmente solo se emplea uno de ellos. Combinando la capacidad de ambos enlaces es posible aumentar el ancho de banda total, la disponibilidad y la movilidad que el usuario percibe.

Este Proyecto Fin de Carrera se basa en el desarrollo llevado a cabo durante una estancia de prácticas Erasmus Placement en las oficinas de Berlín de Deutsche Telekom Innovation Laboratories, dentro del departamento de Seamless Network Control, bajo la supervisión de Dr. Nico Bayer.

El objetivo principal es diseñar un entorno en el que ambas interfaces del dispositivo, Wi-Fi y móvil, puedan ser usadas al mismo tiempo, y donde el tráfico discurra por una de ellas indistintamente, como si se tratara de un solo enlace. El reparto del tráfico se puede hacer de distintas formas, pero se investigaron dos enfoques.

La primera cuestión a resolver es el modo de habilitar ambas interfaces simultáneamente, ya que normalmente siempre que el Wi-Fi está activo la conexión de datos se mantiene en reposo. Una vez se haya solucionado, las diferentes maneras de dirigir el tráfico podrán ser analizadas y probadas. El primer enfoque, un reparto simplemente en función de la sesión HTTP a la que pertenecen los paquetes se implementará. En este escenario se podrá evaluar cómo de eficiente es el uso simultáneo de las interfaces y en qué situaciones tiene sentido repartir el tráfico.

Una vez que se sabe si dividir el tráfico es eficiente, el siguiente objetivo será buscar un enfoque más inteligente para realizar el reparto. Algunas herramientas y mecanismos necesitarán ser modificados para poder lograrlo. Al separar el tráfico por sesiones, una de ellas puede acarrear el envío de un gran número de paquetes, mientras que el otro enlace puede estar casi vacío, o quizá uno de los enlaces tiene mala señal. Por esto es necesario encontrar una manera inteligente de planificar el tráfico teniendo en cuenta las condiciones del entorno.

Un objetivo secundario, pero que puede ayudar a llevar a cabo las demás tareas, es crear una aplicación simple que permita controlar el teléfono sin recurrir a una ventana de comandos.

Estado del arte

El proyecto incluye el uso de distintas tecnologías móviles como Firefox OS, 3G (UMTS) o Wi-Fi, así como herramientas y complementos de programación, como OpenVPN, el módulo de bonding del kernel Linux o JSON.

Firefox OS

Firefox OS es un sistema operativo de código abierto basado en Linux. Las WebAPIs hacen posible el acceso a las capacidades hardware del terminal, lo que es muy relevante para llevar a cabo el proyecto. La interfaz de usuario al completo es una aplicación web, capaz de mostrar y ejecutar otras aplicaciones web.

JSON

JSON es un formato de texto ligero para intercambiar datos. Será muy útil en el proyecto para transmitir información entre las distintas estructuras e identidades.

UMTS

La tercera generación de comunicaciones móviles comprende diversas aplicaciones en telefonía de voz inalámbrica, acceso móvil a Internet, acceso fijo a Internet, video llamadas y televisión móvil. Las redes de telecomunicación 3G soportan servicios y ofrecen una tasa de transferencia de al menos 200 kbps. Lanzamientos posteriores (3,5G y 3,75G) ofrecen acceso móvil de banda ancha de varios Mbps a smartphones y módems móviles. Cada generación móvil está caracterizada por nuevas bandas de frecuencia, mayores tasas de datos y tecnología de transmisión no compatible hacia atrás.

WLAN

Hoy en día todo smartphone es capaz de acceder a Internet a través de Wi-Fi. Una red inalámbrica de área local (WLAN) es una red sin cables que conecta varios dispositivos que usa un método de distribución inalámbrico (generalmente espectro ensanchado u OFDM) en un área limitada como un hogar, un colegio o una oficina. Esto permite a los usuarios moverse en el área de cobertura y seguir conectados a Internet.

NIC bonding

Network Interface Card (NIC) bonding es un método de lograr un ancho de banda mayor combinando múltiples interfaces en una sola interfaz lógica. Las interfaces unidas aparecen como un único dispositivo, compartiendo incluso la dirección MAC. Linux

posee un módulo que proporciona esta funcionalidad. Existen distintos algoritmos para balancear la carga, como por ejemplo, round-robin.

El bonding es un mecanismo muy importante para lograr los objetivos del proyecto, pudiendo combinar el tráfico de las dos interfaces de un teléfono.

OpenVPN

OpenVPN es una solución software segura para crear túneles entre VPN. Soporta diferentes configuraciones, incluyendo acceso remoto seguro a recursos de una red interna o privada. Es compatible con la mayoría de sistemas operativos y permitirá crear túneles a través de las interfaces del móvil.

DSL Community

El Proyecto DSL Community [11] está siendo desarrollado por Deutsche Telekom para suplir la mayor demanda de banda ancha. En su enfoque, el aumento de velocidad se consigue combinando las conexiones DSL de un usuario con la de varios vecinos cuando los éstos no las están usando. De este modo, los vecinos podrían usar el ancho de banda inactivo de otros clientes. Para esto es necesario que ambos vecinos posean un rúter inteligente [12].
El proyecto "Multipath Traffic Bundling with Firefox OS" puede considerarse un proyecto al que engloba DSL Community, ya que traslada los mismos mecanismos a un entorno móvil.

Desarrollo

Las distintas fases y pasos del desarrollo se describen a continuación, así como las modificaciones llevadas a cabo, del sistema operativo y distintas herramientas.

Una característica importante es la posibilidad de habilitar la interfaz Wi-Fi y la móvil al mismo tiempo. Este paso es imprescindible para poder lograr los objetivos.

Algunas aplicaciones y herramientas fueron instaladas o desarrolladas para controlar las características del teléfono, mientras que otras herramientas de Linux que podían ser útiles tuvieron que compilarse para la arquitectura específica del teléfono.

A continuación se describen los dos enfoques desarrollados y cómo se han llevado a cabo.

Planificación basada en sesiones

Inicialmente, dos dispositivos Geeksphone Keon estaban disponibles para el proyecto. El Keon fue uno de los primeros dispositivos para desarrollo de Firefox OS, por lo que es totalmente compatible.

El primer paso fue actualizar los teléfonos para acceder a la versión más reciente del sistema operativo y realizar los cambios necesarios. Una vez se hicieron los cambios, el

sistema estaba listo para compilar, lo que tiene unos requisitos mínimos para el PC que realiza la compilación.

Este proceso se hizo siguiendo los pasos especificados en la web de Mozilla, pero para continuar eran necesarios algunos cambios más. Era necesario hacer ciertas modificaciones en el código fuente, así como en las opciones del kernel.

Una de las modificaciones más importantes era la que permite utilizar ambas interfaces al mismo tiempo para la conexión a Internet. Hoy en día, los teléfonos móviles no permiten mantener ambas interfaces activas al mismo tiempo, excepto por contadas situaciones, como para grandes descargas de datos. Gracias a Firefox OS es posible acceder al código fuente y modificarlo fácilmente.

El modo en que se modificó el código es algo tosco y poco elegante, pero es suficiente para comenzar a realizar experimentos. El cambio se debe hacer antes de compilar el sistema.

Hay dos segmentos de código que se corresponden con dos situaciones del estado de las interfaces. La primera situación sucede cuando se establece una conexión Wi-Fi después de que la conexión de datos esté activa. Normalmente la conexión de datos quedaría en reposo, sin ser utilizada, por lo que se requiere que se mantenga activa, sin borrar los datos de ruta de la tabla de enrutado. La segunda situación se produce cuando el Wi-Fi se encuentra en funcionamiento y es la conexión de datos la que se activa después. En lugar de mantener la conexión de datos inactiva, se debe activar e insertar la información de rutas.

Tras realizar los cambios se puede volver a compilar el sistema operativo e instalado en el teléfono.

El primer enfoque consistía en separar el tráfico según la sesión HTTP. De esta forma se podían realizar dos descargas del mismo tamaño, simulando un reparto de tráfico equilibrado a la mitad. Todos los paquetes de una misma sesión se envían por una sola interfaz.

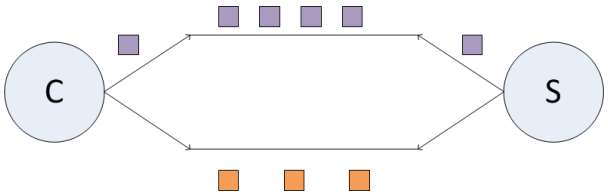

IMAGE 31. PLANIFICACIÓN BASADA EN SESIONES

Para estas pruebas fue muy importante la aplicación móvil MNI. A través de un socket local la web de visualización podía obtener información sobre el consumo de batería gracias al amperímetro. Un programa daemon (bootdaemon) que se inicia al encender el teléfono es el que transmite la información de tráfico recibido y enviado. En la aplicación MNI otro daemon se encarga de ejecutar los scripts que modifican la tabla de rutas e iniciar las descargas.

Para poder dirigir cada sesión por cada interfaz, se tuvieron que descargar dos archivos de distintos orígenes con el mismo tamaño, de forma que se pudiera añadir una ruta distinta para cada archivo. Uno de los archivos se descargaría por la ruta por defecto (Wi-Fi), mientras que para el otro se introdujo la IP de destino, marcando como el siguiente salto la interfaz móvil.

La aplicación MNI es la que se encargaba de hacer todos estos cambios de rutas. En la aplicación además se incluyen dos botones para encender o apagar las interfaces sin tener que acceder a los ajustes del teléfono. Otro botón es el que inicia el daemon de la aplicación, que debe ser ejecutado antes de realizar cualquier otra acción, porque es el que se encargará de llevar a cabo las mismas.

Al pulsar un botón se genera un objeto JSON que contiene el comando a ejecutar por el daemon, que recibe este mensaje.

El bootdaemon crea un socket en el Puerto 5000. Cuando la aplicación MNI se inicia, se establece una conexión con este socket.

Desde ese momento se puede pulsar el botón que ejecuta el segundo daemon, appdaemon. Sin este daemon ninguno de los otros botones será reconocido, por lo que pulsarlos no produciría ningún resultado. Es el bootdaemon el que se encarga de ejecutar el appdaemon, o parar su ejecución.

El bootdaemon también se encarga de recibir las conexiones desde el PC, para transmitir la información de datos transmitidos y tiempo de descarga.

El appdaemon crea un socket en el Puerto 3000. Este daemon espera recibir comandos de la aplicación, que se generan al pulsar los botones.

Los botones interesantes para esta etapa del proyecto son los de la sección "Access Bundling". El primero, "Simple", realiza dos descargas, cada una de un archivo de 10MB mediante la herramienta wget. Las dos descargas se enrutan por una misma interfaz, dependiendo de cuál esté activa.

La opción "Multipath" requería varias operaciones más, ya que es necesario modificar las rutas. Esto se hace en el primer paso, mediante un script que añade la ruta del segundo archivo a través de la interfaz móvil. Cuando las rutas se han actualizado se descargan de nuevo dos archivos de 10MB cada uno, pero con IP distintas. Si las IP donde se aloja el archivo no fueran distintas no sería posible separar el tráfico entre las dos interfaces. Una vez que han terminado las descargas, las rutas vuelven a cambiarse a su estado original para que no modifiquen los resultados de otras pruebas.

Como se ha mencionado, la web de visualización permite ver los resultados de las descargas en tiempo real, recibiendo la información de tiempo y bytes descargados del bootdaemon y la de consumo de batería del amperímetro. La web se conecta al socket del

puerto 5000 a través de Wi-Fi, por lo que el PC debe estar en la misma subred que el móvil, mientras que la información de batería se obtiene a través de un socket local, ya que el amperímetro se conecta mediante USB al ordenador.

Los datos recibidos de ambas fuentes se representan en gráficas. Mientras, se calcula el total de datos transmitidos y el consumo total de batería. Esta información del total se muestra al terminar la descarga.

Planificación basada en paquetes

En lugar de transmitir los paquetes sistemáticamente por una interfaz basándose en la sesión a la que pertenecen, se necesitaba un algoritmo más inteligente para separarlos. Los paquetes no necesitan seguir todos el mismo camino a través de la red, solo es necesario que sea posible ordenarlos al llegar al destino. Se buscó una manera de dividir el tráfico tratando cada paquete de manera individual para enviarlos por una interfaz u otra y a la vez poder unirlos de nuevo en el destino. Esto implica que debe haber una conexión entre el cliente (teléfono) y el servidor a través de ambas interfaces, con ambos extremos siendo conscientes de ello, pero a la vez tratando la situación como si fuera una sola interfaz.

Se vuelve a hablar de DSL Community, donde se une el ancho de banda de dos conexiones para mejorar el servicio prestado. El entorno descrito en [29] es el modelo que se siguió y adaptó para este escenario móvil.

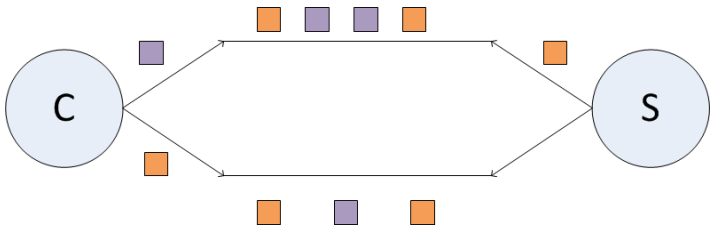

IMAGE 32. PLANIFICACIÓN BASADA EN PAQUETES

Era necesario comprobar que la tecnología usada en [29] sería soportada por el terminal móvil, por lo que fue necesario hacer pruebas.

Uno de los elementos más importantes era el módulo de bonding del kernel, que permite unir varias interfaces en una sola. Puesto que el kernel del teléfono es un kernel Linux, debía ser posible disponer de él, pero en la versión ofrecida por Mozilla del sistema operativo no se crea el módulo durante la compilación. Debido a la poca información disponible sobre Firefox OS al ser un sistema nuevo, lograr este módulo fue un proceso de prueba y error. Al final se optó por replicar la configuración de un PC Linux respecto a bonding, para después compilar el sistema.

Al compilar se genera un archivo con extensión ".ko" en la carpeta /system/lib/modules del teléfono, donde se guardan todos los módulos compilados. Una vez se instala el

sistema en el teléfono el nuevo módulo puede ser utilizado. A pesar de que el módulo es funcional, se notó que existían algunas restricciones, como que no soportaba todos los modos de funcionamiento.

Una vez era posible usar el módulo de bonding, la idea era crear dos túneles, uno a través de cada interfaz, que serían combinados en una interfaz de bonding creada mediante el módulo. Esta nueva interfaz sería la que actúe de Gateway para todo el tráfico. Era necesario crear túneles porque las interfaces directamente no era posible acoplarlas a la interfaz de bonding.

Pero los túneles necesitan ser creados en ambos extremos para establecer la conexión. Es por esto que fue necesario un servidor en el otro lado, que combine de nuevo el tráfico separado de los túneles y de ahí transmitir el tráfico a su destino final. Para poder acceder a ese servidor era necesario que dispusiera de una IP pública que fuera visible desde la situación del teléfono.

Antes de usar un servidor con IP pública, se comprobó el soporte de los teléfonos de la herramienta para crear los túneles. La herramienta elegida para esta tarea fue OpenVPN, que proporcionaba la funcionalidad necesaria, es gratuita y además ofrece la posibilidad de que los túneles estén encriptados.

Se encontró una versión compilada de la herramienta compatible con la arquitectura del teléfono. OpenVPN ofrece distintas formas de configurar y crear los túneles, pero el teléfono no soporta el modo mediante archivos de configuración, por lo que se usó el método mediante línea de comandos.

Existía la posibilidad de crear túneles UDP o TCP. Se decidió usar encapsulación TCP porque permitiría obtener información del estado de los túneles, pudiendo monitorizar la ventana de congestión y otros parámetros TCP. Lógicamente, esto añade un extra de encapsulación a los paquetes, además de requerir que se acuse el recibo de cada uno de ellos. Pero por otro lado, permite que se envíe cualquier tipo de tráfico dentro de ellos, no sólo TCP.

Debido a las pruebas similares hechas para DSL Community, el servidor con IP pública estaba configurado para usar túneles de tipo tun, por lo que se trató de usar este tipo con los túneles del teléfono, pero el comportamiento era incorrecto. La dirección MAC no se estaba manejando correctamente, por lo que los túneles no funcionaban. Por esto se cambió al otro tipo de dispositivos, tap, que funcionan a nivel Ethernet.

La aplicación MNI fue modificada para experimentar con la nueva funcionalidad. Tras las modificaciones era posible especificar las IP locales y remotas de las interfaces de bonding y los túneles, así como crearlos simplemente pulsando un botón.

Esto se hizo de nuevo mediante comandos creados con JSON. Las IP y puertos se leen de los cuadros de texto, que además contienen un valor por defecto, y se envían al appdaemon, también modificado. Al pulsar el botón se establecen los túneles y se asignan como esclavos de la interfaz de bonding configurada.

Cuando se pulsa el botón, se generan varios comandos, que desencadenan distintas acciones del daemon. Con una variable se señalizaba si los túneles debían ser activados o destruidos.

Una web de visualización proporcionada por la oficina de Darmstadt permitía observar el progreso del tráfico a través de los túneles. Era una modificación de la usada en DSL Community, mostrando el tráfico visto desde el servidor. Desde la web se podía controlar qué túneles estaban activos, aunque sólo desde el lado del servidor, por lo que el tráfico llegaba por ambos túneles, pero el servidor enviaba por los que estuvieran activos. Además, se podían visualizar otros parámetros TCP.

Para poder hacer la planificación más inteligente fue necesario modificar las herramientas que controlan el bonding, ifenslave y el módulo de bonding.

Ifenslave se encargaba de asignar los túneles como esclavos de la interfaz de bonding. Con las modificaciones podría además transmitir un valor recibido del usuario que representaría los pesos que debe usar la interfaz de bonding para cada túnel.

Además se desearía incluir un planificador, originalmente programado en Python. Este planificador lee la información del tráfico y las condiciones del estado de los túneles gracias a otro módulo del kernel, TCP_probe. Con esta información calcula la proporción en la que el tráfico debe ser repartido entre los túneles. Esta proporción son los pesos que serán traspasados a ifenslave.

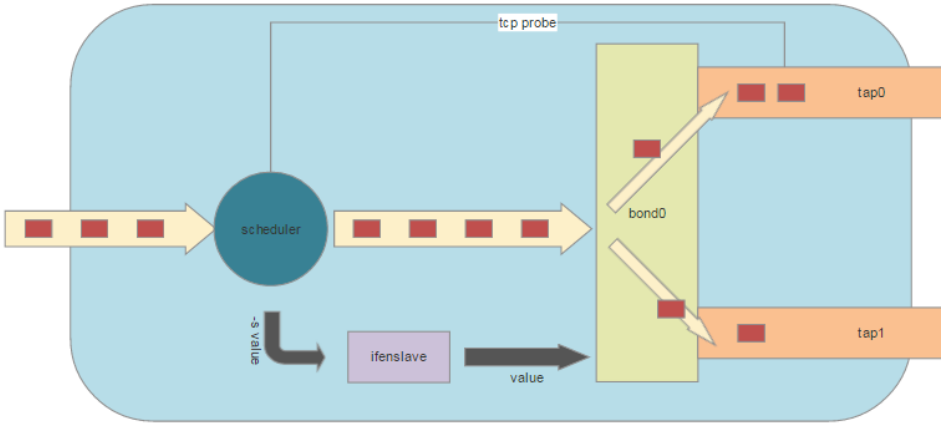

IMAGE 33. ESTRUCTURA INTERNA PARA PLANIFICACIÓN BASADA EN PAQUETES

Para que ifenslave sea capaz de recibir esta información tuvo que ser modificado, para que además vuelva a transmitir esta información al módulo de bonding, que también requirió cambios.

Ifenslave es una herramienta Linux que permite agregar y desagregar dispositivos de red esclavos de una interfaz de bonding maestra. Con un solo comando, varias interfaces se pueden agregar al dispositivo de bonding. Para el proyecto, lo que se necesita es precisamente esto, para combinar las dos interfaces del teléfono. Pero el módulo de bonding del que dispone el teléfono solo admite una política de planificación round-robin.

Una nueva opción se introdujo en el código fuente de ifenslave, especificando los pesos de cada túnel mediante dos dígitos. Cada uno de los dígitos representa el peso de paquetes de cada túnel. Este valor se pasa al módulo de bonding junto con el nombre de la interfaz maestra mediante la generación de una interrupción.

El módulo de bonding requiere un nuevo modo de funcionamiento para poder usar los pesos calculados por el planificador, para así poder usar una política que se adapte mejor al entorno móvil.

El nuevo modo se llamó “static scheduling”. Los paquetes se envían de acuerdo a una secuencia creada a partir de los valores de pesos recibidos del planificador.

Experimentos y Resultados

Planificación basada en sesiones

Las medidas tomadas se obtuvieron descargando dos archivos de 10MB con el uso de la aplicación MNI. El amperímetro proporcionó los datos de consumo de energía. Y gracias a los daemon se obtuvo la información de tiempos de descarga. Se compararon los resultados obtenidos para tres casos distintos: uso de Wi-Fi únicamente, uso de red móvil únicamente y uso de ambos enlaces simultáneamente.
El ancho de banda del Wi-Fi fue modificado creando un punto de acceso propio, mientras que el ancho de banda de la red móvil no se podía modificar. Para cada caso y cada valor de ancho de banda se tomaron diez medidas, obteniendo un valor medio para el análisis.
Los valores de consumo de energía se midieron en mili vatios por hora (mWh), ya que es una unidad típica para baterías.
Al comparar los tres casos, se puede ver que usar el enlace 3G únicamente ofrecía peores resultados que los otros dos casos, excepto cuando el ancho de banda del Wi-Fi es muy bajo. El ancho de banda 3G variaba entre los 4 y 6 Mbps, con lo que en el momento que el Wi-Fi superaba este valor, el uso del enlace 3G por sí solo resultaba más lento.
Cuando los dos enlaces se utilizan al mismo tiempo, el rendimiento mejoraba, especialmente para anchos de banda de Wi-Fi bajos. Alrededor de los 10-12 Mbps el uso del enlace de Wi-Fi por su cuenta resultaba más provechoso que el uso conjunto.
Los resultados de consumo de energía son muy similares a los de tiempo de descarga. Para valores bajos de ancho de banda, el Wi-Fi consume más batería que el 3G o los dos enlaces al mismo tiempo.
Al final, cuando el ancho de banda del Wi-Fi estaba por debajo de los 2,5Mbps, era más rápido el uso de 3G únicamente. A partir de ese valor y hasta los 10Mbps, era más beneficioso el uso de ambos enlaces simultáneamente. Para valores más altos el enlace Wi-Fi era más rápido que las otras dos opciones.
Dependiendo de los requerimientos, el máximo ancho de banda de Wi-Fi para el que tendría sentido combinarlo con el enlace 3G sería 6Mbps según el consumo de batería o 10Mbps según el tiempo de descarga. Cuando el ancho de banda está por debajo de los 2,5Mbps no tiene sentido emplear el enlace Wi-Fi y conviene emplear solo el enlace 3G. Por lo tanto este enfoque tiene una ventana de eficiencia muy pequeña.

Planificación basada en paquetes

Por limitaciones de tiempo este enfoque no pudo ser evaluado por completo. Aún necesita más desarrollo, pero unas primeras pruebas pueden dar una idea de su utilidad.
Mediante una aplicación de terceros se midió la velocidad de la conexión en distintas situaciones del escenario. Desde el servidor se seleccionó si se envía tráfico por uno de los túneles o por ambos, siempre limitando el máximo a 2Mbps. Los resultados obtenidos empleando solo el túnel Wi-Fi son de una velocidad de descarga de 2Mbps. Al usar el

enlace 3G, se obtuvo una velocidad de 1,95Mbps. Mientras que empleando ambos enlaces se obtiene un resultado de 3,46Mbps. Como era de esperar, el uso de los dos enlaces no conlleva la suma de sus velocidades, simplemente mejora respecto al uso de un solo enlace.
Una segunda prueba consistió en usar la funcionalidad implementada en la aplicación MNI, descargando un archivo de 10MB y midiendo el tiempo que lleva la descarga. De nuevo se estableció el máximo de velocidad en 2Mbps para poder visualizar fácilmente el progreso.

Los resultados muestran una mejora en el tiempo de descarga de casi la mitad usando el doble enlace. A pesar de eso, sigue siendo necesario encontrar la política de planificación logra la mayor eficiencia en este escenario, adaptándose a las condiciones del entorno móvil, puesto que el ancho de banda será variable.

El planificador implementa varias políticas que fueron probadas para el entorno móvil. Se probó distintas configuraciones de round-robin, cambiando los pesos para que uno de los enlaces soporte más tráfico que el otro. Para todas las políticas se hicieron varias descargas de 10MB para obtener un valor medio. Ninguna de las políticas obtuvo resultados destacables sobre los demás, tanto que la que obtiene los mejores resultados es la política round-robin simétrica.

REFERENCES

[1] D. Nowoswiat, E. Elkin, *Take full advantage of LTE with VoWi-Fi*, Techzine Tech Blog, February 23, 2015.

[2] Download Booster. ¿Qué es?, September, 2015, http://www.samsung.com/es/support/skp/faq/1051016

[3] Alcatel-Lucent, *Blending Wi-Fi and cellular: Introducing Wireless Unified Networks*, Strategic White Paper, March, 2015.

[4] Deutsche Telekom Innovation Laboratories, July, 2015, http://www.laboratories.telekom.com/public/English/ueber_uns/Pages/default.aspx

[5] Deutsche Telekom Innovation Laboratories Accelerator, October, 2015, http://www.laboratories.telekom.com/public/SiteCollectionDocuments/accelerator.html

[6] *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update*, 2015–2020 White Paper, February, 2016.

[7] N. Dobberstein, H. Wwang, N. Menon, A. Dixon, M. Rana, S. G. Chua, N. Yamamoto, *The Mobile Ecosystem in Asia Pacific*, A. T. Kearney, 2011.

[8] Nokia Siemens Networks, *Signaling is growing 50% faster than data traffic*, White Paper, 2012.

[9] Qualcomm Incorporated, *A 3G/LTE Wi-Fi Offload Framework: Connectivity Engine (CnE) to Manage Inter-System Radio Connections and Applications*, White Paper, June, 2011.

[10] S. Dimatteo, P. Hui, B. Han, and V.O.K. Li, *Cellular Traffic Offloading through WiFi Networks*, 2011 IEEE 8th International Conference on Mobile Adhoc and Sensor Systems (MASS), 2011.

[11] Firefox OS – Mozilla | MDN, October, 2015, https://developer.mozilla.org/es/Firefox_OS

[12] Firefox OS phones, October, 2015, https://developer.mozilla.org/en-US/Firefox_OS/Phone_guide/Phone_specs

[13] Firefox Flame, January, 2015, https://developer.mozilla.org/en-US/docs/Mozilla/Firefox_OS/Phone_guide/Flame

[14] JSON, November, 2014 http://www.json.org/index.html

[15] JSON – JavaScript | MDN, November, 2014, https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/JSON

[16] *Special Mobile Group (SMG); Work programme for the standardization of the Universal Mobile Telecommunications System (UMTS) (UMTS 00.01)*, ETSI, August, 1996.

[17] LTE, December, 2014, http://www.3gpp.org/technologies/keywords-acronyms/98-lte

[18] UMTS, December, 2014, http://www.3gpp.org/technologies/keywords-acronyms/103-umts

[19] UMTS and 3G FAQ page, November, 2014 http://www.umtsworld.com/umts/faq.htm

[20] El Sistema de móviles UMTS, November, 2014, http://www.ramonmillan.com/tutoriales/umts.php

[21] *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*,

[22] Linux Ethernet Bonding Driver HOWTO, February, 2015, https://www.kernel.org/doc/Documentation/networking/bonding.txt

[23] What is NIC bonding? A Webopedia Definition, July, 2015, http://www.webopedia.com/TERM/N/NIC_bonding.html

[24] Understanding NIC bonding with Linux, July, 2015, http://www.enterprisenetworkingplanet.com/linux_unix/article.php/3850636/Understanding-NIC-Bonding-with-Linux.htm

[25] OpenVPN – Open source VPN, February, 2015, https://openvpn.net/images/OpenVPNAccessServerDataSheet.pdf

[26] T. Le, L.X. Bui, *Forward Delay-based Packet Scheduling Algorithm for Multipath TCP*, arXiv:1501.03196 [cs.NI], 2015.

[27] A. Singh, C. Goerg, A. Timm-Giel, M. Scharf, *Performance Comparison of Scheduling Algorithms for Multipath Transfer*, Global Communications Conference (GLOBECOM), 2012 IEEE, December, 2012.

[28] D. Jurca, P. Frossard, *Video Packet Selection and Scheduling for Multipath Streaming*, IEEE Transactions on Multimedia, Vol. 9, No. 3, April, 2007.

[29] J.C. Fernández, T. Taleb, K. Hashimoto, Y. Nemoto, N. Kato, *Multi-path Scheduling Algorithm for Real-Time Video Applications in Next-Generation Wireless Networks*, 4th International Conference on Innovations in Information Technology, 2007.

[30] DSL Community, October, 2015, http://www.laboratories.telekom.com/public/english/innovation/exponate/pages/dsl-community.aspx

[31] Video "DSL Community -- higher speed through access bundling powered by T-Labs", November, 2015, https://www.youtube.com/watch?v=yK_ULmPP2_0

[32] Building and installing Firefox OS – Mozilla | MDN, October, 2014, https://developer.mozilla.org/en-US/Firefox_OS/Building_and_installing_Firefox_OS

[33] Rickshaw toolkit for JavaScript, October, 2014, http://code.shutterstock.com/rickshaw/

[34] N. Bayer, A. Ghazzawi, *Investigation of resource bundling in Wireless Access Networks*, Telekom Innovation Laboratories, 2014.

[35] Private Cloud Bonding Service – Zero Outages, May, 2015, http://vodc.zerooutages.com/vodc/services_VPN_Bonding.xrn

[36] Ifenslave manual page, January, 2015, http://manpages.ubuntu.com/manpages/raring/man8/ifenslave-2.6.8.html

[37] Video "Traffic Bundling with Firefox OS", March, 2015, https://youtu.be/Lf9QKVpEmG8

[38] La libertad mató a Firefox OS, December, 2015, http://www.xataka.com/moviles/la-libertad-mato-a-firefox-os