Achieving scale: Large scale active measurement from PlanetLab (LABS)

Jordan Augé, Marc-Olivier Buob, Timur Friedman (UPMC)

4th PhD School on Traffic Monitoring and Analysis (TMA), 2014

OBJECTIVES

Measuring and monitoring widely distributed infrastructures, such as the Internet, is a challenging task that requires the use of sophisticated tools and methodology.

In this lab, we set as an objective to measure the deployment of the Google Public DNS services, well known under the 8.8.8.8 IP address, and we will also investigate root DNS servers.

STRUCTURE OF THE DOCUMENT

The subject is composed of four parts.

The first part, presents a set of tools and webservices of interest for those willing to better understand the Internet architecture, and will introduce some basic concepts that will help us get a better understanding on how the DNS servers are managed.

In the second part, we will target more specifically the Google Public DNS and try to get some evidences about its architecture, thanks to carefully designed measurements from PlanetLab nodes. This will allow us to design a measurement methodology that we will appy on our measurements.

Before running large scale measurements, the third part will familiarize yourself with the tools we have previously presented, namely:

> TDMI : for running large scale distributed measurements on top of PlanetLab,

> TopHat : for aggregating and combining heterogeneous datasets,

> Manifold: the software that runs those different services.

As an application case, we will characterize the location of nodes in the PlanetLab testbed to evaluate whether it is representative of the whole Internet.

Finally, in the last part, we will proceed to large scale measurements towards both the root DNS servers and Google Public DNS. The first series of measurements will allow us to benchmark our method, since there is plenty of available information for DNS servers. We will then map the deployment of Google Public DNS servers as much as possible.

REQUIREMENTS

This lab refers to some tools and datasets available on a specifically built virtual machine based on VirtualBox. See the TMA PhD school website for more information.

PRELIMINARY

We have already set up a PlanetLab slice and created a SSH keypair for the lab, that will allow you to access the nodes. Please execute the following commands on your virtual machine. They will retrieve the keypair as well as install a set of dependencies that will be useful for the lab.

```
wget http://www.top-hat.info/download/tma-2014/tma2014-scale-setup.sh
chmod +x tma2014-scale-setup.sh
su
./tma2014-scale-setup.sh
```

A. INTERNET ARCHITECTURE

The Internet is made of the interconnection of multiple independent networks, called *Autonomous Systems*, or AS. These ASes exchange traffic according to routing agreements implemented via the BGP protocol.

A.I IP addresses, AS, and their geography

A.I.1. Can you name different types of Autonomous Systems ? Give an example for each type.

Solution:

- Academic: Renater, JANET, etc.
- ISP: Orange;
- Content provider: Google, Akamai, etc.;
- Transit provider: Level 3

A.I.2. Determine your IP address (use the `ifconfig` command).

Solution: Two possibilities:

- a public IP address:

```
/sbin/ifconfig eth0

eth0      Link encap:Ethernet  HWaddr 00:21:cc:60:58:6e
          inet adr:132.227.127.116  Bcast:132.227.127.255  Masque:255.255.255.0
          adr inet6: fe80::221:ccff:fe60:586e/64 Scope:Lien
[...]
```

- a private IP address:

```
/sbin/ifconfig wlan0

wlan0     Link encap:Ethernet  HWaddr 8c:a9:82:ae:50:4a
          inet adr:192.168.0.160  Bcast:192.168.0.255  Masque:255.255.255.0
          adr inet6: fe80::8ea9:82ff:feae:504a/64 Scope:Lien
[...]
```

A.I.3. Determine your public IP address. Can you geolocalize it ? In which AS is it located ? You can copy/paste the URL of the following webservices : ifconfig.me[1], MaxMind[2] and Team Cymru IP-to-ASN mapping service[3]

A.I.4. Can you explain how Team Cymru manages to provide such a mapping ?

Solution: For the public IP address:

```
curl ifconfig.me
```

```
132.227.127.116
```

For the private IP address:

```
curl ifconfig.me
```

```
137.194.164.55
```

```
https://www.maxmind.com/fr/geoip_demo
```

Adresse IP	Code de pays	Localisation	Code postal	Coordonnées
132.227.127.116	FR	Paris, Paris, Île-de-France, France, Europe		48.8667, 2.3333
137.194.164.55	FR	Paris, Paris, Île-de-France, France, Europe		48.8667, 2.3333

FAI	Organisation	Domaine	Indicatif départemental
Universite Pierre et Marie Curie	Universite Pierre et Marie Curie		
Telecom ParisTech	Telecom ParisTech	enst.fr	

```
https://asn.cymru.com/
```

AS	IP	AS Name
1307	132.227.127.116	FR-U-JUSSIEU-PARIS,FR
1712	137.194.164.55	FR-RENATER-ENST Ecole Nationale Superieure des Telecommunications,,FR

A.II Google Public DNS

In this lab, we will study the DNS service offered by Google under the 8.8.8.8 public IP address.

> Google Public DNS is a Domain Name System (DNS) service offered by Google. It functions as a recursive name server providing domain name resolution for any host on the Internet. The service was announced on 3 December 2009, in an effort described as making the web faster and more secure. *source: Wikipedia - Google Public DNS*

A.II.1. Under which AS is 8.8.8.8 hosted ?

[1] `http://ifconfig.me`
[2] `https://www.maxmind.com/fr/geoip_demo`
[3] `https://asn.cymru.com/`

Solution:

AS	IP	AS Name
15169	8.8.8.8	GOOGLE - Google Inc.,US

A.II.2. Can you geolocate it using MaxMind ? Do you have an idea about the reason ?

Solution:

Adresse IP	Code de pays	Localisation	Code postal
8.8.8.8	US	États-Unis, Amérique du Nord	

Coordonnées	FAI	Organisation	Domaine	Indicatif départemental
38, -97	Level 3 Communications	Google		

MaxMind is more accurate for IP addresses located at the edge of the network, where the users and associated revenues are.

A.III Internet paths

We will now study the path taken by your traffic when you contact the Google DNS server. We will start using some PlanetLab nodes. You can connected to the node `ple2.ipv6.lip6.fr` by SSH via the following command:

```
ssh upmc_tma@ple2.ipv6.lip6.fr
```

Your user is the name of the slice, composed of your origin institution (here UPMC) and an identifier (here: tma). Authentication is made thanks to your private key.

A.III.1. Thanks to the `traceroute` command, print the IP- and AS-level path towards 8.8.8.8. You can use the command `man traceroute` to get some help with its parameters.

Solution:

```
$ traceroute -A 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  132.227.62.65 (132.227.62.65) [AS1307/AS2200]  0.480 ms  0.403 ms  0.377 ms
 2  10.1.1.1 (10.1.1.1) [AS65534]  1.471 ms  1.470 ms  1.450 ms
 3  r-interco2-lab-upmc.reseau.jussieu.fr (134.157.167.124) [AS1307/AS2200]  2.895 ms  3.253 ms  3.233 ms
 4  r-jusrap-reel.reseau.jussieu.fr (134.157.254.124) [AS1307/AS2200]  3.194 ms  3.178 ms  3.151 ms
 5  interco-6.01-jussieu.rap.prd.fr (195.221.127.181) [AS2200]  2.716 ms  2.691 ms  2.113 ms
 6  vl165-te3-2-jussieu-rtr-021.noc.renater.fr (193.51.181.102) [AS2200]  3.242 ms  6.136 ms  6.126 ms
 7  te0-1-0-3-paris2-rtr-001.noc.renater.fr (193.51.189.225) [AS2200]  28.169 ms  28.156 ms  28.129 ms
 8  * * *
 9  193.51.182.197 (193.51.182.197) [AS2200]  5.938 ms  5.913 ms  5.889 ms
10  72.14.238.234 (72.14.238.234) [AS15169]  6.216 ms  6.203 ms  7.051 ms
11  209.85.245.70 (209.85.245.70) [AS15169]  4.767 ms 209.85.245.83 (209.85.245.83)
        [AS15169]  4.794 ms 209.85.245.72 (209.85.245.72) [AS15169]  4.557 ms
12  209.85.246.167 (209.85.246.167) [AS15169]  9.765 ms 216.239.43.233 (216.239.43.233)
        [AS15169]  9.735 ms  10.929 ms
13  72.14.238.41 (72.14.238.41) [AS15169]  10.919 ms 72.14.238.43 (72.14.238.43)
        [AS15169]  9.617 ms 216.239.49.45 (216.239.49.45) [AS15169]  9.589 ms
14  * * *
15  google-public-dns-a.google.com (8.8.8.8) [AS15169]  9.415 ms  9.328 ms  9.257 ms
```

A.III.2. How is the mapping IP-to-ASN done by `traceroute` ? You can use Google to search for some information.

Solution: According to the man:

```
-A     Perform AS path lookups in routing registries and print results
directly after the corresponding addresses.
```

You can try:

```
whois -h whois.radb.net <network_IP>
```

```
route:      8.8.8.0/24
descr:      Google
origin:     AS15169
notify:     radb-contact@google.com
mnt-by:     MAINT-AS15169
changed:    radb-contact@google.com 20140326
source:     RADB
[...]
```

A.III.3. This node was located in Paris, France. Do the same measurement from another PlanetLab node in Denver, in the US: `planetlab2.cs.du.edu`.

Solution:

A.III.4. What types of the AS traversed ? You can use Google to find some information.

Solution: They are mostly academic networks (RENATER, UCAR, we would find GEANT from other nodes, etc.)

A.III.5. If this was representative of PlanetLab, what would be your conclusions about the types of networks involved in the testbed ? What consequence might this have on measurements ?

Solution: If PlanetLab is mostly deployed in academic networks, we might have a strong bias in our measurements towards such network (which might be operated differenty). We might also lack a view from commercial networks, and miss some aspects of the whole Internet.

A.III.6. Again, supposing this was representative, what does this suggest about the peering agreements of Google for 8.8.8.8 with other ASes ?

Solution: Google seems to have peerings agreements with a large number of AS.

B. ANYCAST

In this section, we will continue using PlanetLab nodes to understand the routing behaviour of 8.8.8.8. Open two terminals connected to the nodes we used previously, and which are located in different locations in the world (EU and US):

> ple2.ipv6.lip6.fr

> planetlab2.cs.du.edu

B..1. From one node, launch a few `ping` probes towards destination 8.8.8.8 (use the -c option). Is the performance good ?

Solution:

```
ping -c 10 8.8.8.8
```

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_req=1 ttl=45 time=8.81 ms
64 bytes from 8.8.8.8: icmp_req=2 ttl=45 time=8.82 ms
64 bytes from 8.8.8.8: icmp_req=3 ttl=45 time=7.97 ms
64 bytes from 8.8.8.8: icmp_req=4 ttl=45 time=8.32 ms
64 bytes from 8.8.8.8: icmp_req=5 ttl=45 time=9.32 ms
64 bytes from 8.8.8.8: icmp_req=6 ttl=45 time=8.72 ms
64 bytes from 8.8.8.8: icmp_req=7 ttl=45 time=8.80 ms
64 bytes from 8.8.8.8: icmp_req=8 ttl=45 time=8.42 ms
64 bytes from 8.8.8.8: icmp_req=9 ttl=45 time=9.23 ms
64 bytes from 8.8.8.8: icmp_req=10 ttl=45 time=8.49 ms

--- 8.8.8.8 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9017ms
rtt min/avg/max/mdev = 7.973/8.693/9.320/0.402 ms
```

The delay is rather small. The performance is OK.

B..2. If you were runnning the same ping on other nodes, again the performance would be roughly the same. Can you conclude something ?

Solution: The delay from various places in the Internet towards 8.8.8.8 is mostly good. Google must have its servers geographically close to all nodes.

B..3. Several factors originating from the PlanetLab node itself, or from the network can affect your measurements. Can you cite a few ?

Solution: The load on the nodes, delays due to virtualization, congestion in the network links could all affect the measurements.

B..4. What can we do to get an rough estimation of the propagation delay ?

> **Solution:** We can take the minimum delay of a series of probe. In particular, if the network links are not overloaded, we can reasonably expect to get a sufficient estimation of the propagation delay.

B..5. Get an estimation of the propagation delay towards 8.8.8.8 from both nodes.

> **Solution:**
>
> ```
> ping -c 10 8.8.8.8
>
> EU -> 8.8.8.8
> rtt min/avg/max/mdev = 8.371/11.313/18.981/2.759 ms
>
> US -> 8.8.8.8
> rtt min/avg/max/mdev = 11.021/11.131/11.361/0.166 ms
> ```

The propagation delay are $d_{EU} = 8.371$ms, and $d_{US} = 11.021$ms.

> According to Google, as of 2013, Google Public DNS is the largest public DNS service in the world, handling more than 130 billion requests per day. *source: Wikipedia - Google Public DNS*

Maybe you already get an idea about the deployment of Google Public DNS servers ? Let's investigate a bit more…

B.I Localizing PlanetLab nodes

As a preliminary step, we want to get the localization of PlanetLab nodes. This data is not available in the web interface, but fortunately, PlanetLab provides a XMLRPC API that can be called to retrieve latitude and longitude information from the nodes.

The following Python script would allow us to retrieve latitude and longitude from the nodes, by calling two API functions: `GetNodes` and `GetSites`[4]. We will see later how to retrieve this information more conveniently.

```
vim node-lat-lon.py
```

```
#!/usr/bin/env python
#! -*- coding: utf-8 -*-

import xmlrpclib, pprint

API_URL    = "https://www.planet-lab.eu:443/PLCAPI/"
AUTH = {
    'AuthMethod': 'password',
    'Username'  : 'my-planetlab-username',
    'AuthString': 'my-planetlab-password',
}
```

[4] `https://www.planet-lab.eu/db/doc/PLCAPI.php`

```
srv = xmlrpclib.ServerProxy(API_URL, allow_none=True)

nodes = srv.GetNodes(AUTH, {}, ['hostname', 'site_id'])
sites = srv.GetSites(AUTH, {}, ['site_id', 'latitude', 'longitude'])

map_sites  = dict()
for site in sites:
    map_sites[site['site_id']] = site

for node in nodes:
    site = map_sites[node['site_id']]
    latitude = site.get('latitude')
    longitude = site.get('longitude')
    if not latitude: latitude = 0
    if not longitude: longitude = 0

    print "%s\t%f\t%f" % (node['hostname'], latitude, longitude)
```

```
./node-lat-lon.py > node-lat-lon.dat
grep ple2.ipv6.lip6.fr node-lat-lon.dat
grep planetlab2.cs.du.edu node-lat-lon.dat
```

```
ple2.ipv6.lip6.fr       48.852500       2.278490
planetlab2.cs.du.edu    39.679800       -104.963000
```

B.II Some speed of light considerations...

Let's compute the geodesic distance (the "as the crow flies" distance) between the two nodes. This is the minimal distance between the two point at the surface of the earth. Various models and approximation exist to compute this distance.

We advise you to use Python; you can use the python-geopy module [5], which should already be installed in your VM.

You can also use the language of your choice but we might not be able to offer answers or support. Search for packages allowing you to compute the geodesic distance between two nodes (eg. using the Vincenty formula, but others could be appropriate also).

Note that this script will be reused in the latest part of the lab, for the treatment of large scale datasets.

For your information, in R, we can use the `distVincentyEllipsoid` command from the `geospherepackage` [6] [7].

```
> library(geosphere)
> lip6 <- c(2.27849, 48.8525);
> ud <- c(-104.963, 39.6798)
> distVincentyEllipsoid(lip6, ud) / 1000
```

```
[1] 7881,050
```

[5] `http://geopy.readthedocs.org/en/latest/`

[6] `http://www.inside-r.org/packages/cran/geosphere/docs/distVincentyEllipsoid`

[7] You can load R packages using the following command: `library(PACKAGE);`

Solution:
Python version:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from geopy.point    import Point
from geopy.distance import distance

lip6 = Point(48.852500, 2.278490)
ud   = Point(39.679800, -104.963000)

print distance(lip6, ud)
```

```
7881.05029852 km
```

We thus have : $r = 7881.050$km.

We will assume those nodes are connected by optical fibre, whose index of refraction can be considered to be $r_f \approx 1.52$. The index of refraction of a material is calculated by dividing the speed of light in a vacuum $c_0 \approx 300000$km/s by the speed of light in the medium.

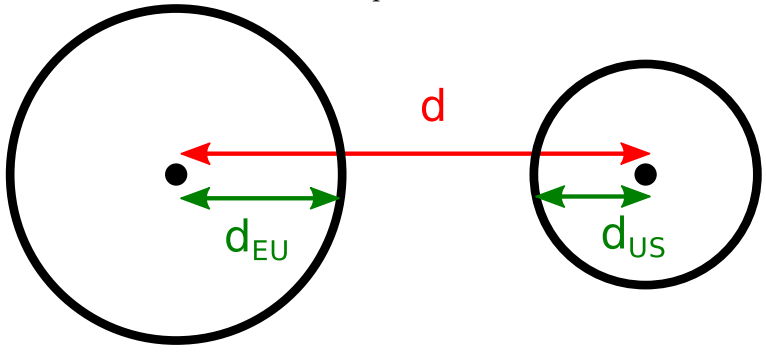
B.II.1. Compute the minimum bound on the amount of time the light would take to travel from one node to another.

B.II.2. Compare this result to the delay measurements you got previously ? What can you conclude ? Making a figure will help.

Solution:

$$t = \frac{d}{c_0/r_f} x1000 = \frac{7880km}{299792458km/s/1.52} x1000 \approx 39ms = d$$

At best, the light would take 39ms to travel between nodes. In our measurements, we got $d_{EU} + d_{US} = 8.371 + 11.021 < d = 39$ms, which proves that we have two distinct nodes.



B.III Understanding anycast

B.III.1. From your understanding on how routing works, can you explain how works this deployment, which is denoted as *BGP anycasting*, or simply *anycast* ?

Solution: Different nodes are assigned the same IP address (the service IP address), and the corresponding prefix is advertised in BGP from those different network locations.

When a BGP router receive several concurrent routes towards the same destination prefix, it elects the best one on which it will route the traffic. This is usually two different paths towards the same destination, in our present case, we have different paths towards different destinations.

This mechanisms is thus using the routing layer to choose the closer node (as seen by BGP).

B.III.2. How can administrators manage the different nodes remotely from a central location ?

Solution: Usually they assign two IP addresses to the same machine: an anycasted service adddress (eg. 8.8.8.8) and a management IP address, which is unique to each instance.

B.III.3. Beyond DNS, do you know or can you guess what anycast is used for in the Internet ?

Solution: 6-to-4 gateways, Multicast RDV point, CDNS, AS112, etc.

B.III.4. In your opinion, what are the pros and cons of using anycast for a service ?

Solution: Pros: resilience, load balancing, smaller delays, resistance against DDOS, unique contact address, etc. Cons: the best route for BGP is not necessarily the best for performance, hard to deploy, monitor, etc.

B.III.5. Do you see how the same functionality could be implemented (providing the same resources at different points on the network) without relying on the routing layer ?

Solution: Equivalent functionality could be done on top of DNS. This is was is done is some CDNs.

We have a method to distinguish between several anycast instances, based on delay measurements from known landmarks. This method is usually referred as the *geo-inconsistency* method.

Our next steps will be to evaluate the efficiency and accuracy of this method, and to measurement the deployment of the Google Public DNS service from PlanetLab.

To make our life easier, we will use the Manifold tool we have introduced before to run the measurements from PlanetLab towards 8.8.8.8.

C. Introduction to Manifold

This section is an introduction to Manifold in order to get familiar with the tool and discover its functionalities. This tool will make our life easier in order to perform the large number of distributed measurements that will be needed for understanding the DNS architecture.

We will use the TopHat service, running at UPMC, which is a deployment of Manifold where a set of data sources of interest have been preconfigured. It includes the services from geolocalization and IP-to-ASN

mapping we have used previously, as well as TDMI (TopHat Dedicated Measurement Platform) which will allow us to launch active measurements such as `ping` and `traceroute` from PlanetLab nodes.

Due to the way Internet works, it is natural that our results will depend on the number and location of our vantage points. Let's try to get some insight in this.

C.I Using Manifold shell

Manifold should have been installed by the script you ran at the beginning of this lab. It provides a shell that will allow us to issue queries to the service interactively.

```
manifold-shell -x -U https://clitos.ipv6.lip6.fr:7080 -L ERROR -z anonymous
```

The `-x` option inform the shell to contact the remote service by XML/RPC. The following URL is an alternative deployment of TopHat we have set up for this lab, since the service is not publicly open yet. We then ask the tool to display only error messages. Finally, we use anonymous authentication for the purpose of this lab.

C.II Discovering metadata

Platforms exposing data through Manifold use a representation similar to the one used in relational databases: tables. It is not surprising that the format for queries is then similar to SQL.

In a first step, we will issue a query towards a special table, `local:object`, which sits in a `local` namespace, to retrieve the set of available tables. This special queries allows us to retreive metadata, or the schema of the information available in Manifold.

Run the following queries to get familiar with the system:

```
>>> SELECT table FROM local:object
```

Let's look at the `ip` table more in details, which gives us some information related to IP addresses. The following query lists the fields available in the table `ip` and their description. These are the fields you can use in a query to get some informations.

```
>>> SELECT columns.name, columns.description FROM local:object WHERE table == 'ip'
```

Metadata describe the full range of available information, in this case related to an IP address, even if they do not belong to a single platform. In our setup, this information is drawn from interconnected platforms named `maxmind`, `tc` (Team Cymru) and `dns`. This can be verified by giving the name of the platform as the namespace for the query.

```
>>> SELECT columns.name, columns.description FROM maxmind:object
        WHERE table == 'ip'
>>> SELECT columns.name, columns.description FROM tc:object
        WHERE table == 'ip'
>>> SELECT columns.name, columns.description FROM dns:object
        WHERE table == 'ip'
```

C.III Query databases and webservices

Let's use Manifold to obtain the same date as in Section A..

C.III.1. First, contact the individual platforms (using the namespace) to request respectively country and AS information.

C.III.2. Remove the namespace and issue a single query to the integrated view.

C.IV Some statistics about PlanetLab

Information about PlanetLab nodes and sites that we have previously retrieved from the XMLRPC API via a Python script is also available transparently in Manifold, via the `node` table. Notice that there is no need to make two separate calls anymore (to the equivalent of `GetNodes()` and `GetSites()` functions), this is done automatically by Manifold.

The shell offers you the possibility to store the results of your query inside a variable and dump it into a CSV file.

```
>>> \$variable_name = SELECT \dots FROM \dots
>>> SHOW
>>> SHOW \$variable_name
>>> DUMP \$variable_name INTO "/path/to/file.csv"
```

C.IV.1. Run a Manifold query to get a list of nodes, their country and their ASN.

Solution:

```
>>> $nodes = SELECT hostname, country_name, asn FROM node
>>> DUMP $x INTO /home/tma.nodes.csv
```

C.V Analysis of PlanetLab diversity

Run the R software, it will allow you to do some simple statistics about the data.

```
R
```

You can run the follwing commands to load the file you just created and make some statistics.

```
nodes <- read.csv("/home/tma/nodes.csv")
summary(nodes)
summary(nodes$country_name)
summary(nodes$asn)
\dots
```

C.V.1. What can you conclude about PlanetLab diversity ?

Solution: PlanetLab nodes are deployed in many AS and countries and we can hope to exploit this geographical and topological diversity for measurements.

C.VI Adding local data sources

Depending on the remaining time, you might want to skip this section, and proceed directly to the last part : "Measuring the DNS architecture". In this case, just read through this section to get an idea of the functionalities of Manifold.

One remark that is often done for measurement studies based on PlanetLab is the bias towards academic networks, since a huge proportion of the nodes is hosted by universities, and far less on commercial networks. You might already have noticed this previously.

One way to verify this would be to cross the data about PlanetLab nodes with a characterization of the ASes they belong to. An old but interesting dataset is provided by GeorgiaTech [**?**].

We provide a copy of this dataset (in CSV format) in the archive you downloaded at the beginning of this lab. This dataset is available online[8].

In order to join this local data with the information offered by TopHat, we need to configure and run a local Manifold router. We first add the TopHat platform we used to query directly via the shell. You can copy/paste these commands.

```
manifold-add-platform tophat TopHat manifold none
        '{"url": "https://clitos.ipv6.lip6.fr:7080/"}'
```

As for the local CSV file, Manifold already provides an adapter allowing to query this data source. It can be configured simply by configuring a new platform with the corresponding file path. Usually, column names are guessed from the CSV file, but they can be given to Manifold via the platform configuration (which is needed in our case). You can copy/paste these commands.

```
manifold-add-platform georgiatech
        "Georgia Tech Autonomous System Taxonomy Repository"
        csv none '{"as":{"filename": "/tmp/as2attr.txt", "fields":
        [["asn", "int"], ["as_description", "string"],
        ["num_providers", "int"], ["num_peers", "int"],
        ["num_customers", "int"], ["num_prefixes_24", "int"],
        ["num_prefixes", "int"], ["asn_class", "string"]],
        "key": "asn"}}' 0
manifold-enable-platform georgiatech
```

Finally, we then run the shell with an integrated router, with the set of platforms we have configured.

```
manifold-shell -z router
```

C.VI.1. Looking at metadata, verify that the `ip` table now includes those additional fields (mainly `asn_class`.

Solution:

C.VII Verify PlanetLab academic bias

C.VII.1. Request for each planetlab node its hostname, country, AS number and AS class, and store this inside a csv file (eg. `/home/tma/nodes.csv`).

Solution:

C.VII.2. Like previously done, use R to see the diversity in terms of AS types. What can you conclude ?

C.VII.3. What consequence this observation might have on the discovery of anycast instances ?

Solution:

[8] `http://www.ece.gatech.edu/research/labs/MANIACS/as_taxonomy/data/as2attr.tgz`

D. MEASURING THE DNS ARCHITECTURE

D.I In search of ground truth

We have devised a methodology to measure the deployment of anycast instances (we will see later how to extend the reasoning to more than 2 nodes). Though, we do not know neither its efficiency, nor its accuracy.

A significant issue often encountered in measurements is that usually we have no ground truth to which to compare our results. Most of the time we are searching for evidences like we have done here, and it is hard to know about the issues of the model, the measurement artefacts, or the bias introduced by our measure. A significant example is the topology of the Internet, which remains largely unknown. At best we can estimate some of its properties.

The measurement of Google public DNS deployment is no exception to the rule.

Fortunately for us, some root DNS server deployments use anycast, and their deployment is public[9]. (although it might sometimes be incomplete or not up to date). In addition, they often embed debug functionalities that can help us identify which anycast instance we are talking to, even though they all answer with the same address. This is described in RFC4892[10] : "Requirements for a Mechanism Identifying a Name Server Instance":

cmttFor some time, the commonly deployed Berkeley Internet Name Domain (BIND) implementation of the DNS protocol suite from the Internet Systems Consortium [BIND] has supported a way of identifying a particular server via the use of a standards-compliant, if somewhat unusual, DNS query. Specifically, a query to a recent BIND server for a TXT resource record in class 3 (CHAOS) for the domain name "HOSTNAME.BIND." will return a string that can be configured by the name server administrator to provide a unique identifier for the responding server. (The value defaults to the result of a gethostname() call). This mechanism, which is an extension of the BIND convention of using CHAOS class TXT RR queries to sub-domains of the "BIND." domain for version information, has been copied by several name server vendors.

A refinement to the BIND-based mechanism, which dropped the implementation-specific label, replaces "BIND." with "SERVER.". Thus the query label to learn the unique name of a server may appear as "ID.SERVER.".

For example, both the F and K root servers use anycast and answer those CHAOS queries. We will use those deployments to benchmark our methodology, before applying it to 8.8.8.8.

For example, the `dig` command, identify the instance of the DNS server `f.root-servers.org` and `f.root-servers.org` we are talking to:

D.I.1. Using the second PlanetLab node (in the US), run the previous `dig` command to identify the instance you are talking to. Can you guess something from the naming ?

D.I.2. Repeat the measurement a couple of times. If you notice differences, what do you notice ? How do you explain these differences ?

D.II Obtaining data and ground truth

Obtaining a ground truth requires us to run simultaneous measurements of delay (using `ping`), and identification of the anycast instance (using `dig`) from a large number of nodes.

The integrated dataset of measurements targetting the F-ROOT and K-ROOT root DNS servers, as well as the Google Public DNS has been prepared in advance:

You can download the datasets at this address:

`http://www.top-hat.info/download/tma-2014/datasets/`

The file is composed of the following columns, separated by a space character:

> hostname: the hostname of the PlanetLab node;

> latitude: its latitude;

[9] http://root-servers.org/

[10] `http://www.ietf.org/rfc/rfc4892.txt`

> longitude: its longitude;

> delay: the propagation delay towards the anycast instance (as measured by `ping`);

> instance (or *unknown* when it is not available): the name of the anycast instance (as measured by `dig`).

D.III Extending the method to multiple nodes

Previously, we have introduced the *geo-inconsistency* method to distinguish between two anycast instances. We will now extend this method to multiple instances.

Suppose we are looking at the measurement from node i, at position p_i. The delay to the anycast instance is d_i. Denote A_i the area that can be reach by the light during the time d_i.

D.III.1. Using set notation, with nodes 1 and 2, what is the necessary condition on A_1 and A_2 so that we can distinguish the two anycast instances with the *geo-inconsistency* method ?

D.III.2. What is the necessary condition for 3 nodes ? for N nodes ?

D.III.3. In practice, this condition will not be verified by all our measurements. What should be done in order to be in a position to apply the method ?

D.III.4. How can we discover as many instances as possible ? Clearly formulate the problem.

D.IV Implementing the method

D.IV.5. Make a program that will try to determine as much instances as possible, based on the previous datasets. You can make an approximated algorithm here.

Solution:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys, time
from geopy.distance import distance
from geopy.point import Point

infile = sys.argv[1]
outfile = sys.argv[2]

# Index of refraction for optical fiber
FIBER_RI = 1.52
SPEED_OF_LIGHT = 299792.458 # km/s

class Disc(object):
    def __init__(self, hostname, instance, ping, latitude, longitude):
        """
        ping (float): (in ms)
        """
        self._point     = Point(latitude, longitude)
        self._radius    = ping * SPEED_OF_LIGHT / FIBER_RI / 10000 # in km
        self._hostname  = hostname
```

```
        self._instance  = instance

    def get_point(self):
        return self._point

    def get_radius(self):
        return self._radius

    def overlap(self, other):
        """
        Two discs overlap if the distance between their centers is lower than
        the sum of their radius.
        """
        return distance(self.get_point(), other.get_point()) < (self.get_radius()

    def __str__(self):
        lat, lon, alt = self._point
        return "%s %s %s" % (lat, lon, self._radius)

class Discs(set):
    def overlap(self, other):
        for disc in self:
            if disc.overlap(other):
                return True
        return False

discs = Discs()

f = open(infile)
for line in f.readlines():
    hostname, instance, ping, latitude, longitude = line.strip().split()
    disc = Disc(hostname, instance, float(ping), float(latitude), float(longitude)
    if not discs.overlap(disc):
        discs.add(disc)
f.close()

print "Number of anycast instances found:", len(discs)
print ""

f = open(outfile, 'w')
print >>f, "latitude longitude radius"
for disc in discs:
    print >>f, disc
f.close()
```

D.IV.6. Run your program against `froot.dat` and `kroot.dat`.

Solution:

```
./find-instances.py froot.dat froot-instances.csv

Number of anycast instances found: 28

./find-instances.py kroot.dat kroot-instances.csv

Number of anycast instances found: 24
```

D.IV.7. Looking at the root server website[11], what do you think of the completeness of the method ?

Solution: F-root has 55 sites. K-root has 17 sites.

D.IV.8. What would be necessary for a better coverage ?

Solution: More vantage points would be necessary, especially situated in commercial networks or those having local anycast instances. Also, having nodes close to the instances will make the measured delay lower, and thus will improve the accuracy of our method.

D.IV.9. What about the method accuracy ? does it present inconsistencies (two different instances that are in fact the same) ?

Solution:

```
> instances <- read.csv('froot-instances.csv', sep=" ")
> summary(instances$instance)

ams1a.f.root-servers.org ams1b.f.root-servers.org atl1a.f.root-servers.org
                       1                        2                        1
atl1b.f.root-servers.org bcn1b.f.root-servers.org cdg1a.f.root-servers.org
                       1                        1                        4
cdg1b.f.root-servers.org fra1b.f.root-servers.org gru1b.f.root-servers.org
                       1                        2                        1
hkg1b.f.root-servers.org kix1a.f.root-servers.org kix1b.f.root-servers.org
                       1                        1                        1
lax1b.f.root-servers.org ord1b.f.root-servers.org pao1b.f.root-servers.org
                       1                        3                        1
prg1a.f.root-servers.org rom1a.f.root-servers.org sin1a.f.root-servers.org
                       1                        2                        1
yow1a.f.root-servers.org yow1b.f.root-servers.org
                       1                        1
```

[11] http://root-servers.org/

```
> instances <- read.csv('kroot-instances.csv', sep=" ")
> summary(instances$instance)

 k1.apnic.k.ripe.net  k1.ficix.k.ripe.net  k1.grnet.k.ripe.net
                   1                    2                    2
   k1.nap.k.ripe.net  k1.tokyo.k.ripe.net k2.ams-ix.k.ripe.net
                   1                    1                    4
   k2.bix.k.ripe.net k2.poznan.k.ripe.net k3.ams-ix.k.ripe.net
                   1                    3                    3
 k3.denic.k.ripe.net    k3.nap.k.ripe.net  k3.tokyo.k.ripe.net
                   3                    1                    2
```

D.IV.10. What might be the reasons for these inaccuracies ?

Solution:

> model for distance

> speed-of-light calculations

> wrong coordinates for nodes

> bugs ?

D.IV.11. Run your program against `google.dat`. What are your results ?

Solution:

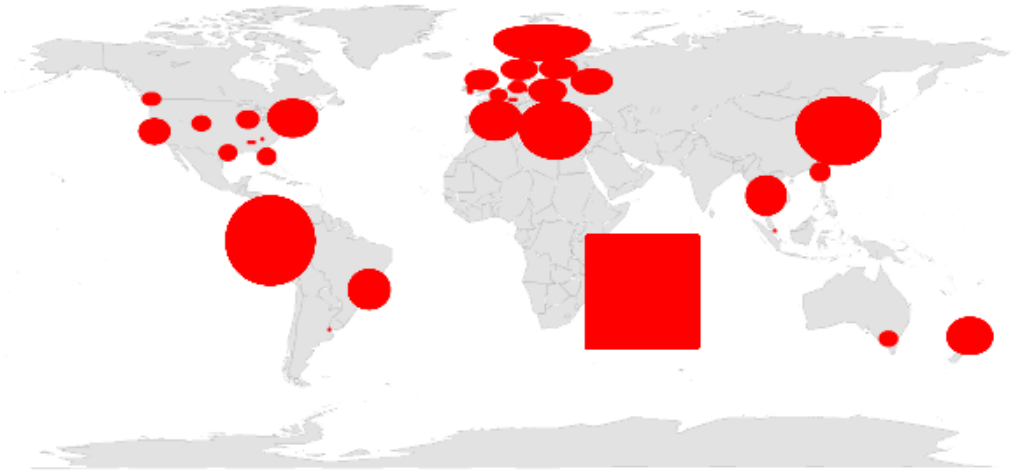
```
./find-instances.py google.dat google-instances.csv

Number of anycast instances found: 24
```

D.IV.12. If you still have time, plot your results on a map.

Solution:

```
#!/usr/bin/Rscript

library(maps)

dat <- read.csv('google-instances.csv', sep=" ")

deg2rad = function(deg) return(deg*pi/180)
DISTANCEDEG = function(long1, lat1, long2, lat2) {
R=6371; d=acos(sin(lat1)*sin(lat2) + cos(lat1)*cos(lat2) * cos(long2-long1)) * R
return(d)
```

```
}

plot_disc <- function(disc) {
    X=disc["longitude"]
    Y=disc["latitude"]
    L=disc["radius"]
    VX=seq(X-20,X+20,length=L/2+1)
    VY=seq(Y-20,Y+20,length=L/2+1)
    VtX=rep(VX,each=length(VY))
    VtY=rep(VY,length(VX))
    ZDeg=deg2rad(cbind(VtX,VtY))
    D1=DISTANCEDEG(ZDeg[,1],ZDeg[,2],deg2rad(X),deg2rad(Y))<L
    points(VtX[D1],VtY[D1],pch=19,cex=.2,col="red")
}

png("google-instances.png")
map("world", col = "#e2e2e2", fill = TRUE, bg = "white", lwd = 0.05)
apply(dat, 1, plot_disc)
dev.off()
```



D.IV.13. Given your understanding of Internet routing, and the results you have observed for paths and delays, what might be a good strategy for deploying anycasted instances ?

Solution: Deploying instances in places where a lot of AS are present, such as IXPs.

D.IV.14. Do you have ideas on how to further analyze anycasted services ?

Solution:

> Triangulation

> Know position in POPs