Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Design and development of a smart car gateway

MASTER DEGREE: Master of Science in Telecommunication Engineering & Management

AUTHOR: Adrià López Molina

DIRECTOR: Rafael Vidal Ferré

DATE: October 27th 2016

Títol: Design and development of a smart car gateway

Autor: Adrià López Molina

Director: Rafael Vidal Ferré

Data: 27 d'octubre de 2016

Resum

A dia d'avui, el nou paradigma de l'Internet de les coses (*Internet of Things*) està al focus de la investigació i innovació, suposant un canvi en el com interactuarem amb el nostre entorn. Aquest paradigma es basa en la connectivitat total dels dispositius i instruments que diàriament fem servir.

Dins d'aquest marc, aquest projecte té com a objectiu l'estudi de solucions que ens permetin millorar la connectivitat dels vehicles i dels seus ocupants traient profit de l'accés a més d'una tecnologia sense fils.

Concretament, l'estudi que presenta el document es basa en l'anàlisi dels protocols Multipath TCP i Mobile IPv6 (NEMO) amb la finalitat de poder suportar la comunicació en entorns de mobilitat. Ambdós protocols serviran com a base pel disseny d'una porta d'enllaç entre el vehicle i l'entorn que pugui suportar la millor connectivitat possible en aquest tipus d'escenaris.

A més, es proposa l'ús d'informació de tipus Open Data, principalment oferta per organitzacions territorials o proveïdors de serveis, que ens ajudarà a aportar un valor afegit a la pressa de decisions i a les funcionalitats de la porta d'enllaç.

Finalment, es detalla un cas d'ús en el qual es busca poder aplicar el coneixement adquirit per tal d'aterrar els resultats teòrics i empírics a una situació quotidiana. Partint de la definició d'un algoritme que permetrà a la porta d'enllaç conèixer les diferents casuístiques que poden donar-se a una ruta aleatòria, l'objectiu és que el sistema sigui capaç de decidir en cada moment on es garantirà la millor connectivitat tenint en compte d'altres factors com, per exemple, els punts de connexió Wi-Fi dins la ruta i les situacions on poder estalviar consum de dades sense perdre qualitat en la descàrrega de continguts per part de l'usuari.

Title: Design and development of a smart car gateway

Author: Adrià López Molina

Director: Rafael Vidal Ferré

Date: October, 27th 2016

Overview

Nowadays, the new paradigm of the *Internet of Things* is on the focus of research and innovation, leading to a change in how we interact with our environment. This paradigm is based on the full connectivity for all devices and instruments that we use daily.

Within this context, this project aims to study solutions that allow us to improve the connectivity of vehicles and their occupants taking advantage of the access to more than one wireless technology.

Specifically, the study is based on the analysis of the protocols Multipath TCP and Mobile IPv6 (NEMO) in order to be able to support communications in mobile environments. Both protocols will serve as a basis for the design of a gateway between the vehicle and the environment that can support the best possible connectivity in such scenarios.

In addition, Open Data information is proposed to be used, datasets mainly offered by regional organizations or service providers, in order to help us to bring an added value to the decision process and new functionalities to the gateway.

Finally, a use case is presented with the main objective to apply the acquired knowledge in an everyday situation. Based on an algorithm definition that takes care of the different situations that can occur in a random route, the gateway can decide the best option to ensure the optimum connectivity. This is done taking into account factors like, for example, the presence of Wi-Fi hotspots in the route or the situations where data consumption savings can be achieved without losing quality in content downloading.

INDEX

INTRODUCTION

Connected cars look set to be the consumer devices that are going to be one of the driving forces behind the Internet of Things.

According to many reports, as the Gartner's one published on January 2015, by the end of the decade, 250 million 'smart' vehicles will be on the world's road, offering drivers mobile internet access, communication capabilities between other vehicles and road infrastructure and the first mass-market elements of autonomous driving.

One of the first regulated projects of this kind in the EU is *eCall* [1]. In 2015, the European Parliament approved a new regulation regarding this technology, which requires all new cars be equipped with *eCall* system from April 2018. In case of a crash, an eCall-equipped car automatically calls the nearest emergency centre and, in case that nobody can talk, also sends the exact location of the accident.

Another real case is *MirrorLink*, a device interoperability standard that offers integration between a smartphone and a car's infotainment system. Today this system is implemented in some car models, but from November 2015 ETSI and the Car Connectivity Consortium signed a co-operation agreement in which the first one will formally explore adopting *MirrorLink* as en ETSI Technical Specification [2].

Nowadays, car manufacturers as Opel are mounting smart car systems in their vehicles. Opel OnStar [3] provides personal connectivity and service assistant in some models and allows the users to connect up to seven devices to the car network, download routes and traffic information and get real-time information about car conditions and future maintenance actions.

Following and supporting these approaches, this thesis explores the analysis of capabilities and requirements in order to design a smart car gateway that allows the driver to take advantage from the connected car, as long as we can obtain the best possible connectivity.

In order to do this, TCP/IP protocols Multipath-TCP and Mobile IP are proposed and analysed considering the mobility particularities that implies an urban scenario with 3G/4G and Wi-Fi coverage.

First chapter is focused on presenting both protocols and its capacity to take profit of two radio interfaces. Then, the benefits that open data could give to the system are presented. The objective is to contextualize these main lines and provide the theoretical base for the following chapters.

On the other hand, the technical requirements needed for the chosen protocols implementations are presented in the second chapter in order to study the technologies behaviour in a test scenario.

Third chapter presents the results of NEMO and MPTCP performance tests, alone or combined. It starts by presenting the test bed where the tests are done and continues, talking about its basic settings, the signalling messages for each protocol and the customization of the fixed part of the network, in order to reproduce a realistic scenario. Finally, it presents the obtained results for the protocols performance in the test bed.

Meanwhile, chapter 4 presents a decision algorithm that, with the help of external open data sources and using the acquired knowledge in the performance study, tries to determine the best moment to do a handover in a random route. Afterwards, this algorithm is applied in a particular case in a particular route in the city of Barcelona.

In addition, this document includes two annexes. In the first one, the setting methods of MPTCP and NEMO are extended. Test bed preparation, configuration scripts and first tests done are also shown. Moreover, the second annex presents more open data examples and an implementation of specific operating system for the connected car.

CHAPTER 1. INITIAL ANALYSIS

1.1. Protocols

1.1.1. Multipath TCP

Multipath TCP is a set of TCP extensions defined in RFC 6824 [4] that allows a single TCP connection to send and receive data using different IP addresses simultaneously.

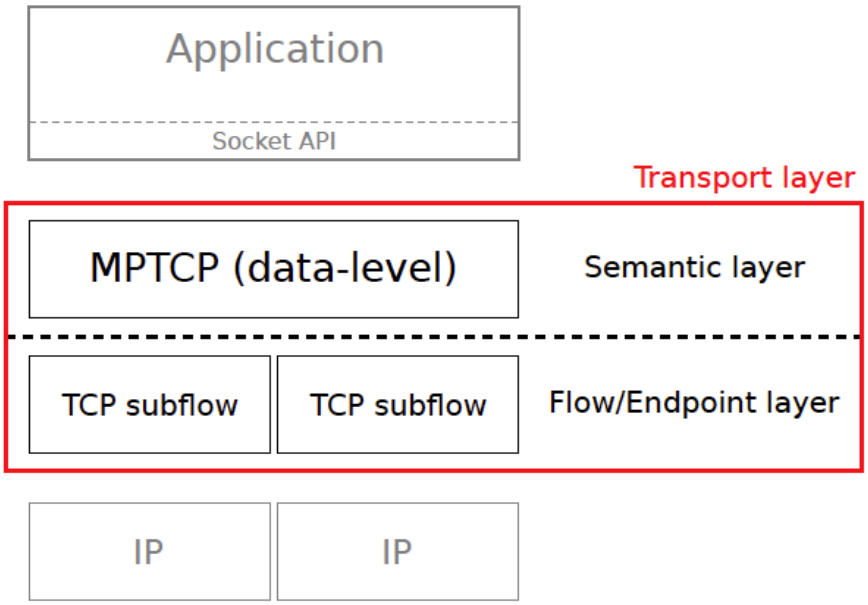


Fig. 1.1 MPTCP protocol stack, adapted from RFC 6128.

With normal TCP, there is only one data flow per connection, between to pairs of IP addresses and ports. With MPTCP, data belonging to a same connection can be transmitted with different source and destination IP addresses and ports. Packets having the same source and destination IP address and ports belong to a same subflow.

At the MPTCP connection level (also called data-level by RFC 6824), a connection can use several subflows and the data is reordered and ready to be delivered to the application socket. Also, as visible on Figure 1.1 (which presents high-level architecture of MPTCP by RFC 6128 [5]) the transport layer is divided in two parts: the application-oriented semantic layer (which ensures the reliable data transmission) and the network-oriented flow and endpoint identification (that focuses on congestion control).

MPTCP is also a protocol that allows switching between all the interfaces of a specified device and only works with TCP applications. When establishing a

connection, and MPTCP-enabled host opens a TCP connection by sending a SYN packet, the destination (also a MPTCP-enabled host) reply with an SYN/ACK response and finally the first host completes the three-way handshake with an ACK packet. If the destination host or any other middlebox on the path does not support MPTCP, the connection falls back to normal TCP. Figure 1.2 shows a successful MPTCP connection establishment.

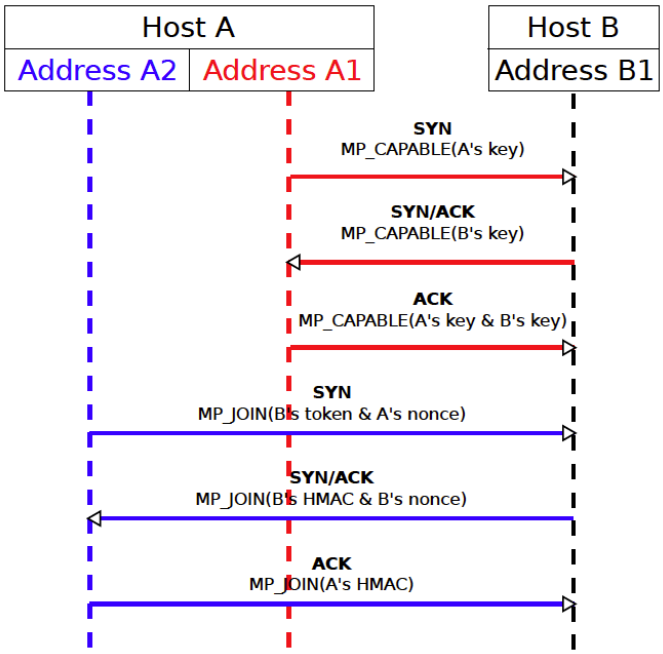


Fig. 1.2 MPTCP initial handshake and subflow opening (RFC 6824).

Finally, it is interesting to look at the expected performance of MPTCP and describe the main characteristics and benefits of this protocol.

- **Reliability:** MPTCP helps to recover from link failures if several interfaces are available on any endpoint. For example, imagine an embedded device that has 3 operative interfaces: Ethernet, Wi-Fi and 3G, where the first one (Ethernet) is the default interface. In the case where this interface is disconnected, the protocol would activate any of the other. And in the same way it would make the same operation if the second one loses coverage or connectivity.

  In this way, Multipath TCP protocol (MPTCP) allows not to lose device connectivity, always being transparent to the user. Links may be added or dropped as the user moves in or out of coverage without disrupting the end-to-end TCP connection.

  Also MPTCP increases the redundancy and can balance a single TCP connection across multiple interfaces.

- **Throughput:** global throughput is expected to be at least as good as the available throughput on the best path. The total throughput that an MPTCP connection can achieve would be lower than the sum of the

throughputs of individual TCP flows in network layer. Anyway, MPTCP increases the throughput, which becomes the sum of all available link-level channels.

- **Latency:** the impact of this characteristic is difficult to estimate. It depends on how the sender splits the data between the different subflows.

1.1.2. Mobile IP

Mobile IP is a protocol designed (RFC 6275 [6]) to allow user mobility between networks maintaining a permanent IP address and without changing it.

The main features and advantages of Mobile IP are:

- Continuous connectivity to the network.
- Maintain a permanent IP when user moves.
- Maintain the connectivity between a mobile device and a static node.
- Reduce the effects of changes in location of the mobile device.

An extension of this protocol is NEMO (Network Mobility Basic Support Protocol). NEMO (RFC 3963 [7]) supports mobility for entire mobile networks that move and attach to different points in the Internet. It allows session continuity for every node in the mobile network as the network moves and can work with all type of applications because mobility is supported at network layer.

In order to maintain the IPv6 addressing for the mobile network, the mobility capabilities that NEMO provides are distributed between the *Mobile Router* (MR) and the *Home Agent* (HA) entities.

An unchangeable IPv6 *Mobile Network Prefix* (MNP) is delegated by the home network to the MR for assigning addresses to the *Mobile Network Nodes* (MNN). Following the NEMO model, upon the reception of a *Router Advertisement* (RA) message from the *Access Router* (AR), the MR is aware of the existence of a new network. In this case, the MR, which already has a fixed IPv6 address within its home network (*Home Address* or HoA), generates a new auto configured IPv6 address within the new visited network (*Care-of address* CoA, immediately notified to HA).

Only the MR and the HA are aware of the network change, since MNNs continue connected with MR using the same address configured by the MNP. Hence, when any device outside the home network communicates with any of the MNNs, it uses the MNP-generated address as destination, and packets follow the route towards the home network. Then HA redirects these IPv6 packets to the current IPv6 CoA of MR, which finally distributes the packets within the mobile network. HA and MR perform an IPv6 into IPv6 encapsulation to create a mobility tunnel.

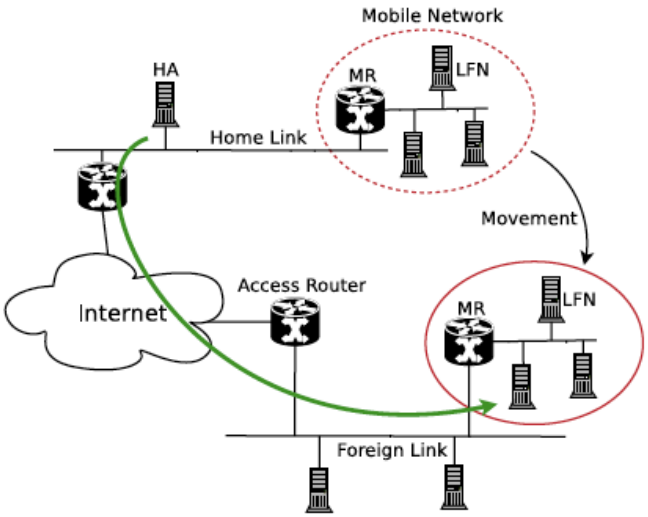


Fig. 1.3 Entities involved in NEMO Basic Support.

In this way, the main objective is to preserve session continuity between *Correspondent Nodes* (CNs) and all MNNs behind the MR, while this one changes its point of attachment (see Fig. 1.3).

NEMO is used as part of ITS (*Intelligence Transport System*) implemented in a vehicle [8]. In this particular case, the smart car gateway will implement a Mobile Router (MR), which will keep the car connected to IP level.

1.2. Open data

Open data is considered as a key point in the smart car gateway. The fact to include external information in the system will give relevant information to the user in order to be used in many driving circumstances or in entertainment applications.

To include this kind of external information in the smart car gateway, we can follow two methods:

- Preload of the external information in the smart car gateway.
- Retrieve of the information using external servers or services.

The first possibility resides on storing the information inside the system of our smart car, but it can be hard to maintain and should be the user who performs these operations. This method may be followed, for example, with some kind of static information that does not change over the time.

The second option is related to get updated information but avoiding maintaining tasks. This can be done thanks to some online available services that will allow us to periodically download the information and store it in the

system. Normally, these services can be accessed through specific API's (*Application Programming Interfaces*).

Depends on the API, the content can be fully or partly downloaded. Some datasets are prepared to get only the updates but others require getting the full information in every query made to the service. Many of them are from this last type and, therefore, we should consider the required time to get the full file in order to decide the optimal type of connection.

For example, Wi-Fi hotpots' file of Barcelona City [9] can only be fully retrieved from its API. The file has a size of 186KB and the expended time to download it from the server is about 685ms, using the home's Wi-Fi just before leaving the parking.

1.2.1. Mobile coverage

In this context and as first step, we will initially consider a dataset from the Catalan Government in order to get updated information about mobility coverage in all the territory [10]. This is a collaborative project where the population shares its own mobile data in order to define a centralized coverage map and indicate information about:

- Mobile coverage level
- Telecommunications provider
- Mobile Technology (2G, 3G or 4G)

The information is provided in CSV format and each file contains:

- Date and hour of the measure
- Position coordinates (Latitude/Longitude in EPSG 3857 format)
- Signal level in ASU format [11]
- Used technology (2G, 3G or 4G)
- Service Operator (Movistar, Vodafone, Orange and Yoigo)

These CSV files are updated every two months and the smart car gateway will be programmed to retrieve the new files thanks to the connection to the web services available by the open data website.

The following images show a map of the mobile coverage in all the Catalonia territory and a map of the 4G's coverage in Barcelona city area, with the information retrieved from the May's file.

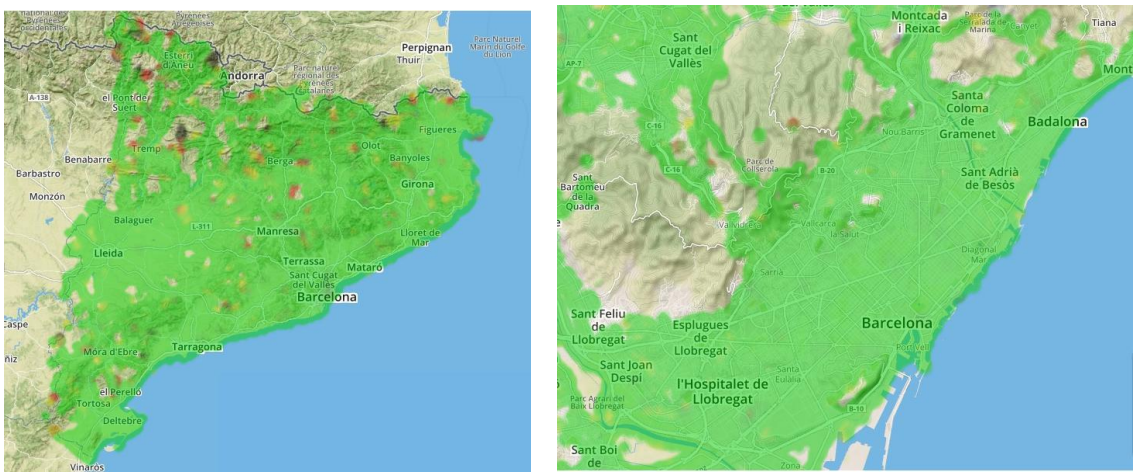


Fig. 1.4 Mobile coverage in Catalonia and 4G's coverage in Barcelona

As explained before, data is collected anonymously from smartphone users and the information is somewhat indicative. Therefore, the more measures taken at one point, more correct interpretation of the signal level will be.

1.2.2. Wi-Fi coverage

Another web service initially to consider and include in our system should be the information of available Wi-Fi access points in the territory. In order to unify the information from the Wi-Fi coverage and the 3G coverage (detailed in the previous section), it was selected the web service "*Wi-Fi hotspots*" [9] offered by the Barcelona City Council. That will allow us to obtain this kind of information also along the Catalan territory but specifically in Barcelona city area. To match both data sources, the chosen area to load initially on the system will be the city of Barcelona.

The information is also available in CSV format (with a new file each month) and the data contained in the files is the following one:

- Hotspot coordinates (Latitude/Longitude in UTM31 ED50 format)
- Address of the hot spot
- Local equipment that gives the service
- Contact of the local equipment

Next figure shows the hotspots location retrieved from the files.

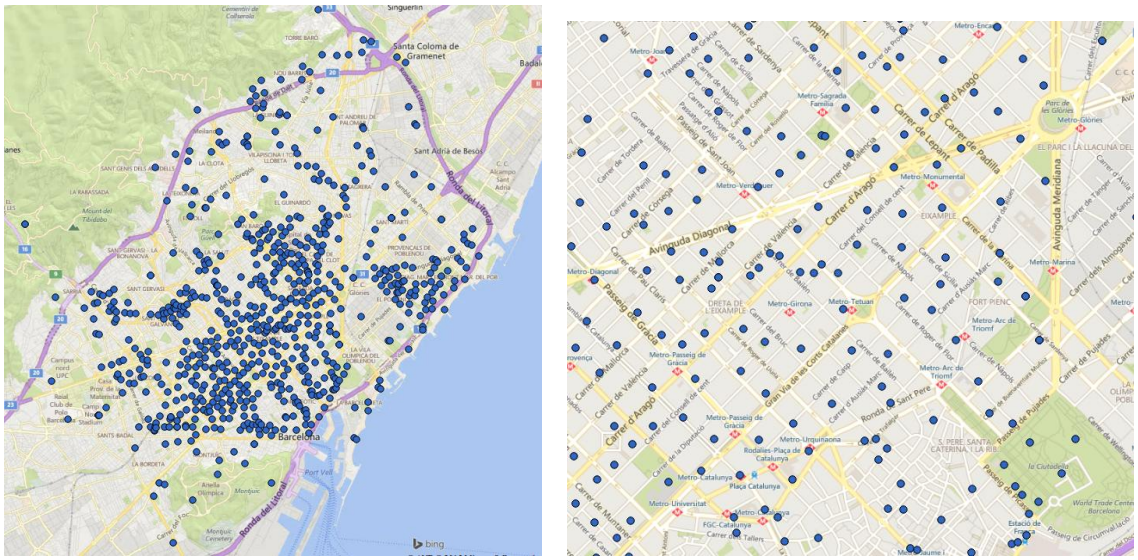


Fig. 1.5 Wi-Fi hotspots in Barcelona city

Once we can get this updated information about mobile coverage (3G/4G) and Wi-Fi hotspots location, we could use this information to define route strategies and decide when to make the handovers between networks while driving.

Based on the results of the experiments presented in the next chapter we can decide the thresholds for when to connect or disconnect from one technology to the other, but always maintaining the connectivity of the car.

1.2.3. Traffic status

Traffic status is also a useful and available information that we can retrieve from external services. This data can help us in order to know the real status of the roads with information about incidents, congestions, roadworks and adverse weather.

We can use this API in order to get an XML generated file and load it into our system, but also we can retrieve the data by the connection to the government WMS server [12] directly from the car.

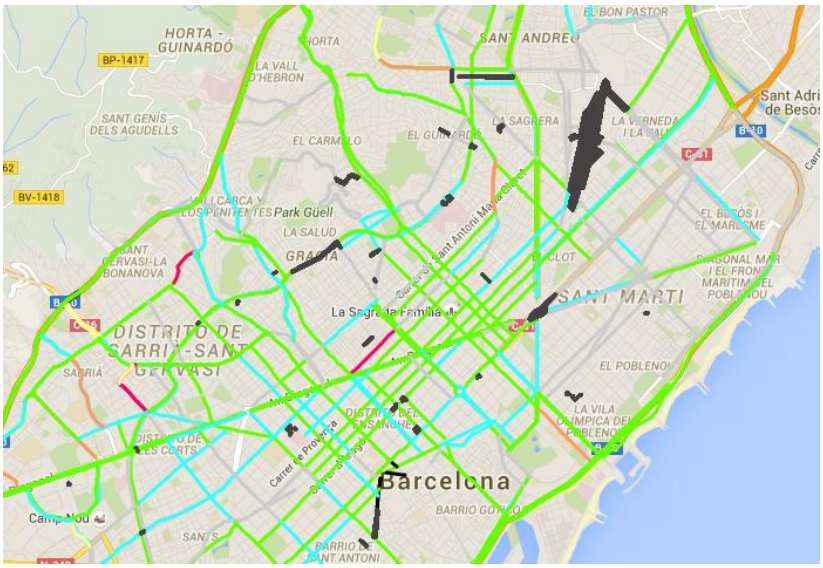


Fig. 1.6 Traffic status and affectations

Some specific information that we can get with this real-time service:

- Traffic incidents by road and section (Km points).
- Roadworks by road and section.
- Time spent to cross roads sections.

With that, we could better approximate the time spent to make a route not only by the fact that we can know all about the instant traffic status, but also because the service can give us the predicted traffic status in about a maximum of 15 minutes.

1.3. Opportunities

1.3.1. Multihoming

Multihoming makes reference to a device that is connected to more than one network. For example, in our case, a smart car gateway could be connected using Wi-Fi or 3G/4G networks. The benefits could be:

- Increase the reliability of an IP network.
- Overcome the mobile wideband coverage problems.

Multihoming has some variants:

- *Single Link, Multiple IP address*
  - When the host has multiple IP addresses but only one physical upstream link.
  - If the single link fails, connectivity is down for all addresses.

- *Multiple Interfaces, Single IP address per interface*

  - When the host has multiple NIC (*Network Interface Controllers*) and each interface has one (or more) IP addresses.
  - If one of the links fails, then its IP address becomes unreachable, but the other IP addresses may still work.
  - Multipath is used to switch network interfaces when one of them fails, in order to not to lose the connectivity (for other interface still working). Also Multipath allows using every available network interface.

- *Multiple Links, Single IP address*
  - Main variant of Multihoming. With the use of a routing protocol (in most cases BGP) the routing element announces this address to its upstream links.
  - If one of the links fails, the protocol notices this on both sides and traffic is not sent over the failing link any more.
  - This method is usually employed to multihome a site and not for single hosts.

- *Multiple Links, Multiple IP address*
  - This method uses a specialized Link Load Balancer (or WAN Load Balancer) appliance between the firewall and the link routers.
  - It allows use of all links at the same time to increase the total available bandwidth and detects link saturation and failures in real time to redirect traffic.
  - Also is used to control routing between the separate address spaces used by each interface.

MPTCP is designed to take profit of Multihoming. Regarding the improved resilience and increased throughput that Multihoming should provide, MPTCP naturally detects if one of the interfaces stops working and allows communications to continue over the other paths without a break. Besides, also MPTCP distributes the traffic over the paths to make more efficient use of the available capacity on the several paths accessed through the different interfaces.

Moreover, a mobile node maintains continuous communications as it moves though areas served by dissimilar access networks. Handover is “smooth” since all the interfaces are used simultaneously, i.e. make-before-break handovers. In fact, better performance will be achieved when the handovers are “smooth”.

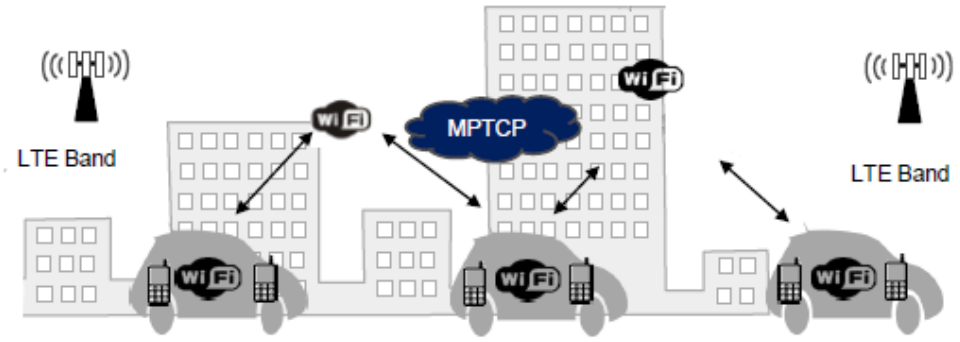


Fig. 1.7 Handovers between Wi-Fi hotspots and LTE network.

In the case of NEMO it has been studied its interaction with Multihoming (RFC 4980 [13]) basing on the proposition of a taxonomy to classify the possible configurations.

A possible Multihoming architecture can be based on including MPTCP and Mobile IP modules in the MN and the CN stacks, with Mobile IP mechanisms placed underneath the MPTCP layer [14]. Its approach separates established session mobility (MPTCP) from initial contact when the mobile is roaming (Mobile IP). Mobile IP is also included to ensure reachability in case of simultaneous movement.

A simple scenario could be the following: the connection is initiated between MN's HoA and the address of the CN, with Mobile IP playing its standard role when the CN initiates the connection and the MN is away from home.

If the MN is already away from home, or as soon as it moves away, the Mobile IP layer of the MN creates a binding between HoA and one of the CoAs available at the visited network. Mobile IP layer also notifies the MPTCP layer of the host's stack so that, as soon as communication is established, MPTCP can add the CoAs as additional addresses for the MPTCP connection, and then distribute the traffic based on the available capacity for each path.

In fact, since the MN has multiple interfaces, it may have multiple CoAs and even multiples HoAs. These can be added to the MPTCP connection and traffic distributed across them appropriately. Also, when the MN moves, MPTCP uses the new CoA and Mobile IP updates the HoA-CoA binding.

1.3.2. Open data integration

Focusing on the driver, a lot of data could be included in the gateway. For example, it can allow setting up some real-time alerts about traffic status or

weather to get the optimal route to destination. Also some additional information can be included in order to reduce costs or centralize vehicle maintenance book in car systems for both the user and the manufacturers.

But in this project, we will only consider the useful external information needed to improve the decisions made by the gateway in a given route. As it will be extended in following chapter 4, the main objective is to define a decision algorithm that allows the gateway to determine the best situation where to make the handover. Open data origins would help the gateway to know more about the environment and use this information to enhance the decision processes.

In this context, the Wi-Fi hotspots location in Barcelona city, the 3G/4G coverage areas and the traffic status are the main used datasets to enrich the decision algorithm.

The first one will give the gateway information about the presence of hotspots in the route in order to decide if we could make the handover or not to a Wi-Fi network with the aim, for example, to reduce 4G data consumption. On the other hand, 3G/4G coverage areas will help us to know if we can maintain the connectivity when Wi-Fi connection is not available. It is assumed that we will have 3G/4G coverage in the studied urban scenario, as we will demonstrate for the case of Barcelona in chapter 4. And finally, the traffic status is a key point in order to allow the gateway to calculate the total time of the route and the time spent to cross hotspots coverage areas.

Wi-Fi hotspots and 3G/4G coverage datasets can be preloaded in the gateway because this information does not change very often. On the other hand, traffic status should be retrieved via API connection to external services (for example, like the provided by Google) because of its dynamism.

CHAPTER 2. SOFTWARE IMPLEMENTATION

2.1. Software requirements

2.1.1. Multipath TCP

There are several ways to install the Multipath TCP Kernel Implementation. In this section, we'll explore them in order to decide which one is the best to perform the tests.

All the related information about Multipath TCP could be found in the official protocol website (http://www.multipath-tcp.org/) and the implementation code is also available (https://github.com/multipath-tcp) if we want to develop new features on the actual stack (not in our case).

The release of protocol's implementation used in the project is the v0.89, but also older versions can be acquired. It's important to mention them and specify their compatibility with the different Linux distributions.

- *MPTCP v0.89:* Linux Kernel v3.14 compatible
- *MPTCP v0.88:* Linux Kernel v3.11 compatible
- *MPTCP v0.87:* Linux Kernel v3.10 compatible
- *MPTCP v0.86:* Linux Kernel v3.5.7 compatible

The implementation of MPTCP can be installed as/under:

- Compiling the source code (available on GitHub, as previously detailed).
- Installing it directly from apt-repository.
- Under the following Linux Distributions:
  - Debian Squeeze and Debian Wheezy (via apt-repo)
  - Ubuntu from v12.10 (via apt-repo)
  - Fedora 19 to 21 (pre-compiled images)
  - CentOS 7 (pre-compiled images)
  - Gentoo (pre-compiled images)
  - ArchLinux (via specific Package)
  - OpenSUSE (apt-repo)

  Important to mention:
    - Apt-repository available for all 64-bit Linux flavours.
    - Apt-repository for 32-bit systems only available in Debian Squeeze or Debian Wheezy.
- OpenWRT (Linux distribution for embedded devices)
- Android Systems:
  - Samsung Galaxy S2 (Android ICS)
  - Google Nexus S
  - Google Nexus 4 (Android from 4.2.1 to 4.3)
  - Google Nexus 5 (Android 4.4)

- User-mode-linux setup
- PlanetLab
- Amazon EC2
- Using Vagrant (a software tool that allows to set up development environments on a virtual machine) to get a guest running with MPTCP.

All the possibilities mentioned above require Kernel's Linux modification to run correctly.

2.1.2. Mobile IP

UMIP is an open source implementation of Mobile IPv6 and NEMO Basic Support for Linux. It supports the following IETF RFC:

- RFC6275 (Mobile IPv6)
- RFC3963 (NEMO)
- RFC3776 and RFC4877 (IPsec and IKEv2)

We can find all the related information about this implementation in its website (http://umip.org/), and also the details about its source code, contributions and installation modes.

It is important to mention that this implementation runs on a mobility-ready kernel such as kernel sources since 2.6.26 and kernel versions from 3.8.2 (in 3.* kernels, mobility is broken up to version 3.8.1).

The implementation of UMIP (Mobile IPv6 & NEMO) can be installed from:

- The source code (compiling it).
- Specific Packages through apt-repository.

CHAPTER 3. TEST BED EVALUATION

3.1. Test bed description

The main objective in this chapter is to evaluate the performance of MPTCP and NEMO running separately and combined. In order to do that, a test bed has been designed and implemented. The test environment has to reproduce similar conditions as in the case of a mobile device interacting with MPTCP-enabled and NEMO-capable network devices.

After some attempts and alternative setups, Ubuntu 14.04.1 with 3.14 kernel is the combination selected in order to support both protocols running at the same time.

Another important fact to mention is that, in our case, this NEMO implementation does not support MCoA (*Multiple Care of Address*). MCoA allows having multiple CoA's in order to avoid the configuration of a new CoA each time the gateway changes from an access point to another. This fact makes that its combination with MPTCP has sense because MPTCP will be in charge to maintain the connectivity in handover situations.

In terms of hardware, the test environment will use two physical laptops with one of them running three virtual machines. The following figure illustrates the setup.

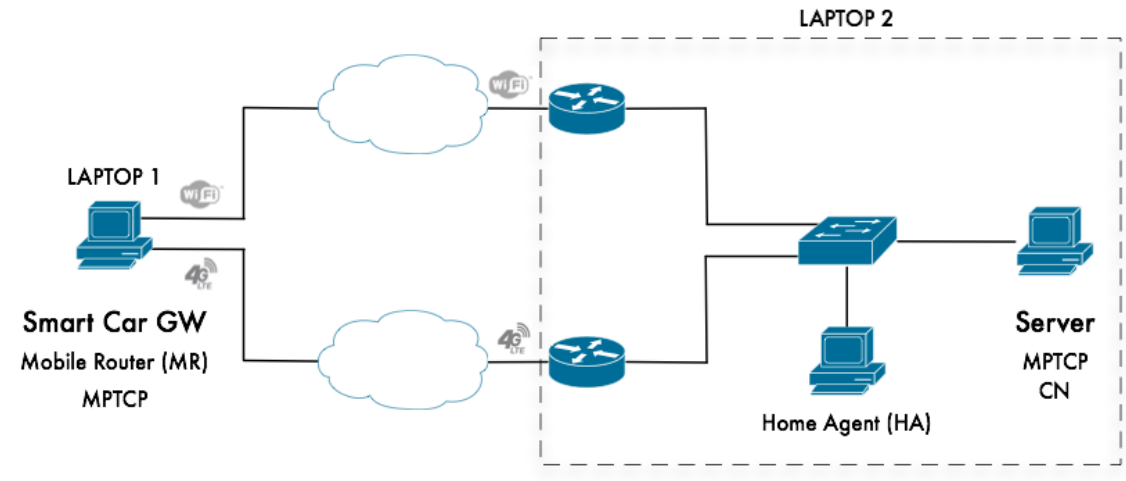


Fig. 3.1 Implemented Test Bed

Hardware is disposed as follows:

- 1 Physical laptop running NEMO as a Home Agent (HA) **[Home Agent]**.
  - With 1 virtual machine running Multipath TCP **[Server]**.
  - With 1 virtual machine as gateway 1, acting also as access point **[GW with Wi-Fi interface]**.

  - With 1 virtual machine as gateway 2, acting also as access point **[GW with 4G interface]**.

- 1 Physical laptop running Multipath TCP and NEMO, acting as a Mobile Router (MR) **[Smart car gateway]**.

As seen in previous section 1.1.3, this disposal corresponds to the case of multiple interfaces with a single address per interface.

The virtual machine acting as a gateway 2 simulates its 4G interface using bandwidth limitation software TC [15] on its own Ethernet interface. With the help of that tool, bandwidth was limited to 40 Mbps (average obtained bandwidth after field tests of 4G technology in an outdoor experiment). Wi-Fi interface works in 802.11g mode with a maximum speed of 54 Mbps. However, we want to reproduce a public Wi-Fi access, the throughput is limited to 256 Kbps [16].

Table 3.1. Bandwidth limitations on each interface (Wi-Fi & 4G)

Interface	Bandwidth
Wi-Fi	256 Kbps
4G	40 Mbps

3.1.1. Test bed settings

We will use the test bed to understand the impact of the delay in the exchange of signalling messages on both protocols.

To conveniently prepare the scenario in order to achieve this goal, we should first look for typical delay values that then, we will apply to perform the study. In order to do that, we will start by considering the signalling messages of registration mechanisms in both protocols.

Figure 3.2 shows NEMO signalling messages in a transmission defining the following times.

- **Tra:** time until the reception of the Router Advertisement.

- **Tcoa:** configuration time of CoA address, from the reception of the RA message to the send of the Binding Update (BU) to the HA in order to notify the new IP. Duplicate Address Detection mechanism is not activated.

- **Tmipv6:** time spent from the sending of the Binding Update (BU) by the MR to the reception of the Binding Acknowledgement (BA) message from the HA. In this time the HA configures the tunnel and the needed routes to reach the MR.

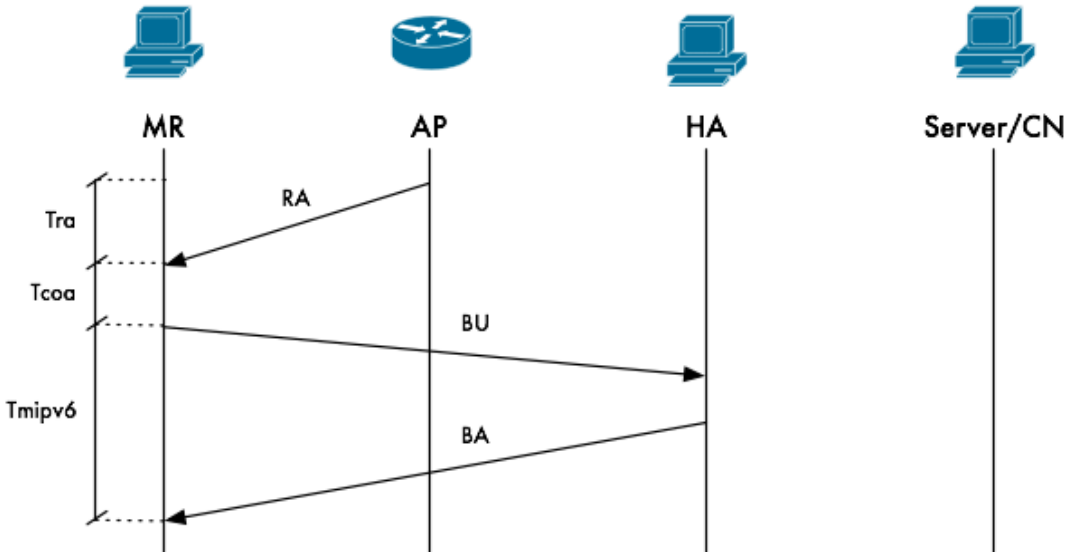


Fig. 3.2 NEMO Signalling messages

On the other hand, figure 3.3 shows MPTCP signalling messages. In this case, there is only one time to consider.

- **Tmptcp:** time spent to establish a MPTCP subflow using a three-way handshake. As shown in the figure it is augmented with MPTCP specific option MP_CAPABLE inserted in the SYN and SYN+ACK, with the aim that both endpoints know that the other end supports MPTCP. MP_CAPABLE option in the third message of the three-way handshake is inserted to allow the server to postpone the state creation until the end of the handshake. Hence, MPTCP runs exactly like TCP. On the other hand we should take into account that for any additional MPTCP subflow, it will be done by another regular TCP three-hay handshake.

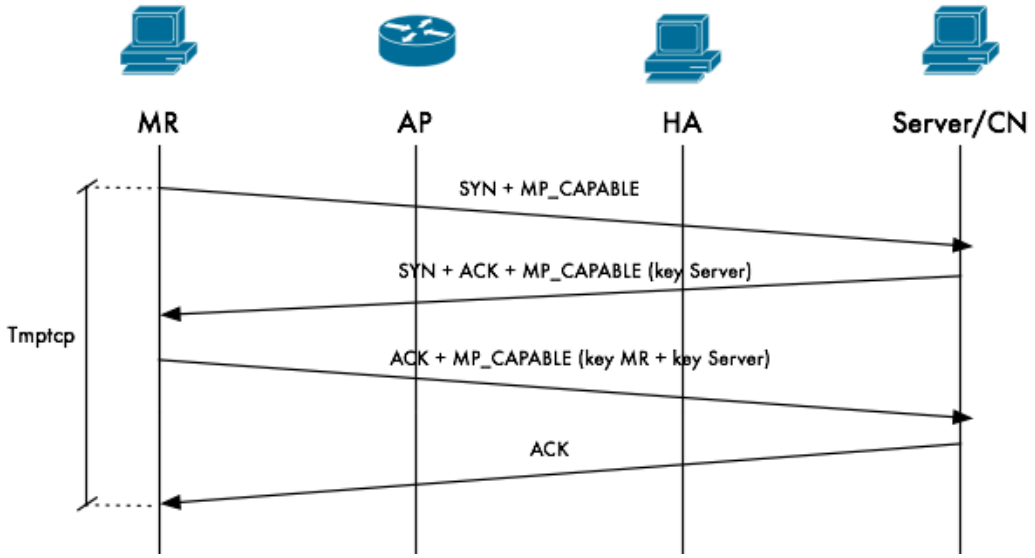


Fig. 3.3 MPTCP Signalling messages

The fact that this implementation of NEMO doesn't support route optimization implies that all messages will always pass through HA. Having this in mind, we can identify three different types of RTT (*Round-Trip delay Time*) between all the devices in our scenario with NEMO and MTCP activated.

- RTT MR-AP: between MR and AP.
- RTT MR-HA: between MR and HA.
- RTT MR-Server/CN: between MR and Server/CN

That can be decomposed as follows:

- RTT MR-HA = RTT MR-AP + RTT AP-HA
- RTT MR-Server/CN = RTT MR-AP + RTT AP-Server/CN

In order to simplify it, we will consider that RTT MR-AP is a constant value associated to the access technologies on each interface. In our case, we have a Wi-Fi interface and a 4G interface. Then, our focus will be in the impact of the delays of the fixed part of the network.

To assign realistic values of RTT MR-AP in both interfaces, a set of measures were done in the test bed with the help of ping [17] command. The results obtained for each interface are shown below:

- RTT MR-AP (*Wi-Fi interface*): 2,5 ms
- RTT MR-AP (*4G interface*): 1,4 ms

In the other hand, RTT AP-HA and RTT AP-Server/CN are the associated values to the fixed network. We can approach these values by knowing the total RTT in the scenario with values RTT MR-HA and RTT MR-Server/CN, which can be estimated by another set of measures.

To do that, some field tests were done in a real scenario to local, regional, national and international sites in order to define specific values for RTT MR-HA and RTT MR-Server/CN. With the help of Fing application [18] and its utility to ping external hosts, a set of 5 measures (each one composed by an average of 50 ping executions) for each site has been done with the following average results.

Table 3.2. RTT obtained values in the field tests

Host Type	Host	RTT (3G)	RTT (4G)	RTT (Wi-Fi)
Local site	www.barcelona.cat	60 ms	87 ms	33 ms
Regional site	www.lacaixa.es	392 ms	91 ms	37 ms
National site	www.google.com	438 ms	108 ms	53 ms
International site	www.helsinki.fi	455 ms	141 ms	155 ms

And with these results, we can obtain an approximated value of the RTT in the fixed network for each access technology.

Table 3.3. Fixed network RTT for 4G interface considering previous results

Host Type	Total RTT (4G)	RTT MR-AP (4G)	RTT fixed network (4G)
Local site	87 ms	1,4 ms	85,6 ms
Regional site	91 ms		89,6 ms
National site	108 ms		106,6 ms
International site	141 ms		139,6 ms

Table 3.4. Fixed network RTT for Wi-Fi interface considering previous results

Host Type	Total RTT (Wi-Fi)	RTT MR-AP (Wi-Fi)	RTT fixed network (Wi-Fi)
Local site	33 ms	2,5 ms	30,5 ms
Regional site	37 ms		34,5 ms
National site	53 ms		50,5 ms
International site	155 ms		152,5 ms

The previous values are real but related to particular websites. For this reason we will round them to get representative values in order to observe a gradual evolution of the delay impact. Therefore, we will choose the values of 30ms, 60ms, 90ms, 120ms and 150ms in order to study the impact of the RTT for the fixed network (RTT $_{AP\text{-}HA}$ and RTT $_{AP\text{-}Server/CN}$) in our scenario. These values will be configured in the test bed with the help of TC/netem [19] command.

3.2. Protocol Performance

We should try to reproduce a real situation in order to make the measurements. For that reason, the smart car gateway (mobile node, acting as MR) will try to download a file from the MPTCP capable server (acting as CN) to emulate the traffic, as for example, in the case where the car wants to update its gateway's firmware to the newest version from the manufacturer's server.

There are some tests that will imply a link failure. The procedure to simulate this failure will be the activation/deactivation of the interfaces with the help of *ifconfig* [20] command.

3.2.1. NEMO Performance

The main indicator to analyse in NEMO performance is the total time that the mobile router (MR) spends in the handover from an access point (i.e. gateway 2 with 4G interface) to another one (i.e. gateway 1 with Wi-Fi interface). As

explained in chapter 4, this time is also important in order to define optimal strategies for handovers when, for example, crossing a road with a specific speed and covered with several Wi-Fi access points.

Handover time is measured while traffic is generated from the Server to the MR. In our case, when the smart car gateway tries to download the firmware file mentioned before. Also *Wireshark* was used in order to obtain all the traffic traces in the MR and in the Server with the aim to analyse the packets and measure the total time and the individual ones.

As mentioned before, this NEMO implementation does not support Route Optimization. This fact implies that the transmission between Server and MR always passes through the HA.

As seen in previous section (see Figure 3.2), the signalling messages associated to NEMO are Tra, Tcoa and Tmipv6. The duration of the handover for MIPv6 is the addition of these three times.

Figure presented below shows the results acquired in the tests without delay addition. The following bar chart represents the minimum, maximum and average value of the accumulated individual times for NEMO (Tra, Tcoa and Tmipv6) when there is coverage of the two networks and one of them is forced down for the handover.

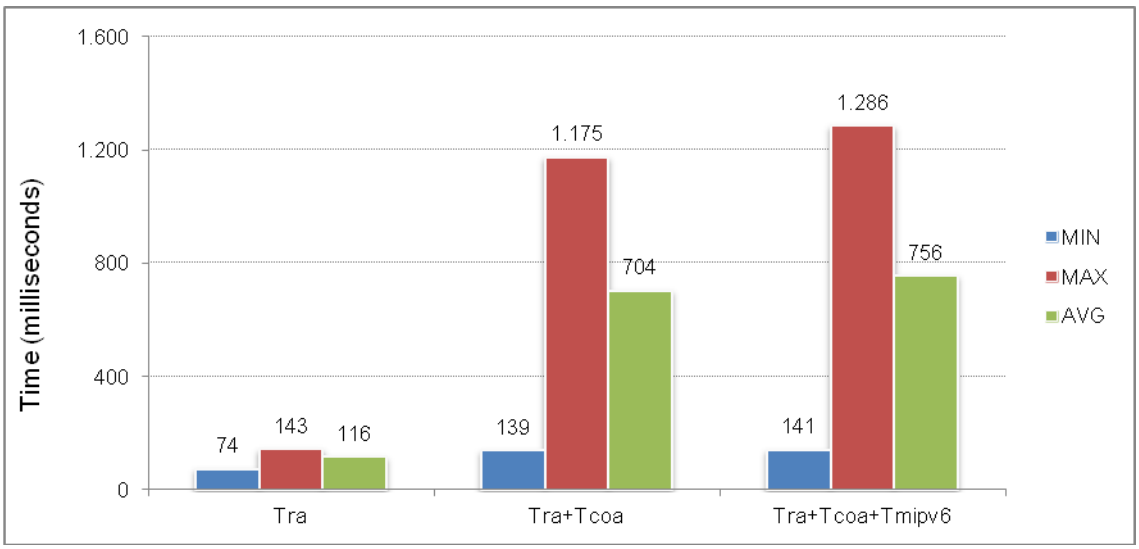


Fig. 3.4 Obtained time results when running NEMO without delay addition.

All these results were obtained after 30 samples and we can observe that the total time is highly variable. Minimum time is 141 milliseconds and the maximum one is 1,286 seconds.

This variability is due to the CoA's configuration time, ranging from about 30 milliseconds to about 1 second, becoming the time that affects at most the total handover time. It is measured since the MR receives an RA (RA's are send every 3 ms) until it sends the BU to the HA.

The following figure shows throughput measurements without delay addition for NEMO in each possible mode regarding to active interfaces: 4G only or Wi-Fi only.

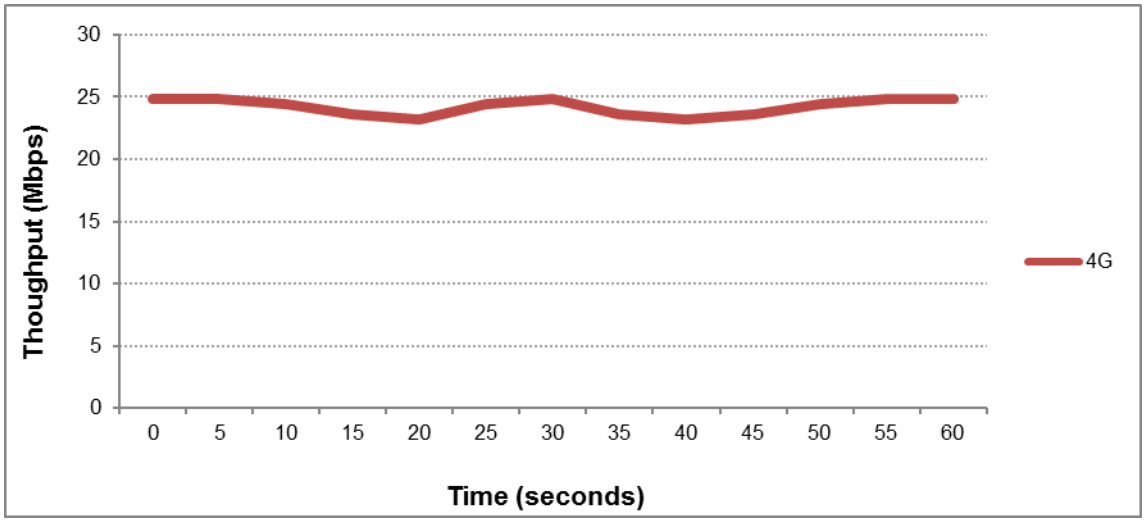


Fig. 3.5 Obtained throughput results with NEMO in 4G interface.

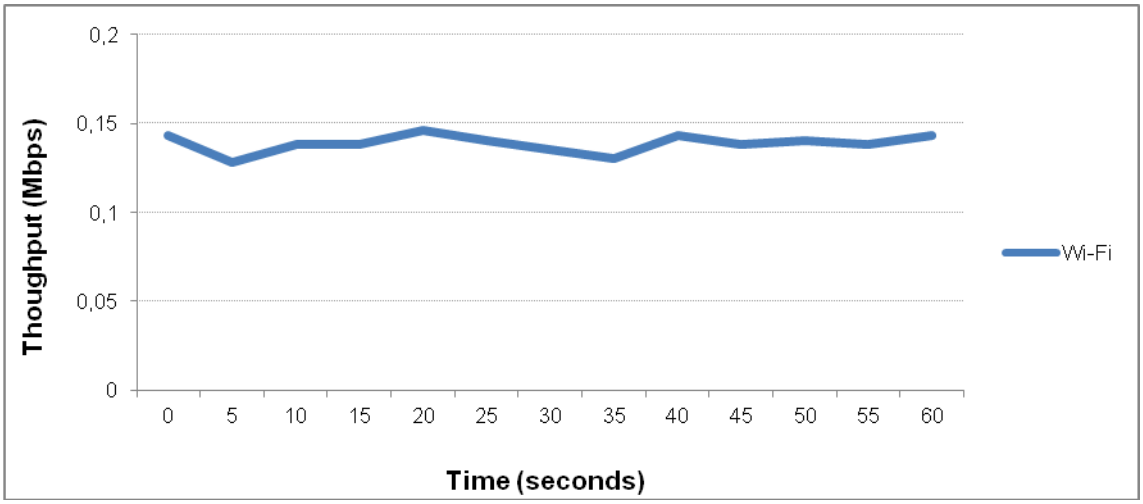


Fig. 3.6 Obtained throughput results with NEMO in Wi-Fi interface.

Analysing the results, we obtain that the utilization of theoretical maximum throughput is about 61% in the case of 4G technology and 54% in the case of public Wi-Fi.

Note that MIPv6 suffers an increase in packet size due its larger IP header (about 20 bytes more than a standard TCP/IP packet size). This could be a key point when comparing the behaviour of NEMO respect to MPTCP or a non-mobility scenario.

The next step is to check the effect of the added delays. As explained in previous section 3.1, delays from 30ms to 150ms in steps of 30ms are used in order to study several realistic cases. The following chart shows the obtained results. Time measures are taken considering 30 samples for each case.

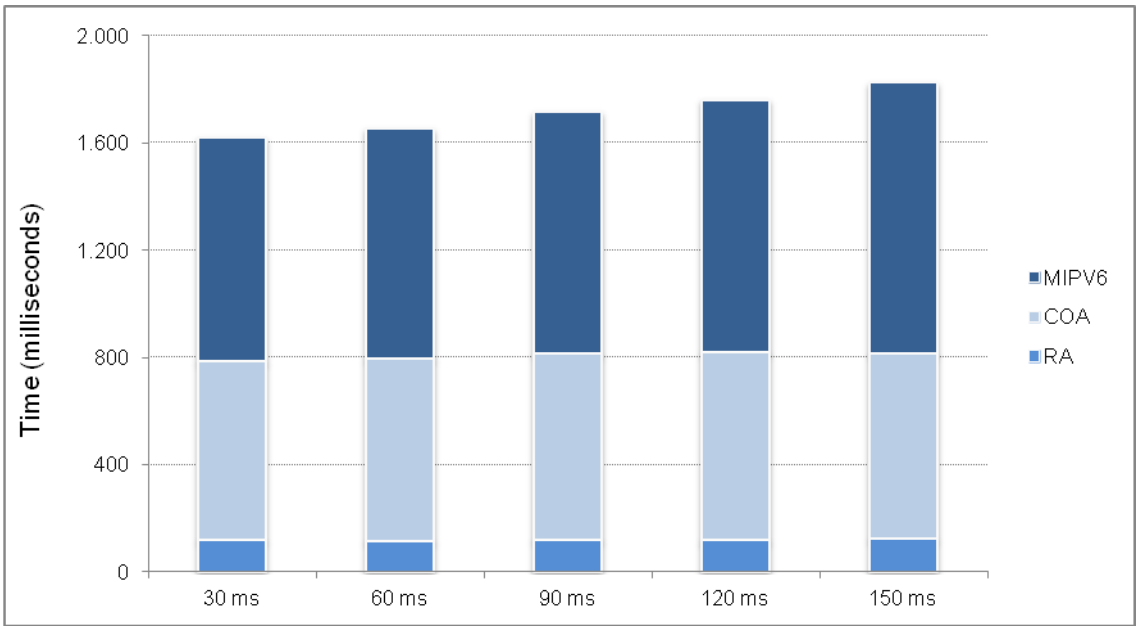


Fig. 3.7 Obtained time results considering delays from the MR with NEMO.

Each bar represents the time for each individual time: Tra, Tcoa and Tmipv6. As we can observe, the total amount of time increases at higher delay values produced by the increase of Tmipv6 component. This increase also corresponds to the value of the added RTT. The other components, associated with the reception of RA (Tra) and the configuration of the CoA (Tcoa), remain stable in most of the cases.

In terms of throughput, a set of measures has been made focusing on the moment when a handover is produced from the 4G interface to the Wi-Fi interface. The following figures show the results for each case (Figure 3.9 shows the detail when 4G interface is disconnected and only Wi-Fi is active).

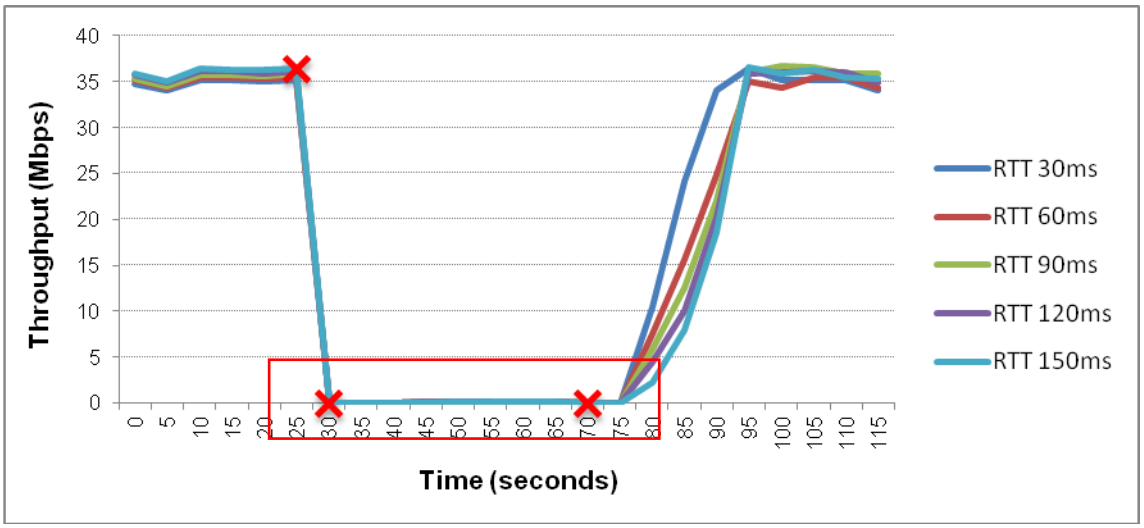


Fig. 3.8 Throughput measures with added delays in a handover situation.

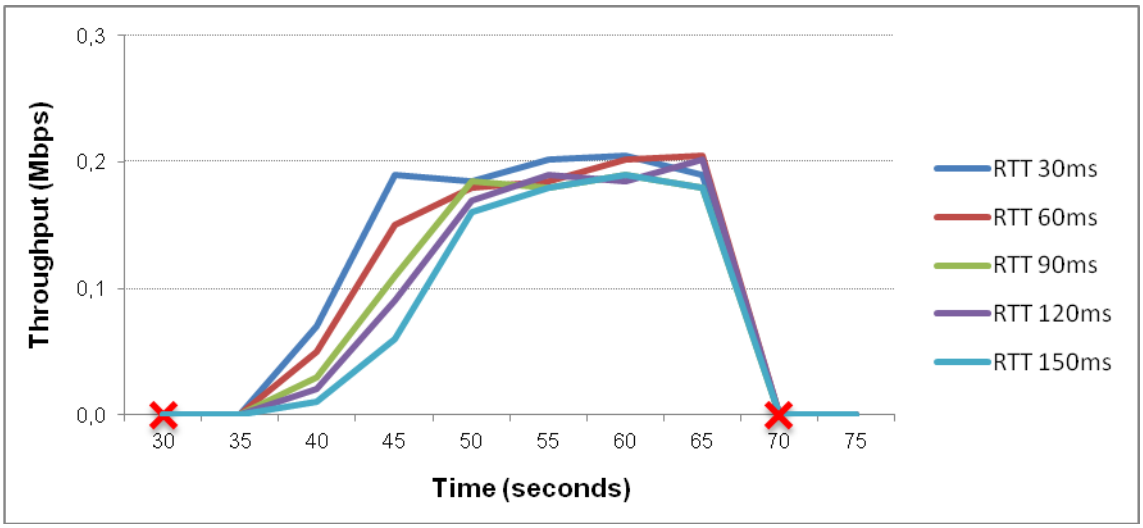


Fig. 3.9 Zoom when 4G interface is disconnected and Wi-Fi interface is active.

As we can see, when making the handover from 4G to Wi-Fi, we have a portion of time (in a range between 3-6 seconds) without connectivity. This behaviour can be explained by the fact that a new CoA needs to be configured in the new access point.

A possible solution that could reduce this effect would be the usage of MCoA functionality in NEMO. Having multiple COA's will allow us to not having to configure the new address when changing the access point. It can be achieved a *make-before-break* behaviour but only in situations where we can predict a handover.

However, as explained in section 3.1, our implementation of NEMO does not support this feature and therefore we could not test it.

Now we could assume that the handover time can fluctuate between 141 milliseconds and 1,83 seconds, considering the total time for the handover when configuring the higher added delay of 150ms (Figure 3.7). This could affect us the way to determine the handovers on the route followed by the car, given that in some situations the speed that the car has and therefore the time it takes to cross the coverage area of a given AP, may be less than the greater of the experimental values in completing the total handover from one network to another. In this kind of situations and also considering the portion of time that we can lose the connectivity during the handover, the best option would be to not change the network even if we can pass from a 4G coverage to a Wi-Fi coverage (for example, to reduce 4G data consumption). This will be extended in chapter 4 with a real example.

3.2.2. MPTCP Performance

MPTCP tests were oriented in order to know how the protocol is able to add the different IP addresses of the active interfaces in our smart car gateway and use

them to send appropriated subflows when the MR moves from one network to another.

Regarding on the active interfaces, three situations can be given on the effect of MR's mobility.

- **Wi-Fi only:** when the mobile node is connected to the Wi-Fi path and disconnected from 4G one.
- **4G only:** when the mobile node is connected to the 4G path and disconnected from Wi-Fi one.
- **Both Wi-Fi and 4G:** when the mobile node is connected to both paths.

The last case can occur when switching from a network to another (*make before break* MPTCP mechanism) or when a better utilization of the links can be reach in situations where both type of connections can be available. In this last case, we will get a better throughput in the data transmission as explained in previous sections.

The following chart shows the obtained throughput results in all the possible situations with the help of *iperf* [21]. Measures were taken in an interval of one minute.

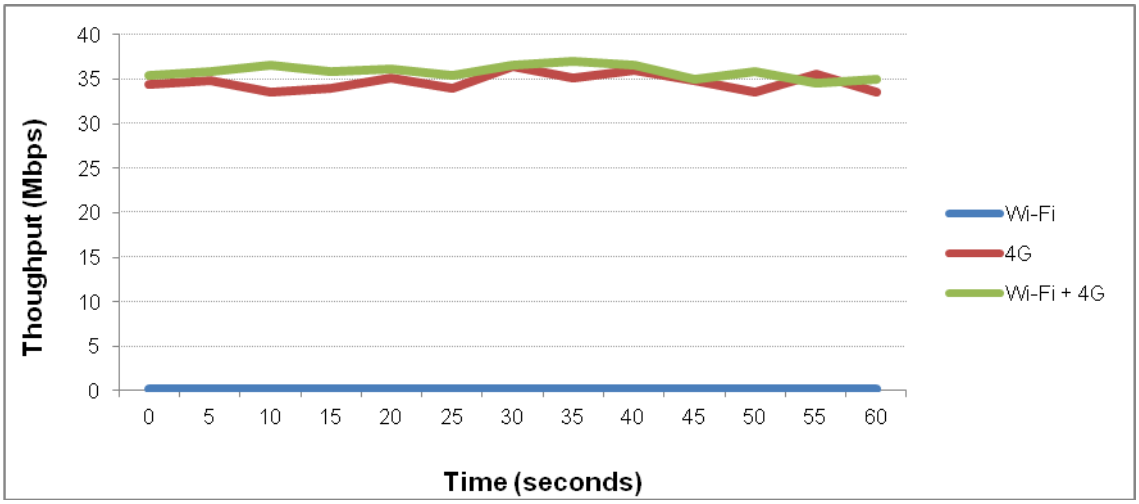


Fig. 3.10 Obtained throughput results in each mode with MPTCP.

Analysing each technology separately, we obtain a utilization of theoretical maximum throughput of 89% in the case of 4G and 71% in the case of public Wi-Fi. The following figure shows in detail the results obtained in the case of public Wi-Fi.

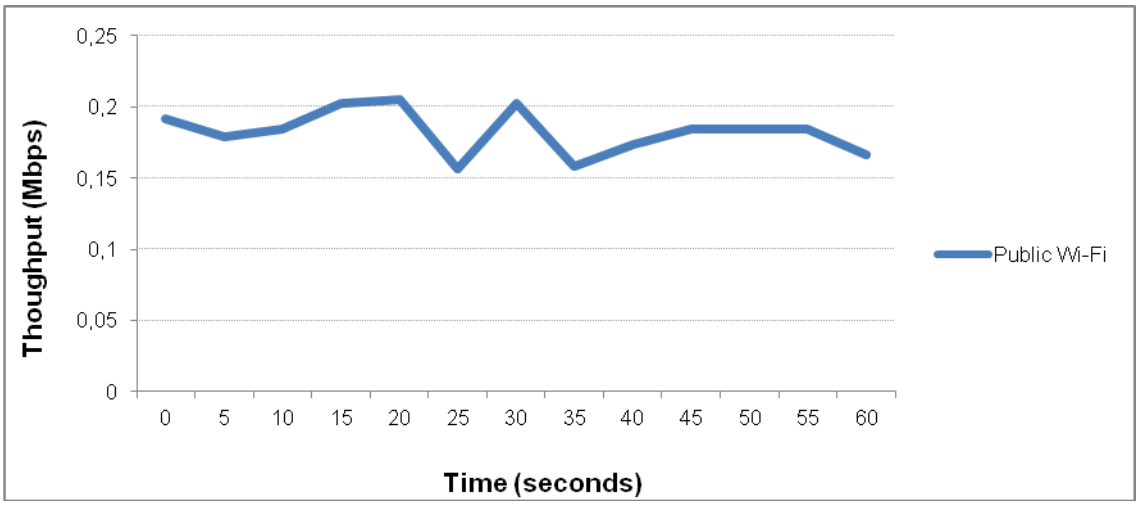


Fig. 3.11 Obtained and detailed throughput results of public Wi-Fi.

But as we can see in the figure 3.10, using MPTCP considerably increases the throughput. However, it seems that MPTCP does not manage to reach full throughput utilization on both interfaces in Wi-Fi + 4G mode.

If it were the case, the throughput obtained would have been close to the sum of the available throughputs of the two interfaces, but the results on figure 3.10 shows that is not the case. The available throughput with concurrent multipath is about 87% (average of 35,1 Mbps) of the sum of the average measured single-path throughput (40,26 Mbps: 40 Mbps of 4G + 256 Kbps of public Wi-Fi).

There is a possible explanation for this apparent underutilization and is related to the coupled congestion control algorithm. It may explain why MPTCP cannot get as much throughput as two separate TCP connections. The coupled congestion control only ensures that MPTCP gets as much throughput as TCP would get on the best path. It is possible to get more throughput but it is not guaranteed and it takes some time. The congestion window increase of MPTCP subflows is slower than normal TCP subflows, and MPTCP takes more time to reach the maximum throughput than two individual TCP flows [22].

This behaviour is necessary to ensure fairness at shared bottlenecks but it may explain bandwidth underutilization when network characteristics are frequently changing for mobile experiments. However, it means that MPTCP is not suited for short transfers because the connection will not have time to reach a high throughput and the different interfaces will be activated and consume energy for little or no benefit.

On the other hand, regarding to the effect of the added delays introduced in previous section 3.1, the following chart shows the obtained results after measures on each different case: 30ms, 60ms, 90ms, 120ms and 150ms. These delays are configured in the gateway when trying to download the firmware file.

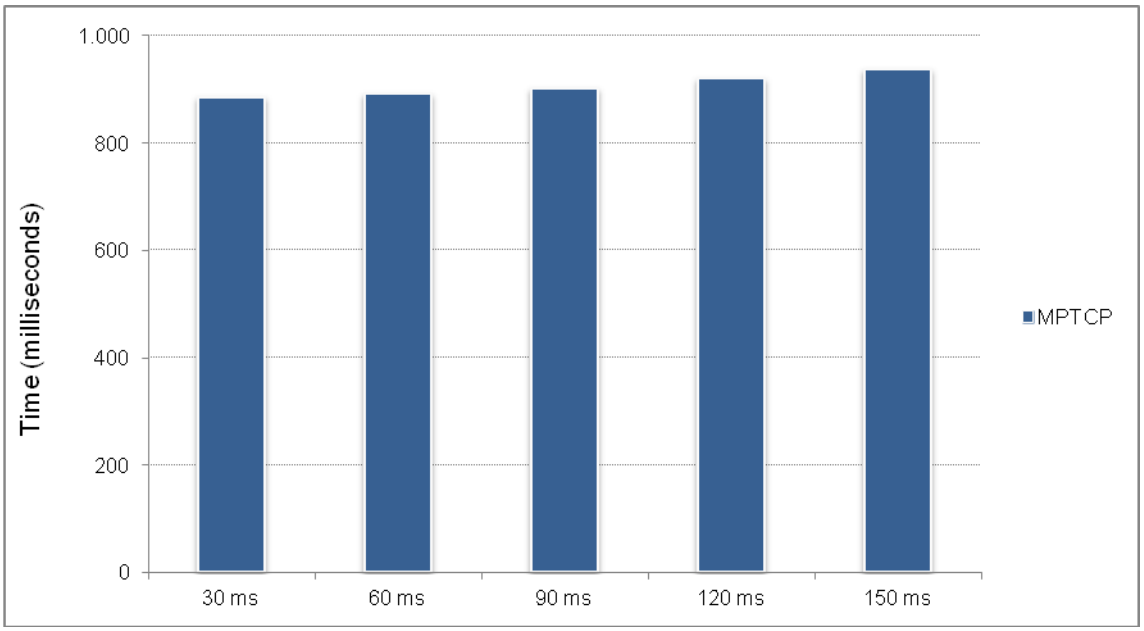


Fig. 3.12 Obtained time results considering delays from the GW with MPTCP.

Each bar represents the time of the three-way handshakes associated with the two MPTCP subflows (Tmptcp, as shown in Figure 3.3) and we can see the variability due to the added delays.

As seen in previous figure 3.3, MPTCP has a message exchange of 2 RTT's instead of 1 RTT in the case of NEMO (see Figure 3.2) and, for that, the impact of the delays is more significant.

In order to analyze the effect of the added delays in terms of throughput, it has been done a set of tests when making a handover from 4G connection to Wi-Fi connection, like in previous section 3.2.1.

The handover is manually made as follows: starting from a 4G connection, it is forced the activation of Wi-Fi interface and finally it is also forced the break of the 4G link. This behaviour allows us to take benefit of the MPTCP capability to work with two different interfaces at the same time and fits in a real situation where in the city the 4G connection remains constant and the Wi-Fi connection appears intermittently (*make-before-break* mechanism).

Following figures show the results (Figure 3.14 shows the detail when 4G interface is disconnected and only Wi-Fi is active).

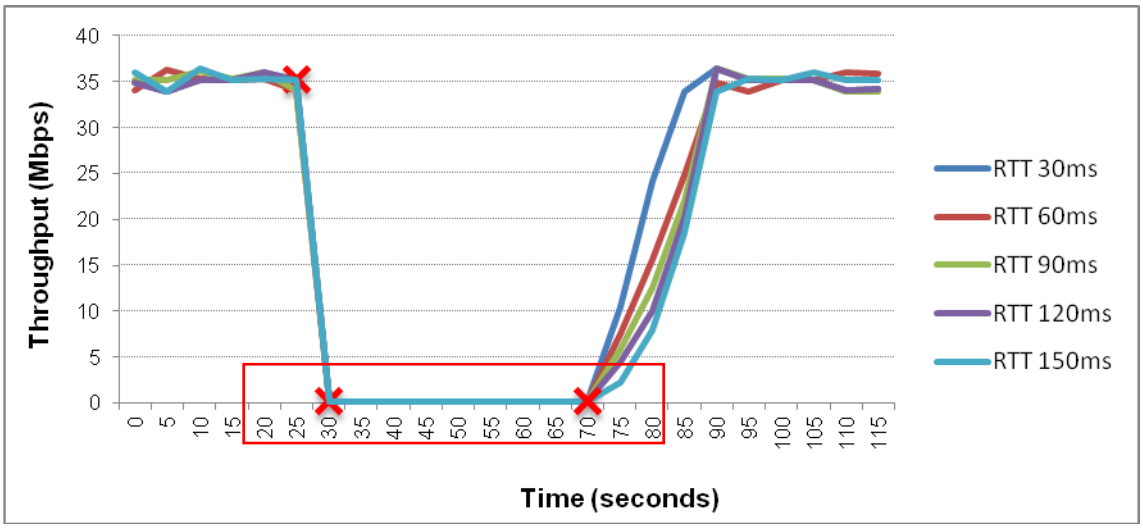


Fig. 3.13 Throughput measures with added delays in a handover situation.

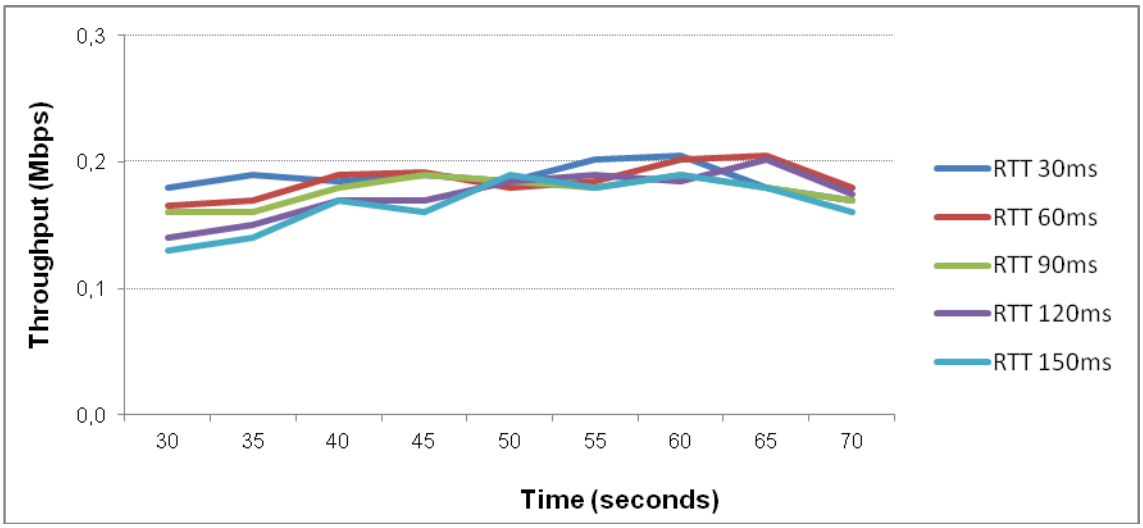


Fig. 3.14 Zoom when 4G interface is disconnected and Wi-Fi interface is active.

In this case we remain connected without any break in the connection. We can see that, when RTT values increase, it's more difficult to take the second connection. Probably this can be caused by the fact that for higher RTT, acknowledgements spend more time to come back.

Unlike in NEMO, we can see that for MPTCP the Wi-Fi connection is better utilized.

Figure 3.15 shows the behaviour when we simulate a mobile scenario in which the MR passes from Wi-Fi network to 4G network. 4G coverage is available for durations of 20 and 25 seconds in two different steps along the test.

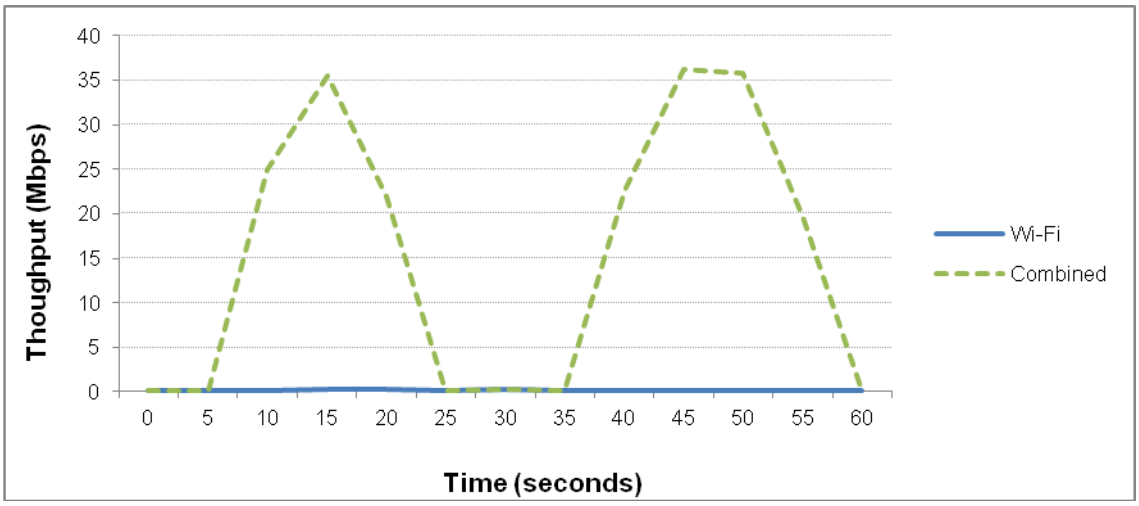


Fig. 3.15 Throughput with MPTCP handover

The 4G path is able to provide a short boost in overall download throughput of a TCP connection initiated over public Wi-Fi. We observe that MPTCP connection is able to remain active and perform handover [23]. This fact will be extended at chapter 4 when analysing the behaviour of the scenario in a mobility environment.

Regarding at the moment when 4G interface is deactivated, we could see that MPTCP recovers quickly to the Wi-Fi connection. MPTCP need to perform a three-way handshake and then continue sending data into the Wi-Fi subflow. In the experiments we obtain that MPTCP reaches the average download speed of Wi-Fi network after 100ms, almost immediately.

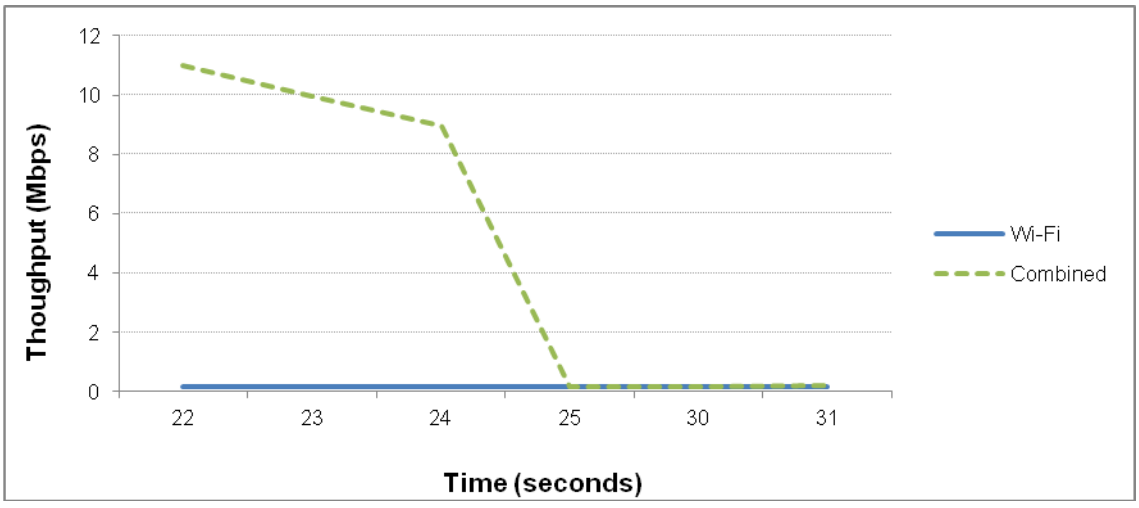


Fig. 3.16 Wi-Fi connection reestablishment.

The last test with MPTCP is intended to reproduce cases when losing Wi-Fi or 4G coverage. When the smart car is receiving the file by its two available interfaces (Wi-Fi and 4G), a failure is manually caused in one of the links. Thanks to MPTCP capabilities, the file transmission was still active due to that

the protocol is able to redirect the traffic initially divided into two subflows to the active interface in that time.

The following charts show all the operations done in the test bed in order to check this behaviour and the obtained results in terms of bandwidth utilization when each interface was connected and disconnected. Measures are taken considering and average RTT of 90ms (average between RTTs used in previous tests). Figure 3.18 shows the time stage when only Wi-Fi interface is active.

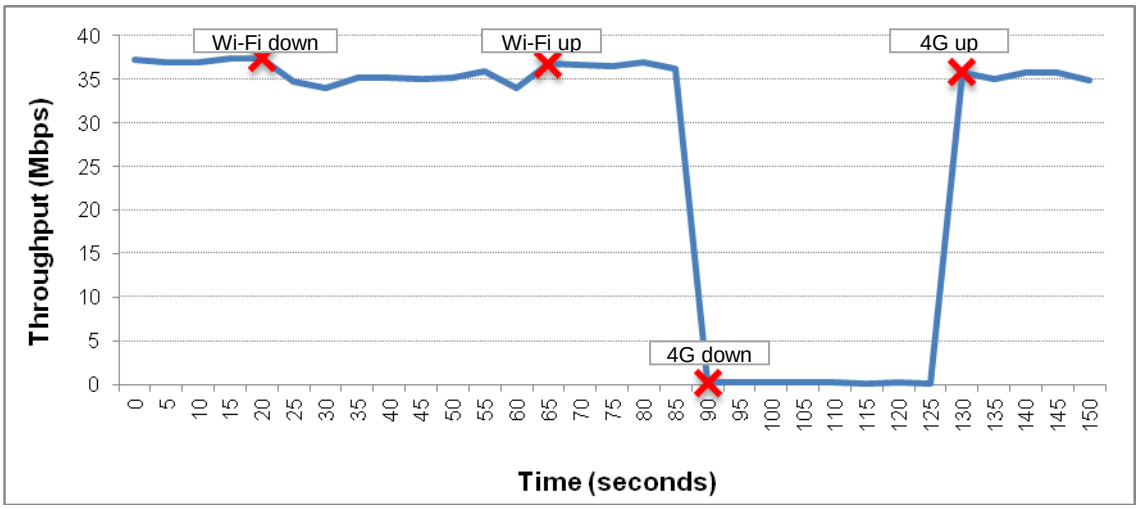


Fig. 3.17 Throughput measures while connecting/disconnecting interfaces

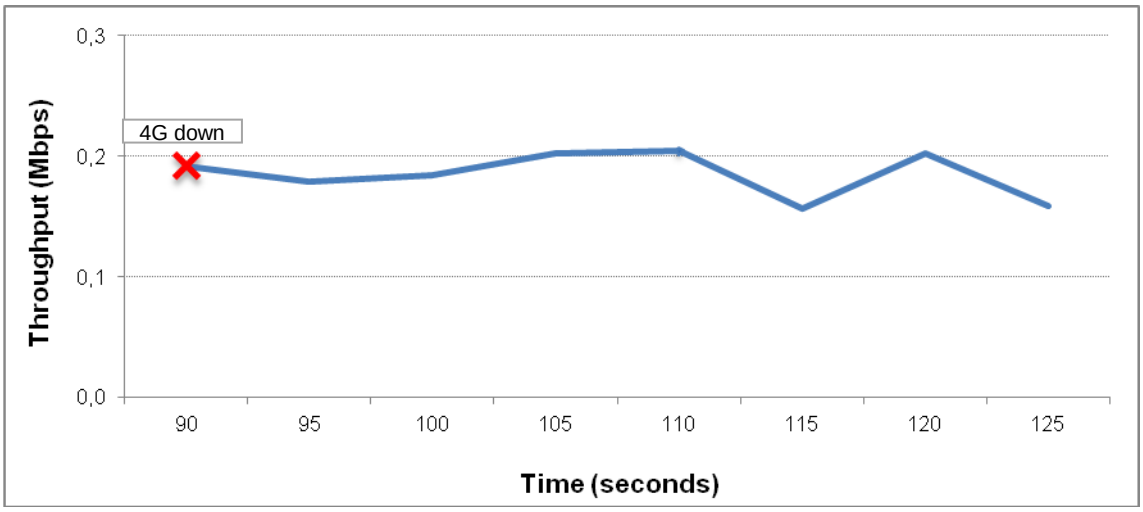


Fig. 3.18 Zoom in time stage when only Wi-Fi interface is active.

As we can see, the file transmission is constant over the time even when any interface is disconnected and the throughput values are similar to the obtained in previous sections. Once both interfaces are active, the throughput is more or less the 87% of the sum of the average single-path throughput. But when any interface is disconnected, file transmission is still active and the throughput is adjusted to the available interface.

When the 4G interface is disconnected, the throughput is reduced considerably to approximately a 1% of the total available throughput when both interfaces are receiving traffic. Anyway, the transmission remains constant without stopping the packet flow. As we can see in the following sections, this fact should reduce data consumption of 4G but decreasing the throughput considerably.

The reasons that cause the link failure are an important fact to take into account regarding on the impact of the added delays in the fixed network. In this particular case, the link failure is caused manually and therefore, MPTCP will work in order to re-establish the connection when the broken link is manually connected again. MPTCP component Tmptcp is affected by the added delays because of the message exchange of 2 RTT's, as observed in the previous section (Figure 3.12). When the broken link is reconnected, Tmptcp will be based on the three-way handshake time that the affected subflow has to make in order to re-join the transmission.

3.2.3. Combination of NEMO and MPTCP

This section presents the throughput measures in a scenario with NEMO and MPTCP running at the same time and also considering the addition of delays in the fixed network.

In this case, we should take into account that the RTT to the Home Agent it is not necessary the same that the RTT to the MPTCP Server. Thus, for example, in terms of RTT it is possible to have a nearby HA and a far MPTCP Server.

In order to approximate this possible situation, the tests will take into account the two extreme cases: when we can have a nearby HA (far MPTCP Server) or a far HA (nearby MPTCP Server). For the first case we will assume a RTT of 30ms to the HA and for the second one a RTT of 150ms. These two situations could help us to understand all the other possible scenarios.

For each case, a set of throughput measures are made regarding to all the possible values of RTT: 30ms, 60ms, 90ms, 120ms and 150ms.

- Nearby HA

As explained before, this scenario is approximated by configuring a RTT to the HA of 30ms, assuming that this could allow us to analyse the behaviour when having a nearby HA and a more or less far Server MPTCP. Following figure shows the results for each value of added delays in this situation.

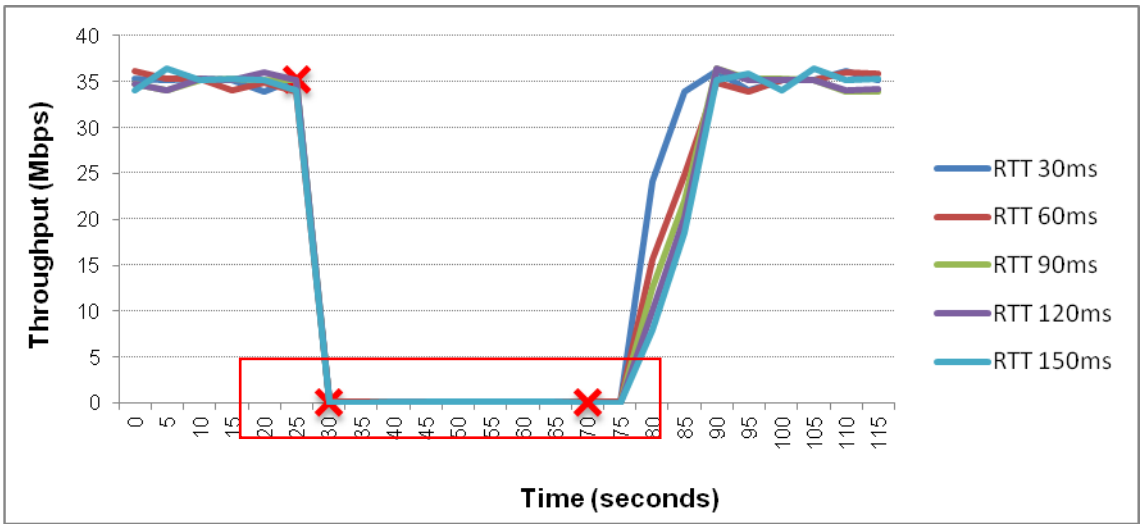


Fig. 3.19 Throughput measures with added delays in a handover situation.

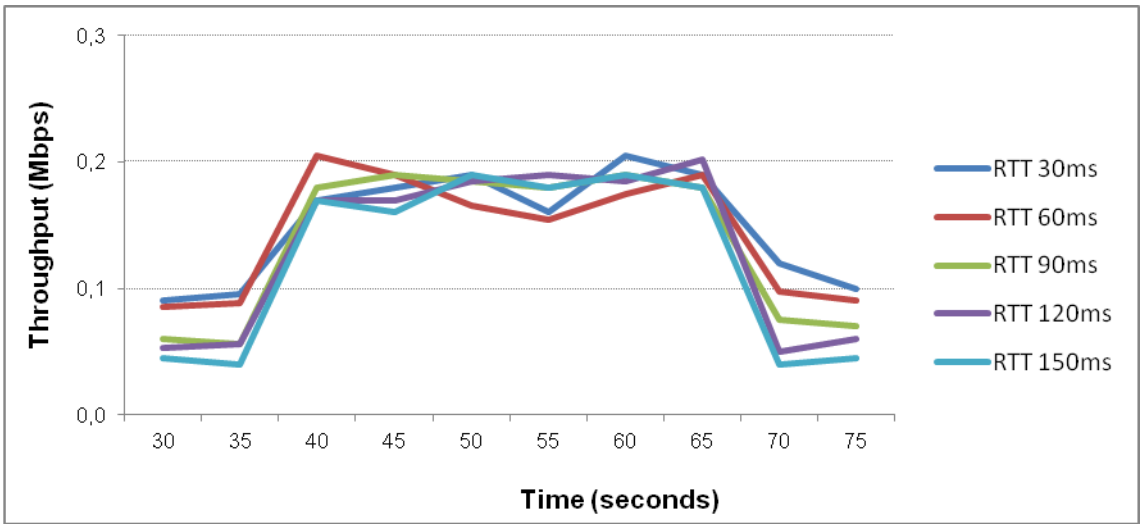


Fig. 3.20 Zoom when 4G interface is disconnected and Wi-Fi interface is active.

We can see that for each case we don't lose the connectivity when making the handover, as seen in previous point when we talked about MPTCP working alone. But now, it is more difficult to take advantage of the Wi-Fi connection.

On the other hand, comparing the results to the case when NEMO is running alone, we can see that this situation lets us to avoid losing the connectivity in the handover.

Trying to understand the throughput variability in each different case of RTT, we will start presenting the signalling messages when both protocols work at the same time. Anyway, as we can see in the received packet flows by the MR, some of the packets are received through the HA and others are received directly to the MR's CoA.

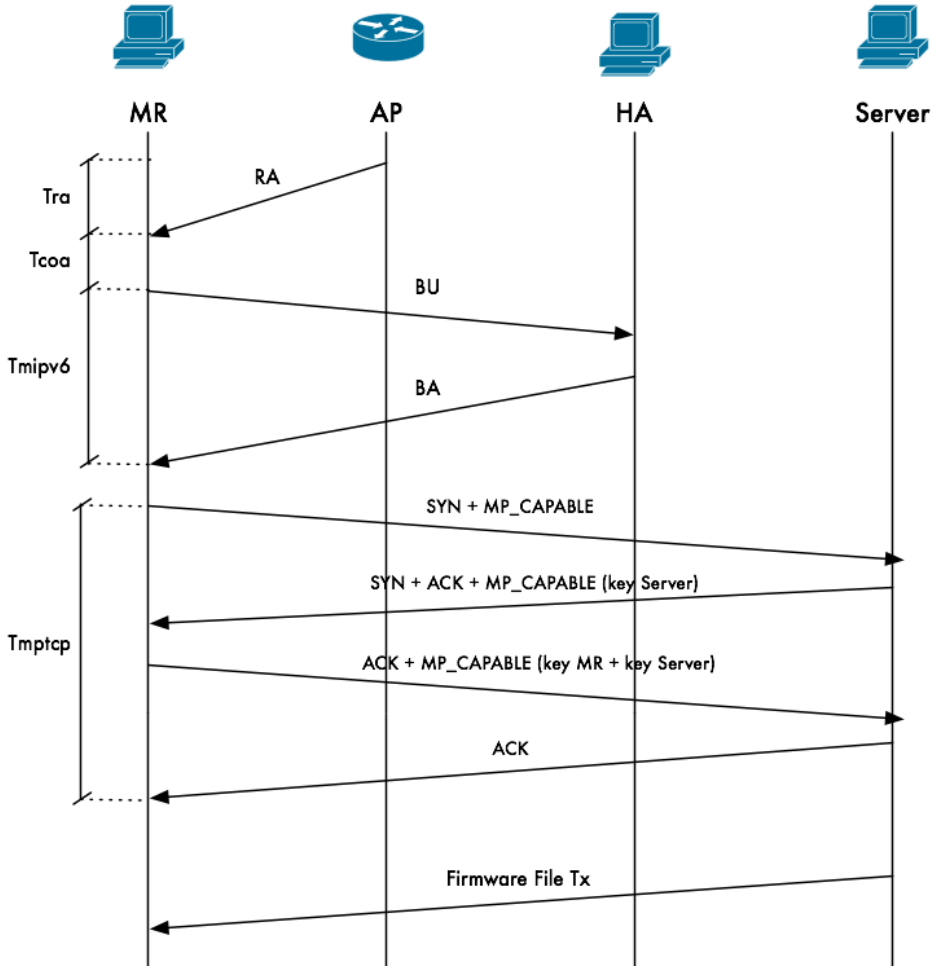


Fig. 3.21 NEMO and MPTCP Signalling messages

The first step is to configure the new CoA in the Wi-Fi network before the BU message is sent to the HA. MIPv6 handover ends when the MR receives the BA message from the HA. Performing all these steps introduces a significant delay and it can be seen when in figure 3.20 the connection takes more time to reach the optimal throughput of the new network.

Also in figure 3.20, when we reach the optimal throughput of Wi-Fi connection, there are fluctuations in the average throughput for most of the cases. A possible explanation can be the fragmentation of IPv6 packets caused by MIPv6 tunnelling, when the HA encapsulates the data packets ready to be tunnelled to the MR's CoA. The HA adds an IPv6 header with the source address the HA address and the destination address the MR's CoA, exceeding the maximum packet size. For that reason, the HA performs fragmentation and this affects to the transmission throughput.

In summary, the combination of MPTCP and NEMO offers better performance compared to NEMO working alone in terms of throughput, but it spends more time to reach the utilization of the second connection as compared to the scenario with only MPTCP running. Another possible benefit regarding to MPTCP working alone is that the combination of both protocols allows us to

support communications working in TCP and UDP mode (MPTCP could only work in TCP mode).

- Far HA

In this case we consider a RTT to the HA of 150ms, approximating the situation where we can have a nearby Server MPTCP and a far HA.

Following figure shows the obtained measures for each value of added delays in this scenario.

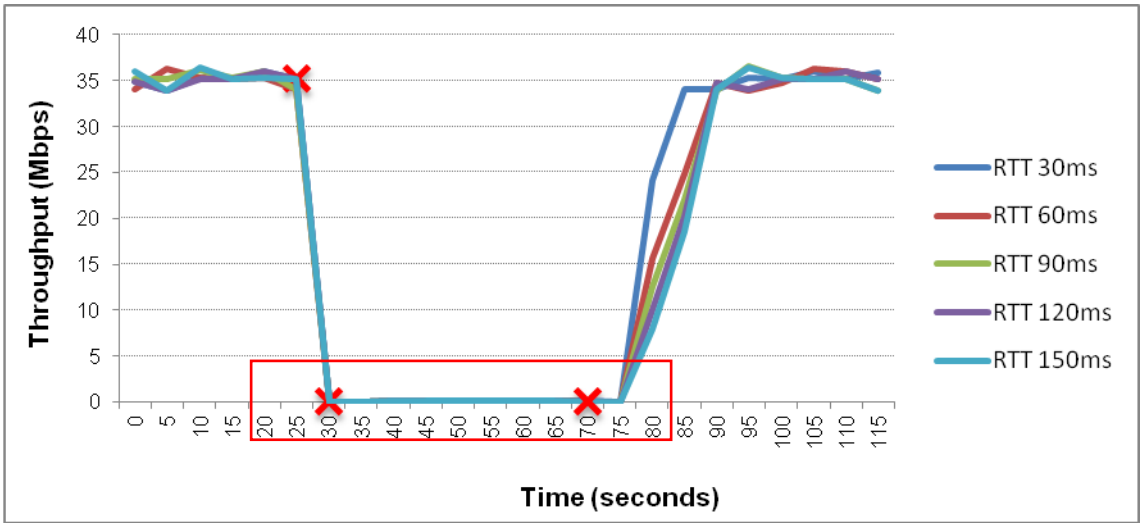


Fig. 3.22 Throughput measures with added delays in a handover situation.

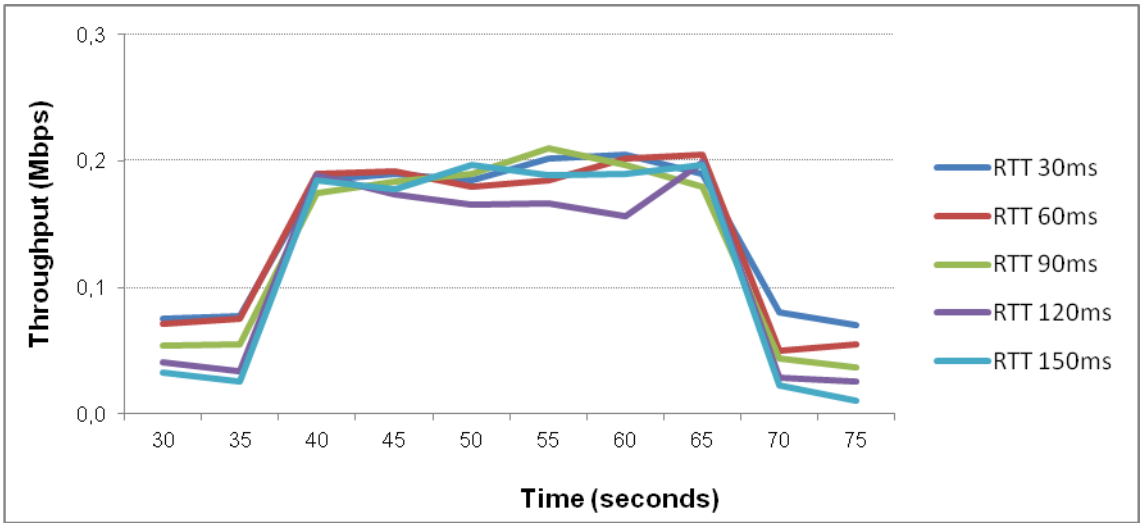


Fig.3.23 Zoom when 4G interface is disconnected and Wi-Fi interface is active.

Now we can see that results are similar to the situation when having a nearby HA: we don't lose the connectivity when making the handover. But unlike previous case, is more difficult to take advantage of the second connection when the handover occurs and this probably happens because of having the

HA further away. This is the main difference between having a nearby or a far HA.

We can also see that the fluctuations in the average throughput caused by the MIPv6 tunnelling in the HA are present in most of the cases, probably more pronounced by the fact that the round-trip delay between the MR and the HA is higher due to the location of the HA.

Anyway, this situation has also a better performance than the case of NEMO running alone in terms of throughput (NEMO loses the connectivity in the moment of the handover) but not good enough compared to the case of MPTCP, since we can see that it's harder to reach the full utilization of the second connection.

3.2.4. Results analysis

Previous benchmark gives us the basic information in order to analyze what will be happen if a handover occurs.

Based on the added delays, the behaviour of each mode (only NEMO, only MPTCP or the combination of both protocols) and the results obtained, we will try to determine the spent time to take benefit of the second connection after the handover, with the aim to decide if it could be made or not.

The case of MPTCP working alone is the easiest one. The time spent to take benefit of the second connection tends to zero because we are using a *make-before-break* approach and the main benefit that we could take into account is how much data consumption would be saved by switching between 4G and Wi-Fi. Finally, it is important to mention that this solution works only with applications that make use of TCP.

In the NEMO case, we should know the time to take profit of the second connection when a handover is made, in addition to how much data consumption would be saved. Taking into account all the possible values of the delays and based on the results obtained in section 3.2.1, the following table shows the spent time to reach an optimal utilization of Wi-Fi connection when a handover from 4G to Wi-Fi is made using NEMO.

Table 3.5. Time to reach an optimal utilization of Wi-Fi connection after handover (NEMO)

RTT	Time
30 ms	14,8 s
60 ms	18,6 s
90 ms	20,2 s
120 ms	23,1 s
150 ms	24,8 s

Now we could assume that, for the case of NEMO working alone, the time spent to achieve a good performance of the second connection can fluctuate between 14,8 seconds and 24,8 seconds depending on the RTT with the HA. In order to determine the optimal moment to make the handover, we can consider this time as the needed one to take benefit of the network switch.

As seen in Figure 3.7, the handover time can fluctuate between 141ms and 1,83s, but this cannot guarantee the optimal performance of the second connection. For that reason, we should consider this new approximation as the best way to predict the optimal moment to make the handover.

Finally, considering the combined mode with NEMO and MPTCP working together, it has been done the same approximation. Next figure shows the needed time to take advantage of the Wi-Fi connection by each different value of RTT. Both cases are shown: with a nearby or a far HA.

Table 3.6. Time to reach an optimal utilization of Wi-Fi connection after handover (Combination of both protocols)

RTT	Time (Nearby HA)	Time (Far HA)
30 ms	10,3 s	10,6 s
60 ms	11,1 s	11,5 s
90 ms	12,4 s	12,8 s
120 ms	12,9 s	13,4 s
150 ms	13,8 s	14,2 s

We can observe that the combination of NEMO with MPTCP gives to the first one a better performance in terms of throughput and spends less time to take benefit of the Wi-Fi connection. Approximately the time is reduced about a 40%, making faster the handover process.

Afterwards, we can approximate the value for total handover time to harness the second connection between 10,3s and 14,2s (minimum and maximum value for both location possibilities). We can observe that combining NEMO and MPTCP could be the best option to face real situations in mobile environments.

3.2.5. Conclusions

Now, we should decide which combination could be the best option in the transmission by focusing on the performance metrics presented above.

In throughput terms, we have seen that MPTCP could reach higher values of bandwidth utilization than NEMO. This in principally caused by the fact that MPTCP could add TCP subflows and maximizes efficiency and, in the other hand, NEMO has an increase on its packet size due to its larger IP header. But

MPTCP is not suited for short transfers because the increase of the congestion window is slower than normal TCP flows, taking more time to reach the maximum throughput. In addition, MPTCP can only work with TCP applications unlike NEMO, which can work with TCP and UDP.

Regarding on the affectation of the delays, MPTCP is more affected than NEMO because of the presence of 2 RTT's on its signalling messages. NEMO has only 1 RTT used in the Binding Update and Binding Acknowledgement messages and is not affected as much as MPTCP.

Finally, in terms related to handover process, NEMO has a total handover time more variable than MPTCP, getting even to lose the connectivity when switching between connections. This variability is due to the CoA's configuration time, component that affects as most the total handover time. But with the implementation of functionalities like MCoA this behaviour could be improved.

Combining NEMO and MPTCP could allow us to reach higher values of throughput and to get more reliability in our gateway instead of using only NEMO. Otherwise, it implies that the transmission will be more affected by the location of the HA and the Server MPTCP and the related delays to each one. It is also important to mention that probably the combination would not have sense if NEMO includes MCoA mechanism in its implementation.

CHAPTER 4. HANDOVER OPTIMIZATION

4.1. Open data for connectivity improvement

Following the lines established in section 1.2, we can use open data APIs in order to get information that allows us to decide when to make a profitable handover between wireless technologies.

We start by assuming that mobile technologies as 3G or 4G will be available while driving the car. In order to reduce data consumption, when Wi-Fi connectivity is available thanks to Wi-Fi free hotspots, probably the best option was to make the handover and connect to these networks.

As we can see in the results of the experiments done in both, NEMO and MPTCP, the average time to detect a node that changes its access network and activate the mechanisms to allow it to continue connected, is between 141 milliseconds and 1,29 seconds. Taking these values into account, handovers that not imply a minimum time of 1,29ms inside the access network coverage area should be avoided.

In addition, it must be remembered that after this time the second connection is not fully exploited because of the effect of the delays. We should consider this time when we can take advantage of the second connection after the handover. Based on all the obtained values in the previous chapter considering all the possible situations, this time fluctuate between 141 milliseconds and 14,2 seconds.

After that, we can consider that in the worst case, if the time inside the access network coverage area exceeds the threshold of 14,2 seconds, the gateway will reach an optimal throughput for the transmission. But we don’t know how long will last the connectivity to the access network.

Using open data sources, we can know how much time we will have Wi-Fi connectivity. Thereby, using information like the location of Wi-Fi hotspots and the traffic status, we should be able to know when we will have an access point and how much time we will spend in its coverage area. Thus, with the combination of external data sources, we will be able to define a strategy in order to decide in which cases is optimal to make the handover.

The decision should also take into account the parameters that we have shown that has impact on protocol performance, like the delays to the HA or the MPTCP server.

Thus, the smart car gateway will embody a decision algorithm to be able to determine the situations within the car route where making a handover is a good option.

Next figure shows a high level vision of this decision algorithm.

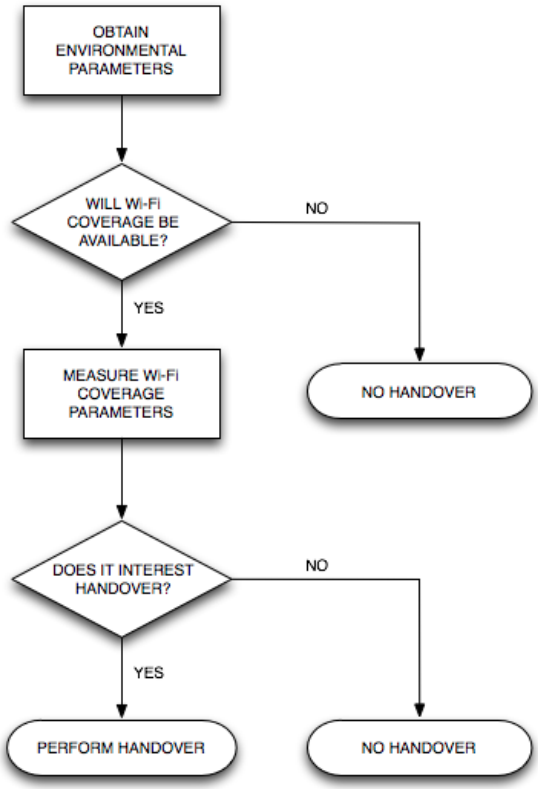


Fig. 4.1 High Level Flow Diagram

The next section shows a deep description of the different steps to be done in each stage.

4.2. Flow diagram description

The decision algorithm (see Fig. 4.2) takes into account all the possible situations regarding to the active protocols: with only NEMO, with only MPTCP and with both protocols working together. The requirement is that one of both protocols will be active in the system. If not, the algorithm will end.

The algorithm follows by checking the open data origins loaded in the smart car gateway. As mentioned in previous sections, data sources can be preloaded in the system or retrieved from an available API. In that case, we should know if there will be Wi-Fi hotspots in our route and how the traffic conditions will be.

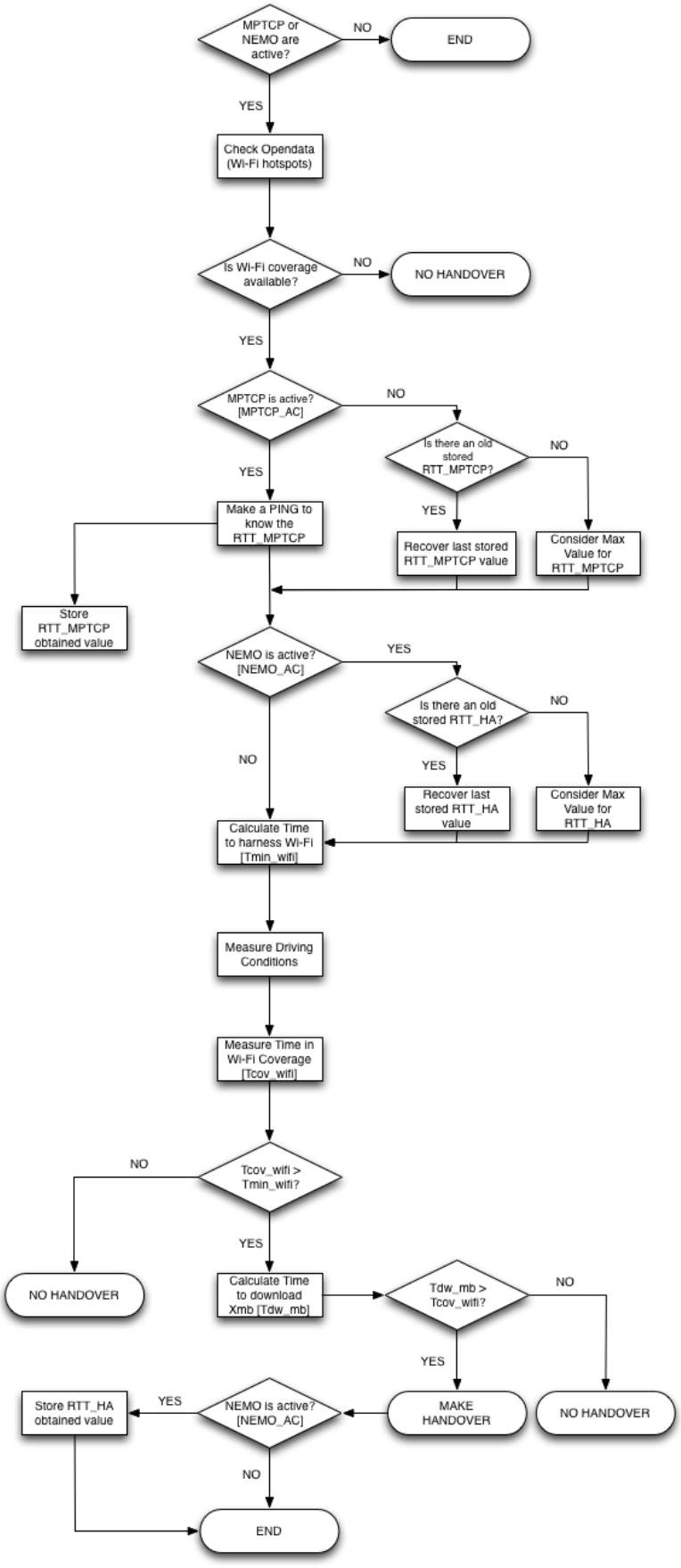


Fig. 4.2 Detailed flow diagram for all the possible situations

The handover decision will take place when the car arrives to a Wi-Fi coverage area inside the route and one of the two supported protocols is active in the system.

The system will now check if MPTCP is active. This is an important check-up in order to determine the best way to calculate the delays. If MPTCP is running, we can ping the host that has the content (server MPTCP) to know the real RTT_MPTCP in one of the available subflows without losing the connectivity. It is assumed that for a specific host, the ping result will be the same independently of the location inside the Wi-Fi municipal network.

If we only had available NEMO, it won't be possible to obtain a real value of RTT to the HA because we will lose the connectivity when making the ping. In this case, we assume that the value of RTT_HA will be the same regardless of the Wi-Fi access point in which we are connected. In order to obtain this value, we will take the time to complete a register to the HA with the BU/BA signalling messages (see Fig. 3.2) using the Wi-Fi network.

If this is the first time that the gateway connects to a Wi-Fi network, we must consider the worst case for the RTT_HA (150ms as seen in previous sections) because its value is unknown. For future occasions, the system can recover a recent value of RTT_HA stored in previous handovers.

Returning to the situation in which we have available MPTCP, the system will ping the MPTCP server to know the real RTT and will store its value for future iterations.

Finally, once the gateway knows the RTT by one of the possibilities mentioned above, it will be able to calculate the needed time to take profit of the Wi-Fi connection according to the working case: only NEMO, only MPTCP or both (section 3.2.4). This time is T_{min_wifi} and would be estimated using the values obtained in chapter 3.

Afterward, the gateway should get some information about driving conditions in each stage of the route. By the help of external information as traffic conditions, combined with the car speed in each sector, the system will be able to determine the time that the car would stay in each Wi-Fi coverage area. This time is T_{cov_wifi}.

Having determined its T_{cov_wifi} and T_{min_wifi}, in that case ($T_{cov\_wifi} > T_{min\_wifi}$) the system will follow to the next step, but if not, the handover will be discarded.

Once the system checks if the car would stay the optimal time inside a Wi-Fi coverage area, i.e. $T_{cov\_wifi} > T_{min\_wifi}$, the final step is to know if a minimum of information can be downloaded (or uploaded). The reason of this checking is to know if the reduction of access velocity will be compensated by a minimal data transfer.

The gateway should measure the time to download X megabytes of information (T_{dw_mb}) in order to know if this minimum information can be downloaded while

staying in the coverage area. If the answer is yes, i.e. $T_{cov\_wifi} > T_{dw\_mb}$, the system will complete the decision flow and the handover will be made. Finally, the gateway stores the obtained value of RTT_HA for future occasions.

The next table summarizes all the variables and parameters considered in the algorithm.

Table 4.1. Variables and parameters employed in the decision algorithm

T_{min_wifi}	Time to take benefit of Wi-Fi connection.
T_{cov_wifi}	Time inside the Wi-Fi coverage area.
T_{dw_mb}	Time to download X megabytes of information.
X	Minimum quantity of bytes to justify the handover.
RTT_MPTCP	Last stored RTT value to the server MPTCP.
RTT_HA	Last stored RTT value to the HA.
MPTCP_AC	Determine if MPTCP is active.
NEMO_AC	Determine if NEMO is active.

Next section shows a random route example where the handover decisions are based on this algorithm.

4.3. Application in a random route

In order to get an example that illustrates this feature, we apply the decision algorithm in a random route between two points to determine the handovers between networks technologies (3G/4G and Wi-Fi) during the driving time. It is assumed that both protocols (NEMO and MPTCP) are active.

Next figure shows the chosen route to apply our decision algorithm. It is comprised between *Carrer Provençals 39* and *Avinguda Diagonal 621*, both in the city of Barcelona.

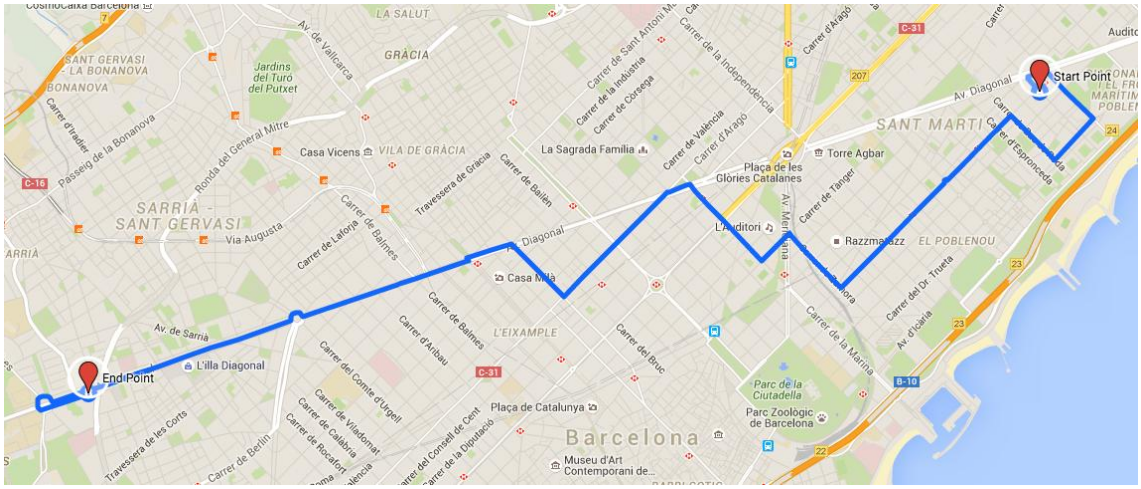


Fig. 4.3 Chosen route to apply handover strategy.

Here the route with Wi-Fi hotspots placed around the path. As we seen before, this is the first check-up by the algorithm in order to determine if we will find Wi-Fi coverage areas.

Fig. 4.4 Wi-Fi hotspots (Open data).

We can observe that there are many Wi-Fi hotspots around the chosen route and all of them can be suitable in order to support the connection of our smart car gateway. The decision algorithm passes to the next step.

Regarding to 3G/4G coverage we will see that in the same area covered by the route we will not have any kind of coverage problem. We can see it in the heat map shown in figure 4.5.

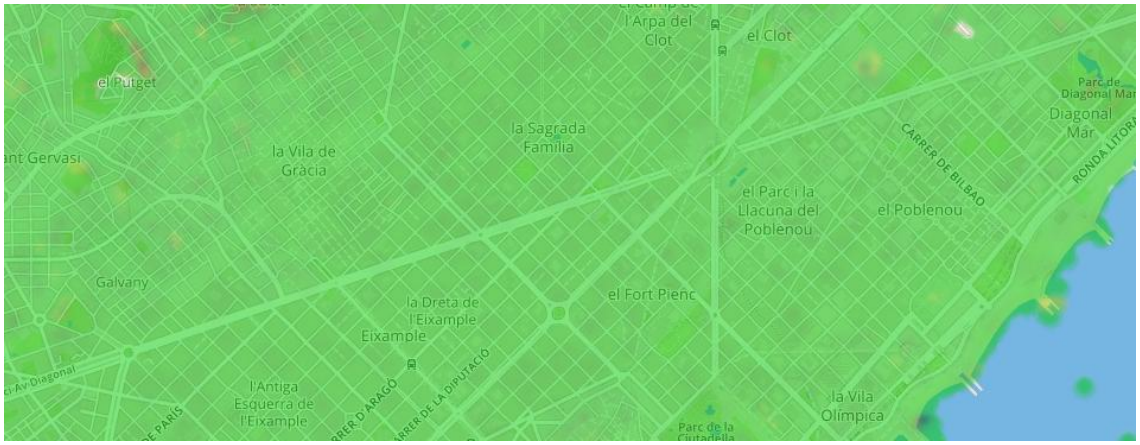


Fig. 4.5 3G/4G Coverage level along the route

From there, the system will check if the relation of the speed, the coverage radius of the hotspots and the time that the car will need to connect the network would fit the conditions set in the decision algorithm.

For doing that, it has to be known the average time to connect to a Wi-Fi hotspot, the typical coverage radius of an 802.11 antenna and the mean of driving time across some established sections in the route. The Wi-Fi access points placed by the Barcelona City Council allow for theoretical coverage within a range of 100 to 150 metres in outdoors spaces, as we can check in service

user manual [16]. For the calculations we will assume the minimum value of 100 metres for the maximum coverage for the antennas.

Regarding to the average time to connect the Wi-Fi network and take advantage of it, first of all the system should obtain the RTT value to the HA. In this example, we consider that MPTCP interface is active and we finally obtain that the real value of the delay is about 150ms.

Afterwards, based on RTT measures (section 3.2.4) we would consider the time of 14,2 seconds, the worst case obtained in tests, as the time to get ready to connect to the new access point and reach an optimum throughput to better exploit the new connection in the worst case of RTT (see table 3.6). For that reason we will only make the thresholds in route's stages where we can have this minimum time (T_{min_wifi}) in order to connect to the Wi-Fi access point.

Table 4.2. Parameters to take into account in the handover's decision

Wi-Fi hotspots coverage – To calculate T_{cov_wifi}	100 metres
Mean time to get ready to connect Wi-Fi hotspots and get an optimal throughput	14,2 seconds
Minimum time inside coverage area to make the handover (rounded) – T_{min_wifi}	14 seconds
Time to download X bytes of information (X = 0) – T_{dw_mb}	0 seconds

Although minimum expected time inside a Wi-Fi coverage area to make the handover is 14 seconds (T_{min_wifi}), we will take into account that we should stay more time inside Wi-Fi zone in order to reach an optimal utilization of the connection.

With all these requirements, the gateway can now divide the entire route into different stages and decide if it's a good handover situation or not, taking into account the average speed and the spent time to cross each one (T_{cov_wifi}). The first step is to select the Wi-Fi hotspots that are at a maximum distance of 100 metres from the route. Figure below shows the selected access points.

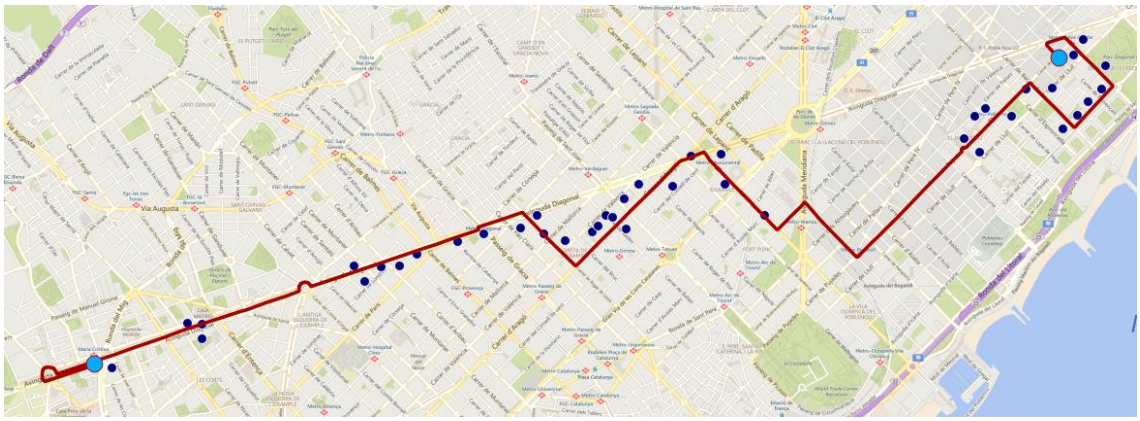


Fig. 4.6 Wi-Fi hotspots at up to 100 metres from the route

The following step is to identify and characterize each stage on the route, as presented below.

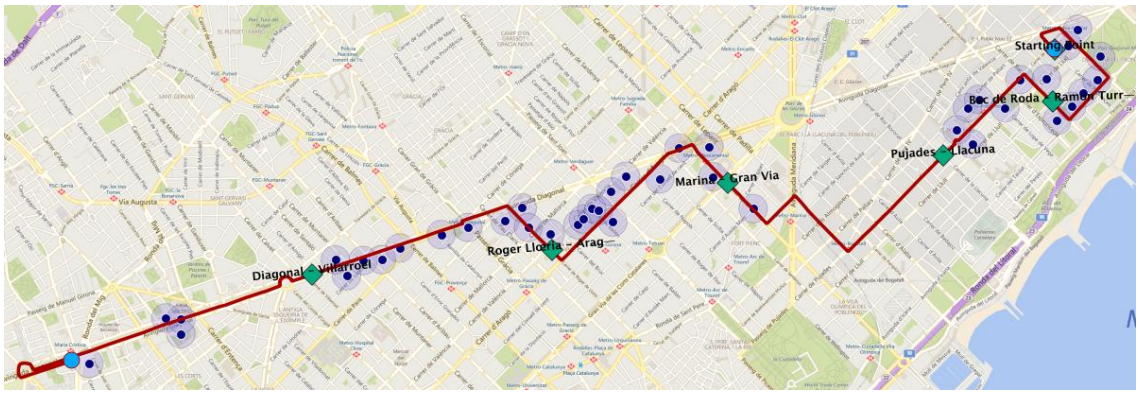


Fig. 4.7 Route Stages (green squares mark them)

Table 4.3. Detail of the route's stages

Stage	Start	End	Distance (m)	Time (min)	Avg Speed (km/h)
1	Starting Point	Bac de Roda - Ramon Turró	1.400	5	16,80
2	Bac de Roda - Ramon Turró	Pujades - Llacuna	900	6	9,00
3	Pujades - Llacuna	Marina - Gran Via	2.450	8	18,38
4	Marina - Gran Via	Roger Llúria - Aragó	1.520	5	18,24
5	Roger Llúria - Aragó	Diagonal - Villarroel	2.150	6	21,50
6	Diagonal - Villarroel	End Point	2.580	7	22,11

The route has a total of 11 km driven in 37 minutes, divided in 6 stages as described in the table shown above. Stages 1, 2, 4 and 5 are initially potential to be steps where we can make the handover and change to Wi-Fi network.

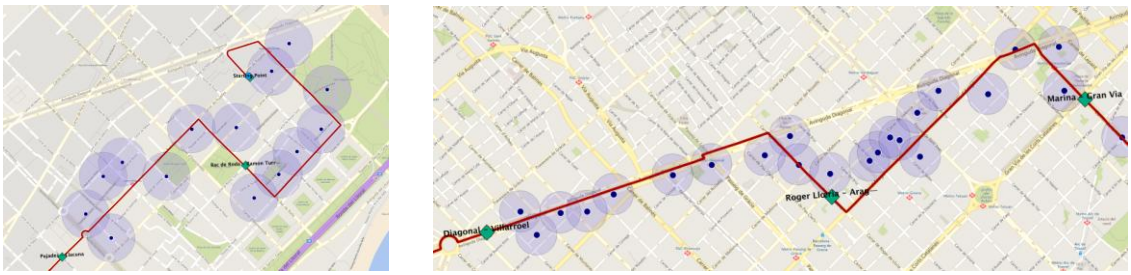


Fig. 4.8 Stages 1, 2 (left) and 4,5 (right).

We can see that over the distance of these stages there are many hotspots that can allow the connectivity with the public Wi-Fi network without interruption (blue circles around the hotspots simulates its coverage area). But in stages 2, 4 and 5 there are shadow zones of 200 metres, 100 metres and 300 metres respectively. In these cases we will make the handover back when crossing

them because the next hotspots in the route allows more time of Wi-Fi connection.

Finally and after all the steps in the decision algorithm, the gateway decides to activate Wi-Fi connection instead of maintaining 3G/4G in each route stage. In this example, it is considered that the minimum of information to be downloaded is 0 bytes (i.e. X = 0 bytes and T_{dw_mb} = 0 seconds).

The final decisions made by the algorithm are the following.

Table 4.4. Active connections on each stage

Stage	Distance (m)	Time (min)	Connection
1	1.400	5	Wi-Fi
2	900	6	Wi-Fi & 4G
3	2.450	8	4G
4	1.520	5	Wi-Fi & 4G
5	2.150	6	Wi-Fi & 4G
6	2.580	7	4G

4.4. Supported applications

Initially, the decision algorithm presented before is designed without thinking the requirements of the applications that could be running when handover decision is taken. In this section, the requirements of several typical multimedia services, as for example audio or video streaming, are shown in order to potentially include them in the handover decision.

Next table shows an overview of different services and its optimal conditions to work [24] [25] [26]. Note that all the listed services are based on TCP (Skype could also work with UDP).

Table 4.5. Optimal Conditions in some types of multimedia services

Service Type	Provider	Quality	Minimum Bandwidth
Audio Streaming	Spotify	160 Kbps	384 Kbps
Audio Calling	Skype	-	30 Kbps
Video Calling	Skype	Min. quality	128 Kbps
Video Streaming	Youtube	SD 240p	300 Kbps
Video Streaming	Youtube	HD 720p	2 Mbps
Video Streaming	Netflix	HD 720p	5 Mbps

As we seen in previous section 3.2.2, we can assume that using MPTCP the bandwidth utilization for public Wi-Fi is about 71% of the theoretical maximum

and for 4G is about 89%. Therefore we can take by reference the values of 181,7 Kbps for public Wi-Fi and 35,6 Mbps for 4G connectivity.

Following these values, public Wi-Fi, will be only able to support audio and video callings with a minimum of quality. For the other cases we cannot guarantee an optimal user experience.

The idea is to allow the user to configure which technology will be used and in which services. For example, the driver could choose to only use audio/video callings when Wi-Fi is available or to use 4G in all the services to guarantee a high quality in all of them, but assuming an increase of data consumption. Only in the first case, the gateway will activate the decision flow defined in the algorithm.

CHAPTER 5. CONCLUSIONS

This thesis proposes the main lines to take benefit of multihoming capacity from MPTCP and NEMO protocols, and also introduces the use of open data in order to achieve a better performance in the smart car gateway.

In this context, it has been made a demonstrator that tries to reproduce the conditions of an urban environment with 3G/4G and Wi-Fi coverage in order to study the protocols.

First of all, in terms of throughput, we obtain that MPTCP has better results than NEMO. MPTCP could reach higher values of throughput due to its ability to add TCP subflows, but it spends more time to reach the maximum throughput than NEMO because of its slower congestion window.

Regarding to the delays, MPTCP is more affected due to the 2 RTT's in its signalling messages. NEMO has only 1 RTT used in the BU and BA messages and is not affected as much as MPTCP.

If we analyze the total time for the handover, NEMO has more variability than MPTCP because of the CoA's configuration time, getting even to lose the connectivity when switching between access points. But if NEMO implements functionalities like MCoA, this behaviour could be improved avoiding the disconnection in the handover process.

Combining both protocols could allow the system to reach higher values of throughput and more reliability. But it implies that the transmission will be more affected by the location of the HA and the server MPTCP and the related delays to each one. But if we consider the implementation of MCoA in NEMO, as mentioned before, probably this combination would not make sense.

Finally, we have seen that the developed decision algorithm could allow the gateway to decide the best moment for the handover, taking into account all the acquired knowledge about the protocols behaviour and including external information to enhance the decision process. This will allow the system to consider the handovers only when we can reduce data consumption in situations when Wi-Fi network is available and the gateway can take benefit of the connection without a drastic losing of quality in content downloading.

5.1. Future work

After all, there are some aspects that are not taken into account and probably could define the future lines to be considered in order to extend the project.

The following list tries to explain some of these future considerations.

- The handover decision algorithm takes into account a constant speed of the car. This could be improved by taking into account the real speed of

the car using external API's (from Google, for example) or a GPS system. Also this information could be used to estimate the speed in each route stage before leaving home.

- In order to improve the user experience, we can also develop new features in the decision algorithm of the gateway. For example, imagine the case where the user wants to download a specific content while driving to its workplace. It is possible to include a system that allows the gateway to calculate the optimal route (taking into account information about real time traffic status or the presence of Wi-Fi hotspots) and to predict the specific coordinates where to make the handover before leaving home. Thanks to this feature, it could be possible to manage in a smarter way the content downloading and the data consumption.

- The possibility to implement the gateway in an embedded device with real radio interfaces. For example, in a Raspberry PI board with both protocols enabled in order to include the decision algorithm and test it in real driving situations. This option has been explored with no success because a kernel implementation that supports the mobility protocols has not been found. This has been the main reason that has prevented the development of the smart car gateway in a real device.

Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANNEXES

TITLE: Design and development of a smart car gateway

MASTER DEGREE: Master of Science in Telecommunication Engineering & Management

AUTHOR: Adrià López Molina

DIRECTOR: Rafael Vidal Ferré

DATE: February 22th 2015

ANNEX I. MOBILITY SOLUTIONS CONFIGURATION

I.1 Multipath TCP

I.1.2 MPTCP test bed

In order to test Multipath TCP implementation, there are different ways to build a test bed scenario. Finally, the option chosen from the list shown in "*Software Requirements*" section is the one referred to Vagrant [27] in order to get a guest running MPTCP.

This software tool allows us to set up a development environment in order to test the protocol. It provides an easy to configure and portable work environments built on top of VirtualBox (and some other virtualization software).

It runs a virtual machine based on Ubuntu 14.04.1 LTS (Trusty Tahr 64-bit) operating system with the MPTCP kernel implementation (3.14.0-89-mptcp). The built test bed using vagrant is the following:

Fig. I.1 Set up machines in the test bed.

Both physical machines are running Vagrant and using MPTCP. Each one has assigned 1 core of an Intel Core i5 processor at 2,4GHz, RAM memory of 1024Mb and 3 network interfaces: a wired Ethernet connection (eth1), an 802.11g wi-fi connection (eth2) and a 3G connection (eth3) using an iPhone 5S running iOS 8 in one of the machines (server) and a Nokia Lumia 735 in the other machine (client).

It's important to mention some features related to the network connections that will affect the performance of MPTCP. For example, the maximum bandwidth in each network type (**tested with network tools*) and the symmetry of the link (in this case, 3G has an asymmetric link):

- Wired Ethernet Connection: 100Mbps
- 802.11g Wi-fi Connection: 54 Mbps
- Vodafone 3G Connection*: about 12 Mbps (DL) and 3 Mbps (UL)

For example, we can observe that Wi-fi has a yield of about 30% lower than Ethernet. Fact that will affect MPTCP while managing subflows.

Fig. I.2 Network schema with the three network interfaces.

Vagrant shares network drivers with the host machine and lets us to configure and maintain many interfaces as we need. Two steps are necessary in order to include and configure a network interface in Vagrant: edit Vagrant's configuration file called "*Vagrantfile*" that includes VM's boot parameters and configure routing tables once VM runs Linux OS.

When all the previous steps are correctly done, we can start using Vagrant's machines to develop our benchmark. Finally, the map of the IP addresses in the test bed environment is the following.

CLIENT – Network interfaces

- Ethernet connection (eth1):
  - IP: 192.168.2.103
  - Mask: 255.255.255.0
  - Gateway: 192.168.2.1

- Wi-fi connection (eth2) – 802.11g:
  - IP: 192.168.1.103
  - Mask: 255.255.255.0
  - Gateway: 192.168.1.1

- 3G connection (eth3) – Nokia Lumia 735:
  - IP: 172.20.10.3
  - Mask: 255.255.255.240
  - Gateway: 172.20.10.1

SERVER – Network interfaces

- Ethernet connection (eth1):
  - IP: 192.168.2.101
  - Mask: 255.255.255.0
  - Gateway: 192.168.2.1

- Wi-fi connection (eth2) – 802.11g:
  - IP: 192.168.1.101
  - Mask: 255.255.255.0

- Gateway: 192.168.1.1

- 3G connection (eth3) – iPhone 5S:
  - IP: 172.20.10.2
  - Mask: 255.255.255.240
  - Gateway: 172.20.10.1

The next step in the set up of the test bed environment is the routing configuration on each host. There are a few steps to follow in order to get ready the scenario.

1. Configure routing rules.

CLIENT

```
#Creation of routing tables (based on the source-address)
ip rule add from 192.168.1.103 table 1
ip rule add from 192.168.2.103 table 2
ip rule add from 172.20.10.3 table 3

#eth1 - ethernet interface
ip route add 192.168.2.0/24 dev eth1 scope link table 2
ip route add default via 192.168.2.1 dev eth1 table 2

#eth2 - wifi interface
ip route add 192.168.1.0/24 dev eth2 scope link table 1
ip route add default via 192.168.1.1 dev eth2 table 1

#eth3 - 3G interface
ip route add 172.20.10.0/28 dev eth3 scope link table 3
ip route add default via 172.20.10.1 dev eth3 table 3

#Default route for the selection process of normal internet-traffic
ip route add default scope global nexthop via 192.168.1.1 dev eth2
```

Fig. I.3 Configuring routing tables on the client.

SERVER

```
#Creation of routing tables (based on the source-address)
ip rule add from 192.168.1.101 table 1
ip rule add from 192.168.2.101 table 2
ip rule add from 172.20.10.2 table 3

#eth1 - ethernet interface
ip route add 192.168.2.0/24 dev eth1 scope link table 2
ip route add default via 192.168.2.1 dev eth1 table 2

#eth2 - wifi interface
ip route add 192.168.1.0/24 dev eth2 scope link table 1
ip route add default via 192.168.1.1 dev eth2 table 1

#eth3 - 3G interface
ip route add 172.20.10.0/28 dev eth3 scope link table 3
ip route add default via 172.20.10.1 dev eth3 table 3

#Default route for the selection process of normal internet-traffic
ip route add default scope global nexthop via 192.168.1.1 dev eth2
```

Fig. I.4 Configuring routing tables on the server.

The aim is to configure routing rules so that packets with source-IP 192.168.1.XXX will get routed over eth2, packets with source-IP 192.168.2.XX will get routed over eth1 and those packets with source-IP 172.20.10.XX will get routed over eth3.

2. Once routing tables are configured, we should run some commands in order to ensure that they're correctly established.

CLIENT

- **Command:** ip rule show

```
vagrant@clientmachine:~$ ip rule show
0:      from all lookup local
32763:  from 172.20.10.3 lookup 3
32764:  from 192.168.2.103 lookup 2
32765:  from 192.168.1.103 lookup 1
32766:  from all lookup main
32767:  from all lookup default
```

Fig. I.5 Ip rule show on the client.

- **Command:** ip route

```
vagrant@clientmachine:~$ ip route
default via 192.168.1.1 dev eth2
10.0.2.0/24 dev eth0  proto kernel  scope link  src 10.0.2.15
172.20.10.0/28 dev eth3  proto kernel  scope link  src 172.20.10.3
192.168.1.0/24 dev eth2  proto kernel  scope link  src 192.168.1.103
192.168.2.0/24 dev eth1  proto kernel  scope link  src 192.168.2.103
```

Fig. I.6 Ip route on the client.

- **Command:** ip route show table X (1, 2 and 3)

```
vagrant@clientmachine:~$ ip route show table 1
default via 192.168.1.1 dev eth2
192.168.1.0/24 dev eth2  scope link
```

Fig. I.7 Ip route show table 1 on the client.

```
vagrant@clientmachine:~$ ip route show table 2
default via 192.168.2.1 dev eth1
192.168.2.0/24 dev eth1  scope link
```

Fig. I.8 Ip route show table 2 on the client.

```
vagrant@clientmachine:~$ ip route show table 3
default via 172.20.10.1 dev eth3
172.20.10.0/28 dev eth3  scope link
```

Fig. I.9 Ip route show table 3 on the client.

SERVER

- **Command:** ip rule show

```
vagrant@servermachine:~$ ip rule show
0:      from all lookup local
32763:  from 172.20.10.2 lookup 3
32764:  from 192.168.2.101 lookup 2
32765:  from 192.168.1.101 lookup 1
32766:  from all lookup main
32767:  from all lookup default
```

Fig. I.10 Ip rule show on the server.

- **Command:** ip route

```
vagrant@servermachine:~$ ip route
default via 192.168.1.1 dev eth2
10.0.2.0/24 dev eth0  proto kernel  scope link  src 10.0.2.15
172.20.10.0/28 dev eth3  proto kernel  scope link  src 172.20.10.2
192.168.1.0/24 dev eth2  proto kernel  scope link  src 192.168.1.101
192.168.2.0/24 dev eth1  proto kernel  scope link  src 192.168.2.101
```

Fig. I.11 Ip route on the server.

- **Command:** ip route show table X (1, 2 and 3)

```
vagrant@servermachine:~$ ip route show table 1
default via 192.168.1.1 dev eth2
192.168.1.0/24 dev eth2  scope link
```

Fig. I.12 Ip route show table 1 on the server.

```
vagrant@servermachine:~$ ip route show table 2
default via 192.168.2.1 dev eth1
192.168.2.0/24 dev eth1  scope link
```

Fig. I.13 Ip route show table 2 on the server.

```
vagrant@servermachine:~$ ip route show table 3
default via 172.20.10.1 dev eth3
172.20.10.0/28 dev eth3  scope link
```

Fig. I.14 Ip route show table 3 on the server.

3. Finally, we can ensure that we're using MPTCP implementation on both machines using the following command (*curl command* [28]).

curl www.multipath-tcp.org

And getting the following response:

On the Server

```
vagrant@servermachine:~$ curl www.multipath-tcp.org
Yay, you are MPTCP-capable! You can now rest in peace.
```

On the Client

```
vagrant@clientmachine:~$ curl www.multipath-tcp.org
Yay, you are MPTCP-capable! You can now rest in peace.
```

Fig. I.15 MPTCP environment correctly configured in Server and Client

After all this steps, the environment is properly configured and we can start to test MPTCP protocol.

I.1.3 Test 1 – MPTCP subflows

The first test done is to ensure that MPTCP subflows are working when we make a request from an MPTCP capable host to another one.

We can start from our client machine and the iperf command request to an MPTCP capable server like www.multipath-tcp.org. Once the iperf command is sent from the client machine, we can observer that one MPTCP subflow is created in order to establish the connection, as shown below.

1. Sending the iperf request: ***iperf –c multipath-tcp.org***

```
vagrant@servermachine:~$ iperf -c multipath-tcp.org
------------------------------------------------------------
Client connecting to multipath-tcp.org, TCP port 5001
TCP window size: 45.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.101 port 45999 connected with 130.104.230.45 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.1 sec  2.50 MBytes  2.07 Mbits/sec
```

Fig. I.16 Iperf command from client to MPTCP project's server

2. Observing the created MPTCP subflow: ***cat /proc/net/mptcp***

```
vagrant@servermachine:~$ cat /proc/net/mptcp
  sl  loc_tok  rem_tok  v6 local_address                remote_address                st ns tx_queue rx_queue inode
   0: 836EB53D E16454C1  0 6501A8C0:B3AF                2DE66882:1389                 01 04 0002E9CA:00000000 19528
```

Fig. I.17 Created MPTCP subflow

On the same way, we also can observe created MPTCP subflows when we sent packets between both machines on our environment, from client to server. In this case, there are two created subflows. Also we can be view a little more detail of the MPTCP connection using the command ***netstat –m*** (after installing net-tools [29], that allows us to analyse MPTCP traffic with the **netstat** command).

```
vagrant@servermachine:~$ cat /proc/net/mptcp
  sl  loc_tok  rem_tok  v6 local_address                remote_address                st ns tx_queue rx_queue inode
   0: 9F1A4448 5B7D1F5A  0 6501A8C0:94BC                6701A8C0:22B8                 01 0C 000FE274:00000000 20196
```

Fig. I.18 Created MPTCP subflows in our Client-Server environment

```
vagrant@servermachine:~$ netstat -m
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       Local Token Remote Token
mptcp       0 1098132 servermachine:38076     192.168.1.103:8888      ESTABLISHED 2669298760  1534926682
```

Fig. I.19 Created MPTCP subflows using netstat-m

I.1.4 Test 2 – Server-Client file transfer

The main objective of this second test is to point out that MPTCP protocol is able to change interfaces on the fly when it's running.

For example, imagine a case in which an interface comes down for any reason. In theory, MPTCP protocol can switch the interface with other available ones and this is precisely the function that this test would check.

The first thing is to explain the scenario in which the test was performed. It is based on a client-server architecture, like the shown in figure 1.1. The actions to be performed and the tools used in the test are detailed below.

1. The first step is to run *netserver* (server command of *netperf* [30]) in the client side. The server will send traffic with *netperf* command and the client has to be configured as the one who receive those packets.

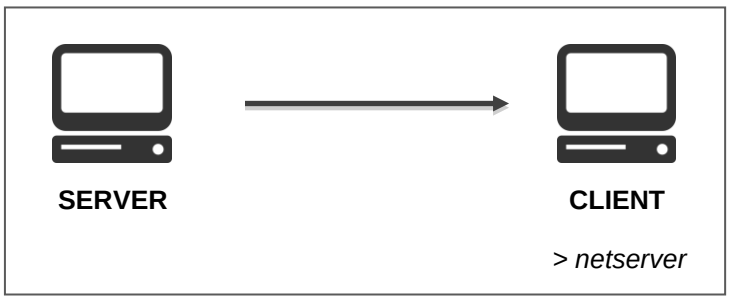


Fig. I.20 Run *netserver* command in client side

2. The following action is to execute *netperf* command in the server side in order to send traffic to the client. But previously, we need a tool that monitors the received packet flows in each interface on the client side. The chosen tool is *bwm-ng* [31], that allows getting all the traffic received in the client in real-time.

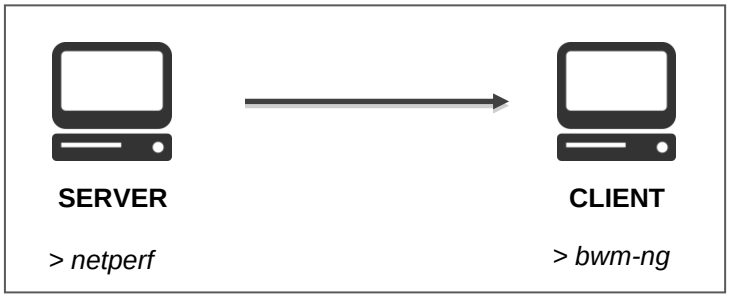


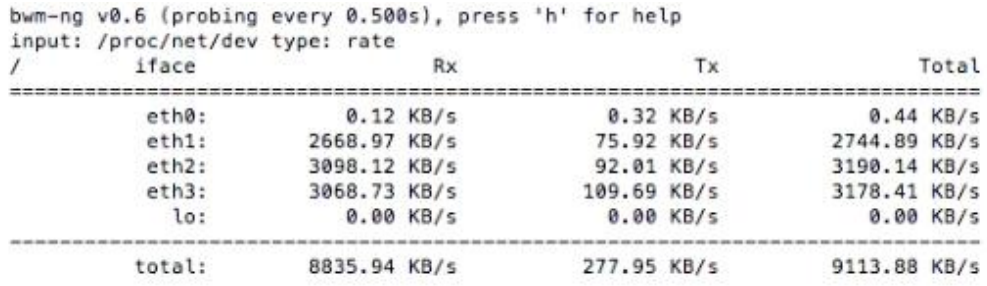
Fig. I.21 Run *bwm-ng* command in client side and *netperf* in server side

Once *bwm-ng* is running, it's time to run *netperf* command. For the test, we will send traffic from server to client (simulating, for example, a download) during 30 seconds and with a buffer size of 10M. The executed command is the following.

netperf –t omni –H 192.168.1.103 –l 30 –T 1/1 –c –C -- -m 10M –V

- *-t omni:* sends a single data-stream
- *-H 192.168.1.103:* destination client's IP address
- *-l 30:* duration of the test
- *-T 1/1:* pin applications on CPU1
- *-c –C:* CPU-usage statistics (at the end)
- *-m 10M:* size of send-buffer
- *-V:* zero-copy send and receive

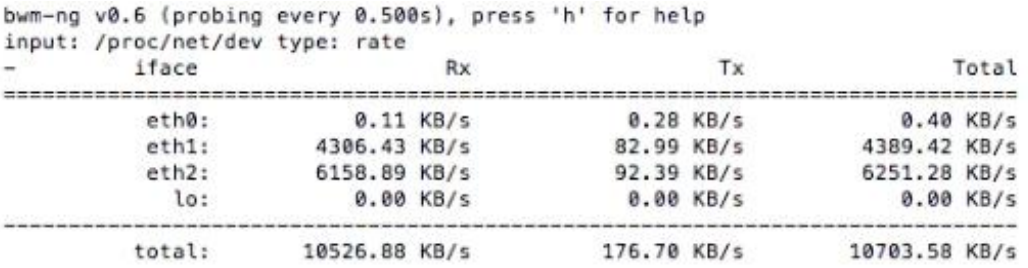
3. At this point, we are able to view that all the 3 interfaces configured in the client are receiving data from the server.



```
bwm-ng v0.6 (probing every 0.500s), press 'h' for help
input: /proc/net/dev type: rate
/         iface                   Rx                   Tx                Total
==============================================================================
            eth0:           0.12 KB/s            0.32 KB/s            0.44 KB/s
            eth1:        2668.97 KB/s           75.92 KB/s         2744.89 KB/s
            eth2:        3098.12 KB/s           92.01 KB/s         3190.14 KB/s
            eth3:        3068.73 KB/s          109.69 KB/s         3178.41 KB/s
              lo:           0.00 KB/s            0.00 KB/s            0.00 KB/s
------------------------------------------------------------------------------
           total:        8835.94 KB/s          277.95 KB/s         9113.88 KB/s
```

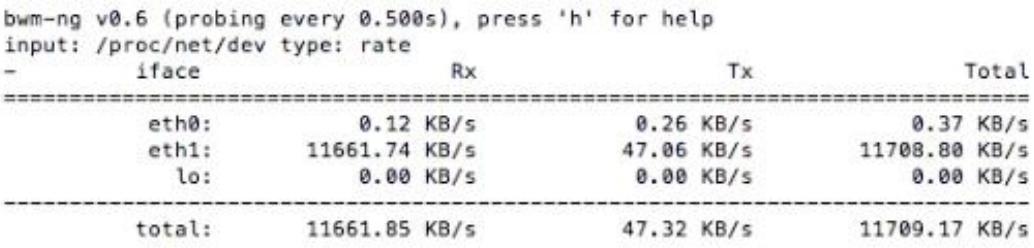
Fig. I.22 Traffic received by client side on its 3 interfaces (eth1, eth2, eth3)

4. During the transmission, if we stop an interface (by doing "*ifconfig ethX down*" in the client) we can see that MPTCP protocol is able to switch the traffic transmission to the other working interfaces in the client side.



```
bwm-ng v0.6 (probing every 0.500s), press 'h' for help
input: /proc/net/dev type: rate
-         iface                   Rx                   Tx                Total
==============================================================================
            eth0:           0.11 KB/s            0.28 KB/s            0.40 KB/s
            eth1:        4306.43 KB/s           82.99 KB/s         4389.42 KB/s
            eth2:        6158.89 KB/s           92.39 KB/s         6251.28 KB/s
              lo:           0.00 KB/s            0.00 KB/s            0.00 KB/s
------------------------------------------------------------------------------
           total:       10526.88 KB/s          176.70 KB/s        10703.58 KB/s
```

Fig. I.23 Eth3 (3G interface) down.



```
bwm-ng v0.6 (probing every 0.500s), press 'h' for help
input: /proc/net/dev type: rate
-         iface                   Rx                   Tx                Total
==============================================================================
            eth0:           0.12 KB/s            0.26 KB/s            0.37 KB/s
            eth1:       11661.74 KB/s           47.06 KB/s        11708.80 KB/s
              lo:           0.00 KB/s            0.00 KB/s            0.00 KB/s
------------------------------------------------------------------------------
           total:       11661.85 KB/s           47.32 KB/s        11709.17 KB/s
```

Fig. I.24 Eth2 (Wi-fi interface) and Eth3 (3G interface) down.

If we made the same test but with a transmission from a non-capable MPTCP host, we can observe that the transmission is only between the interfaces configured in the same IP-address space. For example, if we execute the *netperf* command detailed in step 2, the transmission is only between eth2 on the server (192.168.1.101) and eth2 on the client (192.168.1.103). The other network interfaces do not take part in the download.

Although we can observe that the transmission is stopped (*"errno 60: no response received"*) when we shut down the working interface eth2 (active in the transmission) in the client.

I.1.4 Test 3 – MPTCP connection properties

The third test performed has the aim of retrieve some relevant network properties of the MPTCP transmission such as bandwidth, throughput and RTT.

The scenario is still being the same as the previous test (client-server architecture) and some other tools are used in order to get all the network parameters in every analysed case:

- *Iperf* [21]*:* software to send and receive traffic in the client-server architecture.
- *Tcpdump* [32]*:* software to capture traffic sent in the transmission.
- *Tcptrace* [33]*:* software to characterize and generate statistics of the traffic.
- *Xplot* [34]*:* software to generate graphs of the recorded traffic data.

The followed steps to make this test with the tools described previously are detailed below.

1. First of all, execute the *iperf*'s server command in the client. It will be the destination of the transmission (assuming the same “download scenario” from the previous test).

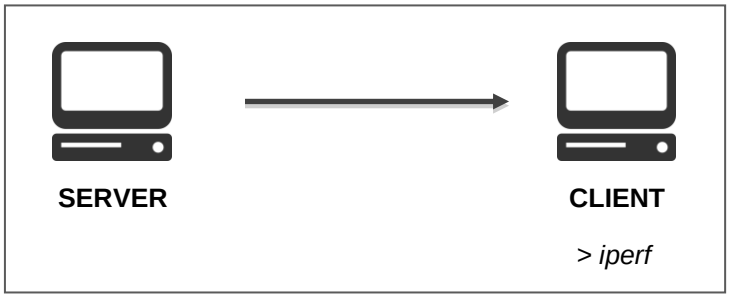


Fig. I.25 Run *iperf* command (as a server) in client side

The full *iperf* command is the following:

Iperf –p 8888 -s

With it, we can start the transmission from the server side (start the download) to a specific port (8888) in the client.

2. To start the download from the server, the following *iperf* command is used:

iperf -f m -t 30 -i 1 -p 8888 -c 192.168.1.103

- *-f m:* bw numbers format (Mbit/s)
- *-t 30:* duration of the test
- *-i 1:* bw reports print every 1 second

- *-p 8888:* specifying destination port (8888)
- *-c 192.168.1.103*: destination client's IP address

In order to get results regarding on the duration of the transmission and the evolution of the download with MPTCP protocol activated, the test are done during 15 sec., 30 sec., 60 sec., 90 sec. and 120 sec. It only needs to change the "*-t 30*" option in the *iperf* command.

3. In parallel with step 2, it will be necessary to record all the transmission data in a file, in order to analyse it after the tests.

   To do this, the software used is *tcpdump*. It allows us to capture the traffic sent by the *iperf* command and store it in a dump file ("*.dmp*" format). The command to execute it is:

   tcpdump -i any -w file.dmp

   With this, we can keep all the data received in all the interfaces ("*-i any*" option) in the file "*file.dmp*".

4. Once the transmission between server and client is finished, and we have all the data saved in the dump file defined in *tcpdump* command, it's time to analyse the main network parameters.

   The *tcptrace* software helps us to do it. It will be the responsible for adding the detailed trace packets recovered from all interfaces in the previous steps and generating summaries of connection and plot files.

   The executed command shows like this:

   tcptrace -xtraffic" -B -R -T" file.dmp

   And it is able to generate plot files from the dump file with all the stored data and regarding on the bandwidth (*-B* option), the RTT (*-R* option) and the throughput (*-T* option). The plot files generated are:

   - *traffic_bytes.xpl*: throughput over time
   - *traffic_data.xpl*: bandwidth over time
   - *traffic_rtt.xpl*: RTT over time

5. Finally, once the plot files are generated, the *xplot* software allows us to print the graphs showing the mentioned network parameters.

   xplot (traffic_bytes.xpl || *traffic_rtt.xpl* || *traffic_data.xpl)*

As mentioned in the second step, this test was done with different transmission times. Then the resulting graphs are shown for each case (15s, 30s, 60s, 90s, 120s).

- **Transmission time:** 15 seconds
  - **Command:** *iperf -f m -t 15 -i 1 -p 8888 -c 192.168.1.103*

Throughput

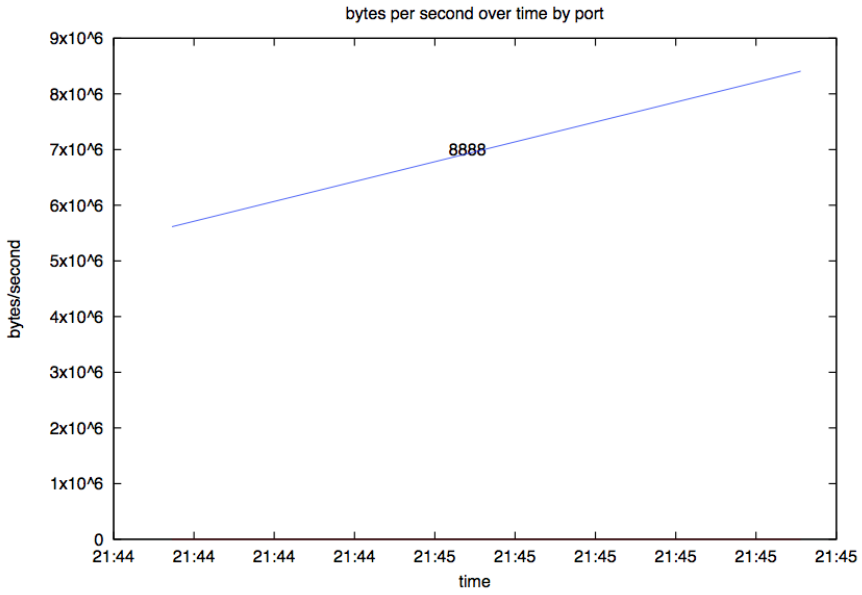


Fig. I.26 Throughput graph for 15s transmission

Bandwidth

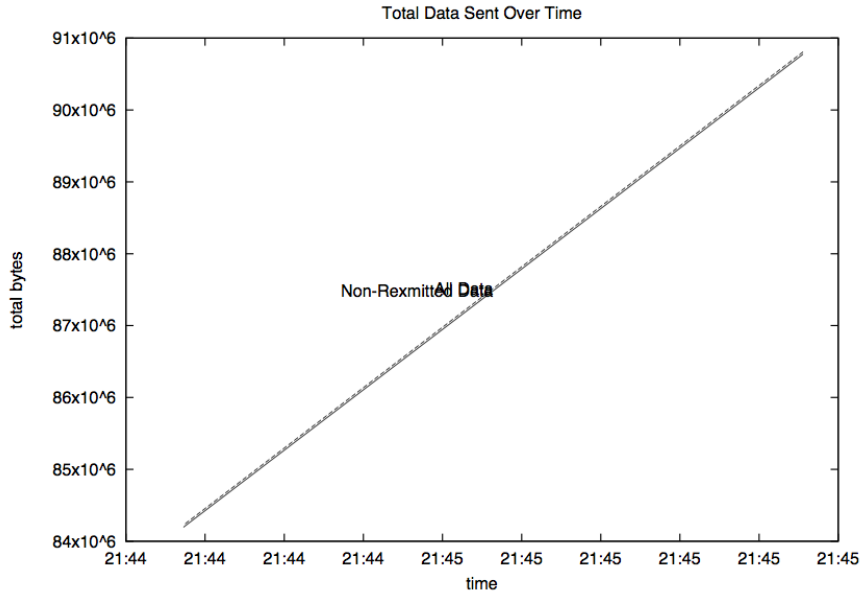


Fig. I.27 Bandwidth graph (slope) for 15s transmission

RTT

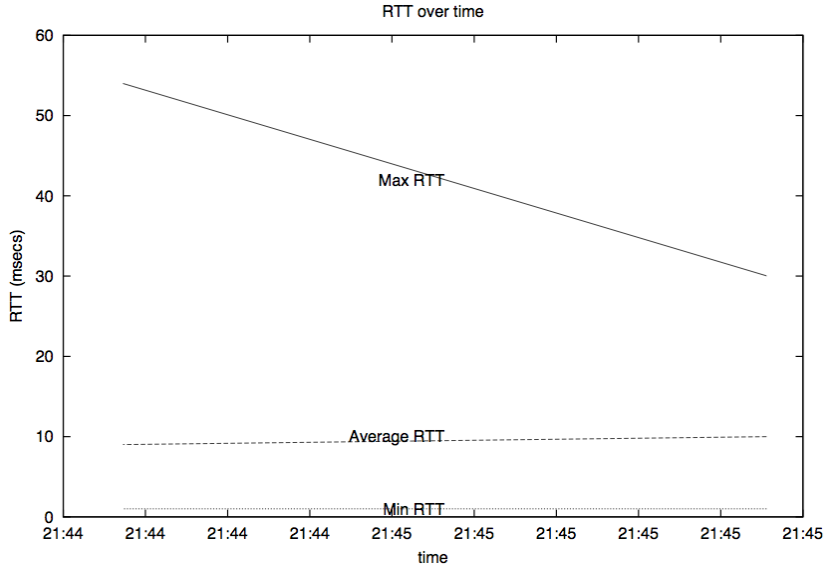


Fig. I.28 RTT graph for 15s transmission

- **Transmission time:** 30 seconds
  - **Command:** *iperf -f m -t 30 -i 1 -p 8888 -c 192.168.1.103*

Throughput

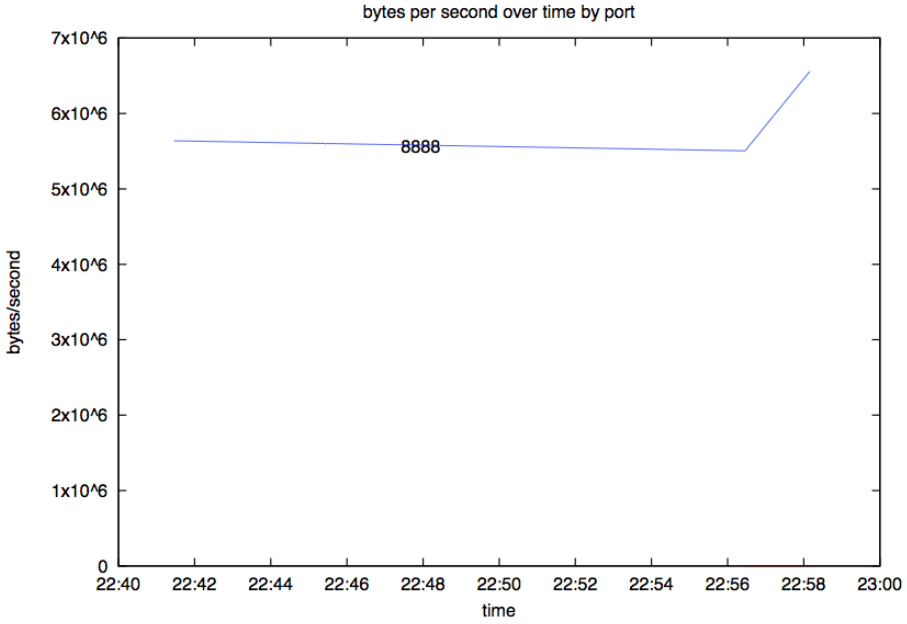


Fig. I.29 Throughput graph for 30s transmission

Bandwidth

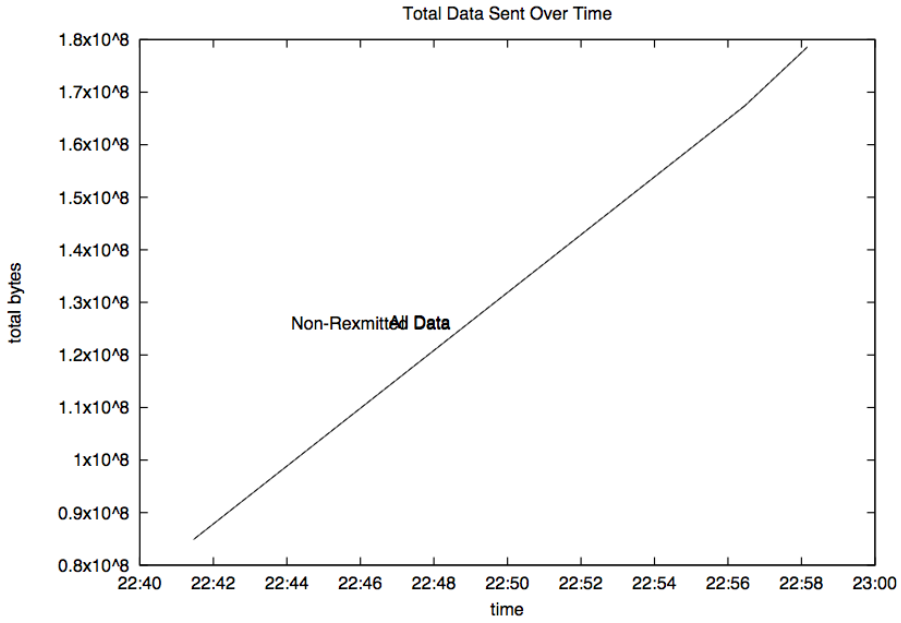


Fig. I.30 Bandwidth graph (slope) for 30s transmission

RTT

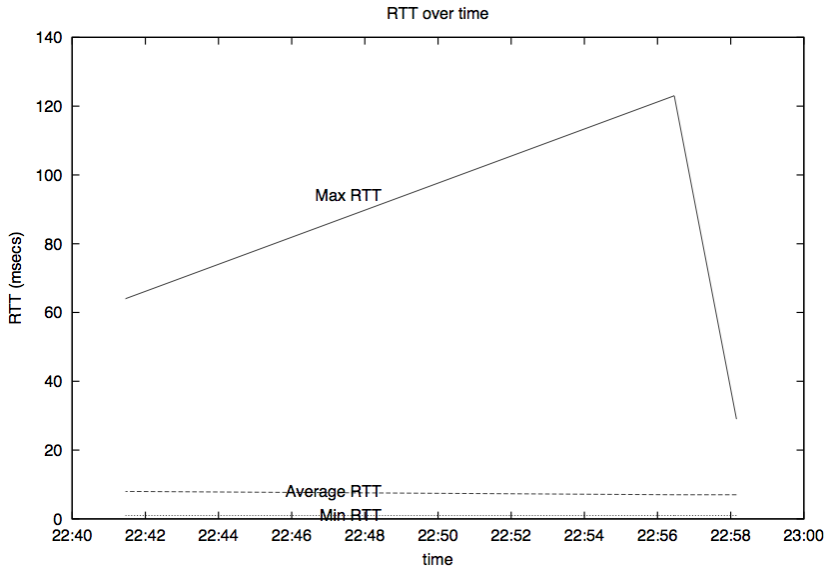


Fig. I.31 RTT graph for 30s transmission

- **Transmission time:** 60 seconds
  - **Command:** *iperf -f m -t 60 -i 1 -p 8888 -c 192.168.1.103*

Throughput

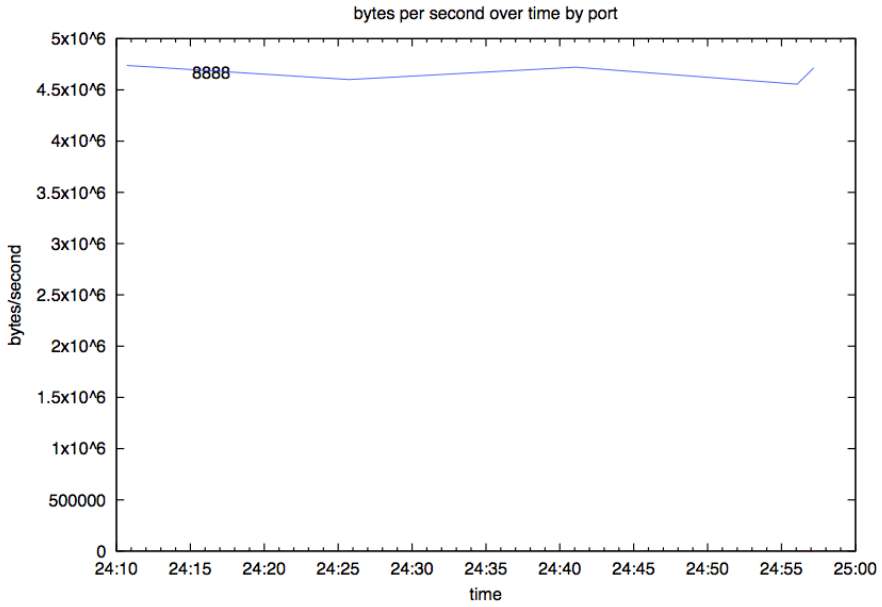


Fig. I.32 Throughput graph for 60s transmission

Bandwidth

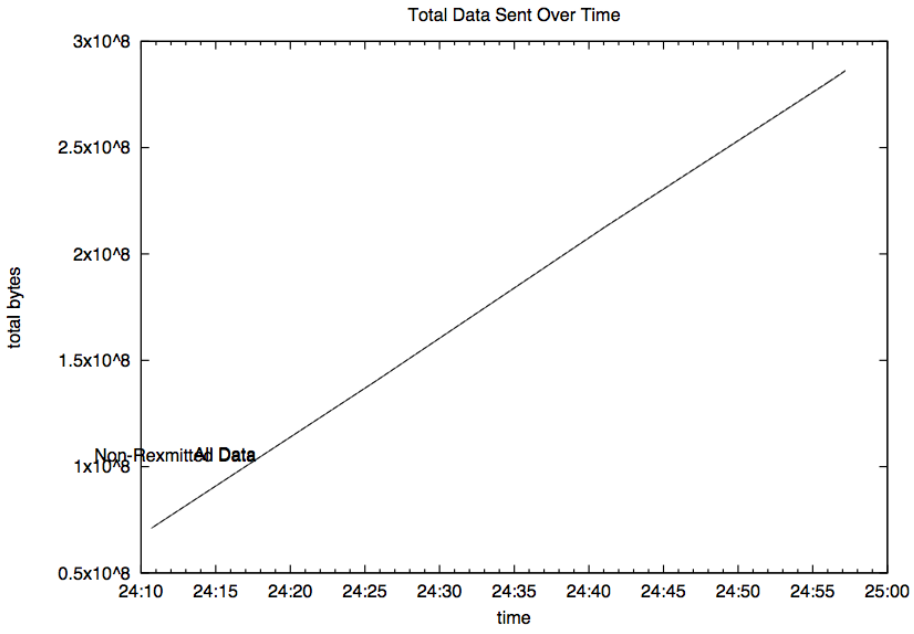


Fig. I.33 Bandwidth graph (slope) for 60s transmission

RTT

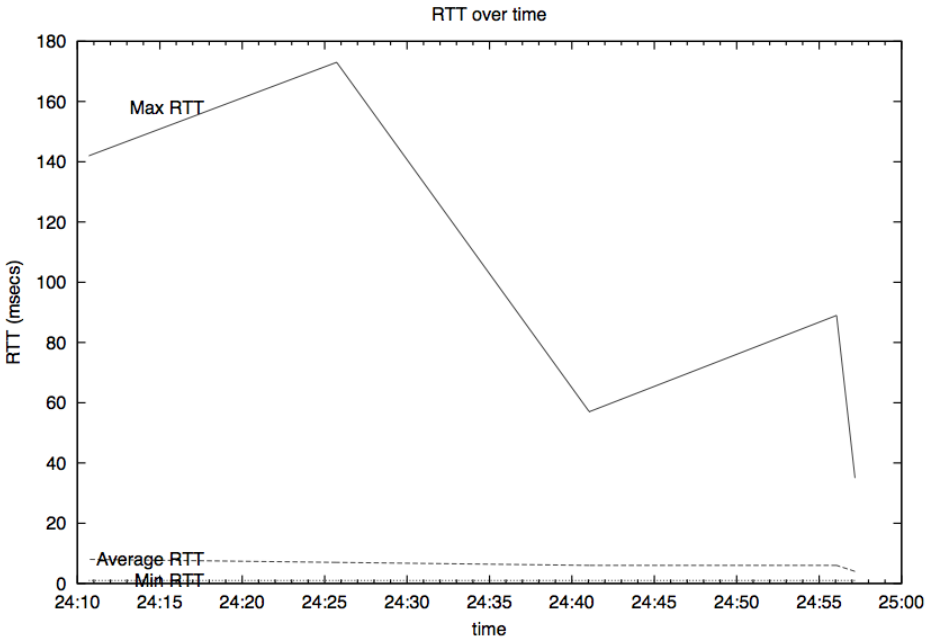


Fig. I.34 RTT graph for 60s transmission

- **Transmission time:** 90 seconds
  - **Command:** *iperf -f m -t 90 -i 1 -p 8888 -c 192.168.1.103*

Throughput

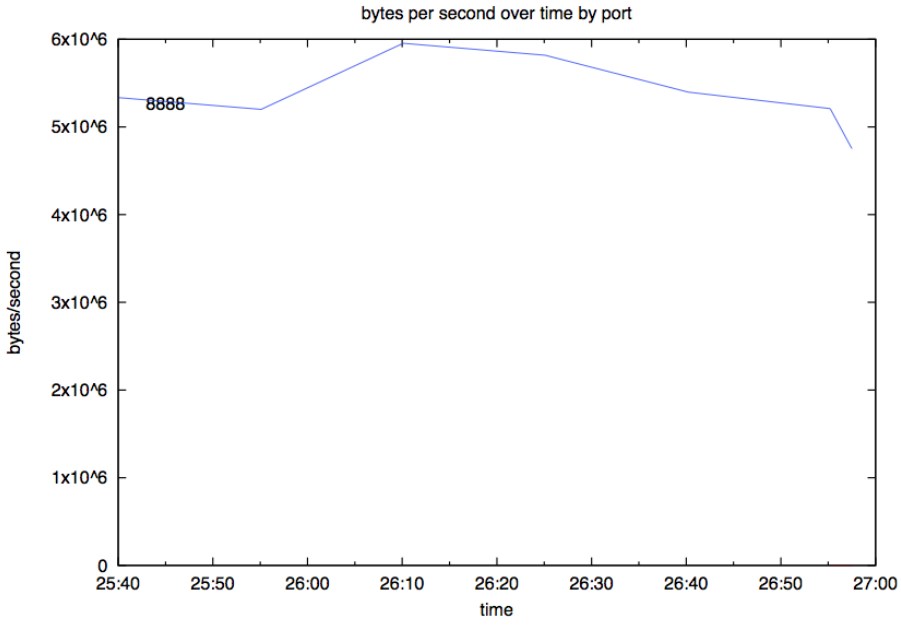


Fig. I.35 Throughput graph for 90s transmission

Bandwidth

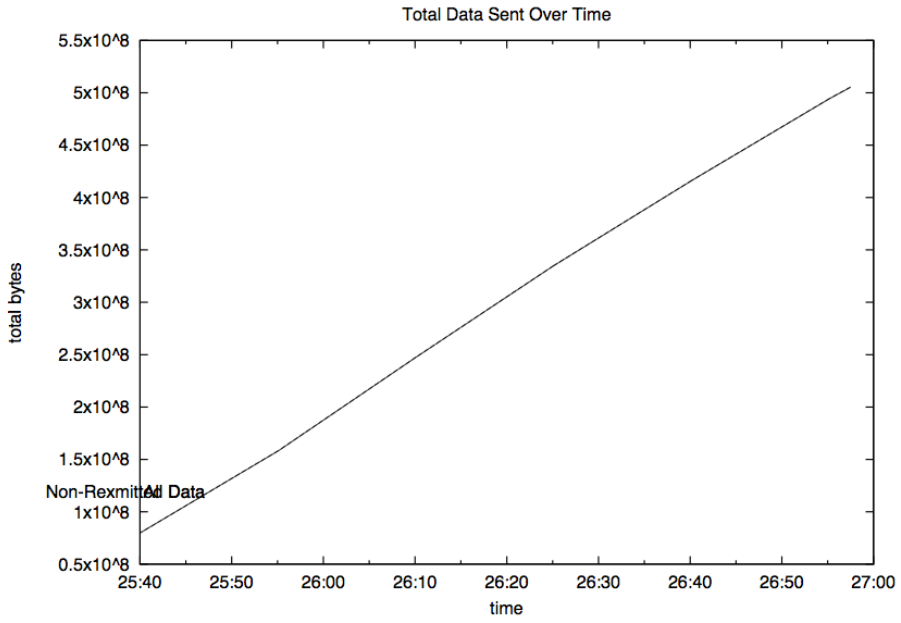


Fig. I.36 Bandwidth graph (slope) for 90s transmission

RTT

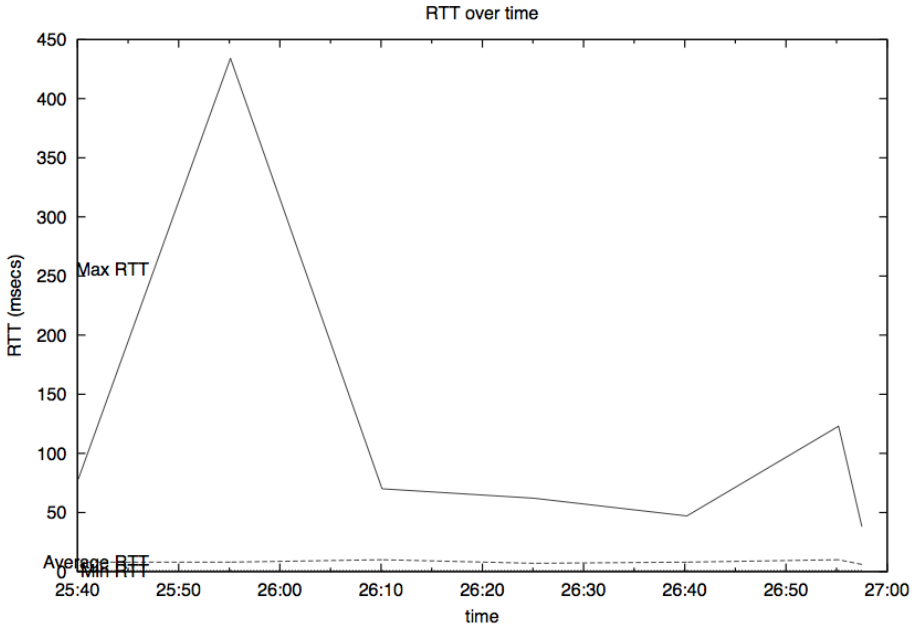


Fig. I.37 RTT graph for 90s transmission

- **Transmission time:** 120 seconds
  - **Command:** *iperf -f m -t 120 -i 1 -p 8888 -c 192.168.1.103*

Throughput

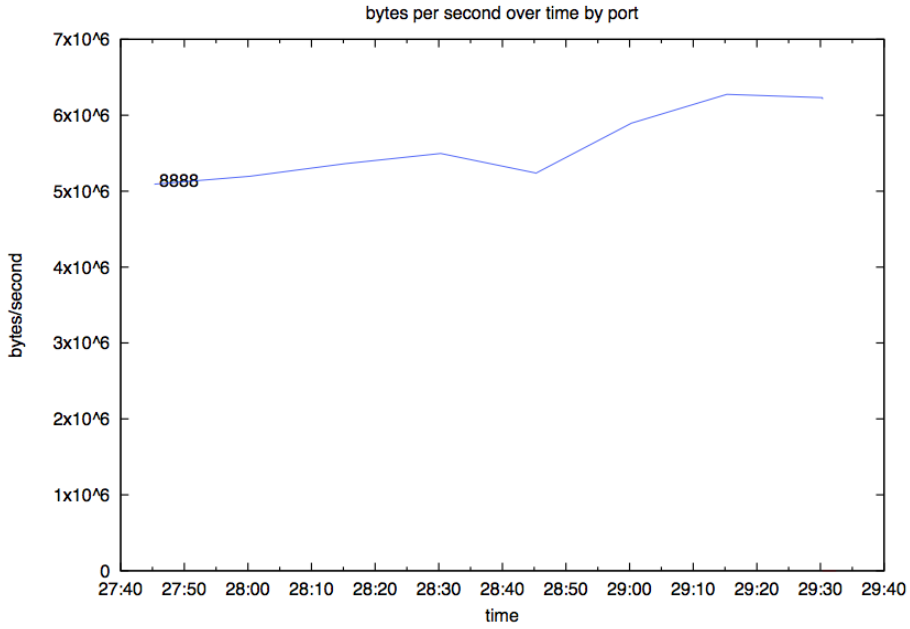


Fig. I.38 Throughput graph for 120s transmission

Bandwidth

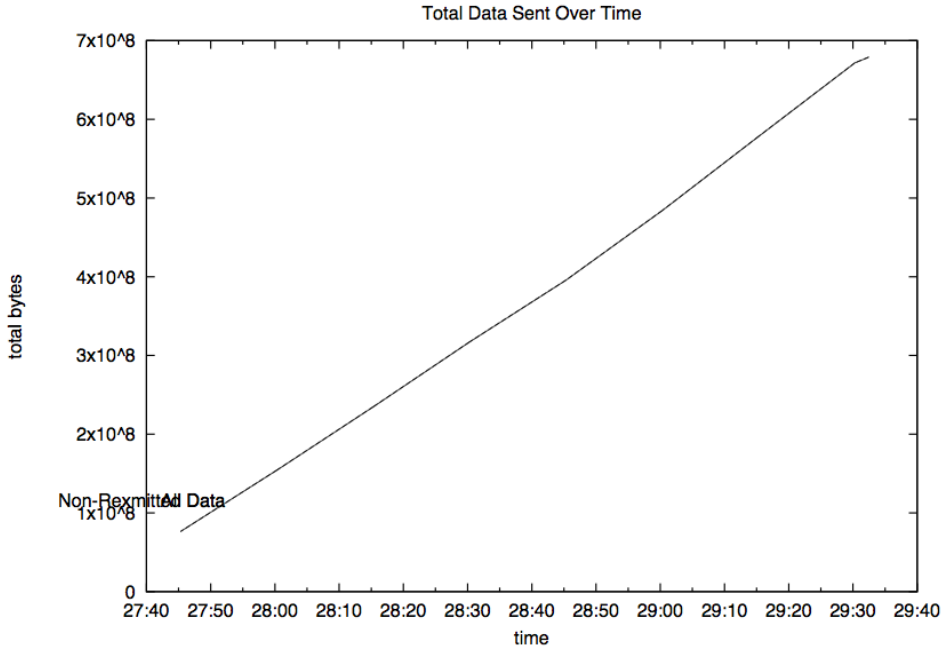


Fig. I.39 Bandwidth graph (slope) for 120s transmission

RTT

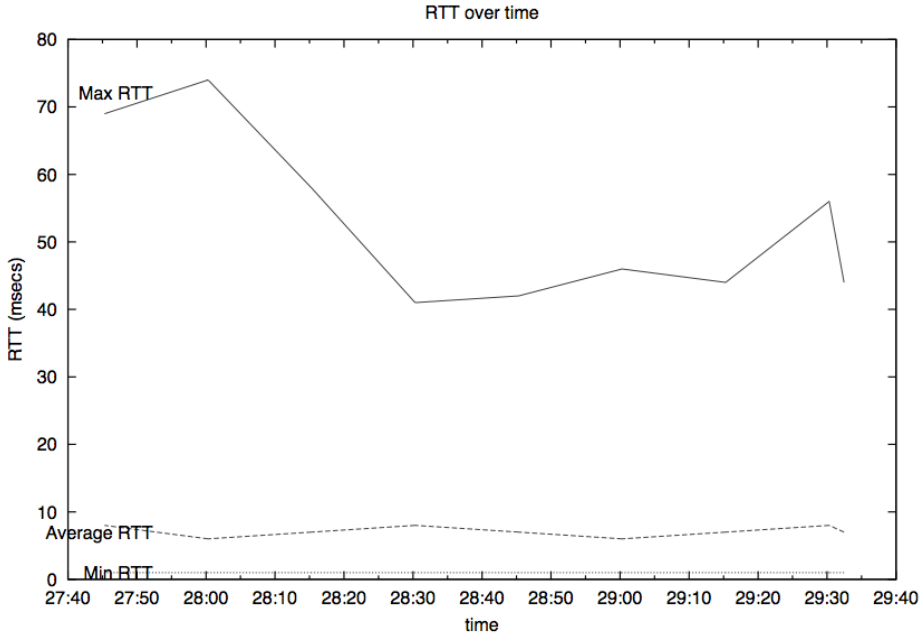


Fig. I.40 RTT graph for 120s transmission

Analysing the graphs we can see that for more duration of the transmission between client and server the throughput and the bandwidth increases.

With the throughput we can see that for higher transmission times, it slows down in the middle of the transmission but as the time increases the slope of the curve exponentially increases too.

For the case of the bandwidth and smaller transmission times, we can observe a perfect linear slope in the relation between received bytes and spent time. But as time increases, the slope change its shape and exponentially increases too.

Regarding on the RTT we can see that this parameter remains constant in all the transmission times. Although we observe some abnormal fluctuations, as for example in the case of 90 seconds' transmission. This is due to anomalous effects in the test bed environment. In the graphs, three different lines are shown: the solid one shows the maximum RTT, the dashed one the average RTT and the dotted one the minimum RTT.

I.2 Mobile IP

I.2.1 MIPv6 test bed – Installation and configuration

For the implementation of UMIP (MIPv6 and NEMO for Linux [35]) it has been used an Ubuntu Linux distribution based on a 3.8.2 patched kernel.

The first step is to compile the appropriated kernel in order to get MIPv6 running by rebuilding kernel modules and activating those that are necessary to enable

the mobility features. The following options must be turned on in the kernel configuration menu.

```
General setup
--> Prompt for development and/or incomplete code/drivers
[CONFIG_EXPERIMENTAL]
--> System V IPC [CONFIG_SYSVIPC]

Networking support [CONFIG_NET]
--> Networking options
    --> Transformation user configuration interface
[CONFIG_XFRM_USER]
    --> Transformation sub policy support [CONFIG_XFRM_SUB_POLICY]
    --> Transformation migrate database [CONFIG_XFRM_MIGRATE]
    --> PF_KEY sockets [CONFIG_NET_KEY]
    --> PF_KEY MIGRATE [CONFIG_NET_KEY_MIGRATE]
    --> TCP/IP networking [CONFIG_INET]
    --> The IPv6 protocol [CONFIG_IPV6]
        --> IPv6: AH transformation [CONFIG_INET6_AH]
        --> IPv6: ESP transformation [CONFIG_INET6_ESP]
        --> IPv6: IPComp transformation [CONFIG_INET6_IPCOMP]
        --> IPv6: Mobility [CONFIG_IPV6_MIP6]
        --> IPv6: IPsec transport mode
[CONFIG_INET6_XFRM_MODE_TRANSPORT]
        --> IPv6: IPsec tunnel mode [CONFIG_INET6_XFRM_MODE_TUNNEL]
        --> IPv6: MIPv6 route optimization mode
[CONFIG_INET6_XFRM_MODE_ROUTEOPTIMIZATION]
        --> IPv6: IP-in-IPv6 tunnel (RFC2473) [CONFIG_IPV6_TUNNEL]
        --> IPv6: Multiple Routing Tables
[CONFIG_IPV6_MULTIPLE_TABLES]
        --> IPv6: source address based routing
[CONFIG_IPV6_SUBTREES]

File systems
--> Pseudo filesystems
    --> /proc file system support [CONFIG PROC FS]
```

Fig. I.41 Enabling mobility options in the kernel

With this, the new kernel could be installed by doing the following *make* [36] commands.

```
# make
# make modules_install
# make install
# make headers install
```

Fig. I.42 Installing the patched kernel

Once the kernel is installed and running on the Linux distribution, the last step to get UMIP is to download the source code and compile it with the next commands.

```
$ cd /usr/src/
$ git clone git://git.umip.org/umip.git
$ cd umip/
```

Fig. I.43 Downloading the source code from UMIP repository

```
$ autoreconf -i
$    CPPFLAGS='-isystem    /usr/src/linux/usr/include/'
./configure --enable-vt
$ make
# make install
```

Fig. I.44 Compiling and installing UMIP

The option "*--enable-vt*" enables a virtual terminal in order to virtualize the Binding Cache table or the Binding Update list in the Home Agent (HA) or the Mobile Node (MN).

With all the steps detailed above, UMIP is correctly installed in our Linux distribution. Now, the next step is to configure it in our scenario with one machine acting as the Home Agent (HA) and another one acting as a Mobile Node (MN).

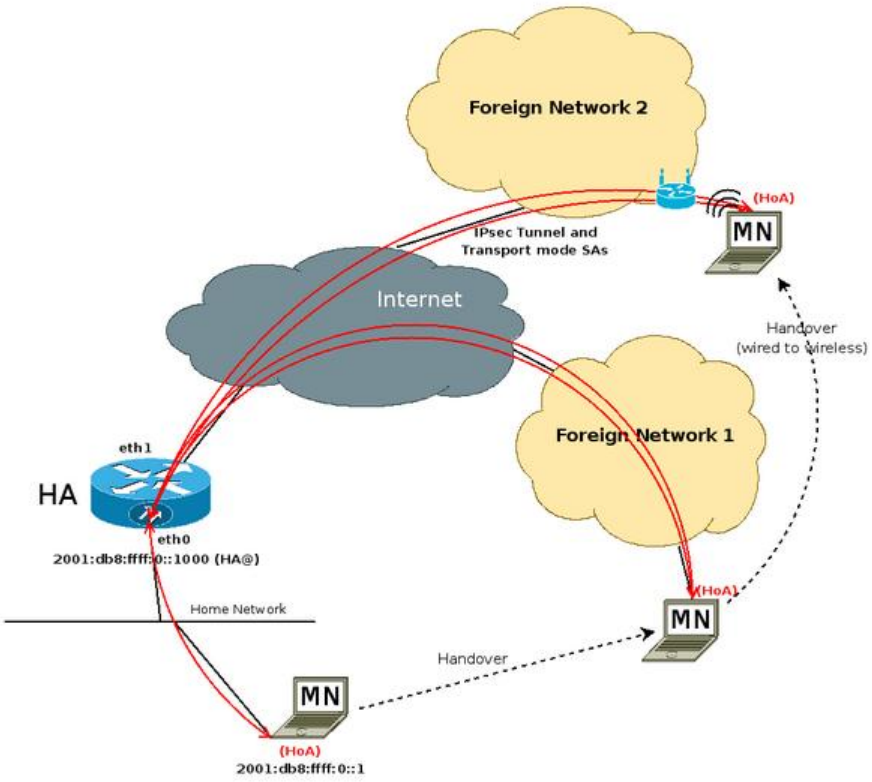


Fig. I.45 MIPv6 Test Bed setup

Both HA and MN are based on the same Linux distribution mounted in two different physical machines and with the following setup:

- HA:
  - Connected to the Internet by eth1 interface.
  - The interface eth0 is the one connected to the home link of the MN.
  - The address configured on eth0 is 2001:db8:ffff:0::1000/64.

- MN:
  - The Home Address (HoA) of the MN is 2001:db8:ffff:0::1/64 (configured on interface eth0).

Regarding to this specifications, the configuration files for each node are detailed below (*mip6d.conf* file in both machines).

Home Agent

```
# UMIP configuration file for a MIPv6 Home Agent
NodeConfig HA;

# Set DebugLevel to 0 if you do not want debug messages
DebugLevel 10;

# Replace eth0 with the interface connected to the home link
Interface "eth0";

# Binding information
BindingAclPolicy 2001:db8:ffff:0::1 allow;
DefaultBindingAclPolicy deny;

# Enable IPsec static keying
UseMnHaIPsec enabled;
KeyMngMobCapability disabled;

# IPsec Security Policies information
IPsecPolicySet {
    HomeAgentAddress 2001:db8:ffff:0::1000;
    HomeAddress 2001:db8:ffff:0::1/64;

    # All MH packets (BU/BA/BERR)
    IPsecPolicy Mh UseESP 11 12;
    # All tunneled packets (HoTI/HoT, payload)
    IPsecPolicy TunnelPayload UseESP 13 14;
    # All ICMP packets (MPS/MPA, ICMPv6)
    IPsecPolicy ICMP UseESP 15 16;
}
```

Fig. I.46 UMIP Home Agent configuration file

Mobile Node

```
# UMIP configuration file for a MIPv6 Mobile Node
NodeConfig MN;

# Set DebugLevel to 0 if you do not want debug messages
DebugLevel 10;

# Enable the optimistic handovers
OptimisticHandoff enabled;

# Disable RO with other MNs (it is not compatible
# with IPsec Tunnel Payload)
DoRouteOptimizationMN disabled;

# The Binding Lifetime (in sec.)
MnMaxHaBindingLife 60;

# List here the interfaces that you will use
# on your mobile node. The available one with
# the smallest preference number will be used.
Interface "eth0" {
    MnIfPreference 1;
}
Interface "wlan0" {
    MnIfPreference 2;
}

# Replace eth0 with one of your interface used on
# your mobile node
MnHomeLink "eth0" {
    HomeAgentAddress 2001:db8:ffff:0::1000;
    HomeAddress 2001:db8:ffff:0::1/64;
}

# Enable IPsec static keying
UseMnHaIPsec enabled;
KeyMngMobCapability disabled;

# IPsec Security Policies information
IPsecPolicySet {
    HomeAgentAddress 2001:db8:ffff:0::1000;
    HomeAddress 2001:db8:ffff:0::1/64;

    # All MH packets (BU/BA/BERR)
    IPsecPolicy Mh UseESP 11 12;
    # All tunneled packets (HoTI/HoT, payload)
    IPsecPolicy TunnelPayload UseESP 13 14;
    # All ICMP packets (MPS/MPA, ICMPv6)
    IPsecPolicy ICMP UseESP 15 16;
}
```

Fig. I.47 UMIP Mobile Node configuration file

The **IPsetPolicySet** configured in both scripts (HA & MN) uses the below IPsec SAs (*setkey.conf* in both machines).

```
# IPsec Security Associations
# HA address: 2001:db8:ffff:0::1000;
# MR HoAs:    2001:db8:ffff:0::1/64;

# Flush the SAD and SPD
flush;
spdflush;

# MN1 -> HA transport SA for BU
add 2001:db8:ffff:0::1 2001:db8:ffff:0::1000 esp 0x11
    -u 11
    -m transport
    -E 3des-cbc "MIP6-011--12345678901234"
    -A hmac-sha1 "MIP6-011--1234567890" ;

# HA -> MN1 transport SA for BA
add 2001:db8:ffff:0::1000 2001:db8:ffff:0::1 esp 0x12
    -u 12
    -m transport
    -E 3des-cbc "MIP6-012--12345678901234"
    -A hmac-sha1 "MIP6-012--1234567890" ;

# MN1 -> HA tunnel SA for any traffic
add 2001:db8:ffff:0::1 2001:db8:ffff:0::1000 esp 0x13
    -u 13
    -m tunnel
    -E 3des-cbc "MIP6-013--12345678901234"
    -A hmac-sha1 "MIP6-013--1234567890" ;

# HA -> MN1 tunnel SA for any traffic
add 2001:db8:ffff:0::1000 2001:db8:ffff:0::1 esp 0x14
    -u 14
    -m tunnel
    -E 3des-cbc "MIP6-014--12345678901234"
    -A hmac-sha1 "MIP6-014--1234567890" ;

# MN1 -> HA transport SA for ICMP (including MPS/MPA)
add 2001:db8:ffff:0::1 2001:db8:ffff:0::1000 esp 0x15
    -u 15
    -m transport
    -E 3des-cbc "MIP6-015--12345678901234"
    -A hmac-sha1 "MIP6-015--1234567890" ;

# HA -> MN1 transport SA for ICMP (including MPS/MPA)
add 2001:db8:ffff:0::1000 2001:db8:ffff:0::1 esp 0x16
    -u 16
    -m transport
    -E 3des-cbc "MIP6-016--12345678901234"
    -A hmac-sha1 "MIP6-016--1234567890" ;
```

Fig. I.48 IPsecSAs configuration file on both machines

The Home Agent also needs to advertise the Home Link prefix in its Home Link using Router Advertisements. For that purpose, **radvd** software [37] is used with the below configuration (*radvd.conf* file in HA).

```
# Home Agent radvd configuration file
# Replace eth0 with the interface connected to the home link
interface eth0
{
    AdvSendAdvert on;
    MaxRtrAdvInterval 3;
    MinRtrAdvInterval 1;
    AdvIntervalOpt on;
    AdvHomeAgentFlag on;
    AdvHomeAgentInfo on;
    HomeAgentLifetime 1800;
    HomeAgentPreference 10;

    # Home Agent address
    prefix 2001:db8:ffff:0::1000/64
    {
        AdvRouterAddr on;
        AdvOnLink on;
        AdvAutonomous on;
    };
};
```

Fig. I.48 Radvd software configuration file

Finally, with all these files configured in each machine, we can execute the following scripts in order to get UMIP running in our test bed scenario (with a HA and a MN).

In the HA (*mipv6-ha.sh*)

```
#!/bin/sh
set -e

#
# Make sure below as your environment
#
PATH=/usr/local/sbin:/usr/sbin:/sbin:${PATH}
mip6d_conf=/usr/local/etc/mip6d-ha.conf
setkey_conf=/usr/local/etc/setkey.conf
radvd_conf=/usr/local/etc/radvd.conf


case "$1" in
    start)
      # Router
      echo 1 >/proc/sys/net/ipv6/conf/all/forwarding
      # Proxy ND
      echo 1 >/proc/sys/net/ipv6/conf/all/proxy_ndp

      # set IPsec configuration
      setkey -f $setkey_conf

      # invoke mip6d
      mip6d -c $mip6d_conf

      # For low performance systems, you can add sleep here
      # to make sure mip6d has started
      #

      # RD
      radvd -C $radvd_conf

        echo "Starting MIPv6 HA."
      ;;
    stop)
      killall radvd
      killall mip6d

      # XXX: Deleting IPsec SAs can be here if you want.
      #

      # XXX: Disabling router and/or Proxy ND can be here if you
want.
      #

        echo "Stopping MIPv6 HA."
      ;;
    *)
      echo "Usage: ${0##*/} {start|stop}"
      exit 1
      ;;
esac

exit 0
```

Fig. I.49 Home Agent UMIP starting script

In the MN (*mipv6-mn.sh*)

```
#!/bin/sh
set -e

#
# Make sure below as your environment
#
PATH=${PATH:+$PATH:}/usr/local/sbin:/usr/sbin:/sbin
mip6d_conf=/usr/local/etc/mip6d-mn.conf
setkey_conf=/usr/local/etc/setkey.conf
#device used by mip6d.
dev=eth0

case "$1" in
    start)
      # disable to send RS from kernel.
      #echo 0 > /proc/sys/net/ipv6/conf/all/router_solicitations
      #echo 0 > /proc/sys/net/ipv6/conf/default/router_solicitations
      #echo                                0                                >
/proc/sys/net/ipv6/conf/all/router_solicitation_interval
      #echo                                0                                >
/proc/sys/net/ipv6/conf/default/router_solicitation_interval

      # set IPsec configuration
      setkey -f $setkey_conf

      # bring up the interface which MN will use.
      ifconfig $DEV up

      # invoke mip6d
      mip6d -c $mip6d_conf
        echo "Starting MIPv6 MN."
      ;;
    stop)
      killall mip6d

      ifconfig $DEV down

      # XXX: Deleting IPsec SAs can be here if you want.
      #

      # XXX: enable to send RS from kernel here if you want.
      #

        echo "Stopping MIPv6 MN."
      ;;

    *)
      echo "Usage: ${0##*/} {start|stop}"
      exit 1
      ;;
esac

exit 0
```

Fig. I.50 Mobile Node UMIP starting script

I.2.2 MIPv6 test bed – Checking connectivity

The first thing done after booting the protocol on both machines, is to test if the MN is reachable from the HA when it is in the Home Link, for example with ping6 [38].

After this, the next step is to try to move the MN from the Home Link to a foreign Network. In that case it has been used a mobile phone in its network sharing mode, acting as an AP for the machine configured as MN. Also with that, the MN is still reachable because it was registered to the HA.

We can check that the HA is ready to serve by calling "*pl*" command in its Virtual Terminal and getting a home prefix valid lifetime with positive value.

```
# telnet localhost 7777
mip6d> verbose yes
yes
mip6d> pl
eth0 2001:db8:ffff:0::1000/64
valid 467897 / 467900 preferred 42800 flags OAR
mip6d>
```

Fig. I.51 Checking if HA is ready to serve

This registration can be checked in the Binding Update List on the MN and in the Binding Cache on the HA. It can be done accessing to the Virtual Terminal previously configured on UMIP in the installation stage.

- Checking Binding Cache on the HA:

```
# telnet localhost 7777
mip6d> verbose yes
yes
mip6d> bc
hoa 2001:db8:ffff:0::1 status registered
  coa 2001:db8:ffff:f300:feed:beef:feed:beef flags AH--
  local 2001:db8:ffff:0::1000
  lifetime 965 / 1000 seq 54192 unreach 0 mpa - / 0 retry 0
```

Fig. I.52 Binding Cache on the HA

- Checking Binding Update List on the MN:

```
# telnet localhost 7777
mip6d> verbose yes
yes
mip6d> bul
== BUL_ENTRY ==
Home address    2001:db8:ffff:0::1
Care-of address 2001:db8:ffff:f300:feed:beef:feed:beef
CN address      2001:db8:ffff:0::1000
 lifetime = 8,  delay = 7000
 flags: IP6_MH_BU_HOME IP6_MH_BU_ACK
 ack ready
 dev eth0 last_coa 2001:db8:ffff:f300:feed:beef:feed:beef
 lifetime 4 / 8 seq 71508 resend 0 delay 4(after 1s) expires 3
       mps 2482411 / 2482469
```

Fig. I.53 Binding Update List on the MN

In both validations ("*bc*" command on the HA and "*bul*" command on the MN) we can observe that the CoA (*Care-of Address: 2001:db8:ffff:f300:feed:beef:feed:beef*) which is bound to the Home Address *2001:db8:ffff:0::1* is registered to the Correspondent Node (in that case, the HA) whose address is *2001:db8:ffff:0::1000*.

I.2.3 Adding NEMO Basic Support

Once we have the MIPv6 test bed running there are some modifications to be done in order to turn the HA into a NEMO HA and the MN into a Mobile Router (MR).

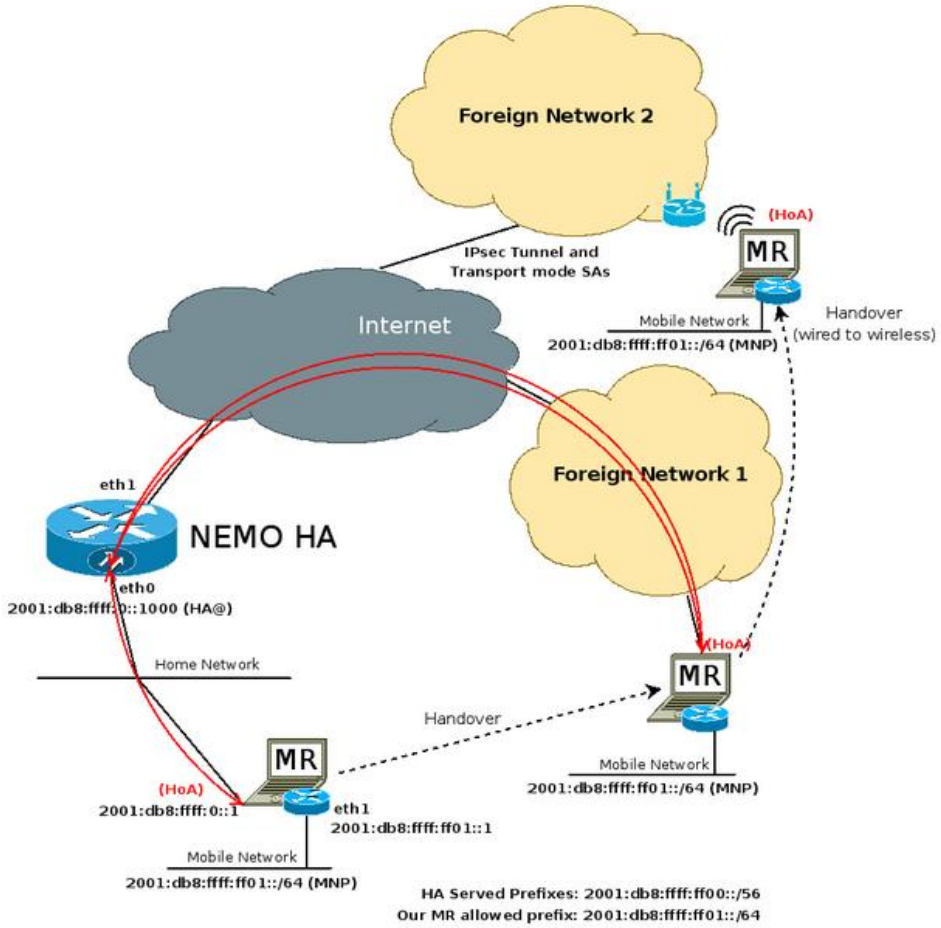


Fig. I.54 NEMO Test Bed setup

As shown in figure 1.54, the mobile network is connected to the eth1 interface of the MR, and also the MR advertises a prefix (called *Mobile Network Prefix*, MNP) in his mobile network. In the test bed, the MNP is *2001:db8:ffff:ff01::/64* and the MR also configures the address *2001:db8:ffff:ff01::1* on its ingress interface (eth1).

The MNP used by the MR must be routable towards the HA, and for that purpose the address space *2001:db8:ffff:ff00::/56* dedicated to a set of MNP is assigned to the HA. This prefix also must be advertised by the HA in the routing infrastructure.

The small modifications to be done in both machines HA and MN are the detailed below.

NEMO Home Agent

- Modify the mip6d running script detailed in MIPv6 test bed configuration by adding the following lines (marked as blue).

```
# UMIP configuration file for a NEMO Home Agent
NodeConfig HA;

# Set DebugLevel to 0 if you do not want debug messages
DebugLevel 10;

# Replace eth0 with the interface connected to the home link
Interface "eth0";

# Accept registrations from Mobile Routers
HaAcceptMobRtr enabled;
HaServedPrefix 2001:db8:ffff:0::/64;

# Binding information
BindingAclPolicy 2001:db8:ffff:0::1 (2001:db8:ffff:ff01::/64) allow;
DefaultBindingAclPolicy deny;

# Enable IPsec static keying
UseMnHaIPsec enabled;
KeyMngMobCapability disabled;

# IPsec Security Policies information
IPsecPolicySet {
    HomeAgentAddress 2001:db8:ffff:0::1000;
    HomeAddress 2001:db8:ffff:0::1/64;

    # All MH packets (BU/BA/BERR)
    IPsecPolicy Mh UseESP 11 12;
    # All tunneled packets (HoTI/HoT, payload)
    IPsecPolicy TunnelPayload UseESP 13 14;
    # All ICMP packets (MPS/MPA, ICMPv6)
    IPsecPolicy ICMP UseESP 15 16;
      }
```

Fig. I.55 NEMO HA Configuration file

- Modify the *radvd.conf* file configured in the MIPv6 test bed configuration by adding the following lines (marked as blue).

```
# NEMO Home Agent radvd configuration file
# Replace eth0 with the interface connected to the home link
interface eth0
{
    AdvSendAdvert on;
    MaxRtrAdvInterval 3;
    MinRtrAdvInterval 1;
    AdvIntervalOpt on;
    AdvHomeAgentFlag on;
    AdvHomeAgentInfo on;
    HomeAgentLifetime 1800;
    HomeAgentPreference 10;

    AdvMobRtrSupportFlag on;

    # Home Agent address
    prefix 2001:db8:ffff:0::1000/64
    {
        AdvRouterAddr on;
        AdvOnLink on;
        AdvAutonomous on;
    };
};
```

Fig. I.56 NEMO HA *radvd.conf* file

NEMO Mobile Router

- Modify the mip6d running script detailed in MIPv6 test bed configuration by adding the following lines (marked as blue).

```
# UMIP configuration file for a Mobile Router
NodeConfig MN;

# Set DebugLevel to 0 if you do not want debug messages
DebugLevel 10;

# Enable the optimistic handovers
OptimisticHandoff enabled;

# Disable RO with other MNs (it is not compatible
# with IPsec Tunnel Payload)
DoRouteOptimizationMN disabled;

# The Binding Lifetime (in sec.)
MnMaxHaBindingLife 60;

# Use NEMO Explicit Mode
MobRtrUseExplicitMode enabled;

# List here the interfaces that you will use
# on your mobile node. The available one with
# the smallest preference number will be used.
Interface "eth0" {
    MnIfPreference 1;
}
Interface "wlan0" {
    MnIfPreference 2;
}

# Replace eth0 with one of your interface used on
# your mobile node
MnHomeLink "eth0" {
    IsMobRtr enabled;
    HomeAgentAddress 2001:db8:ffff:0::1000;
    HomeAddress 2001:db8:ffff:0::1/64 (2001:db8:ffff:ff01::/64);
}

# Enable IPsec static keying
UseMnHaIPsec enabled;
KeyMngMobCapability disabled;

# IPsec Security Policies information
IPsecPolicySet {
    HomeAgentAddress 2001:db8:ffff:0::1000;
    HomeAddress 2001:db8:ffff:0::1/64;

    # All MH packets (BU/BA/BERR)
    IPsecPolicy Mh UseESP 11 12;
    # All tunneled packets (HoTI/HoT, payload)
    IPsecPolicy TunnelPayload UseESP 13 14;
    # All ICMP packets (MPS/MPA, ICMPv6)
    IPsecPolicy ICMP UseESP 15 16;
}
```

Fig. I.57 NEMO MR Configuration file

- Create a *radvd.conf* file (not created in MIPv6 test bed configuration) in order to let the MR to advertise its MNP in the mobile network using Router Advertisements (RA).

```
# Mobile Router radvd configuration file
# Replace eth1 with your ingress interface name
interface eth1
{
    AdvSendAdvert on;
    MaxRtrAdvInterval 3;
    MinRtrAdvInterval 1;
    AdvIntervalOpt on;
    IgnoreIfMissing on;

    # Mobile Router address on the ingress interface
    prefix 2001:db8:ffff:ff01::1/64
    {
        AdvRouterAddr on;
        AdvOnLink on;
        AdvAutonomous on;
        AdvPreferredLifetime 60;
        AdvValidLifetime 120;
        AdvLinkMTU 1280;
    };
      };
```

Fig. I.58 MR radvd software configuration file

After doing these modifications in the HA and the MR configuration files, the correct configuration of NEMO can be checked as in the previous point 1.2.2. Connectivity tests are the same as in MIPv6 test bed scenario.

ANNEX II. OPEN DATA AND OPERATING SYSTEMS

II.1 Open data examples

II.1.1 Overview

Another important feature that the smart car gateway will give to the user is the access to some relevant information related to:

- Roads
- Routes
- Traffic
- Current City Points of Interest

For example, the user can be in touch with real-time information of traffic status and alternative routes if a traffic jam is happening. Or also can be informed of where to park the car in the city (public parking).

The main objective is to give to the user all the possible information in order to help him and optimize his routes.

Following this line, there are some public organizations offering open data. One of them is *OpenDataBCN* (http://opendata.bcn.cat/opendata/en), an open data platform powered by the Barcelona City Council. This site allow us to download and use the following information that can be oriented to the driver's service:

- Traffic status in Real-Time
- Traffic incidents
- Routes
- Parking in the City
- Public Transport
- Equipment and services in the City

Also in Barcelona, we can find additional traffic information like, for example, traffic cameras (http://www.bcn.cat/transit/en/cameres.html) that will allow the user to view the traffic status and possible traffic jams in real-time.

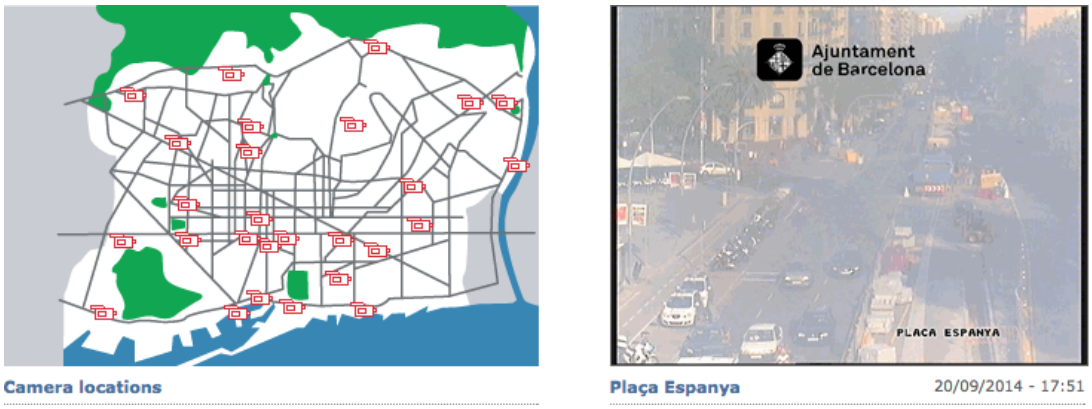


Fig. II.1 Traffic camera locations and example of one of them

The information mentioned above is also given in another open data website from the Catalan Government (http://dadesobertes.gencat.cat/), but in this case in all the Catalan territory. Also in this website, the information related to speed radars in roads and all the road maps of Catalonia can be found. In resume, the available information in this site is:

- Roads' Maps (in order to present information on them).
- Traffic Cameras in entire Catalonian roads.
- Information about traffic speed radars.
- Traffic incidents in real-time.

Another important information to give to the user is the related with gas stations and fuel prices. It can be find in another open data organism, *Datos.Gob* (http://datos.gob.es/), powered by the Spanish Government.

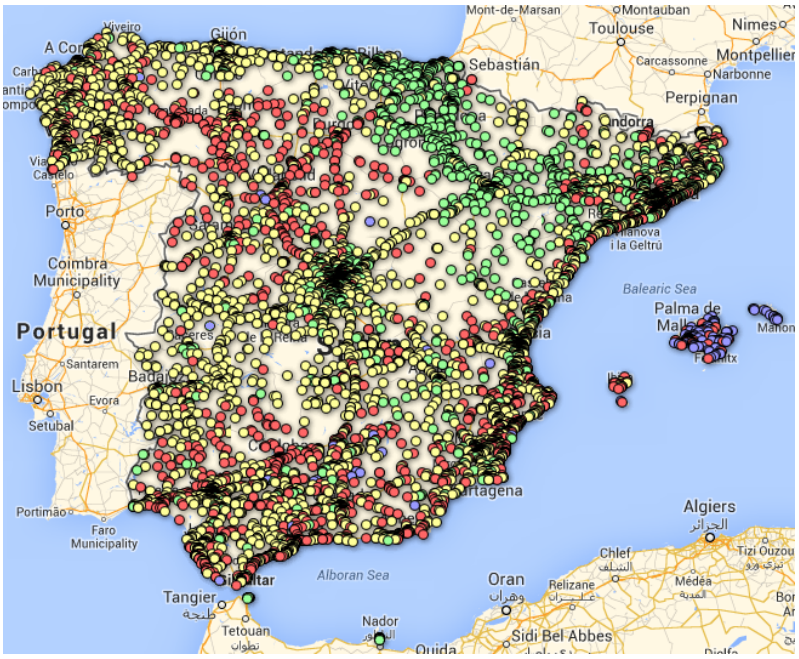


Fig. II.2 Gas Stations and Fuel Price Map

This data can allow the driver to refuel in the best gas station based on the distance to it and the fuel price.

In summary, all of this kind of information can be presented to the user as an additional functionality of the smart car gateway.

II.1.2 Open data – Spanish territory

The Spanish Government from its own open data initiative, *Datos Gob*, gives the possibility to get free information about the country, population and other many aspects.

In our case and thinking in the smart car gateway, we can use information like:

- Roads' Maps (in order to represent information on them).
- Gas Stations and Fuel Price
- Information about traffic

Gas Stations and Fuel Price

For example and regarding the information about gas stations and fuel price in each one, we can represent the data as shown below.

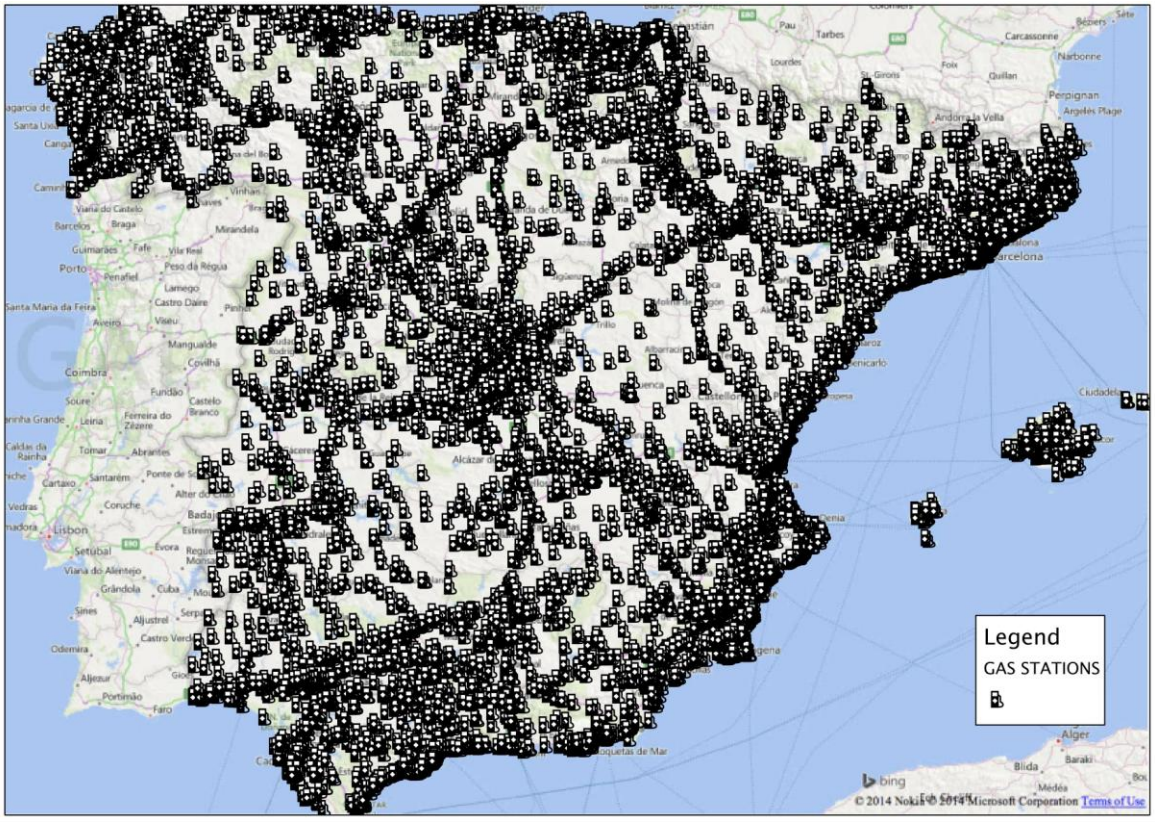


Fig. II.3 Gas Stations

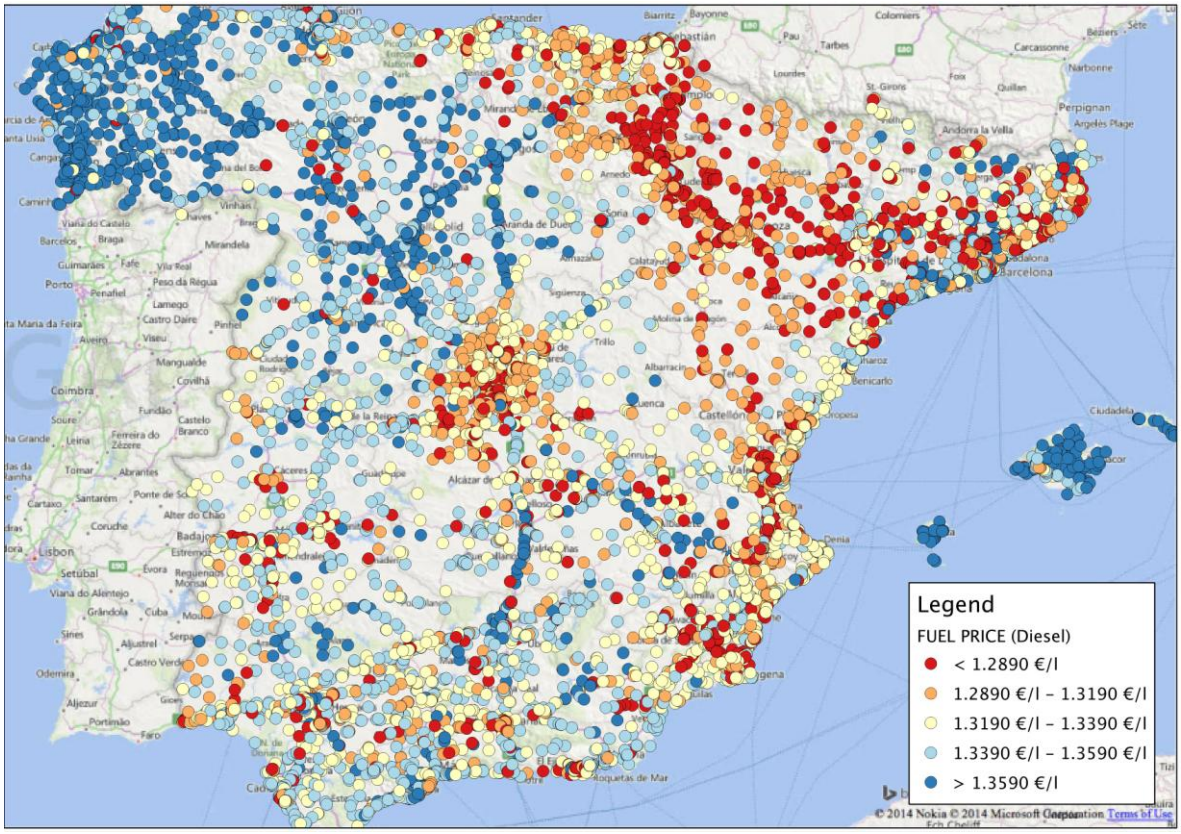


Fig. II.4 Fuel Price (Diesel)

In this case we can offer the driver the knowledge about the presence of a gas station in all the roads and the price of fuel on each one based on car's fuel type.

Roads' Map

The given information about roads in the Spanish territory is classified regarding on road type (primary and secondary). The next picture shows the entire road network in Spain.

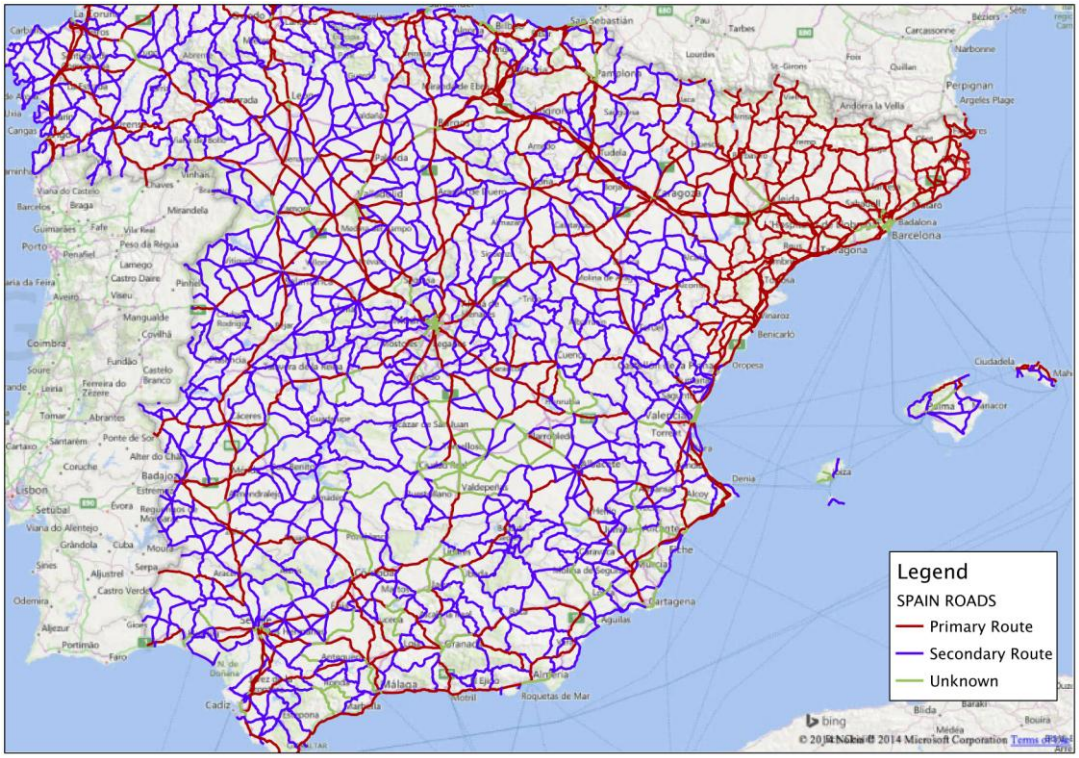


Fig. II.5 Road Network in Spain

And also this information can be combined with the previous one related to gas stations and fuel price in each one of them. The following map shows a case where a driver can decide which gas station is better regarding on fuel price.

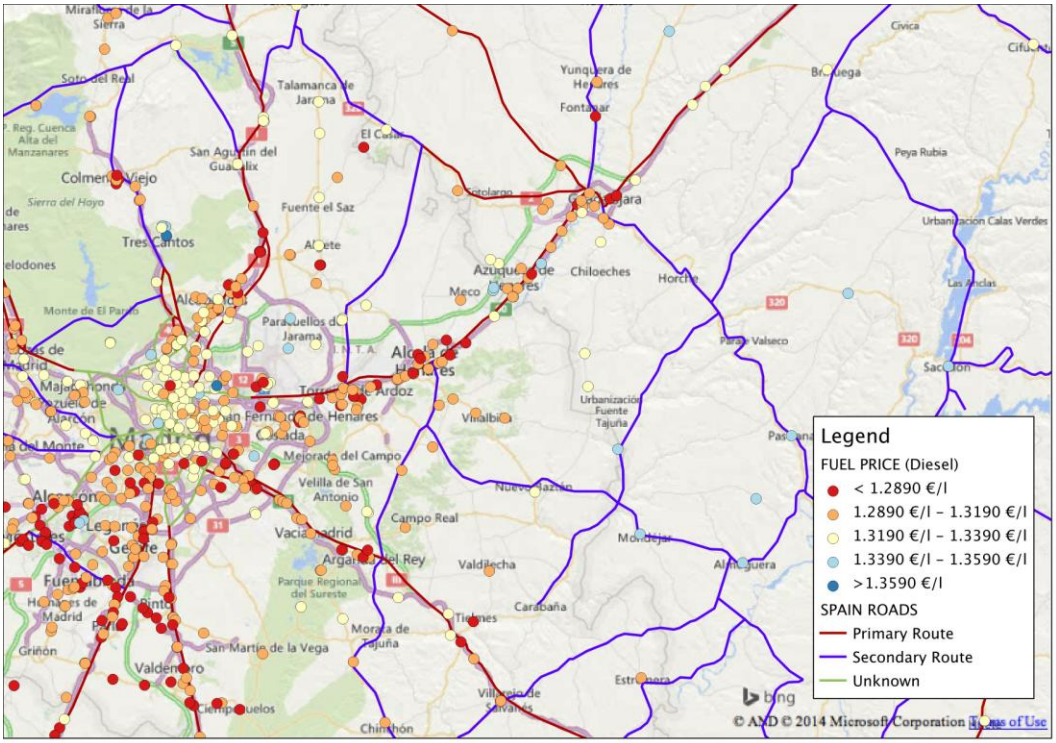


Fig. II.6 Roads and Fuel Price information

II.1.3 Open data – Catalonian territory

As mentioned previously, the Catalan Government offer an open data website with some interesting information that can be included in the smart car gateway. Resuming, the information taken in this website is:

- Roads' Maps (in order to represent information on them).
- Traffic Cameras in entire Catalonian roads.
- Information about traffic speed radars.
- Traffic incidents in real-time (traffic status).

Following an example for each case will be shown.

Roads' Map

The information is offered in *Shapefile* mode. This kind of format allows using the data in many GIS software for its editing, modelling and styling, and also with the possibility to join other datasets in order to show the information in map view.

The following image shows the roads' map *Shapefile* loaded in a GIS Software like Quantum GIS.

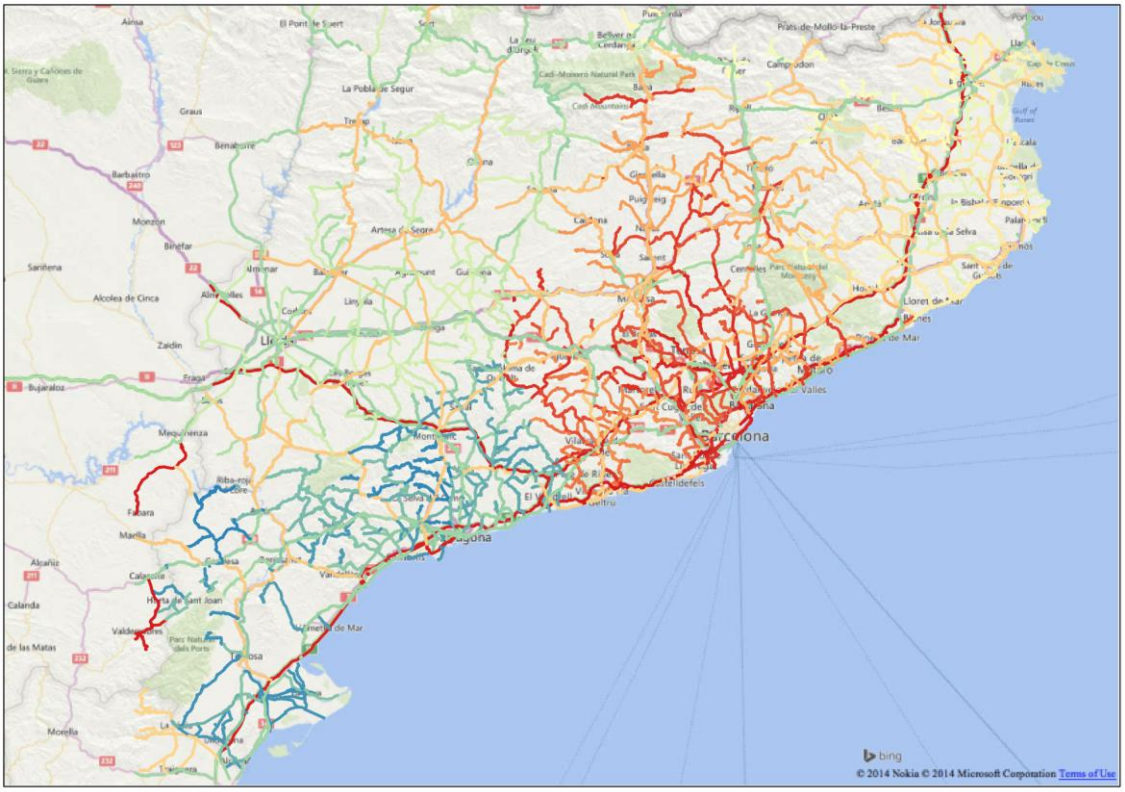


Fig. II.7 Roads' Map *Shapefile*

In this layer we can print information about traffic congestion or routes in a better way that will help the user to view the information more clear.

Traffic Cameras

Also it will be possible to show to the driver real-time camera captures of the roads. As the following image shows, this open data website offers us this kind of information.

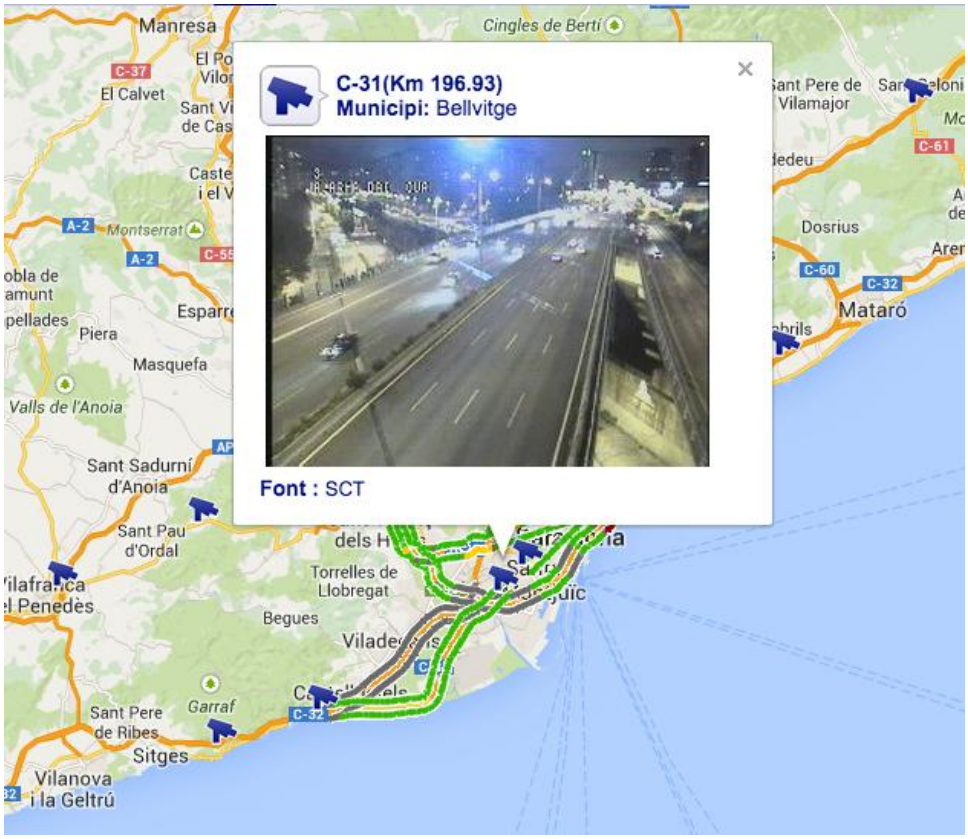


Fig. II.8 Traffic cameras real-time information

This information can also allow us to get the real-time status of the traffic in order to be used by our gateway to finally choose the best route the driver can get to the destination.

Traffic Speed Radars

Another open data source that Catalan Government offers is the location of fixed traffic speed radars in the roads.

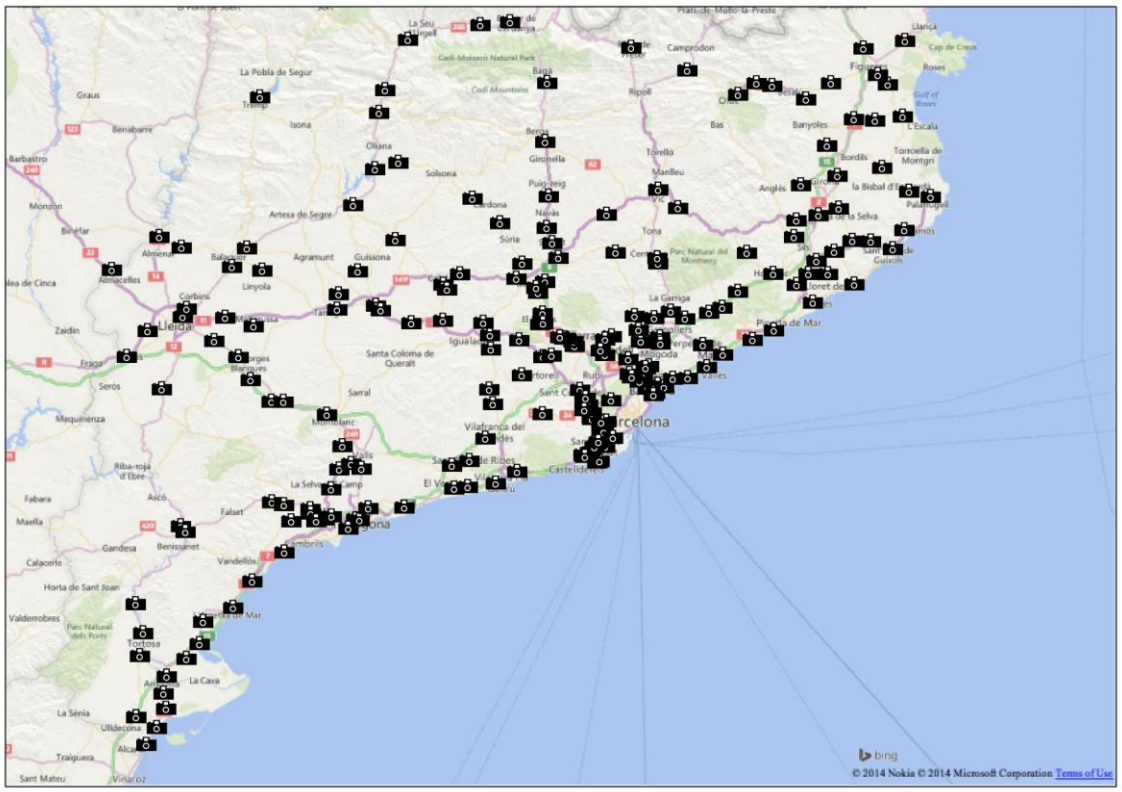


Fig. II.9 Fixed Speed Radars in all the Catalonian territory

Fig. II.10 Roads and Speed Radars

In the picture above we can see both data sources combined, traffic speed radars and road. A feature that we can give to the user can be the possibility to notice when crossing in a road where speed radar is placed.

Traffic Status

Regarding to the available traffic information, we can find a WMS server where to download data about roads and its status.

For example, and shown in the picture below, this service allows us to represent traffic flows and incidents in all the Catalan roads.

Fig. II.11 Traffic status and incidents in Catalan roads

This information can help us to give to the user all the traffic information in real-time and help him to choose the best route to the destination. We'll be able to decide which route is better in order to avoid any problem in our drive.

II.1.4 Open data – Barcelona City territory

In Barcelona City Council Open Data catalogue we can find similar information about traffic (status, cameras, incidents, etc.) like we found in Catalonian Government Open Data website.

But in this case some specific information about the city, like public parking location (fig. II.12), can be useful when the driver comes to the city.

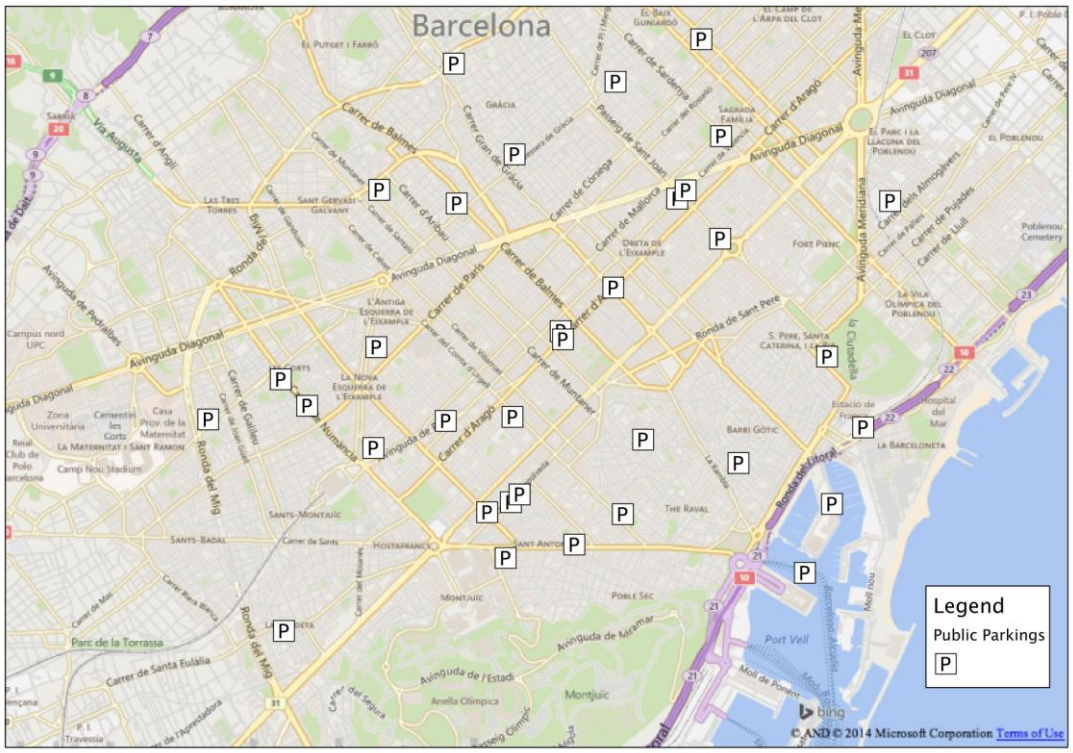


Fig. II.12 Public parking location in Barcelona City area

II.2 Automotive Grade Linux (AGL) Demonstrator

Automotive Grade Linux (AGL) is a collaborative open source project developing common, Linux-based software stack for the connected car. It isn't a protocol itself, it just gives the framework to develop smart car functionalities.

The concept of AGL is to provide a technology platform developers can use to jump-start their development work. It's not a production system. To be a production-ready system, AGL will need to be adopted to target hardware platforms and outfitted with a custom user experience.

By the moment, with AGL we can obtain the platform where to place our development, for example implementing features based on MPCTP and Mobile IPv6 protocols (AGL doesn't give this kind of protocols implementation).

AGL is based on a Fedora Linux distribution. In the project's official website (http://automotive.linuxfoundation.org/) we can find an implementation of the stack where to start the development. The test done with AGL consists in just run the virtual machine offered by the project (VMWare virtual machine). Then, some pictures of the virtual machine's implementation are shown.

Fig. II.13 AGL Demonstrator screens: Car status and Home.

Below, some features given by the framework:

- Car Status
- Web browser
- Music player
- Video player
- Telephone
- Maps (Navigation)

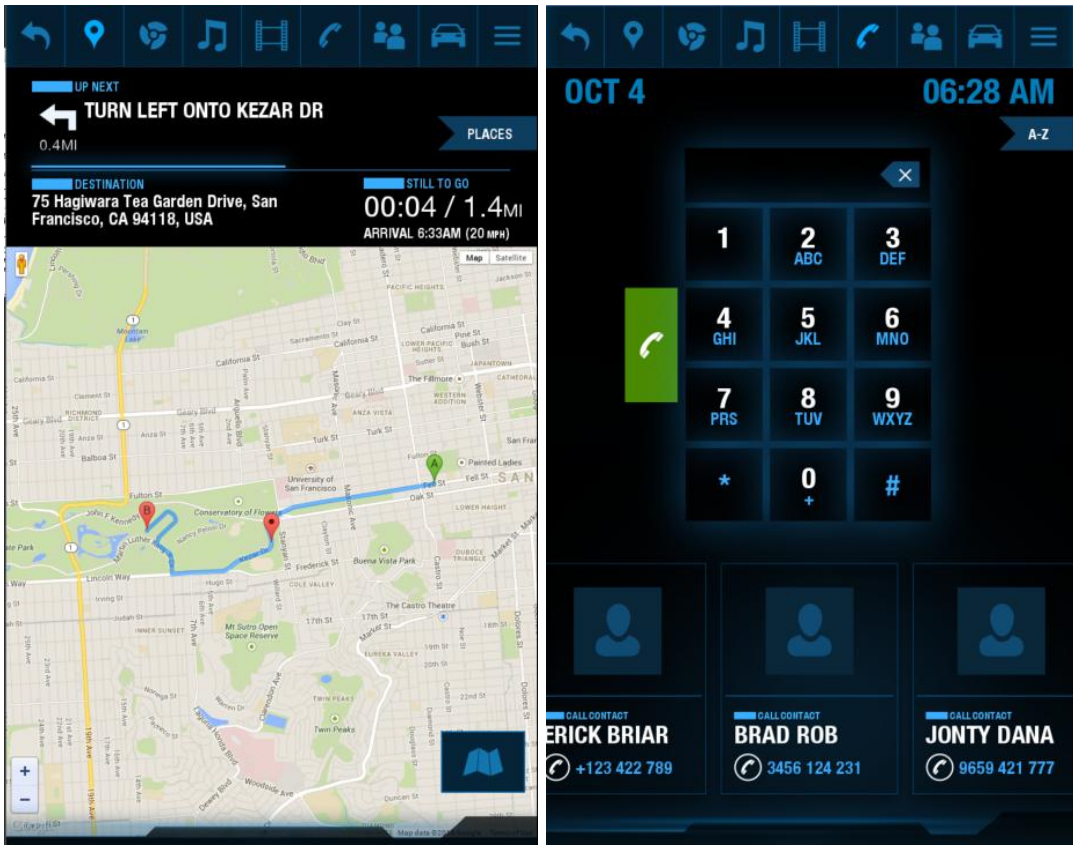


Fig. II.14 AGL Demonstrator screens: Navigator and Phone.

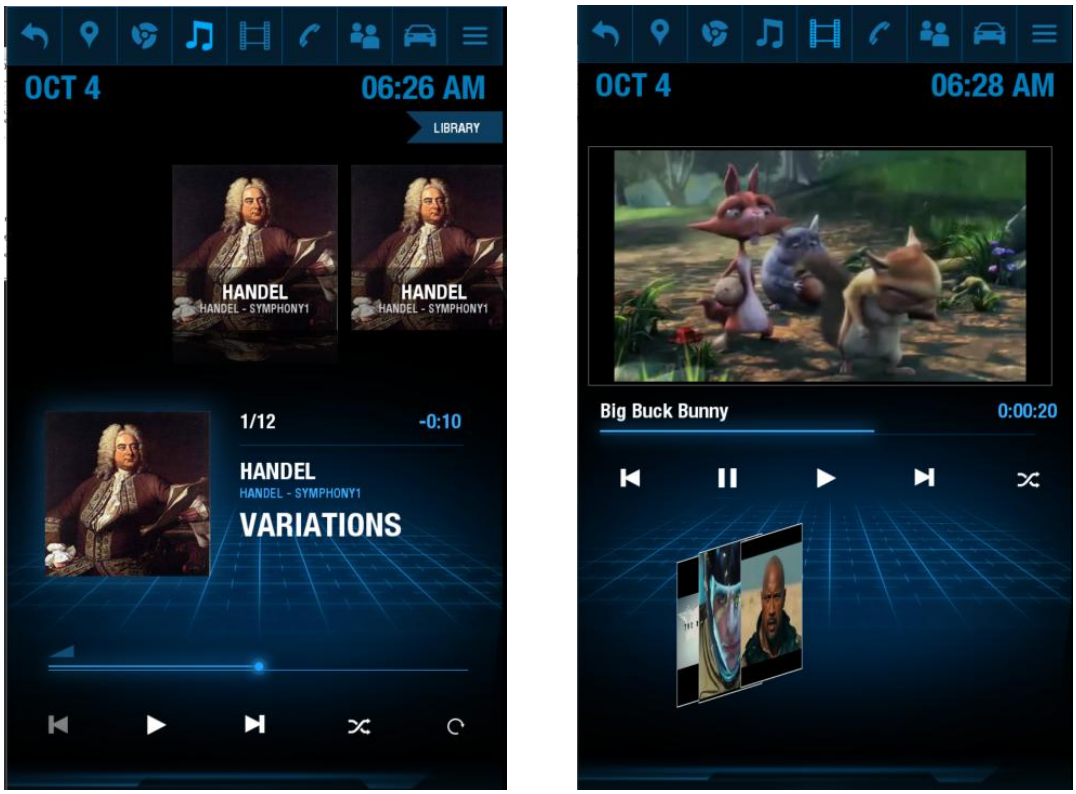


Fig. II.15 AGL Demonstrator screens: Music and Video Players.

REFERENCES

[1] European Commission, “eCall: Time saved = lives saved,” Digital Single Market (Digital Economy & Society), Jan. 2016. [Online]. Available: https://ec.europa.eu/digital-single-market/ecall-time-saved-lives-saved

[2] European Telecommunications Standards Institute (ETSI), “MirrorLink co-operation agreement between Car Connectivity Consortium and ETSI”. [Online]. Available: https://ec.europa.eu/digital-single-market/ecall-time-saved-lives-saved

[3] Opel OnStar, “Compact Connectivity Star: New Astra with Opel OnStar and IntelliLink,” [Online]. Available: http://media.opel.es/media/intl/en/opel/news.detail.html/content/Pages/news/intl/en/2015/opel/08-20-connectivity-onstar.html

[4] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, “TCP Extensions for Multipath Operation with Multiple Addresses,” RFC 6824, Internet Engineering Task Force, Jan. 2013. [Online]. Available: http://tools.ietf.org/rfc/rfc6824

[5] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, “Architectural Guidelines for Multipath TCP Development,” RFC 6182, Internet Engineering Task Force, Mar. 2011. [Online]. Available: http://tools.ietf.org/rfc/rfc6182

[6] C. Perkins, Tellabs Inc., D. Johnson, Rice University, J. Arkko, Ericsson, “Mobility Support in IPv6,” RFC6275, Jul. 2011. [Online]. Available: http://tools.ietf.org/html/rfc6275

[7] V. Devarapalli, Nokia, R. Wakikawa, Keio University, A. Petrescu, Motorola, P. Thubert, Cisco Systems, “Network Mobility (NEMO) Basic Support Protocol,” RFC3963, Jan. 2005. [Online]. Available: https://tools.ietf.org/html/rfc3963

[8] European Telecommunications Standards Institute (ETSI), “Intelligent Transport Systems”. [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/intelligent-transport

[9] Ajuntament de Barcelona, “Wifi hotspots,” OpenDataBCN, Sep. 2015. [Online]. Available: http://opendata.bcn.cat/opendata/en/catalog/WIFI

[10] Generalitat de Catalunya, “Dades recollides per l’aplicació Cobertura mòbil,” Dades Obertes Gencat, Jun. 2015. [Online]. Available: http://dadesobertes.gencat.cat/ca/cercador/detall-cataleg/?id=7710

[11] Wikipedia, “Mobile phone signal, ASU,”. [Online]. Available: https://en.wikipedia.org/wiki/Mobile_phone_signal#ASU

[12] Wikipedia, “Web Map Service,”. [Online]. Available: https://en.wikipedia.org/wiki/Web_Map_Service

[13] C. Ng, T. Ernst, E.Paik, and M. Bagnulo, “Analysis of Multihoming in Network Mobility Support,” RFC 4980, Internet Engineering Task Force, Oct. 2007. [Online]. Available: https://tools.ietf.org/html/rfc4980

[14] “Boosting mobility performance with Multi-Path TCP”. [Online]. Available: http://e-archivo.uc3m.es/bitstream/handle/10016/10314/boosting_valera_FUTURE_2010_ps.pdf

[15] “TC Man Page”. [Online]. Available: http://linux.die.net/man/8/tc

[16] Ajuntament de Barcelona, “Wifi hotspots - Service User Manual,” OpenDataBCN, Sep. 2015. [Online]. Available: http://www.bcn.cat/barcelonawifi/docs/en/manual.pdf

[17] “Ping Man Page”. [Online]. Available: http://linux.die.net/man/8/ping

[18] iOS App, “Fing – Network Scanner”, iTunes Store, [Online]. Available: https://itunes.apple.com/us/app/fing-network-scanner/id430921107?mt=8

[19] “TC/Netem Man Page”. [Online]. Available: http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

[20] “Ifconfig Man Page”. [Online]. Available: http://linux.die.net/man/8/ifconfig

[21] “Iperf – The TCP/UDP Bandwidth Measurement Tool”. Project website: https://iperf.fr/

[22] S.Barré, C. Pasch, and O. Bonaventure, “Multipath TCP: From Theory to

Practice," ICTEAM, Université catholique de Louvain, May. 2011.

[23] N. Williams, P. Abeysekera, N. Dyer, H. Vu, Greenville Armitage, "Multipath TCP in Vehicular to Infrastructure Communications," [Online]. Available: http://caia.swin.edu.au/reports/140828A/CAIA-TR-140828A.pdf

[24] "Live encoder settings, bit rates and resolutions". [Online]. Available: https://support.google.com/youtube/answer/2853702?hl=en-GB

[25] "How much bandwidth does Skype need?". [Online]. Available: https://support.skype.com/en/faq/FA1417/how-much-bandwidth-does-skype-need

[26] "Netflix-Internet Connection Speed Recommendations". [Online]. Available: https://help.netflix.com/en/node/306

[27] "Vagrant – A tool for building complete development environments". Official Documentation website: https://docs.vagrantup.com/v2/

[28] "Curl Man Page". Available: http://curl.haxx.se/docs/manpage.html

[29] "Net-Tools: Linux Networking Base Tools". Project website: http://sourceforge.net/projects/net-tools/

[30] "Netperf Benchmarking Tool". Project website: http://www.netperf.org/netperf/

[31] "Bwm-ng: Bandwidth Monitor NG". Project website: http://www.gropp.org/?id=projects&sub=bwm-ng

[32] "Tcpdump Man Page". Available: http://www.tcpdump.org/tcpdump_man.html

[33] "Tcptrace". Official Homepage: http://www.tcptrace.org/index.html

[34] "Xplot Man Page". Available: http://manpages.ubuntu.com/manpages/gutsy/man1/xplot.1.html

[35] UMIP – Mobile IPv6 and NEMO for Linux. [Online]. Available: http://umip.org/

[36] “Make Man Page”. Available: http://linux.die.net/man/1/make

[37] Linux IPv6 Router Advertisement Daemon [radvd]. [Online]. Available: http://www.litech.org/radvd/

[38] “Ping6 Man Page”. [Online]. Available: http://linux.die.net/man/8/ping6