Vol. 46 No. 6 IPSJ Journal June 2005

Regular Paper

Improvement of Consistency among AS Policies in IRR Databases

Masasi Eto,[†] Youki Kadobayashi[†] and Suguru Yamaguchi[†]

This paper presents an architecture which investigates the consistency of AS policies in all of the publicly accessible Internet Routing Registry (IRR) databases in the world. We also propose an architecture to prevent the increase of inconsistency. Since inconsistency hampers the connectivity between ASs, the consistency of IRR databases is crucial for the stable operation of the Internet. Through our investigations, we have found that a significant proportion of AS policies are either specified incorrectly or outdated. Based on this observation, we propose and implement a system that detects these inconsistencies and notifies operators so that they can be corrected. Finally, we evaluate these systems.

1. Introduction

The Border Gateway Protocol (BGP) is crucial to the overall reliability of the Internet [1),2)], but faults in BGP have been known to disrupt large regions of the Internet. For example, in April 1997, AS7007 accidentally announced the route information to most of the Internet and disrupted connectivity for more than two hours [3)]. In April 2001, AS3561 propagated more than 5,000 improper route announcements from one of its downstream customers, again leading to global connectivity problems [4)~6)].

To alleviate this problem, the Internet Routing Registry (IRR) is required to be a tool which increases the efficiency of BGP network operation. IRR holds policies, written in Routing Policy Specification Language (RPSL) [7)], that are registered by operators of ASs. These policies contain information such as AS numbers, the maintainer of the AS, and routing policies.

However, operators generally view the IRR as an obscure and difficult system rather than a useful tool for network operations. This lack of understanding prevents the widespread use of the IRR.

One of the reasons for this skepticism is ascribed to inconsistencies in IRR databases. The router configurations generated from IRR databases cannot be trusted because they may contain inconsistencies which make communication between the ASs impossible.

In this research, we have investigated the consistency of AS policies in all the public IRR databases: 55 IRR servers in the world such as RIPE, RADB and IRRs mirrored by RADB.

A key finding of this investigation is that a significant proportion of the AS policies are either specified incorrectly or outdated. Based on this result, we propose a system which supports to reduce most of the observed inconsistency. Our proposal aims at stable and less labor-intensive Inter-domain routing with the IRR.

2. IRR

An IRR is a global Internet resource database that stores routing policies such as AS numbers and prefix information. An IRR consists of several objects (Route object, Aut-num object, Maintainer object, etc.). Policies registered in IRR are written in RPSL, which is designed to describe specific routing information by import and export sentences in Aut-num object. Operators can generate the vendor-specific router configurations from the policy data [8)].

2.1 IRR Operation

Unlike Domain Name Server (DNS), the organization who operates IRR server is not regulated. Therefore some Regional Internet Registries (RIRs), National Internet Registries (NIRs) and many Internet Service Providers (ISPs) operate their own IRR servers in the world. Especially, RADB, RIPE and APIRR operated by Merit Network Inc, RIPE NCC and APNIC respectively are known as the representative IRR services. They are classified roughly *Public IRR* and *Private IRR*. Public IRR makes registered information available to the public so that everyone can use the information to check some ISPs' routing information. Private IRR holds nondisclosure information such as routing information of ASs who form private peerings

† Graduate School of Information Science, Nara Institute of Science and Technology

between them.

Public IRRs form data mirroring each other and in current operation, data of most IRRs are accumulated to the representative IRR servers described above.

2.2 Registering to IRR

Operators of ASs can register information about their ASs to one or more IRR(s) at any time. On the other hand, some Local Internet Registries (LIRs) force their customer ASs to register information to their own IRRs, and some LIRs substitute registering information for the customers.

2.3 Problem of Current IRR Operation

However, in its current operation, it is difficult to keep IRR database consistent for the following reason. That is, to register correct routing information to IRR, operators need to check the consistency of them between each peer AS. However, in the recent Internet, an AS holds quite many peerings between many other ASs, so that operators have to bear a higher burden to check all of the consistency.

As a result, the database will contain many inconsistencies, and when router configurations are generated from this database, peer connectivity between ASs will be lost. Otherwise, an unintended link selection may occur.

On the other hand, IRRs do not hold all of the AS objects on the Internet, because operators of ASs are not forced to register their information in an IRR. This issue makes operators view an IRR as an incomplete database, causing a vicious circle which prevents the increased of utilization of IRRs[9].

3. Classification of the Inconsistency

In this section, we discuss the definition of inconsistencies that could prevent peer connectivity between ASs.

The inconsistencies are roughly classified into the following two types:

- Inconsistency in Routing Information Import:
  There are less description of routing information in export sentence or there are too much description of routing information in import sentence.
- Inconsistency in Routing Information Export:
  There are less description of routing information in import sentence or there are too much description of routing information in export sentence.

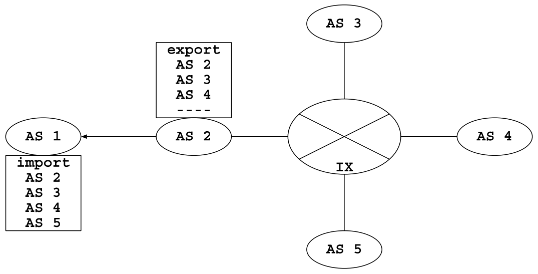

Fig. 1 Inconsistency in routing information import.

```
Aut-num: AS 1
as-name: EtoNet
...

import: from AS 2
        accept AS 2, AS 3, AS 4, AS 5
...
```

Fig. 2 Policy registered by AS 1.

In the following subsections, we first explain each type of inconsistency, and then describe classified inconsistencies in more detail.

3.1 Inconsistency in Routing Information Import

If one AS wishes to establish connectivity with other ASs and configures to import their routing information, and if the peer AS does not export their routing information to the AS, the AS cannot get connectivity to those ASs. We explain this problem as the *inconsistency in routing information import* in the following example.

As shown in **Fig. 1**, AS 1 and AS 2 operate under the contract that AS 2 provides transit for the traffic from AS 1 to AS 3, AS 4 and AS 5. According to this contract, the operator of AS 1 registered the policy shown in **Fig. 2** which is configured to import the routes for AS 2, AS 3, AS 4 and AS 5 from AS 2. On the other hand, the policy registered by the operator of AS 2 is shown in **Fig. 3**; in this policy, the route of AS 5 is missing by accident. The router configurations generated from these policies by RtConfig[10] are shown in **Figs. 4** and **5**. In these configurations, each network address presents the route of each AS. Note that in Fig. 4, AS1 imports routes of AS2, AS3, AS4 and AS5. On the other hand, in Fig. 5, AS2 exports only AS2, AS3 and AS4. If the operators commit these configurations to their routers as

```
Aut-num: AS 2
as-name: SaiNet
...

export:  to AS 1
         announce AS 2, AS 3, AS 4
...
```

Fig. 3 Policy registered by AS 2.

```
import proto bgp as AS 2 {
  192.168.2.0 masklen 24 exact;
       //route information of AS 2/
  192.168.3.0 masklen 24 exact;
       //route information of AS 3/
  192.168.4.0 masklen 24 exact;
       //route information of AS 4/
  192.168.5.0 masklen 24 exact;
       //route information of AS 5/
  all restrict;
}
```

Fig. 4 Configuration on a router in AS 1.

```
proto bgp aspath .* origin any {
  192.168.2.0 masklen 24 exact;
       //route information of AS 2/
  192.168.3.0 masklen 24 exact;
       //route information of AS 3/
  192.168.4.0 masklen 24 exact;
       //route information of AS 4/
  all restrict;
}
```

Fig. 5 Configuration on a router in AS 2.

they are, the router of AS 1 cannot receive the route of AS 5 so that AS 1 cannot establish connectivity with AS 5.

3.2 Inconsistency in Routing Information Export

If peer ASs configure to export the expected routing information, and if the AS does not import their routing information from the peer ASs, the AS cannot establish connectivity with those ASs. In the following example, we describe this problem as the *inconsistency in routing information export*.

In **Fig. 6**, assume that AS 1 and AS 2 registered the policies shown in **Figs. 7** and **8**. In this case, the transit provider (AS 2) registered the proper policy according to the contract. However, in the policy of AS 1, the sentence required to import the route of AS 1 is missing by accident. The router configurations generated from these policies are shown in **Figs. 9** and **10**. In this case, although AS2 exports routes of AS2, AS3, AS4 and AS5, AS1 imports only AS2, AS3 and AS4. As a result, AS 1 cannot establish the connectivity with AS 5.

Based on these premises, we have classified more details of these inconsistencies as shown in **Table 1**, in which *AS-SET* object is an aggregate of Aut-num object. Like an AS specifying another AS as a peer, an AS can also specify an AS-SET as a peer.

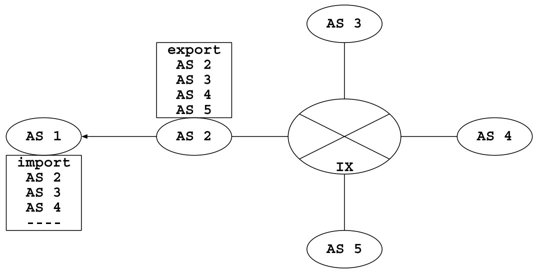

Fig. 6 Inconsistency in routing information export.

```
Aut-num: AS 1
as-name: EtoNet
...
import:  from AS 2
         accept AS 2, AS 3, AS 4
...
```

Fig. 7 Policy registered by AS 1.

```
Aut-num: AS 2
as-name: SaiNet
...
export:  to AS 1
         announce AS 2, AS 3, AS 4, AS 5
...
```

Fig. 8 Policy registered by AS 2.

```
import proto bgp as AS2 {
  192.168.2.0 masklen 24 exact;
       //route information of AS 2/
  192.168.3.0 masklen 24 exact;
       //route information of AS 3/
  192.168.4.0 masklen 24 exact;
       //route information of AS 4/
  all restrict;
}
```

Fig. 9 Configuration on a router in AS 1.

Table 1 Classification of inconsistencies.

Inconsistencies in routing information import	peer AS-SET does not exist on IRR database peer AS does not exist on IRR database peer AS does not export any routes to the AS peer AS does not export route which the AS imports
Inconsistencies in routing information export	peer AS-SET does not exist on IRR database peer AS does not exist on IRR database peer AS does not import any routes from the AS peer AS does not import route which the AS exports

```
proto bgp aspath .* origin any {
  192.168.2.0 masklen 24 exact;
       //route information of AS 2/
  192.168.3.0 masklen 24 exact;
       //route information of AS 3/
  192.168.4.0 masklen 24 exact;
       //route information of AS 4/
  192.168.5.0 masklen 24 exact;
       //route information of AS 5/
  all restrict;
}
```

Fig. 10 Configuration on a router in AS 2.

```
import:  from AS 2
   action pref = 10 ; med = 0 ;
   community.append(10250, {3561,10}) ;
   accept AS 2, AS 3, AS 4
```

Fig. 11 import sentence with action attribute.

3.3 Other Attributes

Although there are several other attributes and parameters in an Aut-num object, we focus on inconsistencies classified in Table 1 for this research.

In import and export attributes of an Aut-num object, RPSL defines *action* attributes that enable more detailed configurations, such as *MED*, *Local Preference* and *community* parameters, which are described in **Fig. 11**.

In this case, an action attribute sets a constraint on imported route prefixes for each parameter. Of course, it is possible that these parameters are inconsistent with the specification of the peer AS's action attribute.

One of inconsistencies that may disturb connectivity between peering ASs is the inconsistency caused by description of Well-Known Community in COMMUNITY path attribute. When an AS sets *NO_EXPORT* COMMUNITY path attribute to a certain export route, the route will not be announced to the peer AS. On the other hand, if the peer AS configured the policy to import the route, their policies are inconsistent and the connectivity may be disturbed.

In this research, to simplify the classification of inconsistencies, we focus on only the inconsistencies described in Section 3.1 and Section 3.2. Then we will consider the Well-Known Community problem as the future work.

Therefore, we focused on inconsistencies classified in Table 1 in this research.

4. Methodology

In this research, we aim to establish a mechanism to conduct accurate inspection on a global scale with high efficiency. To accomplish this goal, we need the following three systems: a system for advance inspection of a policy's consistency before it is registered; a system to aggregate all IRR databases in the world; and a system to perform a total inspection of consistency in the aggregated IRR databases.

4.1 Advance Inspection of Consistency

We propose and implement a system to investigate the consistency of AS policies in IRR databases globally. When an operator intends to register his/her AS's policy, it is difficult to check whether the policy is consistent between peering ASs. This fact may cause inconsistencies to arise between policies of the operator's AS and peering ASs, and we therefore need a system to inspect the consistency of the policy before it is registered with the IRR database.

4.2 Aggregation of IRR Databases

We decided to collect and aggregate all policies of IRR databases in the world, to inspect consistency in more detail.

When any organization gains an AS number from the RIR, it needs to register its policy with the IRR database. However, the IRR server is managed by any organizations as we mentioned in Section 2.1. Therefore, the policies of each AS are distributed to each of the IRRs. To inspect the consistency of AS policies, we have to collect and store them in one place. Because the databases are open to the public, we decided to collect all 55 of the accessible IRR databases including RIPE, RADB and APNIC. In this paper, we call the collected databases the *Unified IRR Database*.

4.2.1 Unified IRR Database

In this section, we describe the algorithm of the Unified IRR Database. Basically, we extract all of aut-num objects from each IRR database and store them in the Unified IRR Database. Aut-num object in the Unified IRR Database consists of AS number, import sentences and export sentences. At this moment, we should notice that there are duplicate aut-num objects registered into multiple IRRs. In this case, we merge them based on the following rules.

(1) Restructure the duplicate aut-num objects as a aut-num object and store import and export sentences in it.

(2) If multiple sentences import *different* routes from *same* peer AS, store both sentences in separate. Treat multiple export sentences as same.

(3) If multiple sentences import *same* routes from *same* peer AS and they have *different* actions, check the updated date and adopt the most up-to-date sentence.

(4) In the above step, if the updated dates of both sentences are same, determine they are inconsistent and discard them. Because we can not decide automatically which sentence is the valid direction.

We have considered another algorithm that is to adopt the most up-to-date object and discard the other instead of merging them. However, we decided to merge them because the operator might divide the AS's information into several parts and register them into multiple IRRs intentionally.

4.3 Total Inspection of Aggregated IRR Databases

By aggregating IRR databases as described above, we can perform a total inspection of all the accessible IRR databases currently in operation. Then, as described in Section 4.1, we can make clear the need for advance inspection of the policies.

5. Designing the Inspection System

Before implementing these systems, we have to discuss the adequacy of our methodology compared with alternative proposals for aggregation of IRR databases. Then we have to consider the synchronization of the Unified IRR Database and the Public IRRs.

5.1 Using Whois Query

As an alternative proposal, we could use whois query to perform advance inspection. IRRd has whois interface to provide us registered ASs' information. When we send a query that specifies a particular AS number to an IRR server, the IRRd sends back to us a response that includes the specified AS's information[8]. Using this function, we can implement the following system: when the system receives a query of policy inspection, it sends whois queries to IRRs, then it can collect peer ASs' policies.

On the other hand, as described in Section 4.2, each AS's policy is deployed in IRRs that are also repeated throughout the world. Besides that, there is no appropriate way to know which IRR the required policy is in, so that it is difficult to perform advance inspection if all the IRR databases are not integrated.

For this reason, we decided to construct the Unified IRR Database instead of using a whois query.

5.2 Database Synchronization

We decided to update the Unified IRR Database every thirty minutes for the following reason. In current operation of IRR, many IRRs mirror their information each other every thirty minutes. On the other hand, there are some IRRs that mirrors once a day. Therefore, the Unified IRR Database updates its database every thirty minutes in accordance with the shortest cycle of the other IRRs. In this case, the time lag between the Unified IRR Database and a certain IRR may be one hour at a maximum. Considering that there are IRRs that mirrors once a day, we determine that this time lag does not lead to critical problems of inspection.

6. Proposed System and Implementation

To investigate and prevent the inconsistencies defined in Table 1, we propose *Policy Check Server*, which consists of three main components as follows.

- To inspect consistency between ASs, we need the complete set of policies for all the accessible IRR databases in the world. Therefore, we constructed the Unified IRR Database, which includes those policies, by *DBGenerator*.
- *Database Checker* inspects how many in-

```
[ specify peering AS ]
    extract import, export sentences from input AS object ;
    if (the peering AS (or AS-SET) exists on database) {
        create "Autnum" object as a peering AS ;
    } else {
        warn as an inconsistency "Peer AS (AS-SET) doesn't exist on IRR database" ;
    }
[ inspection of import sentence ]
    for (number of import sentence of input AS) {
        for (number of export sentence of peer AS) {
            if (the export sentence of peer AS specify input AS as a peer) {
                if (the sentence doesn't export required routes) {
                    warn as an inconsistency  "Peer AS doesn't export route which the AS imports ;
                }
            } else {
                warn as an inconsistency "Peer AS doesn't export any routes to the AS ;
            }
        }
    }
[ inspection of export sentence ]
    for (number of export sentence of input AS) {
        for (number of import sentence of peer AS) {
            if (the import sentence of peer AS specify input AS as a peer) {
                if (the sentence doesn't import required routes) {
                    warn as an inconsistency "Peer AS doesn't import route which the AS exports ;
            } else {
                warn as an inconsistency "Peer AS doesn't import any routes from the AS" ;
            }
        }
    }
```

Fig. 12 Inspection algorithm.

consistencies exist on the Unified IRR Database.

- *Policy Checker* inspects whether a policy which the operator of an AS is about to register is consistent with the policies of its peering ASs.

6.1 DBGenerator

DBGenerator collects policies from IRRs, and extracts import and export sentences. Most policies are mirrored from RIPE, RADB and APNIC. They are available to everyone, with constraints about redistribution. DBGenerator then injects the AS objects into the database, which is constructed by PostgreSQL database.

6.2 Database Checker

Database Checker inspects how many inconsistencies exist on the Unified IRR Database. It inspects all the policies in the Unified IRR Database according to the algorithm shown in **Fig. 12**.

The algorithm consists of three phases.

(1) Database Checker specifies the peer AS by import or export sentences, and holds the peer AS as an AS object. If the peer AS does not exist on the Unified IRR Database, Database Checker records this fact as an inconsistency.

(2) Database Checker compares import sentences of the input AS and export sentences of the peering AS. If the peering AS does not have an export sentence which specifies the input AS as a peer like this:
   - export: <u>to *input AS*</u> announce AS 3 ... ,

   Database Checker records this fact as an inconsistency. Otherwise, Database Checker determines whether the peering AS exports the route prefix which the AS intends to import from. If it does not, Database Checker records the fact as an inconsistency. In the next phase, Database Checker compares the export sentences of the input AS and the import sentences of the peering AS.

(3) Database Checker outputs the collected inconsistencies to a log file.

As we explained in Section 2, IRRs do not hold all of the AS objects in the Internet. Regarding this issue, the following situation can be assumed.

Since an AS imports 50 route prefixes from its peer AS, if the peer AS is not registered with any IRR, it is assumed that Policy Checker issues 50 warnings for every route prefix. However, the inconsistencies are based on a single factor: the peer AS is not registered with any IRR. To eliminate these redundant warnings, we bind up these inconsistencies in one inconsistency, which is "peer AS does not exist on IRR database". Policy Checker is designed to detect this inconsistency using the algorithm shown above (Fig. 12).

6.3 Policy Checker

Policy Checker gives an operator the opportunity to inspect the policy which he/she intends to register with an IRR database. Policy Checker keeps all of the latest entries of an IRR database as the Unified IRR database, which is made by DBGenerator, so that it is suitable for Policy Checker to be managed inside an IRR server.

The flow of the process is as follows.

(1) The policy input by the operator is transmitted to Policy Checker.

(2) Policy Checker creates an AS object from the input policy and transmits it to Database Checker.

(3) Database Checker then inspects the consistency between input policy and the

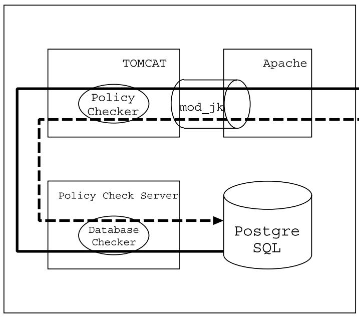

Fig. 13 Basic components of Policy Checker.

Table 2 List of objects and attributes.

Objects / Attributes	Number
aut-num object	11,702
as-set object	4,543
import sentence	98,726
export sentence	96,094

peer AS's policy, as described in Section 6.2.

(4) Database Checker returns the collected inconsistencies to Policy Checker.

(5) Policy Checker generates an HTML document from the result of the inspection, and displays it on the operator's web browser.

The process of inspection starts on a web-based interface which is deployed as a Java Servlet on TOMCAT (Web application server). If any inconsistencies are detected, Policy Checker displays warnings on the operator's web browser. The operator of the AS may then correct the corresponding entries and register the consistent policy. The basic composition is shown in **Fig. 13**.

7. Analysis of Inspection Results

We built up the Unified IRR Database from 55 public IRRs such as RIPE, Level3, RADB, Cable and Wireless, APNIC, Verio and so on. Most of these databases are mirrored by RADB, so that we obtained them from RADB IRR server. The list of the collected objects and attributes in the Unified IRR Database is shown in **Table 2**.

As a result of our investigation, we have found that 64.8% of the 11,697 registered ASs in IRRs have at least one inconsistency, as shown in Table 1 for IRR databases. These results are shown in more detail in **Fig. 14**, which indicates that there is variation in the number of inconsistencies according to the AS number. In

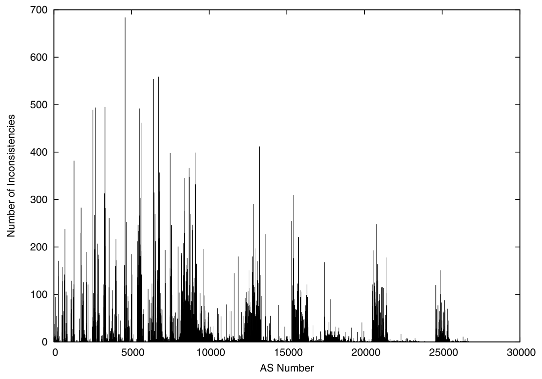

Fig. 14 Number of inconsistencies in each AS.

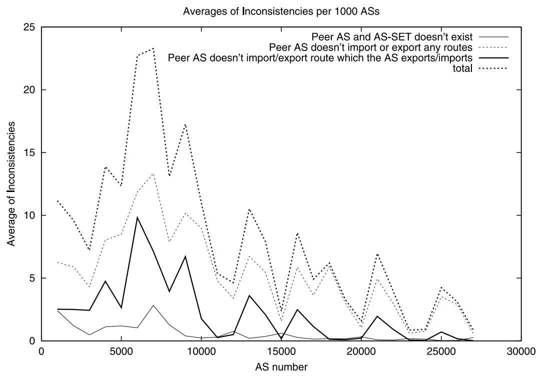

Fig. 15 Averages of inconsistencies per 1,000 ASs.

other words, inconsistency decreases as the AS number becomes larger.

We assume that ASs who have smaller AS numbers have many inconsistencies because of following reason. AS numbers are assigned in an order from smaller one to larger one by RIRs, therefore, smaller AS numbers are assigned earlier time than larger AS numbers. The difference of the assigned time between smaller and larger AS numbers may affect the numbers of the inconsistencies. In other words, it is thought that an AS which has a smaller AS number tends to have many peers, so that the AS has many import or export sentences and many inconsistencies.

In Fig. 14, we divided the AS numbers into every 1,000 numbers and calculated the average of the number of inconsistencies in each slot. As the result, **Fig. 15** shows the averages of inconsistencies per 1,000 ASs. By this figure, it became clear that old ASs (ASs which have smaller AS numbers) have much inconsistencies than newer ASs.

Figure 15 also shows the share of each type of the inconsistencies classified in Table 1. One of

Table 3 Details of inconsistencies.

	Classification	Number of inconsistencies	Rate (in 194,820)
1	Peer AS-SET does not exist on IRR database	482	0.2%
2	Peer AS does not exist on IRR database	7,971	4.0%
3	Peer AS does not export any routes to the AS	36,333	18.6%
4	Peer AS does not import any routes from the AS	34,710	17.8%
5	Peer AS does not export route which the AS imports	11,436	5.8%
6	Peer AS does not import route which the AS exports	17,753	9.1%
	Total	108,685	55.8%

the notable features of this figure is that inconsistency of *Peer AS doesn't import/export route which the AS exports/imports* increases as the AS number becomes smaller. This fact means that although many old ASs specify peer ASs in their routing policy, they do not import or export necessary routes. We suppose that old ASs may have many peer ASs, so that it is difficult to describe the correct routing information.

Details of the inconsistencies in all import and export sentences are shown in **Table 3**, which displays inconsistencies between peering ASs. In Table 3, the "Rate" column shows the rate of each inconsistency in all 194,820 of the *import* and *export* sentences. The two categories *Peer AS does not export any routes to the AS* and *Peer AS does not import any routes to the AS* constitute 20% of all the import and export sentences.

In other words, although a particular peer AS exists in the IRR database, the peer AS does not specify the AS as a peer. RPSL is designed to describe the routing configuration, particularly for import and export sentences. Although operators can increase their efficiency of operation on the BGP network by generating router configurations from IRR database automatically, we found out that this functionality is hardly used at all.

We also suppose old ASs have many inconsistencies because of the following reason. That is, there is a possibility that old ASs do not maintain objects in IRR databases any more. To prove this assumption, we need to examine updated dates of old aut-num objects that have especially many inconsistencies in the future.

8. Evaluation

8.1 Database Checker

Database Checker spent 2,107.967 seconds to inspect the whole database. Since the number of registered Aut-num objects is 11,697 entries, it takes an average of 0.18 seconds to inspect each Aut-num object. However, this figure is just an average for each inspection, and actually the time for each inspection increases linearly with the number of import and export sentences. It is thought that the number of policies in IRR databases will increase, so that it is necessary to reduce the time for the inspection.

Because Database Checker sends several queries to PostgreSQL database for every import and export sentence while it inspects the policy, it is clear that the majority of the time required for the inspection process is spent for disk I/O. In future work, therefore, we will improve the performance of Database Checker by optimizing the algorithm to reduce the number of physical disk I/Os.

8.2 Policy Checker

Policy Checker can inspect consistency precisely, because it uses the same methods as Database Checker. With this fact and the Consistency Chain (see Section 10.1.1), it is possible to prevent increases of inconsistency in IRR databases.

9. Related Work

The Routing Registry Consistency Check (RRCC) project[9] reports inconsistency between IRR databases and the real Internet. Tools which detect inconsistencies are available on the Web.

By inconsistency, they mean route prefixes which are not advertised on the real Internet but are registered on the IRR database, and route prefixes which are not registered on the IRR database but are advertised on the real Internet.

On the other hand, our research detects inconsistency among the registered policies. Since both of these areas of research aim to improve the integrity of IRR databases by correcting their inconsistencies, these two studies complement each other.

10. Conclusion and Future Work

A mechanism for preventing increases of inconsistency in IRR database records has been presented, which we call Policy Check Server. Policy Check Server consists of two components, *Policy Checker* and *Database Checker*.

We defined inconsistency as comprising two categories, inconsistency in routing information import and export. Both can potentially disrupt the connectivity between peering ASs.

Based on this classification, we proposed Policy Check Server. Policy Checker gives an operator the opportunity to inspect the policy which he/she intends to register on an IRR database, and Database Checker is a system to investigate the consistency of AS policies in all accessible IRR databases in the world.

As a result of the investigation by Database Checker, we have found that 64.8% of ASs have inconsistencies. We advocate that the operator of a particular AS should take the consistency between other ASs' policies into consideration before registering it on IRR.

10.1 Future Work

In the near future, we intend to apply Policy Check Server to JPIRR, an IRR server maintained by JPNIC, and provide a service to inspect consistency.

Moreover, we will consider the detail classification of inconsistencies such as Well-Known Community problem as we mentioned in Section 3.3.

We also need to consider the aggregation algorithm of the Unified IRR Database. In this paper, we discarded duplicated sentences on multiple IRR databases. We discuss other algorithms that takes in both of duplicated sentences.

10.1.1 Consistency Chain

By correcting inconsistencies between peering ASs, we believe that it is possible to exchange route information between ASs that are not directly peering. Eventually, it will be also possible to improve the consistency of all IRR databases.

For example, consider the situation shown in **Fig. 16**. In this situation, the requirement is to give AS 1 connectivity to AS 2 and AS 3. To complete this requirement, each AS has to declare that it will import or export expected routes.

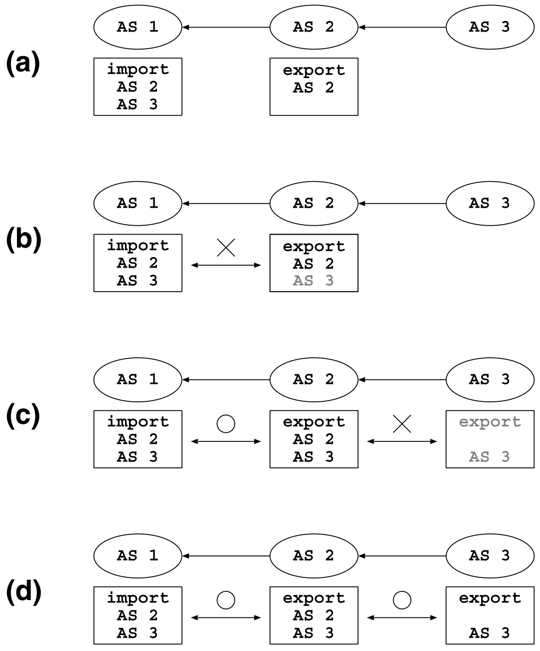

Fig. 16 Consistency chain.

(1) At the initial state (Fig. 16(a)), AS 2 does not export the route of AS 3 to AS 1. Furthermore, AS 3 does not export the route of AS 3 itself to AS 2. At this state, the route of AS 3 is never transmitted to AS 1, so that AS 1 cannot establish connectivity to AS 3.

(2) At this state, if operators use Policy Checker, it tells them that AS 2 does not export the expected route to AS 3, so that the operator of AS 2 would be able to correct the corresponding entry (Fig. 16(b)).

(3) However, at the next state (Fig. 16 (c)), Policy Checker tells the operator of AS 3 that AS 3 does not export any routes to AS 2. On this warning, the operator of AS 3 would be able to subjoin an entry properly.

(4) As a result, the policies of each AS are corrected and AS 1 is able to receive the route of AS 3 (Fig. 16 (d)).

Finally, connectivity between AS 1 and AS 3 is established. This connectivity is established only when operators use Policy Checker and correct the corresponding entries. Thus, by using Policy Checker, it is possible to check consistency between ASs that are not directly peering.

In the near future, we intend to investigate this functionality of *Consistency Chain* in all accessible IRR databases.

Acknowledgments The authors would like to thank members of JPNIC IRR Planning Team for many constructive discussions. The authors also wish to thank Professor Ian R.L. Smith for many advices for editing this paper.

References

1) Rekhter, Y. and Li, T.: A border gateway protocol 4 (bgp-4) (Mar. 1995). http://www.ietf.org/rfc/rfc1771.txt
2) Huitema, C.: *Routing in the Internet*, SHOEISHA (2000) (in Japanese).
3) Misel, S.A.: Wow, AS7007!, NANOG mail archives (Apr. 2001). http://www.merit.edu/mail.archives/nanog/1997-04/msg00340.html
4) Farrar, J.: C&W Routing Instability, NANOG mail archives (Apr. 2001). http://www.merit.edu/mail.archives/nanog/2001-04/msg00209.html
5) Mahajan, R., Wetherall, D. and Anderson, T.: Understanding bgp misconfiguration, *SIGCOMM'02* (Aug. 2002).
6) Griffin, T.G. and Wilfong, G.: On the correctness of ibgp configuration, *SIGCOMM'02* (Aug. 2002).
7) Alaettinoglu, C., Villamizar, C., Gerich, E., Kessens, D., Meyer, D., Bates, T., Karrenberg, D. and Terpastra, M.: Routing policy specification language (rpsl) (June 1999). http://www.ietf.org/rfc/rfc2622.txt
8) Hori, Y., Ikenaga, Z., Kadobayashi, Y. and Goto, S.: *Interconnection of Networks*, Iwanami Shoten (2001) (in Japanese).
9) Gunduz, E., Kerr, S., Robachevsky, A. and Damas, J.L.S.: Routing registry consistency check, Technical report, RIPE NCC (Dec. 2001).
10) Routing Arbiter Project. RAToolSet. http://www.isi.edu/ra/RAToolSet/

(Received January 5, 2004)
(Accepted March 1, 2005)
(Online version of this article can be found in the IPSJ Digital Courier, Vol.1, pp.216–225.)

Masasi Eto received his LL.B degree from Keio University, Tokyo, Japan, in 1999, received the MS degree from Nara Institute of Science and Technology (NAIST), Nara, Japan, in 2003. He is currently a Ph.D. student in NAIST. His research interests include auto-configuration of the Internet working and secure web applications.

Youki Kadobayashi received his Ph.D. degree in Computer Science from Osaka University in 1997. He is currently an Associate Professor in the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. His research interests include content internetworking, overlay networks, quality of services in the application-layer, middleware security, and secure operating systems.

Suguru Yamaguchi received the Master of Engineering degree in Computer Science from Osaka University, Osaka, Japan, in 1988, received the Doctor of Engineering degree in Computer Science from Osaka University, Osaka, Japan, in 1991. He is a professor of Nara Institute of Science and Technology. He has been also a member of WIDE Project, since its creation in 1988, where he has been conducting research on network security system and other advanced networkings for wide area distributed computing environment.