Università degli Studi di Roma Tre
Dipartimento di Informatica e Automazione
Via della Vasca Navale, 79 – 00146 Roma, Italy

How to Extract BGP Peering Information from the Internet Routing Registry

G. Di Battista[1], T. Refice[1], and M. Rimondini[1]

RT-DIA-108-2006 **Maggio 2006**

(1) Dipartimento di Informatica e Automazione,
Università di Roma Tre,
Rome, Italy.
{gdb,refice,rimondin}@dia.uniroma3.it

Work partially supported by European Commission - Fet Open project DELIS - Dynamically Evolving Large Scale Information Systems - Contract no 001907 and by the Miur Project ALGO-NEXT: Algorithms for the Next Generation Internet and Web: Methodologies, Design, and Experiments.

ABSTRACT

We describe an on-line service, and its underlying methodology, designed to extract peering information from the Internet Routing Registry. Both the method and the service are based on: a consistency manager for integrating information across different registries, an RPSL analyzer that extracts peering specifications from RPSL objects, and a peering classifier that aims at understanding to what extent such peering specifications actually contribute to fully determine a peering. A peering graph is built with different levels of confidence. We compare the effectiveness of our method with the state of the art. The comparison puts in evidence the quality of the proposed method.

1 Introduction

The Internet Routing Registry (IRR) [12, 7] is a large distributed repository of information, containing the routing policies of many of the networks that compose the Internet. The IRR was born about ten years ago with the main purpose to promote stability, consistency, and security of the global Internet routing. It consists of several registries that are maintained on a voluntary basis. The routing policies are expressed in the Routing Policy Specification Language (RPSL) [14, 20, 15]. The IRR can be used by operators to look up peering agreements, to study optimal policies, and to (possibly automatically) configure routers.

There is a wide discussion about the current role of the IRR [24]. Some people consider it outdated and almost useless. Others have put in evidence its importance to understand the Internet routing and that it contains unique and significant information. Anyway, it is undeniable that the IRR keeps on being fed by many operators, that useful tools have been developed to deal with the IRR (see, e.g., IRRToolSet [3]), and that several research issues on the Internet routing are, at least partially, based on the content of the IRR. However, as pointed out in [24], extracting information from the IRR is far from trivial: the policies written in RPSL can be quite complex, the level of accuracy of the descriptions largely varies, and, also because of its distributed nature, the IRR contains many inconsistencies [18].

The purpose of this paper is to describe an on-line service, and its underlying methodology, that extracts peering information from the IRR. We believe that our service can have beneficial effects both for operators and for several research projects.

For example, the RIPE offers an IRR consistency check service (RRCC) [23, 11] that aims at detecting unregistered peerings. It verifies whether a peering that can be inferred from operational routing data is also described, in some form, into the IRR. We will show later that currently the RIPE service extracts peerings from the IRR in a way that is much less accurate than the one presented in this paper. Actually, the need of a better analysis of the content of the IRR is pointed out by the RIPE itself that considers this as a long term goal [11].

On the research side, Mahadevan et al. [19] presented a comparison of several characteristics of the AS-level topologies built on the basis of different data sources, including the IRR. They also proposed a metric to characterize such topologies. Zhang et al. [25] derived an AS-level topology combining IRR data with BGP routing information collected from multiple sources, such as RouteViews [6], looking glasses, and route servers. They showed that the data from the RIPE registry reveal topology information which cannot be found in other sources. Siganos et al. [24] developed a tool, called Nemecis [5], that checks the correctness of IRR data and their consistency with respect to BGP routing table information. They argued that 28% of ASes have both correct and consistent policies and that RIPE is by far the most accurate registry. Carmignani et al. [16] presented a service for the visualization of IRR data. We shall compare the level of accuracy of the methods for extracting peerings from the IRR used in the above papers with respect to ours.

The main results presented in this paper can be summarized as follows.

- We describe a method and a on-line service to extract peering relationships from the IRR. Both the method and the service are based on: a consistency manager for integrating information across different registries, an RPSL analyzer that extracts peering specifications from RPSL objects, and a peering classifier that aims at understanding to what extent such peering specifications actually contribute to fully determine a peering. A peering graph is built with different levels of confidence.

- We prove the effectiveness of our method by showing that it allows to discover many more peerings than the state of the art.
- We provide an implementation of our method as an on-line service, available at `http://tocai.dia.uniroma3.it/~irr_analysis`.
- As a side effect, our study highlights how the different RPSL constructions are actually used to specify peerings.

The paper is organized as follows. Section 2 provides an overview of the IRR and of RPSL. In Section 3 we present our methodology and the system we developed to extract peering information from RPSL data. We use our system to analyze the data set specified in Section 4. The problems arising from the use of several registries are faced in Section 5. We explain in Section 6 how the peerings are discovered. Section 7 shows how to build a topology based on the data we extract. A quantitative comparison with the state of the art is done in Section 8. Future work is described in Section 9.

2 Background

The Internet is divided into tens of thousands of administrative domains called *Autonomous Systems* (*AS*), each usually adopting a unique routing protocol and consistent routing policies. An AS is identified by a number. The *Border Gateway Protocol* (*BGP*) [21, 22] is the routing protocol used to exchange reachability information between ASes. BGP allows to define complex routing policies that affect the propagation of BGP announcements. Two ASes that exchange routing information using BGP are said to have a *peering* between them; the ASes which have a peering with an AS A are termed *peerers* of A.

There are many publicly available registries that describe both the allocation of Internet resources and BGP routing policies. The *Regional Internet Registries* [17] (e.g., RIPE [9], ARIN [1]) are in charge of maintaining information over wide geographic regions. The *Local Internet Registries* (e.g., VERIO [13], LEVEL3 [4]) describe the policies of the customers of a specific ISP. Taken together, all these registries form the *Internet Routing Registry* (*IRR*). The main purpose of the IRR is to support a consistent global configuration of routing policies. It is also possible to automatically create BGP filters and router configurations from registry information by using tools such as IRRToolSet [3].

The registration and maintenance of routing policies are performed on a voluntary basis by network operators, who may register such policies at one or more registries. As a consequence, information therein may be incorrect, incomplete, or outdated. Indeed, some large ISPs and Internet Exchange Points rely on the IRR for route filtering and do not allow their customers to participate in BGP routing unless they document their routing policies in a registry.

The routing policies stored in the IRR are described using the *Routing Policy Specification Language* (*RPSL*) [14, 20] or its more recent variant *RPSLng* [15], which introduces support to both multicast and IPv6. RPSL is an object-oriented language that defines 13 classes of objects. Routing policies are described in the `import`, `export`, and `default` attributes of `aut-num` objects. In turn, `aut-nums` may reference other objects that contribute to the specification of the policies, such as `as-sets` and `peering-sets`.

What follows is a portion of an RPSL `aut-num` object from the RIPE registry which describes the routing policies of AS137 (last updated 08/30/00). The portion of the `import` (`export`) attribute following the `from` (`to`) keyword is a very simple example of *peering specification*. The object indicates that AS137 accepts any route sent to it by AS20965 and by AS1299 and propagates to AS1299 all the routes originated by ASes belonging to the `as-set` named AS-GARR (an `as-set` is an RPSL object that specifies a set of ASes). This implies that AS137 has a peering with AS20965 and AS1299.

```
aut-num: AS137
import: from AS20965 action pref=100;
      from AS1299 action pref=100;
      accept ANY
[...]
export: to AS1299 announce AS-GARR
[...]
changed: noc@garr.it 20000830
source: RIPE
```

In our service we make use of *Peval*, a low level policy evaluation tool conceived to write router configuration generators. Peval is part of the *Internet Routing Registry Toolset* (*IRRToolSet*) [3] suite. Peval takes as input an RPSL expression and evaluates it by applying RPSL set operators (`AND`, `OR`, `NOT`) and by expanding `as-sets`, `route-sets`, and AS numbers into the corresponding sets of prefixes. Alternatively, Peval can stop the expansion at the level of ASes. We access the IRR data also through the *Internet Routing Registry Daemon* (*IRRd*) [2], a freely available stand-alone IRR database server supporting both RPSL and RPSLng.

3 An IRR Peering Extraction Service

Our method for extracting peerings has been implemented and is available as an on-line service at `http://tocai.dia.uniroma3.it/~irr_analysis`. The service produces, on a daily basis: (i) General statistics on the IRR (number of objects defined in each registry, amount of overlapping information between registries, etc.). (ii) A set of pairs of ASes, corresponding to peering relationships extracted from the IRR. Each pair is labeled with information about the context where it has been found, like the type of policy and the registry. The architecture of the service is composed by the following main blocks.

Basic Info Registry Analyzer : provides preliminary information on the registries. For example, it computes the number of `aut-nums` and `as-sets` inside each registry. Also, it computes the “amount of overlap” between pairs of registries. Further, it deals with the evolution over time of registries, measuring the number of everyday updates. Such basic information is useful for giving a correct interpretation of the results obtained by using the service.

Inter-Registry Consistency Manager : starting from a set of registries that, considered as a whole, may contain inconsistent information, constructs a purged new consistent version of the IRR. RPSL objects with the same key appearing in different registries are compared. A choice is done relying on the timestamp of the last change and in terms of the semantics of the attributes.

RPSL Peering Specification Analyzer : extracts from the IRR the peering relationships between ASes. This is done by analyzing the body of RPSL objects. The relationships extracted in this phase are *candidate peerings* for the subsequent elaboration. In this step we also evaluate the current usage of the RPSL syntax constructions for expressing peerings. This block exploits IRRd and Peval.

Peering Classifier : classifies the computed candidate peerings according to their relative matchings in order to understand to what extent they contribute to fully specify a peering. The output of this step is a peering graph, that can be constructed with different levels of confidence.

The above blocks will be detailed in the following sections.

4 Data Set

The registry data we use throughout this paper has been downloaded from [10, 8] on 03/31/06. At that time there were 68 registries available for download, which are listed in Table 1. The registries are sorted according to their size in terms of number of `aut-num` objects registered inside them (2nd column). Void registries are omitted.

ripe	11468	92%	host	10	90%	reach	2	50%
apnic	3299	84%	ottix	9	33%	nestegg	2	100%
radb	2695	77%	csas	9	100%	gw	2	100%
arin	555	41%	rogers	8	100%	bendtel	2	50%
verio	498	42%	risq	8	100%	univali	1	100%
dodnic	254	11%	crc	8	62%	soundinternet	1	100%
altdb	249	63%	deru	7	0%	panix	1	0%
savvis	180	75%	sprint	6	16%	openface	1	100%
epoch	137	100%	bcnet	5	60%	koren	1	100%
level3	126	40%	vdn	4	25%	gts	1	100%
bell	74	98%	rgnet	4	100%	gt	1	100%
aoltw	53	3%	mto	4	25%	fastvibe	1	100%
jpirr	43	34%	easynet	4	100%	eicat	1	100%
sinet	28	10%	digitalrealm	4	100%	ebit	1	100%
arcstar	16	6%	look	3	100%	area151	1	100%
chtr	11	0%	retina	2	50%			

Table 1: `aut-num` in the registries before and after resolving inter-registry inconsistencies.

Table 2 indicates the level of overlapping between the largest registries. For each pair of registries (R_{i_1}, R_{i_2}) the table provides the number of `aut-num` objects that are registered both in R_{i_1} and in R_{i_2}. The main diagonal (R_i, R_i) reports the count of `aut-nums` appearing in registry R_i only.

	apnic	arin	radb	ripe	verio
apnic	2688	1	423	19	113
arin	1	463	37	7	14
radb	423	37	2037	50	45
ripe	19	7	50	11238	23
verio	113	14	45	23	310

Table 2: Overlapping `aut-nums` between registries.

Figure 1 gives an idea of the amount of work of the operators on the IRR over time. Namely, it shows the daily percentage of size variation of the RIPE registry (that is by far the most popular) over the period 11/14/05–04/26/06. The plot shows that the RIPE registry keeps on being updated on a regular basis and that it grows of about 2% per month. Our reference date (arrow in the plot) has been selected to be one with an average number of updates.

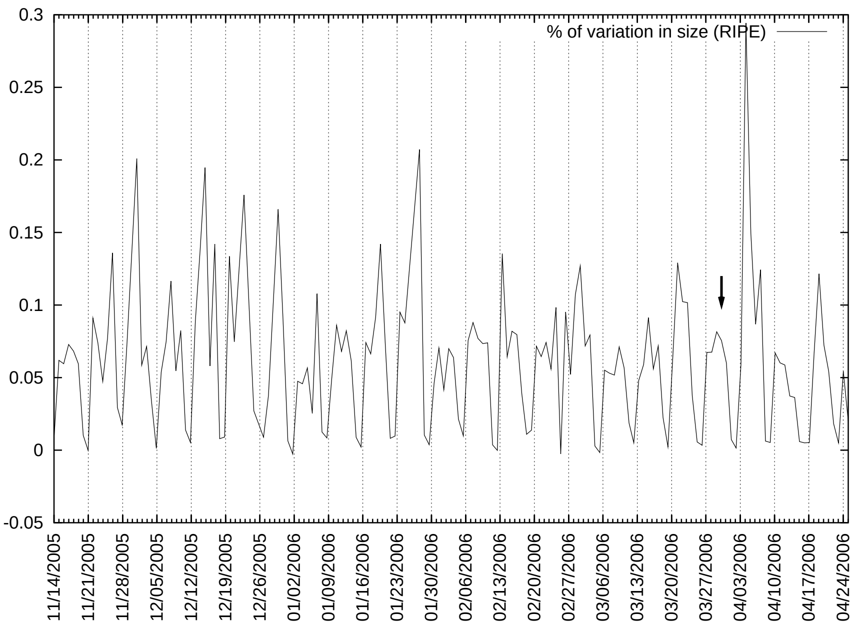

Figure 1: Daily growth of the RIPE registry.

5 Integrating Registries

RFC 2622 [14] considers the IRR system as a whole. However, the IRR is composed by several registries, and the same object may be defined in many of them. For example, in our data set, AS2510 is registered both in APNIC and in JPIRR. Of course, the presence of multiple definitions of the same object can lead to inconsistencies. Our Inter-Registry Consistency Manager takes care of resolving them. It takes as input a set of registries and processes them in order to build a new repository where each RPSL object is defined only once. Whenever it detects for a certain RPSL object the presence of multiple definitions (possibly coming from different registries), it examines all the definitions in order to determine which of them contains the most significant information. Such definition is kept in the final repository, while all the others are discarded. In what follows, a triple (x;y;z) represents a number of `aut-nums`, `as-sets`, `peering-sets`, respectively. In our data set we have (19,800;7,798;149) overall definitions. Among them, (18,735;7,478;149) are unique. Hence, potential inconsistencies affect at most (1,065;300;0) objects.

If an RPSL object is defined multiple times, the most informative definition is selected. We call *stub object* an `aut-num` object which misses information about BGP policies or a `set` object which misses the specification of the set members (consider that some attributes of RPSL objects are optional). Operators sometimes use stub objects as "placeholders" which can be referred to inside other parties' BGP policies. Our data set contains (3,133;206;11) stub definitions. If we detect that an object appears in more than one registry, we discard its stub instances. Since stub objects do not provide useful data about the existence of peerings, this does not cause any loss of information.

However, it may still be the case that several registries contain non-stub instances of a single RPSL object. If this happens, we select the instance with the most recent update timestamp, that is contained in the `changed` attribute. After removing the stub definitions and selecting the most recent timestamp, the potential inconsistencies affect at most (44;77;0) objects. Note that, even if the `changed` attribute is optional, in our data set there is only one definition that misses the timestamp over 2,271,446 objects in the IRR.

Yet, if there are (at least) two instances with the same most recent date, we select the definition belonging to the registry with highest *rank*. We rank the registries according to their size. This choice is somehow arbitrary. However, a registry with a higher number of objects often provides more reliable information than the others. Also, as shown above, the choice impacts very few objects. Last, we have inspected the objects that have multiple definitions with the most recent date and discovered that in most cases their definitions coincide. Of course, other rankings could be applied without impacting the general structure of the method.

The third column of Table 1 shows the percentage of the remaining `aut-num` objects per registry after running the Inter-Registry Consistency Manager. It is interesting to observe that RIPE has the highest absolute number and the highest percentage among the top 5 registries.

6 Discovering Peerings through RPSL Analysis

In this section we detail the procedure we apply to extract peering information from RPSL data. As already stated in Section 2, peering specifications only appear in the `[mp-]import`, `[mp-]export`, and `[mp-]default` attributes of `aut-num` objects. Hence, `aut-nums` are the starting points of the peering extraction.

What follows is a fragment (25 lines, `ASX1-ASX13` represent ASes) of RPSL code that puts in evidence many of the problems encountered while discovering peerings in the IRR. We now show how to extract from this fragment the peerers of `ASX5`.

```
1.  peering-set: ASX1:PRNG-Y1  4. peering-set: PRNG-Y2
2.  peering: PRNG-Y2           5. peering: ASX7
3.  peering: ASX6

6.  as-set: ASX1:AS-Z1         9.  as-set: ASX2:AS-Z2
7.  members: ASX8, ASX9        10. members: ASX2, ASX4
8.  mbrs-by-ref: MNTR-ASX1

11. aut-num: ASX10
12. member-of: ASX1:AS-Z1
13. mnt-by: MNTR-ASX1

14. aut-num: ASX5
15. import: { from ASX2:AS-Z2 accept 100.0.0.0/8;
16.         } refine {
17.           from ASX1 ASX2 accept 100.1.0.0/16;
18.         } except {
19.           from ASX3 accept 100.1.1.0/24;}
20. export: to ASX1:PRNG-Y1
21.         to ASX1:AS-Z1 except ASX9
22.         announce 100.1.1.0/24
23. mp-export: to ASX11 at 2001::1 announce 2001::/48
24. default: to ASX12 action pref=10
25. default: to ASX13 100.1.1.1 at 100.1.1.2
```

By scanning such a code with the RIPE RRCC scripts [11], the following peerers are found: `ASX1`, `ASX3`, `ASX12`, `ASX13`. They come out by examining the lines 17, 19 (`import from`), 24, 25 (`default to`). However, such peerings are neither correct nor complete. On one hand, the peering between `ASX5` and `ASX1` does not hold, since the `refine` semantics require to compute the intersection between `ASX2:AS-Z2` and ASes `ASX1`, `ASX2`. On the other hand, there are peerers of `ASX5` that have not been discovered. The peerers `ASX6` and `ASX7` can be inferred only by considering all the ASes that belong to the `peering-set` used at line 20 and defined at lines 1-5. Further, the peerers `ASX8` and `ASX10` can be inferred only by considering all the ASes that belong to the `as-set` used at line 21 and defined at lines 6-8,11-13. Finally, the peerers `ASX2` and `ASX11` are not discovered because the scripts in [11] support neither multiple peerings appearing in the same `from` expression, nor the `mp-export` attribute. Even if the example is not taken from the real life IRR, it is a patchwork of pieces of code that are quite common in RPSL objects.

We now show our method for extracting peerings from the RPSL code. We describe how we build a set of candidate peerings, which we use later to identify peerings.

For each `aut-num` object A we compute three sets $import(A)$, $export(A)$, and $default(A)$ of candidate peerers corresponding to the `[mp-]import`, `[mp-]export`, and `[mp-]default` attributes, respectively. We describe our procedure with reference to the `import` attributes. The other attributes are processed in a similar way. If A defines a private AS, it is discarded (it should not be visible in the Internet).

An `import` attribute may contain a simple or a structured policy. A simple `import` policy may contain several peering specifications. In this case $import(A)$ is the union of the candidate peerers corresponding to such peering specifications. If a peering specification is a `peering-set`, it is recursively expanded into its members. If it involves information about routers (e.g., interfaces, `inet-rtrs`, `rtr-sets`), they are removed. We keep only AS names, `as-set` names, set operators, and the keyword `AS-ANY`. The resulting expression is evaluated by using Peval. The output of Peval, consisting of a set of ASes, contributes to the set of candidate peerers.

A structured policy is a policy that has `except` and/or `refine` operators. In this case, *import*(A) is still the union of several candidate peerers, but such candidate peerers are determined in a different way. First, we extract the two arguments of the `except` (`refine`) operator, which are simple policies. Then, we process such policies as above, thus obtaining two sets of peerers. The union (intersection) of these two sets is our set of candidate peerers. If there are multiple `except/refine` expressions, we process them iteratively.

If an `aut-num` has many `import` attributes the above procedure is repeated for each one.

Finally, private ASes in *import*(A) are removed.

Some technical issues should be pointed out. For example, a peering specification may contain the `AS-ANY` keyword. `AS-ANY` is either used "alone" (e.g. `import .... from AS-ANY ....`) or in a structured policy. In the first case one could argue that there is an AS that has a peering with all the other ASes, which is clearly unrealistic. Hence, in this case we discard the peering specification. Else, if `AS-ANY` is used inside a structured policy, we apply the above algorithm. Last, observe that also `inet-rtr` objects may contain information about peerings. However, we do not consider such peerings meaningful unless they appear in an `[mp-]import`, `[mp-]export`, or `[mp-]default` attribute of an `aut-num`.

Table 3 shows the incidence of RPSL constructions in the specification of peerings.

`aut-num` **objects**	**Action**	**Uses Peval**	**Occurrences**
having a `default` attribute	Supported	No	4,851
having an `mp-import`, an `mp-export`, or an `mp-default` attribute	Supported	No	220
having a `peering-set` object in (*)	Supported	No	16
having an `as-set` object in (*)	Supported	Yes	939
having AS-ANY in (*) without further specifications	Discarded	No	660
having AS-ANY in (*) within a `refine` expression	Supported	No	24
having an `and`, an `or`, a `not`, or an `except` operator in (*)	Supported	Yes	5
having a `refine` or `except` expression in (*)	Supported	No	29
registering a private AS	Discarded	No	1
Private ASes in (*)	Discarded	No	86
`inet-rtr` objects having `peer` attributes	Discarded	No	217

Table 3: Incidence of different RPSL constructions in the specification of peerings. (*): an `[mp-]import`, an `[mp-]export`, or a `[mp-]default` policy.

Table 4 shows the number of peering candidates extracted from the registries.

ripe	342995	bell	974	risq	67	look	16	soundinternet	8
verio	118999	fastvibe	968	sinet	50	eicat	15	gw	8
radb	19309	level3	558	ottix	38	nestegg	14	digitalrealm	8
apnic	13979	epoch	439	jpirr	38	mto	14	univali	6
reach	9402	dodnic	389	csas	36	area151	14	gts	2
savvis	1593	gt	219	retina	22	openface	10	easynet	2
arin	1233	rogers	134	crc	22	bendtel	10	aoltw	2
altdb	1068	host	79	bcnet	18				

Table 4: Peering candidates per registry.

7 Constructing a Peering Graph

Once a peering candidate has been extracted from the IRR, it is classified according to the following two categories. Let A and B be the two ASes participating in the peering candidate. $A \xrightarrow{E} B$ represents the fact that A registered an export policy allowing BGP announcements to be sent to B. In turn, $A \xrightarrow{I} B$ indicates that B registered a policy according to which B accepts incoming announcements from A. The peering candidates are also tagged with the registries from which they have been extracted.

At this point, the peering candidates are used to determine whether there actually is a peering between two ASes. For example if, for two ASes A and B, we have found four peering candidates of type $A \xrightarrow{E} B$, $A \xrightarrow{I} B$, $A \xleftarrow{E} B$, $A \xleftarrow{I} B$, it means that both A and B have fully considered their partner in the peering. Hence, we call this peering "full peering" (A—B). Of course, there can be cases when the policies describe a peering only partially. For example, we might have only $A \xrightarrow{E} B$, $A \xrightarrow{I} B$, in which case the announcements from A to B are described in the policies, while there is no evidence of policies allowing announcements from B to A. We call this situation "half peering" ($A \overset{1/2}{\text{---}} B$).

Table 5 shows all the possible relationships between two ASes. The column Peering Type associates a symbol to each possible situation. The column # of Peerings counts the peerings of each category. The column Single Registry reports the percentage of cases where all the candidate peerings contributing to the peering are in a single registry. We detail such percentage for the RIPE registry. A self peering refers to an AS that registers a peering with itself.

The peering types of Table 5 can be used to construct Internet topologies with different levels of confidence.

8 Comparison with Previous Work

In order to compare the peerings discovered with our techniques with those discovered with previous approaches we ran on the same data set we used in our experiments the piece of code that RIPE uses for peering extraction in the RRCC service [23, 11]. The

Policy Type				Peering Type	# of Peerings	Single Registry	RIPE Only
$A \xrightarrow{E} B$	$A \xrightarrow{I} B$	$A \xleftarrow{E} B$	$A \xleftarrow{I} B$				
√	√	√	√	$A \overline{} B$	42,599	96,7%	94.6%
	√	√	√	$A \overset{3/4\neg E}{\overline{}} B$	1,373	84.6%	80.3%
√		√	√	$A \overset{3/4\neg I}{\overline{}} B$	1,013	88.8%	82.2%
√				$A \overset{1/4E}{\overline{}} B$	34,155	100%	7.7%
	√			$A \overset{1/4I}{\overline{}} B$	13,997	100%	23.7%
√	√			$A \overset{1/2}{\overline{}} B$	114	90.4%	57.9%
√		√		$A \overset{1/2E}{\overline{}} B$	19	78.9%	47.4%
√			√	$A \overset{1/2AB}{\overline{}} B$	143,342	100%	58.4%
	√		√	$A \overset{1/2I}{\overline{}} B$	51	72.5%	66.7%
Total (including Self-Peerings)					236,663		
Self-Peerings					195		

Table 5: Classification of the peerings discovered in the IRR

peerings obtained in such a way can be considered analogous to our peering candidates. By using the RIPE code we obtained 295,587 *RRCC peering candidates*, that are much less than our overall amount of 512,758 peering candidates (see Tab. 4). By aggregating the RRCC peering candidates with the method of Section 7 we obtained 108,521 *RRCC peerings*. Again, much less than our 236,663 peerings (see Tab. 5). Further, there are 102 RRCC peerings that we did not find. We discovered that 100 of them involve private ASes and the remaining 2 come from an incorrect processing of the RRCC code of the `and` operator. A comparison with [16] gave similar results.

Comparing our results with the ones presented in [19, 25, 24] is not easy. In fact, they refer to the versions of the IRR of 04/07/04, 10/24/04, and 06/22/03, respectively. To the best of our knowledge, no repository is available with IRR historical data. We have a repository of such data in the interval described in Fig. 1 but, unfortunately, such interval does not cover the above dates. The authors of [19] provide the peerings extracted from the IRR on 04/07/04. The work in [25] is supported by a Web site providing several files of peerings. It is updated on a daily basis, yet the peerings discovered in the IRR are unavailable. Also the work in [24] has a Web site [5] that allows to interactively explore the peerings detected on 11/08/05. Again, such date is not covered by our archives.

Hence, only a rough comparison is possible. The topology of [19] reports 56,973 peerings while [25] reports the discovery of 70,222 peerings. Both refer to the RIPE registry only. Paper [24] reports 127,498 peerings referred to the entire IRR. All such figures are very far from our results.

9 Future Work

We think that the data contained into the IRR are a unique source of valuable information and therefore consider the results presented in this paper as a starting point and a necessary premise for future research on the topic. The availability of an effective peering extraction technique opens, at least, the following perspectives.

We can study how BGP announcements would spread over the Internet according to the policies registered in the IRR. This would give a better estimate of the consistency of IRR data against actual routing and would bring about the opportunity of performing specific actions on the IRR to improve its consistency. One could even emulate the entire (or a significant portion of) Internet by configuring virtual routers with the policies documented in the IRR. We believe this could be of great help in understanding the behaviour of Internet and in forecasting, preventing, or debugging abnormal or unsafe routing scenarios.

10 Acknowledgements

We would like to thank Lorenzo Colitti and Guido Drovandi for their precious contribution to this work.

References

[1] ARIN. `http://www.arin.net/`.

[2] Internet Routing Registry Daemon (IRRd). `http://www.irrd.net/`.

[3] Internet Routing Registry Toolset (IRRToolSet). `http://www.isc.org/index.pl?/sw/IRRToolSet/`.

[4] LEVEL3. `http://rr.level3.net/`.

[5] Nemecis. `http://ira.cs.ucr.edu:8080/Nemecis/`.

[6] Oregon Route Views Project. `http://www.routeviews.org/`.

[7] Overview of the IRR. `http://www.irr.net/docs/overview.html`.

[8] RADB database. `ftp://ftp.radb.net/radb/dbase/`.

[9] RIPE. `http://www.ripe.net/`.

[10] RIPE database. `ftp://ftp.ripe.net/ripe/dbase/`.

[11] Routing Registry Consistency Check (RRCC). `http://www.ripe.net/projects/rrcc/`.

[12] The Internet Routing Registry: History and Purpose. `http://www.ripe.net/db/irr.html`.

[13] VERIO. `http://rr.verio.net/`.

[14] C. Alaettinoglu, C. Villamizar, E. Gerich, D. Kessens, D. Meyer, T. Bates, D. Karrenberg, and M. Terpstra. Routing Policy Specification Language (RPSL). IETF RFC 2622, June 1999. `http://www.ietf.org/rfc/rfc2622.txt`.

[15] L. Blunk, J. Damas, F. Parent, and A. Robachevsky. Routing Policy Specification Language next generation (RPSLng). IETF RFC 4012, March 2005. `http://www.ietf.org/rfc/rfc4012.txt`.

[16] A. Carmignani, G. Di Battista, W. Didimo, F. Matera, and M. Pizzonia. Visualization of the High Level Structure of the Internet with Hermes. *Journal of Graph Algorithms and Applications*, 21(6):53–61, 2002.

[17] D. Karrenberg, G. Ross, P. Wilson, and L. Nobile. Development of the Regional Internet Registry System. *Internet Protocol Journal*, 4(4):17–29, 2001.

[18] S. Kerr. RIPE Database Inconsistencies. RIPE Meeting 43, September 2002. `http://www.ripe.net/ripe/meetings/ripe-43/`.

[19] P. Mahadevan, D. Krioukov, M. Fomenkov, B. Huffaker, X. Dimitropoulos, K. Claffy, and A. Vahdat. The Internet AS-Level Topology: Three Data Sources and One Definitive Metric. *ACM SIGCOMM Computer Communication Review*, 36:2006, 2006.

[20] D. Meyer, J. Schmitz, C. Orange, M. Prior, and C. Alaettinoglu. Using RPSL in Practice. IETF RFC 2650, August 1999. `http://www.ietf.org/rfc/rfc2650.txt`.

[21] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). IETF RFC 1771, March 1995. `http://www.ietf.org/rfc/rfc1771.txt`.

[22] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). IETF RFC 4271, January 2006. `http://www.ietf.org/rfc/rfc4271.txt`.

[23] J. Schmitz, E. Gunduz, S. Kerr, A. Robachevsky, and J. L. Silva Damas. Routing Registry Consistency Check (RRCC). RIPE Document 201, December 2001. `ftp://ftp.ripe.net/ripe/docs/ripe-201.txt`.

[24] G. Siganos and M. Faloutsos. Analyzing BGP Policies: Methodology and Tool. In *IEEE INFOCOM*, March 2004.

[25] B. Zhang, R. Liu, D. Massey, and L. Zhang. Collecting the Internet AS-Level Topology. *ACM SIGCOMM Computer Communication Review*, 35(1):281–311, 2005.