BGP AS / ISP Security Ranking

Raphaël Vinot

January 30, 2011

Abstract

For an Internet Service Provider, Autonomous system numbers (ASNs) are a logical representation of the other ISP peering or communicating with his autonomous system. ISP customers are using the capacity of the Internet Service Provider to reach Internet services over other AS. Some of those communications can be malicious (e.g. due to malware activities on an end-user equipments) and hosted at specific AS location. In order to provide an improved security view on those AS numbers, a trust ranking scheme will be implemented based on existing datasets of compromised systems, IPs of malware Command and Control servers and existing datasets of the ISPs.

BGP ranking is a free software to calculate the security ranking of a specific AS number. The system is gathering external datasets (e.g. Dshield, Shadowserver, Arbor ATLAS) to calculate the ranking over the time. The purpose is to show the malicious activities of a specific AS and maybe limits its impact on other ASes.

Contents

List of Figures

Part I

Context, motivation and non-technical view

1 Context

1.1 Internet and the peering

Internet is a decentralized network with a bunch of operators, each operator has his proper network and they exchange traffic together. This exchange is called *peering* [1]. Without this exchanges, Internet can not work.

There is three different types of peering [2]:

- Transit: you pay a provider to provide you a transit across its network
- Peer: you exchange traffic with an other provider freely for *mutual benefit*
- Customer: you sell an access across your network to your customers

The first of all *mutual benefits* between two peers will be the reduction of the costs of the exchanges. But it will also provide redundancy, improve the capacity of each network by increasing the number of possible routes and globally improve the performances.

Two ISPs of the same size exchange traffic freely but when they need to exchange traffic with bigger providers (upstream providers), they will have to pay. When a smaller provider (downstream provider) want to peer with an ISP, he will pay too.

1.1.1 Border Gateway Protocol (BGP)

The Border Gateway Protocol is the protocol [3]used by everyone on the Internet, without knowing it exists. This protocol is commonly presented as the routing protocol of the Internet, all the core routers of the Internet are using it to communicate together.

It is a path vector protocol: the routing table contains the destination network, the IP address of the next router on the way to this network and the path (in this case, the list of ASNs) to reach the destination.

In the practice, every router maintains a routing table to associate a network with the path of Autonomous system (AS) to this network, there is an example of entry on page 44.

There is always more than one possible route to reach a particular network, the decision is made using the path but also the network policies.

On a security point of view, we can for example blackhole the traffic to a certain destination known as providing a Command & Control server and this way block the communication between this server and the clients, or to protect a subnet under DDoS attack coming from an other AS/subnet.[4]

1.1.2 Autonomous system (AS)

The operators on the Internet exchange their routes, it is obvious that this operators need a way to identify each other without using IP Addresses. That is why there are Autonomous systems numbers

(ASNs) [5], an operator which wants to announce IP Addresses on the Internet needs at least one ASN, an AS will announce one or more networks.

The ASNs and subnets are assigned by the IANA to the Regional Internet Registries (RIRs) which will sell it to its clients. The RIR responsible for the Europa region is RIPE-NCC.

1.2 Resources

1.2.1 Routing Information Service (RIS)

This service is described like this on the website[6]:

> RIS is a RIPE NCC project that collects and stores Internet routing data from several locations around the globe. RIS offers tools that bring this data to the Internet community.

The service provide a dump of the routing database directly extracted from their routers. This dump is exported each eight hours and contains around 350.000 different routes. An other very important point to keep in good memory is that the routes are volatile, each new version of the routing database has between 800 and 1000 changes. A change is a new route: an ASN announce a new network or a dropped route: the network is not announced by this ASN anymore. Thanks to this dump, it is possible to compute the number of IP addresses assigned to each ASN.

If you want to see graphs and visualize the evolution of the routes by AS and by networks, you should visit the Routing Information Service website [6].

This service allow also to fetch in real time the ASN assignation of an IP address and usually to get a small textual description of this AS. An example on page 44.

1.2.2 Whois

The well-known whois protocol allows us to fetch information on the owners of IP addresses. Even if the malicious ISP controls some objects of the whois entries (e.g. route-object updated by a bot master), they can be considered as datasets because it will give us information on the bot master.

A whois entry contains much more information than a riswois entry but is not so often updated, about once a week. The most interesting information you can find in the whois entry is the name, the address and the phone number of the owner of the block of IPs.

We should speak of whois databases and not whois database because there are many servers providing different databases: each RIR maintain a whois database with the information he wants.

To get the most precise information on a particular IP Address, it is important to query the right server. The implementations of whois clients maintain an list of assignations to know for each IP what is the server to query. The implementation of Debian contains also the whois servers of some Local Internet Registries (LIRs), a subdivision of the RIRs.

Almost all the databases are incompatible with each other, the keys and the information are different.

1.2.3 Datasets of suspicious activities

It is possible to find some datasets freely available on the Internet but a lot of them are not and that for different reasons:

- protection of the sources: each company has his proper way to gather information. If the attacker know this methods, he will attempt to be more stealthy and it will be more complicated for the company to find him again
- the company providing the dataset sells this information to the ISPs which have interest in it

It exists a large variety of datasets including IP and/or prefixes for malicious activities. But their level of trust is variable, the methods used to generate the datasets can vary a lot from a source to another. To be exhaustive on this description of the datasets we have to say that many data sources gives only information on a particular ASN.

Arbor ATLAS/Active Threat Feed The dataset provided by Arbor is not freely available and their quality is quite high because the reports are analyzed and classified. Arbor explains his work in an FAQ [8] and the gathering of the data is explained like this:

> Data is captured by using a distributed network of sensors running a number of data capture and analysis tools. These sensors can:
>
> - Interact with attackers to discover what activity they are attempting
>
> - Capture full payloads and classify them
>
> - Characterize scan traffic to know the reputation of the traffic he is routing and mitigate the malicious activities originating from their customers.

And this information are merged with logs, statistics and reports to have a complete view of the threads and reduce the false positives.

Atlas provide also some more information such as an URL to get more information, the coverage time and category of the attack.

Dshield The lists of Dshield are public and they are generated using only the firewall logs of the users [7], the precision and the quality is less good than for Arbor but there is however a small analyze because Dshield provide two lists.

The first one is a full dump of the today's information, no filtered at all, contains around one million entries and many false positives like private address. But this list stays interesting because of the big amount of IPs it contains and it is useful to correlate it with other sources.

The second contains only the "Top 100" of the daily dump and less false positive but should not be used as blocklist says Dshield.

abuse.ch ZeuS Tracker This list is only concentrated on one unique thread: ZeuS. On the FAQ, ZeuS is described like this[9]:

> ZeuS (also known as Zbot / WSNPoem) is a crimeware kit, which steals credentials from various online services like social networks, online banking accounts, ftp accounts, email accounts and other (phishing). The web admin panel can be bought for 700$ (source: RSA Security 4/21/2008) and the exe builder for 4'000$ (source: Prevx 3/15/2009).

The detection of ZeuS Command&Control servers is done by traffic sniffing and study of the captured (in Honeypots) Zeus clients. This list contains all known ZeuS Command&Control servers and is expected to be used as a blocklist, it is highly reliable.

Shadowserver The mission of the Shadowserver Foundation is presented like this on the website[10]:

> The Shadowserver Foundation is an all volunteer watchdog group of security professionals that gather, track, and report on malware, botnet activity, and electronic fraud. It is the mission of the Shadowserver Foundation to improve the security of the Internet by raising awareness of the presence of compromised servers, malicious attackers, and the spread of malware.

They are working the same way as Arbor by providing reports on particular AS, only to the members but they does not have to pay for it.

Abusix It is completely different: Abusix does not provide list but permit to find easily the abuse address of a provider by the IP address of the attacker, will send a mail to this address and tell what the problem is and on what IP address. The data coming from Abusix are usually good but should be verified because it is always possible to report a false positive.

This raw-datasets will be aggregated and analyzed to see their evolution in the time and generate statistics.

1.3 Threats

Internet has always been an interesting place for the cybercriminals: virus and attack on Information Systems exists since the computers exist but now that almost everybody can have an Internet access (with a computer or a mobile device) it is a way more easy for the attackers to gain access to more computer: for a vast majority of the population IT security is absolutely not in their scope and they have no clue on it.

In the past, the malwares were often used to beak a computer or make them crash. Not anymore: already in 2007, Kaspersky proclaim the *death of "non-profit" malicious software*[11]. Now this activity is a real business with organizations doing malwares, administrating Botnets for their own needs or renting a part of the botnet to clients [12].

One of the most used way to commit illegal activities on the Internet these days are the Botnets.

1.3.1 Botnet

A botnet is a network of devices (Bots) infected by a malware which are controlled by a Command & Control Server and administrated by a Botmaster. The Botmaster is often an organization which rent a certain amount of infected computers to performs suspicious activities.

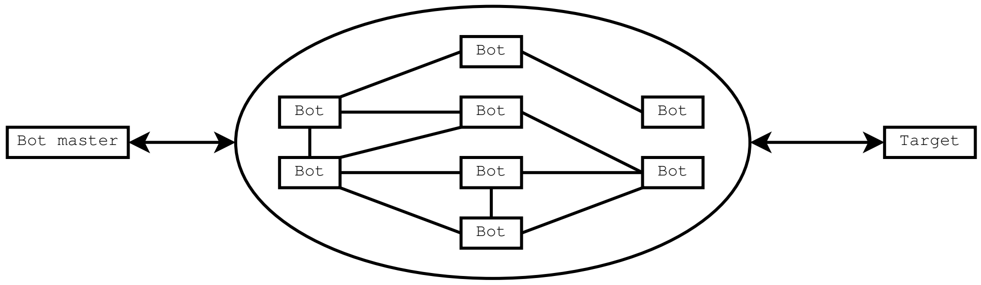

Figure 1: Botnet - Basics

A botnet may be used to perform DDOS attack on servers, exchange data, provide phishing sites, send Spam, distribute Malwares and many other.

The big advantage for the cybercriminals is that they are (virtually) invisible because they never attack directly their targets and are never seen as attacker in the logfiles. An other advantage is that it costs almost nothing to them: they does not need to earn thousands of computers, they just need to develop a malware and let it working.

1.3.2 Usage of the network

The Bot-master will communicate with his Bots to give orders, the bots will sometime communicate together (P2P based botnet[13]) and they will attack their targets in case of DDoS or attempt to make the victims coming on the servers in case of phishing.

As we can see, the communication is the key of a Botnet. We can easily understand that if we are able to stop the communication between the master and the bots, or know the members of the botnet and do not browse on websites hosted on them, the botnet will be armless. This techniques are called black-holing: all the data coming or to the designated hosts will be silently dropped.

1.4 Monitoring the network

As we can see, there is a gap between the world where everybody will peer together without any control, just to make Internet faster and the real world where there is Botnets and companies selling those to perform DDoS, send Spams and provide phishing websites. We need a way to be able to know what is happening in his network and respond to the attacks.

1.4.1 Confidence

An other point to keep in good memory is that the Internet is based on confidence... but confidence without any control will not work: if an ISP decide that he does not care what is happening on his network he may be blacklisted by the other ISPs. Or at least his bandwidth will be reduced, even if the the suspicious traffic is originating from a third ISP.

This is why it is a necessity for each Internet operator to know what for traffic he is routing. This task is critical for every ISP and of course for the satellite operators too: they often provide an Internet access to their customers.

1.4.2 Reputation and ranking system

To know the reputation of the traffic he is routing and mitigate the malicious activities originating from their customers, an ISP can for example use a ranking system based on datasets of suspicious activities.

There are at least three goals which can be achieved by using a ranking system:

- Disrupting the malware Command & Control communication:
  - drop all the traffic from and/or to an AS or a network (blackholing)[14]
  - modify the performance and/or the bandwidth of the traffic coming from and to the C&C server (traffic shaping)
- Limiting the *effectiveness* of the Botnet, reduce the propagation of the malware by informing the other ISPs or the customers that a particular AS is a bad peer or that the websites hosted there should not be trusted
- Assessment between ISPs: the ranking can be influenced by other ISPs and it is possible to improve or limit the peering with an other ISP

To monitor properly the network, we need to aggregate information coming from different sources and analyze it.

1.5 Key points in the analysis

The data are coming from different sources, it is important because this sources sources will cover different threads ant it will allow us to have a more global overview on the malicious ASNs.

To classify properly the information, the least common denominator, present in each datasets: CIDR block and / or ASN.

1.5.1 Size and quality

As we saw, we have some poor-quality sources but with a lot of entries they are very useful to see trends. The other, with an higher quality but less entries will be used to emphasize the trends of the first sources: they have already been analyzed and contains much less false positive. It is the first and most important point of the analysis: we have to be able to make the difference between the low-quality entries and the high-quality one.

Solution This differentiation will be done by using a weight on each entries: the best sources will have an high weight and the other a lower one. Like this, they will not be lost in the big amount of entries coming from the other, low-quality datasets.

1.5.2 Format

Every dataset has his proper format, they are obviously incompatible from one data sources to another: we will find plain text, CSV and XML files. Some files contains only the IP addresses, some other the type of the attack and a timestamp. Some files are updated every couple hours, most of them every 24 hours. But the insertion in the database has to be same for every source.

Solution Every source will have his proper module, extracting the needed information of the files and return this information to a standard interface which will insert it in the database. This way, the overhead induced by the new datasets will be highly reduced. More in formations on this particular point on page 29.

1.5.3 Type of the information

Last but not least, some of the datasets contains only information on a particular AS. We already spoke of this datasets: they are usually provided by companies selling this in formations to ISPs. This datasets may badly influence the results by increasing the weight of a particular AS.

Solution It is actually not fully implemented but the goal will be to offer to each ISP interested by this system to obtain an account on the website and manage this datasets on the way he wants: compare the datasets with the global data or have a view on the evolution of his ASN.

After all this theoretical part, we will now give you a small overview of the system without too much details of the implementation. But it will allow you to have a better view of the basic principles, the architecture of the system and on the requirements imposed by the future usages.

2 Implementation

After this exhaustive presentation of the context, it is now time to give some information on the implementation and what are the requirements of the application.

The high-level diagram of the implementation is available on page 15.

2.1 Highlights

There are two requirement on this project: the code has to be opensource and as fast as possible to allow to use the results on a corporate network.

2.1.1 License

The whole project is opensource because its purpose is to allow every company interested by this project to use it. But we also want that if this companies use the code and make modifications, they participate to the project.

That's why we choose a GPL license and more precisely the version 3 of the Affero GPL. The advantage of this license (Affero) is that if someone want to use the system and only allow the users to use it over the network, he will have to release the code [17]. It is not the case for the standard version of the GPL but for this project, it is very important because the system will most of the time / always be accessed through the network.

Note the code of the system is released opensource [20], not the datasets and the results, for the reasons exposed above.

2.1.2 Usages and consequences

The code will be used to filter the network flow:

- *divert filtering*: the packets will be stopped, send to the userspace, analyzed, modified, whatever and may be pushed back into the IP stack [18]
- *blackhole filtering*: the packet are silently dropped

To achieve this goal it is possible to use flow spec[21]: it is a Quality of Service but for the BGP protocol. This RFC allows for example to increase or decrease the priority of the traffic originating from a particular Subnet or AS.

Flow spec can also be used to do ingress filtering on the border of the ISP infrastructure. If the source IP address of a flux is not in the range announced by the originating AS, the traffic will be discarded, even if this traffic comes from *behind an Internet Service Provider's (ISP) aggregation point* [19].

The first consequence is that the process of computing the ranking of an AS (and later of a subnet) has to be fast enough to allow the user to manipulate the traffic originating from a malicious source when it is detected in the datasets. The definition of *fast enough* depends on the the frequency of the update of the sources.

The solution we decide to use is to do a *mapreduce*-like processing[15]: it means to be able to run as often as possible more than one process on the same task. To do it, we will share the information between the processes using a key/value storage server called Redis [16].

2.2 High-level view

The ranking system can be reduced as two different parts: the aggregations of the data and the ranking himself.

2.2.1 Aggregation

The first data inserted into the database will be the one from the datasets, typically the IP Addresses. At the same time, two other processes will find in the database all the new entries by looking for the one without RIS and Whois entries and fetch the most up-to-date available information from the Internet.

The most important information to fetch is the RIS Whois entry because it contains the ASN announcing the IP and the block of the IP. Both of this information are very important because even if the ranking is actually only based on the ASNs, it is planned to implement a ranking based on the subnets.

Note it is possible to deactivate the fetching of the whois entries in the configuration: it is only useful to have complementary information on a particular IP address but this fetching takes quite a lot of time (see on page 32 for more information) and use an huge amount of space in the database.

It is deactivated by default.

2.2.2 Ranking

Based on the data in the database, we will be able to compute the ranking *for one ASN* with the following formula:

$$R = 1 + \frac{(SUM(Occurrences*source\_impact)*SUM(Vote))}{AS\_Size}$$

Occurrences all the IPs from the ASN, by sources. Each Occurrence per source is unique even if we see multiple the same IP address for the same source.

Source_impact the value assigned to the source, it will depend of his quality

Vote the vote against this AS given by an other user of the system (actually not implemented)

AS_Size The number of IPs announced by the AS when the ranking is computed

We can see the same IP address among the different sources, the formula is not limiting that as this is usually a good sign of malicious activities: the information is validated by multiple sources.

Note Only The ISPs allowed to access to the system will be able to vote against other ISPs.

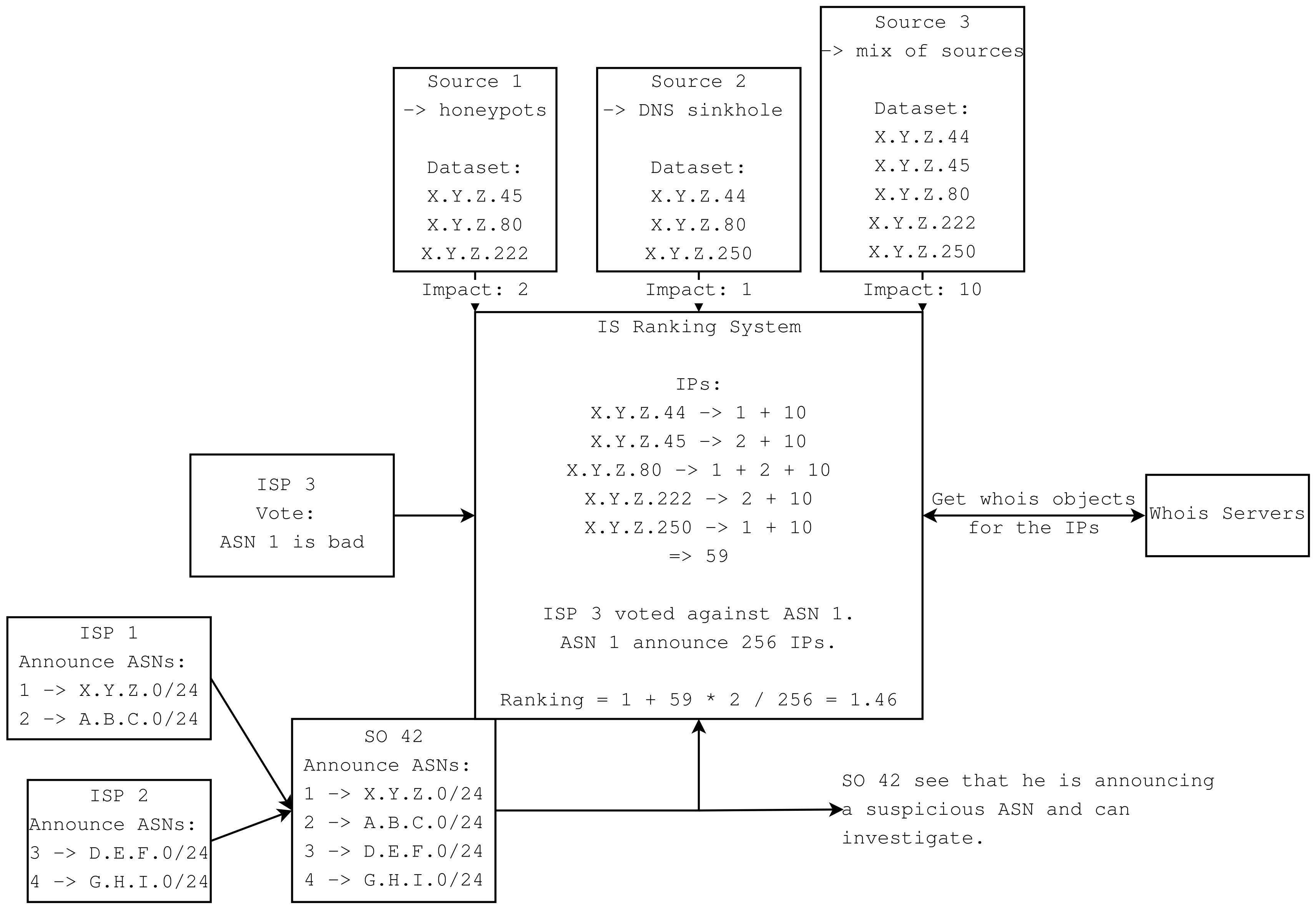

Figure 2: High level

Part II

BGP AS / ISP Security Ranking

3 Data storage

We have two types of data to store in this project:

- static information: they are written one time in the database and never updated/deleted anymore, only read. As the performances are not really important, we use a relational database.
- temporary, volatile information: they are written/updated/deleted very often and need a fast access. A key/value storing solution is a good choice.

3.1 Relational database

The system officially supports MySQL as relational database but because we are using the python module *SQLAlchemy*, all the databases supported by it may be used [22]. Be aware that depending on the database you will use, you might experience some (performance) problems, mostly if you decide to use SQLite. But PostgreSQL for example should work fine.

This database contains actually all the static information used by the ranking system. Static information means all the information extracted of the datasets and the whois entries, when they are in the database, they will never be modified anymore. The voting in formations are also saved into this database as well as the history of the ranking used to generate the graphs.

On top of *SQLAlchemy*, the module *elixir* is used because it implements the active record pattern[23]: it allows the developer to see the entries of the database as objects and simplify a lot the usage of the databases. The most important point is that we will never use any SQL Queries. On a security point of view, it will reduce the risk of SQL injections.

Note the users will never write anything in the database and have a read only access through the web interface: all the write process is done by the scripts we will explain in the next part of this document.

3.2 Key/value store

The Key/value store used by the system is Redis. Depending on the part of the program, Redis is used differently:

3.2.1 Shared memory

The whole program use a lot of processes and most of them need to exchange information. It would be possible to use shared memory between processes using the system API but it is a lot easier to put all the information in a shared database which will deal on its side with the problem of the simultaneous access to the resources.

The process which need this information will pick them, do his work and give the result to the next process through the same or an other database.

3.2.2 Caching

It is common that an IP address is present more than one time in a dataset, or the same IP in different datasets. It is important to know that the whois entries have a life time of at least 24 hours: they are not updated that often on the whois servers. This is why a caching system is very useful to improve the performance of the software: it is absolutely unnecessary to fetch more than one time a particular whois entry for at least 24 hours.

Redis implements a time to live command, we just have to give a TTL of 24 Hours to each cached entry and it will be automatically deleted after that amount of time.

3.2.3 Fast access

The other usage of Redis is quite obvious: we need to be able to do a lot of queries in a small period of time. It is the case for the the whois database for example: the most we do queries per second, the fastest we update the ranking database.

Redis provide a command called *redis-benchmark*, it gives the following results with the code present on the master branch of today (27.08.2010) on our test server:

```
$ ./redis-benchmark -q
PING: 45477.27 requests per second
PING (multi bulk): 50545.45 requests per second
SET: 53475.94 requests per second
GET: 52408.38 requests per second
INCR: 51287.18 requests per second
LPUSH: 53545.45 requests per second
LPOP: 52392.67 requests per second
SADD: 52652.63 requests per second
SPOP: 51020.41 requests per second
LPUSH (again, in order to bench LRANGE): 53223.40 requests per second
LRANGE (first 100 elements): 4835.59 requests per second
LRANGE (first 300 elements): 1786.67 requests per second
LRANGE (first 450 elements): 1249.69 requests per second
LRANGE (first 600 elements): 932.05 requests per second
```

The test server is a Intel(R) Xeon(R) (Quad Core) with 2.66GHz per core, 8Gb RAM and running Ubuntu 10.4.

Redis define several types of values [24] the following types are used in the software:

String it is the simplest type: we just put into the database a key and a value.

```
$ ./redis-cli set mykey "my binary safe value"
OK
$ ./redis-cli get mykey
my binary safe value
```

Set it is a list of elements but each element is unique

```
$ ./redis-cli sadd myset 1
(integer) 1
$ ./redis-cli sadd myset 2
(integer) 1
$ ./redis-cli sadd myset 3
(integer) 1
$ ./redis-cli smembers myset
1. 3
2. 1
3. 2
```

4 Implementation, Technical overview

You can find directly the global diagram representing the part of the system gathering the data on page 47. And on page 48 for the ranking.

4.1 MySQL databases

Two databases are in use: the first one contains only the data of the datasets. The second one, the results of the ranking and the voting information.

4.1.1 Data from the datasets

The database contains the the following tables:

IPs contains only the IPs extracted of the datasets. For each entry, the IP is the primary key.

This table allows us to dump easily all the IPs we found in the datasets.

IPsDescriptions for each IP found in the datasets, we create a new entry in this table.

The following information are extracted of the dataset, if they exists:

- The name of the class where the IP is coming from. This name has to be different for each class: it will be used for the ranking
- The timestamp of the insertion into the database: it is most of the time the current day but it can also be used to re-import old datasets, this functionality is actually not implemented. See on page 34 for more information
- The date of generation of the list: it is the date given by the file or set to the current day if we don't have this information in the dataset
- The number of times the same IP is found in the same dataset

  Note if there is in the dataset a timestamp for each entry, we will create a new description each time, even if the same IP is present more than once.
- The type of the infection

- Other raw information given by the dataset

The following information are set in the next part of the processing:

- The whois entry
- The source of the whois entry
- Information on the ASN announcing the IP

ASNs is the same as the IPs table: it contains only the ASNs and for each entry, the ASN is the primary key.

This table allow us to dump easily all the ASNs of all the IPs we found in the datasets.

ASNsDescriptions a new entry is created if there is any new information or if an information is updated.

The following information is extracted of the RIS whois entries:

- The owner announcing the IP-block containing the IP
- The IP block
- The source of the RIS whois entry

  Note It will always be riswhois.ripe.net but in the future, we may decide to use a BGP session directly on a router.

4.1.2 Ranking and Voting

The two tables used to display the graphs are the following:

Sources contains only the names of the sources we can find in the database. For each entry, the source name is the primary key.

It is used to display the results by source.

History contains the information needed to display the graphs and the one used to compute a report

- the ASN of this rank
- the timestamp of the entry, set to the creation time
- the rank in IPv4 and IPv6

  Note this rank is unbalanced: it allows us to modify the weight of each source

There is also two other tables, user and votes. They are actually not used but will allow us in a near future to implement the voting functionality, for the authorized users.

Users user database, used for the login. Contains only a user name and a password.

Votes Save the history of the votes, contains the ASN and the value of the vote. It will also be possible to put a commentary.

4.2 Data Aggregation

4.2.1 Fetching the datasets

It is not the case for all of them but all the datasets freely available on the Internet are automatically downloaded.

When the download is finished, the process check if the file is new. Only if the file is new, it will be moved in the directory checked by the parsing process of this particular source.

Note there is one process for each source.

4.2.2 Parsing the datasets

As we know, the new files are copied in a special directory, for each source. The parser check periodically this directory and if there is a new file, it is parsed.

The parser will extract all the information of the datasets and put them into the database.

Note as for the fetching, there is one process for each source.

A description of all the actual modules is available on page 30. If you want to learn how to write a new module, please read on page 30.

When the parsing is finished, the system will automatically fetch the rest of the information it needs and there is neither further module to write nor any other action to proceed: the raw information is processed and computed automatically to generate the ranking and build the reports.

4.2.3 Getting the (RIS) Whois entries

There are two processes: the first one for the RIS Whois entries and the second for the whois entries.

They are working like this:

1. Ask the redis database if there is an entry for the current IP
2. If not, put the IP in the set of pending queries (there are two sets, the first one is called *ris* and the second *whois*)
3. Ask periodically if there is an entry for this IP

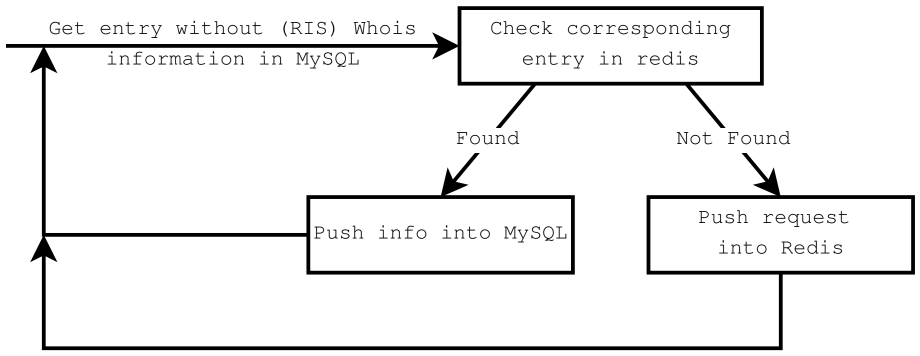
Get entry without (RIS) Whois information in MySQL
Check corresponding entry in redis
Found
Not Found
Push info into MySQL
Push request into Redis

Figure 3: (RIS) Whois process

This process does not do any network connexions with any other computer (if redis is running on localhost).

Note it is also multiprocessed: each process getting (RIS) Whois entries will receive a certain interval of IPsDescriptions. This interval is based on the number of allowed processes (in the default configuration: 4) and the number of RIS entries to fetch.

There is also a limit on the size of the interval: this way it is possible to reduce the RAM usage without influencing the speed.

4.2.4 Fetching the (RIS) Whois entries

There are one or more processes fetching the (RIS) Whois entries for each available Whois server (in the default configuration: 4 by server). The sets of pending queries are identified by the name of the server to query.

They are working like this:

1. Get a pending query from the redis database
2. Fetch the entry from the server
3. Set the entry in the database with the query as key and the entry as value

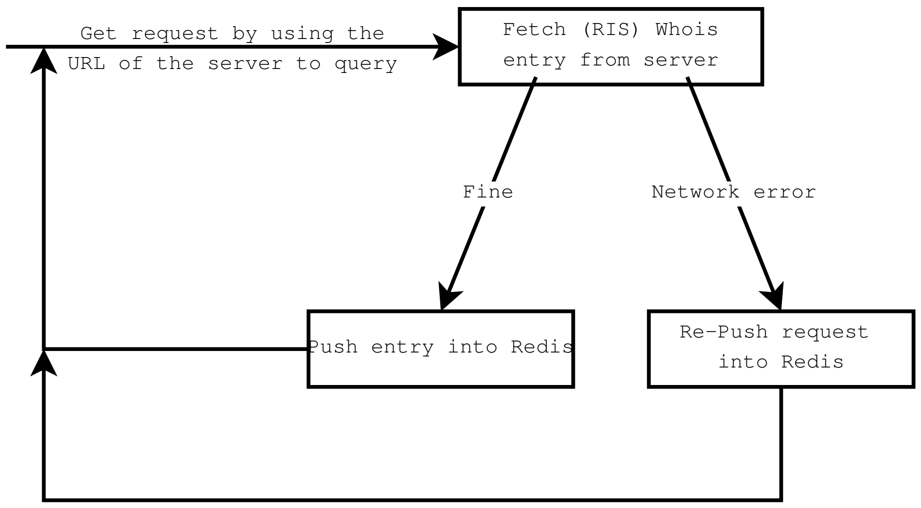
Get request by using the URL of the server to query
Fetch (RIS) Whois entry from server
Fine
Network error
Push entry into Redis
Re-Push request into Redis

Figure 4: (RIS) Whois fetching process

It is not always possible but we always attempt to use the keepalive capabilities of the servers: it is almost five time faster to stay connected than to reconnect between each query. Actually, only the servers of the RIPE are supporting this option. Of course, the local whois server does too.

Note the network is sometime unstable or the remote servers may have connectivity problems. This is why we catch all the network error and attempt to do the same query later.

4.2.5 Sorting the whois entry

As we already saw, all the whois entries are not on the same server and to be able to fetch the right whois entries of an IP, we need to find the server to query. This task will be performed by an other process:

1. Get a pending query from the redis database
2. Find the assignation by using the algorithm described on page 24
3. Set the entry in the database with the server name as key and the query as value

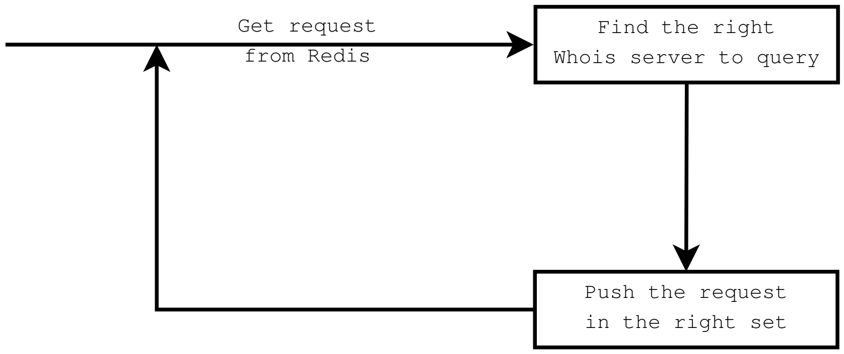

Figure 5: Whois sorting process

Note it does not concern the RIS Whois entries: RIPE NCC provide a global RIS Whois server which is able to answer to every query.

Tow UML diagrams representing the process of getting from redis, fetching and sorting the whois query for the RIS Whois (on page 45) and Whois (on page 46) entries are available in the annexes.

4.3 Ranking

Based on the information extracted from the datasets and the ones fetched from the Internet, we are able to compute a ranking.

4.3.1 Routing information

The routing information are extracted from the dump of the BGP table of RIPE NCC. We are using the routing table collected by the Remote Route Collectors (RCC) called rrc00, it is the biggest one and he “knows” all the routes of the Internet. It allows us to build a database with all the ASNs and the networks they announce.

The raw data are saved in in MRT format [25] and we need to use the libbgpdump library [26] to convert the raw data to a readable format and be able populate the database.

The processing of the file is also multiprocessed, the converted file is splitted and one process populating the database if launched on each file (by default, 4 at the same time).

When populated, the database is used to compute the number of IPs announced by the ASNs: it is an very important information because the rank of the ASNs is based on it. A small subnet with a certain amount of suspicious IPs will have an higher rank than a big subnet with the same amount of suspicious IPs.

4.3.2 Computing

The routing database has to be fully populated before computing the ranking. Elsewhere, we can not be sure that the number of IPs announced for an ASN is right. The easiest way is to sequentially populate the routing database and after that compute the ranking. There is a new dump every eight hours, it is enough if the ranking is computed every eight hours too.

For each ASN found in the database, the following actions are achieved:

1. Counting the number of IP announced by the ASN
2. Counting the number of IPs, by sources, present in the database for the current day
3. For each source, dividing the number of IP in the database by the total number of IP announced
4. Saving in the database

Note as usual, it is multiprocessed: each process compute the ranking of a certain interval of ASNs (by default, 4 processes at the same time).

Like for importing old datasets, it is possible to compute the ranking for old entries but we will also have the same problems. See on page 34 for more information.

5 Libraries

5.1 IPs Keys generator

5.1.1 Context

We have on one side a list of IP sets, each of them is connected to a particular information and on the other side IP addresses.

This library aims to provide a fast method to get the better information corresponding to this IP addresses. To find this information, we have to find the smallest subnet where the IP is.

Usage The library need to know if we are working with IPv4 or IPv6 and the first and the last IP of the interval to compute.

5.1.2 Example

The list of sets and corresponding information:

- `1.0.0.0 - 1.255.255.255 : Big provider`
- `1.1.1.0 - 1.1.1.255 : Small provider`
- `1.1.1.1 - 1.1.1.5 : Company selling websites`

IP address : `1.1.1.3`

This IP address is in all the sets but the most accurate information is that it is owned by the company selling websites.

5.1.3 Algorithm

First idea Push all the IPv4 and IPv6 in the database.

Pros

- Easy
- Fast: access to the information in O(1)

Cons

- there is about $4,3 \cdot 10^9$ IPv4 and about $3,4 \cdot 10^{38}$ IPv6, it is impossible to push in the database all the IPv4 (and obviously all the IPv6)

Second idea To reduce drastically the number of keys, the easiest solution is to push in the database the first and the last element of the interval.

Pros

- Easy
- Few keys

Cons

- Huge amount of queries to redis: we have to find the best interval and it can take some time: in the worst case, it will be O(n) with n the number of IPs in the interval.

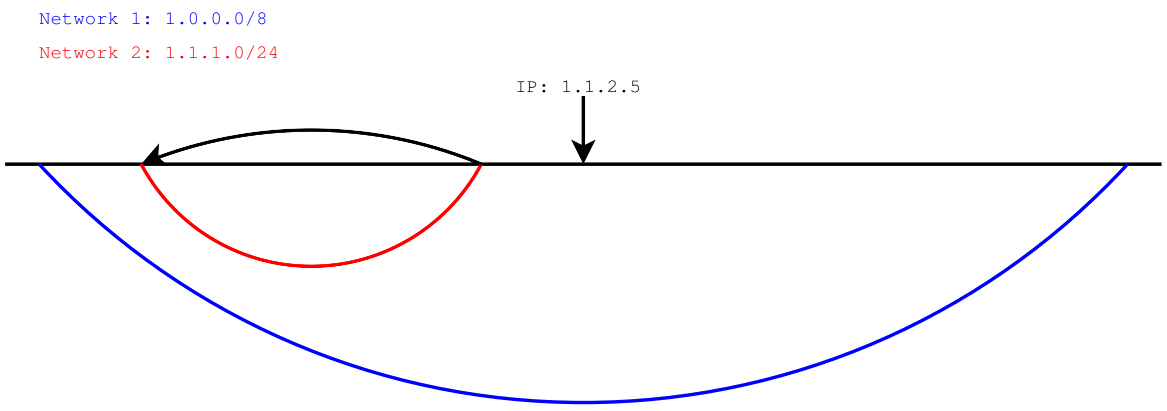

Figure 6: IPs Keys generator 1

We have to do a query for each IP until we find the interval containing ours. Even if we are able to jump the intermediate networks, it will take too much time.

For a network like 1.0.0.0/8 with more than $16{\cdot}10^6$ IPs, it takes more than 15s.

Third Idea The keys saved in redis are strings, the easiest way is to find an algorithm based on the string representation of an IP address.

For the first and the last IP of an interval, we split the address with the . (IPv4) or the : (IPv6), find the equal part and push it into the database. Equals part means the identical string in the two IPs, from left to right, and stop on the first different character.

The value of this key will be the following: *<integer representation of the first IP of the interval>*_*<integer representation of the last IP of the interval>* and be used to ensure that the IP of the query is really in this interval.

Example 1. Generate a key for a simple interval

Interval: 1.1.1.0 - 1.1.1.255

Equal Part: 1.1.1

Value: 16843008_16843263

Example 2. Generate a key for a more complex interval

Interval: 1.1.1.0 - 1.1.5.255

Equal Part: 1.1

But be want more precision, the key will be the following: 1.1.{1-5} => 5 keys

Example 3. Generate a key for a very complex interval

Interval: 1.2.3.0 - 1.4.5.255

Equal Part: 1

The keys will be the following: 1.2.{3-255}, 1.3, 1.4.{0-5} => 252+1+6 = 259 keys

IPv6 the idea is the same but the intervals are usually big and there is never such intervals as in the third example, we always push intervals like in the first and second example, usually only like the second.

To get the smallest interval, we will split the IP address in the same way and begin to get the longest possible key.

Example 4. For the IP `1.1.1.1`, we will begin with `1.1.1` then `1.1` and finally `1`.

The second problem is that we will have sometime more than one value at a key: in IPv4, a lot of intervals are smaller than a /24. It is actually not a problem in IPv6 because even though the subnets are very big, they not shared between operators. But sooner or later it will used too.

This is why we want to be able to find the smallest network within a set of intervals.

To do it, we just have to split the the interval keys found for a IP, check if it is in this interval. When one if found, we have to verify that there is no other smallest network containing the IP.

Example 5. Looking for the smallest network

IP: 1.1.1.2

Network 1: 1.1.1.0 - 1.1.1.255

Network 2: 1.1.1.1 - 1.1.1.5

The equal part of the two networks and of the IP is *1.1.1*. In Redis, the key *1.1.1* contains the decimal representation of the two networks.

The two networks will be checked:

1. the IP is in Network 1
2. the IP is in Network 2 and Network 2 is contained in Network 1

   The best network for 1.1.1.2 is Network 2.

Pros

- Fast: the key will be find in O(3) for IPv4 and O(7) for IPv6. After that, we have to find the right interval in the set but it is just a comparison of a small number of values and goes very fast.
- Few keys

Cons

- More complicated

5.2 File splitting

5.2.1 Context

One of the requirements is that the system has to to allow mapreduce-like processing. The problem with files is that it is not possible to parse it with more than one process at the same time. It would be great to be able to split a big file into smaller one and launch different processes on this files. This is the aim of this library. It has also to be included in an existing code and should be as transparent as possible.

Usage The library needs only a file to split, the number of files to generate and the separator.

5.2.2 Example

The following lines are extracted of the dump of the BGP Routing table, the whole file is around 1Gb. It is very important to split it, at the right place. The right place will be the empty line between the two announce.

```
TIME: 08/19/10 08:00:00
TYPE: TABLE_DUMP_V2/IPV4_UNICAST
PREFIX: 1.9.0.0/16
SEQUENCE: 0
FROM: 193.136.5.1 AS1930
ORIGINATED: 08/18/10 13:33:40
ORIGIN: IGP
ASPATH: 1930 20965 3549 4788
NEXT_HOP: 193.136.5.1
TIME: 08/19/10 08:00:00
TYPE: TABLE_DUMP_V2/IPV4_UNICAST
PREFIX: 1.9.0.0/16
SEQUENCE: 0
FROM: 193.0.0.56 AS3333
ORIGINATED: 08/19/10 06:44:26
ORIGIN: IGP
ASPATH: 3333 1273 4788
NEXT_HOP: 193.0.0.56
COMMUNITY: 1273:12250 4788:200 4788:210
```

5.2.3 Algorithm

Separator The separator is used to split the file at a particular point: there is usually blocks in the files we want to split and we cannot cut the files within this blocks. usually it is a `\n`.

Splitting First, we get the size of the file and divide it by the number of files we want.

To split we just "jump" of the size we compute and look for the next separator. The data in the interval is finally copied in a new file.

When the end of the file is reached, we return the list of generated files. The original file is not removed.

5.3 Whois Client

5.3.1 Context

As we already explained on page 7, there is not only one whois server but many. The whois client has to be able to ask the right server.

5.3.2 Initialization

The whois client is based on the assignations given by the whois package of Debian/Ubuntu.

The ranges are pushed into the database using the library described on page 24. Each resulting key is associated to its whois server: `<IP_key>|<whois_server_URL>`

Each server may have some special options:

- port: port to connect to on the server
- pre: string prepended to the query
- post: string appended to the query
- keepalive: other prefix, makes the query keepalive (the server doesn't close the connexion after each query)

This options looks like this in the database: `<whois_server_url>:<option_name>|<option>`

The options are set depending on the server: on some servers, it is possible to reduce the response to the most precise information, and reduce the size of the entry in the database. An other very interesting option is to fetch only the entry in English (useful on some Asiatic servers).

This options have to be tested on each server. The default configuration is the best choice.

5.3.3 Usage

The client is used by the fetching process, described on page 21.

Finding the best range The best range is found using the library described a on page 24.

Fetching When the best range is found, the fetcher initialize the connection to the server using the information put in the database during the initialization and fetch the entry. If possible, the connection is not closed but very few servers authorize it, it slows a lot the process. That is why we have also a proper whois server, described on page 32.

5.4 Whois Parser

The basic idea of the parser comes from the project pywhois [27].

5.4.1 Context

Every Whois server has his proper format for the response. It would be interesting to implement a simple way to handle every response and be able to parse it automatically to get only the information we want.

5.4.2 Implementation

It is possible in python to create new arguments "on the fly" by overloading the __getattr__ function. The idea is quite simple: if the argument does not exist in the class (standard python functions), we attempt to get it from an other way.

In our case, we will use a dictionary: each entry in the dictionary will have as key the name of the argument to get and as value a regexp that will be used to extract the information we want. For every server we want to handle, we need to write a python dictionary.

Dictionary

```
RIS = {
'route'      : 'route[6]?:[ ]*([^\n]*)',
'origin'     : 'origin:[ ]*AS([^\n]*)',
'description': 'descr:[ ]*([^\n]*)'
}
```

By using this dictionary, when a RIS Whois entry is passed to the class, we just have to do whois.route to get the route.

The class is initialized with the URL to the server which did the query. This way, it is possible to select the right dictionary.

6 Modules

Important Note the format of the modules described there is almost deprecated: will change in the next weeks, see on page 39.

6.1 Abstract part

Every module is a subclass of IPUpdate. This class is responsible for the insertion of the data extracted from the datasets into the MySQL database. The module will only define the functions used to extract the information of the dataset.

6.2 Existing modules

6.2.1 Format

Plain text The easiest format, the datasets of Dshield, abuse.ch and Abusix are provided in plain text.

XML/RSS Arbor provide its list in a RSS file.

CSV The lists of Shadowserver are in CSV. It is good to know that the order of the fields in the three lists is different... That is why there is three modules.

6.2.2 Types

Type 1 the dataset contains only IP addresses and a date for the whole file (or at least, the module set a date).

Type 2 the dataset contains more information like the type of the infection, a date for each entry and maybe more. All of this will be saved.

6.3 Write a new module

A module is a class, the name of each class hast to bee unique: is allows us to know the origin of all the entries in the database. The modules have also to define an home directory for their datasets.

That is all for the standard part, there are actually two types of modules :

6.3.1 First type

The dataset contains a list of IPs, it may also give the date of generation of the report, it will be extracted too.

This type require:

- The function *parse*: extract the IPs
- A variable *datetime*: if possible extracted of the file. If there is no date in the file, the date will be set to *today* (datetime.date.today()).
- The variable *module_type* set to 1

6.3.2 Second type

The reports provide more information on the attackers. They provide the IP but also the time of the attack, and the type attack (the name of the malware, if known).

This type require:

- The function parse which return a table of table with each line like: [IP, date, infection, rest of the line]
- The variable module_type set to 2

Note the atlas report is an XML file, the information we need are extracted of the file and saved as a string in the database. Shadowserver is a CSV report, the whole line will be pushed in the database.

6.4 Automation

The automation is almost done through the configuration file but if you write a new module, you will have to edit *lib/modules/__init__.py* and append the module to import first.

6.4.1 Downloading

It is not always possible to download automatically the new datasets but if it is public, you just have to edit *etc/bgp-ranking.conf* and add in the section *raw_fetching* an entry like:

```
uniq_name = module directory URL
```

And restart the service *etc/init.d/start_fetch_raw_files.py.*

6.4.2 Parsing

To start a process watching in the root directory of a new module, edit *etc/bgp-ranking.conf* and add in the section *modules_to_parse* an entry like:

```
classname = impact
```

impact is the weight of the new module on the global ranking.

And restart the service *etc/init.d/start_parse_raw_files.py.*

6.5 Other Features

6.5.1 Configuration

All the configuration is defined in *etc/bgp-ranking.conf.* You can almost configure all the system. More details in the file itself.

6.5.2 Modularity

You can run each initscript in the order you want: they are all totally independent.

Example the parsing will not be done until the files are fetched but if the parsing service has nothing to do, it will just wait a certain amount of seconds (defined in the configuration file) and check again if there is something to do.

Because the sharing of the information between the services is done using Redis it is possible to run each part of the system on a different computer.

6.5.3 Multiprocessing

Except the fetching of the datasets, each part of the system support multiprocessing: for each service, you can configure the number of process in the configuration file.

7 Whois server

This is an other project, all the details will not be given in this document, only the most important points.

7.1 Context

Querying the whois servers through the Internet has two disadvantage:

1. it takes a big amount of time: some queries take almost one second. With the number of query we have to do, it is a impossible.
2. after a certain amount of queries (around 10.000) some servers will blacklist our IP

The project is fully opensource [28].

7.2 Goal

The most challenging part of the whois server is that it has to be faster than the querying through the Internet.

7.3 Algorithms

To initialize the whois database, we will re-use many of the libraries developed for the Ranking system:

- File splitter (on page 27) to use many processes to initialize the database
- IP keys generator (on page 24) to generate the IP keys
- Whois parser (on page 29) to split the entries of the dump

Because all the databases dumps are very different, all the scripts to initialize the databases have many differences but the idea is always the same: each IP block has a unique identifier and Keys of entries associated to the block look like this in redis:

```
<unique_id>:<entry_name> - <entry_key>
```

Example: `1:owner - RIPE-1234`

The entry key is used to extract the information on the key.

Example: `RIPE-1234 - <information on the key>`

The entry names are as normalized as possible: like this we just have to define a list of names and try to get all of them when a client makes a query and return a dump of all the strings extracted from the database.

7.4 Results

With a local whois server, the queries takes always the same time: around 0.05 seconds. It is 5 times faster than the querying in "keepalive mode" of the RIPE and almost 20 times than the "normal mode" offered by the other servers.

7.5 Problems

A whois server needs a big amount of memory: we have to generate a lot of keys to be able to link all the information together and at the end, the whois database of the RIPE needs almost 3Gb of ram. If we want to have all the whois databases on a single server, it will need at least 8Gb (it is just an estimation).

Generating the tables takes a lot of time: depending on the size of the dumb, it may need around one hour.

An other quite annoying problem is that on a 32bits system, it is not possible for a single process to use more than 4Gb of memory [29], even with a PAE kernel which allow the whole system to recognize until 64Go of memory. This problem can be solved by using a 64bits system.

7.6 Improvements

It is actually not possible to do queries on ASNs. This functionality will be implemented as soon as possible.

We actually do not have a dump of all the whois databases: only the RIPE, ARIN and LACNIC gave us a dump, AFRINIC and APNIC do not. It would be great to program the rest of the modules and to be able to run on a single server all the databases.

A website may be developed in the future.

8 Website

8.1 Used technologies

Cheetah [30] is a templates system in python, it allows us to generate pages on the fly.

Cherry-py [31] is a web server able to use the cheetah templates.

8.2 Status

The website is a work in progress. It is possible to do the following the tasks on the website:

- View the "worst ASNs" of the last 24 hours, by sources and a merged view of all sources.
- Display a graph of a list of ASNs, by sources and a merged view of all sources.
- Display a graph for a particular ASN, using all the entries of the history
- Display the latest IPs (last 24 hours) found for a particular ASNs, their block and the description of the ASN

9 Known bugs and problems

9.1 Handling old datasets

It is possible but not totally implemented and we will have a nasty problem: the association between the IP and the ASN is based on the actual version of the RIS Whois database and this database evolve quite often. If we want to be able to import old data we need also to implement a module which will use the corresponding version of the BGP Routing table.

It is possible to solve this problem but it will be quite complicated and not very useful. This functionality will be not implemented in a near future.

9.2 IPv6 and ranking

IPv6 is fully supported by the aggregation system but not for the ranking: we need first to find an other formula because the size of the assignations by ASN is so big in IPv6 that the result of the division of the number of IPs found by the number total of IPs is almost zero.

This problem will be fixed as soon as possible.

Part III

Usage

10 Installation

The ranking system needs some other applications to work. The following has been tested with Ubuntu 10.4 and will work with other distributions but some of the name will changes and it is possible that you have other dependencies.

10.1 Dependencies

All the dependencies are opensource and freely available on the Internet.

10.1.1 Package of the depots

Some of the dependencies are available in the depots, we usually chose this method because it is easier.

It is the case for the five following Python libraries (the names are the names of the packages in Ubuntu) :

- python-ipy [32]: allow us to do many operations on the IP addresses (normalize, cast...), works in IPv4 and IPv6
- python-dateutil [33]: used to convert a string to a date, recognize a lot of string representations.
- python-elixir[34]: the library implementing the Active Record model pattern to access to the database
- python-mysqldb[35]: the python connector to MySQL (if you decide to use an other database, you will need an other connector
- python-feedparser [36]: used to parse the entries coming from Atlas, which are provided as RSS feeds

The following packages are not directly used by the application but are used to compile other programs:

- build-essential [37]: meta-package installing GCC, make... and all the necessary packages necessary to compile a program
- zlib1g-dev [38] and libbz2-dev [39]: dependencies of libbgpdump

10.1.2 Package outside the depots

The depots usually contains not the latest version of the packages but it is sometimes important for us to use some functionalities only present in the latest version of the application.

It is the case for this two programs:

- cheetah [30]: it is the templates system used by the website. The version 2.4.2 is necessary.
- cherry-py [31]: it is the server supporting the cheetah templates and we need at least the version 3.1.2

An other possibility is that the needed program does not exists in the depots:

- libbgpdump [26]: generate a dump of the RIS routing database from the MRT format to plain text
- Rgraph [40]: generate graphs in javascript

10.1.3 Live versions

For some programs we want to use the very latest version of the program:

- redis [41]: the version 1.2.6 is quite slow and the version 2.X is already very stable and usable. The ranking system is actually using the trunk but it is also possible to use the version 2.0 when it will be released.
- redis-py [42]: the python wrapper for redis. We are using the live version for the server, the client has to support and use the latest functionalities that offer the server.
- whois server [28]: necessary only if you want to use it. Allow you to do whois queries in local but need a lot of memory.

10.2 Ranking system

When all the libraries/programs are installed, you will be able to run the ranking system.

10.2.1 Services

First of all, you will have to be sure that the MySQL and the redis server are running. Feel free to tweak the default configuration of the both servers but the default configuration works fine.

You should create on MySQL a user and allow him to do at least the following operations: CREATE TABLE, CREATE INDEX, SELECT and UPDATE. Of course, you can use the root user but it is always better to create a dedicated user for the different services running on the server.

When it is done, create as root two databases: one for the data and the other for the ranking.

If you want to use the whois server and when the redis server is started you have first to initialize the database with the sources you want (RIPE, ARIN, LACNIC). To do it, launch the following scripts, depending on your needs: *lib/init/init_{arin,ripe,lacnic}.py*

And start the listener by launching this script: *etc/init.d/start_whois_server.py*

10.2.2 Configure the system

First of all, you have to fetch the project, it is in a git repository. To get it, do the following:

```
git clone git://gitorious.org/bgp-ranking/bgp-ranking.git bgp-ranking
cd bgp-ranking
```

The configuration is in `etc/bgp-ranking.conf` and there is at least two things you want to change: the root directory of the application. it you are in the directory bgp-ranking, just do `pwd` and replace the actual value by the result. The second is the MySQL section: set your own user, password and the correct names for the databases.

As you can see, you can modify a lot of variables, most of them should not be changed but you can for example increase or reduce the number of processes or activate the fetching of whois entries.

10.2.3 Initialize the databases

The last point before starting the system is to initialize the databases by using the scripts present in *lib/db_ init*: *init_ ranking.py* and *init_ voting.py* will respectively initialize the database containing the data and the database containing the history.

You have also to initialize a redis database containing the assignations of IP addresses to the whois servers (*init_ assignations_ redis.py*).

11 Start the application

11.1 General information

All the initscripts are in *etc/init.d*, they are all totally independent and you can start each of them in the order you want.

Example: the parsing will not be done until the files to parse are fetched but if the parsing service does not find a file to parse, it will just wait a certain amount of seconds (defined in the configuration file) and check again later if there is something to do.

Because the sharing of the information between the services is done using Redis it is possible to run each part of the system on a different computer.

Except the fetching of the datasets, each part of the system support multiprocessing: for each service, you can configure the number of process in the configuration file.

11.2 Launching

This scripts have always to be started:

- *start_fetch_bview.py*: start a process that fetch the dump of the routing database and put it in the directory checked by *start_push_update_routing.py* if the dump is new
- *start_fetch_raw_files.py*: start a process that fetch the raw files from the sources and put it in the directory checked by *start_parse_raw_files.py* if the file is new (more information chapter 4.2.1 on page 20)
- *start_parse_raw_files.py*: start a process that parse the raw files, more information chapter 4.2.2 on page 20
- *start_get_ris_entries.py*: start the process described chapter 4.2.3 on page 20
- *start_fetch_whois_entries.py*: start the process described chapter 4.2.4 on page 21
- *start_push_update_routing.py*: start a process that check if there is any new routing dump. If there is a new, it will be pushed into the database. The second part of this script will compute the ranking based on the information of the last 24 hours

This two scripts are not absolutely necessary, they will just allow us to fetch the whois entries. They can not start if this fetching is deactivated in the configuration file :

- start_get_whois_entries.py: start the process also described chapter 4.2.3 on page 20
- start_sort_whois_queries.py: start the process described chapter 4.2.5 on page 22

Part IV

Improvements and future usage

12 Improvements

12.1 Databases

The first change to do on the databases will be to merge the two, the one containing the data of the datasets and the one containing the ranking:

- there is a table containing only the sources in the ranking database. This sources are also present in the dataset database, as field. It will be merged
- the ASN table of the dataset will be used in the ranking database: the History table has a field containing the ASN.

12.2 Configuration

It is actually possible to modify the weight of a source through the configuration file, and only through it. When it is changed, the graphs are automatically redraw.

The improvement will be to give the possibility to the user to change this weight directly through the website.

12.3 Importing datasets

Actually, to be able to import new datasets, you have to write a sub-class to ip_update. The problem is that you have to understand how this class works and it is not that easy.

A big improvement would be to use a redis database as interface between the modules and the class that push the new raw data in the database.

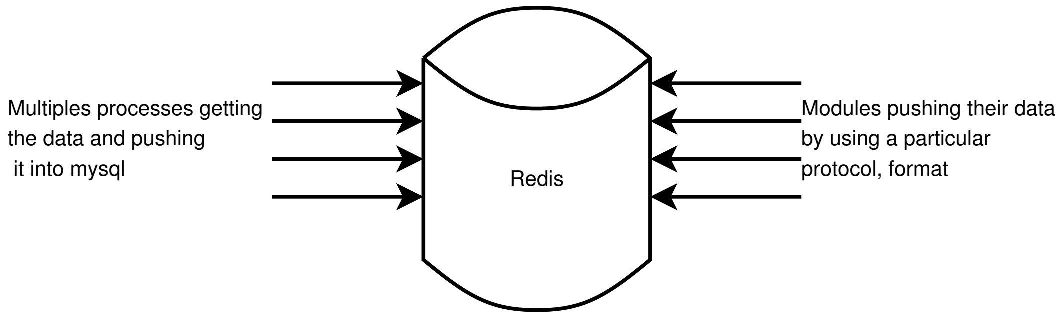

Figure 7: New type of modules

12.3.1 Basics

For each new entry, we need at least two information:

- an IP address
- the source of the IP

A third information is very important but if the module does not give it, it will be set to "today": the timestamp

There is two other fields but they only exist on some particular datasets:

- the type of the infection
- a text containing other fields, separated by semicolons

12.3.2 Format

It is not possible to put directly the IP into the set used by the process pushing the new entries into the database for the simple reason that it is possible to have the same IP more than one time at the same moment: if it comes from different sources. The entries put in the set called *ip* will have this format: *<ip>_<source>* because if the same IP is present more than one time in a particular dataset, the module should silently remove the next occurrences.

Note each module has to ensure that if the same IP is present more than one time in a dataset, it will not be pushed in the redis database.

And if possible the module will push into redis an entry for this other fields:

```
<ip>_<source>:timestamp -> timestamp (timestamp in UTC)
<ip>_<source>:infection -> infection
<ip>_<source>:raw -> other in formations
```

12.3.3 Multiprocessing

Every module can work at the same time.

The process pushing the new entries into the MySQL database will look in the *ip* set and pick the new entries if it find something.

WARNING: The modules have to push first all the other fields **before** pushing the IP into the set, elsewhere we will have inconsistent entries.

12.3.4 Advantages

This way, it would be possible to write modules in the language you want and just push the new information into redis. The new module will just have to respect a format but will be completely independent of the existing code.

It would also be possible to push new data directly through the network, from sensors.

12.4 Modules

When the new modules system will be implemented, some new modules will arrive: it would be for example very easy to use the logs generated by fail2ban which block the IP addresses of clients attempt to do SSH brute force on servers.

12.5 Ranking

It is actually only possible to obtain a ranking by ASN. In a near future, a ranking by subnet will be implemented.

It is also planed to permit to the user to vote against a particular AS. It will soon be possible to see the ranking with and without the weight given by the other users.

12.6 Website

There is a lot of work to do on the website: the rendering is actually very static. It would be interesting to see the details of a certain AS during a certain amount of time, or only the graph for a particular subnet.

Everything is possible: all this information are saved in the database and it is possible to do query on it. As soon at possible, the website will be updated to allow the users to get more information.

To implement the voting system properly, the website need a login functionality to identify the users and allow them to vote.

13 Future usages

It is difficult to say but the most obvious usage will be the generation of blacklists based on the ranking by ASN and subnets.

It is also thinkable to write a Firefox module that interrogate the server before connecting to a website to know if the IP is known or if there is a lot of suspicious IP in the same block or provided by the same AS.

References

[1] How the 'Net works: an introduction to peering and transit - Ars Technica 6

[2] The Art of Peering: The Peering Playbook 6

[3] RFC 4271: A Border Gateway Protocol 4 (BGP-4) 6

[4] BGP Policies - Matthew Caesar and Jennifer Rexford 6

[5] Exploring Autonomous System Numbers by Geoff Huston, APNIC in the The Internet Protocol Journal - Volume 9, Number 1 7

[6] Routing Information Service 7

[7] About Dshield 8

[8] FAQ Arbor 8

[9] FAQ ZeuS 9

[10] Shadowserver Mission 9

[11] Kaspersky Security Bulletin 2007 9

[12] Botnet price for hourly hire on par with cost of two pints 9

[13] P2P Botnet 10

[14] Blackhole your malware 11

[15] MapReduce 13

[16] Redis Official Website 13

[17] Why AGPLv3 - MongoDb 12

[18] Divert filtering 13

[19] RFC 2827- Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing 13

[20] BGP AS / ISP Security Ranking 13

[21] RFC 5575: Dissemination of Flow Specification Rules 13

[22] SQLAlchemy - Supported Databases 16

[23] Active record pattern 16

[24] A fifteen minutes introduction to Redis data types 17

[25] MRT routing information export format - Version 11 23

[26] Source of libbgpdump 23, 36

[27] Official Website of pywhois 29

[28] Official website of the new Whois Server 32, 36

[29] Maximum Memory and CPU Limitations for Linux Server 33

[30] Cheetah template - Official Website 34, 36

[31] CherryPy - Official Website 34, 36

[32] IPy - Official Website 35

[33] python-dateutil - Official Website 35

[34] Elixir - Official Website 35

[35] MySQLdb - Official Website 35

[36] Feedparser - Official Website 35

[37] Package build-essential - On Ubuntu 35

[38] Package zlib1g-dev - On Ubuntu 35

[39] Package libbz2-dev - On Ubuntu 35

[40] http://www.rgraph.net/ - Official Website 36

[41] Redis - Official Website 36

[42] Redis-py - Official Website 36

Example of BGP Entry :

TIME: 07/19/10 08:00:00
TYPE: TABLE_DUMP_V2/IPV4_UNICAST
PREFIX: 41.138.240.0/21
SEQUENCE: 10022
FROM: 91.103.24.2 AS42109
ORIGINATED: 07/18/10 23:36:39
ORIGIN: IGP
ASPATH: 42109 41965 41877 3356 174 42652 12684 12684
NEXT_HOP: 91.103.24.2
COMMUNITY: 3356:2 3356:22 3356:86 3356:503 3356:666 3356:2067

The riswhois entry of 74.125.43.99 the 30th of August 2010:

route: 74.125.42.0/23
origin: AS15169
descr: GOOGLE - Google Inc.
lastupd-frst: 2010-03-18 20:10Z 202.249.2.20@rrc06
lastupd-last: 2010-08-29 19:50Z 80.81.192.220@rrc12
seen-at: rrc00,rrc01,rrc03,rrc04,rrc05,rrc06,rrc07,rrc10,rrc11,rrc12,rrc13,rrc14,rrc15,rrc16
num-rispeers: 103
source: RISWHOIS

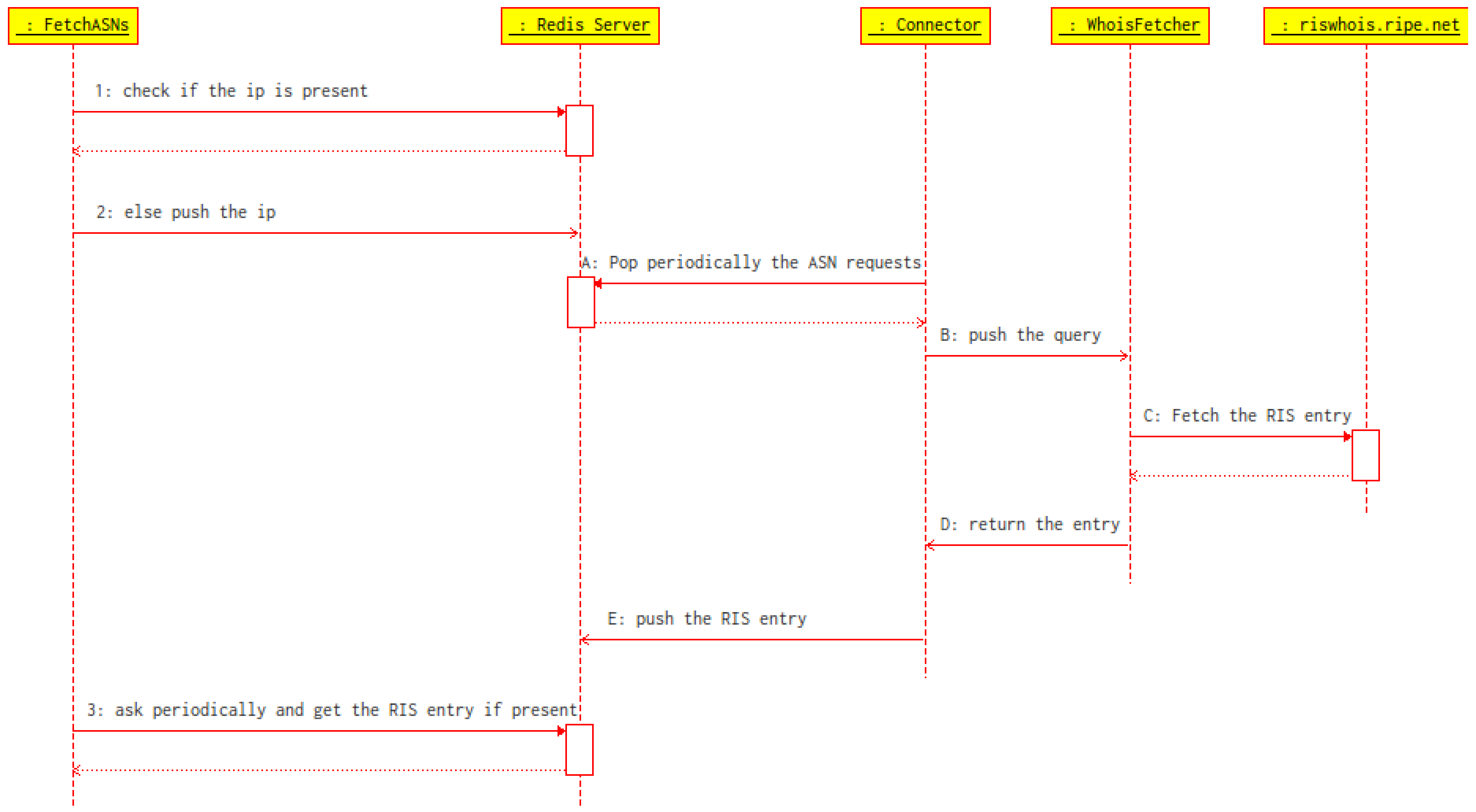

Figure 8: Get RIS Whois entries

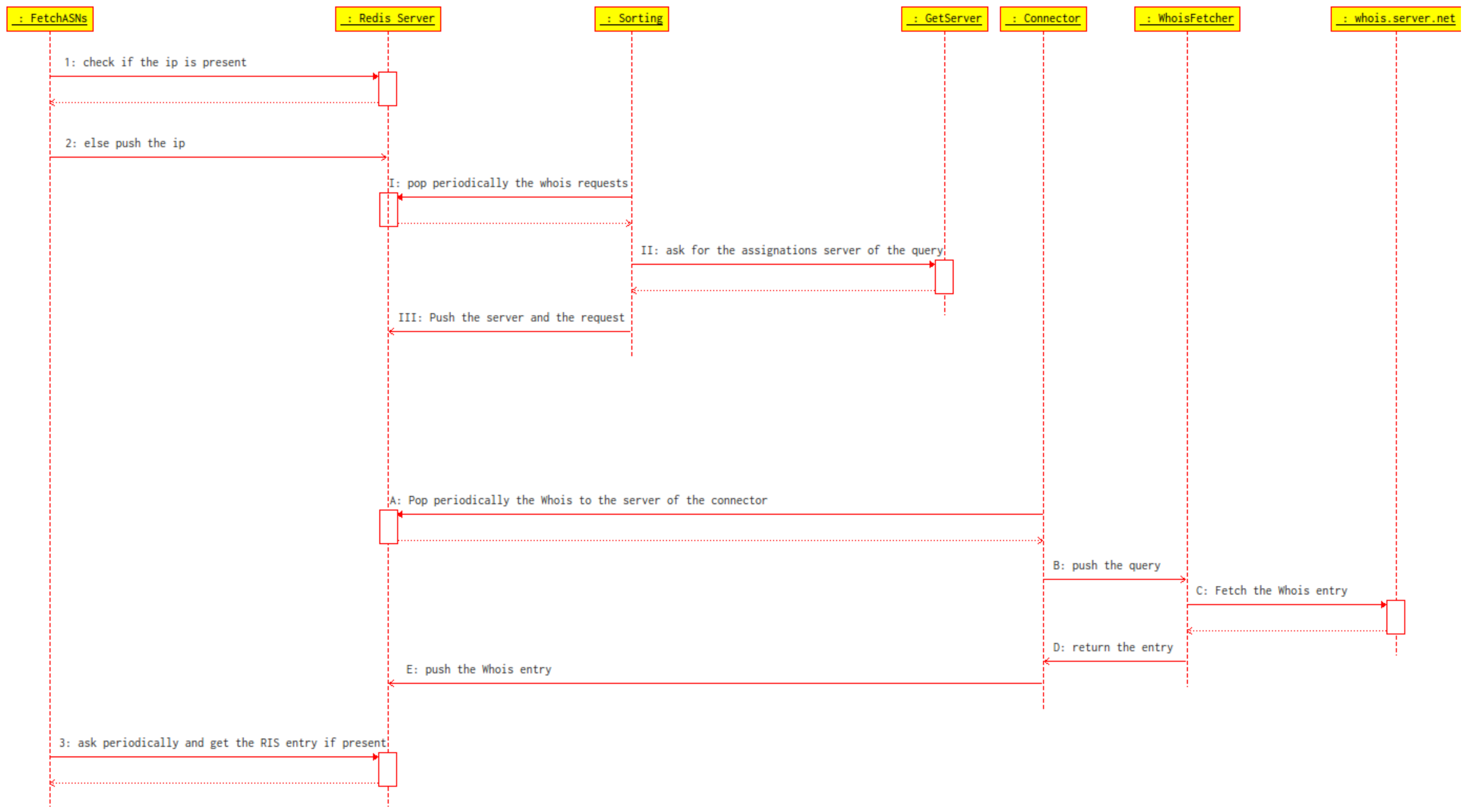
: FetchASNs
: Redis Server
: Sorting
: GetServer
: Connector
: WhoisFetcher
: whois.server.net
1: check if the ip is present
2: else push the ip
I: pop periodically the whois requests
II: ask for the assignations server of the query
III: Push the server and the request
A: Pop periodically the Whois to the server of the connector
B: push the query
C: Fetch the Whois entry
D: return the entry
E: push the Whois entry
3: ask periodically and get the RIS entry if present

Figure 9: Get Whois entries

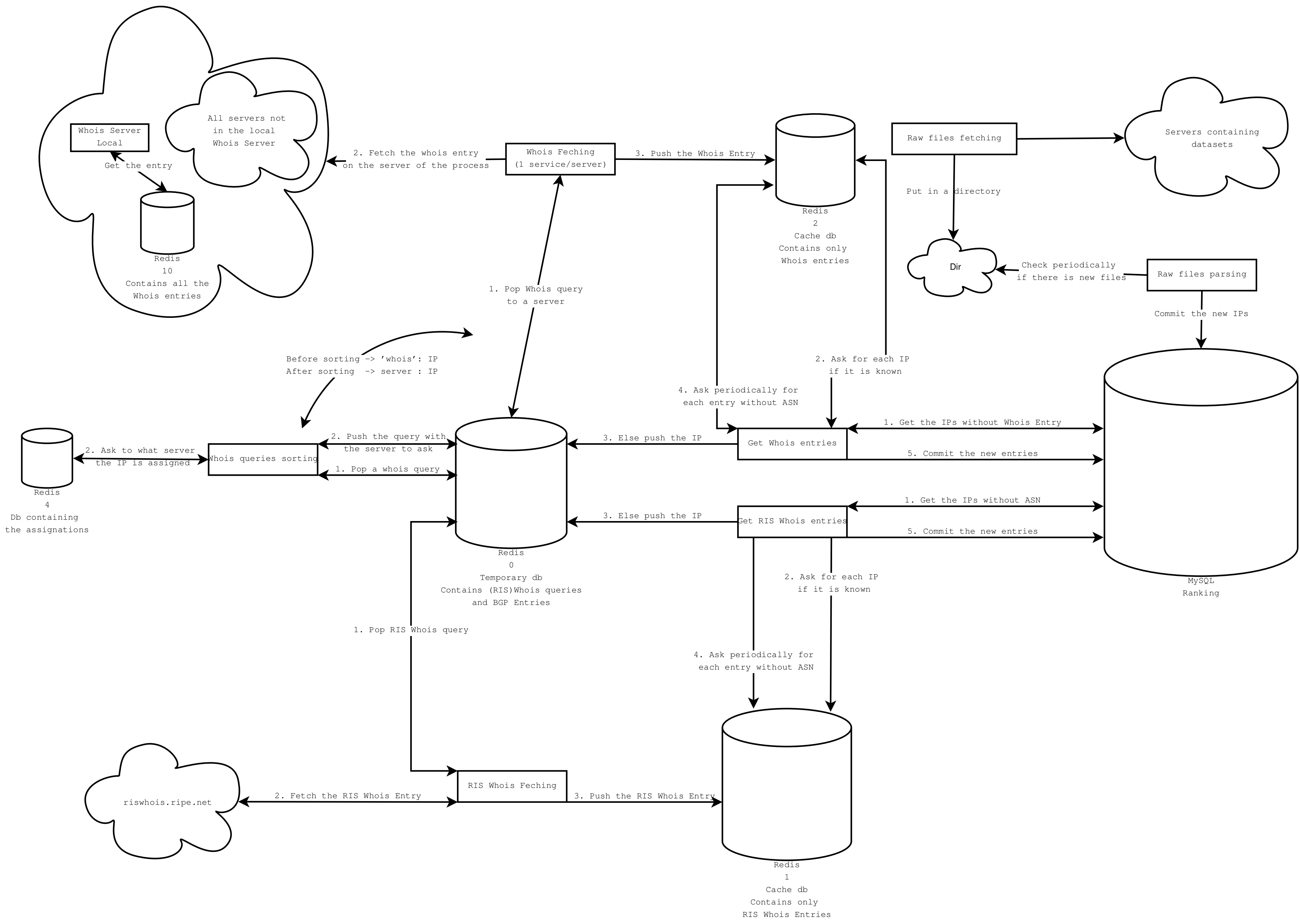
Whois Server Local
Get the entry
All servers not in the local Whois Server
Redis 10 Contains all the Whois entries
2. Fetch the whois entry on the server of the process
Whois Feching (1 service/server)
3. Push the Whois Entry
Redis 2 Cache db Contains only Whois entries
Raw files fetching
Servers containing datasets
Put in a directory
Dir
Check periodically if there is new files
Raw files parsing
Commit the new IPs
1. Pop Whois query to a server
Before sorting -> 'whois': IP
After sorting -> server : IP
2. Ask for each IP if it is known
4. Ask periodically for each entry without ASN
Redis 4 Db containing the assignations
2. Ask to what server the IP is assigned
Whois queries sorting
2. Push the query with the server to ask
1. Pop a whois query
3. Else push the IP
Get Whois entries
1. Get the IPs without Whois Entry
5. Commit the new entries
3. Else push the IP
Get RIS Whois entries
1. Get the IPs without ASN
5. Commit the new entries
MySQL Ranking
Redis 0 Temporary db Contains (RIS)Whois queries and BGP Entries
1. Pop RIS Whois query
2. Ask for each IP if it is known
4. Ask periodically for each entry without ASN
riswhois.ripe.net
2. Fetch the RIS Whois Entry
RIS Whois Feching
3. Push the RIS Whois Entry
Redis 1 Cache db Contains only RIS Whois Entries

Figure 10: Technical global view

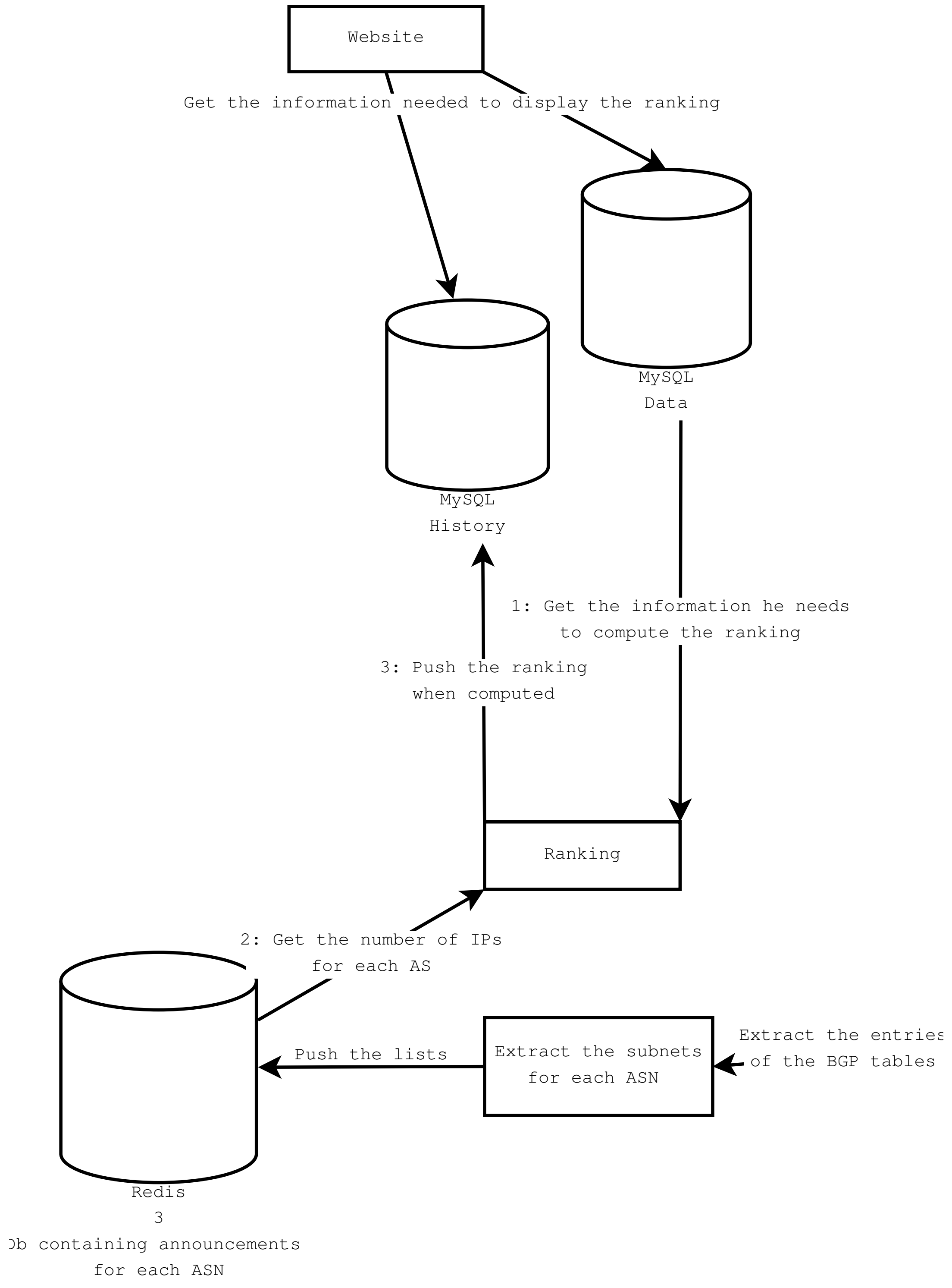

Figure 11: Technical global view - Ranking