UNIVERSITÀ DI PISA
Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

An IP Geolocation Approach Based on Smartphones of a Mobile Crowdsourcing System

Relatori:

Prof. Luciano Lenzini

Ing. Alessio Vecchio

Candidato:

Enrico Sallusti

Anno accademico 2012-2013

"The saddest aspect of life right now is that science gathers knowledge faster than society gathers wisdom."

Isaac Asimov

Abstract

In the recent years the attention of the scientific community has been focused on the research field of the geolocation of hosts on the Internet, or IP[1] geolocation. The motivations that make this topic so interesting are various: the pop up of new location-aware applications like smartphones apps, web site contents and advertisement, or the importance of finding the sources of malwares and viruses, or else academic studies on the way people use the Internet and so on. There are numerous projects dedicated to the IP Geolocation, since it is a major challenge: as a matter of fact, there is no direct relationship between the IP address of a host and its geographic location. These projects, of academic and commercial nature, try to find a way to get the best possible geolocation accuracy in order to give a full working service to the users. The contribution of this thesis is to design an IP geolocation approach using the tools provided by The PORTOLAN PROJECT [2, 3], a Internet measurement system based on traceroute that aims at obtaining the Internet graph at the Autonomous system abstraction level and building maps of the signal coverage through smartphone-based crowdsourcing, developed by the departement of Information Engineering of the University of Pisa and the Institute of Information and Technology of the Italian National Research Council. This geolocation approach is based on the theoretical background provided by a project called SPOTTER, developed by the Eötvös Loránd University of Budapest [4, 5]. We cooperated with SPOTTER authors to carry out the experimentation on the implemented geolocation service and the discoveries made within this work let us build some hypotheses useful to develop a landmark selection algorithm and to tune up the measurements in order to lower the error rate of the geolocation process. This thesis work could be considered as a starting point for the development of a tool which makes possible to a user of the PORTOLAN Android App to geolocate any IP addresses on the Internet.

Table of Contents

List of Figures

List of Tables

1 Introduction

The Internet is a global system of interconnected computer networks based on the TCP/IP standard Internet protocol suite. It serves several billion users worldwide and it is considered one of the greatest invention since the telephone, because it has revolutionized the computer and communications world like nothing before. It is, indeed, a *network-of-networks* that consists of millions of private, public, academic, business, and government networks, of local to global scope, that are linked by a broad array of electronic, wireless and optical networking technologies. In this introduction chapter we present a brief history of the Internet, and a description of the importance of the Internet Topology Analysis. To follow we describe the field of IP Geolocation and the motivations behind it, then we give a quick view at the PORTOLAN PROJECT and finally we explain the objectives of this thesis work.

1.1 The internet: history, structure, protocols

1.1.1 A brief history of the Internet origins

It is universally recognized that the history of the Internet [6] can be bring back to the researches of the 1960s at DARPA (Defense Advanced Research Projects Agency), firstly known simply as ARPA. Researches were commissioned by the U.S. government in collaboration with private commercial interests in order to build robust, fault-tolerant, and distributed networks between computers. In 1965 the first wide-area computer network ever built was created: it connected the TX-2 computer in Massachusetts to the Q-32 in California with a low speed dial-up telephone line. All this came together in September 1969 when the first packet switching node was installed at UCLA (University of California, Los Angeles) and the first host computer was connected. Later on two more nodes were added and, by the end of 1969, four host computers were connected together into the early ARPANET. ARPANET was initially funded by the U.S. Department of Defense for military use and for the research laboratories in the U.S. and in the universities.

The funding of a new U.S. backbone by the National Science Foundation in the 1980s, as well as private funding for other commercial backbones, led to worldwide participation in the development of new networking technologies; at this time many heterogeneous networks existed and they merged to create a more complex wordlwide network. Commercial ISPs (Internet Service Providers) began to emerge in the late 1980s and early 1990s. Since the beginning of the 1990s, the Internet has undergone impressive, exponential, growth, which can be appreciated in terms of the number of links that are added everyday, as well as in the numbers of users and in its economical value. This growth led to the current Internet, which connects more than two billion users through a global communication infrastructure that consists of thousands of service providers of different business types such as regional or international transit providers, content providers, enterprise and academic networks, access providers, and content distribution networks.

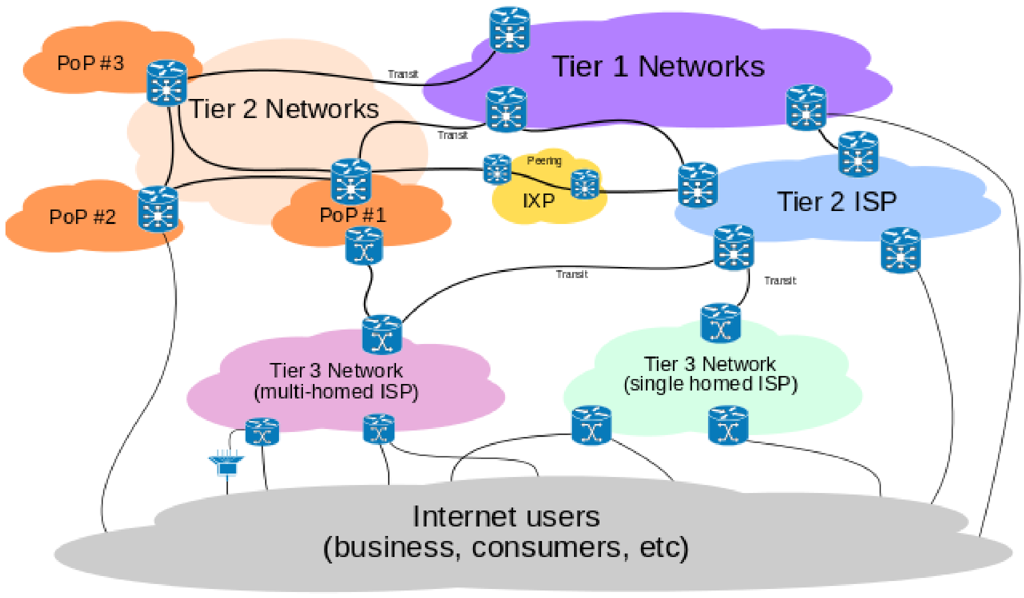


Figure 1.1: The actual Internet infrastructure [http://en.wikipedia.org/ auth: Ludovic.ferre]

In the next section we explain the basis of the Internet topology and how the nodes are interconnected.

1.1.2 The Internet topology

The actual Internet is obviuosly more complex than the Internet of the 1990s, due to the exponential growth of the number of links and nodes. The Internet is so extended that still now we know that despite the increasing effort to unveil the connectivity map (at the Autonomous System Level[1]) at least 35% of the links are still missing from all known databases. Moreover, less conservative estimations indicate that the percentage of the link remaining hidden is greater (more than 50%) [7]. Furthermore every day a great number of links are created and many are suspended or else are completeley deleted; this makes even more difficult to discover every possible link.

The core of the actual Internet is a multi-tier hierarchy of IP[1] transit providers that are connected because of commercial and business agreements

[1]*An Autonomous System (AS) is a collection of connected Internet Protocol (IP) routing prefixes under the control of one or more network operators that presents a common, clearly defined routing policy to the Internet.*

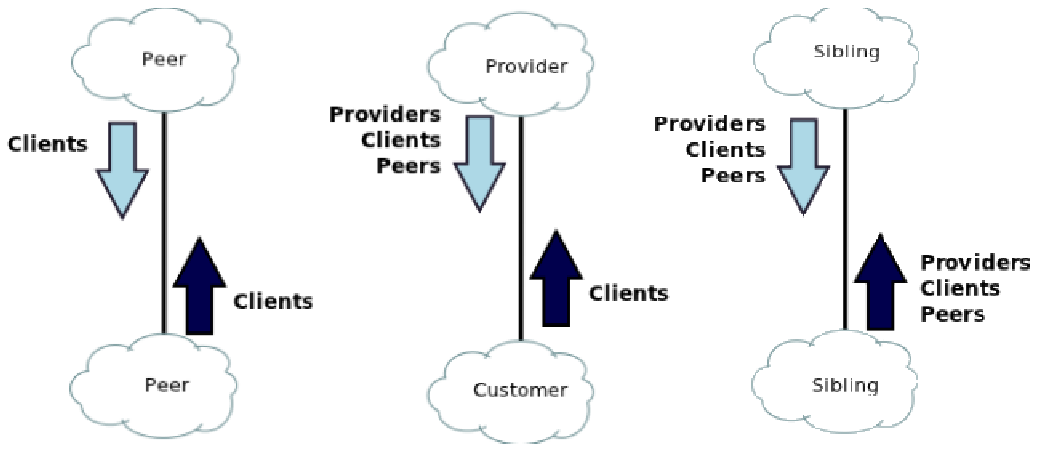


Figure 1.2: Autonomous System Relationships

(see fig. 1.1). When the Internet was opened to the commercial markets, and for-profit Internet backbone and access providers emerged, the network routing architecture was decentralized with new exterior routing protocols, in particular the BGP (Border Gateway Protocol [8]). According to Wikipedia[2] there are 13 Tier-1 networks that can reach every other network on the Internet without purchasing IP transit or paying settlements: those are transit-free network that peer with every other tier-1 network, but not all transit-free networks are tier 1 networks. The regional tiers or Tier-2 networks purchase IP transit from Tier-1 to reach at least some portion of the Internet, and at last we have Tier-3 networks that offer residential and small-business access to the Internet.

As we already said the ISPs *talk* between them because of commercial or business agreements: this is possible because the BGP allows each AS to choose its own administrative policy in selecting the best route, and announcing and accepting routes . The commercial agreements between ISPs can be classified into [9] (see fig. 1.2):

- *Customer To Provider*: A customer pays its provider for connectivity to the rest of the Internet. Therefore, a provider does transit traffic for its customers. However, a customer does not transit traffic between two of its providers.

- *Peer To Peer*: Peers agree to exchange traffic between their respective

[2] *http://en.wikipedia.org/wiki/Tier_1_network#List_of_tier_1_networks*

customers free of charge.

- *Sibling To Sibling*: A mutual-transit agreement allows a pair of ISPs to provide connectivity to the rest of the Internet for each other.

In this thesis work we are not strongly interested in the dinamics of the communications between computer networks but we focus mainly on the possibility of mapping logical address on the Internet (IP addresses) in a geographical context. In the next section we introduce the problems that arises when studiyng the IP Geolocation.

The Portolan Internet topology measurement system project The PORTOLAN[3] system is, as described in [2, 3], a Internet measurement system based on traceroute that aims at obtaining the Internet graph at the Autonomous system level and building maps of the signal coverage through smartphone-based crowdsourcing. Basically, the system is composed by a Client side, an Android App installed on several mobile phones, and a Server side. The PORTOLAN system is developed by University of Pisa and IIT/CNR.

The idea behind this project is to collect information on the topology and the structure of the Internet with the contribution of volunteers. Another substantial difference with other existing projects is that PORTOLAN tries to discover the Internet topology from the edge of the Internet. Connected to their wireless access networks, which can be WiFi or mobile network (such as GSM, UMTS, GPRS, 3G or the new LTE network), the smartphones can probe the Internet from the periphery up to the Internet core and from end-user to end-user.

The most difficult portion of the Internet to be detected by the existing measurement systems is the very same periphery on which PORTOLAN is focused; this make PORTOLAN an extremely useful measurement system on the panorama of the existing projects, in order to discover new links on the Internet. More information on this project can be found in chapter 3 and in the dedicated scientific papers[2, 3], or else directly in the web site.

[3] *http://portolan.iet.unipi.it/*

1.1.3 IP geolocation

The Internet Protocol (IP) is the principal communication protocol in the Internet protocol suite for relaying datagrams across the network. the IP, together with the TCP[4], has the task of delivering packets from a source host to a destination host. For this purpose, this protocol provides an addressing system that has two functions: identifying hosts and providing a logical location service. An IP address is tipically a binary number, but expressed in a human readable *dot-decimal numeric notation*; in particular the Internet Protocol Version 4 addresses (we will not consider IPv6 addresses in this thesis) are commonly written using the *quad-dotted notation* of four decimal integers, ranging from 0 to 255 each. An Ipv4 address consists of 32 bits, which may be divided into four octets .

The interest on IP geolocation, i.e. the geographical localization of Logical IP addresses on the Internet, arised in the last decade. The motivations that focus the interest around this topic are various: the pop up of new location-aware applications like smartphones apps, web site contents and advertisement, or the importance of finding the sources of malwares and viruses, or even spammers, or else scientific studies on the way people use the Internet and more. There are various projects, of commercial or scientific nature, dedicated to the IP geolocation (we will see them in chapter 2), since it is a major challenge, i.e. there is *no* direct relationship between the IP address of a host and its geographic location.

Basically, there are two ways of getting the location of network hosts which are not known *a-priori*: the first method is to use passive measurements, based on the information collected by registries, or databases on the Internet like WHOIS [10] databases, DNS LOC records or DNS names [11]. The second approach is based on active measurements to approximate IP locations, like the time needed to reach a certain destination on the internet and then to return back to the source (Round Trip Time); this approach try to find a relation between the delay experienced by an IP packet on the Internet sent to a certain host and the spatial distance. Moreover, some of the projects based on this approach use topological informations to refine the geolocation process.

The next chapter is dedicated to these commercial and academic projects

[4] *http://tools.ietf.org/html/rfc793*

that make use of one of the two approaches or the other. In the next section, instead, we present the Internet measurement system developed by the University of Pisa and the IIT/CNR. In this thesis work, an IP geolocation service has been integrated in the framework of PORTOLAN system.

Spotter project SPOTTER[5] is a web service based on a novel probabilistic geolocation approach developed by the Eötvös Loránd University of Budapest where the relationship between network delay and geographic distance is statistically analysed. Spotter authors show in [4, 5] that the distribution of spatial distances for a given delay follows a universal distribution and is independent of a landmark's position from where the measurement was performed. While the state-of-the-art techniques use separate calibration data for each landmark to determine their internal models (see chapter 2), this method handles all these data together to derive this delay-distance model. In this thesis work we developed an IP Geolocation approach that is based on SPOTTER theories.

1.2 Objectives and contents of this thesis

The main purpose of this thesis is to design a new IP Geolocation approach based on smartphones inside the PORTOLAN PROJECT context. This approach has to:

- Be based on RTT measurements;
- Take advantage of the theoretical background provided by SPOTTER approach (see Chapter 2, section 2.2);
- Be stable and efficient in a mobile environment (it has to work as a tool for an Android App);
- Be transparent to the smartphone owner, who can not be bothered by the measurements.

We developed both in the client side and in the server side a specific module for the geolocation, and we added a Database in the server side where to

[5] *http://spotter.etomic.org/Default.aspx*

store the measurements taken with the smartphones. The experimentation was taken in collaboration with the University of Budapest and in particular with the SPOTTER project authors.

The thesis is structured into five more chapthers. Chapter 2 gives an overview on the existing projects on the IP Geolocation field and then it focus on the SPOTTER project and its theoretical bacgkground.

Chapter 3 describes the design process of the geolocation module, which includes a brief description on how the PORTOLAN system works, the integration with previously implemented components and of components developed by third party.

Chapter 4 shows the implementation of every single component of the geolocation module, of their sub-components, of the communication protocols between them and also focuses on the modification produced to previously implemented components on the system.

Chapter 5 describes the experiments carried out for testing the implemented modules and the validation and evaluation of the achieved results.

Finally, chapter 6 concludes this thesis and gives an overview on some of the possible future developments.

2 IP Geolocation Techniques

In this chapter we want to give a full view on the most important techniques developed in the relatively new field of IP geolocation. As we will see in section 2.1 there are two opposed approaches to geolocation: based on passive or active methods. We then focus in section 2.2 on a web geolocation service called SPOTTER based on a probabilistic model to determine the most likely position of a certain IP address. The information presented in this chapter is useful in order to comprehend how the geolocation actually works and then being able to build a system to geolocate IP addresses.

2.1 State of the art

This section offers an overview of the IP geolocation state of the art.

During the recent years several IP geolocation projects have emerged, all of them aim at giving an accurate approximation of the location of network hosts which are not known *a-priori*.

Many of these projects use a passive measurements approach, based on the information collected by registries like WHOIS or DNS records, that are unreliable due to the fact that the addresses contained on those registry usually differs from the real locations of its routers (in case of large ISP those errors grows larger).

The second, more mature approach, considers active delay and topology measurements to approximate IP locations. These projects try to find a relation between network delay (i.e Round Trip Time) and spatial distance.

2.1.1 Passive methods

Many of the existing geolocation projects use WHOIS databases, DNS LOC records or DNS names to determine the location of a given host. From WHOIS one can retrieve the name and street address of the organization which registered the address block (examples of registries are ARIN[1] in north america, RIPE NCC[2] in Europe and APNIC[3] in Asia).

However, for a large ISP or a geographically dispersed organization the registered street address usually differs from the real location of its hosts, as previously stated. A similar problem arises in the use of DNS names, since the names can be both useful or misleading due to the naming conventions of the ISP[12].

Other registry based techniques include commercial approaches where one obtains the description of the geographic layout of an ISP's network and internal routing policies. There are several commercial solution that can be easily found (some are freeely available) like *Ip2location, MaxMind, Tamo Soft* and *IPligence*. For a much finer search *traceroute* command is used to

[1] *https://www.arin.net/*
[2] *http://www.ripe.net/*
[3] *http://www.apnic.net/*

find clues to the location of the IP address as the names of the routers through which packets flow from the source to the destination host might hint at the geographical path of the final location.

In general, these passive methods are very accurate, but in some cases their errors are very large. For IP-to-country databases, some vendors claim to offer *98%* to *99%* accuracy although typical *Ip2Country* database accuracy is more like *95%*. For IP-to-Region, accuracy range anywhere from *50%* to *75%* if neighboring cities are treated as correct [13]. In [4] and [5] the authors demonstrate that those method can't be always reliable. They tried to locate GEANT [14] routers in Budapest using *MaxMind*[4], which is one of the leading passive IP-to-location database, and it provided *fake* locations for them (according to *MaxMind* the routers were located in Cambridge, where GEANT is registered).

2.1.2 Active methods

These methods apply a completely different approach and they take advantage of active measurements to overcome the above limitations. *IP2Geo* [15], for example, contains a tool called *Geoping*, which tries to approximate the geographical location of network hosts on the basis of packet delay measurements.

A more complete approach is the simultaneous application of several delay constraints to infer the location of a network host. This is done by constraint based geolocation (CBG) techniques[16]. CBG introduces a triangulation-like method to combine the distance estimates from all landmarks. To estimate delay-distance relation, each landmark measures the delay from itself to all the others. In general, each delay measurement defines a circle around the landmark from where the delay was measured. The possible locations of the target node are determined by intersecting all of these circles as you can see in Fig. 2.1. In most of the cases this intersection produces a region in which the target node must be located.

Another technique is where the topology information and latency measurements are used together in the location estimation. This method type is called

[4] *http://www.maxmind.com*

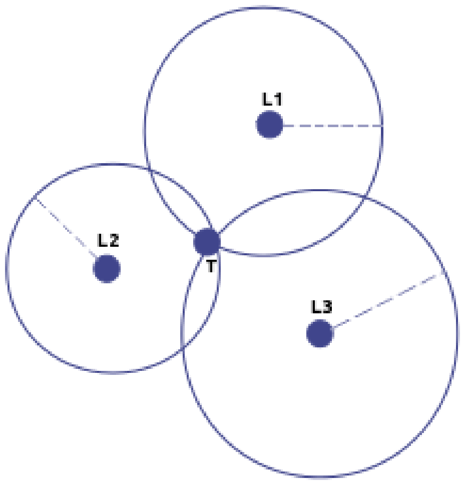


Figure 2.1: Example of CBG technique

topology based geolocation (TBG) [17]. TBG localizes all the intermediate routers between the landmarks and the target node, using *traceroute* techniques. This approach is based on link-latency estimations and on precise topology discovery.

In [5] the authors improved these two approaches demonstrating how an accurate approximation of the propagation delay can improve the accuracy of distance estimation. They developed a path-latency model where they separate the propagation and per-hop delays in the overall packet latency. Besides, they introduced the importance of one-way delay measurements, instead of classical RTT delays. This type of constraint yields additional information into the geolocation process by limiting the overall physical length of a given measurement path. The same authors then, based on these studies, developed a geolocation service on which this thesis work rely, called SPOTTER[4].

In the next subsection we give a detailed technical description of SPOTTER and his theoretical background since we cooperated strongly with his authors to make this service work with the PORTOLAN system.

2.2 Spotter

This section offers a detailed overview of SPOTTER geolocation service.

The first subsection take focus on the theoretical background on which the service is based, then the second subsection briefly describes the world surface tesselation method used, while the final subsection present the algorithm of the web interface.

2.2.1 Delay-Distance model

Spotter is a geolocation service based on a probabilistic model to determine the most likely position of the target based on signal propagation delay values between the landmarks and the target. Let L denote the landmark given by its latitude and longitude coordinates:

$$\mathbf{L} = (L_{lat}, L_{lng}).$$

Let T represent the target node, whose actual $\mathbf{T} = (T_{lat}, T_{lng})$ coordinates are unknown. In the following the target's position will be described with a random variable:

$$\tau = (\tau_{lat}, \tau_{lng}).$$

The spatial probability density function of τ determines how likely the target is at given regions of the globe. In this approach the density function depends on the location of the landmark and the signal propagation delay between L and T[5]. For a given L and a fixed delay d the function is denoted by $g_{d_i}^{L_i}(\tau)$. With this notation the conditional probability that T falls into region H is provided by

$$P(\mathbf{T} \in H | L \bowtie d) = \int g_d^L(\tau) d\tau, \tag{2.1}$$

where the condition $L \bowtie d$ indicates that d is the propagation delay between L and T. As delay measurements do not carry any information on packet contents, constraints can be derived only on their distance. Hence, the first important assumption is that the behavior is isotropic, as the distance probabilities are equal in all geographic directions from L. g_d^L function then defines a ring-like surface around the landmark. Considering that a single landmark

[5]Note that L and T represents the node while $\mathbf{L}$ and $\mathbf{T}$ are coordinates (lat, lng).

does not provide a well-defined location for the target, a multiple landmark model is much more convenient. Suppose we have n Landmarks: $L_1,...,L_n$ and d_i propagation delay from L_i to the target T, given that $i \in \{1,2,\dots,n\}$ with $1 \leq i \leq n$. The target position is described with a random variable τ, as previously said. The conditional probability that the target is located in a certain H region can be expressed as the product of every landmark densities:

$$P(\mathbf{T} \in H | L_1 \bowtie d_1, \dots, L_n \bowtie d_n) =$$

$$= \prod_{i=1}^{n} P(\mathbf{T} \in H | L_i \bowtie d_i) =$$

$$= A_H \prod_{i=1}^{N} \int g_d^L(\tau) d\tau \tag{2.2}$$

- $A_H = P(\mathbf{T} \in H)^{1-n}$ depends on the area of H and can be substituted with a constant for all regions having the same area (we'll see next what this means);
- $g_{d_i}^{L_i}(\tau)$ represent the corresponding spatial probability densities. The basic principle of the localization method is that the estimated position must fall into the region designated by the $g_{d_i}^{L_i}(\tau)$ functions such that the joint probability in (2.2) is maximized for a reasonably sized region H.

To be able to apply the previous results $g_{d_i}^{L_i}(\tau)$ had to be calculated. In [4] the two fundamental hypothesis are:

First, as assumed early in this section, g_d^L is considered an isotropic function, since there is no reason to doubt that signal propagation characteristics agree in different geographical directions. As a consequence, g_d^L it is described with its f_d^L radial profile.

Second the f_d^L distribution it is considered independent of the actual location of L, and thus it is substituted with f_d, that is to say a distribution common for all landmarks. This is the most important and controversial hypothesis of this model, as it has a strong impact on the quality of the service but has not sufficiently been explained on this paper work, as the authors presented only a few validation tests (refer to [4] on section IV.B for more details).

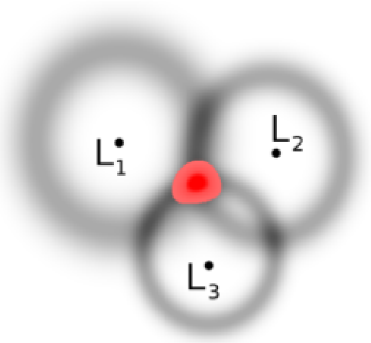


Figure 2.2: Example of an estimated region

To determine the spatial probability density of τ the surface of the globe is divided into a finite number of cells with equal area (like this A_H can be substituted with a constant) and calculate (2.2) over them (see subsection 2.2.2 for details about the tesselation method used by SPOTTER). This way we obtain a probability value for each cell, which is a much finer information on the possible position of the target than the one provided by the CBG method described in [16] (see section 2.1.2). To deliver position estimations from the individual cell probabilities Spotter provide two differents methods:

- In case an estimated region of the target is needed, the algorithm calculate the union of the 3 most probable cells like in figure 2.2. The regions emerge where all the involved rings provide high probability values;

- On the other hand, for certain purposes an estimated region is not appropriate, instead a single *best* target location value is necessary. Due to the nature of the probabilistic approach of SPOTTER there are several choices to define a single location. For instance, one can simply select the center of the estimated region or use the coordinates corresponding to the maximum for mean value of the probability distribution.

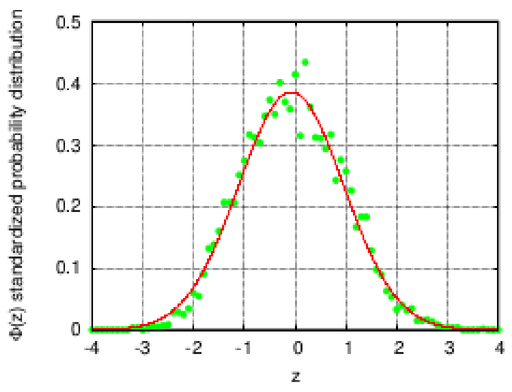


Figure 2.3: f_d approximation with the standard normal distribution

The approximation of f_d is done by an analysis of real world calibration data collected in *PlanetLab*. In brief the authors determined the probability distribution of the standardized data values and as you can see in Figure 2.3, where each dot represents a measurement between two nodes, there is a good match with the standard normal distribution. This means that f_d is well approximated by a normal distribution at any fixed delay d. The above observations enable a simple approximation of f_d with a normal distribution:

$$f_d(s) \approx \frac{1}{\sqrt{2\pi}\sigma(d)} \exp\left(-\frac{(s-\mu(d))^2}{2\sigma^2(d)}\right), \tag{2.3}$$

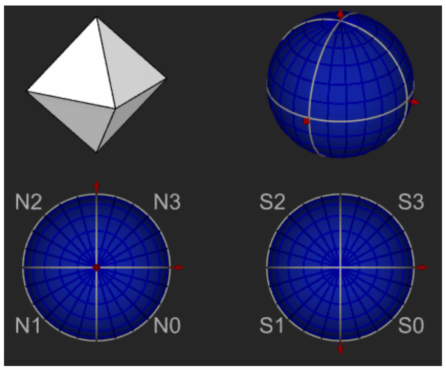
where s is a random variable describing the distance. During the evaluation process the (2.3) distance distribution is used to calculate the $g_d^L(\tau)$ spatial probability densities. For a given delay $L \bowtie d$ this is done by the following formula:

$$g_d^L(\tau) = A_d f_d(S(L,\tau)). \tag{2.4}$$

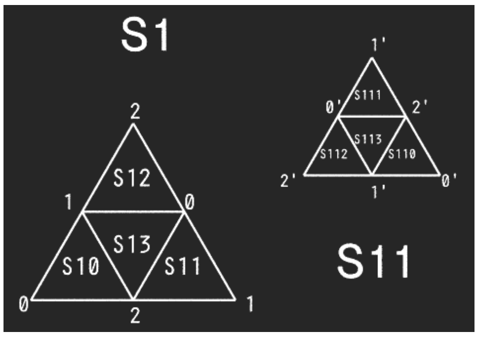
Here, $S(L,\tau)$ represents the great-circle distance between L and τ, while A_d denotes the normalization factor.

To validate Spotter hypotheses the authors in [4] examine the localization accuracy of its approach and compare the results with the performance of the state of the art delay models of CBG and Octant[6]. Comparing the numerical results Spotter shows significantly better performance and it appears to

[6] *http://www.cs.cornell.edu/~bwong/octant/*



(a) Level 0 and views of the projection of the octahedron onto the sphere.



(b) Trixels division example.

Figure 2.4: HTM hierarchy.

be more robust against measurement anomalies (due to his probabilistic approach).

As we said earlier in 2.2 to substitute A_H with a constant value the surface of the globe is divided into a finite number of cells with equal area. In the next subsection we describe the HTM technique, used in Spotter to subdivide the surface of the earth with great precision.

2.2.2 The Hierarchical Triangular Mesh Tesselation

The Hierarchical Triangular Mesh (HTM) [18] is a method to subdivide the spherical surface into triangles of nearly equal shape and size. The HTM gives a very efficient indexing method for objects localized on the sphere, organized in levels. It start with an octahedron, this will be level 0. As you project the edges of the octahedron onto the sphere 8 spherical triangles are created, 4 on the Northern and 4 on the Southern hemispheres. Four of these triangles share a vertex at the pole and the sides opposite the pole form the equator. You can imagine these by orienting a regular octahedron so that two of its vertices are at the poles, and the other 4 are equally spaced on the equator. The spherical polygons are the projection of the edges of the octahedron onto the circumscribing sphere.

As you can see in figure 2.4a the 8 spherical triangles $[N_0 ; N_3]$ and $[S_0 ; S_3]$ represent respectively the 4 northern and southern spherical triangles. These are called level 0 *trixels*. Each trixel can be split into four smaller trixels by introducing new vertices at the midpoints of each side, and adding a great circle arc segment to connect the new vertices with the existing one; trixels division repeats recursively and indefinitely to produce smaller and smaller trixels.

This subdivision scheme (shown in figure 2.4b) suggests a way of labeling the trixels. Each trixel has three vertices labeled 0, 1 or 2. The opposite midpoints are labeled 0', 1' and 2', respectively. The newly created trixels receive a label formed by the name of the parent appended with one of {0, 1, 2}, indicated by the vertex shared with the parent. The central trixel is suffixed with a '3'. Smaller trixels have longer names. The length of the name of the trixel also indicates its level. Points in this decomposition are represented by a leading 1 bit and then the level 0 trixel number $[0\ldots7]$ and then the successive sub-trixel numbers $[0\ldots3]$. This gives each trixel and its center-point a unique 64 bit identifier, called an HTMID that represents a particular trixel in the HTM hierarchy (the smallest one is 8). Though the division process can continue indefinitely, the 64-bit representation runs out of bits at level 31. Level 25 is good enough for most applications as it has an accuracy of about 0.6 meter on the surface of the Earth.

2.2.3 The service algorithm

Spotter implementation follows a modular structure: the system is accessed via a web interface where the user can enter the target domain name or IP address to be localized, then *PlanetLab* landmarks measure delays to the target and the results are forwarded to the evaluation module. This module applies the probabilistic model described in subsection 2.2.1 to determine the location of the requested IP address. After evaluation, the expected target position is returned to the user, while the individual cell probabilities are visualized on Google Maps. Let's see in detail the algorithm steps:

1. *Targets choice*: Users enter the targets domain name or IP address to be localized on the web interface[7];

2. *Data Collection*: PlanetLab landmarks measure delays to the target and the results are forwarded to the evaluation module. Spotter measures

[7] *http://spotter.etomic.org/Default.aspx*

10 round-trip delays (RTT) from each PlanetLab landmark to the target node. The evaluation module extracts the base values (tipically the minimum round-trip delay for each landmark). This way the effect of routers queuing can be significantly decreased [5];

3. *Model Evaluation*: Execution of the probabilistic delay-distance model through the division of the globe surface in regions H via the Hierarchical Triangular Mesh (see 2.2.2). At a given resolution level Spotter determines the probability value for each HTM cell by approximating the integral in 2.2;

4. *Visualization*: The outcome of the evaluation is a spatial probability distribution and its moments. The Spotter web interface displays the estimated region and the expected location on a Google Maps application.

[4] shows that SPOTTER accuracy is greater than in CBG and TBG and every hypothesis is verified, but something missing in the paper is the method used to select the landmarks for the experimentation, as they only say that the selection is made between PlanetLab routers. We think that an algorithm for the filtering of landmarks (based on geographical and logical constraints, i.e the country or the Autonomous System number of the target) is of great importance to enhance the accuracy of the geolocation. In this thesis work we determined some hypotheses that allow the implementation, on future work, of an algorithm that can be used to select the best landmarks available as we'll discuss later.

3 The PORTOLAN System Design

The PORTOLAN[1] system is, as described in [2, 3], a Internet measurement system based on traceroute that aims at obtaining the Internet graph and building maps of the signal coverage through smartphone-based crowdsourcing. Basically, the system is composed by a Client side, an Android App installed on several mobile phones, and a Server side. In this thesis work, as already introduced in chapter 1, we developed a geolocation service based on RTT measurements and on the SPOTTER service, used by the PORTOLAN system to find the geographic position (latitude and longitude) of an IP address. The chapter is structured into four parts. In sections 3.1,3.2 and 3.3 we give an overwiev of PORTOLAN system client and server side as developed in [2, 3]. Then in section 3.4, we show how the geolocation measurement subsystem works in the Android app. In section 3.5, we focus on the description of the Database used to collect the measurements taken from the mobile devices. Finally, in section 3.6 we describe the web interface developed to send the targets list to the devices and then gather the information collected in the database, used by SPOTTER authors to calibrate their service to work with our system.

[1] *http://portolan.iet.unipi.it/*

3.1 The PORTOLAN system

As previously said the PORTOLAN system is composed by a Server side (described in section 3.2) and a great number of Android smartphones with the Portolan Client App installed. The paradigm of *crowdsourcing* used by Portolan, is, according to [19], the practice of outsourcing tasks to a large group of people. The crowdsourcing encourages the best-qualified and most creative participants to join in on a project and it can involve possibly milions of people in helping the scientific research. Furthermore PORTOLAN project follows a *bottom-up* approach where the phones are used as mobile monitors. In this approach the measurements are taken from the edge of the Internet, and not from the top, like in common approaches followed by previous research projects (*top-down*).

3.1.1 Current research projects

There are two different approaches used to discover the topology of the Internet: based on passive measurements or active measurements. We present a brief description of the most important research projects that follow these approaches:

Passive measurements based Research projects like RouteViews[2], PCH[3] and RIPE RIS[4] make use of passive data gathering for Internet topology discovery, i.e. they provide raw BGP data that can then be used to discover the topology at the AS level. The Border Gateway Protocol (BGP) is the *de facto* routing protocol used to connect routers, called BGP speakers, that belong to distincts ASes. Two BGP speakers forms a transport protocol connection between each others. When the connection is established, a speaker sends its entire routing table to the peer. Every time there is a change in the speakers routing table, the peers exchanges BGP update messages. The research projects already mentioned collect these messages using monitors that act as BGP speakers (called Route Collectors). Despite the significant measurement

[2] *http://www.routeviews.org/*
[3] *https://www.pch.net/home/index.php*
[4] *http://www.ripe.net/data-tools/stats/ris/routing-information-service*

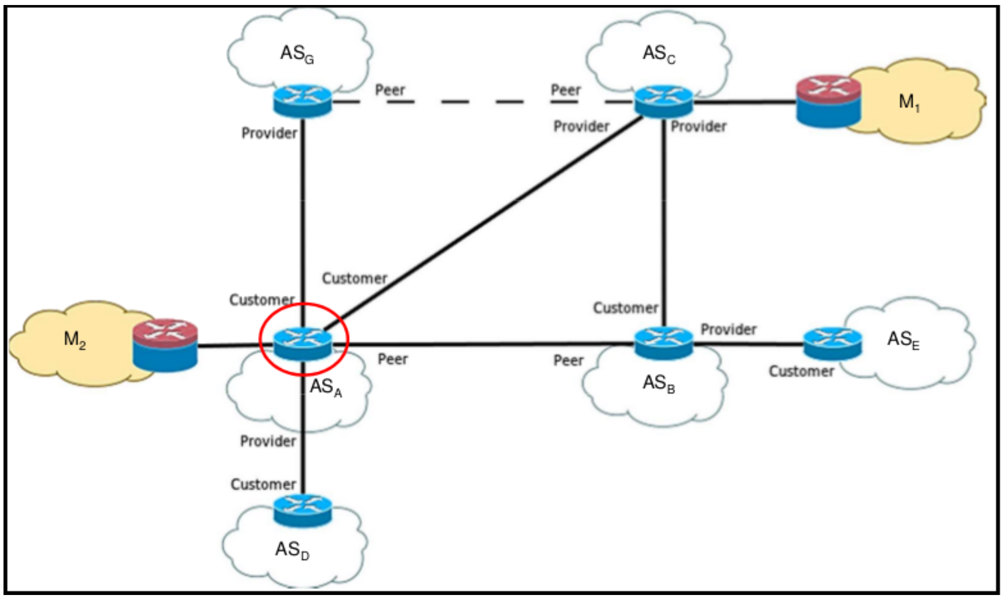


Figure 3.1: Link between AS_G and AS_C is lost in M_1 wich is at the top of the graph, respect to M_2 located on the bottom edge

efforts, these methods have not been useful to fully discovered the Internet Topology [20]. The previous projects failed because of two main reasons:

- the measurement campaigns are carried out with a small number of Route Collectors, e.g. 8 RC for the RouteViews project and 15 for RIPE RIS;
- these monitors are often located closer to the Internet core than to the Internet edges. This prevents the monitors from observing the large part of the Internet near a vast majority of Internet users as you can see in the example of Fig. 3.1.

Active measurements based The most used active measurement method is traceroute. Traceroute is a computer network diagnostic tool for displaying the path and measuring transit delays of packets across an Internet Protocol network. The history of the route is recorded as the RTTs of the packets received from each successive host in the path. Traceroute allows the discovery of the topology at the IP interface level. As previously stated PORTOLAN topology discover is traceroute based. The most important research projects that takes advantage of active measurements are CAIDA ARCHIPELAGO[5] and DIMES[6].

3.1.2 PORTOLAN system features

The PORTOLAN server is responsible for:

- Managing the smartphones, giving them IDs that can serve as key to permits the communication between the server and the clients;
- managing and supervising the measurement campaigns, through the use of these IDs;
- storing the collected measurements.

On the other hand the PORTOLAN client side, besides the measurements, provides differents services to motivate the user to give his contribute to the project such as:

- a Traceroute and a Ping tool;
- a tool to estimate the maximum throughput of a network;
- a WiFi scanner;
- a mobile network signal analyser.

In this thesis work a geolocation module has been added to the original architecture, both in the server and in the client infrastucture. We'll describe the features of the PORTOLAN system in the next two section.

3.2 The server structure

In this section we'll describe the architecture of the PORTOLAN system server side as it is in the present moment, since there are some evolutions ongoing in this context. In order to have a simple way to control the complexity due to the increasing number of smartphones and, as a consequence, the increasing

[5] *http://www.caida.org/projects/ark/*
[6] *http://www.netdimes.org/new/*

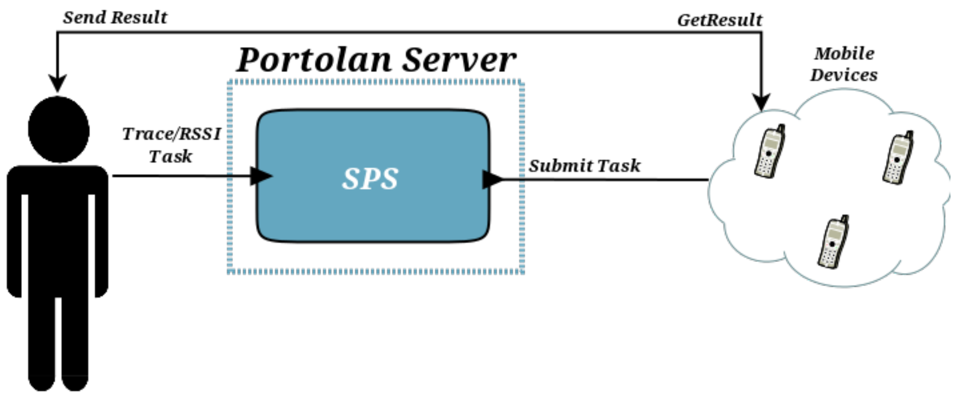


Figure 3.2: PORTOLAN Basic structure

number of data, and to give the user a standard interface for the interactions with the server, the PORTOLAN system takes advantage of the *Sensor Web Enablement* framework of Open Geospatial Consortium (OGC). The component used is called SPS (Sensor Planning Service): this is a Java Servlet that offers an XML web interface to manage the operations to the sensor network. In detail the most common operations are:

- Submit task;
- Get the feasibility of a task;
- Cancel a previously submitted task;
- get confirmation on the status of a previosuly submitted task.

As the name suggests, this framework is normally used with sensor networks. In this context the server considers the smartphone network as a mobile sensor network; thus the human user can submits tasks with the help of this framework.

The system architecture presented so far is summarized in Fig. 3.2, nevertheless this structure presents some limitations due to the fact that SWE framework was developed to work with a relatively small number of sensors. There are three main issues that must be considered in our case:

- Availability

  The mobile devices can't be always available, since they are smartphones on which we have no control;

- NAT Reachability

  The devices can reside in a private network, thus they have a private IP address and they connect to the Internet via a gateway. The only way to reach such devices is by maintaining a persistent connection between the server and the devices. It becomes difficult to set up this connection for a large number of devices;

- Scalability

  The number of smartphones that contribute to the PORTOLAN project is growing fast. The structure as we have seen till now can not afford to work with a large number of devices.

In order to pass through limitations imposed by the nature of mobile devices and by the SWE framework, a new system architecture that differs from the basic architecture shown in figure 3.2 has been designed. Instead of registering all the smartphones with the SPS, now it has been developed a module that registers to the SPS as a single sensor. This module then communicates with the mobile devices. This way the user becomes unaware of the complexity of the system, i.e. the user submits tasks to the system and the system is responsible for sending the tasks to the most suitable devices. The system knows which device to select according to the following properties: country (mandatory), geographic area, Autonomous System, network type, provider name, mobile ID [3]

To make the system scalable *Proxies* have been added to the original architecture, one for each country theoretically (currently only one proxy is available, in Italy). A mobile device registers itself with the proxy of his geographic area, via a Proxy Assigner that is located in the main server. The proxy assigner sends information about the proxies at the moment of the first installation of the client. From this moment on the device polls the proxy with a regular frequency in order to ask if there are available tasks; if at least there is one task available, the proxy sends the task to the device that polled to it.

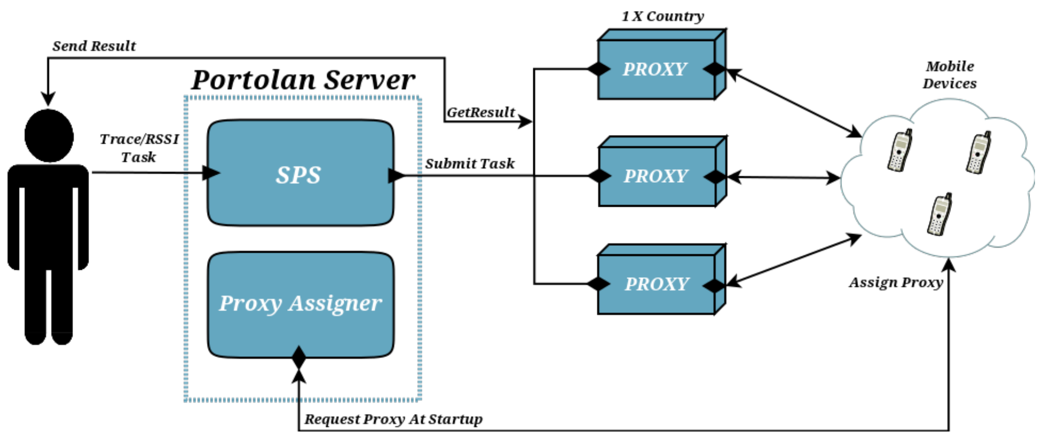


Figure 3.3: PORTOLAN architecture revisited

In conclusion, we list all the steps of PORTOLAN system working principles (for a detailed description refer to [3]):

1. The user submits a task to the SPS;
2. SPS module splits the original task in microtasks and delivers them to the proxy identified by the original task;
3. The proxy waits for mobile devices polls, and assigns microtasks to the most suitable devices;
4. When the execution of a microtask is terminated, the mobile device sends the results to his proxy and notifies of microtask completion;
5. When the execution of a task is terminated, the proxy notifies the human user of task completion via a module that sends him an e-mail.

See Fig. 3.3 for a schematic resume of the most important features of PORTOLAN system architecture

3.2.1 Google Cloud Messaging

As already mentioned earlier, even if the user submits a task to the system he doesn't have control on which device the submitted task will be executed; this was necessary due to ensure the transparency of system complexity to

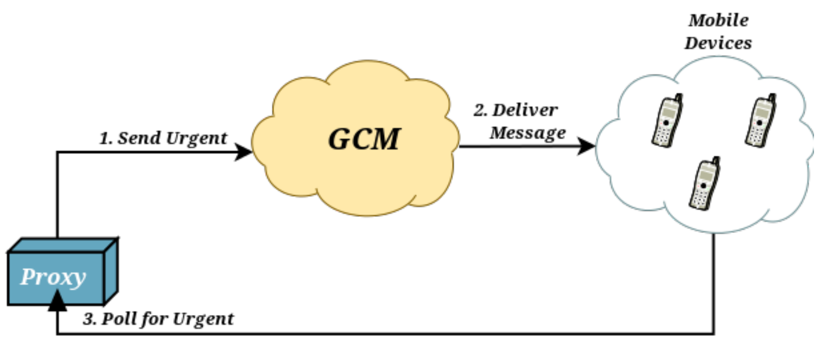


Figure 3.4: Example of GCM urgent message delivery

the user. The normal behaviour of the system lets the smartphones ask their proxy for a task if there is at least one, this way the proxy can select the most suitable devices among the ones that polled it and not among every device available. This way, as we expect that the number of devices will grow larger quickly, the complexity of the selection algorithm is cutted down. Actually, there is one way to change this behaviour, that takes advantage of Google Cloud Messaging Service (GCM) provided by GOOGLE[7]. GCM is a service that allows to send data from a server to Android powered devices, even if the application is not running. In order to use this service there are some important conditions:

- The application must have a GCM registration ID;
- Android OS versions under 2.2 are not supported;
- Under version 4.0.4, the device must have at least one logged in Google account;
- The server must get an API key for using the service.

When the user submits a task he can set a flag, named *Urgent*, wich gets the mobile devices to poll immediately instead of waiting for the time interval to expire. When the proxy receives an urgent task it send via GCM service a message to all mobile devices it handles, notifying the arrival of the task as showed in Fig. 3.4. All mobile devices receiving the message immediately

[7] *http://developer.android.com/google/gcm/index.html*

poll to the proxy, which assigns them microtasks from the urgent task, if they satisfy task requirements. This mechanism allows to keep a low polling frequency when tasks to execute are not urgent, and, on the other hand, when an urgent task is submitted, to force polling from mobile devices, in order to assign urgent microtasks as quickly as possible.

In this thesis work, in addition to the usage just described, we use GCM service to send geolocation campaign requests to mobile devices, which will be our landmarks (From now on we refer to the mobile devices as *landmarks* in order to harmonize with SPOTTER terminology [4]). In brief, the user submits a geolocation campaign via a web interface where he can specify a list of targets, or a single target as well, to be pinged by landmarks[8]. Then, the system asks the GCM service to deliver the measurements requests to all smartphones registered to GCM. An algorithm for the selection of landmarks is still under development; in this thesis work we create the conditions for the design of this algorithm by presenting some hypotheses arised from the experimentation taken on geolocation measurements (see Chapter 6).

3.2.2 Geolocation Service module

In addition to the architecture showed earlier in this section we developed a specific module for the geolocation composed by two Java Servlets and a PostgreSQL[9] Database where to store the results sent by the landmarks after a measurement campaign. In order to start a campaign the user has to fill a form in a web interface (see Section 3.6) that is responsible for sending the requests to the landmarks. This is done by GCM which, in a best effort fashion, tries to deliver the message to all connected smartphones, as we already described, and then notifies the user that the operation was a full success or if some device has not been reached. When a landmark receives a message from GCM it controls if the measurements can take place (i.e. if there are no tasks already running) and, if so, it starts taking the measurements. This is done in background without notifications, as we decided not to bother the smartphone user every time the service starts. When the target list is empty

[8]The Packet InterNet Grouper (Ping) network tool is used to measure the RTT between the landmark and the target

[9]*http://www.postgresql.org/*

and the measurements are completed the landmark automatically sends back the results to the server, which stores them on a Database. Results can be examined later via a web interface.

3.3 Client Apps

The PORTOLAN Android App is responsible for executing the microtasks sent by proxies after the polling operation and to send the results back to the server when the execution is terminated. As we introduced in 3.1, besides the measurements, this app provides differents services to the users to motivate them to contribute to the project: a traceroute and a ping tool, a maximum throughput estimator, a WiFi scanner and a mobile network signal analyser. In this section we introduce the principles of the software and its modules.

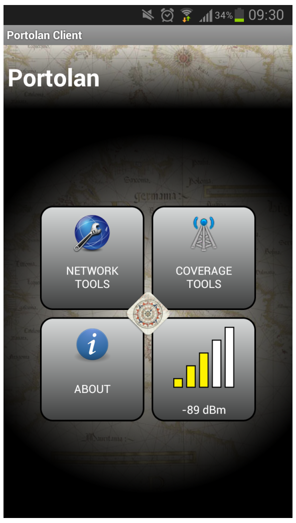


Figure 3.5: Home Screen of the PORTOLAN App

The Android app runs background measurements, i.e. traceroutes and pings, and retrieves data on the signal coverage of cellular networks. Since we do not want to bother the user, the Internet graph measurements are limited to max 200 traceroutes per day, i.e. peak traffic rate $\leq$1 KB/s and average traffic rate $\leq$ 2 MB/day. Signal coverage measurements does not cost on battery or bandwidth, as the Portolan app passively collects signal strength samples using GPS positions generated by other apps. Since the geolocation module is still a prototype, even if the user is unaware of the measurements, we did not put limitations on the max numbers of RTT measurements, or on the maximum traffic rate. The application stops its background activity when the battery level decreases under 40%. No personal data are sent to PORTOLAN server; thus, it is not possible to connect any measurement to the device, or the user identity, that performed them.

Traceroute The smartphones execute the Internet topology analysis via *Traceroute*. The approach is the one provided by MDA (Multipath Detection Algorithm), an evolution of Paris traceroute that provides an exaustive research of every possible path between a source and a destination. Paris Traceroute is able to detect the presence of load balancers by implementing heuristics to recover from their effects. However, Paris Traceroute discovers only one of the available paths from a source towards a destination, this is why we use MDA (for more informations on MDA refer to [21]). In addition to the traceroute tool used by the device when a task is requested, there is a manual service available for the user in the network tools activity. Another available tool is the AS Traceroute, or ISP Traceroute, wich, instead of returning the list of crossed IPs, like classic traceroute, lists the traversed Internet Service Providers (ISPs) in the path to the target. In Fig. 3.6 the steps of a traceroute task are showed.

Ping The manual ping tool employs the Android native ping command and offers a simple graphic interface to the user.

Maximum Throughput Estimation This module let the user get an estimate of the maximum throughput that can be achieved along an Internet path whose end points are the client that starts the measurement and the PORTOLAN Server. In order to get the estimate the client performs a defined set of attempts. In particular, it implements 20 attempts in download and 20 in upload. Moreover the tool provides the trend of experienced throughput for each attempt in form of graphical plots. As the measure is an estimation, if there is an high packet loss during the measurement it is possible that the results may differ from the actual values.

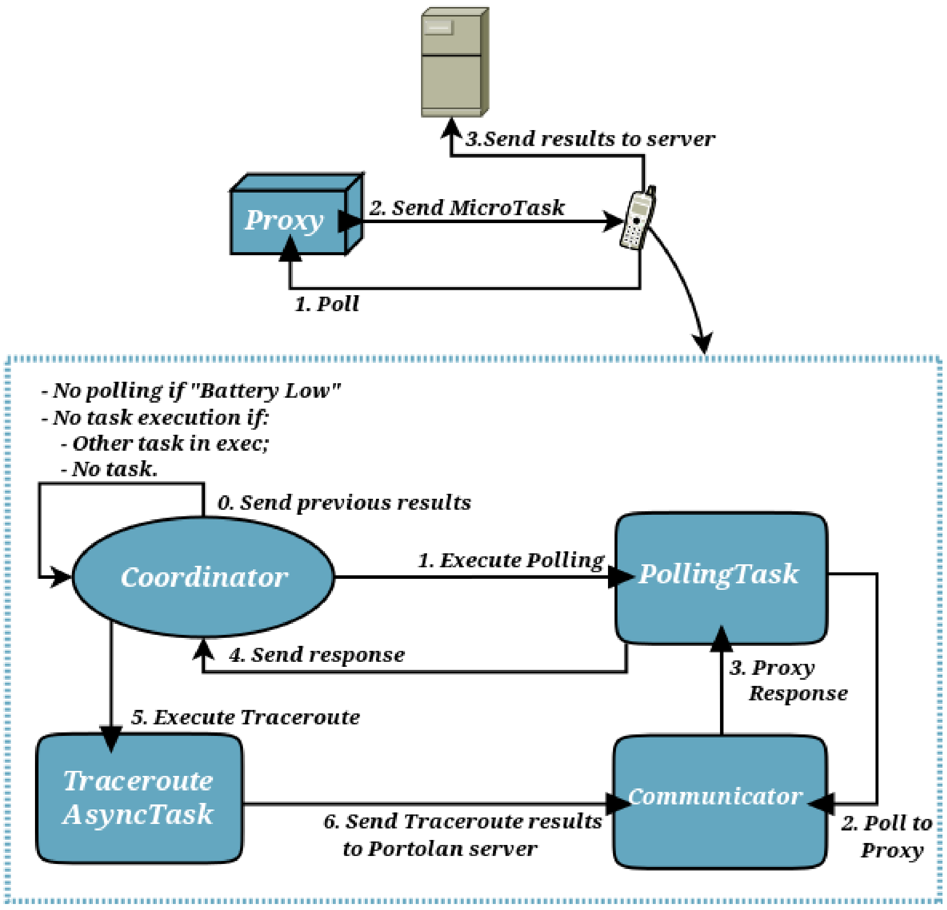


Figure 3.6: Example of Traceroute execution

WiFi Scanner This service scans all availables Wireless LANs in range and shows a list of them, specifying the network signal strenght and the type of authentication required. It is also possible to exclude those networks that require authentication from scanning.

GSM/UMTS Coverage This tool is used to test the GSM/UMTS signal strength and to get informations about cells of the smartphone operator. Also it is possible to track the signal coverage along a path. The results are then collected on a layer of Google Maps[10] that can be checked online.

In the next section we show how the geolocation service works in the Android app, and how it coordinates with the existing tasks.

3.4 Geolocation measurements

In this thesis work we designed and developed a geolocation system both on the server side and on the client side. In this section we introduce the operations executed on the Android client app. More information about the implementation can be found in chapter 4.

The measurements of geolocation tasks are completely invisible to the user, since we don't want the user to be bothered by any notification. The fundamental steps of the geolocation service are:

1. The landmark receives a geolocation campaign request by GCM. The request is processed, and if possible, the geolocation service is started;

2. The geolocation service starts a traceroute task towards the target then, when the traceroute is finished and the results are stored, it starts the RTT measurements. We use traceroutes in our geolocation module to have a more complete view over the possible paths taken by the RTT measurements towards a target;

3. When the measurements are over, both the traceroute and the RTT measurements results are sent to the PORTOLAN server.

[10] *http://portolan.iet.unipi.it/outcomes.html*

To describe the measurements we follow the steps of the geolocation in the same order.

Step 1: Receiving the request As we mentioned in 3.2.1 the GCM service is used in the original system architecture to send urgent tasks to clients. In order to send a geolocation campaign task to all the landmarks registered with the system we decided not to pass through the SPS, like in the traceroute tasks. This happens because we give the user the opportunity to perform synchronyzed measurements, i.e. all landmarks receive the task at the same time. In brief, when the landmark is in an idle state it waits for messages sent by GCM; this does not bother the polling service served by the coordinator, as GCM is based on *Intents*. In Android, an Intent is an abstract description of an operation to be performed; it is used for performing late runtime binding between the code in different applications (see [**?**]). When a message arrives from the GCM the application starts an *IntentService*, that is used to handle the operations described by the Intent. Clearly the landmark has to be registered with the GCM service, or it will never receive messages. When a message arrives, the landmark process it as it is possible that the operation contained in the message is relative either to an urgent traceroute or a geolocation campaign. If it is the case of a geolocation campaign request, the landmark collects the list of target to be located, the number of measurements, the waiting time between these measurements and other option that are mostly useful for testing purposes. All this parameters are inserted by the user via the web interface developed in the server side (see Section 3.6).

Step 2: Taking the measurements The geolocation service in the Android app is the heart of the entire process. Its purpose is not to give directly to the user the geolocation of a certain target but to collect a great number of RTT measurements through the execution of *pings* toward the target.

PING is a computer network administration utility used to test the reachability of a host on an IP network and to measure the round-trip time (RTT) for messages sent from the originating host to a destination computer. The name comes from active sonar terminology which sends a pulse of sound and listens for the echo to detect objects underwater. PING operates by sending

Internet Control Message Protocol (ICMP) echo request packets to the target host and waiting for a response. In the process it measures the time from transmission to reception (the RTT) and records any packet loss. The Ping tool is normally present in all modern operating systems but in Android you must define a function that runs a native unix command in order to execute it. Moreover, to modify the behaviour of the probes sent by the Ping command you must be a privileged user. Due to this probems we use the same traceroute function developed inside the PORTOLAN project with different options that simulates the behaviour of the original Ping tool.

We need a great number of pings because we use them to calibrate the SPOTTER service (it uses fixed landmarks), which then will give us the geolocation results on a certain target, or a list of targets. As we already said, not only pings, but also traceroutes measurements are carried out in order to have a more complete view on what happen to the IP packets sent towards the target.

In the last version of the application, when the system has examined the list of targets and all the parameters passed by the GCM, it select the first target and start a traceroute. After the traceroute ended (if there were network errors, the system tries to execute again the traceroute), the system store the results in the phone memory and start the execution of the RTT measurements, in the form of pings. The pings are executed as threads, i.e. every ping is executed by a thread that saves the results of its ping in memory. Then, when all pings are executed, i.e. when all threads are done, the results are combined in a string that will be then sent to the PORTOLAN server.

Step 3: Sending the results When the measurements are over the geolocation service takes the results collected in the landmark internal memory and modifies the format to prepare an HTTP *Post* to PORTOLAN Server. Both the traceroute and the pings are sent in the same string; it is the server duty to split the message in two parts and save the results to its database. If there are network failures, like timeouts on the server side or connection losses on the landmark, the service tries to send the results for three times; if even after these attempts the system fails to send the results, the burden is passed to the Coordinator service which every 60 seconds wakes up and controls if there is data that needs to be sent to the PORTOLAN server.

In the next section we describe the structure of the database used to collect the geolocation results.

3.5 Portolan Geolocation DataBase

In this section we focus on the way RTT measurements are collected by the PORTOLAN system. The results of all measurements campaign are sent from the landmarks to a Servlet via HTTP *Post*. The servlet then processes the information and stores them in a PostgreSQL Database. In the following we describe all the tables that were created in the DataBase.

geolocationPing Table see table 3.1

The geolocationPing table is used to collect the RTT measurements, taken via the geolocation service. The information stored on this table will be used to execute the probabilistic delay-distance model described in 2.2.1.

Field	Type
cid	*Char Var* NOT NULL
pid	*Char Var* NOT NULL
burst	*Integer* NOT NULL
nseq	*Integer* NOT NULL
target	*Char Var*
tstamp	*Tstmp w t zone* NOT NULL
coordinates	*Char Var*
accuracy	*Double Precision*
rtt	*Double Precision*
bsid	*Char Var*

Table 3.1: GeolocationPing Table

- *cid*: The ID used for the measurements campaign. It is provided by the server, based on the information entered by the user on the web interface (*key*); e.g. *test2013-04-19-09:41:31*

- *pid*: stands for PhoneID, it is assigned by the proxy assigner and has an univocal correspondance with the landmark (*key*); e.g. *DID13672382925552084382417001714*

- *burst*: when > 1 is used to test the service on the landmark. The first burst of pings is the only one used in the normal execution while the others can be set up as a countercheck to verify the results of the first one (*key*);

- *nseq*: Sequence number of pings in a burst [0 ; MAX_PING-1] (*key*);

- *target*: The target we want to geolocate (*key*); e.g. *193.206.142.81*

- *tstamp*: Timestamp stored when the ping starts; e.g. *2013-04-19 09:41:31.522+02*

- *coordinates*: The landmark coordinates (*format: LAT LONG*); e.g. *43.7157638 10.3955942*

- *accuracy*: The accuracy of the positioning system for the landmark (*in meters*);

- *rtt*: The RTT taken with the precision of microseconds; e.g. *109.192000031471*

- *bsid*: Stands for Base Station Identification. In case of a mobile network we collect the BSID of the BTS, so we can use an online Database to locate the BTS. In the case of a Wifi network we have "*wifi*" written on this field instead of the BSID. e.g. *270060603605488222*

Traceroute Table (See Table 3.2)

This table is used to collect informations on the traceroutes executed by the geolocation service. As previously stated, we use these information mostly to have a more complete view over the possible paths taken by the pings towards a target. Every row with the same tuple (*CID*,*Target*,*PID*) represents a hop of the same traceroute campaign.

- *CID*: The ID used for the measurements campaign (*key*);

Field	Type
CID	*Char Var* NOT NULL
Target	*Char Var* NOT NULL
Timestamp	*Tstmp w t zone* NOT NULL
Hop	*Integer* NOT NULL
Interface_IP	*Char Var*
NextHop_IP	*Char Var*
RTT	*Double Precision*
Skip_TTL	*Integer*
AS_number	*Char Var*
PID	*Char Var* NOT NULL

Table 3.2: Traceroute Table

- *Target*: The target we are tracing (key);
- *Timestamp*: Timestamp stored when the traceroute starts (key);
- *Hop*: It is a progressive number that specifies the hop (key);
- *Interface_IP*: The source Interface at the specific hop;
- *NextHop_IP*: The destination Interface at the specific hop;
- *RTT*: The RTT taken between *interface_IP* and *NextHop_IP* interfaces, with the precision of microseconds;
- *Skip_TTL*: The number of routers that did not respond to traceroute request before the specific hop;
- *AS_number*: It refers to the Autonomous System number of the source interface specified in the *Interface_IP* field at the specific hop;
- *PID*: stands for PhoneID, it is assigned by the proxy assigner and has an univocal correspondance with the landmark (*key*).

Network Table (See Table 3.3)

This table is used by the server as a support to store informations on the Autonomous System on which the landmark is connected. Moreover is used to distinguish the access technology used by the landmark; this is important because we experienced that the RTT variations behaviour differ accordingly

Field	Type
CID	*Char Var* NOT NULL
Access_ASnum	*Char Var*
Net_type	*Char Var*
PID	*Char Var* NOT NULL
Timestamp	*Tstmp w t zone* NOT NULL

Table 3.3: Network Table

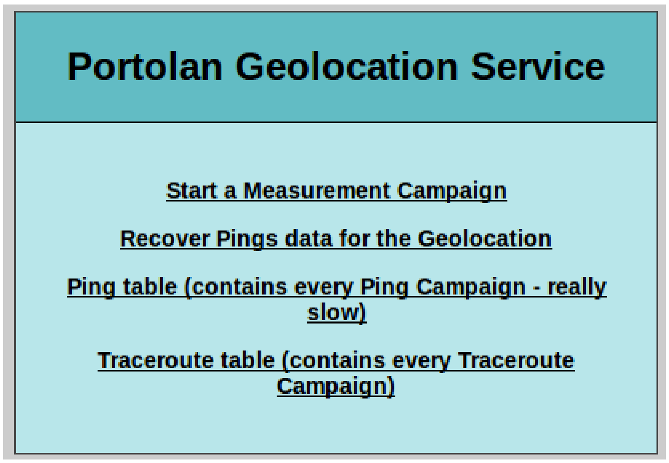
to the technology used (we will describe this phenomenon in depth in chapter 5).

- *CID*: The ID used for the measurements campaign (*key*);
- *Access_ASnum*: AS number of the ISP which the landmark is connected to;
- *Net_type*: The tecnology used by the landmark when the measurements were taken. The possible technologies are:
  - WiFi, in case of a Wireless network;
  - GSM, EDGE, GPRS, UMTS, HSDPA, HSUPA, HSPA+, LTE, in case of a mobile network.

  The *Net_type* field is used to distinguish between technologies when the user wants to retrieve the geolocation data via the web interface (we'll see this in details in section 3.6);
- *PID*: stands for PhoneID, it is assigned by the proxy assigner and has an univocal correspondance with the landmark (*key*);
- *Timestamp*: Timestamp stored when the AS number is found (key);

3.6 PortolanSpotter Web Interface

In this last section of chapter 3 we present the interface developed during this thesis work whose aim is to trigger measurements on smartphones and to retrieve collected data. At the end of this section we'll be able to show a complete vision of the structure of PORTOLAN system.


Portolan Geolocation Service

Start a Measurement Campaign

Recover Pings data for the Geolocation

Ping table (contains every Ping Campaign - really slow)

Traceroute table (contains every Traceroute Campaign)

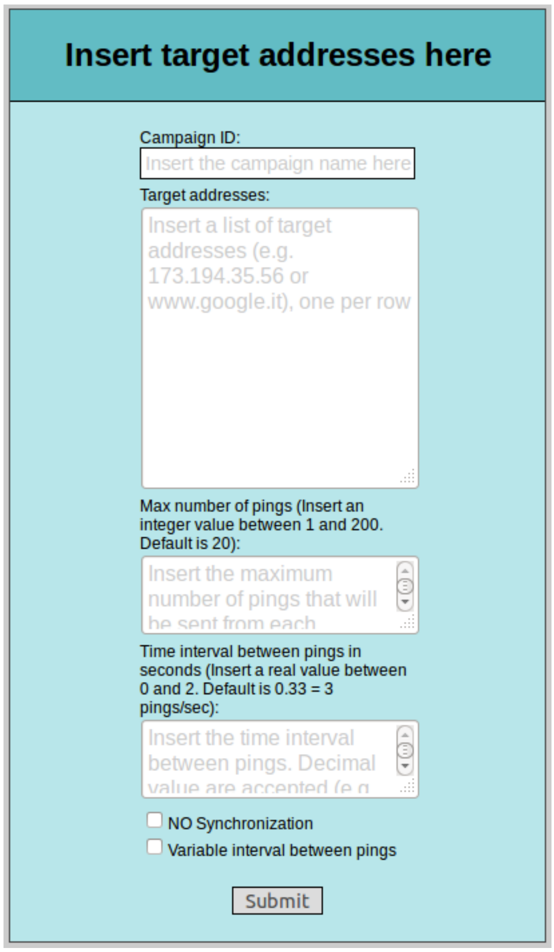
Figure 3.7: Portolan-Spotter web interface index

As you can see in Fig. 3.7, the web interface offers the following options:

- *Start a Measurement Campaign*: this link takes the user to a page where he can specify the parameters nedeed to start a measurement campaign that will be sent to every landmark registered to the GCM service;
- *Recover Pings data for the Geolocation*: through this link the user can gain access to the results of an entire campaign, after having filled a form. These results are used then to calbrate SPOTTER;
- *Ping Table*: This link takes to a page that shows every ping ever stored in the *geolocationPing* Table described in 3.5;
- *Traceroute table*: This link takes to a page that shows every traceroute ever stored in the *Traceroute* Table described in 3.5.

In the following we describe in detail the first two options.

Start a Measurement Campaign When starting a new campaign the user has to provide an identifier, which has to be used subsequently to retrieve the data of such campaign (it will be used as the *cid,* see 3.5). In addition, the user has to specify the target (or a list of targets) of the campaign and, if he want, he can modify the number of RTT measurements (number of pings) for that campaign and/or the time interval between pings. The system awakes the landmarks, which start pinging the specified target. Once finished, the results


Insert target addresses here

Campaign ID:
Insert the campaign name here

Target addresses:
Insert a list of target addresses (e.g. 173.194.35.56 or www.google.it), one per row

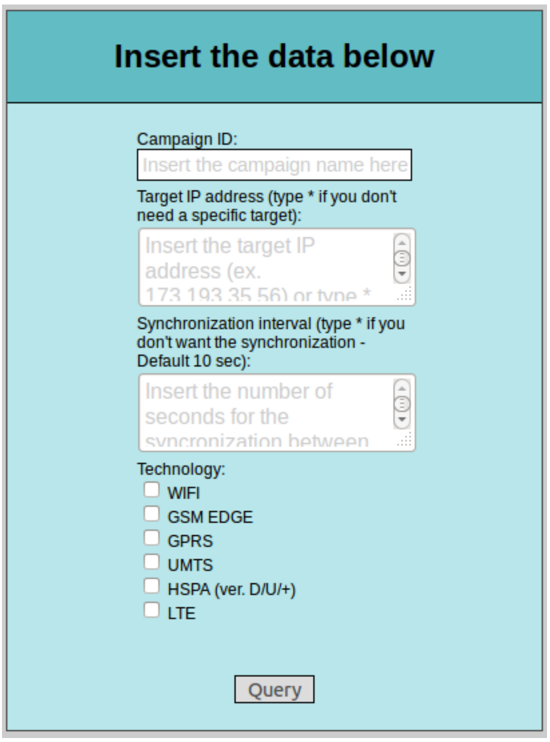
Max number of pings (Insert an integer value between 1 and 200. Default is 20):
Insert the maximum number of pings that will be sent from each

Time interval between pings in seconds (Insert a real value between 0 and 2. Default is 0.33 = 3 pings/sec):
Insert the time interval between pings. Decimal value are accepted (e.g.

☐ NO Synchronization
☐ Variable interval between pings

Figure 3.8: GCM Message Sender

of the measurement campaign are stored into a database. Awakening of mobile devices is performed in a best-effort fashion as already specified in 3.2.2, because of GCM peculiarity. The system gives the possibility to *s*ynchronize the measurements, i.e. the landmark probes the same target all together. It is important not to ignore the fact that if the number of PORTOLAN clients increases, then the probing process may be recognized as a DDoS attack by the targets. Instead, if the *NO Synchronization* flag is set, the landmarks waits for a random time (the range is [1 sec; 15 sec]) before starting the geolocation service. In addition to that the user can set a flag to let the landmark use a variable time interval between pings, e.g. instead of using a fixed time interval of 1 second, the landmark decreases the interval of 1/MAX_PINGS



Insert the data below

Campaign ID:
Insert the campaign name here

Target IP address (type * if you don't need a specific target):
Insert the target IP address (ex. 173.193.35.56) or type *

Synchronization interval (type * if you don't want the synchronization - Default 10 sec):
Insert the number of seconds for the syncronization between

Technology:
- [ ] WIFI
- [ ] GSM EDGE
- [ ] GPRS
- [ ] UMTS
- [ ] HSPA (ver. D/U/+)
- [ ] LTE

Query

Figure 3.9: Query to DataBase

seconds at each ping[11]. Not all landmarks are always available: some of them may be turned off or they could be in a no service area. Once the campaign is started, the system will display a new screen with the status of the request and the number of involved devices.

Recover Pings data for the Geolocation Through this form the user can retrieve data from the database described in section 3.5. There are differents options available, in depth:

- *Campaign ID*: this field is used to insert the search key, i.e. the *cid*, given by the system when the user starts the measurement campaign;
- *Target IP addresses*: In the second field the user can insert a specific target or the character * in case he want to retrieve the results generated by all the targets for that specific campaign;

[11]MAX_PINGS represents the maximum number of pings executed by the landmark on a geolocation campaign. It can be changed by the user (the default value is 20).

- *Synchronization interval*: As mentioned, awakening of landmarks is a best-effort process. This means that some devices may start sending probes with some delay. Through this field, the user can specify the size of the synchronization window (in seconds): the system will return the largest set of results that fits within the specified interval. The default value for the synchronization interval is 10 seconds. If the user set the flag *No synchronization*, when a new campaign is submitted, the synchronization window is not needed. To remove the window the user has to insert * in the text box.

- *Technology*: since the landmarks are distinguished by the technology used to connect to their ISP, as described in section 3.5, it is possible to filter the results on the base of the communication technology used by the landmarks when they performed the measurements (WiFi or Mobile network subtypes).

The results given by this query are in the form of a table, composed by fields of the tables described in 3.5. The fields are the following: cid, *pid, burst, nseq, target*, *tstamp*, *coordinates*, *accuracy*, *rtt*, *bsid*, *net_type*, .

Since we described at a high abstraction level the features of PORTOLAN system, we are now able to resume these aspects in a schematic way (in fig.3.10 you can see the full structure of the system).

If you want to examine in depth the aspects concerning the tracerouting or the control plane of PORTOLAN system we recommend [2, 3]; on the other hand, in Chapter 4 of this Master's thesis we present the implementation of the geolocation service at a more accurate abstraction level.

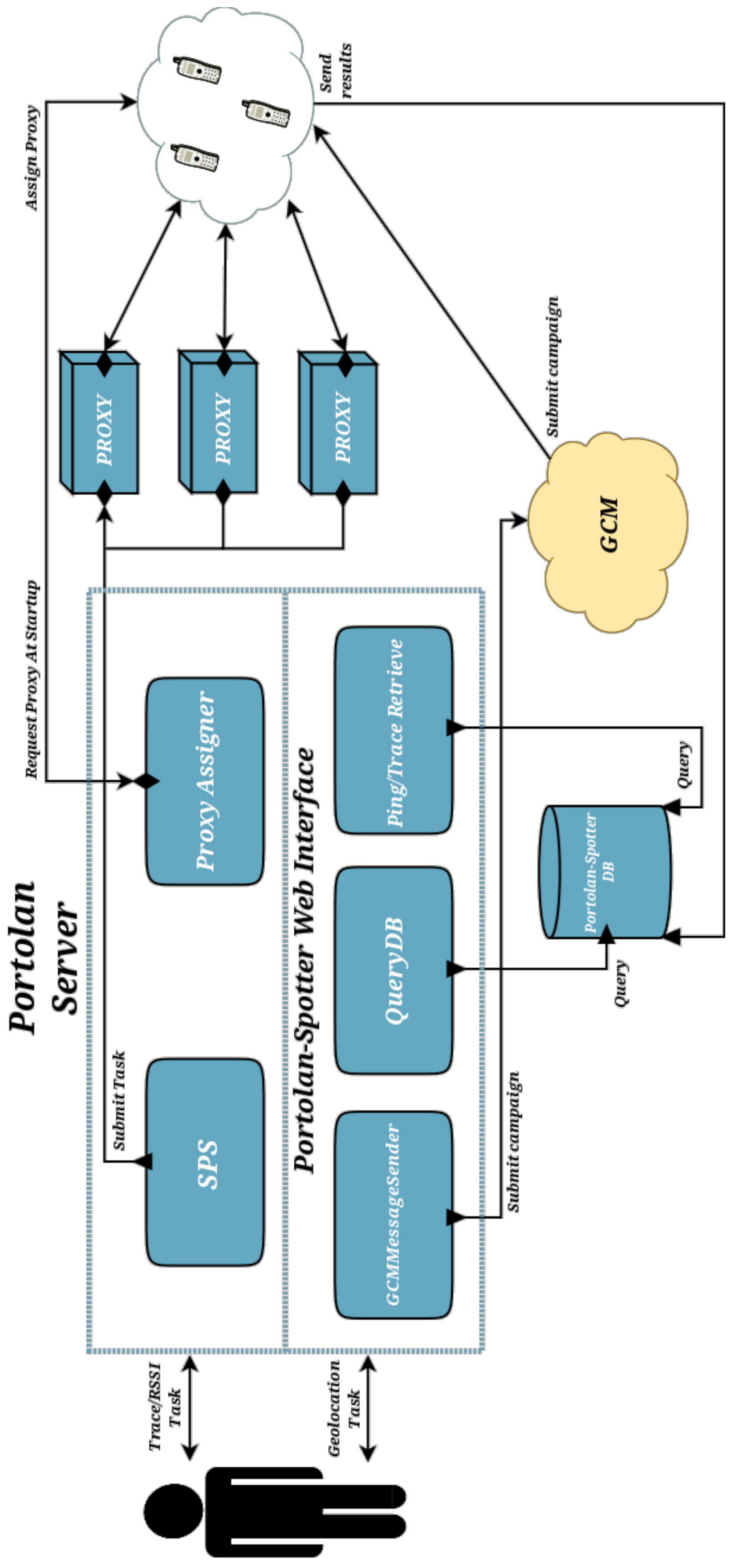


Figure 3.10: The architecture of the PORTOLAN system

4 Implementation

In chapter 3 we discussed the PORTOLAN system design, starting from the components implemented before this thesis work, in [2, 3], to the newly introduced services. We gave an overview of the main components of the system and their interactions at a high level of abstraction. In this chapter we provide a detailed description of system modules developed for the geolocation service. In the first section of this chapter we summarize system modules and their features, and show which of them were implemented in this thesis work. Then, in the following sections, we explain in detail each one of the modules implemented in this thesis work, subdividing them in server and client components.

4.1 System structure

The PORTOLAN system has a complex, yet modular, architecture that can be separated into two macro area:

1. PORTOLAN Server
    - SPS module
    - Proxy Assigner
    - Portolan-Spotter module
    - Proxies
    - GCM
2. PORTOLAN Android Client
    - Traceroute module
    - RSSI module
    - Maximum Throughput Estimator module
    - WiFi Scanner module
    - Geolocation module

In this thesis we focus on the Portolan-Spotter module and the Geolocation module. For a detailed description of the other modules of the PORTOLAN system refer to [2, 3].

Geolocation Measurements features the geolocation module or, to be precise, the Portolan-Spotter module does not rely on SPS like the other modules (see chapter 3). The human user inserts the requests in a web form. The module passes the burden to GCM, which then sends the tasks to landmarks. When landmarks complete measurements, they send back the results to the Portolan-Spotter module, which analyses and stores them on a Database.

The clients are responsible for the execution of tasks, whichever task they receive. If it's the case of Internet analysis they poll their assigned proxy for receiving a microtask to execute, but if the task concerns geolocation measurements they wait for GCM messages. When execution is finished, in the first case they send data to their proxy and, finally, they notify it about task completion; while in the second case they send the results to the Portolan-Spotter module.

In the next two sections we concentrate upon the following modules implemented in this thesis work, which are the ones concerning the geolocation measurements; more precisely:

- The Portolan-Spotter module which is responsible for preparing RTT measurements campaign, inserting measurements results into a database and retrieving these results from the same database;
- The landmarks functionality to take measurements, connect with GCM and with the server;
- All changes on previously developed code to avoid errors and to integrate the geolocation service with the other implemented modules.

Clearly, the programming language used to implement these modules is Java[1]. In particular, the server side modules are Java Servlets[2] that are runned by an Apache Tomcat web server[3], the database management system used is PostgreSQL[4], while the client side is developed using the Java Android API 16[5]. Refer to the official documentation in [22, 23, 24, 25, 26] for a detailed description of the methods, the classes and the libraries specified in the next sections.

4.2 Portolan-Spotter module

As explained in chapter 3 the architecture of PORTOLAN server has been modified to let the user control the landmarks in order to make them measure RTTs delays towards a target or a list of targets. As you can see in Fig. 4.1 four modules have been added:

[1] *http://www.oracle.com/it/technologies/java/overview/index.html*
[2] *http://www.oracle.com/technetwork/java/index-jsp-135475.html*
[3] *http://tomcat.apache.org/*
[4] *http://www.postgresql.org/about/*
[5] *http://developer.android.com/about/versions/android-4.1.html*

1. *GeolocationResultInsert* is used by the landmarks to send the results of a geolocation campaign to the Portolan-Spotter Database;

2. *GCMMessageSender* is responsible for retrieving the data inserted by the user in the web interface presented in section 3.6 par. "*Start a Measurement Campaign*", finding the android terminals that are registered with GCM and then preparing a message that will be sent to landmarks via GCM;

3. *QueryDB* uses the information inserted by the user in the web interface presented in the "*Recover Pings data for the Geolocation*" paragraph of section 3.6 to recover data, useful to calibrate SPOTTER, stored in the Portolan-Spotter Database;

4. *Ping/TraceRetrieve* is used to retrieve all pings or traceroutes stored in the Portolan-Spotter Database.

4.2.1 Geolocation Result Insert

This module is the link that connects the landmarks with the Portolan-Spotter Database. As we already stated, when the measurements requested by the user are completed, the landmarks send the results to the server which is responsible for analysing and storing the data. Since it is a Java Servlet, the module waits for clients incoming connections. The method used by the landmarks to submit the results is HTTP *Post*, even if the servlet is programmed to accept HTTP *Get* requests as well. The landmarks send the results in the form of a string; table 4.1 lists the *Post* parameters required to insert the data in the database (see table 4.2 to a list of the possible server HTTP response status). The most important parameters are *traceroute* and *ping*, since they contain information relative to the measurements taken by the landmarks.

In detail the format of traceroute and ping data is:

- *traceroute*: The character used to separate the triple (*CID*, *target*, *timestamp*) from the *n* hops of a traceroute campaign is the unix end-of-line character '\n', while the one used to separate every hop is the 'T'. To identify the end of the traceroute entry a second '\n' is used. Finally,

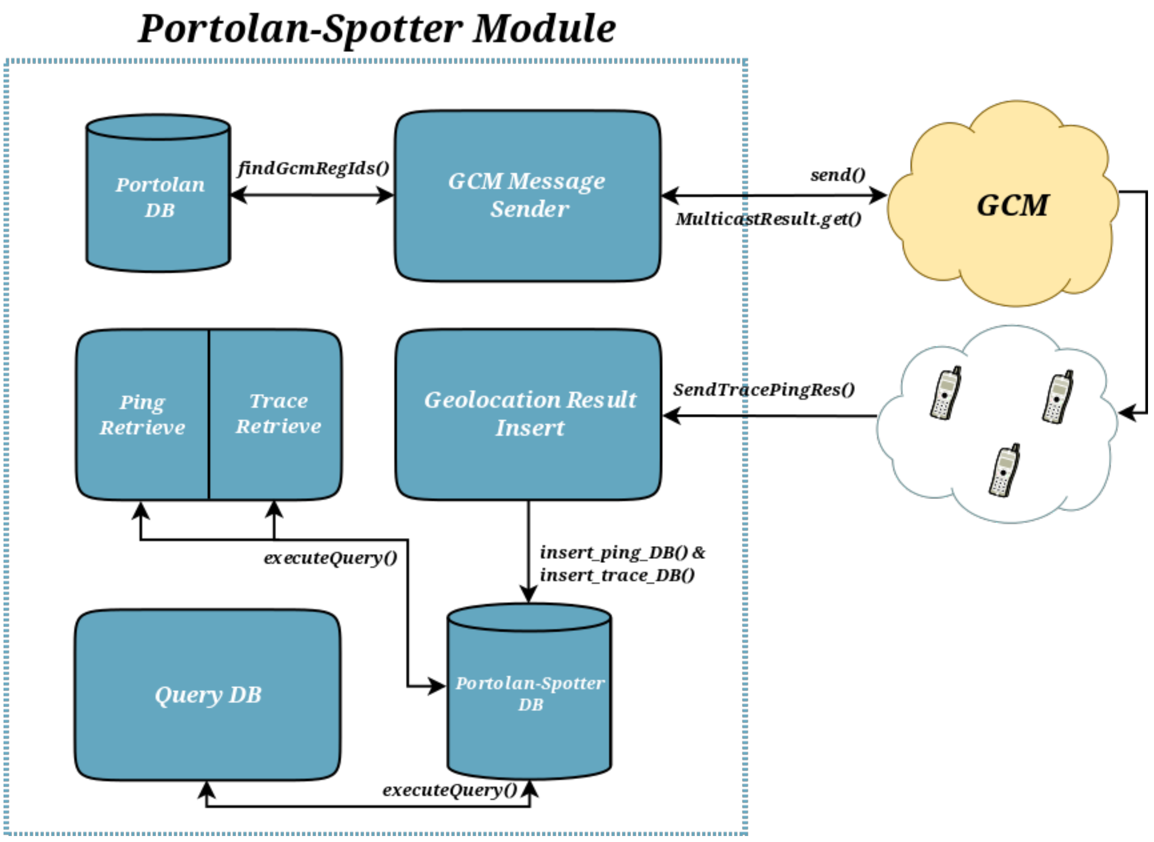


Figure 4.1: Portolan-Spotter module schematics

Parameter name	Parameter value
op	*what the message is about*
type	*type of the operation*
traceroute	see description in 4.2.1
ping	see description in 4.2.1
network	*One of the possible network types and subtypes*
pid	*The smartphone identifier, given by the server*

Table 4.1: HTTP *Post* parameters sent by the landmarks

Request	Response
Valid request	*HTTP 200 OK*
Invalid Request	*HTTP 400 Bad Request if no data received*
	HTTP 400 Bad Request if errors in op and type

Table 4.2: HTTP response status for the *GeolocationResultInsert* module

the field separator is the comma character. In section 3.5 are described the fields contained in a hop;

- *ping*: As already stated in section 3.6, the user can choose the number of pings that will be performed by the landmarks for a certain campaign. The character used to separate the pings is the '\n', while the fields of a single ping are separated by the comma character (for a detailed description of this fields see section 3.5).

Moreover, the server recovers from the landmark its IP address, using the *getRemoteAddr()* method, in order to find the number of the Autonomous System it belongs to; this information is used to fill the *Access_ASnum* field in the *Network* table of the Portolan-Spotter Database described in section 3.5.

After this the servlet fills the *geolocationPing*, *Traceroute* and *Network* tables, respectively. This task is fulfilled by two methods:

- *insert_ping_DB()*: the first operation executed by this method, in order to connect with the Portolan-Spotter database, is to load the PostgreSql JDBC driver. JDBC (Java DataBase Connectivity) is a Java interface that defines how a client can access a database independent from the database management system. In order to connect with the database, after having loaded the driver (the one used is the "*org.postgresql.Driver*"), JDBC provides the *getConnection()* method. This method needs the url of the database and its username and password. After initializing the DB connection, the servlet can filter the string content, using split methods; then, it executes the JDBC *executeUpdate()* method in order to insert the data in the *geolocationPing* table.

- *insert_trace_DB()*: the behaviour of this method is similar to the *insert_ping_DB()* method one. This method fills both the *traceroute* and *Network* table. In order to find the information relative to the Autonomous System number of a certain hop of a traceroute, a *whois [10]* command throught the use of *radb.net* database is executed on the source interface at every step.

After these operations the servlet ends its execution. From now on the information inserted in the campaign identified by the *cid* field are available for the user.

In the next section we present the servlet used to prepare a measurement campaign and send it to the landmarks.

4.2.2 GCM Message Sender

This module is used to manage all data inserted by the user when starting a new campaign. The servlet is used in combination with the "*Start a measurement campaign*" web interface where, as explained in Section 3.6, the user can specify the options that will be sent to the landmarks via the GCM service, in order to start a new measurement campaign. The parameters inserted by the user are listed in table 4.3, for a detailed description refer to par. "*Start a measurement campaign*" of Section 3.6.

The first operation done by the servlet is to prepare the connection with the PORTOLAN database, in order to search for landmarks registered with the GCM service. After these initialization operations the servlet gets the parameters inserted by the user and elaborates them in order to check if there are errors, i.e. the targets need to be one per row in order to be recognized by the system and the time interval between pings must be a number in the $]0;2]\,sec$ interval. If there are no errors in this step the servlet can prepare the

Parameter name	Parameter value
cid	*The identifier for the campaign*
targetList	*list of targets, one per row (IP or host address)*
pingsNum	*Number of pings sent per target*
timeIntvalBtwPings	*Pings rate*
nosync	*Synchronization between pings (Y/N)*
dinterval	*Variable rate between pings (Y/N)*

Table 4.3: GCM parameters that are sent to the landmarks

message that will be sent via GCM to the landmarks and it searches for the devices registered with the service in the *android_terminals* table in PORTOLAN database, using a method called *findGcmRegIDs()*.

To prepare and send the messages to the landmarks via GCM, the servlet make use of the *Message.Builder* and *Sender* classes of GCM API. Then the servlet waits for results, using the *MulticastResult* methods *getTotal()* and *getSuccess()*; these two methods returns respectively the total number of messages sent and the maximum number of messages that have been sent succesfully. Clearly, if the two methods return the same value, this means that all messages have been sent (*attention*: this does not means that all messages have been received by landmarks).

Depending on the results returned, the servlet finally displays a message to the user using the values returned by the *getTotal()* and *getSucces()* methods; moreover it specifies the *campaignID*, that is composed of: the *cid* inserted by the user and a timestamp, taken at the time of the request, appended to it. This information is used for searching the data for that specific campaign. In Table 4.4 a list of the possible response status returned by the servlet is showed.

Request	Response
Valid request	*HTTP 200 OK*
Invalid Request	*HTTP 400 Bad Request if there are errors in text*
	HTTP 400 Bad Request if parameters are missing
	HTTP 400 Bad Request if parameters are invalid
Server Error	*HTTP 500 Internal Server Error*

Table 4.4: HTTP response status for the *GCMMessageSender* module

In the next section we present the servlet used to make a query to the

Portolan-Spotter database in order to recover the data inserted by the landmarks.

4.2.3 Query DB

This module is used in combination with the *Query to database* web interface showed in 3.6 to recover the data relative to a certain campaign previously submitted by the user. As shown in table 4.5 there are several parameters that can be set up via the web interface.

Parameter name	Parameter value
campaignID	*The identifier for the campaign*
targets	*list of the targets, one per row (IP or host address)*
tech_X	*one or more access technology*
sync	*time interval of the sync window (in seconds)*

Table 4.5: Query parameters example

Not all the parameters are mandatory, except for the campaign ID, the target(s) and at least one type of communication technology. As the name of the servlet suggests, depending on the parameters passed from the web interface to it, a different query is executed. Before executing one of the queries the parameters passed are examined in order to set the response status of the servlet (see Table 4.6).

Request	Response
Valid request	*HTTP 200 OK*
Invalid Request	*HTTP 400 Bad Request if invalid campaign id*
	HTTP 400 Bad Request if invalid target or targets
	HTTP 400 Bad Request if no technology selected
Server Error	*HTTP 500 Internal Server Error*

Table 4.6: HTTP response status for the *QueryDB* module

The query changes heavily depending on the value of *sync* parameter; that is to say, if the syncronization is enabled the query takes the starting measurement time for all landmarks for a certain campaign and orders them in chronological order (using the *tstamp* field). Then these pings are splitted in synchronization windows of *x* seconds (the default value is 10 seconds) and it is examined the window in which the largest set of landmarks started the measurements; the query then returns to the user the pings data of these landmarks with their corresponding *phoneID*s. Clearly, the pings showed by the query must match the requirements established by the other parameters inserted by the user.

If, conversely, the *sync* flag is not enabled the query simply recovers all pings that match the other parameters (cid, target, tecnology).

In order to connect with the Portolan-Spotter database, the JDBC PostgreSql driver is used, in the same way described in Subsection 4.2.1. Since this query is a SELECT, the method used to execute the query is the *executeQuery()*. After the execution of the query the servlet prepares an HTML page where to show the results to the user and fills it with these data; then the communication flow is closed and the servlet ends its service.

4.2.4 Ping and Trace Retrieve

This module, composed by two servlets, is used to recover all geolocation measurements (pings and traceroutes), of all campaigns ever requested on all landmarks registered with the PORTOLAN system. The methods used to recover the information stored on the Portolan-Spotter database are the same already described in Subsection 4.2.3. The only substantial difference is given by the quantity of data that the servlet has to load. these two queries can produce very large result sets; if a *SELECT* statement returns too much data, memory can be exhausted. In order to bypass this problem JDBC provides the *setFetchSize()* method, which can be used to control the size of the buffer used by the driver when fetching rows; this will prevent memory problems.

4.3 Client modules

The landmark waits for incoming messages from the GCM. When a message is received, the landmark controls if it is in the right format, and that starts the measurements to send back to the PORTOLAN server. In the previous section we showed how the user can set up a geolocation campaign task and how he can retrieve the results sent by the landmarks when the task ends. Instead,

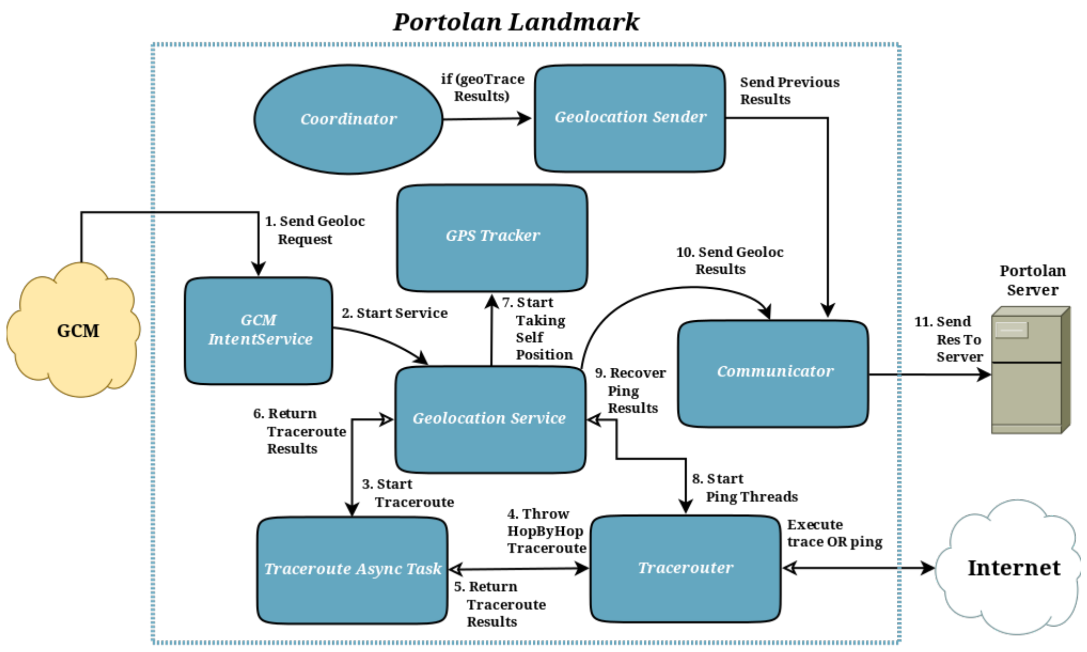


Figure 4.2: Client operation flow

in this section we show how things work on the clients side. As highlighted by figure 4.2, there are three new modules implemented specifically for the geolocation service and five modules that have been modified. In particular:

1. New modules:
   - *GeolocationService*;
   - *GeolocationSender*;
   - *GPSTracker*.
2. Modified modules:
   - *GCMIntentService*;
   - *TracerouteAsyncTask* and *Tracerouter*;
   - *Communicator*;
   - *Coordinator*.

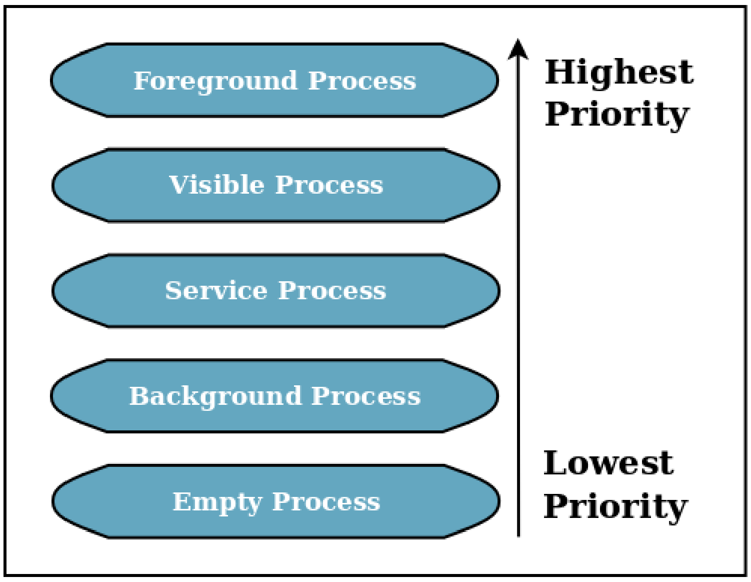


Figure 4.3: Android priority hierarchy

4.3.1 Geolocation Service

As already stated, this module is the heart of the entire geolocation process. For this reason this module should never have to be interrupted during the measurements.

In Android there is a particular applications priority policy due to the limited resources that a smartphone can provide. The Android system is responsible of ensuring that these limited resources are managed efficiently and that both the operating system and the running applications remain responsive to the user. The system tries to maintain an application process for as long as possible, but eventually needs to remove old processes to reclaim memory. To determine which processes to keep and which to kill, the system places each process into a priority scale based on the components running in the process and the state of those components; this hierarchy is showed in fig. 4.3.

In order to prevent the system to kill the Geolocation process during the measurements, we decided to make the service run in the foreground. More precisely, the service runs in foreground only for the time needed to perform the measurements, which can be variable, depending on the number of pings and targets and on the time interval between pings. After this initial clarification, we can present the algorithm implemented by the module in order to perform the RTT measurements:

1. the *GCMIntentService* module retrieves data from the Intent that started the service. The collected fields are: the list of targets (*targetList*), the identifier for the campaign *(cid*), a flag used to determine if the time interval between pings is constant or variable *(flagVariableIntval*), the maximum number of pings to send in a burst (*MaxPings*), the time interval between concurrent pings *(timeIntval*). When all fields are collected, the *GeolocationService* module is started:

   a) if the Intent is not null and the *targetList* string is not empty then the *GeolocationService* module:

      i. splits the *targetList* string in a List of targets[6];

      ii. shuffles the list of targets. If the user submits a target list and it is pinged from e.g. 100 landmarks, without some sort of randomization the landmarks go through the list sequentially in the same order and almost at the same time. Without this shuffling is possible that a target could consider the measurements as Ddos attacks [27].

   b) else the *GeolocationService* module is stopped.

2. the *GeolocationService* module starts a thread, using the method *newScheduledThreadPool()* of the *Executors* class, that is scheduled every *RoundIntVal*[7] milliseconds in order to manage the measurements towards the targets. This thread is scheduled once per target:

   a) The thread controls if there are targets to analyse: if so, it takes the first target in the list and sends information about this target to a BroadcastReceiver (*GeolocationReceiver)* specifying the option "*StarttraceOp*". After that it stops its execution. When the *RoundIntVal* expires the thread is started again by the *GeolocationService* module (restart from point 2).

   b) else, if there are no targets left, the thread stops the *GeolocationService* module.

[6]using the *java.util.LinkedList* class [22]

[7]*RoundIntVal = (MaxPings * timeIntval) * nburst + offset*

3. The *GeolocationReceiver* method *onReceive()* is triggered with the option specified in point *2.(a),* then the *TracerouteAsyncTask* module is executed in order to collect traceroute data towards the target:
    a) to be sure that the measurements are consistent, before and after the traceroute operation the *TracerouteAsyncTask* module controls if the network changed (if the measurements are performed within a WiFi network), or if the cell changed (if the measurements are performed within a mobile cellular network):
        i. if the connection did not change then the *TracerouteAsyncTask* module uses the *output_trace_ping()* method to build the string that will be sent to the PORTOLAN server, as described in 4.2.1;
        ii. else the *GeolocationReceiver* retries the operation starting from 3.(a):
            A. if for the second time the operation is not succesfully completed then skip to point 4, specifying that the traceroute has failed.
4. When the traceroute operation is completed, the *TracerouteAsyncTask* module sends to the *GeolocationReceiver* the results, using the option "*startpingOp*", and returns. Then the *GeolocationReceiver*:
    a) appends the triple (*cid*,*target*,*timestamp*) to the result string (as described in 4.2.1);
    b) gets the network type and subtype at the moment of measurements and save them in memory;
    c) starts the localization of the landmark using the *GPSTracker* module (the landmark coordinates are then stored in the *SharedPreferences*).
5. the *GeolocationReceiver* schedules a thread that executes every *timeInterval* milliseconds doing the following operations:
    a) if the number of bursts completed is less than the *MAX_BURST* constant value then:

   i. if the number of pings executed is less than the *MaxPings* value then starts a worker thread (*PingThread*) that will execute the operations needed to ping the target, that is:

      A. finds the base station identifier of the landmark if the *get_network()* method returns a mobile subnetwork;

      B. creates the socket for sending probes to the target. In order to avoid conflicts between different *PingThread*s the source port number of the socket is different for all of them;

      C. uses the *trace()* method of *Tracerouter* class (for details refer to 4.3.5) to send the ping towards the target;

      D. If the *trace()* method returns an *ICMP* echo reply "*Destination Unreachable - Port Unreachable*" the target has been reached. Thus, the *PingThread* creates the string described in the ping section of 4.2.1;

      E. puts the string in a *thread-safe*[8] data structure, then closes the socket stream and returns.

   ii. else waits until all threads have finished, then stores the pings results in the *SharedPreferences* and clears the *thread-safe* data structure.

b) else the measurements are completed, then the thread:

   i. if the target list is not empty stops the execution and returns to point 2;

   ii. else recovers traceroute and ping results from the landmark memory and sends them using the *sendTracePingRes()* method of the *Communicator* class (see 4.3.6):

      A. if the communication with *Portolan* server is interrupted then retries (max 3 times). If after the third time it is still

[8]*A piece of code is thread-safe if it manipulates shared data structures in a way that guarantees safe execution by multiple threads at the same time. In this case, multiple threads write in the data structure pings results in a dedicated memory location (identified by a numeric key).*

impossible to send the results then stops the *Geolocation-Service* module. It will be *GeolocationSender* responsibility to send the results later.

B. else if the results are succefully sent, stops the the *GeolocationService* module.

4.3.2 Geolocation Sender

This module is used to send previously obtained geolocation results. More precisely, the module is an Android *AsyncTask* started by the *Coordinator* class if there are geolocation data (ping and/or traceroute) that can be found in the shared memory of the landmark. An asynchronous task allows to perform background operations without using threads; it is an helper class that can be used to perform short operations. The *Coordinator* class is used, indeed, to coordinate the operations executed automatically by the landmark. Every *POLL_INTERVAL* seconds the coordinator, among other things, controls if there is something saved in the shared memory of the phone. If there are other services running or if there is nothing to do, like polling or sending results, the *Coordinator* stops and set its wake timer to trigger after *POLL_INTERVAL* seconds. If, instead, there are previous results in memory, the *Geolocation-Sender AsyncTask* receives from the *Coordinator* the information needed to send them. Then the *AsyncTask* is responsible of creating a string from the raw results in the format specified in 4.2.1 and then to send this string using the *sendTracePingRes()* method of the *Communicator* class. When the operation is done the phone memory is freed and the task is stopped.

4.3.3 GPS Tracker

This class extends the *Android Service* class and implements the *LocationListener* interface in order to request updates about the location of an Android phone. It is an helper class used by the geolocation service to retrieve the location of the landmark at the moment of the measurements. The service is started before taking the measurements and stopped after the last ping of the last burst of pings. Every *PingThread* (see 4.3.1) calls the *canGetLocation()* method before executing a ping, to control if the location can be collected. If so, then the thread collects the triple (*latitude*, *longitude*, *accuracy*) and appends it to the result string of the single ping; otherwise, the three fields are appended to the string as 0 values.

4.3.4 GCM Intent Service

This is the counterpart of the *GCMMessageSender* seen in Subsection 4.2.2. When a message arrives from the GCM the application starts the *GCMIntentService*, that handles the operations described in the message. This module has not been newly implemented but it has been modified from the original application; it was already present because not only the geolocation module but also the Internet topology analysis relies on the GCM service (an example is given by the *urgent* flag described in 3.2.1).

This *IntentService* extends the functionalities of the *GCMBaseIntentService*. First the landmark has to be registered to the GCM service through the *GCMRegistrar* class methods, executed when the application is started for the first time (after the user agreed with the disclaimer). Then, after the device has been registered, the *onRegistered()* method is called and its *registrationId* argument, returned by the GCM service, is collected by the system. This GCM*id* is used to identify the landmark with Google service. If an error occurs during the registration, it is handled by the *onError()* method. When the device is correctly registered with GCM service, every time a message arrives from the server, the *onMessage()* method handles the message. If another service is already running, which could be another geolocation campaign or a traceroute or the signal analysis, the operation is stopped and the *onMessage()* returns, else the system retrieves the *op* field from the GCM message:

- if the string content is equal to "*trace*" this means that an urgent poll is needed, so the *Coordinator* service is started [2, 3];
- else if the value is "*geo*" the message contains a new geolocation task to execute. In this case the system collects the information contained in the message, like the targets to ping or the id of the campaign (see table 4.3), and checks if there are errors in the values of these fields. At this point the system runs the *GeolocationService* but, if the *NO Synchronization* flag was set in the web interface described in 3.6, then it

postpones the service for a random time in the range of [1 ; 15] seconds, using the *AlarmManager set()* method.

4.3.5 Traceroute Async Task

The second module modified in order to work in the geolocation task is an *AsyncTask* originally created for the Internet topology analysis module. As it's name suggests this *AsyncTask* is used mainly to execute a traceroute towards a certain target: in this thesis we describe only the modification introduced to let the task work in the geolocation module, more information about other functionalities can be found in [2, 3].

The task is started by the *GeolocationService* module before taking the RTT measurements via the *Ping* tool; as previously stated we decided to use traceroutes to have a more complete view over the possible paths taken by the pings towards a certain target. When the *AsyncTask* is executed the arguments *mTr_Ping, mServer* and *mMda* are used by the system to determine if the traceroute is requested by the geolocation module or the Internet topology analysis module or else is requested manually by a user via the network tools of the PORTOLAN App; the first argument, *mTr_Ping,* is the only one that was added in this thesis work.

The most important modification produced to the normal functionalities of this class is the *output_trace_ping()* method which uses the results of the traceroute to build the *trace* string that is then sent to the server. Then, using the arguments specified before, if the traceroute is requested by the geolocation module the system sends back the string just created to the geolocation broadcast receiver implemented in the *GeolocationService* module. This way the geolocation module knows that the traceroute to the specified target is done and so can start the RTT measurements towards the same target.

TracerouteAsyncTask uses the *Tracerouter* helper class to execute the traceroute function implemented in C language and then integrated in the PORTOLAN app, through the Android Native library (*Android NDK*); the only modification introduced is given by the timeout (5 seconds instead of the previously set 10 seconds) of the probes sent by the system to find the various possible path taken by the pings towards a certain target.

4.3.6 Communicator

This helper class was introduced in the PORTOLAN App to group all the methods needed to communicate with the server side. Since the geolocation module needs to send the results directly to the Portolan-Spotter module in the PORTOLAN server we added a single method that does the work. This method, called *sendTracePingRes()* takes five arguments from the Geolocation module:

1. *traceres*: a string that contains the traceroute results given by the *TracerouteAsyncTask;*
2. *pingres*: this string contains all pings executed by the *GeolocationService* towards all the targets of a single campaign;
3. *net*: a string that contains the network type and subtype returned by the *get_network()* method. We remember that the value can be one of the following: WIFI, EDGE, GPRS, UMTS, HSPA*(*D/U/+*)*, LTE*;*
4. *server*: the address of the module of PORTOLAN server where to send the geolocation results;
5. *pid*: the *phoneID* of the landmark.

the operation and type fields inserted in the Http POST request to the server are described in table 4.1 of subsection 4.2.1.

The next chapter focus on the experimentation taken in this thesis work, together with SPOTTER authors, in order to validate the results returned by the RTT measurements collected by the PORTOLAN system.

5 Experimentation

In this chapter we want to show the various tests carried out within the PORTOLAN system in order to validate this thesis work on the Geolocation module. Moreover we want to build the hypotheses on the landmark selection algorithm that will be developed in the next future using the results and the discoveries of the experimentation on the designed module. We will describe two of the realized experiments that were particularly significant because they were useful to discover how the system reacts to specific network conditions. Section 5.1 is dedicated to the *Access Technology* T*est* that proves how in a wireless environment a significant delay is introduced in the RTT measurements, then section 5.2 describe the *Ping Interval Test* and the *week Test* used respectively to establish a treshold on the waiting time between consecutive pings executions an to discover if there are differences when the measurements are taken in a particular hour or day of the week.

5.1 Access Technology Experiments

The first significant measurement test was carried out because we wanted to see the responses of PORTOLAN system in different network environments, i.e in WiFi or mobile networks. We used only one landmark to do the test and it consisted in 1 Paris Traceroute (see 3.3) and 500 pings per target, where pings were separated by a time interval of 10 seconds each.

Data were collected in the three tables described in 3.5, with a difference: the *geolocationPing* table (previously called *Ping* table) presented two fields that are now deprecated, but at the time of the test were important to understand the behaviour of the system. These two fields are:

- *RTT_AT_TTL_1*: The RTT taken with the precision of microseconds just for the first hop of the measurement. We took this measurement because we noticed that the first hop in a WIFI/GSM/UMTS environment is the source of a long delay. As a first approximation, we may assume that the difference between the smallest RTT towards the final target and the smallest RTT associated with the first hop can be a reasonable metric for calculating the distance. In other words, we can assume that the first router is located next to the associated BTS (Base Transceiver Station) or WiFi access point and that the related landmark is close to the BTS or WiFi access point. Note that we are able to find the location of the BTS using the BSID (see 3.5) which is, thus, an approximation of the landmark position;

- *Ping_type*: As we already said, in the first versions of the implementation we made a distinction between pings at first hop and pings towards the target. To verify the problem arised in a wireless environment described in the *RTT_AT_TTL_1* field we developed two algorithms:
  - in the first one the two pings were executed sequentially (first hop, then target);
  - in the second one we implemented a concurrent execution where the two pings were represented by two threads executed at the same time in order to experiment the same network conditions;

The test consisted of four different measurement campaigns with different conditions:

1. HSDPA(3G) test with the italian mobile operator "*Wind Infostrada*";
2. HSDPA(3G) test with the italian mobile operator "Tim Telecom Italia";
3. WiFi test with the italian internet provider "Tiscali";
4. WiFi test within the GARR (Gruppo per l'Armonizzazione delle Reti della Ricerca) network, from the University of Pisa to the targets.

The targets were located in the same GARR network, and in particular in the cities of Bari (Puglia) and Palermo (Sicily).

The HSDPA(3G) tests revealed a high delay in the first hop. We thus measured this delay by sending a probe with ttl = 1 option[1] immediately after the ping sent towards the target; this way we have a high probability of being in the same traffic conditions. After this, we took the smallest RTT of the 500 pings and did the same with the RTT of the first hop and noticed that the difference between these two values is more similar to the RTT taken with the WiFi network. We can not know how threads are schedulated in an Android system, so there is no guarantee that a Ping is sent immediately or if there is a queue due to other applications sending data on the network (the same can happen for the ping response). We suppose then that using the difference between the smallest RTTs is a good way to *clean* the measurement from the delay of the technology.

Regarding the WiFi tests, the delay is much smaller (~1ms) than the one measured in 3G and this is because normally a wifi router (or an access point) is located near the mobile device. In a tipical WiFi environment the number of devices connected with the router is smaller than the number of devices connected to a BTS; this means that the router has to serve less devices than the ones served by a BTS, i.e the queues are smaller.

Another important cause of delay is the number of hops and the route taken by the pings; this is why we used Paris Traceroute. We know that the pings

[1] The TimeToLive value can be thought of as an upper bound on the time that an IP datagram can exist in the network. The TTL field, present in the IPv4 header, is set by the sender of the datagram, and reduced by every router on the route to its destination.

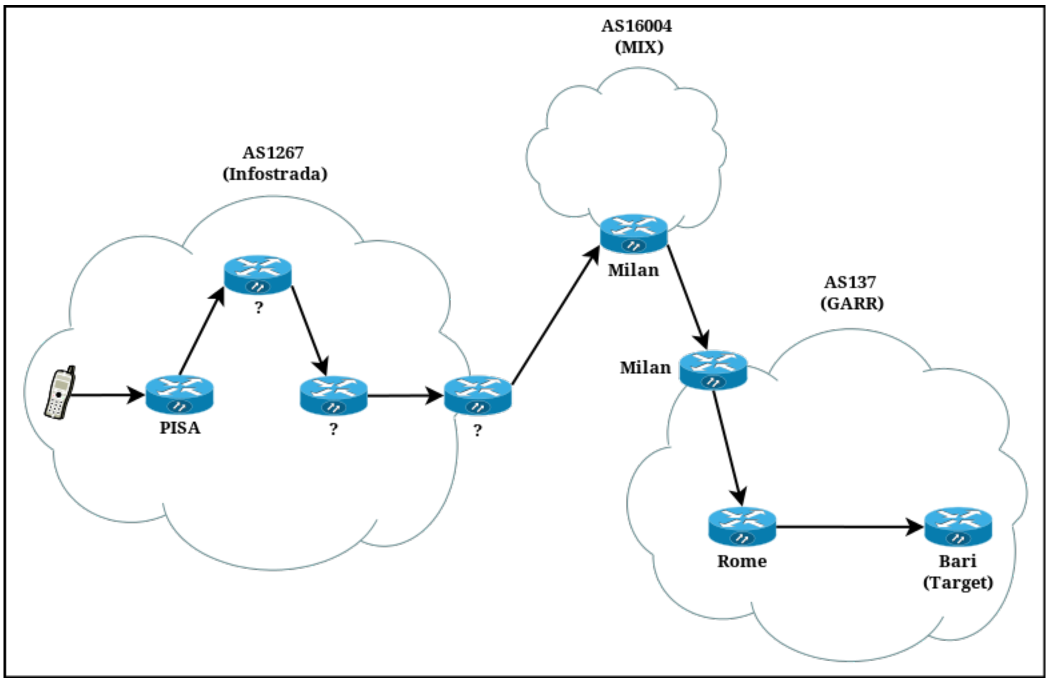


Figure 5.1: detail of the taken path on HSDPA_WIND campaign from Pisa to Bari

follow the same routes as the Paris traceroute launched just before the pings, with the same source and destination ports. In this test we found these conditions:

1. HSDPA_WIND campaign took 8 hops to the target located in Bari (see figure 5.1)

    - 4 hops in AS1267 (Infostrada). We could not find where the routers are located but we start from Pisa;
    - 1 hop in AS16004 (MIX) in Milan;
    - 3 hops in AS137 (GARR) Route: Milan -> Rome -> Bari.

    the trace for the target located in Palermo has the same first 5 hops. The differences are in the GARR network; 4 hops in AS137 (GARR) Route: Milan -> Bologna -> Palermo.

2. HSDPA_TIM campaign took 11 hop to the target located in Bari

    - 9 hops where we couldn't find the AS number;

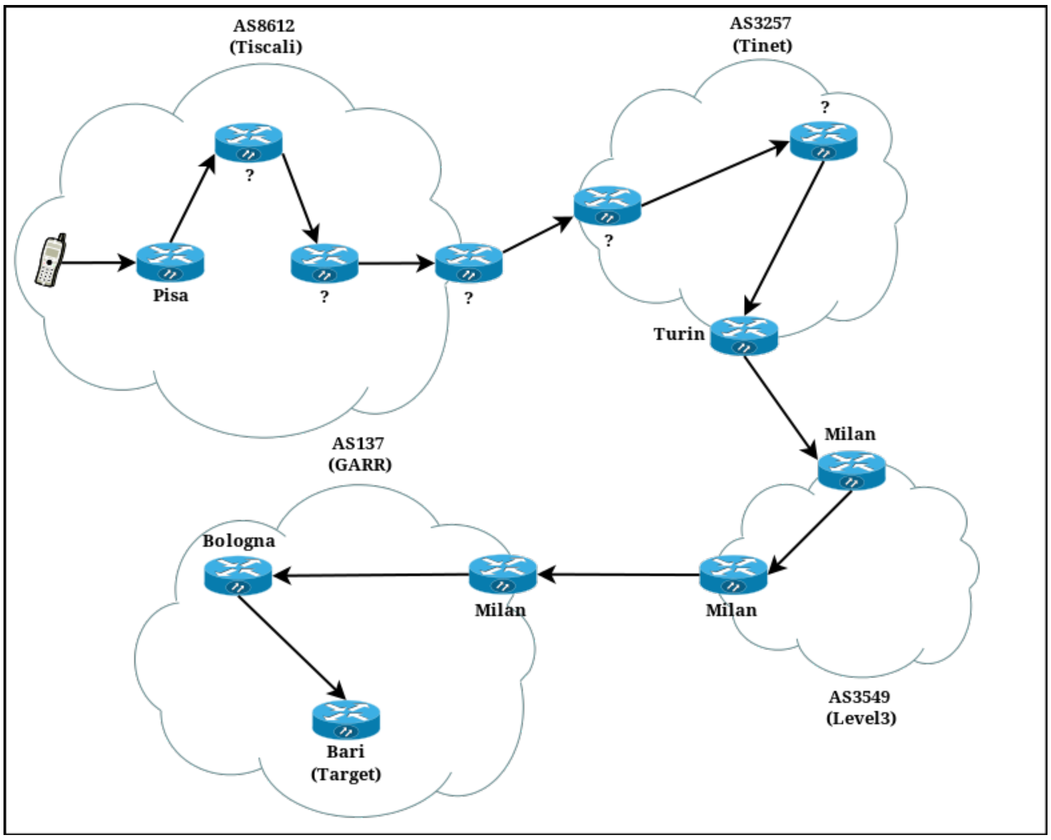


Figure 5.2: detail of the taken path on WIFI_TISCALI campaign from Pisa to Bari

- 1 hop in AS24796 (NAMEX) in Rome;
- 1 hop in AS137 (GARR) Route: Rome -> Bari.

the trace for the target located in Palermo took 12 hops and it has the same first 10 hops. The differences are in the GARR network; 2 hops in AS137 (GARR) Route: Rome -> Bologna -> Palermo.

3. WIFI_TISCALI campaign took 12 hops to the two targets; the difference is only in the hops in the Tiscali network where we can not know all routers locations (see figure 5.2)
    - 4 hops in AS8612 (Tiscali) from Pisa;
    - 3 hops in AS3257 (Tinet), according to the Looking Glass tool of Tiscali[2] one of the routers of the path, probably the last, is located

[2] *http://www.ip.tiscali.net/lg/*

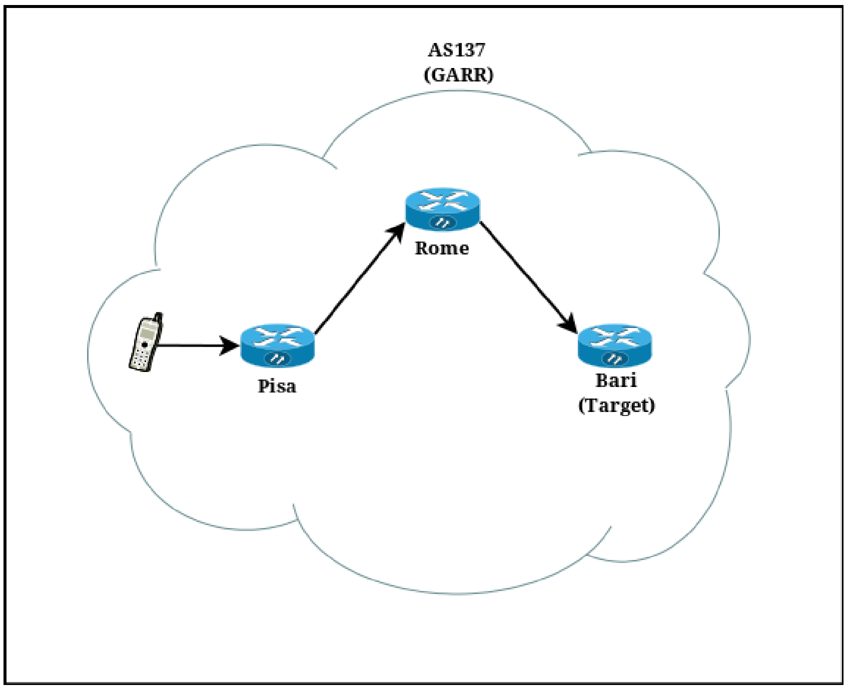


Figure 5.3: detail of the taken path on WIFI_UNIPI campaign from Pisa to Bari

in Turin;

- 2 hops in AS3549 (Level3) in Milan;
- 3 hops in AS137 (GARR) Route: Milan -> Bologna -> Bari/Palermo.

4. WIFI_UNIPI campaign took 7 hops to the target located in Bari and the same for Palermo. All of them are in the same AS (GARR) so we assume that with OSPF(Open Shortest Path First) the shortest route is taken (see figure 5.3)
    - Route to Palermo: Pisa -> Rome -> Bologna -> Palermo;
    - Route to Bari: Pisa -> Rome -> Bari;

The best results, i.e. the minimum delays, were found in the WIFI_UNIPI test; this is not unexpected since the landmark access technology used is WiFi and there are no BGP routes but only OSPF ones.

Test results The results of this test are extremely relevant because they are necessary to design an algorithm for the correct selection of landmarks in the geolocation process. As we already said in chapter 2, when describing SPOTTER web service, an algorithm for the filtering of landmarks, based on geographical and logical constraints, is of great importance to enhance the accuracy of the geolocation process.

Two major conclusions can be made from the intensive measurements campaign. In order to lower the delay of RTT measurements and then approximate the correct distance to the target it is a good idea to select landmarks:

1. in the same Autonomous System where the target is located or at least "near" the AS of the target (1 hop of distance between ASes). If this condition can be provided the protocol used to route the pings to the target will be OSPF, which tipically chooses the best path to a target located in the same Autonomous System (or if the landmark is in a neighbour AS we can assume that BGP will forward directly the IP datagram to the near AS);

2. connected to Internet via WiFi. This way the delay generated in the first hop is cutted down compared with the delay on a mobile technology (and also avoids operations like calculating the differences between RTTs to refine the measurements). Furthermore the position of the landmark connected with a wireless router (or access point) is less approximated compared with the position given by a BTS.

Table 5.1 summarizes the results of pings with statistical information about the pings of the 4 different campaigns (*minRTT*,*maxRTT*,*avgRTT*,*stddevRTT* and the same for the *RTTttl1*).

5.2 Ping Interval Experiments

In this section we show three experiments, composed by a series of test campaigns with different parameters, where each one was performed in order to find out the optimal sampling time between consecutive probes in a measurement campaign. We will describe the three experiments in the order of execution since they are interrelated. The experiments were executed from a single landmark towards the same target in all the campaigns, more precisely the target is an IP address of the GARR network located in Padova.

Cid	Target	NetType	minRTT	minRTTttl1	diffRTT	maxRTT
6480	193.206.142.81	WIFI_IET_UNIPI	16.6319999694824	1.79999995231628	14.8320000171661	805.817000031471
6480	193.206.137.201	WIFI_IET_UNIPI	52.9479999542236	4.24199998378754	48.7059999704361	904.572000026703
51237	193.206.142.81	WIFI_TISCALI	60.7300000190735	1.64800000190735	59.0820000171661	360.564999997616
51237	193.206.137.201	WIFI_TISCALI	59.6010000109673	1.98299998044968	57.6180000305176	362.458000004292
30253	193.206.142.81	HSDPA_WIND	80.933000087738	57.9839999675751	22.949000120163	970.641999959946
30253	193.206.137.201	HSDPA_WIND	89.6909999847412	38.7520000934601	50.9389998912811	1266.13899993896
6423	193.206.142.81	HSDPA_TIM	55.9380000829697	36.7439999580383	19.1940001249313	981.957000017166
6423	193.206.137.201	HSDPA_TIM	49.2549999952316	26.5499999523163	22.7050000429153	957.149000048637

maxRTTttl1	avgRTT	avgRTTttl1	stddev	stddevRTTttl1
939.331000089645	118.753394001245	119.630796000242	103.859992549343	120.099867495094
943.848000049591	192.716566004992	159.030032000303	162.51519172197	147.131897739308
165.801999986172	168.736256000519	4.85718399822712	88.3879791984374	10.2950730981975
6.7140000462532	114.109253998756	3.82083600008488	65.7482367750027	0.656877899547801
959.350999951363	395.522223994493	297.604481997013	252.941469880338	147.713391908248
958.921000003815	390.062566948073	258.443675214069	259.068071074178	141.574641952328
989.319000005722	378.47376999712	187.444597996235	225.792853999987	105.4831012947
800.804000020027	386.508387295926	185.695395494094	222.777325074129	101.637195846056

Table 5.1: Resume of the first measurements test

***Ping Interval* Campaigns** Unlike in the *Access Technology* test, described in section 5.1, the database tables used to carry out this first experiment are exactly the same showed in section 3.5. The realized tests for this experiment can be classify in 4 different categories:

- *3G Burst*: it consists of a series of 3G campaigns, tipically HSDPA, where the sampling time is constant and each campaign executes two bursts of pings (e.g. in the campaign *3g_burst_intval_0.1sec* every 100*ms* a ping is sent).
- *WiFi Burst*: same as *3G Burst* but with WiFi as access technology. All measurement campaigns are always within the GARR network.
- *WiFi PC Test*: series of test carried out with a laptop within the GARR network, as a countercheck of the *WiFi Burst* campaigns.
- *Variable Interval Test*: single test executed in a 3G environment with a decreasing interval time, starting from *INTERVAL* second down to the limitation of the hardware (the mechanism is showed in section 3.6)

The motivation around these experiments is to find out a critical treshold in the waiting time between consecutive probes that we can use as nominal sampling value for one of the specified categories of measurement campaign. The expected effect is that below a certain value the system experiences a phase transition, i.e. the behaviour changes.

What we discovered, in collaboration with SPOTTER authors, analysing this series of tests is that:

- We experienced large fluctuations, which we think are caused basically by the network setup between devices and Internet providers (actually, in the 3G campaigns the fluctuations are more evident). In many cases the samples of the two bursts of pings showed very different behaviour; we concluded that, with an high probability, the network was loaded very differently for *burst1* and *burst2* in certain campaigns;
- In the 3g campaigns for dense burst (waiting time $< 0.6s$) the samples are correlated since consecutive RTTs decrease linearly. This means that some queue is emptying, and the samples evolution reveals the

speed of this emptying. This queue must be close to the client (with an high probability the source of this queue is the BTS) because the increase step seems random and it is not affected by the sampling traffic. The conclusion that can be made from this results is that it seems that (for the 3g case in particular) going below a 0.6*s* waiting time may oversample the network; conversely, for higher gaps (more than 0.8*s*) the minimum RTT could be approximately twice as large. Therefore, from this experiment results, it seems that ~0.6*s* can be a good sampling choice.

In figure 5.4 and 5.5 we show a collection of plots that describes the evolution of the samples, sorted by sampling time and access technology. In particular in fig. 5.4 you can see that:

- the fluctuations are evident in the 0.3*s* case;
- when the sampling time is below 0.8*s* the number of samples with RTT > 400*ms* grows larger;
- The minimum sample value is ~20*ms* in all campaigns, so we can conclude that probably this is a good approximation of the real RTT towards the target;
- The queue emptying effect can be seen in all the plots but it seems not so strong as in the 3g case. As a matter of fact, when the queue is emptying the differences between consecutive RTTs in the queue is larger than in the 3g case. The 0.7*s* case or else the 0.5*s* case of WiFi and 3g tests make this point.

Instead, in the *3G Burst* series of test, whose plots are described in fig. 5.5, we can see that:

- The fluctuations are more or less evident in all plots;
- Below 0.7*s*, but more strongly below 0.6*s,* the queue emptying effect described before is visible;
- The minimum sample value is more variable than in the WiFi case, and this is always a consequence of the BTS network policies. Despite that

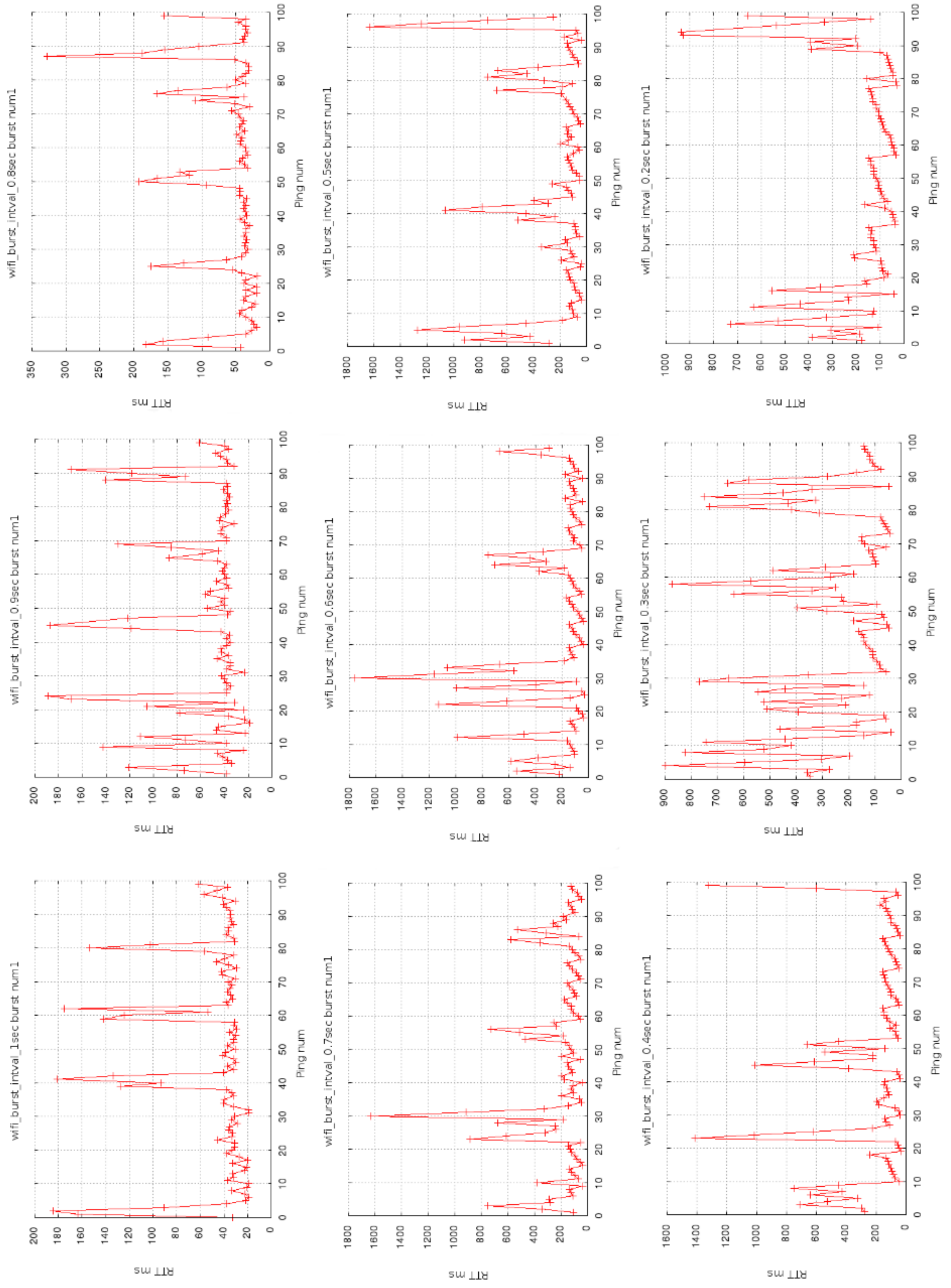


Figure 5.4: WiFi behaviour when lowering the sampling time

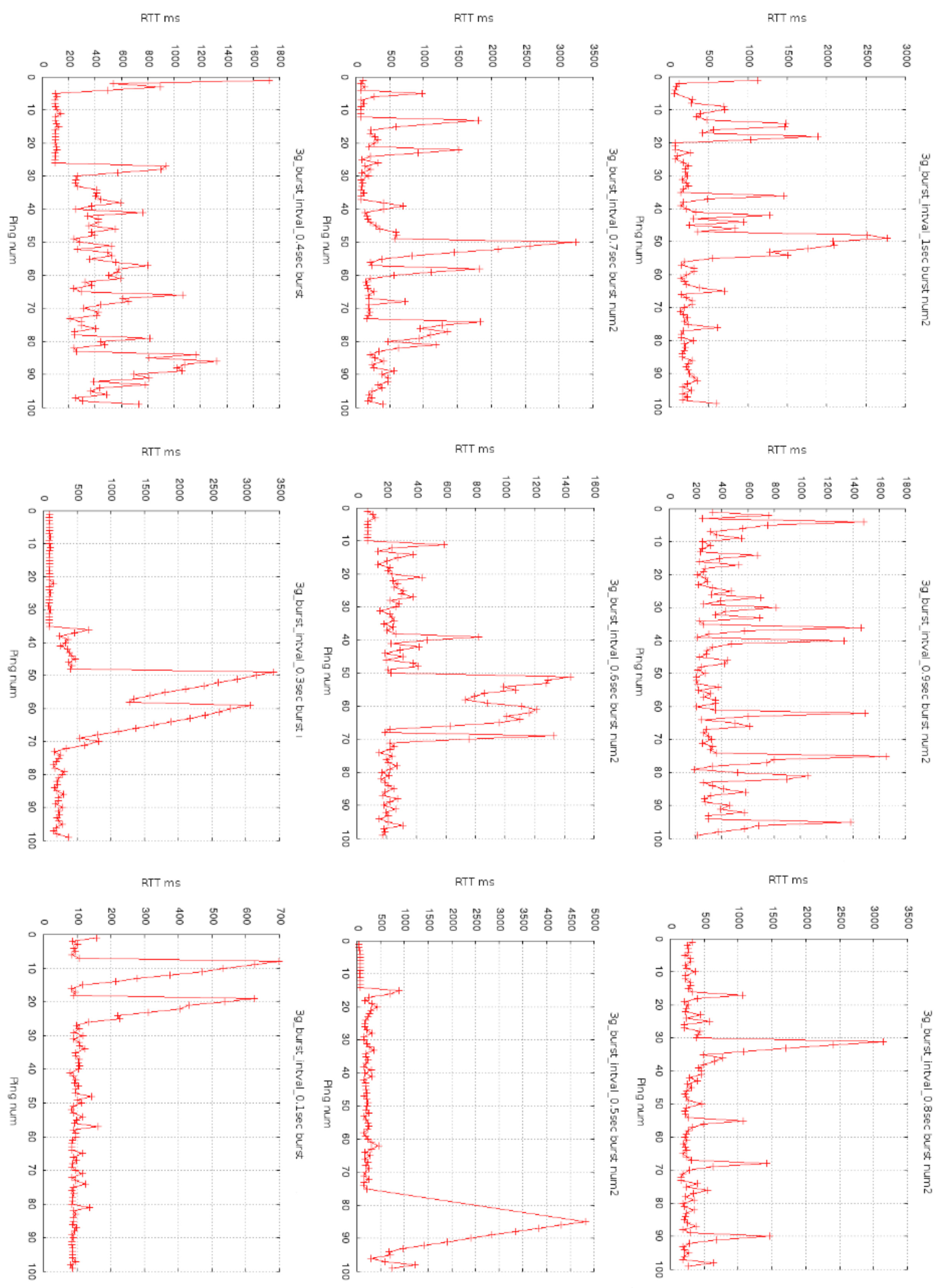


Figure 5.5: 3G behaviour when lowering the sampling time

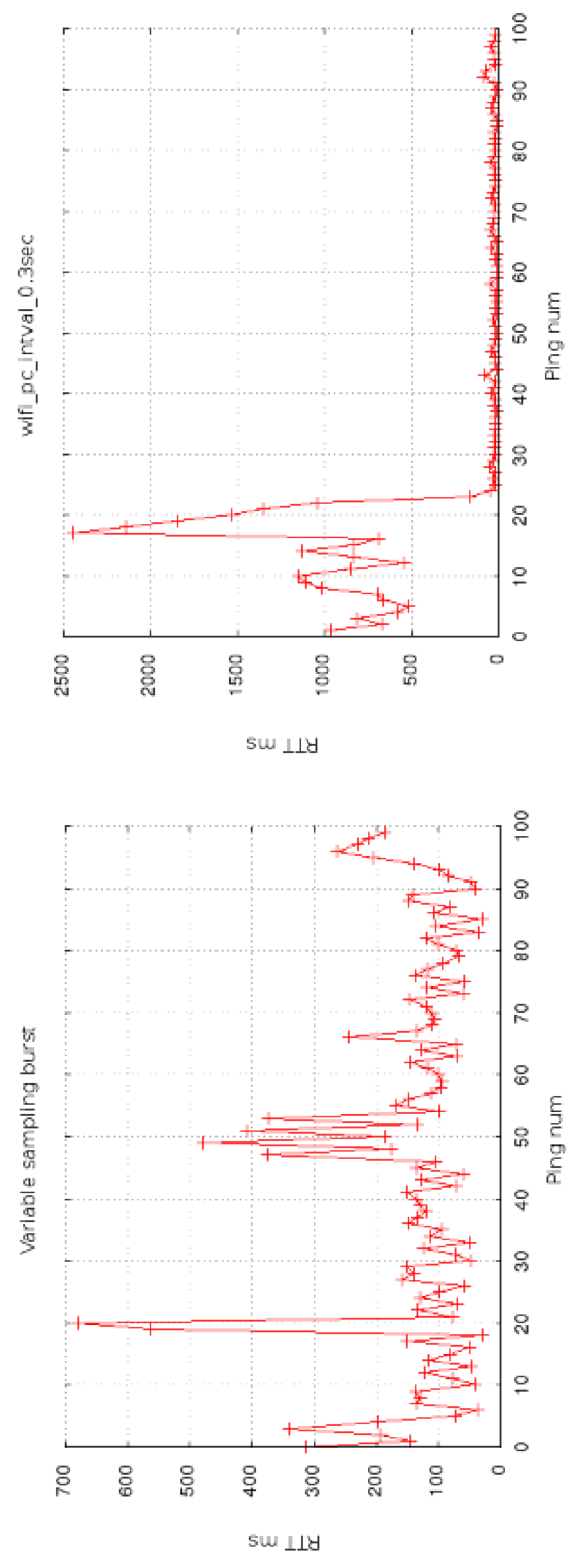


Figure 5.6: *Variable Interval Test* and *WiFi Pc Test* example plots

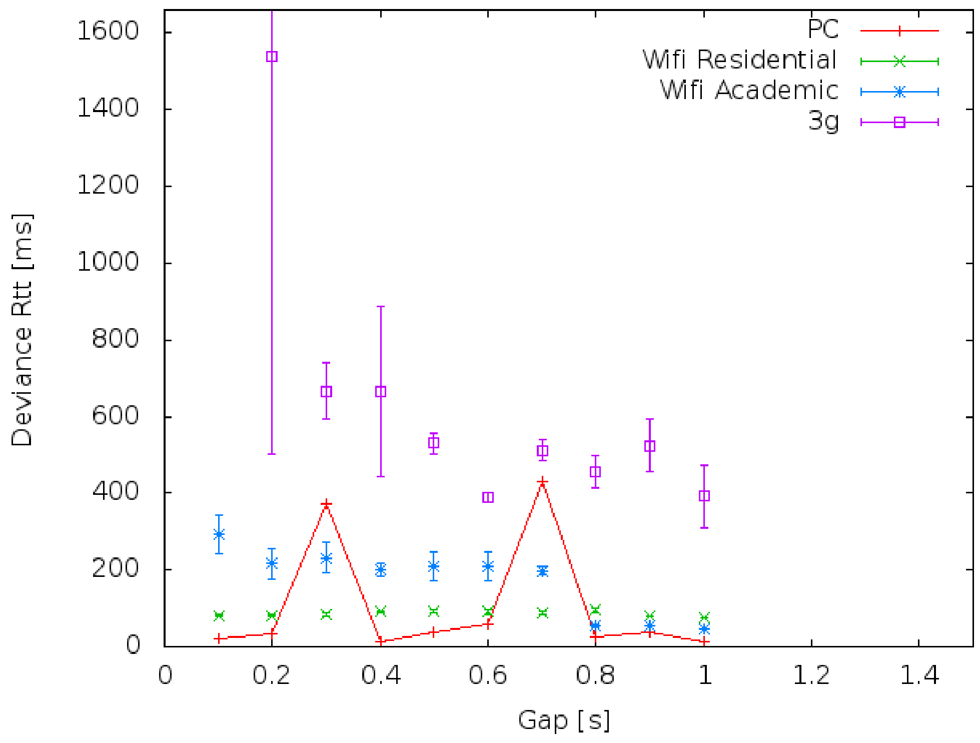


Figure 5.7: Variation of the RTTs for each technology

we can assert that probably the smallest RTT is ~80*ms*; this confirm the results of the *Access Technology* experiment, i.e. the delay at first hop has a large impact on the minimum RTT when using a mobile technology.

As we already said two more series of tests were executed to have a countercheck on the results obtained with the geolocation module of the PORTOLAN system. The first were executed using the WiFi of a laptop and the *Ping* tool of a Linux system, while the second make use of the PORTOLAN system with a variable sampling time between probes and a 3G access. As you can see in fig. 5.6 the hypotheses arisen from the 3G and WiFi bursts seems confirmed.

In the *"Variable sampling burst"* plot after the 55th sample the queue effect is more evident and the parabolic behaviour means that the queue empties out and immediately after is refilled with the following samples; in other words, below 0.5*s* the system starts to oversample the network.

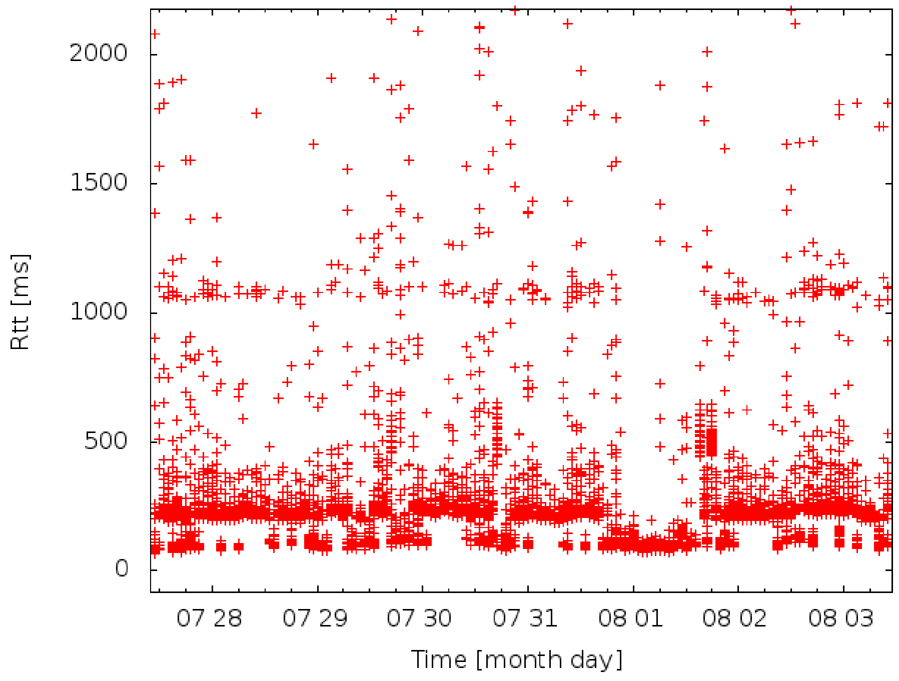


Figure 5.8: The 3 RTT levels can be found where the concentration of samples is higher

In conclusion in fig 5.7 we summarize the results arisen from this series of tests. This plot shows, for each technology, the dependency on different sample times of the variation of RTT measurements.

***Week* Campaigns** With the series of tests described above we expected that, below a certain value, the system experiences a phase transition; actually, this effect can be seen in some of the plots but we need more information to be sure that this phenomenon is consistent.

It is an interesting question whether the measured RTT variations depend on the time when the measurement campaign take place. In order to validate these hypotheses we designed another experiment:

- *weekTest* campaigns: we developed a mechanism that starts a new test campaign every hour for an entire week to see if there are daily/weekly variation on the network behaviour. We left the landmark in the same position for the entire week, this way we were sure that the BTS was always the same.

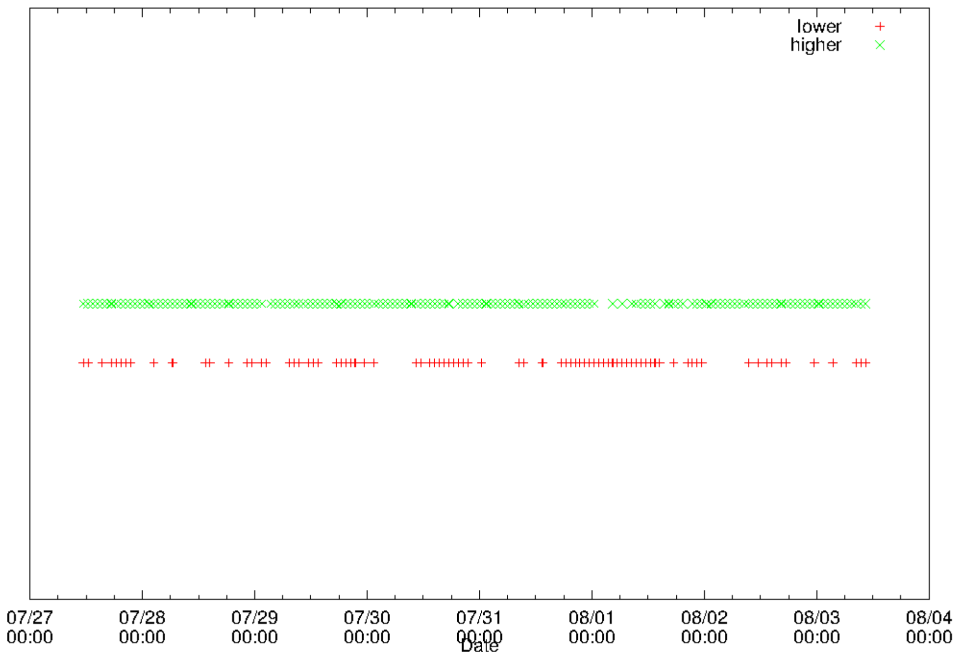


Figure 5.9: Time - RTT variation dependency

- The sample time has been set to 0.7*s*, while the number of sample has been set to 100 and only one burst of pings has been executed for each campaign. As we already stated, the target for the entire experiment was always the same (located in Padova).

The number of samples that were collected in this experiment is of 16.526 different pings, grouped in 168 batches (every batch is one hour long). In cooperation with Spotter authors we analysed these samples and we plotted all the samples together in order to check the time evolution of the RTT.

We discovered that the samples spread around three RTT levels, i.e. ~109*ms*, ~212*ms*, ~1080*ms* (the evolution can be seen in fig. 5.8). The third level can be neglected, because there are not so much samples as the other two levels and the values are too high for our geolocation process (we need the minimum values of RTT).

This discovery allows us to subdivide the samples in two RTT regions: higher and lower than ~150*ms*. At this point, we analysed the time dependency of the samples in this two levels and we found that, probably, there is not a direct relation between the RTT variation and the time. If the variations were caused by the traffic condition, it would be legit to assume that in the night measurements the number of samples in the lower region would be higher (less users connected, more bandwidth available), but, in this experiment, the number of samples in the higher level and the number of samples in the lower level does not change drastically depending on a particular time. This means that, probably, the variations depend more on the used technology than on the network traffic, i.e. the numbers of users connected to the BTS (this is showed in fig. 5.9).

Since the experiments left us a certain number of uncertainties, especially in the just discussed topic, we prepared new experiments that will not be discussed in this thesis. This is due to the fact that we are still in an initial phase of the studies on the IP geolocation field; nevertheless, we think that in the forthcoming future we would gain the knowledge necessary to improve our geolocation service.

6 Conclusion and future work

The goal of this thesis work is to design a new IP Geolocation approach based on RTT measurements performed by mobile monitors, i.e. landmarks in the geolocation terminology. Considering that in the recent years there is an exponential growth of new smartphones capable of connecting to the internet via wireless technologies such as WiFi, Gsm, Gprs, Hsdpa and more, we decided to use smartphones as mobile landmarks. The desired approach needs to satisfy the following requirements:

- Be based on RTT measurements;
- Be stable and efficient in a mobile environment;
- Be transparent to the smartphone owner, who can not be bothered by the measurements.

In order to accomplish these tasks we used the tools provided by the PORTOLAN PROJECT, a Internet measurement system that aims at obtaining the Internet graph at the Autonomous system abstraction level and building maps of the signal coverage through smartphone-based crowdsourcing. Moreover, we cooperated with the authors of a geolocation web service called SPOTTER in order to carry out the experimentation on our implemenation.

We designed and developed a new module over the original architecture of the PORTOLAN system in order to send the measurement campaigns to the smartphones and to retrieve these measurements. This module was subdivided in client and server sides; in particular, the client side is an application that runs in Android smartphones (the iOS version is still in developement). On the server side we added a new Database where to store the measurements collected by the landmarks; in addition, we created a web interface for specifying the campaigns to be performed by mobile devices and then retrieve the received data.

The discoveries made within the experimentation phase of this thesis work allowed us to build a system of hypotheses useful to develop a landmark selection algorithm and to tune up the measurements in order to lower the error rate of the geolocation process.

For the forthcoming future the implementation of the algorithm for the selection of landmarks is scheduled. Moreover, we want to mantain the cooperation with SPOTTER authors in order to calibrate their service to work with our approach and then provide a full working IP geolocation service.

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Luciano Lenzini for offering me the opportunity of working on a particularly interesting research field as the IP geolocation. Deepest gratitude is also due to my second supervisor Ing. Alessio Vecchio, who helped me a lot when it comes to deal with the Hungarian colleagues from the Eötvös Loránd University of Budapest, especially in the last days of the drafting of this thesis. I do not want to forget my tutors, Valerio and Adriano: without them this thesis would not have been ready in time.

Special thanks go to my *Red Lab* colleagues: Giovanni (I owe you one million coffee), Alessandro "*Zio*", Gloria (thanks for the cakes!), Angelo, Gordon, Dario and all the others not directly related with the *Red lab*. I would like to thanks also all my friends from Palermo and Pisa; the problem is that you are so many that I do not have enough space, but if I do not thanks Assia she will kill me, so thank you Assia for being my lovely housemate, and if I do not mention Stefano I think it would not be fair (we did it, finally!).

Thank you from the deep of my heart, Benedetta, for always being there for me and for supporting me when things are difficult (don't worry, i would not embarass you more!).

Last but not least, I would have not been here in Pisa to study if it was not for my parents, my family.

Thank you.

Bibliography

[1] J. Postel, “Internet Protocol.” RFC 791 (Standard), Sept. 1981. Updated by RFCs 1349, 2474.

[2] A. Faggiani, E. Gregori, L. Lenzini, S. Mainardi, and A. Vecchio, “On the feasibility of measuring the internet through smartphone-based crowdsourcing”, in *WiOpt*, pp. 318–323, IEEE, 2012.

[3] E. Gregori, L. Lenzini, V. Luconi, and A. Vecchio “Sensing the Internet through crowdsourcing”, in Proceedings of the Second IEEE PerCom Workshop on the Impact of Human Mobility in Pervasive Systems and Applications (PerMoby), May 2013, pp. 248–254

[4] S. Laki, P. Mátray, P. Hága, T. Sebők, I. Csabai, G. Vattay “Spotter: A Model Based Active Geolocation Service” In Proceedings of IEEE INFOCOM 2011, April 10-15, 2011, Shanghai, China (2011).

[5] S. Laki, P. Mátray, P. Hága, I. Csabai, G. Vattay “A Model Based Approach for Improving Router Geolocation” Computer Networks, Volume 54, Issue 9, 17 June 2010, Pages 1490-1501, ISSN 1389-1286 (2010).

[6] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolff, “A brief history of the internet,” SIGCOMM Comput. Commun. Rev., vol. 39, pp. 22–31, Oct. 2009.

[7] R. Cohen and D. Raz. The Internet Dark Matter: on the Missing Links in the AS Connectivity Map. In IEEE INFOCOM, 2006.

[8] Y. Rekhter, T. Li, and S. Hares, “A Border Gateway Protocol 4 (BGP-4).” RFC 4271 (Draft Standard), Jan. 2006. Updated by RFCs 6286, 6608.

[9] Lixin Gao, "On inferring autonomous system relationships in the Internet," Networking, IEEE/ACM Transactions on , vol.9, no.6, pp.733,745, Dec 2001

[10] L. Daigle, “WHOIS Protocol Specification” RFC 3912 (Standard), Sep. 2004.

[11] C. Davis, P. Vixie, T. Goodwin ,I. Dickinson “A Means for Expressing Location Information in the Domain Name System” RFC 1876 (Standard), Jan. 1996.

[12] M. Zhang, Y. Ruan, V. Pai, and J. Rexford, “How DNS misnaming distorts Internet topology mapping”, USENIX Conference, 2006.

[13] Ip2Location, http://www.iplocation.net/.

[14] GÉANT network <http://www.geant2.net/> and Looking Glass service <http://www.stats.geant2.net/lg/>.

[15] Venkata N. Padmanabhan, Lakshminarayanan Subramanian, “An investigation of geographic mapping techniques for internet hosts”, Proceedings of ACM SIGCOMM, p.173-185, August 2001, San Diego, CA, USA.

[16] Bamba Gueye, Artur Ziviani, Mark Crovella and Serge Fdida “Constraint-Based Geolocation of Internet Hosts”, in IEEE/ACM Transactions on Networking, 14(6):1219–1232.

[17] E. Katz-Bassett, J. John, A. Krishnamurthy, D. Wetherall, T. Anderson, Y. Chawathe: “Towards IP Geolocation using Delay and Topology Measurements”, ACM IMC 2006, p71-84, Rio de Janeriro, Brazil (2006).

[18] The HTM library, http://skyserver.org/htm.

[19] http://searchcio.techtarget.com/definition/crowdsourcing.

[20] E. Gregori, A. Improta, L. Lenzini, L. Rossi, and L. Sani, “On The Incompleteness of the AS-level Graph: a Novel Methodology for BGP Route Collector Placement.” to appear in Internet Measurement Conference (IMC) 2012, 2012.

[21] D. Veitch, B. Augustin, R. Teixeira, and T. Friedman, “Failure Control in Multipath Route Tracing,” in INFOCOM, pp. 1395–1403, IEEE, 2009.

[22] http://developer.android.com

[23] http://docs.oracle.com/javaee/6/api/javax/servlet/package-summary.html

[24] http://tomcat.apache.org/tomcat-7.0-doc/index.html

[25] http://www.postgresql.org/docs/9.1/static/index.html

[26] http://docs.oracle.com/javase/1.5.0/docs/api/

[27] M. Handley, E. Rescorla, “Internet Denial-of-Service Considerations” RFC 4732 (Standard), Nov. 2006.