UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Real time distributed BGP Hijack Detection

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Joan Ficapal Vila

In partial fulfilment

of the requirements for the degree in

TELEMATICS ENGINEERING

Advisor: Dario Rossi

Co- advisor: Josep Vidal

Barcelona, June 2016

Abstract

The main goal of this work is to study, design and develop a distributed system which is able to identify BGP hijacks in real time. The system will be deployed over group of computers available as a testbed.

BGP hijacking is considered one of the largest internet security threats with companies such as Google, YouTube, Amazon. This shows the need in our Internet for a system which would be able to detect and show the attacks geolocation, which highlights the importance of this work.

This thesis consists on contributing to improve the iGreedy software building new ways of result visualization and optimizing its functionality. The work includes the injection of some randomized BGP hijacks in order to assess and evaluate the performance of the system.

Resum

El principal objectiu d'aquest treball és l'estudi, disseny i desenvolupament d'un sistema distribuït que sigui capaç d'identificar els atacs al protocol BPG en temps real. El sistema s'implementarà sobre grup d'ordinadors amb funció de "testbed".

Els atacs a BGP són considerats una de les amenaces de seguretat més grans a internet per empreses com Google, YouTube o Amazon. Això ens mostra la necessitat real d'un sistema que sigui capaç de detectar i localitzar els atacs, que posa en evidència la importància d'aquest treball.

Aquesta tesis consisteix en ajudar a millorar el software iGreedy construint noves eines de visualització de resultats i optimitzant la seva funcionalitat. El treball inclou injeccions d'atacs aleatoritzats per tal d'avaluar el funcionament del sistema.

Resumen

El principal objetivo de este trabajo es el estudio diseño y desarrollo de un sistema distribuido que sea capaz de identificar los ataques al protocolo BGP en tiempo real. El sistema será implementado en un grupo de ordenadores con función de “testbed”.

Los ataques BGP son considerados una de las amenazas de seguridad más grandes a internet para empresas como Google, Youtube o Amazon. Esto nos muestra la necesidad real de un sistema que sea capaz de detectar y localizar los ataques, que pone en evidencia la importancia de este trabajo.

Esta tesis consiste en ayudar a mejorar el software iGreedy construyendo nuevas herramientas de visualización de resultados y optimizando su funcionalidad. El trabajo incluye la inyección de ataques aleatorizados para evaluar el funcionamiento del sistema.

Acknowledgements

I would like to thank Danilo Cicalese and Dario Rossi for their support during this project. They both have been my supervisors in my hosting university Telecom Paristech during this project and have provided me the required knowledge introducing me about the very promising iGreedy algorithm and guiding me during the whole stage.

Josep Vidal has been my tutor in Universitat Politècnica de Catalunya, he has showed me the procedure to follow offering help when I've been in need. I would like to thank him as well.

Revision history and approval record

Revision	Date	Purpose
0	06/04/2016	Project Plan creation
1	24/04/2016	Project Plan revision (Critical Review)
2	27/06/2016	Thesis submission

DOCUMENT DISTRIBUTION LIST

Name	E-mail
[Joan Ficapal Vila]	*ficapal18@gmail.com*
[Josep Vidal]	*josep.vidal@upc.edu*
[Dario Rossi]	*dario.rossi@telecom-paristech.fr*

WRITTEN BY: Joan Ficapal Vila		REVIEWED AND APPROVED BY:	
Date	6/07/2016	Date	21/06/2016
Name	Joan Ficapal	Name	Josep Vidal
Position	Project author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables:

1. Introduction

The introduction section will first explain a basis of the two main topics of this project, anycast and BGP Hijack detection, in order to make the reader better understand the problem statement, which will be the following section. The statement of purpose will be exposed at the end.

1.1. Anycast

The Internet Protocol addressing systems have currently five main routing schemes which have the utility of identifying hosts and providing a logical location. Each one is different and can be useful for a particular case:

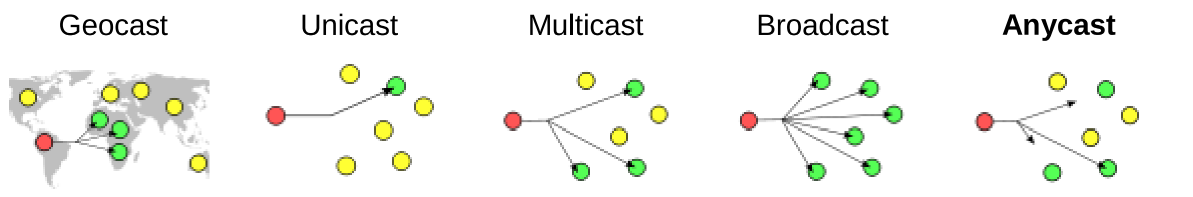


fig. 1 Routing schemes

[https://en.wikipedia.org/wiki/Routing]

While in unicast the message is delivered to a single specific node, or in broadcast is delivered to all nodes of the network, in anycast the message is delivered to anyone out of a group of nodes, typically to the topologically nearest one.

The same IP prefix is advertised from more than one location, and the network is responsible to decide which location to route the request using the routing protocol costs.

1.2. BGP Hijack

The Border Gateway Protocol is a crucial component of internet, it is responsible of determining the routing paths between independently operated networks.

A collection of IP prefixes operated by the same entity is called Autonomous System or AS. As long as the information doesn't leave that AS we will only need an intra- area protocol, such as RIP or OSPF. But if any user wants to interact with another AS, then an inter-area protocol will also be needed to make the routing decisions. Although there are several exterior gateway protocols, BGP is the standardized one.

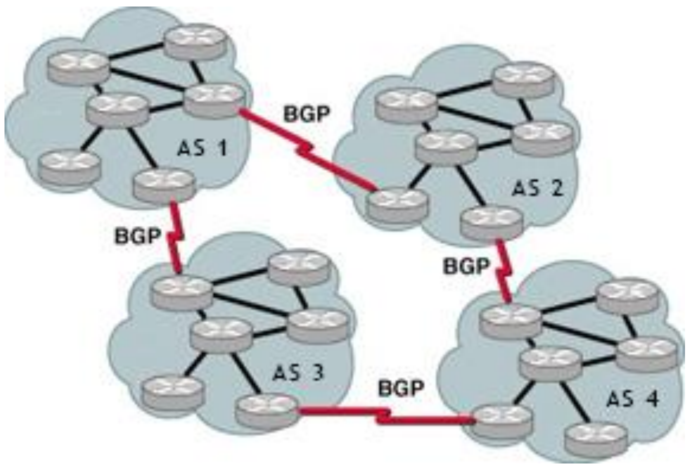


fig. 2 BGP Scheme

[http://ciscorouterswitch.over-blog.com/article-bgp-protocol-is-essential-in-your-ip-network-115059468.html]

Since is BGP who determines how data travels from its source to its destination looking at its available path routes, it is possible to attack the protocol announcing a node with better path or a more specific announcement. This might result as a rerouting of information, and the malicious announcement could cause traffic interception or modifications.

Finally, it is because IP-level anycast is realized through announcements of the same BGP prefix from multiple points, that anycast and hijacks are "syntactically" equivalent for a router speaking the BGP language.

1.3. Problem Statement

Among the feasible techniques to detect anycast instances I will focus on a latency based algorithm which is the basis of the thesis.

Let's imagine we have M Vantage Points distributed around the earth globe, and for a given IP address target, we draw circles centred on the VPs with its radius [Km] based on their own Round Trip Time (RTT) towards the target, assuming the signal to travel at speed of light. If there are any pair of disks that do not overlap, we proof that this target is an anycast address, otherwise it would imply a speed of light violation.

As stated, anycast and BGP hijacks share the success of more than two identical IP prefixes which makes possible to use the described method to solve both problems. However, we need a bigger amount of time for enumerating than detecting. That difference may cross the required limits for detecting BGP hijacks, which usually happen during a short period.

Knowing that each VP can only detect an anycast replica, for a given number of N anycast instances in a dataset, if our purpose is to enumerate then $2 \leq N \leq M$ vantage points are needed. In the other hand, only $2 \leq M$ vantage points would be needed for detecting.

The more vantage points used, the bigger is the time elapsed and its cost. This dilemma is shown in the following plots (fig.3), where the probability is defined as *Detections / N*:

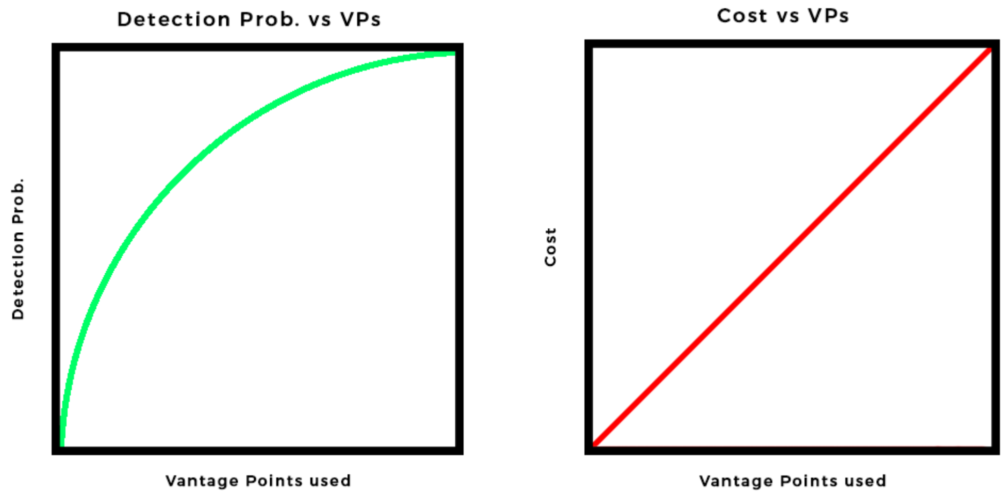


fig. 3 Comparison Detection Prob. vs Cost

For a high detection ratio we also need to increase the amount of vantage points, which will increase the total cost and procedure time. For the reason explained, we assume that there’s not a ‘win-win’ for this problem.

1.4. Statement of purpose

This project is carried out at Telecom ParisTech in the LINCS department and is a continuation of another project. The Erasmus supervisor of the project is Dario Rossi and the UPC Tutor is Josep Vidal.

The project main goals are:

- Become familiar with the previous software version: iGreedy. Understand exactly how it works and learn the required coding languages and APIs (Python, JavaScript, JSON, HTML, Google Maps and Ripe Atlas APIs)
- Create a software that queries Ripe Atlas API last version, capable to stream the information and retrieve as many details about the experiment as possible. Plug the software into iGreedy, which is currently unable to display streaming information. Make the application and its web interface capable to show results dynamically and improve the web visualization system.
- Study, test, and analyze with clustering methods new ways to initialize the geological situation and number of Vantage Points.

Work Packages:

Project: Introduction	WP ref: (WP1)	
Major constituent: research	Sheet 1 of 2	
Short description: Become familiar with the previous software version and learn the required APIs and coding languages.	Planned start date:08/02/2016 Planned end date:22/02/2016	
	Start event: End event:	
Internal task T1: Learn Python Internal task T2: Become familiar with the Ripe Atlas enviroment Internal task T3: Understand the iGreedy functionality. Learn how to use it and how it's built	Deliverables:	Dates:

Project: Build iStreaming software	WP ref: (WP2)	
Major constituent: SW	Sheet 1 of 2	
Short description: Create the Streaming software (iStreaming).	Planned start date: 22/02/2016 Planned end date: 07/03/2016	
	Start event: End event:	
Internal task T1: Ripe Atlas Streaming Query retrieving Internal task T2: Understand JSON environment, and retrieve the results in a proper format Internal task T3: Make it compatible with multiple IPs as input	Deliverables: Project proposal and work plan	Dates:

Project: iGreedy- streaming version	WP ref: (WP3)	
Major constituent: SW	Sheet 2 of 2	
Short description: Plug iStreaming into iGreedy.	Planned start date:08/03/2016 Planned end date:22/04/2016	
	Start event: End event:	
Internal task T1:	Deliverables:	Dates:

Build the application Internal task T2: Learn javascript notions and build the website		

Project: Optimize results using clustering methods	WP ref: (WP4)	
Major constituent: research, simulation, SW	Sheet 2 of 2	
Short description: Study, test, and analyze with clustering methods the Vantage Point optimal initialization.	Planned start date: 17/04/2016 Planned end date: 27/06/2016 Start event: End event:	
Internal task T1: Research possible classification algorithms and implement them. Internal task T2: Analyze results and find optimal solutions Internal task T3: Once the project has been finished, the conclusions and procedure will be explained in the final report.	Deliverables: Critical Review Final Report	Dates:

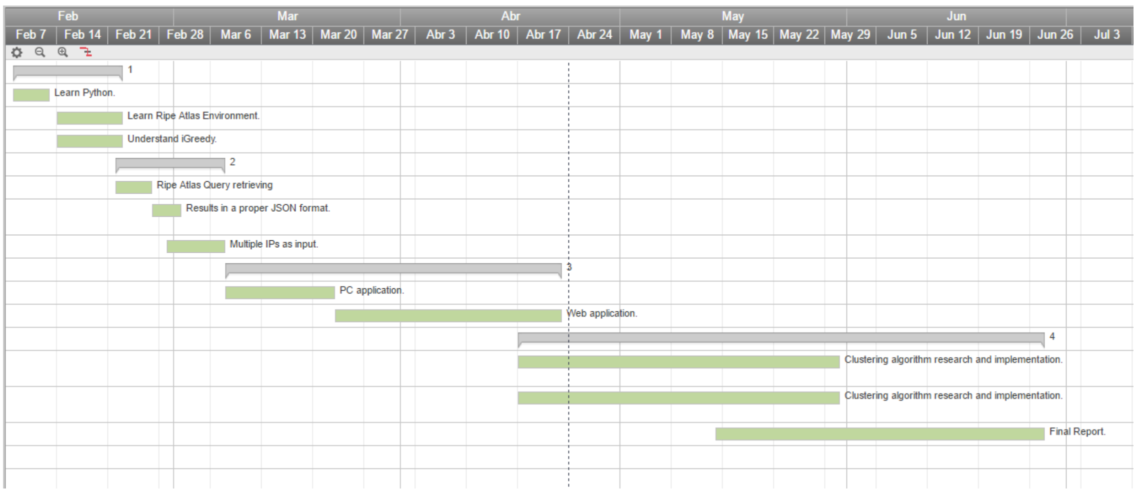
Milestones:

WP#	Task#	Short title	Milestone / deliverable	Date (week)
1	1 2 3	Learn Python. Learn Ripe Atlas Environment. Understand iGreedy.	Be able to continue	1 1-2 1-2
2	1 2 3	Ripe Atlas Query retrieving Results in a proper JSON format. Multiple IPs as input.	iStreaming	2-3 3-4
3	1 2	PC application. Web application.	iGreedy- streaming version	5-7 8-12
4	1 2 3	Clustering algorithm research and implementation. Conclusions. Final Report.	Research and Implementation Conclusions / Final Report	12-16 16-19 16-20

Time Plan (Gantt diagram)**:**

Task Name	Start Date	End Date	Duration
1	**08/02/16**	**22/02/16**	**11d**
Learn Python.	08/02/16	12/02/16	5d
Learn Ripe Atlas Environment.	14/02/16	22/02/16	7d
Understand iGreedy.	14/02/16	22/02/16	7d
2	**22/02/16**	**07/03/16**	**11d**
Ripe Atlas Query retrieving	22/02/16	26/02/16	5d
Results in a proper JSON format.	27/02/16	01/03/16	3d
Multiple IPs as input.	29/02/16	07/03/16	6d
3	**08/03/16**	**15/04/16**	**29d**
PC application.	08/03/16	22/03/16	11d
Web application.	23/03/16	22/04/16	18d
4	**17/04/16**	**27/06/16**	**52d**
Clustering algorithm research and implementation.	17/04/16	30/05/16	32d
Clustering algorithm research and implementation.	17/04/16	30/05/16	32d
Final Report.	14/05/16	27/06/16	32d

table 1 Gantt Diagram

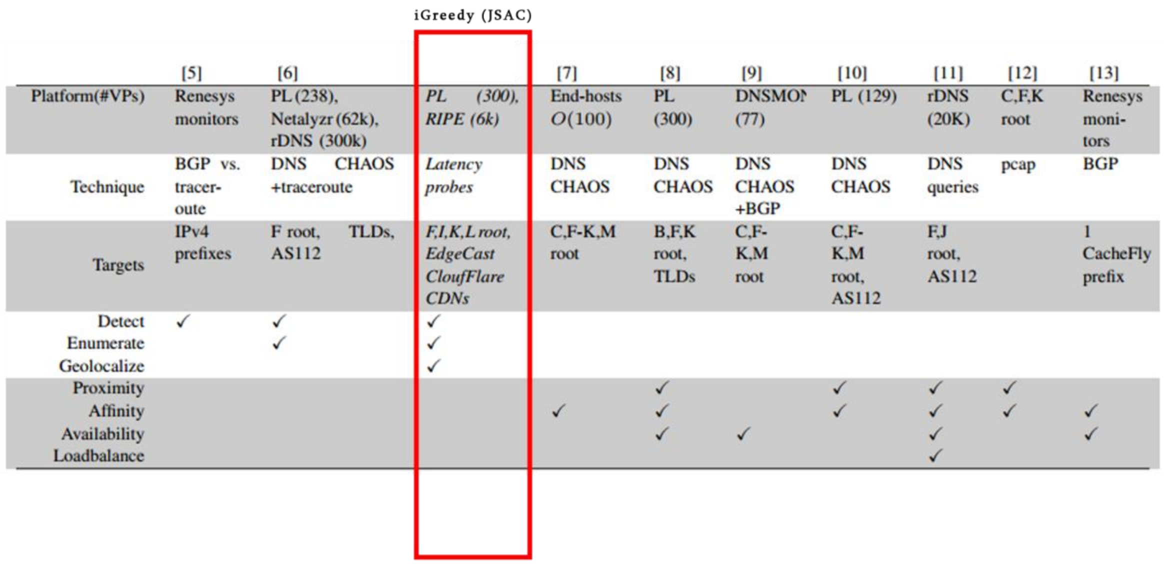


2. State of the art of the technology used or applied in this thesis:

Anycast server enumeration and geolocalization is a topic which has widely concerned the research community to geographically map the Internet infrastructure and identify the various components of the physical Internet.

While latency-based unicast geolocation is well studied, there's a lack of understanding in anycast because techniques such as triangulation at the intersection of several measurement used in unicast do not apply.

Research on anycast has focused either on architectural modifications or the characterization of existing deployments. A large fraction of these studies quantify the performance in current IP anycast deployments in terms of metrics such as proximity, affinity, availability or load balancing. These studies are compactly summarized in the following table (table 2):



	[5]	[6]	iGreedy (JSAC)	[7]	[8]	[9]	[10]	[11]	[12]	[13]
Platform(#VPs)	Renesys monitors	PL (238), Netalyzr (62k), rDNS (300k)	*PL (300), RIPE (6k)*	End-hosts $O(100)$	PL (300)	DNSMON (77)	PL (129)	rDNS (20K)	C,F,K root	Renesys moni-tors
Technique	BGP vs. tracer-oute	DNS CHAOS +traceroute	*Latency probes*	DNS CHAOS	DNS CHAOS	DNS CHAOS +BGP	DNS CHAOS	DNS queries	pcap	BGP
Targets	IPv4 prefixes	F root, TLDs, AS112	*F,I,K,L root, EdgeCast CloufFlare CDNs*	C,F-K,M root	B,F,K root, TLDs	C,F-K,M root	C,F-K,M root, AS112	F,J root, AS112		1 CacheFly prefix
Detect	✓	✓	✓							
Enumerate		✓	✓							
Geolocalize			✓							
Proximity					✓		✓	✓	✓	
Affinity				✓	✓		✓	✓	✓	✓
Availability					✓	✓		✓		✓
Loadbalance								✓		

table 2 State of the art

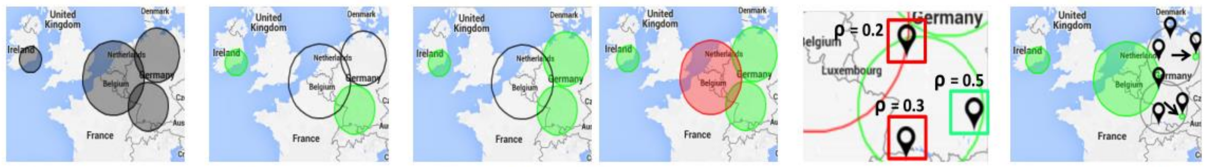
2.1. Latency-Based Anycast Geolocation: Algorithms, Software and Datasets

Fewer algorithms instead exist that allow to detect, enumerate or geolocate anycast replicas. As explained in [1] JSAC, with the exception of iGreedy that this work extends, no other technique exists that is capable in a lightweight and protocol-independent way to find the geolocation of anycast replicas.

This method uses the latency based algorithm explained in the Problem Statement of Introduction section. It achieves the enumeration and its city level geolocalization from a set of known vantage points.

Results of a validation campaign show this algorithm to be robust to measurement noise, and very lightweight as it requires only a handful of latency measurements.

The procedure is shown in fig. 6:



(a) **Measurement**: Map RTT samples to disks centered around VPs

(b) **Detection**: Non-overlapping disks imply speed-of-light violation

(c) **Enumeration**: Solving a Maximum Independent Set (MIS) problem yields non-overlapping disk, each containing a different replica (two steps shown)

(d) **Geolocalization**: Maximum likelihood classification problem (city-level)

(e) **Iteration**: Collapse disks around geolocalized replicas until convergence

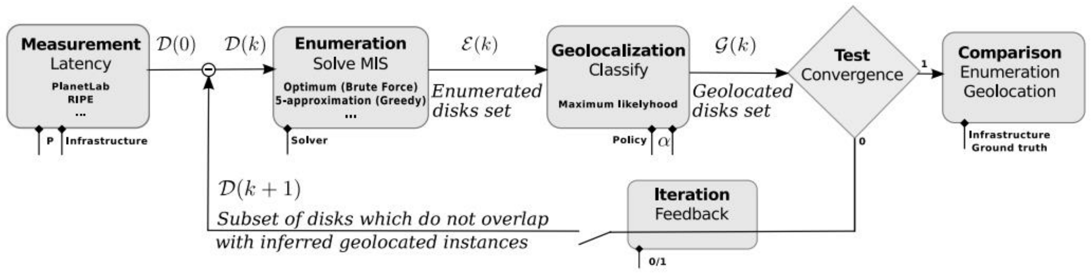


fig. 5 iGreedy num. 1

My background for this work has been mainly provided by this publication [1], it includes:

- The iGreedy technique for lightweight service-agnostic anycast discovery, capable of accurately enumerate and geolocate (>75% true positive geolocation) replicas with a handful of latency measurements.
- Its thorough validation, using multiple targets pertaining to different services (DNS and CDN) from two measurement infrastructures (RIPE Atlas and PlanetLab).
- An open-source implementation of the technique, able to operate on offline datasets, as well as to generate new datasets (from RIPE Atlas);
- A simple environment to visualize on a map the results of detection and classification algorithms.
- A ground truth database.
- A dataset comprising exhaustive latency measurements from two measurement infrastructures (RIPE Atlas, PlanetLab) towards anycast addresses in the ground truth.

2.2. BGP Hijack Detection

BGP lacks any form of path or origin validation, leaving it extremely vulnerable to attacks and misconfiguration. One example is the fact that networks can advertise illegitimate paths that redirect traffic destined for another network to themselves, known as BGP hijacking.

Plenty of work has been done in the past, but we are still far from having a general near real time detection system of the BGP hijacks. The detection methods have been addressing the problem from the control plane [15, 16] or data plane [17,18] measurements. Limitations of data-plane and control-plane measurements separately led to the development of hybrid approaches [19, 20, 21].

However, timing becomes critical because some attacks last few tens of seconds. This may make ineffective the previous publications, introducing the need of a complementary system. For this reason, we want to face the problem from another perspective, correlating anomalies on the data-plane with the control-plane measurements. This approach will allow to have information on data and control planes, before and after the attacks. To achieve this objective, JSAC [1] which this work extends, started building over a system able to perform an IPv4 census in few hours, an algorithm capable to do lightweight and fast scans. It is required to scan quickly and continuously the IPv4 space from a set of vantage points on the data plane.

It achieves the challenge of implementing an effective detection technique able to raise alert taking care of timing, keeping the number of false positive limited. On the other side, the system is able to check suspicious events raised on the control-plane running new measurements and checking in the previous measurements.

3. Methodology / project development:

As the first part of the project (packages 1-3) and the second (package 4), are dealing with different topics and have different goals although both aim to improve iGreedy, the explanation in the following sections will be split in two, the visualization and the vantage points selection improvement.

3.1. Visualization

3.1.1. iStreaming

As previously explained, iGreedy software is capable of enumerating and detecting anycast instances. With an input consisting on an IP target (e. g: www.google.com) and a list of VP that the user is willing to use, the global network of probes produces ping measurements. Once analysed, the results are retrieved in a CSV or JSON format and optionally displayed in the webpage using Google Maps API.

For a better understanding, the previous version of iGreedy structure is shown in fig. 6:

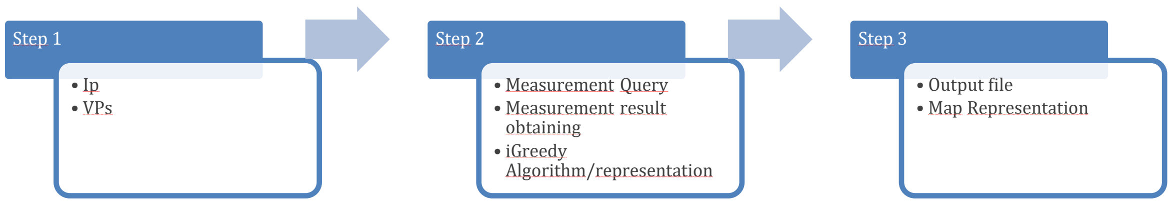


fig. 6 iGreedy num. 2

Testing our iGreedy we noticed that results change dramatically at the beginning, but after few seconds the algorithm makes only minor changes. Contrarily to our interests, in fig. 6 we appreciate that it is composed of a closed process of three steps, one after the other, and the analysis results are shown once all the measurements are retrieved, fixing this way the final retrieval of results. Thereby, we have restructured iGreedy adding the streaming mode and making it able to show both, complete and early results.

To realize it, I needed a previous knowledge of the environment. I spent approximately two weeks learning Python and the functionality and structure of iGreedy. Our software uses RIPE Atlas and Planetlab as global networks of probes, they allow several operations like pinging from multiple points around the world in exchange of credits, so I also learnt how to use the RIPE Atlas Magellan API, which is the library's last version and also allows to fetch streaming measurements. A seminar in Cisco Systems Building, and webinars were included in the learning process as well.

This Ripe Atlas library concedes using all the toolkits on bash command line or Python. It’s been made for the purpose of facilitating the visualization of public data of protocols and services such as Ping, DNS, SSL, Traceroute, HTTP or NTP. For our project we don’t only need ping reports visualization but also measurement creation, which is a little more complicated and has some instabilities because is still in beta version.

Ripe Atlas library is a Linux- based tool that can be installed easily with the two following commands if we already have pip installed:

```
sudo apt-get install python-dev libffi-dev libssl-dev
pip install git+https://github.com/RIPE-NCC/ripe-atlas-tools.git
```

Its requirements are:

- ripe.atlas.cousteau
- ripe.atlas.sagan
- tzlocal
- pyyaml

It’s important to clarify that while using its visualization tools is free, creating any kind of measurements requires an account and has a credit cost. Credits are obtained by paying, helping Ripe Atlas to improve, or hosting your own probes.

We first tried the Python version but we had several problems creating new queries for many probes in a single demand. We supposed it was a problem of that version in particular, because we were able to do it with the past one in non-streaming mode. As said, we knew it was a beta version which could have some instabilities. Our purpose for this early phase was not researching but making things work. For this reason, we moved on the bash commands version where we succeeded.
We query Ripe Atlas server for measurements using a function that has as input the IP target, all the probe IDs, and the user account identification. The query requires curl to be installed and has this structure:

```
curl --dump-header - -H "Content-Type: application/json" -H "Accept: application/json" -X POST -d '{
 "definitions": [
  {
   "af": 4,
   "packets": 3,
   "size": 48,
   "description": "Ping measurement",
   "interval": 240,
   "resolve_on_probe": false,
   "skip_dns_check": false,
   "type": "ping"
  }
 ],
 "probes": [
  {
   "value": " PROBE_ID1, PROBE_ID2, PROBE_ID3, PROBE_ID4 ...",
   "type": "probes",
   "requested": NUMBER_OF_PROBES
  }
 ],
 "is_oneoff": false,
 "bill_to": "ACCOUNT_EMAIL"
}' https://atlas.ripe.net/api/v2/measurements/?key=YOUR_KEY_HERE
```

The last code is passed through *“subprocess”* Python module which allows to spawn new processes as using an actual Linux terminal.

Data retrieval was easier because we directly succeeded implementing it with Python:

```
def run(self):
    atlas_stream = AtlasStream()
    atlas_stream.connect()
    # Measurement results
    channel = "result"
    # Bind function we want to run with every result message received
    atlas_stream.bind_channel(channel, self.on_result_response)
    # Measurement id
    stream_parameters = {"msm":self.id}
    atlas_stream.start_stream(stream_type="result",**stream_parameters)

    # Probe's connection status results
    channel = "probe"
    atlas_stream.bind_channel(channel, self.on_result_response)
    stream_parameters = {} # Here its possible to add the start time
    atlas_stream.start_stream(stream_type="probestatus", **stream_parameters)

    # Timeout all subscriptions after 1200 secs. Leave seconds empty for no
timeout.
    atlas_stream.timeout(seconds=1200)
    # Shut down everything
    atlas_stream.disconnect()

def on_result_response(self,*args):
    """
    Function that will be called every time we receive a new result.
    Args is a tuple, so you should use args[0] to access the real message.
    """
    self.infile=Handler.retrieveResult(self.infoprobes,args[0])
```

Once the run function is called, whenever any new packet arrives in that connection the on_result_response function is called. This function is responsible of retrieving results in a file in JSON and CSV format.
This two pieces of code and two other classes used to control and retrieve are the main basis of our streaming system.

Instead of just modifying our software we decided to build an independent module, thinking about using it in future applications in an easy way. It is called iStreaming (fig. 7):

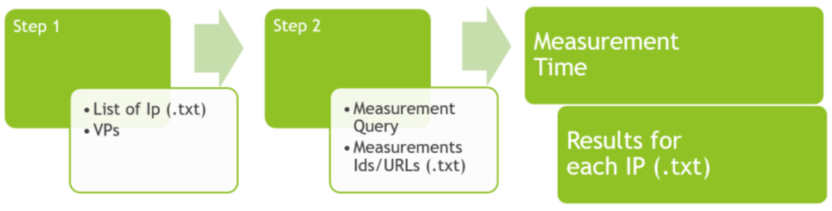


fig. 7 iStreaming

This program is written in Python and improves the two first iGreedy steps. Once iStreaming has read the inputs, it queries RIPE Atlas server asking for streaming measurements and retrieving its identification in a different file as extra output. Appending the measurements ID in RIPE Atlas URL brings us to the concrete measurement information with extra detail, that's useful in the sense of checking them again whenever we want without needing to run iGreedy and consuming credits. It also allows multiple IPs as input and the output format is exactly the same as iGreedy, making it easier to plug.

Finally, the results for each IP are retrieved as soon as obtained and we can move on next steps without waiting for all measurements to be completed.

3.1.2. iGreedy Streaming Version

Now that our module was built, it was time to merge it with iGreedy. Although the main algorithm didn't need big changes, the webpage interface and the output structure did.

The output files are refreshed each second if new measurements are received from our measurement infrastructures, but we still keep the past ones making it possible to represent again all the steps during the performance with a slider that will be shown in results section.

The webpage is not hosted by an online server, it's instead an application inside iGreedy written in JavaScript and HTML which is read by these interpreters in the same computer that is executing iGreedy. Hence, no internet is needed to just visualize if we forget about creating new measurements. The main pros for this format include avoiding the payment of a domain or the hosting and the offline availability. We thought about creating server but we continued this way because it would mean a complexity increase and all the features were working properly.

To modify the webpage and adding new complements I needed to apply basic notions of HTML and JavaScript, but this time I managed myself to success without any courses but some internet sources and solved examples.

In fig. 8 there's the resulting structure:

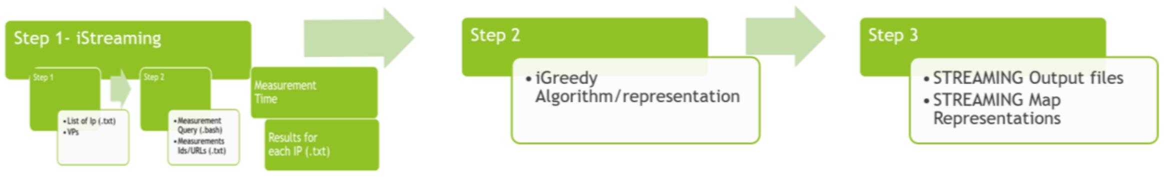

Step 1- iStreaming
Step 1
• List of Ip (.txt)
• VPs
Step 2
• Measurement Query (.bash)
• Measurements Ids/URLs (.txt)
Measurement Time
Results for each IP (.txt)
Step 2
• iGreedy Algorithm/representation
Step 3
• STREAMING Output files
• STREAMING Map Representations

fig. 8 iGreedy streaming version

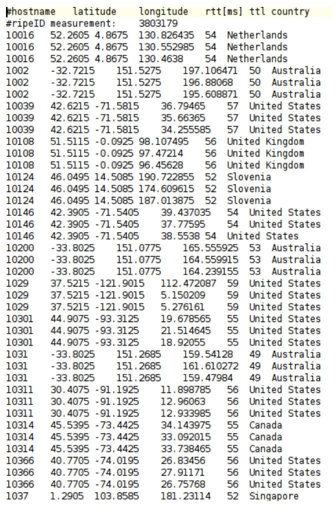
3.2. <u>VP Selection</u>

The aim of this section is finding out if there is any possible way to improve the vantage points initialization. Imagine we have a total amount of M vantage points, if we were able to guess the winner probes, say N, we could use the $M - N$ probes for other IP targets in a parallelized iGreedy.

3.2.1. Database

In order to test each new technique, the first step was modifying iGreedy to make it a simulation tool. With an input consisting on a measurement file, it had to be able to give back as fast as possible the same output format without consuming credits querying RIPE Atlas or Planetlab server. In other words, a similar program optimized for doing large simulations, and skipping the step of querying measurements to obtain reports, which we want to replace with our database.
Given an initial Planetlab measurement dataset of 1400 files from March 2016, we ran each one with the simulation version, then we sorted the outputs by the number of instances detected. Each measurement file has its own target, hence if the amount of detections are equal or higher than 2, it means that there is an anycast instance, otherwise it means that there's not.

Fig. 9 shows an example of a standard RIPE Atlas measurement file, each hostname belongs to a probe and each line contains the latitude, longitude, RTT, TTL and country. Every probe has 3 lines containing the average, maximum, and minimum RTT of the received pings.



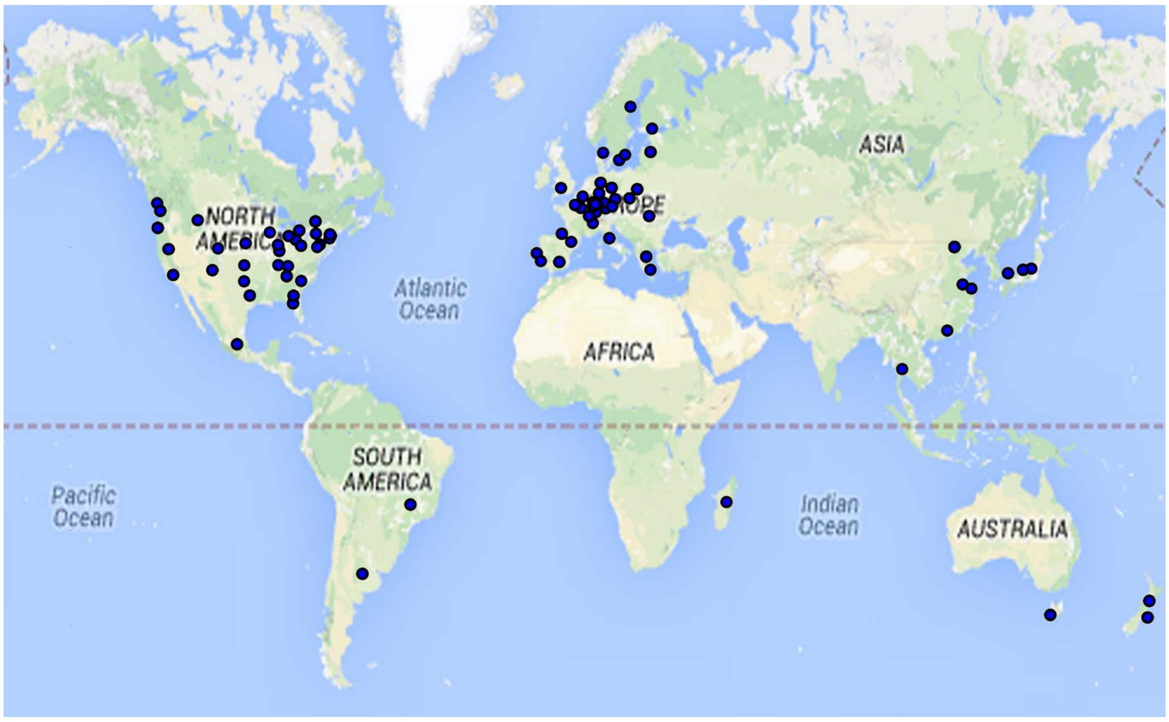
```
#hostname   latitude    longitude   rtt[ms] ttl country
#ripeID measurement:    3803179
10016   52.2605 4.8675  130.826435  54  Netherlands
10016   52.2605 4.8675  130.552985  54  Netherlands
10016   52.2605 4.8675  130.4638    54  Netherlands
1002    -32.7215    151.5275    197.106471  50  Australia
1002    -32.7215    151.5275    196.88068   50  Australia
1002    -32.7215    151.5275    195.608871  50  Australia
10039   42.6215 -71.5815    36.79465    57  United States
10039   42.6215 -71.5815    35.66365    57  United States
10039   42.6215 -71.5815    34.255585   57  United States
10108   51.5115 -0.0925 98.107495   56  United Kingdom
10108   51.5115 -0.0925 97.47214    56  United Kingdom
10108   51.5115 -0.0925 96.45628    56  United Kingdom
10124   46.0495 14.5085 190.722855  52  Slovenia
10124   46.0495 14.5085 174.609615  52  Slovenia
10124   46.0495 14.5085 187.013875  52  Slovenia
10146   42.3905 -71.5405    39.437035   54  United States
10146   42.3905 -71.5405    37.77595    54  United States
10146   42.3905 -71.5405    38.5538 54  United States
10200   -33.8025    151.0775    165.555925  53  Australia
10200   -33.8025    151.0775    164.559915  53  Australia
10200   -33.8025    151.0775    164.239155  53  Australia
1029    37.5215 -121.9015   112.472087  59  United States
1029    37.5215 -121.9015   5.150209    59  United States
1029    37.5215 -121.9015   5.276161    59  United States
10301   44.9075 -93.3125    19.678565   55  United States
10301   44.9075 -93.3125    21.514645   55  United States
10301   44.9075 -93.3125    18.92055    55  United States
1031    -33.8025    151.2685    159.54128   49  Australia
1031    -33.8025    151.2685    161.610272  49  Australia
1031    -33.8025    151.2685    159.47984   49  Australia
10311   30.4075 -91.1925    11.898785   56  United States
10311   30.4075 -91.1925    12.96063    56  United States
10311   30.4075 -91.1925    12.933985   56  United States
10314   45.5395 -73.4425    34.143975   55  Canada
10314   45.5395 -73.4425    33.092015   55  Canada
10314   45.5395 -73.4425    33.738465   55  Canada
10366   40.7705 -74.0195    26.83456    56  United States
10366   40.7705 -74.0195    27.91171    56  United States
10366   40.7705 -74.0195    26.75768    56  United States
1037    1.2905  103.8585    181.23114   52  Singapore
```

fig. 9 Measurement

As previously explained, we only need a minimum of 2 instances to determine a detection, instead of *2 ≤ N ≤ M* for enumerating. Our purpose was improving iGreedys capacity to detect Hijacks, so we cleaned our dataset leaving only the files with exactly 2 replicas as output. In this new dataset which is the one we have used during the study, the maximum detections for each file is always 2.

The new set is composed of 298 files with 140 different probes distributed approximately in the following geographical locations (fig. 10):

fig. 10 Map Probes



Which can also be displayed as fig. 11 representing the number of probes per country.

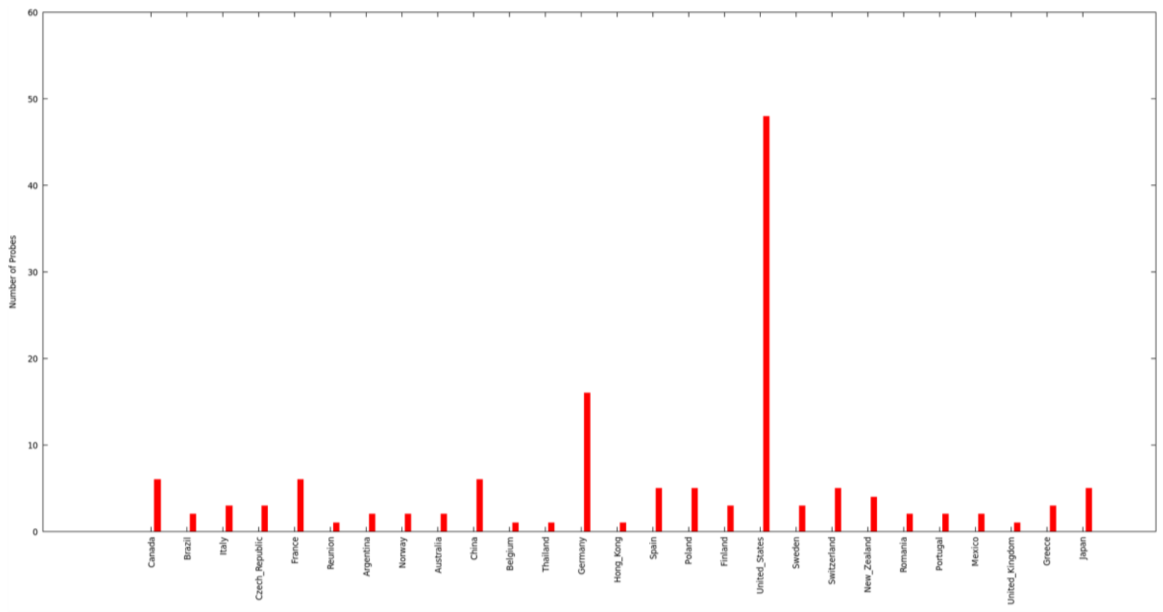
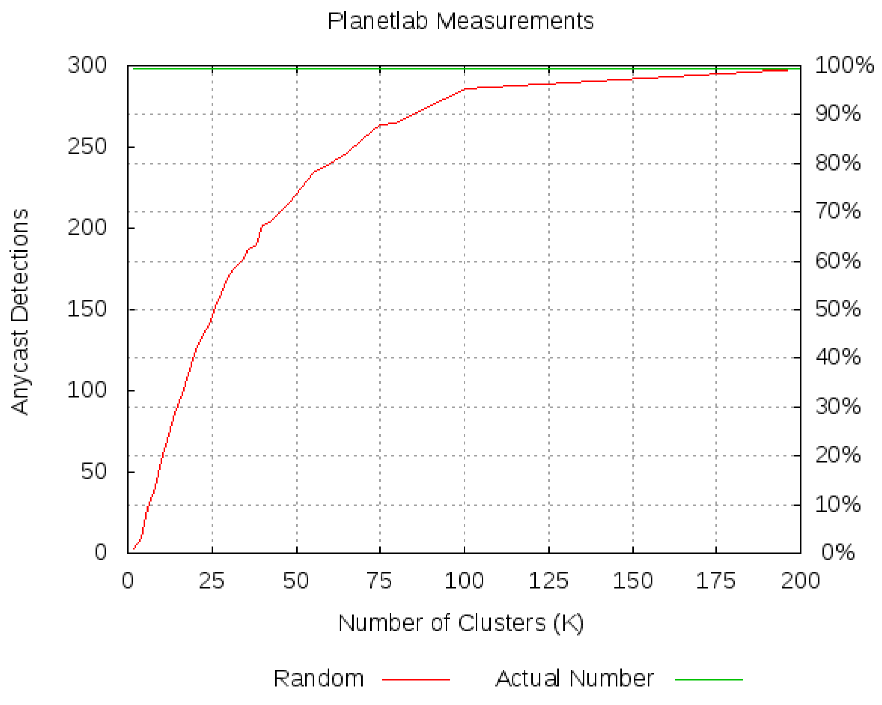


fig. 11 Probes per country

These graphs have a crucial importance for our work because they allow us to conclude that the probes are not distributed uniformly around the world. They are concentrated particularly in United States of America and Europe. Although this could be produced by our file filtration, information from Planetlab web confirms it is quite representative and the actual geographical situation of all probes usually follows these densities.

3.2.2. Procedure

Our research started picking K random probes from each file and simulating them over iGreedy. Doing the sum of the anycast detections we obtain the graph (fig. 12) for Planetlab, which we will use as point of comparison for any further algorithm.
To compare each result with the past ones trying to minimize the luck factor, every experiment shown as a graph in this work is an average of 10 simulations. As a curious fact, each graph has taken approximately between 1 and 5 hours depending on the complexity.



Our experience shows that the closer two anycast replicas are, the more difficult is to analyze them because they get confused. As a consequence, if we had to choose the most difficult scenario, it would be one with only two very close replicas for the same IP.

For this reason, we decided to create clusters using k-means algorithm to select far distanced vantage points. To understand K-means, suppose we have a data set consisting of N observations of a random D-dimensional Euclidean variable x. Our goal is to partition the data set into K clusters. We might think of a cluster as comprising a group of data points whose inter-point distances are small compared with the distances to points outside of the cluster. We could then create µk, composed by K vectors of D dimensions associated with the centers of each cluster. For our scenario, the D dimensions would be 2 if considering latitude and longitude, or 3 if we want to use x, y, z. While the number of VP would be represented by K.
We want to find an assignment of data points to clusters and a combination of µk, such that the sum of the squares of the distances of each data point to its closest µk center is a

minimum. If we define C_k as the representation label of the K different clusters, the mathematical expression would be defined as:

$$\min_{\mu_k, C_k} \left(\sum_{k=1}^{K} \left(\sum_{x_n \in C_k} (distance(x_n, \mu_k))\right)\right)$$

Instead of coding our own algorithm, we used the K-means version provided by Scikit-learn Python library, in pseudocode it would result as:

```
Initialization: choose K random centroids
                    Do classify n samples according to nearest $\mu_i$
                              Compute $\mu_i$ again
                    Until no change in $\mu_i$
          Return $\mu_1$, $\mu_2$… $\mu_c$
End
```

To have a friendly environment, we have also used Matplotlib and NumPy which are widely used together with scikit-learn and for managing big data with Python. Numpy is the fundamental package for scientific computing with Python, able to create powerful N-dimensional array object, sophisticated functions and useful linear algebra capabilities. It’s considered for some people an equivalent of Matlab for Python. Meanwhile, Matplotlib is a plotting library where it’s numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications.

The most basic example of K-means with actual Python code would be as shown below (K=4 clusters):

```
# Imports
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import Handler
from scipy.misc import imread
import matplotlib.cbook as cbook


# Database Prepariation
DB_1=Handler.readprobesinfo()
array_numpy=np.zeros((len(DB_1),2),dtype=object)

# array_numpy is built from x,y,z coordinates of each VP
i=0
for key, value in DB_1.iteritems():
    array_numpy[i,0]=float(DB_1[key][1])
    array_numpy[i,1]=float(DB_1[key][0])
    array_numpy[i,2]=float(DB_1[key][2])
    i+=1
```

```
X=np.array(array_numpy)
kmeans=KMeans(n_clusters=4)

# Train Kmeans
kmeans.fit(X)

# Outputs
centroids=kmeans.cluster_centers_
labels=kmeans.labels_
```

Where DB_1 is built from a function that lists each probe of the input dataset and its information in a dictionary structure. Then, a NumPy array is created in order to properly format the input for training K-means. Once trained, the output centroids and labels are available, which we usually visualize with Matplotlib library.

The following work explored 4 different ways to apply K-means choosing different criteria to select probes within clusters. Although the analysis will be stated in results section, a basic explanation of the methodology applied is described for each one in this section:

I. **K-means Random:** It's the option we started with, we wanted to see its performance to have a comparison base. It consists on picking K random measurements for each input file and simulate iGreedy with set.

II. **K-means Random Trick:** K-means Random algorithm was assigning null values to centroids because no sample were owned by them at the beginning, so instead of having K centers we were obtaining less (fig.13). To solve this problem we "cheated" adding as many as '*K - Resulting centroids'* measurements to each set after applying the algorithm. Finally, these sets are simulated with iGreedy.

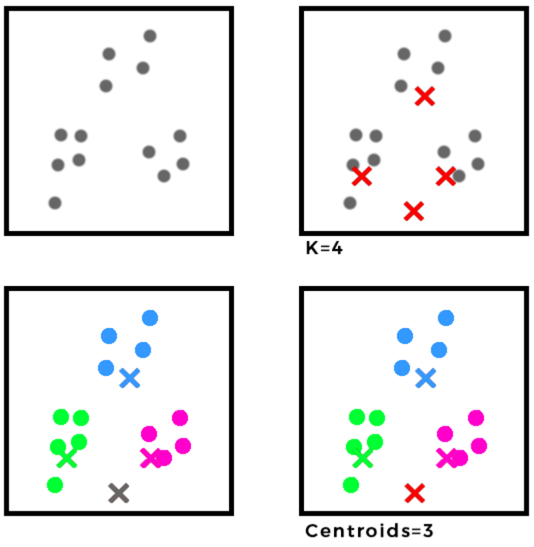


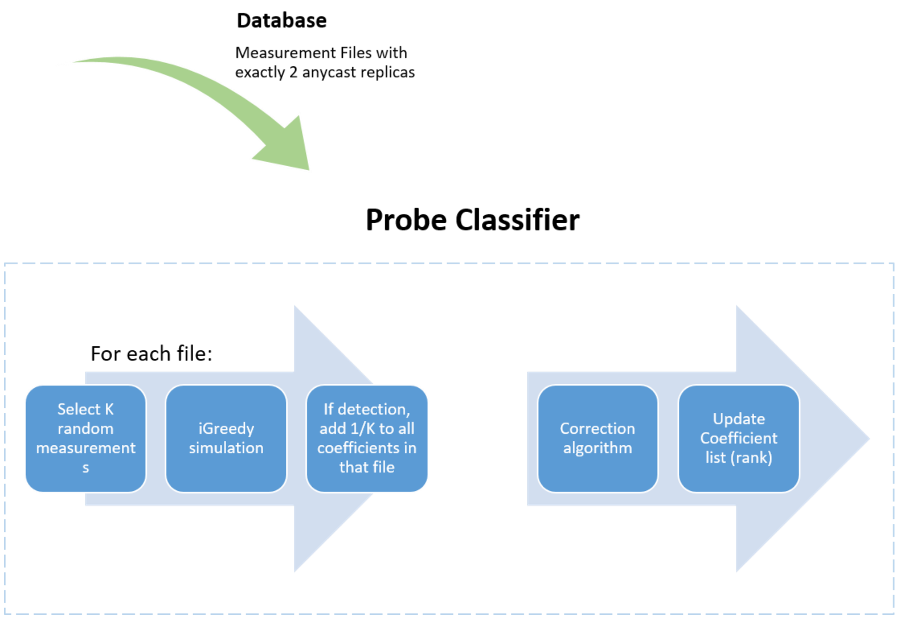
fig. 13 K-means centroid loss

III. **K-means ASN Selection:** The next thing we tried was picking as many different ASN identifications from probes as we could. To achieve this objective, we used two python modules, “ipwhois” which asks the IP of each hostname, and “lookup” which gives domain names information including the ASN number in exchange IPs. Instead of adding a new dimension to K-means we just tried to select for new each cluster new ASN that weren’t selected by other clusters before. We did this way because ASN numbers do not necessarily give information by the proximity of their numbers.

IV. **K-means Country Selection:** We repeated the ASN selection process but using countries, which are provided by our database.

V. **Brute Force Approach**: Due to our bad results, we tested in a brute force approach the random classification without any K-means algorithm. We ran it 1.000.000 times for K= 25, 50, 75, 100 and we picked the best result to check how difficult was our problem, it took 4 days to compute in parallel.
We saw that it was a really difficult problem because the best options were not increasing significantly the amount of detections. But concerning to our methodology, even best cases performed worse than K-means random selection.

VI. **K-means Rank selection:** Although the previous fact, we decided to explore a bit more. If the situation between probes of the same cluster didn’t matter in terms of BGP structure or countries, maybe there were probes in which we could trust more than others, maybe there were super-probes with more power to detect than others. To confirm our theory we built an automatized script (fig. 25) able to create a list of probes with associated numeric values about its ability to detect in multiple scenarios. Using a set of measurement files as input, the algorithm gives an organized list associating each probe a value. The higher the coefficient is, the more importance is given to that probe.
For each file, K probes are randomly selected. If the result of iGreedy simulation gives an anycast instance, then 1/K value is added to all the probes of the input.
Since the vantage points do not appear the same amount of times in files, if we left the values this way the most popular probes would have a higher value because they would participate more. To solve this problem a correction function is also applied, it counts the times each probe appears in the input and divides their coefficient by this value. Finally, the coefficients are normalized.



Database
Measurement Files with exactly 2 anycast replicas
Probe Classifier
For each file:
Select K random measurements
iGreedy simulation
If detection, add 1/K to all coefficients in that file
Correction algorithm
Update Coefficient list (rank)

fig. 14 Rank performer

Once created our organized list, we modified the K-means selector of probes to pick from each cluster the vantage point with the highest value in the rank similarly to country or ASN systems.

4. Results

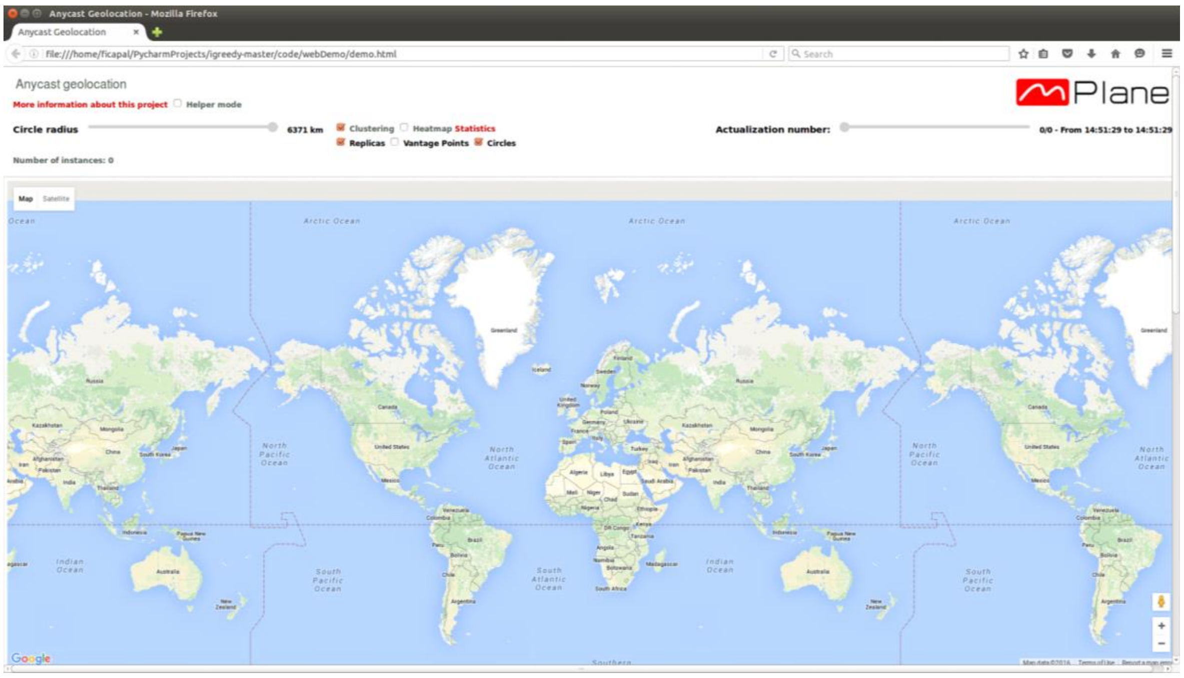
Similarly to Methodology structure, the explanation in the following sections will be divided in two, the Illustration section, linked with visualization results, and the vantage points selection improvement.

4.1. Visualization

The results concerning about the visualization are displayed in the following pictures. This is the answer of iGreedy after targeting the IP address 8.8.8.8 (www.google.com) with 500 RIPE Atlas Vantage Points.

Until the first information packet is retrieved, the map stays without any mark.

Arrival 0:


Anycast Geolocation - Mozilla Firefox

file:///home/ficapal/PycharmProjects/igreedy-master/code/webDemo/demo.html

Anycast geolocation

More information about this project ☐ Helper mode

Circle radius 6371 km ☑ Clustering ☐ Heatmap Statistics ☑ Replicas ☐ Vantage Points ☑ Circles

Actualization number: 0/0 - From 14:51:29 to 14:51:29

Number of instances: 0

fig. 15 Arrival 0

When the first packet arrives, the information is actualized. The first instances are being detected and the bar on the right-top with the title “Actualization number” can now be slid. This tool will show the previous states that have happened in the map indicating with the format hh:mm:ss to hh2:mm2:ss2 the interval being displayed.

Arrival 1:

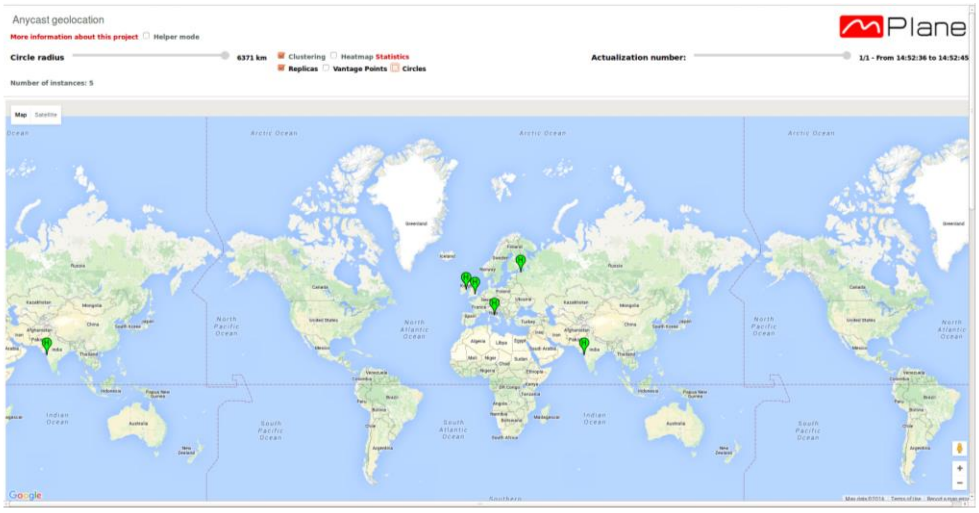

Anycast geolocation
More information about this project ☐ Helper mode
Circle radius 6371 km ✓ Clustering ☐ Heatmap Statistics ✓ Replicas ☐ Vantage Points ☐ Circles
Actualization number: 1/1 - From 14:52:36 to 14:52:45
Number of instances: 5

fig. 16 Arrival 1

Arrival 5:

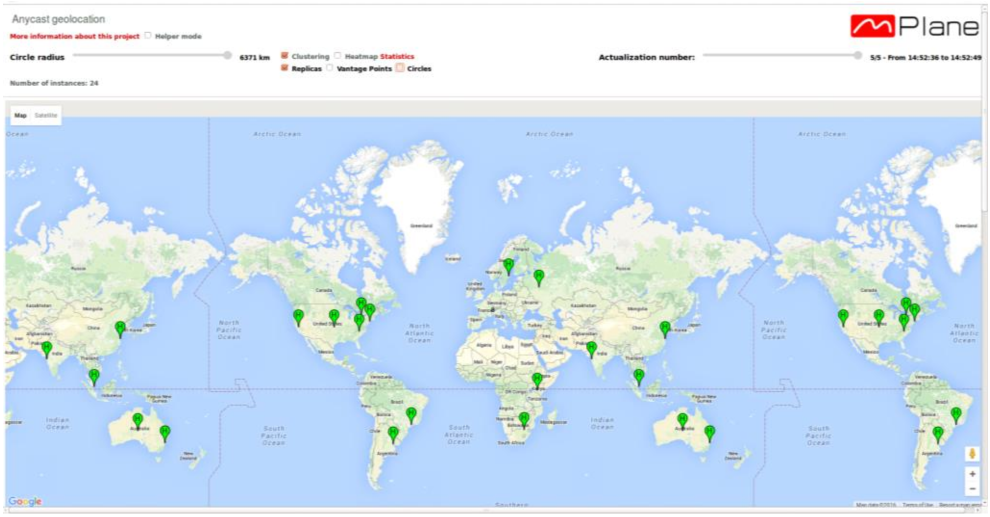
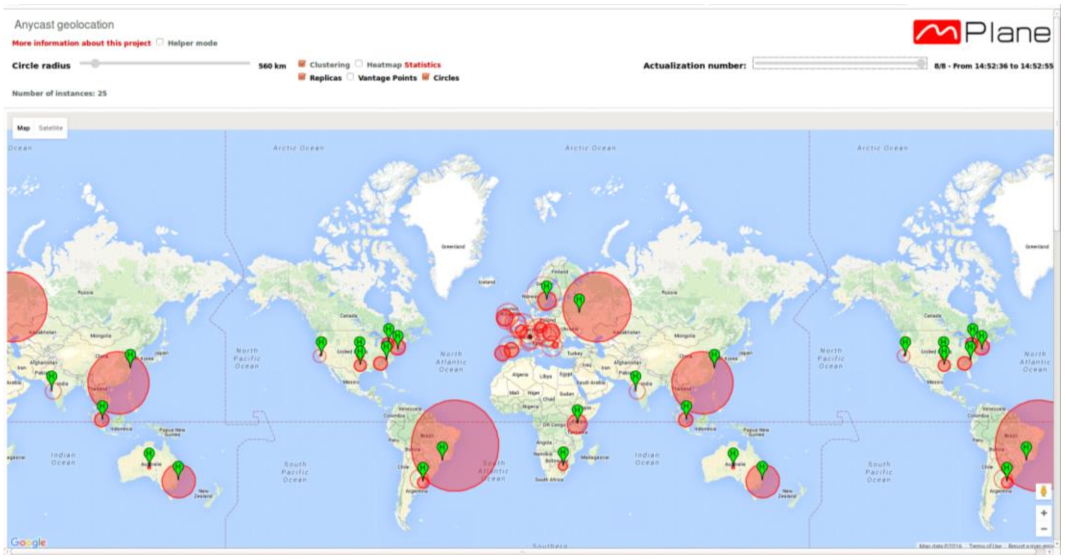

Anycast geolocation
More information about this project ☐ Helper mode
Circle radius 6371 km ✓ Clustering ☐ Heatmap Statistics ✓ Replicas ☐ Vantage Points ☐ Circles
Actualization number: 5/5 - From 14:52:36 to 14:52:49
Number of instances: 24

fig. 17 Arrival 5

It's been some seconds since the last packet and the screen hasn't shown any actualization, which means it should be quite complete. The filled red circles are the ones that have detected an instance, the non-filled indicate a mere measurement.

Arrival 8:


Anycast geolocation

More information about this project ☐ Helper mode

Circle radius 560 km ☑ Clustering ☐ Heatmap Statistics ☑ Replicas ☐ Vantage Points ☑ Circles

Actualization number: 8/8 - From 14:52:36 to 14:52:55

Number of instances: 25

fig. 18 Last Arrival

Moving the actualization slider we select the desired time range (fig. 19).

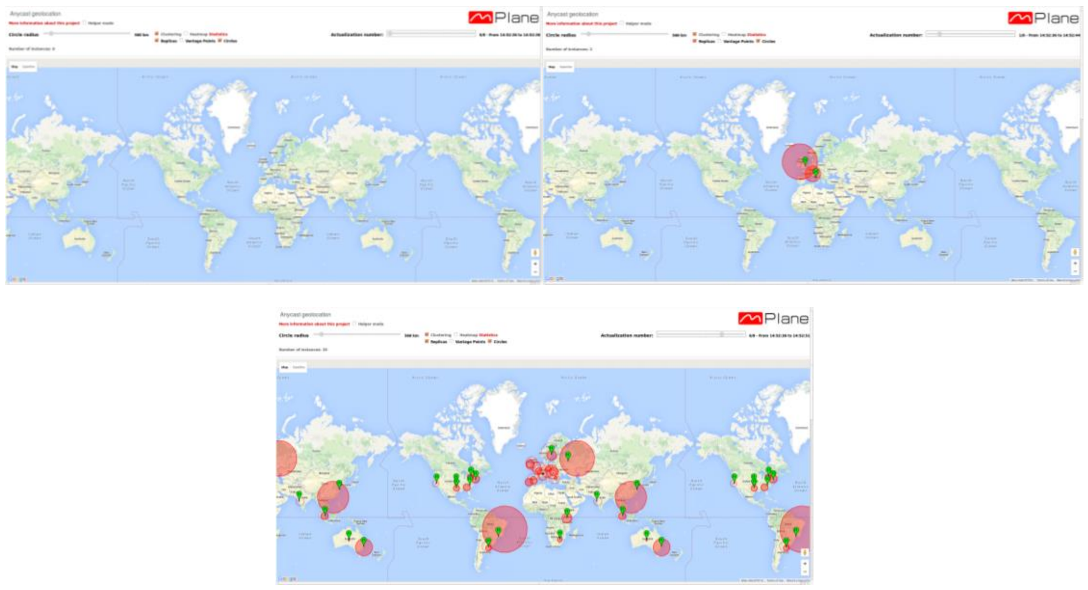
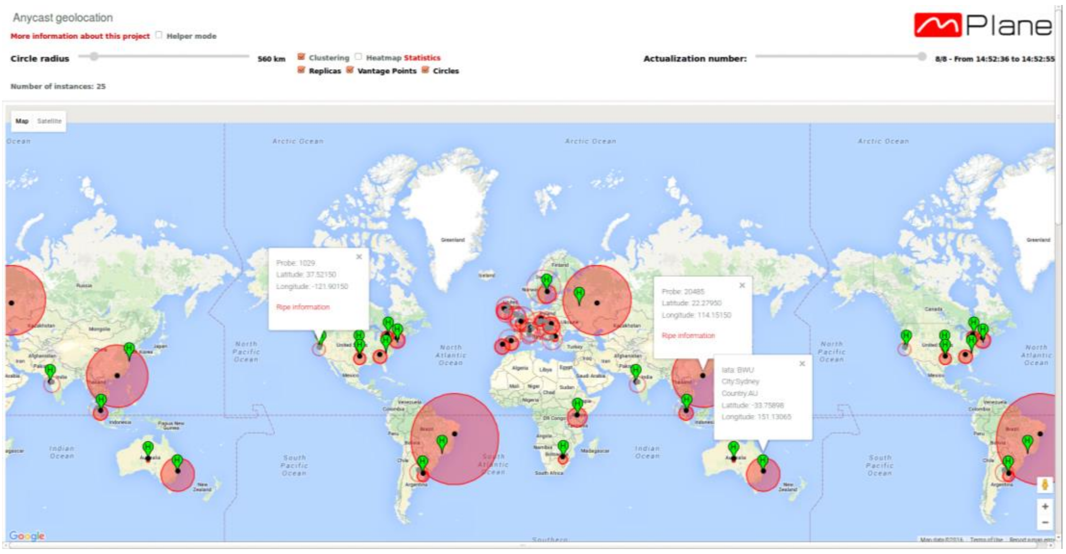


fig. 19 Slider

There are also several other options such as circles visualization with the choice of selecting their maximum radius, or the Vantage Points. They can also be visualized in all the past states of the simulation.

Clicking a probe its information pops up (fig. 20):


Anycast geolocation

More information about this project ☐ Helper mode

Circle radius 560 km ☑ Clustering ☐ Heatmap Statistics ☑ Replicas ☑ Vantage Points ☑ Circles

Actualization number: 8/8 - From 14:52:36 to 14:52:55

Number of instances: 25

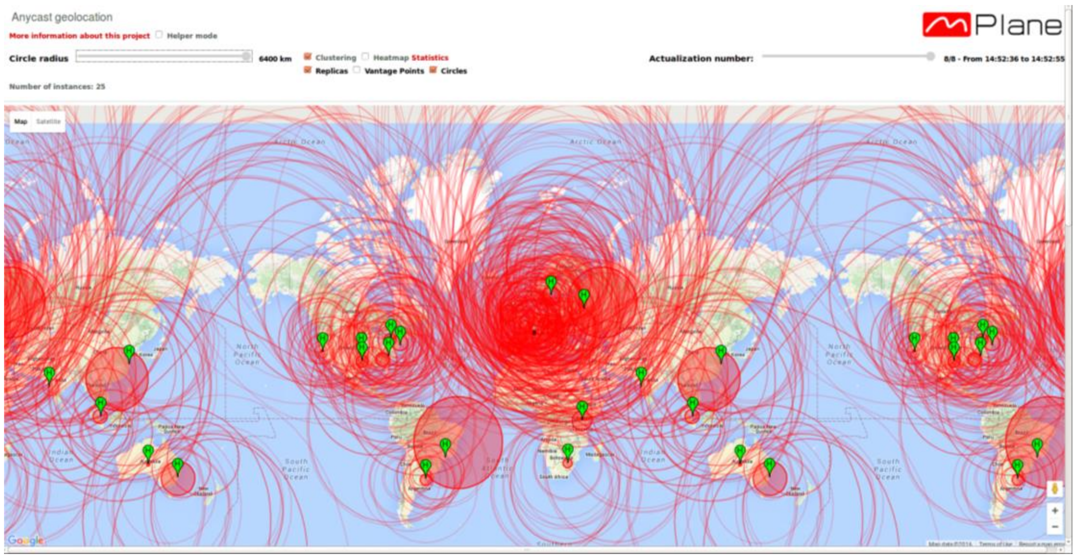
Probe: 1029
Latitude: 37.52150
Longitude: -121.90150
Ripe information

Probe: 20485
Latitude: 22.27950
Longitude: 114.15150
Ripe information

Iata: BWU
City:Sydney
Country:AU
Latitude: -33.75898
Longitude: 151.13065

fig. 20 Probe info

It is possible to choose a circle radius limit from 300 to 6400 Km (fig. 21).


Anycast geolocation

More information about this project ☐ Helper mode

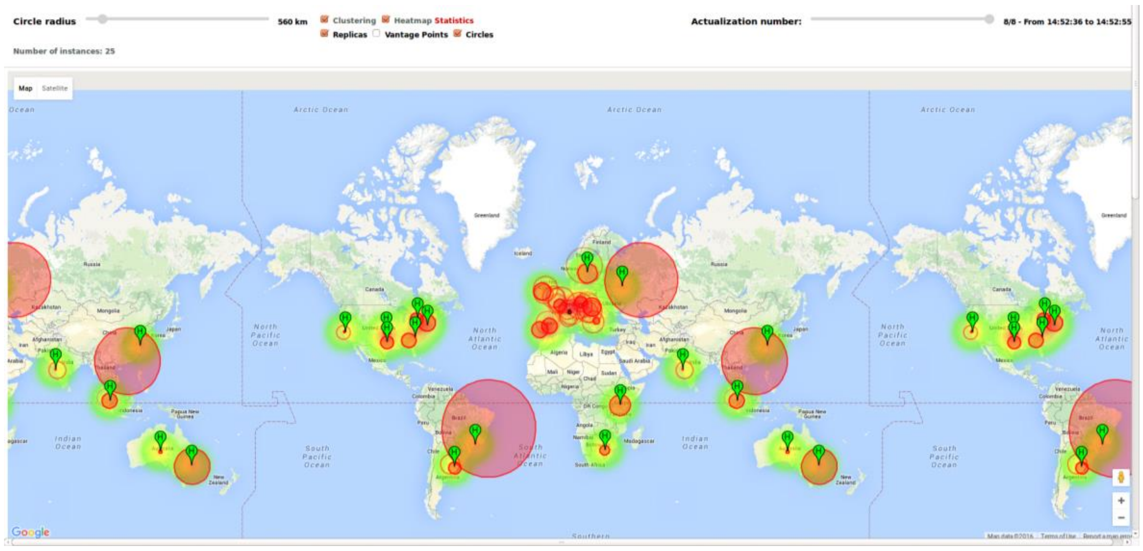
Circle radius 6400 km ☑ Clustering ☐ Heatmap Statistics ☑ Replicas ☐ Vantage Points ☑ Circles

Actualization number: 8/8 - From 14:52:36 to 14:52:55

Number of instances: 25

fig. 21 Instance circles

A heat map showing the ping activity is also available (fig. 22).


Circle radius 560 km ✓ Clustering ✓ Heatmap Statistics
✓ Replicas ☐ Vantage Points ✓ Circles
Number of instances: 25

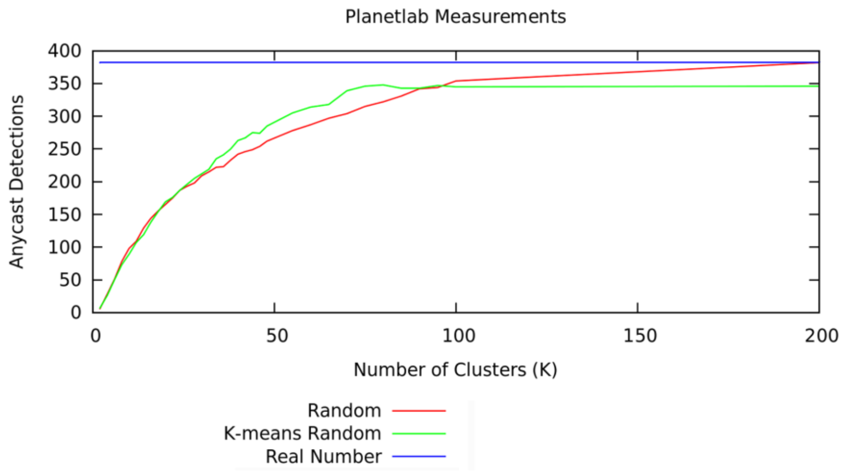
Actualization number: 8/8 - From 14:52:36 to 14:52:55

fig. 22 Heat map

I can conclude that the results are the ones our team expected, iGreedy has its previous functionalities and we have added new ones, and both work in streaming mode.

4.2. VP Selection

The hypothesis we wanted to demonstrate is that the geographical proximity of the probes matter to solve our problem. To prove it, we ran K-means over each measurement file, using K as the number of desired clusters fig. 20.



Our result improved random results for 25 < K < 90, but after the initialization the algorithm was assigning null values to centroids because no sample were owned by them at the beginning, so instead of having K centers we were obtaining less. We were picking as many VP as Centroids which was producing less input measurements for iGreedy simulation, which was logically detecting less replicas.
We solved that problem randomly adding *K – Initial centroids* after performing K-means. The improvement is shown in the graph fig. 21.

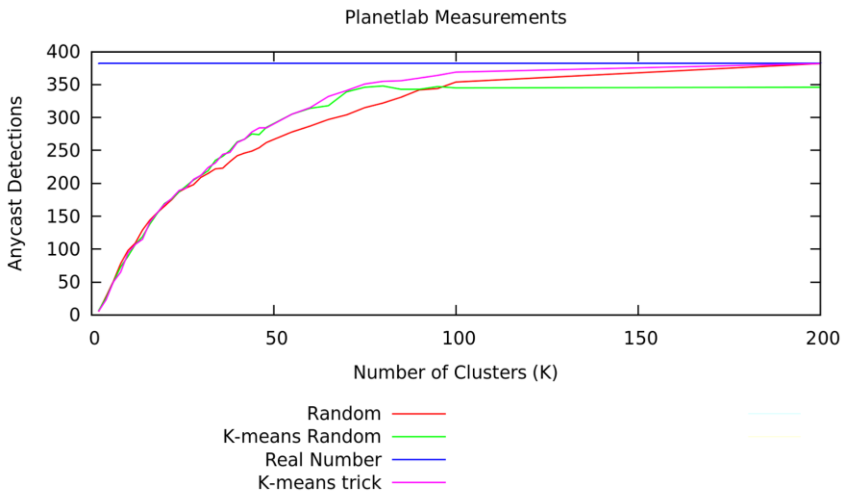


fig. 24 K-means Trick vs None

Leaving apart the improvement from the random simulation, it was quite obvious that randomly selecting the VP inside each cluster shouldn't give the best results. Watching the map we realized that clusters were spread over several countries when they were big, and our new hypothesis was that trying to select VP from as many countries as possible would help for internet topology reasons.
From this point, we started using the dataset explained in *3.2.1-Database* to make sure our results were as accurate and reliable as possible.

We proceeded to modify K-means algorithm assigning for each cluster, vantage points from countries that were not already chosen, if possible. Otherwise a random one was picked from that cluster.
As shown in the graph (fig. 25) we were wrong because it performed nearly similar to the simple version.

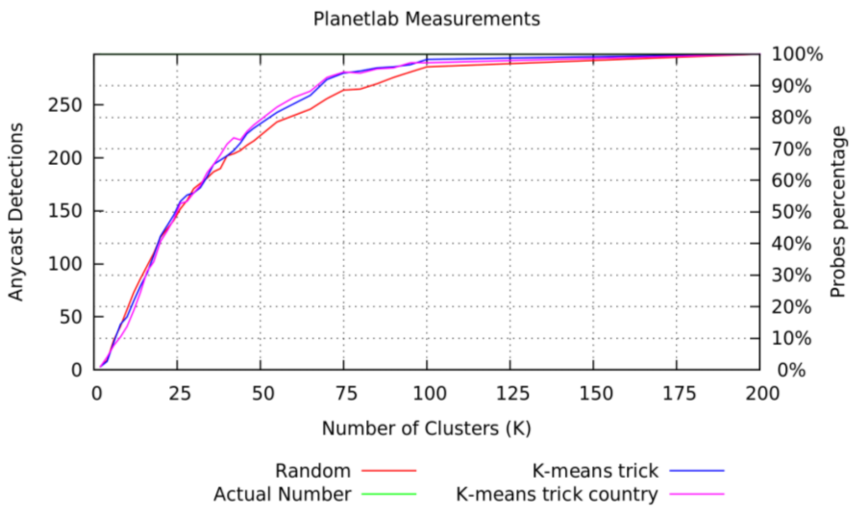


fig. 25 K-means trick vs Country

Maybe if we were trying distinguish techniques from the internet topology we would have to pick a BGP metric, so we tried the same algorithm but instead of using the country of each probe this time we used the ASN (fig. 26), which is a number that identifies each autonomous system.

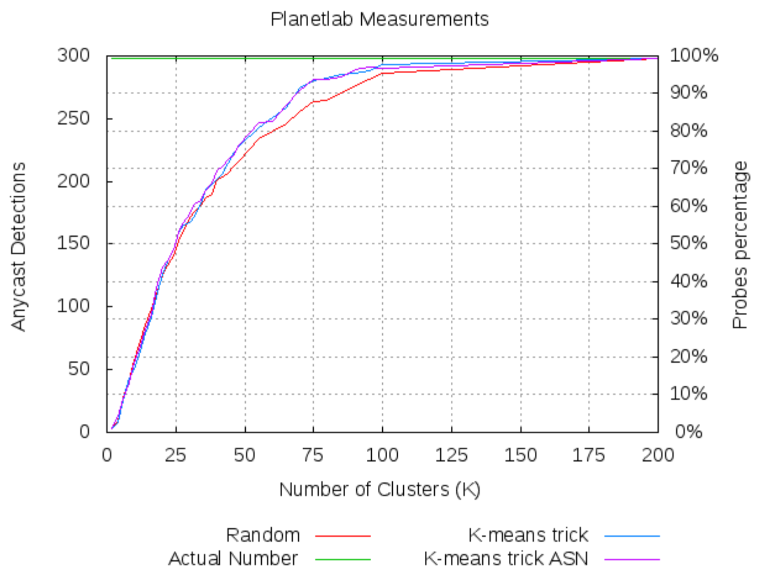


fig. 26 K-means trick vs ASN

Our results (fig.27) were stable but unsuccessful, ASN and country versions were performing similarly to basic version of K-means.

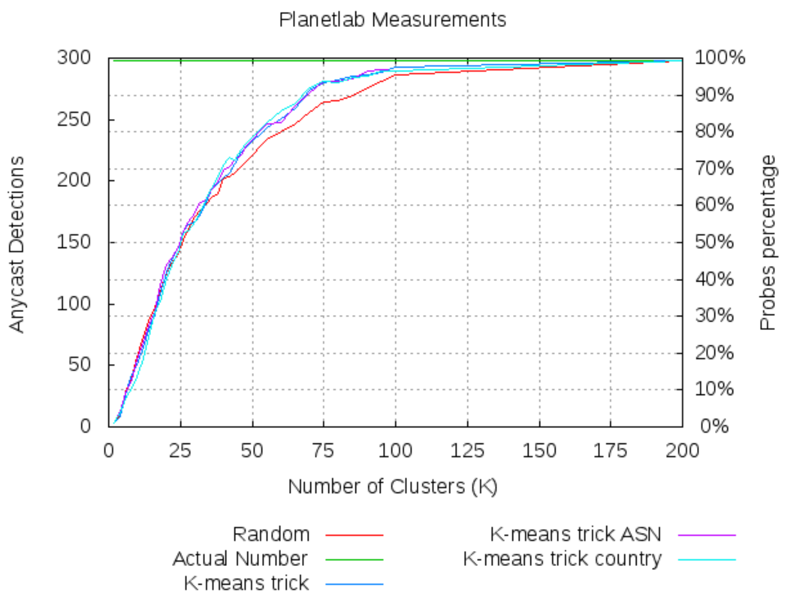


fig. 27 Comparison ASN vs Country

As explained in *Methodology section 3.2.2*, we tested the random algorithm 1.000.000 times to have a reference of how difficult was our problem of optimizing the VP selection. We concluded it was though since the variance for each K case was not high much and K-means-random was giving always better results.

Once we had the brute force approach results, K-means didn't seem as bad as before because it was improving the best case of 1.000.000 random simulations, for this reason we gave it a last try.

We built the rank system which quickly gave significant improvements (fig. 28):

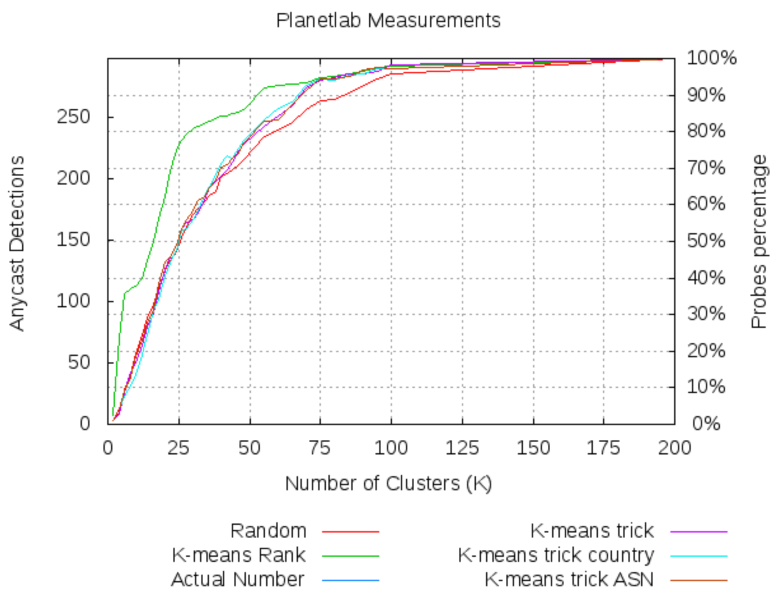


fig. 28 Comparing all K-means

In the next Boxplot graph (fig. 29) the minimum, maximum, average, first and third quantile of the data are also displayed, which indicate the variance of our results. Our conclusion is that the rank version of K-means is also stable.

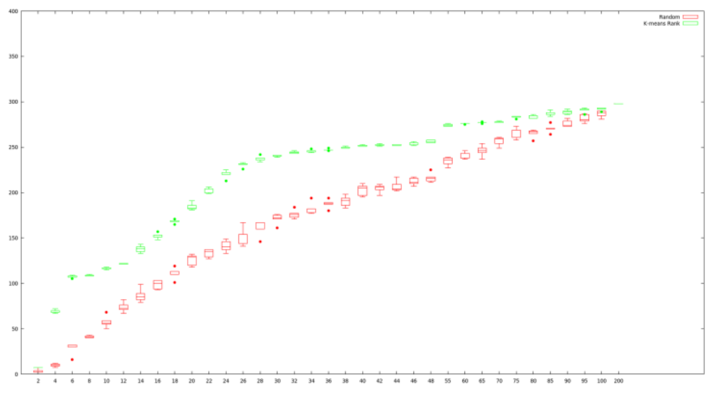


fig. 29 Boxplot K-means rank

5. Budget

This research project has been developed using open software, or with partners resources. Its cost mainly comes in the shape of the time spent by the involved researchers.

	Amount	Wage	Hours spent	Total
Undergraduate engineer	1	8 € /h	500	4.000 €
PhD student	1	16 € /h	50	800 €
			Final Cost:	4.800 €

table 3 Budget

6. Conclusions:

6.1. Visualization:

After this work I can conclude that all the features in the new streaming version of iGreedy work perfectly. The software provides a better visualization and the new structure provides the desired and expected information.

This new version is still as robust and lightweight as the previous one because the main algorithm hasn't changed. Even a single latency sample per vantage point, from a few vantage points is enough to provide satisfactory enumeration and geolocation performance. In reason of its lightweight and its low computational complexity, the technique is capable to perform continuous large scale measurements, which will be helpful for mapping the Internet.

Personally, I've understood how hard is to pick an ongoing complex software to improve the code that was already written. The process of understanding of what was already done was tough, especially with things concerning the webpage interface because I didn't have a strong knowledge basis on that topic. With the iGreedy algorithm and data management was easier because I'm used to Object-Oriented languages similar to Python such as Java or C#.

6.2. VP Selection:

The aim of the second part was trying to determine a way to minimize the number of probes required to detect BGP Hijacks. In this work we have focused on K-means clustering to balance the distance between probes.

After this thesis, we have discovered that what matters the most when selecting geographically close vantage points is its power to detect in general. It seems that not all the probes have the same power since there were probes much better positioned in our rank than others for nearly the same positions.

With the brute force approach we didn't only see we were trying to solve a difficult problem, but also that our initial hypothesis about geographical discrimination was quite right because our random K-means was performing much better than the brute force best cases.

Keeping in mind that this algorithm increases the iGreedy detections instead of improving its capacity to detect the actual hijacks based on [1] JSAC, it's fair to say that the global algorithm is still not perfect since the number of false negative responses quite high.

Finally, our algorithm seems strong when applying the rank to choose within each cluster, improving the probe cost compared to random in 31 for the 90% of detections in our database. Which is a significant amount considering it represents a reduction of 37% probes for a decent result.

7. Future development

We have tested different ways of vantage points initialization for K-means algorithm in this project. Although our results are trustworthy, they are particularized for a specific database where the probes are carefully selected.

When we built our database, we took care of bugged measurements provided by Planetlab or RIPE Atlas. To do so, we searched for detections which were only happening for a combination of 2 specific probes. With the results we realized that there was one probe which was appearing the 30% of times, meaning that percentage was only reachable if that particular probe was present in the input.

When we proceeded to check information about it we found that its latitude and longitude were 1, which would imply being in the middle of the ocean. Probably, it was due to an initialization default problem in Planetlab database, but it was still affecting negatively our results.
In the next figures the difference can be appreciated. The fig. 26 is obtained with the dirty database, and the fig. 27 is the clean one, which grows faster for K between 0 and 80.

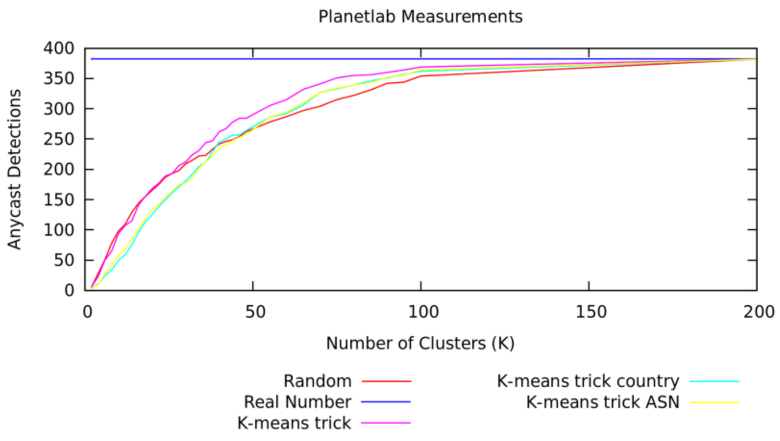


fig. 30 Dirty dataset

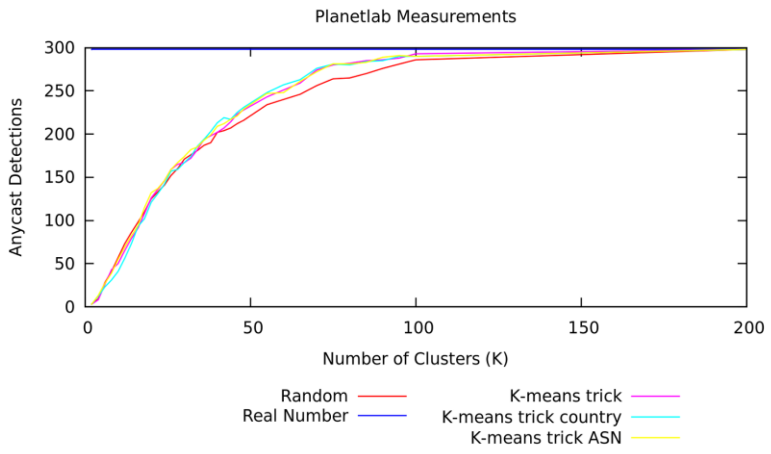


fig. 31 Clean dataset

We should build an automatic software to detect wrong probe content and remove them from the measurements dataset instead of repeating this procedure manually.

We have also left open a new discussion about what we call super-probes. Analyzing the features which make the common probes different to powerful ones could be the key to discover the proper way to choose our vantage points.

Bringing back the brute force approach and its conclusions, we are now surer that the geographical organization of probes should be one of the key factors for iGreedy initialization. Given the fact that random K-means version was not highly improving our system but the rank version was, we propose the research of new methods such as convolutional classification also with super-probes key features.

Bibliography:

[1] [JSAC-16] Cicalese, Danilo, Joumblatt, Diana , Rossi, Dario, Buob, Marc-Olivier , Auge, Jordan and Friedman, Timur , Latency-Based Anycast Geolocalization: Algorithms, Software and Datasets .V.C. Gungor, B. Lu, G.P. Hancke. "Opportunities and challenges of Wireless Sensor Networks in Smart Grid". *IEEE Transactions on Industrial Electronics*, vol. 56, no. 10, pp. 3557-3564, October 2010. DOI: 10.1109/TIE.2009.2039455.

[2] https://www.planet-lab.org

[3] https://www.ripe.net

[4] http://scikit-learn.org/stable/

[5] D. Madory, C. Cook, and K. Miao, “Who are the anycasters,” Nanog, 2013.

[6] X. Fan, J. S. Heidemann, and R. Govindan, “Evaluating anycast in the domain name system.” in Proc. IEEE INFOCOM, 2013.

[7] P. Boothe and R. Bush, “DNS Anycast Stability: Some Early Results,” CAIDA, 2005.

[8] S. Sarat, V. Pappas, and A. Terzis, “On the use of anycast in DNS,” in Proc. ICCCN, 2006.

[9] D. Karrenberg, “Anycast and bgp stability: A closer look at dnsmon data,” Nanog, 2005.

[10] H. Ballani and P. Francis, “Towards a global IP anycast service,” in Proc. ACM SIGCOMM, 2005.

[11] H. Ballani, P. Francis, and S. Ratnasamy, “A measurement-based deployment proposal for ip anycast.” in Proc. ACM IMC, 2006.

[12] Z. Liu, B. Huffaker, M. Fomenkov, N. Brownlee, and K. C. Claffy, “Two days in the life of the DNS anycast root servers.” in Proc. of PAM, 2007.

[13] M. Levine, B. Lyon, and T. Underwood, “Operational experience with TCP and Anycast,” Nanog, 2006.

[14]http://ciscorouterswitch.over-blog.com/article-bgp-protocol-is-essential-in-your-ip-network-115059468.html

[15] M. Lad, D. Massey, D. Pei, Y. Wu, B. Zhang, and L. Zhang, "PHAS: A Prefix Hijack Alert System," in Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15, USENIX-SS'06, (Berkeley, CA, USA), USENIX Association, 2006.

[16] V. Khare, Q. Ju, and B. Zhang, "Concurrent prefix hijacks: Occurrence and impacts," in Proceedings of the 2012 ACM Conference on Internet Measurement Conference, IMC '12, (New York, NY, USA), pp. 29-36, ACM, 2012.

[17] Z. Zhang, Y. Zhang, Y. C. Hu, Z. M. Mao, and R. Bush, "iSPY: Detecting IP Prefix Hijacking on My Own," IEEE/ACM Trans. Netw., vol. 18, pp. 1815-1828, Dec. 2010.

[18] C. Zheng, L. Ji, D. Pei, J. Wang, and P. Francis, "A Light-weight Distributed Scheme for Detecting Ip Prefix Hijacks in Real-time," in Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '07, (New York, NY, USA), pp. 277-288, ACM, 2007.

[19] X. Hu and Z. M. Mao, "Accurate Real-time Identification of IP Prefix Hijacking," in Proceedings of the 2007 IEEE Symposium on Security and Privacy, SP '07, (Washington, DC, USA), pp. 3-17, IEEE Computer Society, 2007.

[20] X. Shi, Y. Xiang, Z. Wang, X. Yin, and J. Wu, "Detecting prefix hijackings in the internet with argus," in Proceedings of the 2012 ACM Conference on Internet Measurement Conference, IMC '12, (New York, NY, USA), pp. 15-28, ACM, 2012.

[21] https://www.caida.org/funding/hijacks [8] D. Cicalese D. Joumblatt, D. Rossi, J. Auge, T. Friedman Characterizing IPv4 Anycast Adoption and Deployment . In ACM CoNEXT, Heidelberg, December 2015.

Appendices:

How to run iGreedy:

Usage:

```
igreedy.py (-i INPUT | -m MEASUREMENT | -h) [-p PLANETLAB] [-r RIPE] [-o OUTPUT]
          [-g GROUNDTRUTH] [-a ALPHA (1)] [-n NOISE (0)] [-t TRESHOLD (\infty)]
          [-b (false)]
```

Mandatory:

```
-i input file
-m IPV4 or IPV6 (real time measurements using the RIPE Atlas and/or PlanetLab vantage points in datasets/*-vps)
```

Optional:

```
-o output prefix (.csv,.json)
-b browser (visualize a GoogleMap of the results in a browser)
-g measured ground truth (GT) or publicly available information (PAI) files
   (format: "hostname iata" lines for GT, "iata" lines for PAI)
-a alpha (tune population vs distance score; was 0.5 in INFOCOM'15, now defaults to 1, more details in TECHREP-16 or INFOCOM'15)
-t threshold (discard disks having latency larger than threshold to bound the error; discouraged)
-n noise (average of exponentially distributed additive latency noise; only for sensitivity)
-r vantage points file or number of random vantage points (datasets/ripe-vps) for real time measurements from Ripe Atlas
-p vantage points file or number of random vantage points (datasets/planetlab-vps)for real time measurements from PlanetLab
-s Streaming mode
```

Run iGreedy on existing measurement, example:

```
./igreedy -i datasets/measurement/f-planetlab
```

Run iGreedy on new measurement, example:

```
./igreedy -m 192.5.5.241 -p 20 -r 5 -b
```

Run Stream mode, example:

```
./igreedy -m 192.5.5.241 -p 20 -r 5 –b -s
```

Glossary

VP: Vantage Points.

URL: Uniform Resource Locator

ASN: Autonomous System Network.

BGP: Border Gateway Protocol

JSON: JavaScript Object Notation

CSV: Coma Separated Values