"Politehnica" University Timişoara
Computer Science and Engineering
Department

Graduation Diploma Project

Supervisors:
Henk Uijterwaal (RIPE NCC)
Ioan Jurca (UPT)

Author:
Dan Ardelean

2002

"Politehnica" University Timişoara
Computer Science and Engineering Department

Tracking Routing Black Holes with the RIS

2002

Acknowledgements

First I want to thank Henk Uijterwaal, New Projects Manager, my supervisor at RIPE NCC, for all the support, the idea of this interesting project, the guidance he provided during my trainee-ship at RIPE NCC and for all the good comments he made on this paper. Next I want to thank Ioan Jurca, my professor and my supervisor at "Politehnica" University Timişoara for all the good comments on this thesis, encouragement and support.

Special thanks go to René Wilhelm, Scientific Programmer, for all the technical help and particularly for helping me with the graphic conversion problems and also for convincing me to write this paper in LaTeX.

I would like to thank Daniel Karrenberg, Chief Scientist at RIPE NCC, for all the comments made on my work and also for providing very good ideas with black-holes plotting issues.

Also, special thanks go to Matthew Williams, Customer Liaison Engineer at RIPE NCC who provided me with very good ideas, particularly concerning reality checking on my project and also the two ratios that he suggested (5.5.2 and 5.5.3). I must also thank the two network engineers of the RIS project: Arife Coltekin and Luigi Corsello for helping me with putting the black holes project into production on the RIS's servers. And of course, I have to thank Olaf Kolkman, Scientific Programmer, for all the help in LaTeX and also Reinhard and Andrew, two intern-students for all the productive talks and help.

Finally, I want to thank all the New projects people at RIPE NCC for being such a great team to work with.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This graduation diploma project is the result of my trainee-ship carried out at RIPE NCC (Réseaux IP Européens - Network Coordination Centre) between February 2002 and June 2002. The purpose of the project is to investigate routing black holes using the data collected by the RIS project. The work presented here allows further analysis of this interesting issue and can lead to a better definition of the black-holes and their causes.

1.1 About RIPE

RIPE (Réseaux IP Européens) is an open collaborative community of organisations and individuals, operating wide area IP networks in Europe and beyond. The objective of the RIPE community is to ensure the administrative and technical coordination necessary to enable operation of a pan-European IP network. RIPE does not operate a network of its own.

Numerous organisations and individuals participate in the work. The result of the RIPE coordination effort is that an individual end-user is presented with a uniform IP service on his or her desktop irrespective of the particular network his or her workstation is attached to. In May 2000, more than 14.000.000 hosts were reachable via networks coordinated by RIPE.

RIPE has no formal membership and its activities are performed on a voluntary basis, except the activities performed by the RIPE NCC. Most of the work happens inside several working groups. Each of these working groups has one or more mailing lists where relevant topics and questions can be discussed. RIPE Working Groups meet 3 times a year during RIPE meetings. Working groups topics include: routing, IPv6, DNS, RIPE database, anti-spam and several others. [1]

1.2 About RIPE NCC

The RIPE NCC performs activities primarily for the benefit of the membership in Europe, The Middle East, The North of Africa and parts of Asia; mainly activities that its members need to organise as a group, even though they may compete in other areas.

The RIPE Network Coordination Centre (RIPE NCC) is one of 3 Regional Internet Registries (RIR) which exist in the world today, providing allocation and registration services which support the operation of the Internet globally.

The services provided ensure the fair distribution of global Internet resources in the RIPE NCC service region, required for the stable and reliable operation of the Internet. This includes the allocation of Internet (IP) address space, inter-domain routing identifiers (currently BGP autonomous system numbers), and the management of reverse domain name space (currently in-addr.arpa, ip6.arpa and ip6.int).

The RIPE NCC also provides services for the benefit of the Internet community at large including the development and maintenance of the RIPE Database, administrative support for the RIPE community, and the development and co-ordination of new projects.

The RIPE NCC currently supports 3150 Local Internet Registries (LIRs) who collectively form the RIPE NCC membership. Membership is open to anyone using the RIPE NCC services, primarily made up of Internet Services Providers (ISPs). [1]

1.3 About the RIS project

The Research and Development Department at RIPE NCC is called New Projects and its goal is to develop new projects which are useful to the RIPE community and to the Internet community as a whole. One of the projects currently developed is RIS (Routing Information Service).

There are a lot of tools to check issues related to Internet connectivity like ping, traceroute, protocol analysers, router debugging commands, etc. The main drawback of these tools is their lack of historical point of view. All these are working on "the time of use" basis. Another drawback is the one related to mobility (with the exception of Looking Glasses, which are not always publicly available, and they offer information about the AS where they were set up).

The RIS project aims at breaking these frontiers by storing in a giant database, routing information collected in several points throughout the world, in particular in Internet exchanges. The giant database makes it possible for the user to visualise and analyse this data both from a global point of view or from a particular point of view that is interesting for the user.

1.4 Terms used with RIS

1.4.1 AS

Routing throughout an internetwork is similar to routing within a single subnet, but with some added complications. An internetwork can be abstracted as a graph, with its nodes being multi-protocol routers and the connected lines representing connections between those routers. Once the graph has been constructed, known routing algorithms, such as the distance vector and link state algorithms, can be applied to the set of multi-protocol routers. This gives a two-level routing algorithm: within each network an interior gateway protocol is used, but between networks, an exterior gateway protocol is used. In fact, since each network is independent, they may all use different algorithms. Because each network in an internetwork is independent of all the others, it is often referred to as an Autonomous System (AS).

At the network layer, the Internet can be viewed as a collection of subnetworks or ASs that are connected together. There is no real structure, but several backbones exist. These are constructed from high-bandwidth lines and fast routers. Attached to the backbones are regional (mid-level) networks, and attached to these are the LANs at several ISPs, companies and organisations. [2]

An autonomous system contains subnetworks that share the same administration policy.

1.4.2 BGP

BGP stands for Border Gateway Protocol, defined in RFC1771. This protocol allows building of a routing table without loops, between the autonomous systems. Routers within an AS can use other internal gateway protocols to exchange routing information.

BGP uses the transport control protocol TCP (port 179). Two peer routers which are using BGP establish a TCP connection and exchange certain messages that confirm the connection parameters. Afterwards, they will exchange information related to how the packets can be routed to a certain destination network. Mainly, this information is the indication of the complete path that a packet should take to reach a certain destination. Such a path is formed by the list of ASs that need to be crossed in the way to reaching the destination. This information allows building of a cycle free table where routing policies can be applied to force some restrictions concerning inter-domain routing.

Any two routers sharing a TCP connection, on which they exchange BGP information are called neighbour routers or peers. Neighbour routers can initially exchange all the information contain in their BGP tables. Starting from that point, the exchanged information will concern only changes to those tables. This gives BGP an incremental property.

BGP keeps a so called version number of the BGP table. This number has to be the same for all the neighbours of that router keeping a copy of its BGP table. Supplemental packets are sent to verify a certain neighbour state and to inform neighbours of errors or special conditions.

If an AS has more than one router that implements BGP and is peering different ASs, then it can be used as a potential transit AS. It is necessary for an AS to ensure that all the networks within that AS are reachable before sending information related to those networks to another exterior peer. This is accomplished by redistributing information to and from the internal link protocols.

Usually two neighbour routers are directly connected, but in certain special cases, a router can have an established session with an external router, even though it has no direct connection to that router's network. In this case we are talking about multihop BGP.

When a router that supports BGP is receiving a notification concerning a certain destination from another AS, BGP will decide which path to choose to reach a certain destination. The decision is based on several attributes like the next AS in the AS path, local preferences, where is the notification coming from, the AS path length, metrics and others. BGP will always advertise to its neighbours the path that it considers to be the best one. [5] This is a key property of the BGP protocol. When a router is advertising a network to its neighbours we say that it sends an announcement for that network. If the routing information for that network is no longer valid, it sends a withdraw, notifying its peers about this change in its BGP table.

The RIS project collects information related to BGP protocol advertisements.

1.4.3 GNU Zebra

GNU Zebra is a software tool to build a router in software, distributed under the GNU Public License which controls TCPIP based protocols. It supports BGP version 4 and some other protocols like RIPv1, RIPv2, OSPFv2.

Opposite with traditional monolithic architectures (like gated), Zebra offers modularity. The RIS project is using Zebra to collect BGP related information.

1.4.4 RRC

A Remote Route Collector (RRC) is basically a machine that is physically placed in a strategic point (generally a big Internet exchange point). Zebra runs on the RRC and collects routing BGP information. This information is periodically transferred to RIPE NCC, where it is stored both as raw files and as data into a mysql database.

What is very important is that these route collectors are not routing servers. They are only collecting routing information. They do not send any advertisement to the outside world (no announcements and no withdraws). Their purpose is only to listen for BGP data and store it for transfer to RIPE NCC.

1.5 RIS Architecture

Now that we understand the basic terms in the RIS project context, we can start looking at the RIS architecture.

The RIS project relies on a number of remote route collectors. Currently there are 8 route collectors, but this number is slightly increasing. The more collectors the project has, the more data can be collected and so the more complete view over the Internet BGP routing table can be achieved. Table 1.1 shows the 8 RRCs along with their locations.

RRC	Location
rrc00.ripe.net	RIPE NCC, Amsterdam
rrc01.ripe.net	LINX, London
rrc02.ripe.net	SFINX, Paris
rrc03.ripe.net	AMSIX, Amsterdam
rrc04.ripe.net	CIXP, Geneva
rrc05.ripe.net	VIX, Vienna
rrc06.ripe.net	NSPIXP2, Otemachi, Japan
rrc07.ripe.net	Netnod, Stockholm
rrc08.ripe.net	MAE-West San Jose, California, USA

Table 1.1: RRC Locations

Each RRC sends the collected data to the main project server at RIPE NCC in Amsterdam. Here, all the data is published on the web as raw files. Also, an insertion process into a mysql database is taking place, so that data can be more easily queried and analysed. Figure 1.1 shows the project's back-end architecture at RIPE NCC.

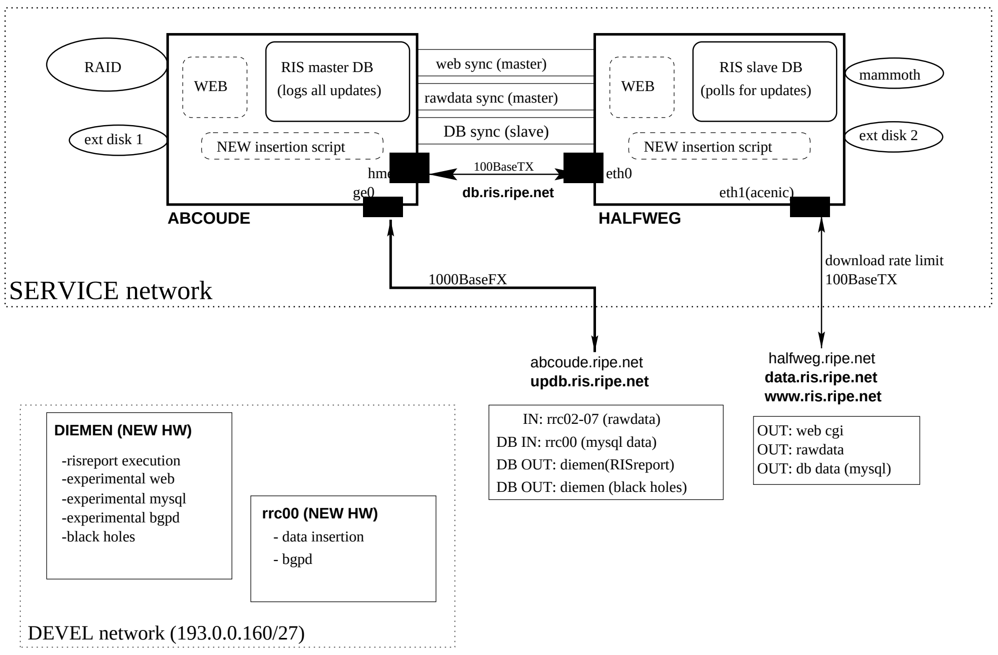

Figure 1.1: RIS back-end architecture at RIPE NCC

The RIS back-end architecture currently relies on 4 dedicated servers. Two of them (Halfweg and Abcoude) are connected in the RIPE NCC's service network and the other two (Diemen and RRC00) are connected to the development network.

The two main servers Halfweg and Abcoude are functioning according to a load balancing scheme, each of them being able to take over the tasks usually performed by the other. RRC00 is, historically speaking, the first remote route collector of the RIS project. It is used as a development and a testing platform for the generic RRC configuration.

The Diemen server is dedicated to services such as RIS Report, analysis, research and development. This is the place where the utilities for the black holes report run and which are further explained in this paper.

Chapter 2

About black holes

2.1 Searching for a definition

From the very beginning I must say that the black holes subject is a research related one. This is mainly the reason why there is no exact, clear and generally adopted definition of a black hole.

A rough definition would be that a black hole is a block of the IP address space for which there is no relevant routing information in a particular routing table, right in the point where that information should have existed. In other words a certain entity is considered to be the destination for that block, but the entity itself does not know how to route that block.

2.2 Some security aspects

Routing is the process by which a packet is send from a location to another over various intelligent elements. Each packet has a source and a destination and so the routing process on each hop host determines the path that a packet has to take to reach its destination. The routing process on the Internet can be divided in two main categories: local routing and wide-area routing. Local routing takes the packet to a host on a certain local network when this packet has reached that network. Wide-area routing is concerned about the inter-network transport.

One of the problems that still needs solving from a practical point of view is the wide-area routing security. Several studies are concentrating on authenticating advertisements that a router sends out. Certain wide-area routing protocols (like BGP) have tried at a certain moment to address these issues in conjunction with Internet routing. Still, some fundamental questions about routing security include the following [6]:

- Does the wide-area routing security reduce to a simple problem of authentication, or it is more a problem related to the protocol itself? In the special case of the wide-are routing, is the threat coming from an attacker who deliberately injects false routes into the Internet infrastructure, or is more an issue that involves building a protocol extension to BGP that would help in case of operation errors?

- Who are those who can produce real threats in such a case, then? Generally speaking, large scale attacks in BGP area involve access to a router connected in an Internet Exchange Point, on a high speed backbone. So, in this case the question is if these entities are vulnerable or we should concentrate more on those who have legitimate access to those routers? Nick Feamster [6] thinks about the fired network administrators of huge Internet Service Providers.

2.3 From BGP to black holes

BGP is used to distribute routing information on the Internet. From BGP perspective, the Internet consists of entities called ASs. Each AS has its own rules when it comes to internal routing. Usually the interior gateway protocols are not suitable for use with the large Internet scale, because they require a complete view of the entire topology or they produce a large amount of traffic because they are sending messages on each output interface. Contrary, BGP has mechanisms which prevent routing cycles among the autonomous systems and it is scalable because of its incremental nature: once the existing routes are synchronised, only advertisements like announcements and withdraws are transferred.

The basic idea is that the Internet is partitioned in administrative domains (ASs) and that each AS announces to the exterior world the set of networks that it knows information about and is willing to share this information with its peers.

For example, in Figure 2.1, AS100 knows how to route the 193.0.0.0/8 network and AS200 knows how to route the 194.0.0.0/8. Therefore, they will send announcements concerning these networks to their peer - AS300. AS300 will then announce these networks to AS400 which it is peering with. So, AS400 will learn that in order to get to 193.0.0.0/8 and 194.0.0.0/8, it first has to transit AS300.

An AS can agree on being a transit AS (like AS300 in our example): if an AS gathers information about how to reach a certain network and if the policy of that particular AS allows it, it will announce that network to

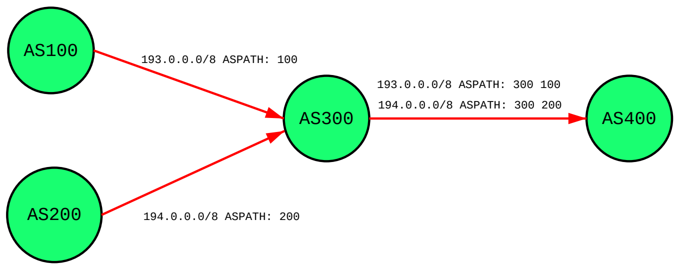

Figure 2.1: BGP announcements example

its external peers and will also allow traffic to that network to be routed thorough itself.

BGP is, in this context, a vector oriented protocol because every announcement includes the ASPATH attribute. This attribute contains the list of ASs that need to be crossed in order to reach the announced network. The ASPATH attribute allow easy detection of loops. Generally, vector oriented protocols scale linearly with the number of nodes (in the BGP case, ASs), while link state oriented protocols scale with the square number of nodes.

The problem with BGP is that is not perfect and we can think of several threats that can come from different sources.

When a certain AS sends an announcement concerning a network that it has no routing information for, we say that it is announcing a black hole. Each peer receives a notification corresponding to that network and the sent packets with that destination will be routed towards the originating AS, where packets will be ignored. Producing a black hole is in this case a very ”efficient” attack from the attacker’s point of view.

2.4 Possible ”uses” of black holes

An example [6] where we can talk about a possible use of black holes and in a non-malicious manner is when an AS announces networks via BGP, and in the next hop attribute it specifies an address where all the traffic to that destination will be ignored. This can be used to diminish the effects of traffic floods for example.

Unfortunately this action can be also used to direct an attack towards a certain network. The one that is planning the attack can create such a black hole by announcing a route to a destination which will ignore the traffic. In this case all the traffic is redirected and the actual network becomes unreachable.

Another type of attack [6] can be conducted in such a way that traffic to a certain destination is redirected by an attacker to a machine that performs the same functionality as one in the real destination network. This allows the attacker to pretend to be someone that is not. Typically this attack can be conducted towards a web site and the consequences can be severe, specially if we are talking about an e-commerce site.

Given all these, we can see an immediate need for a system that would be able to detect this sort of black holes and then analyse them in order to see if their source is an intended attack, a miss-configuration or an administrative action. This will allow appropriate action to be taken in order for the problem to be solved.

Chapter 3

Black holes searching strategy

This project tries to narrow the search for black holes by extracting a set of IP zones where the black holes are most likely to be found. I will explain in the following sections of this chapter how we can use data collected by the RIS project to accomplish this.

But first we should take care of some more terms which can lead to a better understanding of the whole process.

3.1 More terms and explanations

3.1.1 Prefixes

An IP address is a 32-bit number consisting of a network part (N bits) and a host part (32-N bits).

A prefix in the BGP context corresponds to a network address. A network address is a 32-bit number, written usually in dotted decimal notation, accompanied with a value that specifies the number of bits in the 32-bit number which corresponds to the network part. The remaining bits (up to 32) must be 0. For example 1.0.0.0/8 gives information about all the IP addresses from 1.0.0.0 to 1.255.255.255.

3.1.2 More specifics, aggregates

If a certain prefix is completely covered by another prefix, we call the first to be a more specific of the second. For example the prefix 1.1.1.0/24 covers the IP zone 1.1.1.0 to 1.1.1.255. The prefix 1.0.0.0/8 covers the IP zone 1.0.0.0 to 1.255.255.255. As we can clearly see, the first range is completely included in the second one, so we can say that 1.0.0.0/24 is a more specific of 1.0.0.0/8. We also call the 1.0.0.0/8 an aggregate of 1.1.1.0/24.

3.1.3 Prefix aggregation

Prefix aggregation involves expressing 2 or more prefixes by a single prefix. The best way to describe this is by using an example: if we have two prefixes 1.1.1.0/25 and 1.1.1.128/25 the result of the aggregation process will be 1.1.1.0/24. Table 3.1 shows this aggregation process in more detail.

Prefix	Start IP	End IP
1.1.1.0/25	1.1.1.0	1.1.1.127
1.1.1.128/25	1.1.1.128	1.1.1.1.255
1.1.1.0/24 (aggregate)	1.1.1.0	1.1.1.255

Table 3.1: Aggregation process example

Not all the prefixes that have adjacent borders can be aggregated. For example: 1.1.1.128/25 and 1.1.1.2.0/25 share one zone border, but still they cannot be aggregated because more than 1 bit of their network part is different. Also, two prefixes which do not have the same number of network bits cannot be aggregated, even in the case of a common border. For example: 1.1.1.128/25 and 1.1.1.2.0/24.

3.2 RIS dumps

For this study we have used the first 6 RRCs of the RIS project. By combining the 6 views over the Internet we can achieve a single, fairly consistent view of all the prefixes (networks) announced on the Internet.

The RRCs are dumping their BGP table each 8 hours. These dumps are publicly available on the RIS project site. The dump files are in MRT (Multi- Threaded Routing Toolkit) format and basically contain each entry in the BGP table by the time of the dump. Each entry refers to a prefix that the RRC has information about. This information was received by the RRC from a particular AS that it is peering with.

The mrt toolkit contains a tool - route_btoa - which can be used to convert the dump from the binary format to the ASCII format. Table 3.2 shows an example of a BGP dump entry.

If we extract from such an entry, only the announced prefix (i.e. in the case of the example from Table 3.2 - 62.68.0.0/19), we will end up, after processing the whole dump file, with a lot of prefixes. These prefixes combined, give an indication of the IP space covered by the announcements from the entity that created the file.

TIME:	05/01/02 08:38:19
TYPE:	TABLE_DUMP/INET
VIEW:	0
SEQUENCE:	16
PREFIX:	62.68.0.0/19
FROM:	193.203.0.65 AS1273
ORIGINATED:	04/26/02 05:02:27
ORIGIN:	IGP
ASPATH:	1273 6735
NEXT_HOP:	193.203.0.65
MULTI_EXIT_DISC:	100
COMMUNITY:	1273:8000 6735:90
STATUS:	0x1

Table 3.2: BGP dump entry

In our case, if we process a dump file from a certain RRC, this will give us an image over the IP space covered (seen) by that RRC. If we further look also at the FROM field in the BGP entry, we can determine which prefix announcement has been received from which peer.

3.3 The strategy

In searching for black holes we first rely on the information provided by a full feed router. A full feed router is a router that holds the BGP table for virtually all the Internet. RIS's route collectors are peering with such routers.

We have seen earlier how we can obtain a view of the covered Internet space by looking at the prefix announced in each BGP dump entry. If we also look and separate the prefixes that are coming from a peer that gives full feeds, then, ideally, the IP area designated by the full feed would have to be the same as the combined all RRC IP area. But of course this is not happening in the real world. More, the full feed routers show differences among their BGP table.

The question here is where do these differences come from? Some might come from the propagation delay of a certain announcement, some might come from the different filters that are applied by that particular router.

If we do a comparison between the IP space observed on all the RRCs and the space observed by a particular peer that gives full feeds we will end up with a difference. A difference that in theory should not be there, or anyway should be very small (if only propagation delays are involved). But, as we will see, when we will look at the results, this IP space difference is not small, or anyway not that small so that it all comes from propagation delays.

We will further call this space difference - black holes. Although these are not reflecting the exact black holes that I have described earlier, there are reasons to believe that real black holes are contained in this difference. The main two points that we rely on here are on one hand, the ability of a BGP router to filter suspicious announcements and on the other hand the historical character of the RIS.

3.4 Development plan

The development plan consists of the following steps:

- Take all prefixes from all the RRCs
- Remove more specific announcements
- Generate an IP view for all the Internet (RIS perspective)
- Generate an IP view for a peer that gives full feeds
- Compare the two views, extract the difference

This plan is also described in the diagram shown in the Figure 3.1.

3.4.1 Take all prefixes from all the RRCs

This task is achieved by using the dump file storage system of the RIS. Due to the fact that dumps are generated for each RRC at 8 hours intervals, processing should be done each 8 hours, by taking into account the last dump from each RRC. After processing a dump file, duplicates must be removed (it is possible to have more announcements for a certain prefix, with different attributes).

3.4.2 Remove more specific announcements

We have talked earlier about more specifics. In the case of analysing prefixes from parsing the dump files we will end up with a lot of prefixes which have more specifics. To have only one appearance of an IP address in the IP view, we will need to remove those more specifics.

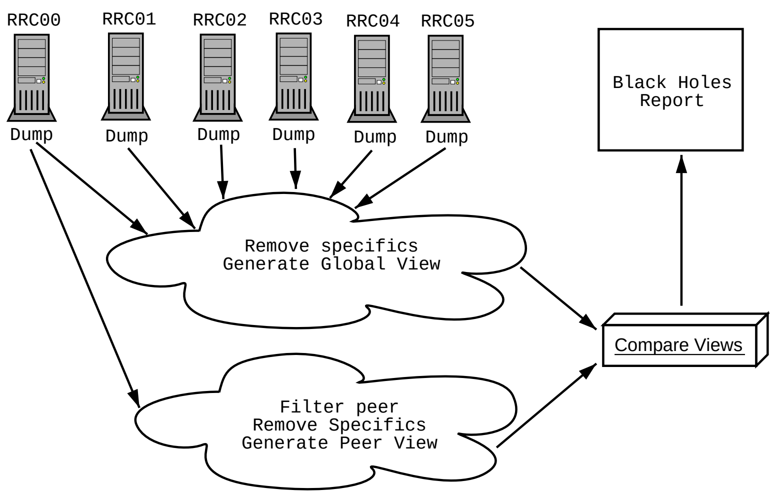

Figure 3.1: Data flow in comparison process

3.4.3 Generate an IP view for all the Internet

After removing specifics we will end up with a list of unique prefixes. In this list, each IP address will appear only once (as a part of certain prefix). This view will allow us to do more statistics which derive from it. How many prefixes are announced? How many prefixes are there after removing specifics? How many are there are after aggregation or after removing specifics and aggregation? Please refer to the following chapters for more explanations on that.

3.4.4 Generate an IP view for a single peer

All the procedures described above can be also carried out for a single peer. The remark is that in the initial step we will need to take care such that only prefixes originated from that peer would follow the remove duplicates and remove specifics chain.

3.4.5 Compare the two views, extract the difference

When comparing the two resulted views - the global view and the view from a full feed we will seek for a difference in IP ranges. An IP range will result for example when we have a prefix in the global view and one of its more specifics in the peer view. In the case of 1.0.0.0/8 (global) and 1.0.0.0/24 (peer) the result will be the range 1.0.1.0 - 1.255.255.255 which can be expressed using prefixes by consequently breaking the initial prefix in smaller prefixes. What you will see in the resulting black holes report will be only IP ranges.

3.5 Displaying results

At the point when we have all this data, the question is how to display it in a form that would be useful and would allow us to have a general view over this issue.

3.5.1 Black Holes

For the black holes the aim was to publish this data on the web, both as IP ranges, and as a graph. The graph would show the black holes distribution over IP space. So, the IP space would have to be represented over the X axis.

3.5.2 Statistics

For the statistics generated along with the black hole tracking process, we have decided to insert this data into rrds - Round Robin Databases. This would allow easy, on the fly, generation of several plots. These statistics can be also used as global status report over the RIS project.

3.6 Comments on aggregation

We have seen what aggregation is, earlier in this chapter. As it can be observed from the development plan, for the black holes we do not use the aggregation technique. This is not needed for the comparison process because after removing specifics each IP address is represented only once in the view that we have over the covered IP space. Still, aggregation is very useful in generating statistics. Aggregating the IP space before or after removing specifics will allow us to measure the global effort of ASs, on one hand, and of registries on the other hand.

With this in mind, we have included in the development plan also the necessary tools that will perform the aggregation process. A remark though, has to be made here: as we have not aimed to gather more information from the dump file, but the prefix announced, when the aggregation is performed, the originating AS is not taken into account. That means that we can only have a global view over a virtual aggregation process that is taking place over all the prefixes announced.

That means that no router can reach the limit of the aggregation process performed by not taking in account the originating AS. We can consider those aggregated prefixes as a lower limit, the limit beyond we cannot cross. This is why these prefixes will be used further in deploying ratios that will show a global effort: either coming from the registries, either from the ASs.

This will be further discussed in the following chapters.

Chapter 4

Tools

To implement the system according to the scheme presented in Chapter 3 several tools were used and others were developed. Table 4.1 shows the tools used to implement the system and Table 4.2 shows the tools developed on the way to tracking black holes.

Tool	Description
route_btoa	Tool to convert dump files from binary format to ASCII format. [12]
Flex	Tool to generate fast lexical analysers. [13]
ROOT	Tool that provides a set of Object Oriented frameworks with all the functionality needed to handle and analyse large amounts of data in an efficient way. [14]
RRDTool	RRDTool is a package that deals with Round Robin Databases - stores, retrieves, produces time graphs. [15]
ps2gif	Tool to convert from Postscript format to gif format. Written by René Wilhelm (`rene@ripe.net`).
mogrify	Tool used for graphics tuning. [16]

Table 4.1: Tools used

The main idea behind the developed tools was not to build a giant tool that will do all the work, but to have small tools for various tasks and to glue them together using bash and perl scripts. All the tools presented in Table 4.2 were developed in C and C++. Tools that were used directly in the development process were Flex for parser code generation and ROOT (more specifically ROOT libraries, which were used to link with the developed tool).

This chapter will discuss these tools, providing short descriptions of the tools used and a more in depth description of the tools developed. We will

Tool	Description
rrcpath	Gets the URL of the latest rrc dump
getprefix	Gets prefixes from a converted ASCII-dump file
rmspecific	Removes more specifics from a sorted prefixes file
aggregate	Aggregates (one step) a file containing prefixes
countip	Counts the number of IP addresses in a file containing prefixes
getprefixgap	Gets the difference in IP ranges by comparing two prefixes files
drawspace	On the fly generation of black holes IP space graph.

Table 4.2: Tools developed

make our tools road map from the process of getting the data from the RIS site to processing it and displaying it.

4.1 rrcpath

The rrcpath tool gets the file name of the latest rrc dump taking as argument a reference point in time.

4.1.1 RIS dumps location and organisation

All RIS dump files can be located at [11]. The data is organised over a time scheme. There are directories for RRCs, years and months. Each month directory contains all the dumps and all the updates collected by a certain RRC for a specific month. Dumps have the following naming format: `bview.<date>.<time>.gz` and the updates `update.<date>.<time>.gz`.

4.1.2 Tool description

The rrcpath tool places itself in a reference point in time and provides the user with the latest dump or update URL path. It can be used in several applications that are using the RIS rawdata. The command line format is the following:

```
rrcpath dump|update <rrc_number> <date.time>
```

Date and time should be expressed in the format yyyymmdd.hhmm. This utility first fetches the directory listing from the RIS web server into a temporary file. After that, it parses the file using code generated by FLEX.

Here is the description file for the FLEX analyser:

```
%{
#include <stdio.h>
#include "flex.h"

int yycode;

%}
%option noyywrap
%option caseless
%option debug
%option yylineno

BVIEW "bview."[0-9]{8}"."[0-9]{4}
UPDATE "updates."[0-9]{8}"."[0-9]{4}

%%
{BVIEW} { yycode = FLEX_BVIEW; return; }
{UPDATE} { yycode = FLEX_UPDATE; return; }
. { }
\n { }
<<EOF>>{ yycode = FLEX_EOF; return; }
%%
```

This helps the generated parser to identify the ”bview” and ”update” patterns. Further analysis over time and date issues are addressed in the C code.

The main problem here is when the dump or update is the month, or even the year before the reference time passed as argument in the command line. In that situation, a new file with the needed directory listing is fetched from the web server. This problem can better be seen with the following example:

```
$ rrcpath dump 01 20010101.0001
http://data.ris.ripe.net/rrc01/2001.12/bview.20011231.2244.gz
```

In the example above we ask rrcpath to give the URL for the latest dump from RRC01 given as time reference point 01/01/2002 at 00:01am. What rrcpath does is that it fetches the directory contents for 01/01/2002 and when it parses the file it doesn’t find any dump previous to the time specified (i.e. 00:01), so it decides to fetch the directory contents for one month before, but because the reference point is in January, it has to go back one year and fetch the directory listing for December 2001. It parses the file and it finds out that the correspond dump file is bview.20011231.2244.gz, so it prints the file's full URL on the standard output.

4.2 route_btoa

Route_btoa is a standard mrt tool which converts a binary dump file into an ASCII format. It is widely used when extracting information from mrt-format file types. This tool is part of the mrt - Multi-Threaded Routing Toolkit.

4.3 getprefix

Getprefix is developed to parse a ASCII converted dump file. It mainly uses a flex generated parser to identify the atoms in the route_btoa generated ASCII file. It can also filter on "per peer" basis. The command line syntax is the following:

```
getprefix [-<as number>] [<file>]
```

If the as number part is missing all the prefixes are extracted. If an as number is specified, only the prefixes learned from that particular AS will appear in the result. If the file is not specified, standard input will be assumed as containing route_btoa generated file.

The definition for the lexical analyser is the following:

```
/* scanner for route_btoa generated ascii files */
%{
#include <stdio.h>
#include "flex.h"

int yycode;

%}
%option noyywrap
%option caseless
%option debug
%option yylineno
```

```
TIMESTAMP time:
PREFIXSTAMP prefix:
PREFIX [0-9]+"."[0-9]+"."[0-9]+"."[0-9]+"/"[0-9]+
FROMSTAMP from:
AS "AS"[0-9]+

%%
{TIMESTAMP} { yycode = FLEX_TIMESTAMP; return; }
{PREFIXSTAMP} { yycode = FLEX_PREFIXSTAMP; return; }
{PREFIX} { yycode = FLEX_PREFIX; return; }
{FROMSTAMP} { yycode = FLEX_FROMSTAMP; return; }
{AS} { yycode = FLEX_AS; return; }
. { }
\n { }
<<EOF>>{ yycode = FLEX_EOF; return; }
%%
```

The C code for getprefix is fairly simple using the flex generated routines:

```
#include <stdio.h>
#include <string.h>
#include "flex.h"

char as[100]="";
char prefix[30];

int main(int argn,char **argv) {
    int i;
    yy_flex_debug = 0;
    yycode        = FLEX_UNKNOWN;
    yyin          = stdin;

    for(i=1;i<argn;i++) {
        if(strncmp(argv[i],"-",1)==0) {
            strcpy(as,argv[i]+1);
        } else {
            yyin = fopen( argv[i], "r" );
            if(yyin==NULL) {
        printf("Unable to open %s\n",argv[i]);
        exit(1);
        }
        }
    }

    yylex();
    while(yycode!=FLEX_EOF) {
        while(yycode!=FLEX_PREFIXSTAMP && yycode!=FLEX_EOF)
          yylex();
        if(yycode == FLEX_PREFIXSTAMP) {
            yylex();
            if(yycode!=FLEX_PREFIX) {
                fprintf(stderr,"Error in source file at
                         line %i Exiting ... \n",
                         yylineno);
                exit(1);
            }
            strcpy(prefix,yytext);
            while(yycode != FLEX_FROMSTAMP) {
                yylex();
                if(yycode==FLEX_TIMESTAMP) {
                    fprintf(stderr,"Error in source file -
                          no origin as for prefix %s at line %i
                          Exiting ...",prefix,yylineno);
                    exit(1);
                }
            }
            while(yycode != FLEX_AS) {
                yylex();
                if(yycode==FLEX_TIMESTAMP) {
                    fprintf(stderr,"Error in source file -
                      no origin as for prefix %s at line %i
                      Exiting ...", prefix,yylineno);
                    exit(1);
                }
            }

            if(strcasecmp(as,yytext)==0 || strlen(as)==0 ) {
                 printf("%s\n",prefix);
                 yylex();
            }
        }
    }
 return(0);
}
```

Getprefix prints all the prefixes announced, according to the given option (all or just on per peer basis) to the standard output. Each prefix is printed on a single line. A typical (partial) output would be:

```
9.141.128.0/24
32.0.0.0/8
53.244.0.0/19
62.13.192.0/19
62.40.128.0/19
62.40.160.0/19
62.40.192.0/19
```

4.4 rmspecific

When starting to develop rmspecific - the tool that removes more specifics prefixes from a file - the following assumption was made: the prefixes file is sorted by each digit in the prefix and mask starting from left to right. This is achieved by a combination of standard Unix tools like awk and sort. The command line syntax is:

```
rmspecific [<sorted prefixes file>]
```

Having the prefixes file sorted in this way, makes it easy for the rmspecific tool to remove specifics in only one file parsing. Sorting the prefixes file in the way described above yields to the same result as the sort over a binary representation of prefixes.

We can easily check if one prefix is included into another. If the prefixes file is sorted ascendly then we can always keep track of a single prefix, called the current prefix. If the next prefix read is included (as IP space) in the current prefix, rmspecific reads the next prefix to be analysed. If the next prefix read is not included it means that the current prefix is a maximum aggregate, so it gets printed to standard output, and the prefix read becomes the current prefix.

Here is the C code of rmspecific:

```
#include <stdio.h>
```

```
#define BUFSIZE 100
#define BYTE    unsigned char

FILE    *in;

typedef struct prefix_struct {
    BYTE addr[4];
    BYTE mask;
    char fulladdress[100];
} prefix;

    void fillprefix(prefix *p,char *str);
    void showprefix(prefix *p);
    void copyprefix(prefix *dest,prefix *s);
    int notincluded(prefix *curent,prefix *above);

int main(int argn, char **argv) {
  prefix above;
  prefix curent;
  char   buffer[BUFSIZE+1];

  if(argn>1) {
    in = fopen( argv[1], "r" );
        if(in==NULL) {
                  printf("Cannot open %s",argv[1]);
                  exit(1);
        }
  } else {
    in = stdin;
  }

  fillprefix(&above,"0.0.0.0/255\n");
  while(!feof(in)) {
        if(fgets(buffer,BUFSIZE,in)==NULL) break;
        fillprefix(&curent,buffer);
        if(notincluded(&curent,&above)) {
          showprefix(&curent);
          copyprefix(&above,&curent);
        }
  }
```

```
}

void fillprefix(prefix *p,char *str) {
  sscanf(str,"%d.%d.%d.%d/%d",
                        &p->addr[0],
                        &p->addr[1],
                        &p->addr[2],
                        &p->addr[3],
                        &p->mask);
  strcpy(p->fulladdress,str);
}

void showprefix(prefix *p) {
  printf("%s",p->fulladdress);
}

void copyprefix(prefix *dest,prefix *source) {
  dest->addr[0] = source->addr[0];
  dest->addr[1] = source->addr[1];
  dest->addr[2] = source->addr[2];
  dest->addr[3] = source->addr[3];
  dest->mask      = source->mask;
  strcpy(dest->fulladdress,source->fulladdress);
}

int notincluded(prefix *curent,prefix *above) {
  int rest;
  int   i;

  if(curent->mask < above->mask) return 1;
  if(curent->mask == above-> mask) return 1;

  // compare first above->mask bits
  i=0;
  for(i=0;i<(above->mask)/8;i++)
  if(curent->addr[i]!=above->addr[i]) return 1;

  rest = above->mask % 8;

  if( (curent->addr[i] >> (8-rest)) !=
          (above->addr[i] >> (8-rest)) )
```

```
                return 1;

  return 0;
}
```

There are 4 functions defined in the program. I will give a short description of each one:

- fillprefix - takes as argument a prefix structure and a string. It processes the string (containing a prefix) and it fills the structure with the corresponding IP digits and mask.
- showprefix - simple function to print a prefix represented as a prefix structure.
- copy prefix - simple function to copy the contents of one prefix structure to another prefix structure.
- notincluded - the function that does the job: it verifies if one prefix is included into another.

The main idea around rmspecific is having the prefix file ordered. This avoids multiple file parsing procedures and makes rmspecific very fast even for fairly large files.

4.5 countip

Countip parses a prefix file and reports the total number of IP addresses seen. It opens the input file, it parses the file, fills a prefix structure using the same fillprefix function as in rmspecific, calculates the number of IP addresses per prefix and adds this number to a global counter. Finally, it prints the global counter to standard output. Here is the abbreviated source code:

```
#include <stdio.h>

#define BUFSIZE 100
#define BYTE    unsigned char

FILE    *in;
unsigned int    count;
unsigned int    pow2(int exp);
```

```
typedef struct prefix_struct {
    BYTE addr[4];
    BYTE mask;
    char fulladdress[100];
} prefix;

    void fillprefix(prefix *p,char *str);

int main(int argn, char **argv) {

    prefix curent;
    char   buffer[BUFSIZE+1];

    if(argn>1) {
        in = fopen( argv[1], "r" );
        if(in==NULL) {
            printf("Cannot open %s",argv[1]);
            exit(1);
        }
    } else {
        in = stdin;
    }

    count = 0;
    while(!feof(in)) {
        if(fgets(buffer,BUFSIZE,in)==NULL) break;
        fillprefix(&curent,buffer);
        count = count + pow2(32-curent.mask);
    }
    printf("%u\n",count);
}

unsigned int pow2(int exp) {
    int result = 1;
    return result << exp;
}
```

4.6 aggregate

The aggregate tool is used to aggregate a sorted prefixes file. Because of its nature, the aggregation cannot be performed in only one step (one parse of the input file). Actually the aggregate tool does only one parsing. It reports then the number of aggregations performed. Based on this, a simple script can do the full aggregation process by repeating the above procedure until the number of aggregations performed is 0. We will further discuss the two aspects in the following subsections.

4.6.1 One step aggregation

This is actually the task of the aggregate tool. It takes a sorted prefixes file as input and it starts parsing it. Whenever it finds two prefixes that can be aggregated together, it will aggregate those two and it will only print the resulted aggregated prefix. The sorted prefixes file helps a lot in this context. If two prefixes are to be aggregated we will find them in consecutive positions. Here follows the C source:

```
#include <stdio.h>

#define BUFSIZE 100
#define BYTE    unsigned char

FILE    *in;

typedef struct prefix_struct {
    BYTE addr[4];
    BYTE mask;
    char fulladdress[100];
} prefix;

    void fillprefix(prefix *p,char *str);
    void showprefix(prefix *p);
    void copyprefix(prefix *dest,prefix *s);
    int canaggregate(prefix *curent,prefix *above);

int aggregate_count;

int main(int argn, char **argv) {
```

```
prefix above;
prefix curent;
char   buffer[BUFSIZE+1];
BYTE   aggreg;

aggregate_count = 0;
if(argn>1) {
    in = fopen( argv[1], "r" );
    if(in==NULL) {
        printf("Cannot open %s",argv[1]);
        exit(1);
    }
} else {
    in = stdin;
}

if(!feof(in)) {
    fgets(buffer,BUFSIZE,in);
    fillprefix(&above,buffer);
}
else exit(0);

if(!feof(in)) {
    fgets(buffer,BUFSIZE,in);
    fillprefix(&curent,buffer);
}
else {
    showprefix(&above);
    exit(0);
}

do {
    aggreg = canaggregate(&curent,&above);
    if(!aggreg) {
        showprefix(&above);
        copyprefix(&above,&curent);
    }
    if(fgets(buffer,BUFSIZE,in)!=NULL);
        fillprefix(&curent,buffer);
} while(!feof(in));
if(aggreg) showprefix(&above);
```

```
    else showprefix(&curent);

    if(aggregate_count==0) return 0;
    else return(1);
}

void fillprefix(prefix *p,char *str) {
    sscanf(str,"%d.%d.%d.%d/%d",
        &p->addr[0],&p->addr[1],
        &p->addr[2],&p->addr[3],
        &p->mask);
    strcpy(p->fulladdress,str);
}

void showprefix(prefix *p) {
    printf("%s",p->fulladdress);
}

void copyprefix(prefix *dest,prefix *source) {
    dest->addr[0] = source->addr[0];
    dest->addr[1] = source->addr[1];
    dest->addr[2] = source->addr[2];
    dest->addr[3] = source->addr[3];
    dest->mask    = source->mask;
    strcpy(dest->fulladdress,source->fulladdress);
}

int canaggregate(prefix *curent,prefix *above) {
    int rest;
    int i;

    if(curent->mask != above-> mask) return 0;

    i=0;
    for(i=0;i<(above->mask-1)/8;i++)
        if(curent->addr[i]!=above->addr[i]) return 0;

    rest = ((above->mask)-1) % 8;
    if( (curent->addr[i] >> (8-rest)) !=
        (above->addr[i] >> (8-rest)) )
            return 0;
    above->mask = above->mask-1;
    sprintf(above->fulladdress,"%d.%d.%d.%d/%d\n",
        above->addr[0],
        above->addr[1],
        above->addr[2],
        above->addr[3],
        above->mask);
    aggregate_count++;

    return 1;
}
```

The function canaggregate is the core function of this tool. It takes two prefixes, and it returns true if they can be aggregated and false if they cannot be. If the prefixes can be aggregated, the aggregated prefix is returned in the prefix structured pointed by the ”above” pointer.

The aggregate tool will print all the prefixes to the standard output. The return value of the tool will show if any aggregation has been performed. If no aggregation has been performed 0 is return, otherwise 1 is returned.

4.6.2 Full aggregation

In order to perform full aggregation over a prefix file, multiple steps (parsings) have to be done. If the aggregate tool returns a 1 value, that means that at least one aggregation was performed in the last step. The parsing process over the file has to be repeated until no more aggregations can be performed. This is achieved by using the following script:

```
#!/bin/bash
. include/paths
. include/timefuncs

until aggregate $1 > $tmps/aggregation00; do
    mv $tmps/aggregation00 $1
done
```

This will ensure that at the end of this script’s execution, the file will be completely aggregated. The only problem with this is the non-deterministic time in which the operation is performed. Execution time may be different, depending on the file contents. Still, tests have shown that this is not really a problem, because the aggregate tool operates quite fast (less than 1 second for 115k prefixes) and so there is no actual consistent drawback in performance because of that. Also, on a regular file originated from an RRC, it takes about 6-7 iterations (calls to aggregate) for the full aggregation process to finish. So, fears that this aggregation process will affect the overall processing time proven to be unfounded.

The aggregation process is not really useful in tracking black holes because, comparing IP space can be done after the removing specifics process, but it helps in generating useful statistics which will be discussed in Chapter 5.

The aggregation tool can be also used wherever someone needs to aggregate prefixes. It is very easy to use, and from what we have seen in production, it is also very fast.

4.7 getprefixgap

Getprefixgap was design to finalise black holes tracking process. This is the tool that produces the file containing the IP ranges representing the potential black holes. It takes as arguments two prefixes files: one reference file and one file to process. The command line syntax is:

```
getprefixgap -<reference file> [<file_to_process>]
```

The output consists of IP ranges, each IP range on one line. These IP ranges represent the IP space that is present in the `<file_to_process>` and is not in the `<reference file>`. One line of output has the following format:

```
<start_ip> <end_ip> <number of ip addresses in the range>
```

To better understand how this works, I will further show a short example. We will have to define the reference file, the file to be processed and then see the output of getprefixgap.

The reference file:

```
1.0.0.0/24
1.0.4.0/24
2.0.0.0/8
```

The file to process:

```
1.0.0.0/16
3.0.0.0/24
```

Running getprefixgap against these two files would yield to the following result:

```
1.0.1.0 1.0.3.255 768
1.0.5.0 1.0.255.255 64256
3.0.0.0 3.0.0.255 256
```

The first two ranges in the result come from the fact that in the file to process we have the prefix 1.0.0.0/16 which is not completely covered in the reference file. As we can see, there are two gaps in that space. The third range comes from the 3.0.0.0/24 prefix which is not present at all in the reference file, so it is converted to an IP range and listed as result.

The number of IP addresses contained in the range is redundant information. This number can be obtained by "subtracting" from the `<end_ip>` the `<start_ip>`. Still, this number is very useful when generating statistics, because having the number already in the file avoids the calculations to be done each time the file is processed.

Here is the getprefixgap's C source code:

```c
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>

#define BUFSIZE 100
#define BYTE    unsigned char
#define IADDR   uint32_t
#define IMASK   unsigned int
#define TRUE    1
#define FALSE   0

typedef struct ipspace_struct {
    IADDR start;
    IADDR end;
} IPSPACE;

    IADDR getstart(IPSPACE ipspace);
    IADDR getend  (IPSPACE ipspace);
    void  setipspace(IPSPACE *result,IADDR start,IADDR end);
    IADDR str2iaddr(char *dotnotation);
    void  str2ipspace(IPSPACE *result,char *dotnotation);
    void  showipspace(IPSPACE ipspace);
    void  showrest_in(void);

FILE *ref=NULL;
FILE *in=NULL;

int main(int argn, char **argv) {
    IPSPACE s_ref;
    IPSPACE s_in;
    IPSPACE temp;

    char buffer[101];

    unsigned int s_ref_start,s_ref_end;
    unsigned int s_in_start,s_in_end;

    int i;
    for(i=1;i<argn;i++) {
        if(strncmp(argv[i],"-",1)==0) {
            ref = fopen(argv[i]+1,"r");
            if(ref==NULL) {
                fprintf(stderr,"Unable to open the reference
                    file. Exiting ...\n");
                exit(1);
            }
        } else {
            in = fopen( argv[i], "r" );
            if(in==NULL) {
                fprintf(stderr,"Unable to open %s\n",argv[i]);
                exit(1);
            }
        }
    }

    if(ref==NULL) {
        fprintf(stderr,"No valid reference file. Exiting. \n");
        exit(1);
    }
    if(in ==NULL) {
        fprintf(stderr,"No valid input file. Exiting. \n");
        exit(1);
    }

    if(fgets(buffer,100,ref)==NULL) {
        printf("Empty reference file. Exiting ...\n");
        exit(1);
    }
    else { str2ipspace(&s_ref,buffer); }

    if(fgets(buffer,100,in) ==NULL) {
        printf("Empty input file. Exiting ...\n");
        exit(1);
    }
    else { str2ipspace(&s_in,buffer); }

    do {
        s_ref_start = getstart(s_ref);
        s_ref_end = getend(s_ref);
        s_in_start  = getstart(s_in);
        s_in_end  = getend(s_in);

        if( s_in_end < s_ref_start) {
            // the in space is bellow the ref space
            // show the in space
            // read more in space
            showipspace(s_in);
            if(fgets(buffer,100,in)!=NULL)
                str2ipspace(&s_in,buffer);
            else exit(0);
        } else
        if( s_in_start > s_ref_end) {
            // the ref space is bellow the in space
            // read more ref space
            if(fgets(buffer,100,ref)!=NULL)
                str2ipspace(&s_ref,buffer);
            else {
                showipspace(s_in);
                showrest_in();
                exit(0);
            }
        } else
        if( s_in_start < s_ref_start ) {
            // the in space has a part bellow the ref space
            // show that space
            // modify in space accordingly
            setipspace(&temp,s_in_start,s_ref_start-1);
            showipspace(temp);
            setipspace(&s_in,s_ref_start,s_in_end);
        } else
        if( s_in_end < s_ref_end &&
            s_in_start >= s_ref_start) {
            // the in space is included in the ref space
            // modify the ref space
            // read more in space
            setipspace(&s_ref,s_in_end+1,s_ref_end);
            if(fgets(buffer,100,in)!=NULL)
                str2ipspace(&s_in,buffer);
            else exit(0);
        } else
        if( s_in_end > s_ref_end &&
            s_in_start >= s_ref_start) {
            // the in space is larger then ref space
            // redimension in space
            setipspace(&s_in,s_ref_end+1,s_in_end);
        } else {
            // the in space is exactly the ref space
            // read in space
            // read ref space
            if(fgets(buffer,100,in)!=NULL)
                str2ipspace(&s_in,buffer);
            else exit(0);
            if(fgets(buffer,100,ref)!=NULL)
                str2ipspace(&s_ref,buffer);
            else { showrest_in(); exit(0); }
        }

    } while(!feof(ref) & !feof(in));

    if(!feof(ref)) {
        showipspace(s_ref);
        while(fgets(buffer,100,ref)!=NULL)
            printf("%s",buffer);
    }

    return(0);
}

IADDR getstart(IPSPACE ipspace) {
    return(ipspace.start);
}

IADDR getend  (IPSPACE ipspace) {
    return(ipspace.end);
}

void setipspace(IPSPACE *result,IADDR start,IADDR end) {
    result->start = start;
    result->end = end;
}

IADDR str2iaddr(char *dotnotation) {
    IADDR result;
    inet_aton(dotnotation,(struct in_addr *)&result);
    return(ntohl(result));
}

void str2ipspace(IPSPACE *result,char *dotnotation) {
    int pos=0;
    int i,mask;
    uint32_t tmp=1;
    char temp[100];

    strcpy(temp,dotnotation);
    while(temp[pos]!='/' && pos<strlen(dotnotation)) pos++;
    if(pos!=strlen(dotnotation)-1) { temp[pos]=0x00; pos++; }
    result->start=str2iaddr(temp);

    mask=atoi(&temp[pos]);

    if(mask==32) { result->end=result->start; }
    else {
        for(i=1;i<32-mask;i++) { tmp = tmp << 1; tmp++; }
        result->end = tmp | result->start;
    }
}

void showipspace(IPSPACE ipspace) {
    IADDR invert_start;
    IADDR invert_end;
    invert_start = htonl(ipspace.start);
    invert_end   = htonl(ipspace.end);
    printf("%s",inet_ntoa(*(struct in_addr *)&invert_start));
    printf(" ");
    printf("%s",inet_ntoa(*(struct in_addr *)&invert_end));
    printf(" ");
    printf("%d",ipspace.end-ipspace.start+1);
    printf("\n");
}

void showrest_in(void) {
    char buffer[101];
    IPSPACE temp;

    while(!feof(in)) {
        if(fgets(buffer,100,in)!=NULL) {
            str2ipspace(&temp,buffer);
            showipspace(temp);
        }
    }
}
```

The first thing to notice in the source code is the definition of a new structure (IPSPACE) that holds the border IPs of an IP range. The IADDR representation for an IP address (in fact a 32 bits numeric value) was chosen, so that the values of this type can be easily compared and so that the data structures are UNIX sockets compatible.

All the processing functions make use of this structure. We will take a look over each defined function and describe its functionality:

- getstart, getend - trivial functions to extract the beginning and ending of an IP region.

- setipspace - trivial function of filling the IPSPACE structure with the corresponding data.
- str2iaddr - function to convert from string IP dotted representation to numeric IADDR representation.
- str2ipspace - function to convert from string representing a prefix to an IPSPACE structure. It makes use of str2iaddr.
- showipspace - function to show an IPSPACE structure by printing the start IP address, the end IP address and the number of IP addresses in between.
- main - this is where all the process is taking place. The two input files are processed and the result is printed on the standard output. The process is described further in this section.
- showrest_in - function to print the remaining prefixes in the file to process in the case that the reference buffer has reach its end of file. This will print the prefixes in IPSPACE output form.

The comparison process considers the two input files sorted. At any time during the comparison process, two IPSPACE structures are holding the information to be compared. The s_ref structure is holding space which originated from the reference file and the s_in structure is holding space originating from the file to process. Initially, each of the structures are holding an IP region corresponding to the first prefix in each file.

At each iteration (see the do { ... } while loop in the main() function) the s_in and s_ref structures are compared and a decision is made concerning either concerning these structures borders, either concerning getting new prefixes from the input files. Here are the situations that can occur:

- All the s_in space is bellow the s_ref space. Getprefixgap will show the s_in space in this case, will read the next prefix from the file to process and will store its borders in the s_in structure.
- The s_ref space is bellow the s_in space. A new reference prefix will be read from the reference file buffer and its borders will be inserted into the s_ref structure.
- A part of s_in space is bellow the beginning of the s_ref space. In that case, getprefixgap will show this ”sub-space” and will modify the s_in structure so that its inferior border will be equal to the inferior border of the s_ref space.

- The s_in space in completely included in the s_ref space. In this case, the s_ref space needs to be modified so that the upper border of the s_in space (+1) becomes the lower border of the s_ref space.
- The s_in has some parts common with the s_ref space, but it goes beyond the s_ref space. The s_in space will be redimensioned so that the lower border will be equal to the s_ref's higher border (+1).
- The s_in space in equal with the s_ref space. The action taken is to read both the reference buffer and the file to process buffer for new prefixes to be inserted as ranges in the s_in and s_ref structure.

The main idea behind the algorithm is to always try to accomplish two tasks:

- read a prefix from the file to process if the prefix read previously from this file is lower (in space) than the prefix read from the reference file
- read a prefix from the reference if the prefix read previously is lower than the prefix read from the file to process.

This gives the algorithm the ability to advance and process both files and be able to extract and show the space present in the file to process and not present in the reference file.

Let's take a look over what happens if end of file is reached for one of the two files. If end of file occurs for the file to process, that means that the process is finished - there is no more space to show, so we can discard all the further contents of the reference file. If end of file occurs for the reference file, it means that all the IP space that follows in the file to process has be shown. That explains the presence of the showrest_in() function, which prints as IP ranges all the prefixes left in the file to process.

4.8 drawspace

Drawspace parses a file generated by getprefixgap and generates a graph showing the read IP ranges in a two coordinates system. On the X axis we have the IP space from 0.0.0.0 to 255.255.255.255. This space is represented in chunks of 65536 IP addresses. That means that one datapoint on the X axis corresponds to a region of 65536 addresses on the global IP space. The Y axis shows the number of IP addresses contained in the ranges read from the input file that are present in a certain 65536 block.

To compute this graph, the ROOT tool was used. Drawspace produces standard Postscript output which has to converted to gif output. To do this the ps2gif and the graphics tuning tools where used.

4.8.1 The ROOT tool

The ROOT system provides a set of OO frameworks with all the functionality needed to handle and analyse large amounts of data in a very efficient way. Having the data defined as a set of objects, specialised storage methods are used to get direct access to the separate attributes of the selected objects, without having to touch the bulk of the data. Included are histograming methods in 1, 2 and 3 dimensions, curve fitting, function evaluation, minimisation, graphics and visualisation classes to allow the easy setup of an analysis system that can query and process the data interactively or in batch mode.

Thanks to the builtin CINT C++ interpreter the command language, the scripting or macro language and the programming language are all C++. The interpreter allows for fast prototyping of the macros since it removes the time consuming compile/link cycle. If more performance is needed the interactively developed macros can be compiled using a C++ compiler.

ROOT is an open system that can be dynamically extended by linking external libraries. This makes ROOT a premier platform on which to build data acquisition, simulation and data analysis systems. [9]

4.8.2 Tool description

Drawspace relies on the ROOT tool to build the generated graph. It mainly uses the ROOT TGraph object, enters data in TGraph structures and then asks the ROOT libraries to build the graph. To be noted here is how easy graphs can be generated using the ROOT analysis tool. Here follows the drawspace's C++ source code:

```
#include <iostream.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>

#ifndef __CINT__
```

```
    #include "TROOT.h"
    #include "TApplication.h"
    #include "TCanvas.h"
    #include "TChain.h"
    #include "TTree.h"
    #include "TFile.h"
    #include "TH1.h"
    #include "TH2.h"
    #include "TF1.h"
    #include "TGraphErrors.h"
    #include "TPaveLabel.h"
    #include "TText.h"
    #include "TLine.h"
    #include "TStyle.h"
    #include "TPostScript.h"
#endif

#define TOTAL_POINTS 65536

class IpAddress {
public:
    UInt_t      ip;

    IpAddress() { ip=0; }
    IpAddress(const IpAddress& copy) { ip=copy.ip; }
    IpAddress& operator= (const IpAddress& copy) {
        ip=copy.ip; return *this;
    } // if you change include if (this==&copy) !

    TString getIpString(void) {
        UInt_t invert_ip;
        invert_ip = htonl(ip);
        TString temp("");
        temp.Append(inet_ntoa(*(struct in_addr *)&invert_ip));
        return temp;
    }

    void ReadIP(TString straddr) {
        inet_aton(straddr.Data(),(struct in_addr *)&ip);
        ip=ntohl(ip);
    }
```

```
};

void InsertRange(IpAddress& start, IpAddress& end,
                                   TGraph& graph);

int main(int argc,char **argv) {
    // ROOT initialisation
    TROOT simple ("RIS", "Plots");
    gROOT->SetBatch();

    filebuf fb_read;
    fb_read.open(argv[1],ios::in);
    istream is(&fb_read);

    TString tstr;
    IpAddress ip_start,ip_end;

    TGraph gr(TOTAL_POINTS);
    gr.SetTitle("Black Holes over IP space");

    for(int i=0;i<TOTAL_POINTS;i++) gr.SetPoint(i,i,0);

    while(!is.eof()) {
        tstr.ReadToDelim(is,' ');
        if(tstr=="") break;
        ip_start.ReadIP(tstr);
        tstr.ReadToDelim(is,' ');
        if(tstr=="") break;
        ip_end.ReadIP(tstr);
        tstr.ReadLine(is);
        if(tstr=="") break;
        InsertRange(ip_start,ip_end,gr);
    }


    TCanvas *c1 = new TCanvas("c1","IP Space",200,10,700,500);
    c1->cd();

    gr.SetFillColor(100);
    gr.Draw("AB");
    TAxis *xa;
    xa = gr.GetXaxis();

    if(xa==NULL) cout << "NULL !!!";

    IpAddress temp;

    xa->Set(5,0,TOTAL_POINTS);
    for(int i=1;i<=5;i++) {
        temp.ip=(UInt_t) xa->GetBinCenter(i)*65536;
        xa->SetBinLabel(i,temp.getIpString());
    }
    xa->LabelsOption("vu");

    gr.Draw("AB");

    c1->Update();
    c1->Print("/dev/stdout");

 return 0;
}

void InsertRange(IpAddress& start, IpAddress& end,
                                   TGraph& graph) {
    Int_t start_point,end_point;
    Double_t x,y;

    start_point=start.ip/65536;
    end_point=end.ip/65536;

    graph.GetPoint(start_point,x,y);

    if(start_point == end_point) graph.SetPoint(
                        start_point,
                        start_point,
                        y+(end.ip-start.ip));
    else {
        graph.SetPoint(start_point,start_point,
                            y+(65536-start.ip%65536));
        for(int i=start_point+1;i<end_point;i++) {
            graph.GetPoint(i,x,y);
            graph.SetPoint(i,i,y+65536);
        }

        graph.GetPoint(end_point,x,y);
        graph.SetPoint(end_point,end_point,y+end.ip%65536);
    }
}
```

The source code is very easy to understand: drawspace reads one by one all the IP ranges from the input file and decides which 65536 blocks these ranges belong to. For each block that has addresses in the read range, a value corresponding to the number of IP addresses in that block is added to the Y value corresponding to that block in the TGraph ROOT object.

Drawspace is used to show the distribution of black holes over the IP space. A simple example of a drawspace generated graph is shown in Figure 4.1.

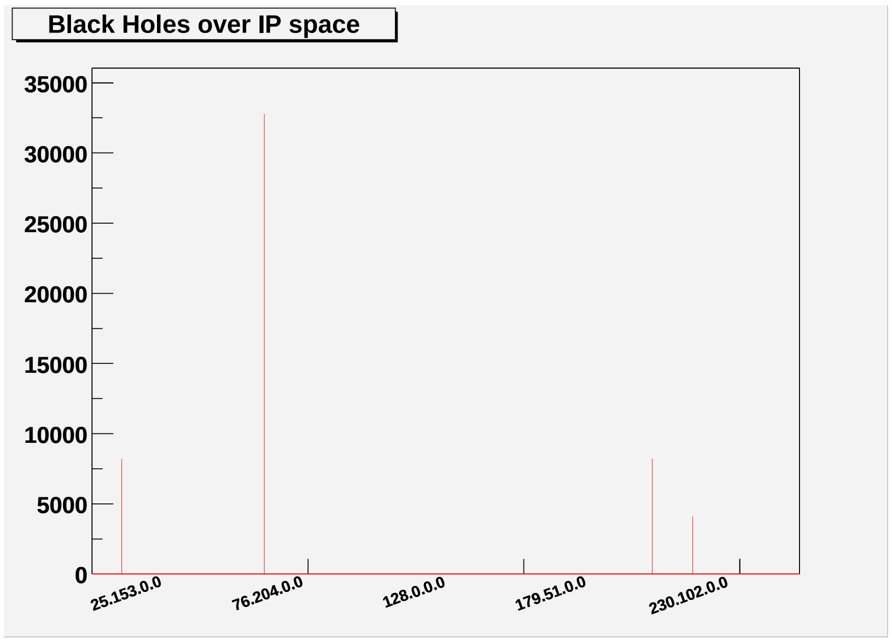

Figure 4.1: Drawspace output example

The graph in the Figure 4.1 was generated by drawspace having as input the following file:

```
#start_ip    #end_ip        #number_of_ips
10.183.0.0   10.183.31.255  8192
62.95.128.0  62.95.255.255  32768
202.183.0.0  202.183.31.255 8192
217.74.192.0 217.74.207.255 4096
```

4.8.3 Graphics format conversion problems

The format produced by the drawspace tool is Postscript. As Postscript can not be published as is on the web, this format has to be converted to a browser displayable format. For the black holes pages ps2gif was used. This tool converts the image from Postscript format to gif format. The problem noticed with this tool was that some lines from the graphs, that appear on the Postscript graph, do not appear in the converted gif image.

The solution to this problem was to enlarge the scale factor of the converted gif, so that the gif output displays all the lines in the Post Script image. That way we have ended up with a large gif. This is why mogrify tool is used to scale the gif to a size suitable for displaying it in a browser. It appears that scaling the gif with mogrify keeps the aspect of the image proportional, without rough information loss.

4.9 Putting it all together

Now that I have described all the tools developed and used in the process, let's see how all these glue up to generate black holes lists and to get data ready for statistics to be available.

The idea is to generate black holes lists by analysing last 8 hours RIS data and store the resulted black holes in a structured way, so that they can be accessed by a CGI script that will show a specific list and will generate a black holes over IP space graph on demand.

Figure 4.2 show the data flow picture of the black holes tracking. This process is repeated each 8 hours thanks to a cron job.

At first, rrcpath is used to get the last dump from each RRC. Each dump is fetched from the RIS web site and passed through route_btoa filter. The result is then passed to getprefix, which will extract all the prefixes which have a corresponding announcement in the dump file. These prefixes will go through a process of sorting and duplicates removal. This is achieved using

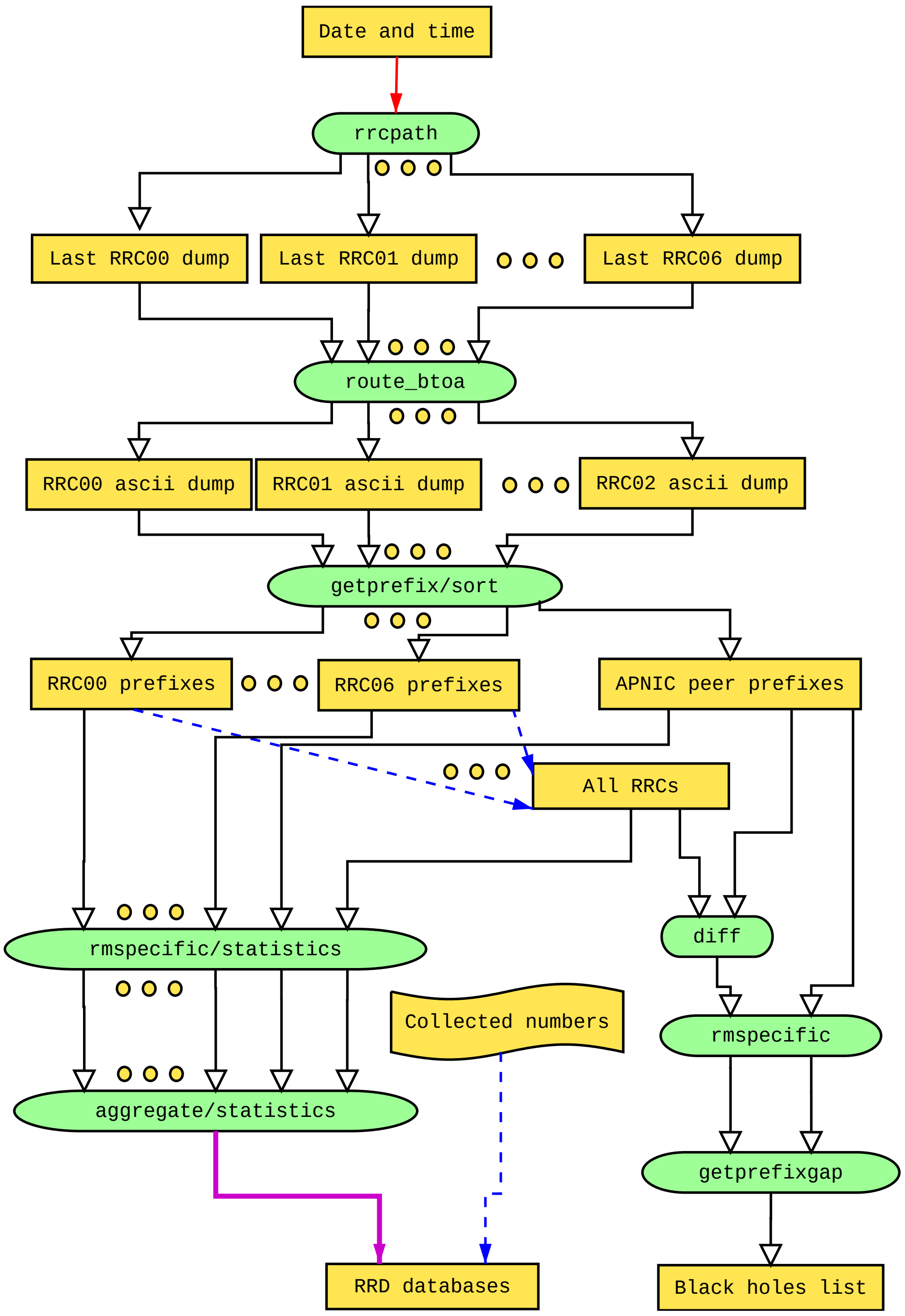

Figure 4.2: Black holes tracking data flow

standard UNIX tools like sort and awk. At the same time prefixes for a certain peer (currently APNIC) are extracted and follow the same process of sorting and duplicates removal. When all dumps are fetched and processed, the files are merged so that a reference point is build, containing all the prefixes announced on all the RRCs. The prefixes from the reference point which are not contained in the peer's view are obtained using the UNIX standard tool - diff. This difference, after the remove specifics process, becomes the file to process for the getprefixgap tool. The reference file for getprefixgap will be the file containing the peer's prefixes, also processed with rmspecific.

The result is a file containing IP ranges which is suitable for drawspace input.

Along all this process certain figures are collected and inserted into RRD databases. Please refer to Chapter 5 for more information on generated statistics. The aggregate tool is a part of this process. It takes the prefixes files generated by rmspecific or prefixes files where duplicates were removed and it aggregates as much as possible (as we have seen earlier in this chapter).

4.10 Displaying results

Displaying results is achieved through several CGIs which offer on demand plots generation. The results can be seen at:

http://www.ris.ripe.net/black-holes

for black holes lists display and:

http://www.ris.ripe.net/analysis

for the generated statistics.

Chapter 5

Generated statistic plots

5.1 Why generate statistic plots?

Plots are very useful when looking to a system as a whole. Of course, it depends on what we actually put in those plots, but if the correct variables are chosen, a plot can provide us with very useful information on how the system behaves, how well it interacts with the outside world, how other external factors influence the system.

In our case, we talk about two systems involved: the RIS and the Internet. Why the Internet? Because the RIS is a fairly good reflection of what happens in routing area over the Internet.

In consequence, having statistics and plots for the RIS can lead to a good image over the RIS operational status, and also to a basic system for monitoring aspects related to Internet routing.

5.2 RIS statistic plots overview

In the process of tracking black holes, a lot of intermediate files are generated (please see Figure 4.2). These files can be used for statistical purposes by counting the number of prefixes in a certain file (which is in fact the actual number of lines in the file) and by counting the number of IP addresses contained in a certain prefixes file. The latter can be achieved by using the countip tool discussed in Chapter 4.

A detailed view of data collection process is presented in Figure 5.1. Five main variables are collected and inserted into RRD databases. This process is done for each dump file collected by each RRC at 8 hours intervals. The actual work is combined with black holes tracking, thus reducing CPU cycles on the processing machine.

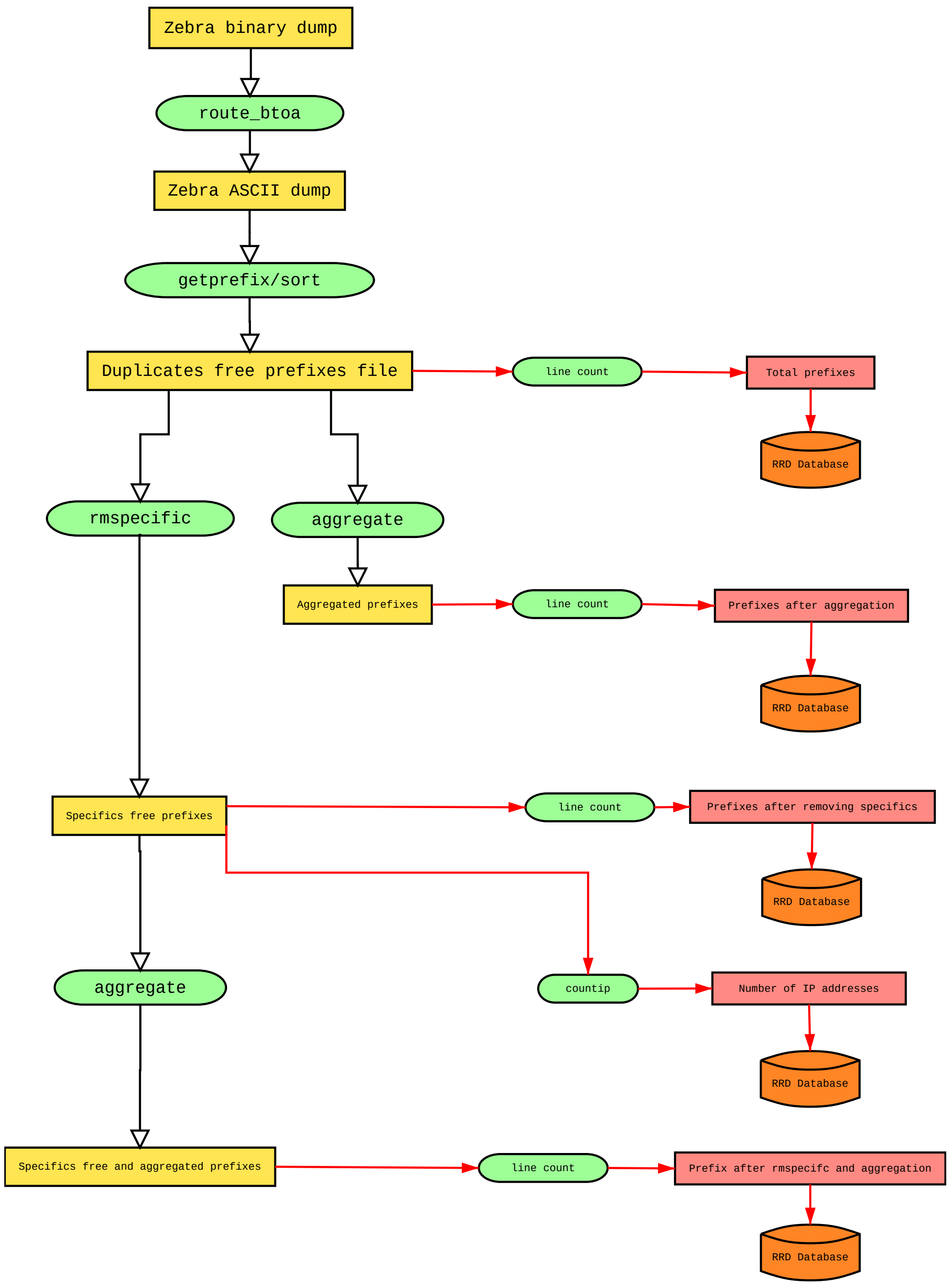

Figure 5.1: Main statistic variables generation process

These variables give a general view over RIS status and also over some trends in the Internet BGP routing area.

5.3 RRD tool

Because we are interested in how the variables develop over time, RRD tool comes very handy in designing the storage place for these variables, but even more, for generating graphs.

RRDTool is written by Tobias Oetiker `<oetiker@ee.ethz.ch>` with contributions from many people all around the world.

RRD refers to Round Robin Database. This is a database technique that works with a fixed amount of data points, and a pointer to the current element. Think of a circle with some dots plotted on the edge, these dots are the places where data can be stored. Draw an arrow from the centre of the circle to one of the dots, this is the pointer.

When data is written, the pointer moves to the next element. As we are on a circle there is no beginning nor an end, you can go on and on. After a while, all the available places will be used and the process automatically reuses old locations. This way, the database will not grow in size and therefore requires no maintenance. RRDTool works with Round Robin Databases (RRDs). It stores and retrieves data from them. [10]

As RRDTool is most suitable for showing variables that develop over time, and since this is exactly what we need in the context of RIS analysis, the plots derived from RIS statistic variables are generated using this tool.

5.4 Variable description

In this section you will find descriptions of the five collected variables, analysis and examples of generated statistic plots.

All plots are over time and data was fed into the RRD databases at 8 hours intervals. A cron job does this, activating automatically at 00:00, 8:00 and 16:00. As you will further notice, we are able to see the development of these variables over a larger period of time (1 year). Special scripts processed data older than late March (when the cron jobs were put in production).

Having the RIS data processed at 8 hours intervals provides a good approximation of BGP table behaviour, specially if we are looking at large trends like 1 year or even several years.

There is also an operational advantage because looking at the graphs, one can identify a sudden drop in number of prefixes received by a certain RRC from its peers and can further investigate the problem, identify its source and notify the parties involved.

Please note that in the following subsections only some of the graphs were presented and analysed. For a complete view please visit:

`http://www.ris.ripe.net/analysis.`

5.4.1 Total prefixes

This is the number of all the prefixes received by a certain entity. Plots are available for each RRC, the APNIC peer and also for all the data collected by the RIS.

Let's take a closer look over these trends. Figure 5.2 shows how this variable developed over the last year for all the RRCs.

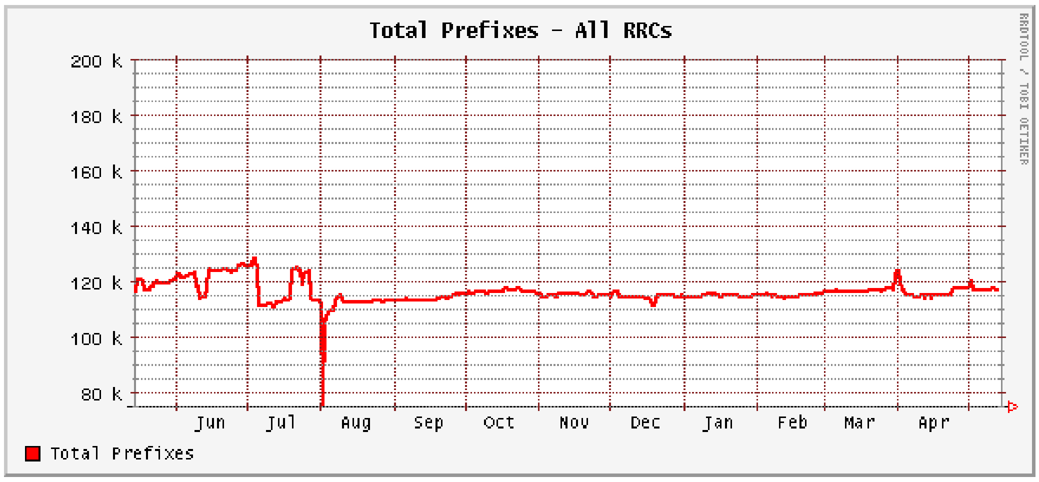

Figure 5.2: Number of total prefixes on all the RRCs
from May 2001 to May 2002

As you can see, during the first two month shown in this plot (June and July) we can notice a slight instability in the number of prefixes observed. The deep short valley that you can see in the beginning of august comes from a certain error in the processing script, which did not take into account the uncompressed files in the rawdata collection. Normally, files in the rawdata collection are compressed, but for the beginning of august some of the dump files for RRC00 were not compressed, resulting in the script assuming data was not there, and therefore the number of prefixes was not accurate for the beginning of august. As this was an isolate event we can focus on analysing the general trend.

Beginning with the month of august, a stability in prefix number evolution can be observed: a very slight increase during September and October and then a decrease in the second half of November. During the next three months, the number of total prefixes was nearly unchanged. Starting with late February and beginning of March, an increase can be observed which leads to a small peak in late March, a stable behaviour during mid April and then an increase in late April which continues in May. Looking at the evolution from August to May we can notice that a slight, but observable grow in the number of prefixes is present.

If we look at May 2001, it started with 120k prefixes announced, and so did May 2002. The overall trend line is still very flat compared to the substantial grow that occurred in the past years (as shown by studies prior to this one).

5.4.2 Prefixes after aggregation

This variable shows the number of prefixes after an aggregation process was performed (without removing more specifics). Figure 5.3 shows the development over time of this variable.

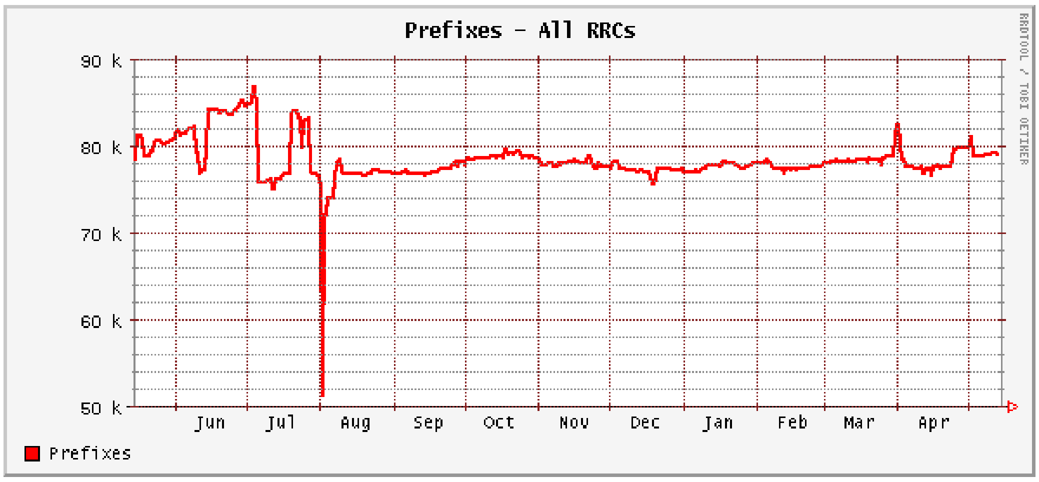

Figure 5.3: Number of aggregated prefixes on all the RRCs from May 2001 to May 2002

What is interesting here is that this graph has approximately the same shape as the graph showing the total number of prefixes (see Figure 5.2). It's only that the upper bound of total prefixes is around 130k, while the upper bound of aggregated prefixes is about 90k.

The shape being the same, means that the variations in number of prefixes do not have much influence over the aggregated prefixes. That means that the variations that we have seen on the total prefixes graph do not come from multi-homing expansion. If this would have been the case, we would notice variations on the total prefixes graph which would have translated in flat lines in the aggregated prefixes graph. That of course, does not mean that multi-homing is not present. The observation here is that it does not influence the variations related to the total number of prefixes.

5.4.3 Prefixes after removing specifics

This variable shows the number of prefixes after a process of more specifics removal. Figure 5.4 shows the RRD generated graph based on this variable.

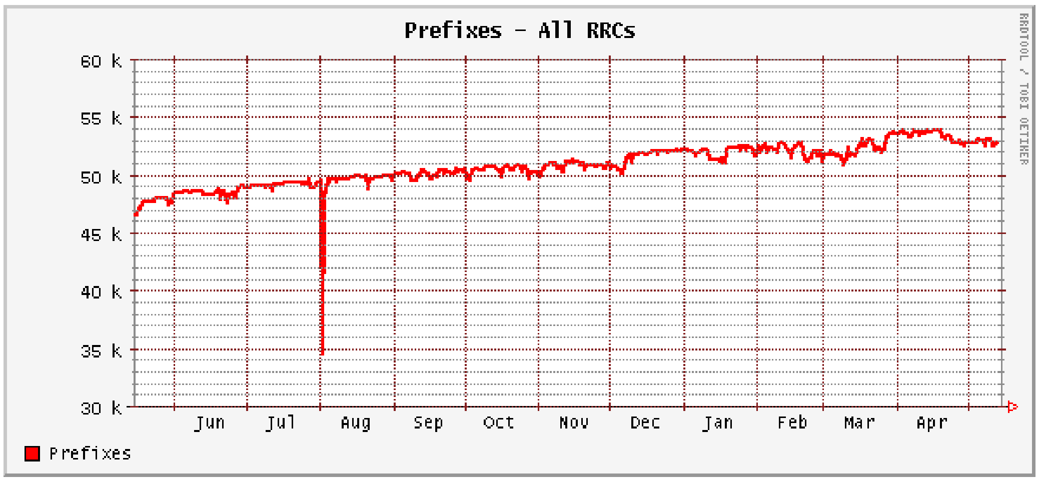

Figure 5.4: Number of prefixes after removing more specifics on all the RRCs from May 2001 to May 2002

If we look at the trend line here, we can clearly see that the number of prefixes is increasing over time. The shape of the graph is quite different from what we have seen before in total prefixes and aggregated prefixes.

The prefixes left after removing specifics are maximum aggregates. If the maximum aggregates increase over time, this behaviour may be the result of more IP space allocations, but causes related to multi-homing can not be excluded. If for a certain reason we would have a prefix that would be slit into several more specifics (due to multi-homing purposes for example) and the actual ”mother” prefix would be no longer announced, this would also lead to an increase in the number of prefixes after removing specifics. To have an idea about this, we would have to compare this trend with the trend in the number of IP addresses announced, i.e. the actual IP space covered.

5.4.4 Number of IP addresses

This variable counts the number of IP addresses actually announced. After removing more specifics, this variable is computed by adding the number of IP addresses contained in every (max aggregate) prefix. Figure 5.5 shows the evolution of this variable over time.

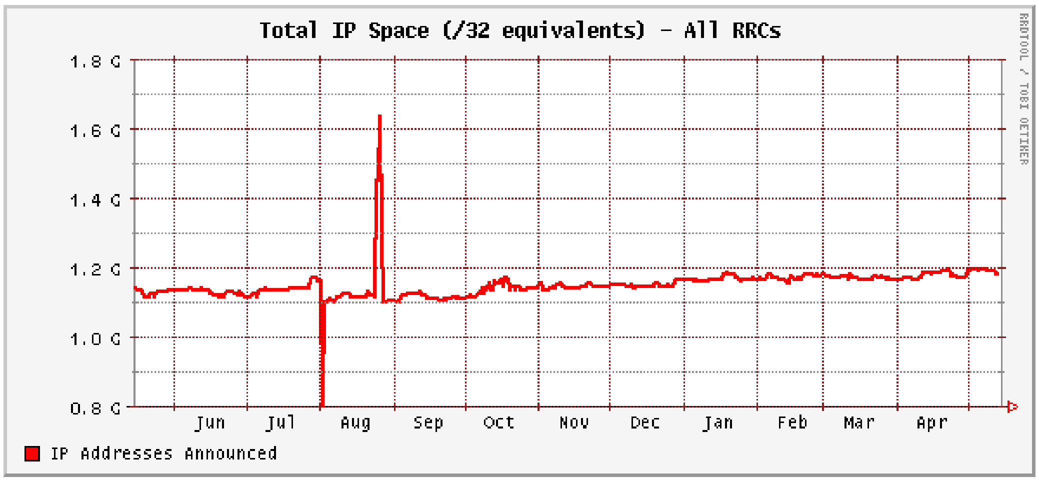

Figure 5.5: Number of IP addresses observed on all the RRCs from May 2001 to May 2002

A slight, but almost constant increase can be observed by looking at the graph line. Of course we may wonder what happened in the beginning of August and what happened in late August, since we have there a deep valley and quite a big peak.

In the beginning of August we have the compressed file situation that I have already explained earlier in this chapter. The peak, tough, is something very interesting. I will explain how I have determined where the peak originated from. I have suspected that this peak would be caused by a bad announcement, an anomaly. Extracting the prefixes from a dump file from 26th of August and sorting them by the prefix's mask, resulted in one /3 being announced. Table 5.1 shows all the details of this prefix.

With the RIS and this statistic plot generation we can actually be able to see when such things like this occur. An action can be taken immediately,

TIME:	08/26/01 16:58:45
TYPE:	TABLE_DUMP/INET
VIEW:	0
SEQUENCE:	8125
PREFIX:	96.0.0.0/3
FROM:	129.250.0.232 AS2914
ORIGINATED:	08/25/01 06:06:07
ORIGIN:	INCOMPLETE
ASPATH:	2914 237
NEXT_HOP:	129.250.0.232
MULTI_EXIT_DISC:	24
COMMUNITY:	2914:420
STATUS:	0x1

Table 5.1: A bogus announcement

or if a certain event happen in the past and we localise it on the graph, then we can use the RIS to track even further this and to identify the cause.

Now that we have seen why we have the huge peak and the deep valley on the graph, we can proceed in analysing the trend line in the graph. We can clearly see an increase trend in the graph.

If in the beginning of June 2001 we had a 1.15G addresses announced, in the beginning of May 2002 the value almost reaches 1.2G IP addresses. This appears to be a healthy grow in address space, which of course comes from the fact that more IP space is allocated by the Internet registries.

Internet Registries allocated during the last year approximately 90M IP addresses and the difference seen on the graph is 50M. This gap comes from the fact that not all space that was requested and allocated is announced.

5.4.5 Prefixes after removing specifics and aggregation

This variable shows the number of prefixes after the process of removing specifics followed by aggregation. Figure 5.6 shows the development over a one year period on this variable.

By looking at this number we can see how contiguous the entire announced space is. If we assume that everything is perfect: Registries are allocating contiguous space and all this space is announced, this number should tend to a very low number. But, of course in the real world there is no such thing as perfection, so this number is looking quite good, actually from this perspective.

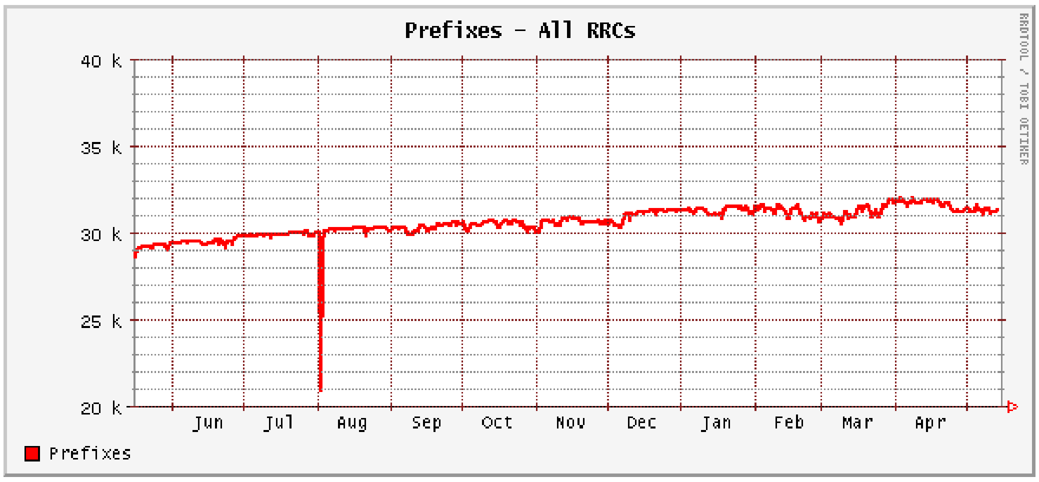

Figure 5.6: Number of prefixes after removing specifics and aggregation from May 2001 to May 2002

This number does not take into account the originating AS, so it shall not be regarded as an indication of the aggregation that can be achieved in a certain AS.

This gives some clues about a collective effort to efficiently allocate and use (announce) IP space. Assuming that the space announced is actually used, this gives an idea about the effort to allocate space that is used with a purpose. To have a better view over these issues, some ratios are presented in the following section.

5.5 Ratios

The variables presented in the previous section are basic variables. These variables were collected in a process of RIS data analysis and their value was stored in RRD databases.

Having these variables computed, we can further analyse them and we can combine them to observe different other trends. In the following subsections three such combinations are presented, but other are also possible.

5.5.1 Total prefixes versus Prefixes after specifics removal and aggregation

This ratio (Figure 5.7) has an decreasing trend. This means that the number of prefixes after specific removal and aggregation grows faster then the total number of prefixes. This could be the result of multi-homing. But this aspect has still to be researched further.

To provide their clients with multi-homing abilities, some ISPs are braking an aggregate, announcing some parts of this aggregate, to give the ability for the client to announce its network also through another provider. This would be a possible explanation for this decrease in this graph's line.

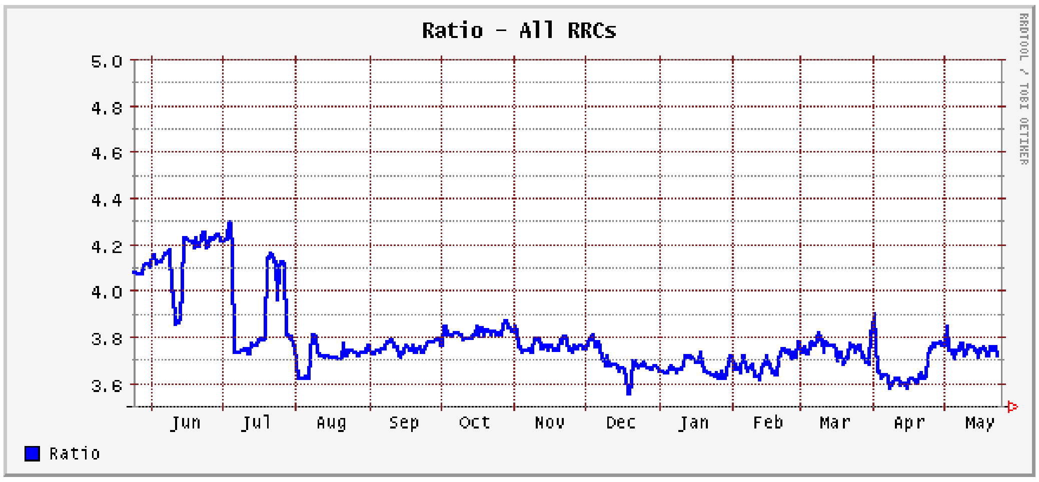

Figure 5.7: Ratio - Total prefixes versus Prefixes after specifics removal and aggregation from May 2001 to May 2002

5.5.2 Prefixes after aggregation versus Prefixes after specifics removal

This ratio (Figure 5.8) shows a clear decreasing overall trend. This ratio shows the efficiency of aggregation by the ASs as a cumulative figure. If this ratio tends to 1 or is decreasing means that the ASs in whole are more efficient in aggregating IP space.

As we have a decreasing trend line, it means that ASs are really good in reorganising their managed address space so that more aggregation is done there. This is a positive thing.

5.5.3 Prefixes after specifics removal versus Prefixes after specifics removal and aggregation

This ratio (Figure 5.9) gives us information about the efficiency of space allocation made to the ASs by the Regional Registries. The graph refers

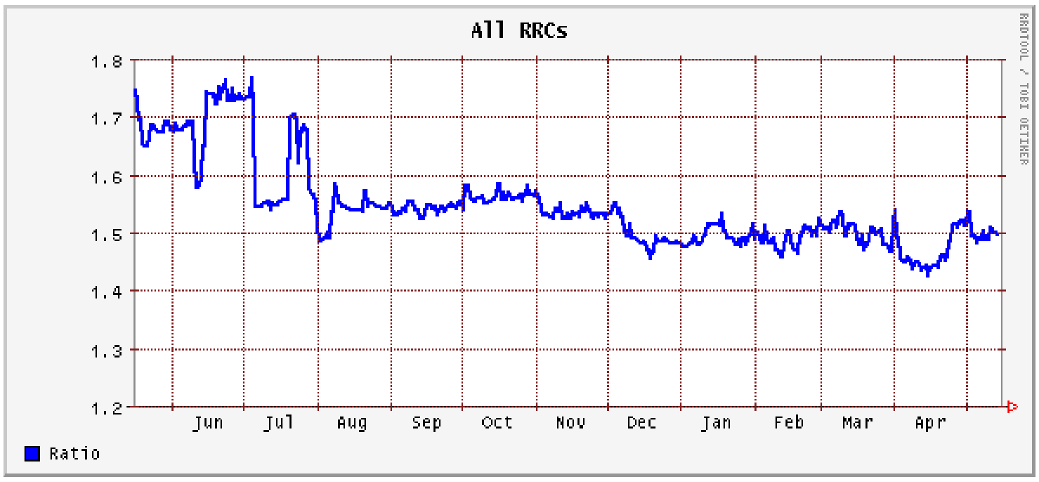

Figure 5.8: Ratio - Prefix after aggregation versus Prefixes after specifics removal from May 2001 to May 2002

only to the space announced. If this ratio tends to be low, it means that the allocated and announced IP space is contiguous and fully announced.

In our case we can see a small increase. This is fairly good behaviour, knowing that registries try to keep up with ever growing demand for IP space. It means that they are keeping up, and more, they are doing a good job, since the increase trend is constant and not having huge hops.

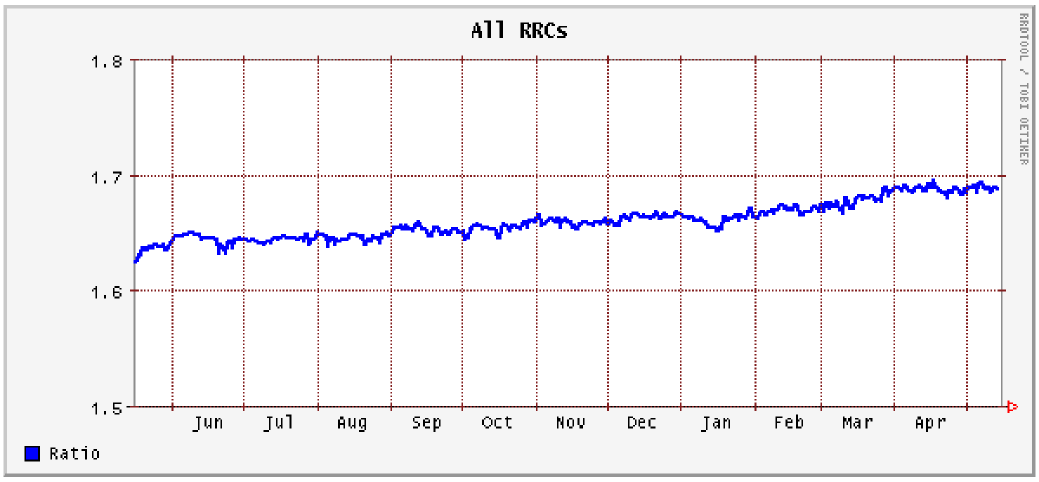

Figure 5.9: Ratio - Prefixes after specifics removal versus Prefixes after specifics removal and aggregation from May 2001 to May 2002

Chapter 6

Results and Perspectives

This chapter shows some examples of the output generated by the black holes tracking system, but also explains what has to be done to improve these results and gives some ideas to go further in researching black holes.

6.1 Black holes tracking status

As we have seen previously, the process of tracking black holes is based on computing the difference in IP space between a reference point (containing all the announcements on all the RRCs) and the space announced by a particular peer.

In this case, I chose APNIC as the peer. The reasons behind this decision were: a peer that gives full feeds was needed and a peer that would advertise as many prefixes as possible was desirable. APNIC gives full feeds, and it was also the peer to announce the biggest number of prefixes to RRC00.

Processed data is available starting from 17/04/2002. Data is computed three times per day at fixed points in time: 00:00, 08:00 and 16:00. Every time that the processing script starts it analyses the last BGP dump from each RRC. The results are available online and one can select a specific date and time for the black holes distribution graph to be generated on demand, starting from the processed IP ranges file. All ranges are also displayed. In the following three subsections we will focus on three examples of black holes lists and the corresponding generated graphs.

A brief overview over the black holes results were included in the presentation over the RIS status, given during the RIPE42 meeting in Amsterdam, the Netherlands. An interesting feedback came from an Internet Exchange operator who was checking up the black holes results during the meeting and actually found an address range announced at his Internet exchange point.

He was aware of the fact that this address was supposed to be a black hole, and should have been filtered by APNIC. This was, in a way, a situation were the black hole was not used maliciously, but rather for operational purposes. This was an interesting feedback and also a "reality check" for the project.

6.1.1 Example 1

First example comes from a file that was generated on 21/04/2001 at 00:00. Figure 6.1 shows the generated graph with the black holes over IP space and Table 6.1 shows some statistics concerning the computed black holes list. The full sequence of IP ranges that generated this graph is shown in Appendix A.

Total space seen (/32 equivalents):	1192203620
Black holes (/32 equivalents):	6324744
Percentile of black holes :	0.53%

Table 6.1: Statistics for Example 1

As you can see from this table, the number of IP addresses that are contained in the black holes space represents 0.53% out of the total IP space announced. This means that we have restrained our search for black holes to just this percentage of the IP space. Still, this does not mean that all the space claim as black holes are black holes. As we have already seen, more research needs to be carry out and to make this list even more narrow. We will also see in the next two examples that this figure is not very typical for a processing result. The typical value is around 0.03% - 0.04%.

If we look at the graph showing how black holes are distributed over the IP space, we can see some high bars around 64.x.x.x, 80.x.x.x, 137.x.x.x, 154.x.x.x and 164.x.x.x. This means that in those areas black holes can be observed that have 65536 addresses in an IP block. The thicker line around 80.x.x.x shows that in that area we have several 65536 IP blocks completely covered with black holes. If we take a look at the full black holes list we can see clearly that in that area we have several IP ranges that go beyond an IP block.

There is an interesting distribution around 207.x.x.x where we have a block covered up to approximately 32000 addresses. Around this point the black holes distribution tends to rise from a low value, increase to this point and then decrease again. Of course, it is very hard to find the "logic" behind such a distribution, but we will see that in this area the distribution seems to be quite stable over time.

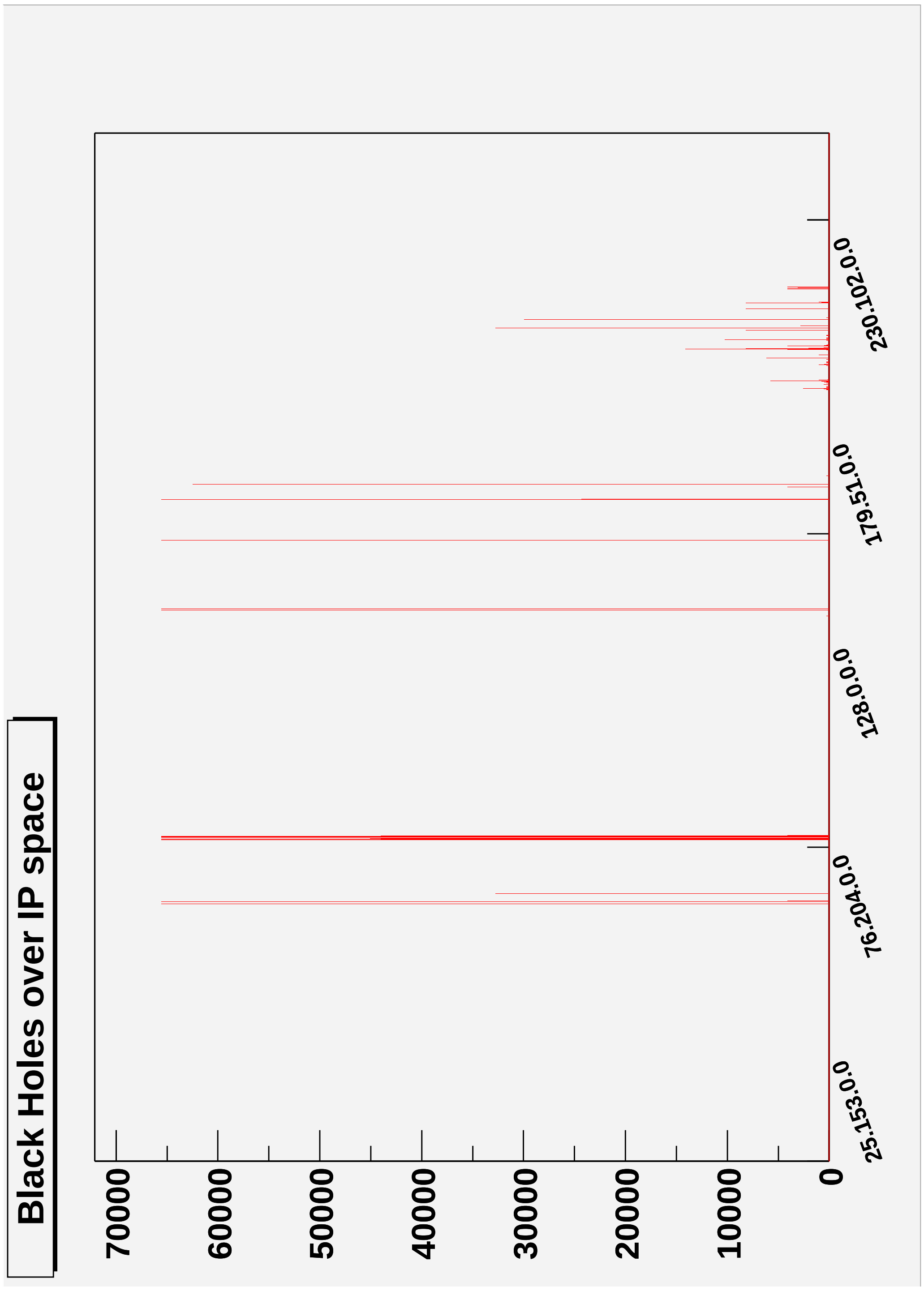

Figure 6.1: Black holes example 1 - 21/04/2002 00:00

6.1.2 Example 2

The second example is the snapshot of black holes processed on 1st of May 2002. Figure 6.2 shows the generated plot. In Table 6.2 you can find the black holes statistics for the list of IP ranges in Appendix B.

Total space seen (/32 equivalents):	1189775924
Black holes (/32 equivalents):	659192
Percentile of black holes :	0.06%

Table 6.2: Statistics for Example 2

The distribution of black holes in the plot is changed from that in the previous example - representing the snapshot taken 10 days before this example's snapshot was taken.

Still, if we look closer we can see that dramatic changes of black holes distribution appear in the first half of the IP space, but in the second half we can see some similarities. Although there are more high bars, meaning more black holes populated space, they are localised around the same areas.

The black holes computing process takes as input the last dump from each of the RRCs. The dumps are taken at 8 hours intervals, the starting point being the time when Zebra was started on each RRC box. This means that the time of the dump is not the same of all the RRC box. The procedure of computing the black holes induces errors related to the difference in time stamps between RRC dumps. Still, a good approximation of black holes behaviour can be observed over an 8 hour interval.

A part of the black holes ”moved” in this 10 days interval (from the timestamp of the first example to the one of the second), but some others were persistent. The area around 207.x.x.x is particularly interesting because, even if some changes can be observed, the shape of black holes distribution is very similar. Where can this come from? This is very hard to tell at first glance, but we can ask the question the other way around: Where this cannot come from?

This behaviour is highly un-probable to be due to the problem of time synchronisation between dumps, or prefixes that are announced and then withdraws in a short period of time, or prefixes that appear in the black holes list due to propagation delay.

We tend to think that this shows that in that area we have to deal with real black holes that tend to be stable in shape. This is very interesting as behaviour.

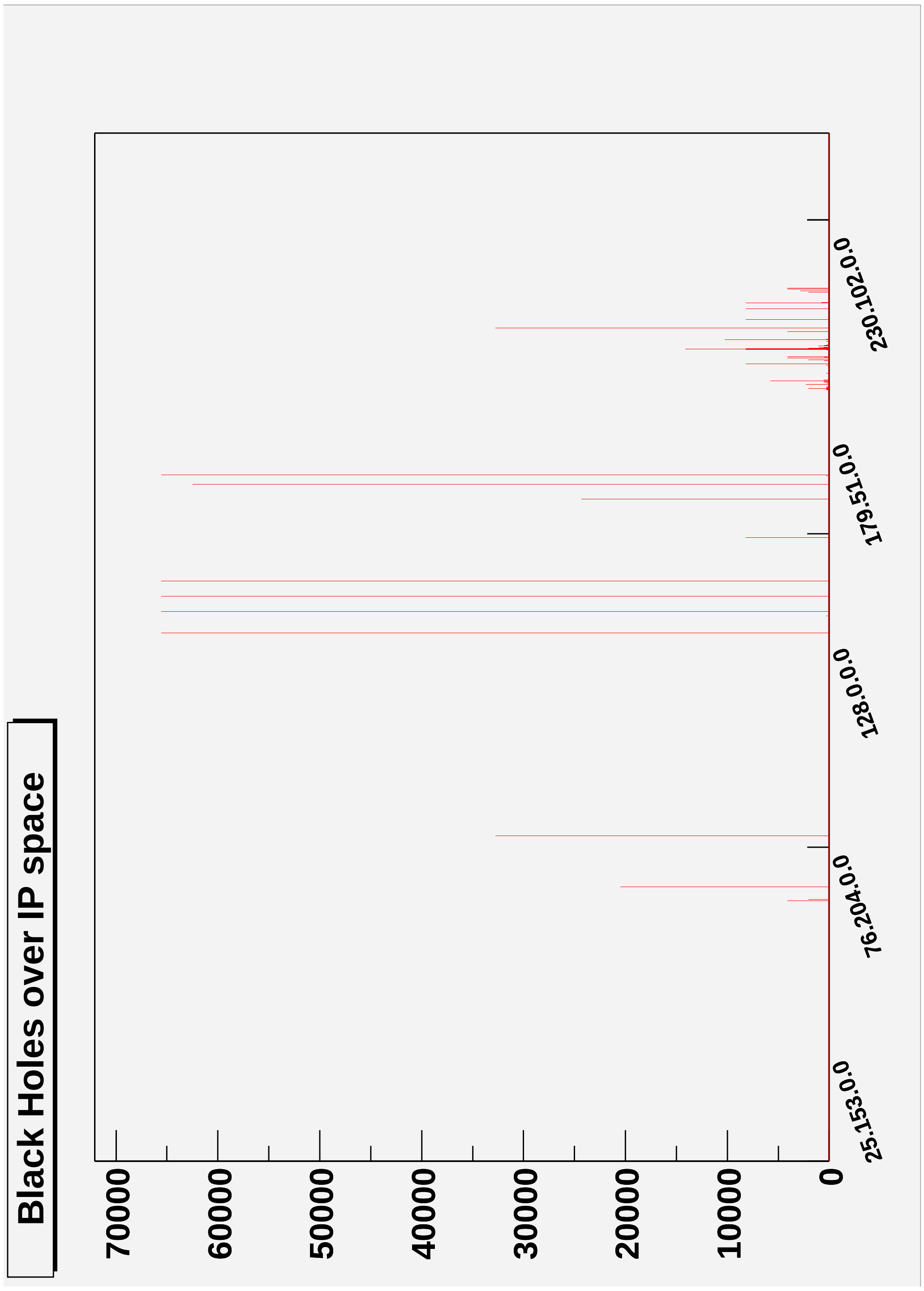

Figure 6.2: Black holes example 2 - 01/05/2002 00:00

6.1.3 Example 3

Figure 6.3 shows another example of black holes on demand generated graph. Table 6.3 shows the statistic for the list of black holes for this example, presented in Appendix C.

Total space seen (/32 equivalents):	1174449176
Black holes (/32 equivalents):	408668
Percentile of black holes :	0.03%

Table 6.3: Statistics for Example 3

As you can see from this table, the black holes percentage over the IP space are in this case 0.03. This is a typical value for the last two month of black hole data. Black hole activity can be observed in the first half of the IP space, also around 147.173.x.x and 158.197.x.x (high bars). In the second half of the IP space, the "shape" that we were talking about in the previous examples is also present.

This example also shows that while some areas of the IP space tend to have high black holes activity, others tend to be "black holes free".

6.2 Perspectives

This project is a research based one. I started along with the RIS team to analyse this problem, and we have managed to narrow the results of the search for black holes. A set of improvements can be added so that these results become even narrower and then one could start looking at the originated prefix and AS of a particular black hole.

Here are some improvements and ideas of how to continue the search for black holes:

- develop a tool to build the BGP table at a certain moment by applying the updates collected by an RRC to a particular dump of that RRC. This will allow a more accurate computation of black holes and even on demand generation, based on virtually a time fully supplied by the user.

- add more peers to be processed. Currently only the APNIC peer is used. This gives the list of black holes only from APNIC's perspective, but adding more peers will allow one to observe if these black holes are the same for another peer, and if not, how much the views differ.

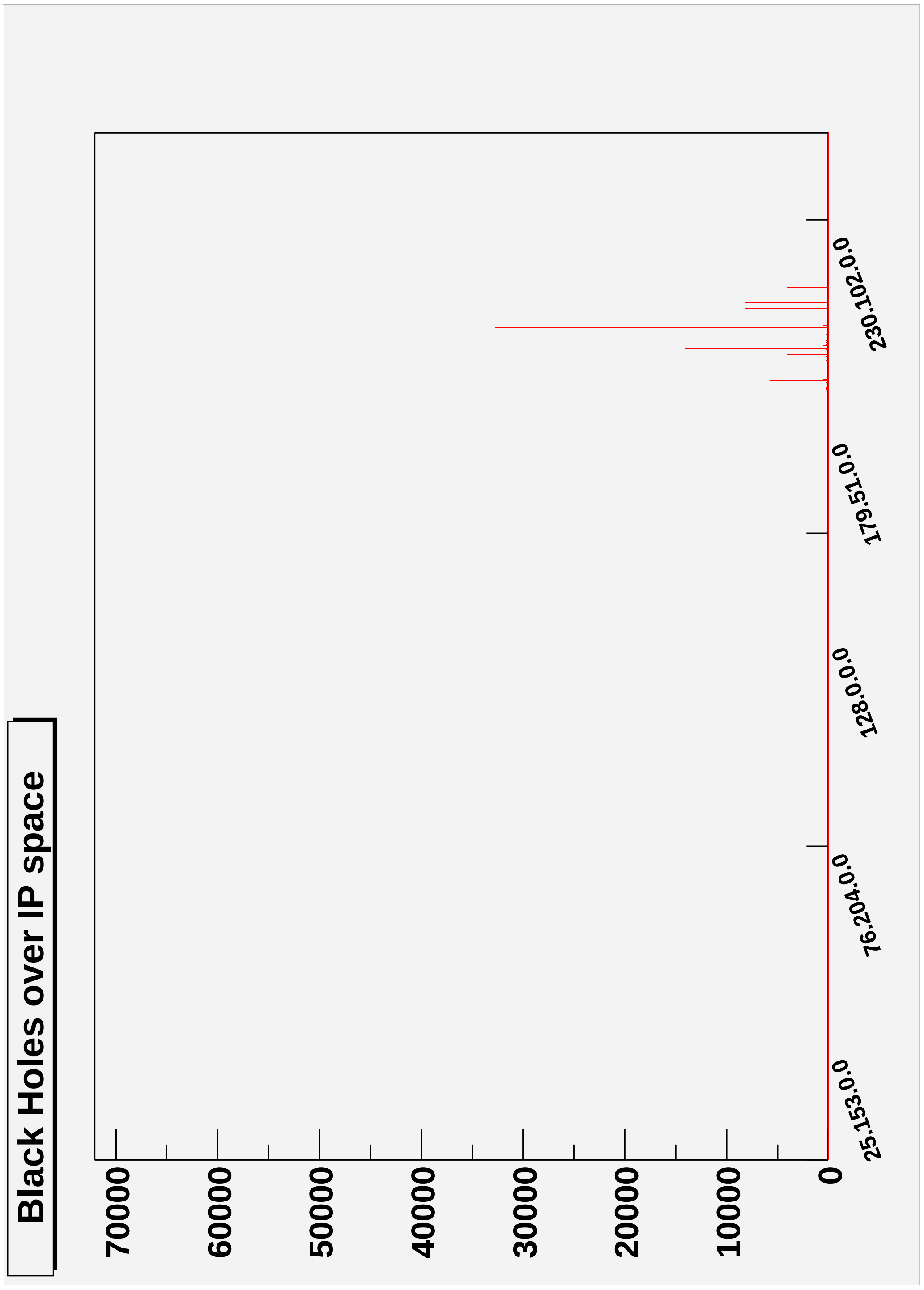

Figure 6.3: Black holes example 3 - 15/05/2002 00:00

- plot black holes percentage over time and determine properties of black holes (origin, life time, etc).
- modify drawspace so that it draws black holes just for a range of the IP space. This would give one's the opportunity to analyse more particular areas of interest of the IP space.
- integrate the tools with the RISReport and use the new database structure rather then the dump files.
- save processed black holes in mysql database tables. This will allow a faster access and a larger spectre of analysis to be done.
- modify getprefix tool (and the others accordingly) so that it gathers more information about a prefix (like the originating AS). This will give the possibility of identifying the AS that actually advertises the prefix that is causing the black holes.

Appendix A

Full black holes list for 21/04/2002 00:00

```
#start_ip      #end_ip         #number_of_ips

61.12.109.96   61.12.109.111   16
64.4.160.0     64.4.160.255    256
64.4.161.0     64.4.161.255    256
64.4.162.0     64.4.162.255    256
64.4.163.0     64.4.163.255    256
64.4.164.0     64.4.164.255    256
64.4.165.0     64.4.165.255    256
64.4.166.0     64.4.166.255    256
64.4.167.0     64.4.167.255    256
64.4.168.0     64.4.168.255    256
64.4.169.0     64.4.169.255    256
64.4.170.0     64.4.170.255    256
64.4.171.0     64.4.171.255    256
64.4.172.0     64.4.172.255    256
64.4.173.0     64.4.173.255    256
64.4.175.0     64.4.175.255    256
64.13.0.0      64.13.255.255   65536
64.149.0.0     64.149.255.255  65536
64.215.96.0    64.215.111.255  4096
66.184.0.0     66.184.127.255  32768
80.10.0.0      80.10.255.255   65536
80.11.248.0    80.11.255.255   2048
80.22.0.0      80.23.255.255   131072
80.26.192.0    80.31.255.255   344064
```

80.36.0.0	80.39.255.255	262144
80.64.16.0	80.64.31.255	4096
80.64.48.0	80.64.63.255	4096
80.64.80.0	80.64.95.255	4096
80.64.112.0	80.64.127.255	4096
80.64.144.0	80.64.191.255	12288
80.64.208.0	80.64.223.255	4096
80.64.240.0	80.65.31.255	12288
80.65.48.0	80.65.63.255	4096
80.65.80.0	80.65.95.255	4096
80.65.144.0	80.65.159.255	4096
80.65.176.0	80.65.191.255	4096
80.65.208.0	80.65.223.255	4096
80.65.240.0	80.65.255.255	4096
80.66.16.0	80.66.31.255	4096
80.66.48.0	80.66.63.255	4096
80.66.80.0	80.66.95.255	4096
80.66.112.0	80.66.127.255	4096
80.66.144.0	80.66.159.255	4096
80.66.176.0	80.66.191.255	4096
80.66.208.0	80.66.223.255	4096
80.66.240.0	80.66.255.255	4096
80.67.16.0	80.67.31.255	4096
80.67.48.0	80.67.63.255	4096
80.67.65.0	80.67.95.255	7936
80.67.112.0	80.67.127.255	4096
80.67.144.0	80.67.159.255	4096
80.67.208.0	80.67.223.255	4096
80.67.240.0	80.67.255.255	4096
80.68.16.0	80.68.31.255	4096
80.68.48.0	80.68.63.255	4096
80.68.66.0	80.68.95.255	7680
80.68.112.0	80.68.127.255	4096
80.68.144.0	80.68.159.255	4096
80.68.176.0	80.68.191.255	4096
80.68.208.0	80.68.223.255	4096
80.69.16.0	80.69.31.255	4096
80.69.48.0	80.69.63.255	4096
80.69.80.0	80.69.95.255	4096
80.69.112.0	80.69.159.255	12288
80.69.176.0	80.69.191.255	4096

80.69.208.0	80.69.223.255	4096
80.69.240.0	80.69.255.255	4096
80.70.16.0	80.70.31.255	4096
80.70.48.0	80.70.63.255	4096
80.70.68.0	80.70.127.255	15360
80.70.144.0	80.70.191.255	12288
80.70.208.0	80.70.223.255	4096
80.70.240.0	80.70.255.255	4096
80.71.16.0	80.71.31.255	4096
80.71.48.0	80.71.63.255	4096
80.71.80.0	80.71.95.255	4096
80.71.112.0	80.71.127.255	4096
80.71.144.0	80.71.159.255	4096
80.71.176.0	80.71.191.255	4096
80.71.208.0	80.71.223.255	4096
80.71.240.0	80.71.255.255	4096
80.72.16.0	80.72.31.255	4096
80.72.48.0	80.72.63.255	4096
80.72.80.0	80.72.95.255	4096
80.72.112.0	80.72.127.255	4096
80.72.144.0	80.72.159.255	4096
80.72.176.0	80.72.191.255	4096
80.72.208.0	80.72.223.255	4096
80.72.240.0	80.72.255.255	4096
80.73.16.0	80.73.31.255	4096
80.73.48.0	80.73.63.255	4096
80.73.80.0	80.73.95.255	4096
80.73.112.0	80.73.127.255	4096
80.73.144.0	80.73.159.255	4096
80.73.176.0	80.73.191.255	4096
80.73.208.0	80.73.223.255	4096
80.73.240.0	80.73.255.255	4096
80.74.16.0	80.74.63.255	12288
80.74.80.0	80.74.95.255	4096
80.74.112.0	80.74.127.255	4096
80.74.144.0	80.74.159.255	4096
80.74.176.0	80.74.191.255	4096
80.74.208.0	80.74.223.255	4096
80.74.227.0	80.74.255.255	7424
80.75.16.0	80.75.63.255	12288
80.75.80.0	80.75.95.255	4096

80.75.112.0	80.75.127.255	4096
80.75.144.0	80.75.159.255	4096
80.75.176.0	80.75.191.255	4096
80.75.208.0	80.75.223.255	4096
80.75.240.0	80.75.255.255	4096
80.76.16.0	80.76.31.255	4096
80.76.42.0	80.76.63.255	5632
80.76.80.0	80.76.95.255	4096
80.76.112.0	80.76.127.255	4096
80.76.144.0	80.76.159.255	4096
80.76.176.0	80.76.191.255	4096
80.76.208.0	80.76.223.255	4096
80.76.240.0	80.76.255.255	4096
80.77.16.0	80.77.31.255	4096
80.77.48.0	80.77.63.255	4096
80.77.80.0	80.77.95.255	4096
80.77.112.0	80.77.127.255	4096
80.77.144.0	80.77.191.255	12288
80.77.208.0	80.77.223.255	4096
80.77.240.0	80.77.255.255	4096
80.78.1.0	80.78.31.255	7936
80.78.48.0	80.78.63.255	4096
80.78.80.0	80.78.95.255	4096
80.78.144.0	80.78.159.255	4096
80.78.176.0	80.78.191.255	4096
80.78.208.0	80.78.223.255	4096
80.79.16.0	80.79.31.255	4096
80.79.48.0	80.79.63.255	4096
80.79.80.0	80.79.95.255	4096
80.79.112.0	80.79.127.255	4096
80.79.129.0	80.79.159.255	7936
80.79.176.0	80.79.191.255	4096
80.79.208.0	80.79.223.255	4096
80.79.240.0	80.79.255.255	4096
80.80.16.0	80.80.95.255	20480
80.80.144.0	80.80.159.255	4096
80.80.176.0	80.80.223.255	12288
80.80.240.0	80.80.255.255	4096
80.81.16.0	80.81.31.255	4096
80.81.48.0	80.81.63.255	4096
80.81.80.0	80.81.95.255	4096

80.81.144.0	80.81.159.255	4096
80.81.176.0	80.81.191.255	4096
80.81.208.0	80.81.223.255	4096
80.81.240.0	80.81.255.255	4096
80.82.16.0	80.82.31.255	4096
80.82.48.0	80.82.63.255	4096
80.82.80.0	80.82.95.255	4096
80.82.112.0	80.82.127.255	4096
80.82.144.0	80.82.159.255	4096
80.82.208.0	80.82.223.255	4096
80.82.228.0	80.82.255.255	7168
80.83.16.0	80.83.31.255	4096
80.83.48.0	80.83.63.255	4096
80.83.80.0	80.83.127.255	12288
80.83.144.0	80.83.159.255	4096
80.83.176.0	80.83.191.255	4096
80.83.208.0	80.83.255.255	12288
80.84.16.0	80.84.31.255	4096
80.84.48.0	80.84.63.255	4096
80.84.80.0	80.84.95.255	4096
80.84.112.0	80.84.127.255	4096
80.84.144.0	80.84.159.255	4096
80.84.176.0	80.84.191.255	4096
80.84.208.0	80.84.223.255	4096
80.85.16.0	80.85.31.255	4096
80.85.48.0	80.85.63.255	4096
80.85.80.0	80.85.95.255	4096
80.85.112.0	80.85.127.255	4096
80.85.144.0	80.85.159.255	4096
80.85.176.0	80.85.191.255	4096
80.85.208.0	80.85.223.255	4096
80.85.240.0	80.85.255.255	4096
80.86.16.0	80.86.31.255	4096
80.86.48.0	80.86.63.255	4096
80.86.80.0	80.86.95.255	4096
80.86.112.0	80.86.127.255	4096
80.86.144.0	80.86.159.255	4096
80.86.176.0	80.86.191.255	4096
80.86.208.0	80.86.223.255	4096
80.86.240.0	80.86.255.255	4096
80.87.16.0	80.87.31.255	4096

80.87.48.0	80.87.63.255	4096
80.87.80.0	80.87.127.255	12288
80.87.144.0	80.87.159.255	4096
80.87.176.0	80.87.191.255	4096
80.87.208.0	80.87.223.255	4096
80.87.240.0	80.87.255.255	4096
80.88.16.0	80.88.31.255	4096
80.88.48.0	80.88.95.255	12288
80.88.144.0	80.88.159.255	4096
80.88.176.0	80.88.191.255	4096
80.88.208.0	80.88.223.255	4096
80.88.240.0	80.89.31.255	12288
80.89.48.0	80.89.63.255	4096
80.89.80.0	80.89.95.255	4096
80.89.98.0	80.89.127.255	7680
80.89.176.0	80.89.191.255	4096
80.89.208.0	80.89.223.255	4096
80.89.240.0	80.89.255.255	4096
80.90.16.0	80.90.31.255	4096
80.90.48.0	80.90.63.255	4096
80.90.80.0	80.90.127.255	12288
80.90.144.0	80.90.159.255	4096
80.90.176.0	80.90.191.255	4096
80.90.240.0	80.90.255.255	4096
80.91.16.0	80.91.31.255	4096
80.91.48.0	80.91.63.255	4096
80.91.80.0	80.91.95.255	4096
80.91.112.0	80.91.159.255	12288
80.91.176.0	80.91.191.255	4096
80.91.208.0	80.91.223.255	4096
80.91.240.0	80.91.255.255	4096
80.92.16.0	80.92.63.255	12288
80.92.80.0	80.92.159.255	20480
80.92.176.0	80.92.191.255	4096
80.92.208.0	80.92.223.255	4096
80.92.240.0	80.92.255.255	4096
80.93.16.0	80.93.31.255	4096
80.93.48.0	80.93.63.255	4096
80.93.80.0	80.93.95.255	4096
80.93.112.0	80.93.127.255	4096
80.93.144.0	80.93.159.255	4096

80.93.176.0	80.93.191.255	4096
80.93.208.0	80.93.223.255	4096
80.93.240.0	80.93.255.255	4096
80.94.16.0	80.94.31.255	4096
80.94.48.0	80.94.63.255	4096
80.94.80.0	80.94.95.255	4096
80.94.112.0	80.94.159.255	12288
80.94.176.0	80.94.191.255	4096
80.94.194.0	80.94.223.255	7680
80.94.240.0	80.94.255.255	4096
80.95.16.0	80.95.31.255	4096
80.95.48.0	80.95.63.255	4096
80.95.80.0	80.95.95.255	4096
80.95.112.0	80.95.127.255	4096
80.95.144.0	80.95.159.255	4096
80.95.176.0	80.95.191.255	4096
80.95.208.0	80.95.223.255	4096
80.95.240.0	80.96.0.255	4352
80.96.2.0	80.96.2.255	256
80.96.5.0	80.96.5.255	256
80.96.11.0	80.96.11.255	256
80.96.15.0	80.96.15.255	256
80.96.24.0	80.96.24.255	256
80.96.27.0	80.96.27.255	256
80.96.32.0	80.96.47.255	4096
80.96.101.0	80.96.101.255	256
80.96.103.0	80.96.103.255	256
80.96.105.0	80.96.110.255	1536
80.96.118.0	80.96.119.255	512
80.96.124.0	80.96.127.255	1024
80.96.129.0	80.96.130.255	512
80.96.132.0	80.96.133.255	512
80.96.135.0	80.96.135.255	256
80.96.138.0	80.96.140.255	768
80.96.144.0	80.96.147.255	1024
80.96.152.0	80.96.153.255	512
80.96.156.0	80.96.160.255	1280
80.96.162.0	80.96.162.255	256
80.96.167.0	80.96.169.255	768
80.96.172.0	80.96.172.255	256
80.96.175.0	80.96.175.255	256

80.96.177.0	80.96.177.255	256
80.96.188.0	80.96.188.255	256
80.96.192.0	80.96.192.255	256
80.96.195.0	80.96.195.255	256
80.96.202.0	80.96.203.255	512
80.96.206.0	80.96.206.255	256
80.96.208.0	80.96.210.255	768
80.96.214.0	80.96.215.255	512
80.96.220.0	80.96.220.255	256
80.96.232.0	80.96.232.255	256
80.96.234.0	80.96.235.255	512
80.96.251.0	80.96.251.255	256
80.97.0.0	80.97.5.255	1536
80.97.7.0	80.97.14.255	2048
80.97.112.0	80.97.191.255	20480
80.106.0.0	80.107.255.255	131072
80.110.0.0	80.110.255.255	65536
80.111.128.0	80.111.255.255	32768
80.124.0.0	80.125.255.255	131072
80.160.0.0	80.191.255.255	2097152
80.214.0.0	80.219.255.255	393216
80.227.48.0	80.227.95.255	12288
80.227.112.0	80.227.127.255	4096
80.227.160.0	80.227.255.255	24576
80.228.128.0	80.229.255.255	98304
80.234.32.0	80.234.127.255	24576
80.239.144.0	80.239.159.255	4096
80.239.176.0	80.239.191.255	4096
80.239.208.0	80.239.255.255	12288
80.240.16.0	80.240.63.255	12288
80.240.80.0	80.240.95.255	4096
80.240.112.0	80.240.127.255	4096
80.240.144.0	80.240.159.255	4096
80.240.176.0	80.240.191.255	4096
80.240.208.0	80.240.223.255	4096
80.240.240.0	80.240.255.255	4096
80.241.16.0	80.241.31.255	4096
80.241.48.0	80.241.63.255	4096
80.241.80.0	80.241.95.255	4096
80.241.112.0	80.241.127.255	4096
80.241.144.0	80.241.159.255	4096

80.241.176.0	80.241.191.255	4096
80.241.208.0	80.241.223.255	4096
80.241.240.0	80.242.0.255	4352
80.242.9.0	80.242.9.255	256
80.242.12.0	80.242.12.255	256
80.242.14.0	80.242.31.255	4608
80.242.48.0	80.242.63.255	4096
80.242.80.0	80.242.95.255	4096
80.242.112.0	80.242.127.255	4096
80.242.142.0	80.242.159.255	4608
80.242.176.0	80.242.191.255	4096
80.242.208.0	80.242.223.255	4096
80.242.240.0	80.242.255.255	4096
80.243.16.0	80.243.31.255	4096
80.243.48.0	80.243.63.255	4096
80.243.80.0	80.243.95.255	4096
80.243.112.0	80.243.127.255	4096
80.243.136.0	80.243.159.255	6144
80.243.176.0	80.243.191.255	4096
80.243.208.0	80.243.223.255	4096
80.243.240.0	80.243.255.255	4096
80.244.1.0	80.244.31.255	7936
80.244.48.0	80.244.63.255	4096
80.244.80.0	80.244.95.255	4096
80.244.112.0	80.244.127.255	4096
80.244.144.0	80.244.159.255	4096
80.244.176.0	80.244.191.255	4096
80.244.208.0	80.244.223.255	4096
80.244.225.0	80.244.255.255	7936
80.245.16.0	80.245.31.255	4096
80.245.48.0	80.245.63.255	4096
80.245.80.0	80.245.95.255	4096
80.245.112.0	80.245.127.255	4096
80.245.144.0	80.245.159.255	4096
80.245.176.0	80.245.191.255	4096
80.245.208.0	80.245.223.255	4096
80.245.240.0	80.245.255.255	4096
80.246.16.0	80.246.31.255	4096
80.246.48.0	80.246.63.255	4096
80.246.80.0	80.246.95.255	4096
80.246.112.0	80.246.127.255	4096

80.246.144.0	80.246.159.255	4096
80.246.176.0	80.246.191.255	4096
80.246.208.0	80.246.223.255	4096
80.246.240.0	80.246.255.255	4096
80.247.16.0	80.247.31.255	4096
80.247.48.0	80.247.63.255	4096
80.247.80.0	80.247.95.255	4096
80.247.112.0	80.247.127.255	4096
80.247.160.0	80.247.191.255	8192
80.247.213.0	80.247.223.255	2816
80.247.240.0	80.248.31.255	12288
80.248.36.0	80.248.63.255	7168
80.248.80.0	80.248.95.255	4096
80.248.128.0	80.248.159.255	8192
80.248.176.0	80.248.191.255	4096
80.248.208.0	80.248.255.255	12288
80.249.16.0	80.249.31.255	4096
80.249.48.0	80.249.63.255	4096
80.249.80.0	80.249.95.255	4096
80.249.112.0	80.249.127.255	4096
80.249.144.0	80.249.159.255	4096
80.249.176.0	80.249.191.255	4096
80.249.194.0	80.249.223.255	7680
80.249.240.0	80.249.255.255	4096
80.250.16.0	80.250.31.255	4096
80.250.48.0	80.250.63.255	4096
80.250.80.0	80.250.95.255	4096
80.250.112.0	80.250.127.255	4096
80.250.144.0	80.250.159.255	4096
80.250.176.0	80.250.191.255	4096
80.250.208.0	80.250.255.255	12288
80.251.16.0	80.251.31.255	4096
80.251.48.0	80.251.63.255	4096
80.251.80.0	80.251.95.255	4096
80.251.112.0	80.251.127.255	4096
80.251.144.0	80.251.159.255	4096
80.251.176.0	80.251.223.255	12288
80.251.240.0	80.251.255.255	4096
80.252.16.0	80.252.31.255	4096
80.252.48.0	80.252.63.255	4096
80.252.80.0	80.252.95.255	4096

80.252.112.0	80.252.127.255	4096
80.252.144.0	80.252.159.255	4096
80.252.176.0	80.252.191.255	4096
80.252.208.0	80.252.223.255	4096
80.252.240.0	80.252.255.255	4096
80.253.16.0	80.253.31.255	4096
80.253.48.0	80.253.63.255	4096
80.253.80.0	80.253.95.255	4096
80.253.112.0	80.253.127.255	4096
80.253.144.0	80.253.159.255	4096
80.253.162.0	80.253.167.255	1536
80.253.176.0	80.253.191.255	4096
80.253.208.0	80.253.223.255	4096
80.253.240.0	80.253.255.255	4096
80.254.16.0	80.254.31.255	4096
80.254.48.0	80.254.95.255	12288
80.254.112.0	80.254.127.255	4096
80.254.144.0	80.254.159.255	4096
80.254.208.0	80.254.223.255	4096
80.254.240.0	80.254.255.255	4096
80.255.16.0	80.255.31.255	4096
80.255.48.0	80.255.63.255	4096
80.255.80.0	80.255.127.255	12288
80.255.144.0	80.255.191.255	12288
80.255.208.0	80.255.223.255	4096
80.255.240.0	80.255.255.255	4096
81.10.128.0	81.10.255.255	32768
81.27.160.0	81.27.175.255	4096
135.173.9.0	135.173.9.255	256
137.67.0.0	137.67.255.255	65536
137.150.0.0	137.150.255.255	65536
154.150.0.0	154.150.255.255	65536
164.216.0.0	164.216.255.255	65536
164.221.96.0	164.221.183.255	22528
164.221.217.0	164.221.223.255	1792
167.224.0.0	167.224.15.255	4096
168.165.0.0	168.165.0.255	256
168.165.5.0	168.165.167.255	41728
168.165.173.0	168.165.173.255	256
168.165.175.0	168.165.252.255	19968
168.165.255.0	168.165.255.255	256

170.169.122.0	170.169.122.255	256
192.0.2.0	192.0.2.255	256
192.31.174.0	192.31.174.255	256
192.42.62.0	192.42.62.255	256
192.50.25.0	192.50.25.255	256
192.71.254.0	192.71.255.255	512
192.77.189.0	192.77.189.255	256
192.102.65.0	192.102.65.255	256
192.102.67.0	192.102.67.255	256
192.102.68.0	192.102.68.255	256
192.102.69.0	192.102.69.255	256
192.102.70.0	192.102.70.255	256
192.102.71.0	192.102.71.255	256
192.102.72.0	192.102.72.255	256
192.102.73.0	192.102.73.255	256
192.102.74.0	192.102.74.255	256
192.102.75.0	192.102.75.255	256
192.149.125.0	192.149.125.255	256
192.152.29.0	192.152.29.255	256
192.168.200.0	192.168.200.255	256
192.231.201.0	192.231.201.255	256
193.108.89.0	193.108.89.255	256
193.108.152.0	193.108.152.255	256
193.111.72.0	193.111.73.255	512
193.188.137.0	193.188.137.255	256
193.253.0.0	193.253.0.255	256
193.253.1.0	193.253.1.255	256
194.14.69.0	194.14.69.255	256
194.56.124.0	194.56.125.255	512
194.56.126.0	194.56.126.255	256
194.69.168.0	194.69.177.255	2560
194.69.179.0	194.69.180.255	512
194.69.182.0	194.69.191.255	2560
194.69.228.128	194.69.228.255	128
194.124.215.0	194.124.215.255	256
194.125.252.0	194.125.253.255	512
194.125.254.0	194.125.255.255	512
194.153.154.0	194.153.154.127	128
198.22.254.0	198.22.254.255	256
198.73.212.0	198.73.215.255	1024
198.80.179.0	198.80.179.255	256

198.102.161.0	198.102.161.255	256
198.102.250.0	198.102.250.255	256
198.183.227.0	198.183.227.255	256
198.235.148.0	198.235.148.255	256
199.33.7.0	199.33.7.255	256
199.164.200.0	199.164.200.255	256
199.242.0.0	199.242.3.255	1024
199.242.4.0	199.242.5.255	512
199.242.6.0	199.242.6.255	256
199.246.230.0	199.246.230.255	256
199.246.231.0	199.246.231.255	256
199.246.232.0	199.246.232.255	256
199.246.233.0	199.246.233.255	256
199.246.234.0	199.246.234.255	256
199.246.235.0	199.246.235.255	256
199.246.236.0	199.246.236.255	256
199.246.237.0	199.246.237.255	256
199.246.238.0	199.246.238.255	256
199.246.239.0	199.246.239.255	256
199.246.240.0	199.246.240.255	256
199.246.241.0	199.246.241.255	256
199.246.242.0	199.246.242.255	256
199.246.243.0	199.246.243.255	256
199.246.244.0	199.246.244.255	256
199.246.245.0	199.246.245.255	256
199.246.246.0	199.246.246.255	256
199.246.247.0	199.246.247.255	256
199.246.248.0	199.246.248.255	256
199.246.249.0	199.246.249.255	256
199.246.250.0	199.246.250.255	256
199.246.251.0	199.246.251.255	256
199.246.252.0	199.246.252.255	256
199.246.253.0	199.246.253.255	256
199.249.191.0	199.249.191.255	256
200.189.35.0	200.189.38.255	1024
202.2.61.0	202.2.61.255	256
202.12.240.136	202.12.240.139	4
202.14.195.0	202.14.195.127	128
202.14.195.128	202.14.195.255	128
202.39.80.0	202.39.95.255	4096
202.68.0.0	202.68.21.255	5632

202.68.23.0	202.68.31.255	2304
202.68.32.0	202.68.47.255	4096
202.68.48.0	202.68.55.255	2048
202.86.64.0	202.86.95.255	8192
202.127.64.0	202.127.67.255	1024
202.127.68.0	202.127.68.255	256
202.127.69.0	202.127.69.255	256
202.127.70.0	202.127.71.255	512
202.177.200.0	202.177.201.255	512
203.1.106.224	203.1.106.239	16
203.3.48.24	203.3.48.27	4
203.5.144.0	203.5.159.255	4096
203.7.194.0	203.7.194.127	128
203.7.194.192	203.7.194.223	32
203.12.37.192	203.12.37.199	8
203.12.193.192	203.12.193.207	16
203.13.198.212	203.13.198.215	4
203.21.141.0	203.21.141.127	128
203.21.141.128	203.21.141.159	32
203.21.141.160	203.21.141.191	32
203.21.141.192	203.21.141.223	32
203.21.141.224	203.21.141.255	32
203.22.189.0	203.22.189.255	256
203.23.255.0	203.23.255.127	128
203.25.176.0	203.25.176.255	256
203.26.63.0	203.26.63.255	256
203.26.126.128	203.26.126.191	64
203.26.152.32	203.26.152.63	32
203.26.152.64	203.26.152.95	32
203.26.152.128	203.26.152.159	32
203.26.152.192	203.26.152.223	32
203.26.152.224	203.26.152.255	32
203.26.189.64	203.26.189.95	32
203.27.118.0	203.27.118.15	16
203.27.118.64	203.27.118.79	16
203.27.118.96	203.27.118.127	32
203.34.73.0	203.34.73.63	64
203.55.43.0	203.55.43.127	128
203.57.38.0	203.57.38.255	256
203.62.176.0	203.62.176.255	256
203.87.27.128	203.87.27.255	128

203.101.24.212	203.101.24.215	4
204.68.155.0	204.68.155.255	256
204.69.190.0	204.69.190.255	256
204.89.214.0	204.89.214.255	256
204.127.0.0	204.127.15.255	4096
204.127.32.0	204.127.47.255	4096
204.127.64.0	204.127.71.255	2048
204.174.112.0	204.174.112.255	256
204.225.156.0	204.225.156.255	256
204.238.255.0	204.238.255.255	256
205.166.4.0	205.166.4.255	256
205.172.179.0	205.172.179.255	256
206.195.121.32	206.195.121.63	32
206.224.32.0	206.224.63.255	8192
207.116.0.0	207.116.127.255	32768
207.254.150.0	207.254.150.255	256
207.254.159.0	207.254.159.255	256
207.254.161.0	207.254.161.255	256
207.254.168.0	207.254.168.255	256
207.254.169.0	207.254.169.255	256
207.254.170.0	207.254.170.255	256
207.254.171.0	207.254.171.255	256
207.254.172.0	207.254.172.255	256
207.254.173.0	207.254.173.255	256
207.254.174.0	207.254.174.255	256
207.254.185.0	207.254.185.255	256
209.146.0.0	209.146.9.255	2560
209.146.11.0	209.146.13.255	768
209.146.15.0	209.146.18.255	1024
209.146.20.0	209.146.28.255	2304
209.146.30.0	209.146.30.255	256
209.146.32.0	209.146.64.255	8448
209.146.66.0	209.146.82.255	4352
209.146.85.0	209.146.95.255	2816
209.146.97.0	209.146.111.255	3840
209.146.113.0	209.146.114.255	512
209.146.116.0	209.146.127.255	3072
210.18.203.0	210.18.203.255	256
212.63.192.0	212.63.223.255	8192
213.203.160.0	213.203.191.255	8192
213.232.106.0	213.232.107.255	512

213.232.114.0	213.232.114.255	256
213.244.124.0	213.244.127.255	1024
217.21.192.0	217.21.207.255	4096
217.75.32.0	217.75.47.255	4096
217.78.67.0	217.78.67.255	256
217.78.68.0	217.78.68.255	256
217.119.132.0	217.119.143.255	3072
217.169.64.0	217.169.79.255	4096

Appendix B

Full black holes list for 01/05/2002 00:00

```
#start_ip      #end_ip         #number_of_ips

64.215.96.0    64.215.111.255  4096
65.40.192.0    65.40.199.255   2048
68.67.80.0     68.67.95.255    4096
68.67.192.0    68.67.255.255   16384
81.10.128.0    81.10.255.255   32768
131.120.0.0    131.120.255.255 65536
135.173.9.0    135.173.9.255   256
136.243.0.0    136.243.255.255 65536
140.165.0.0    140.165.255.255 65536
144.124.0.0    144.124.255.255 65536
155.70.128.0   155.70.159.255  8192
164.221.96.0   164.221.183.255 22528
164.221.217.0  164.221.223.255 1792
168.165.0.0    168.165.0.255   256
168.165.5.0    168.165.167.255 41728
168.165.173.0  168.165.173.255 256
168.165.175.0  168.165.252.255 19968
168.165.255.0  168.165.255.255 256
170.169.122.0  170.169.122.255 256
170.217.0.0    170.217.255.255 65536
192.0.2.0      192.0.2.255     256
192.23.157.0   192.23.157.255  256
192.42.62.0    192.42.62.255   256
192.50.25.0    192.50.25.255   256
```

192.71.29.0	192.71.29.255	256
192.76.121.0	192.76.121.255	256
192.77.189.0	192.77.189.255	256
192.88.9.0	192.88.9.255	256
192.88.10.0	192.88.10.255	256
192.88.89.0	192.88.89.255	256
192.88.90.0	192.88.90.255	256
192.88.93.0	192.88.93.255	256
192.88.94.0	192.88.94.255	256
192.98.98.0	192.98.98.255	256
192.102.83.0	192.102.83.255	256
192.108.80.0	192.108.87.255	2048
192.121.45.0	192.121.45.255	256
192.148.166.0	192.148.166.255	256
192.197.68.0	192.197.68.255	256
193.108.8.0	193.108.15.255	2048
193.108.89.0	193.108.89.255	256
193.110.110.0	193.110.110.255	256
193.110.111.0	193.110.111.255	256
193.188.137.0	193.188.137.255	256
193.253.0.0	193.253.0.255	256
193.253.1.0	193.253.1.255	256
194.55.144.0	194.55.145.255	512
194.69.168.0	194.69.177.255	2560
194.69.179.0	194.69.180.255	512
194.69.182.0	194.69.191.255	2560
194.69.228.128	194.69.228.255	128
194.85.64.0	194.85.71.255	2048
194.124.215.0	194.124.215.255	256
194.125.254.0	194.125.255.255	512
194.132.24.0	194.132.25.255	512
194.153.154.0	194.153.154.127	128
196.32.153.0	196.32.153.255	256
198.51.172.0	198.51.172.255	256
198.143.128.0	198.143.159.255	8192
199.74.151.0	199.74.151.255	256
199.74.152.0	199.74.152.255	256
199.124.16.0	199.124.23.255	2048
199.253.96.0	199.253.111.255	4096
200.34.128.0	200.34.128.255	256
200.34.192.0	200.34.192.255	256

200.68.168.0	200.68.168.255	256
200.68.170.0	200.68.170.255	256
200.85.224.0	200.85.239.255	4096
202.2.61.0	202.2.61.255	256
202.12.240.136	202.12.240.139	4
202.14.195.0	202.14.195.127	128
202.14.195.128	202.14.195.255	128
202.39.80.0	202.39.95.255	4096
202.47.32.0	202.47.63.255	8192
202.68.0.0	202.68.21.255	5632
202.68.23.0	202.68.31.255	2304
202.68.32.0	202.68.47.255	4096
202.68.48.0	202.68.55.255	2048
202.86.64.0	202.86.95.255	8192
202.127.64.0	202.127.67.255	1024
202.127.68.0	202.127.68.255	256
202.127.69.0	202.127.69.255	256
202.127.70.0	202.127.71.255	512
202.130.69.0	202.130.69.255	256
202.137.8.0	202.137.11.255	1024
202.144.233.0	202.144.233.255	256
202.177.200.0	202.177.201.255	512
203.1.72.0	203.1.75.255	1024
203.1.106.224	203.1.106.239	16
203.3.48.24	203.3.48.27	4
203.7.194.0	203.7.194.127	128
203.7.194.192	203.7.194.223	32
203.12.37.192	203.12.37.199	8
203.12.193.192	203.12.193.207	16
203.13.198.212	203.13.198.215	4
203.21.141.0	203.21.141.127	128
203.21.141.128	203.21.141.159	32
203.21.141.160	203.21.141.191	32
203.21.141.192	203.21.141.223	32
203.21.141.224	203.21.141.255	32
203.22.189.0	203.22.189.255	256
203.23.255.0	203.23.255.127	128
203.26.63.0	203.26.63.255	256
203.26.126.128	203.26.126.191	64
203.26.152.32	203.26.152.63	32
203.26.152.64	203.26.152.95	32

203.26.152.128	203.26.152.159	32
203.26.152.192	203.26.152.223	32
203.26.152.224	203.26.152.255	32
203.26.189.64	203.26.189.95	32
203.27.50.0	203.27.50.255	256
203.27.118.0	203.27.118.15	16
203.27.118.64	203.27.118.79	16
203.27.118.96	203.27.118.127	32
203.28.126.0	203.28.126.255	256
203.31.226.0	203.31.226.255	256
203.32.4.0	203.32.4.255	256
203.34.73.0	203.34.73.63	64
203.34.255.0	203.34.255.255	256
203.55.43.0	203.55.43.127	128
203.57.38.0	203.57.38.255	256
203.62.176.0	203.62.176.255	256
203.87.27.128	203.87.27.255	128
203.101.24.212	203.101.24.215	4
204.29.160.0	204.29.160.255	256
204.127.0.0	204.127.15.255	4096
204.127.32.0	204.127.47.255	4096
204.127.64.0	204.127.71.255	2048
204.149.127.0	204.149.127.255	256
206.163.240.0	206.163.255.255	4096
206.195.121.32	206.195.121.63	32
207.116.0.0	207.116.127.255	32768
209.126.32.0	209.126.63.255	8192
212.63.192.0	212.63.223.255	8192
213.203.160.0	213.203.191.255	8192
213.232.106.0	213.232.107.255	512
213.232.114.0	213.232.114.255	256
216.82.88.0	216.82.89.255	512
216.82.92.0	216.82.93.255	512
216.82.96.0	216.82.96.255	256
216.82.100.0	216.82.101.255	512
216.82.109.0	216.82.109.255	256
216.174.120.0	216.174.127.255	2048
216.176.6.0	216.176.6.255	256
216.183.160.0	216.183.167.255	2048
216.183.180.0	216.183.181.255	512
216.183.182.0	216.183.182.255	256

```
217.21.192.0   217.21.207.255  4096
217.31.176.0   217.31.191.255  4096
217.75.32.0    217.75.47.255   4096
```

Appendix C

Full black holes list for 15/05/2002 00:00

```
#start_ip      #end_ip         #number_of_ips

61.0.80.0      61.0.95.255     4096
61.0.96.0      61.0.111.255    4096
61.0.128.0     61.0.159.255    8192
61.0.160.0     61.0.175.255    4096
62.212.192.0   62.212.223.255  8192
64.82.129.0    64.82.129.255   256
64.136.32.0    64.136.63.255   8192
64.215.96.0    64.215.111.255  4096
67.93.0.0      67.93.127.255   32768
67.93.128.0    67.93.191.255   16384
68.20.0.0      68.20.31.255    8192
68.20.32.0     68.20.63.255    8192
80.254.64.0    80.254.79.255   4096
81.10.128.0    81.10.255.255   32768
135.173.9.0    135.173.9.255   256
147.204.0.0    147.204.255.255 65536
158.197.0.0    158.197.255.255 65536
170.169.122.0  170.169.122.255 256
192.0.2.0      192.0.2.255     256
192.42.62.0    192.42.62.255   256
192.50.25.0    192.50.25.255   256
192.77.189.0   192.77.189.255  256
192.98.98.0    192.98.98.255   256
192.102.83.0   192.102.83.255  256
```

192.124.181.0	192.124.181.255	256
192.197.51.0	192.197.51.255	256
193.58.210.0	193.58.210.255	256
193.58.211.0	193.58.211.255	256
193.58.212.0	193.58.212.255	256
193.108.89.0	193.108.89.255	256
193.111.226.0	193.111.226.255	256
193.188.137.0	193.188.137.255	256
193.253.0.0	193.253.0.255	256
193.253.1.0	193.253.1.255	256
194.69.168.0	194.69.177.255	2560
194.69.179.0	194.69.180.255	512
194.69.182.0	194.69.191.255	2560
194.69.228.128	194.69.228.255	128
194.102.116.0	194.102.116.255	256
194.102.117.0	194.102.117.255	256
194.102.192.0	194.102.192.255	256
194.124.215.0	194.124.215.255	256
194.125.254.0	194.125.255.255	512
194.132.24.0	194.132.25.255	512
195.35.80.0	195.35.80.255	256
199.80.103.0	199.80.103.255	256
200.34.192.0	200.34.192.255	256
200.82.144.0	200.82.147.255	1024
200.196.144.0	200.196.159.255	4096
202.2.61.0	202.2.61.255	256
202.12.240.136	202.12.240.139	4
202.14.164.0	202.14.164.255	256
202.20.68.0	202.20.68.255	256
202.39.80.0	202.39.95.255	4096
202.68.0.0	202.68.21.255	5632
202.68.23.0	202.68.31.255	2304
202.68.32.0	202.68.47.255	4096
202.68.48.0	202.68.55.255	2048
202.86.64.0	202.86.95.255	8192
202.127.64.0	202.127.67.255	1024
202.127.68.0	202.127.68.255	256
202.127.69.0	202.127.69.255	256
202.127.70.0	202.127.71.255	512
202.130.69.0	202.130.69.255	256
202.177.200.0	202.177.201.255	512

203.1.106.224	203.1.106.239	16
203.3.48.24	203.3.48.27	4
203.7.194.0	203.7.194.127	128
203.7.194.192	203.7.194.223	32
203.12.37.192	203.12.37.199	8
203.12.193.192	203.12.193.207	16
203.13.198.212	203.13.198.215	4
203.13.220.0	203.13.221.255	512
203.21.141.0	203.21.141.127	128
203.21.141.128	203.21.141.159	32
203.21.141.160	203.21.141.191	32
203.21.141.192	203.21.141.223	32
203.21.141.224	203.21.141.255	32
203.22.189.0	203.22.189.255	256
203.23.28.0	203.23.28.255	256
203.23.110.0	203.23.110.255	256
203.23.255.0	203.23.255.127	128
203.26.63.0	203.26.63.255	256
203.26.126.128	203.26.126.191	64
203.26.152.32	203.26.152.63	32
203.26.152.64	203.26.152.95	32
203.26.152.128	203.26.152.159	32
203.26.152.192	203.26.152.223	32
203.26.152.224	203.26.152.255	32
203.26.174.0	203.26.174.255	256
203.26.189.64	203.26.189.95	32
203.27.118.0	203.27.118.15	16
203.27.118.64	203.27.118.79	16
203.27.118.96	203.27.118.127	32
203.32.4.0	203.32.4.255	256
203.32.155.0	203.32.155.3	4
203.33.135.0	203.33.135.255	256
203.34.73.0	203.34.73.63	64
203.55.43.0	203.55.43.127	128
203.57.38.0	203.57.38.255	256
203.62.176.0	203.62.176.255	256
203.87.27.128	203.87.27.255	128
203.101.24.212	203.101.24.215	4
203.196.159.0	203.196.159.255	256
204.62.253.0	204.62.253.255	256
204.127.0.0	204.127.15.255	4096

204.127.32.0	204.127.47.255	4096
204.127.64.0	204.127.71.255	2048
205.223.128.0	205.223.128.255	256
205.236.8.0	205.236.8.255	256
205.236.9.0	205.236.9.255	256
205.236.10.0	205.236.10.255	256
205.236.11.0	205.236.11.255	256
205.236.12.0	205.236.12.255	256
205.237.23.0	205.237.23.255	256
207.116.0.0	207.116.127.255	32768
207.182.114.0	207.182.114.255	256
207.182.192.0	207.182.192.255	256
207.245.136.0	207.245.136.255	256
207.245.138.0	207.245.138.255	256
212.63.192.0	212.63.223.255	8192
213.179.40.0	213.179.47.255	2048
213.203.160.0	213.203.191.255	8192
213.232.106.0	213.232.107.255	512
216.82.224.0	216.82.239.255	4096
217.21.192.0	217.21.207.255	4096
217.75.32.0	217.75.47.255	4096
217.113.224.0	217.113.239.255	4096

Bibliography

[1] The RIPE Official Website:
`http://www.ripe.net`

[2] Andrew S. Tanenbaum. *Computer Networks*, Prentice-Hall, 1996, ISBN 0-13-394248-1

[3] Ioan Jurca. *Computer Network Programming*, Editura de Vest, 2000, ISBN 973-36-0331-7

[4] Geoff Huston, Telstra. *Scaling Inter-Domain Routing. A view Forward*, The Internet Protocol Journal, December 2001

[5] Pete Loshin. *Border Gateway Protocol RFCs*, Morgan Kaufmann, 2000, ISBN 0-12-455846-1

[6] Nick Feamster. *Security for Wide Area Internet Routing*, 2000,
`http://www.acm.org/crossroads/columns/onpatrol/`

[7] Shishir Gundavaram. *CGI Programming*, O´Reilly & Associates, Inc., 1996, ISBN 1-56592-168-2

[8] Larry Wall, Tom Christiansen & Jon Orwant. *Programming Perl*, Third Edition, O´Reilly & Associates, Inc., 2000, ISBN 0-596-00027-8

[9] René Brun & Co. *ROOT. Users Guide*, June 2001

[10] Alex van den Bogaerdt. *Alex van den Bogaerdt's RRDtool tutorial*, February 2001

[11] *RIS Data Web Pages*,
`http://data.ris.ripe.net`

[12] *Multi-Threaded Routing Toolkit Web Pages*,
`http://www.mrtd.net`

[13] *Fast Lexical Analyser Generator Web Pages*,
`http://www.gnu.org/software/flex/flex.html`

[14] *ROOT Web Pages*,
`http://root.cern.ch/`

[15] *RRDTool Web Pages*,
`http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/`

[16] *ImageMagick Web Pages*,
`http://www.imagemagick.org`

[17] Tobias Oetiker. *The Not So Short Introduction to LaTeX* , August 2001