RPSLyzer: Characterization and Verification of Policies in Internet Routing Registries

Sichang He
University of Southern California
Los Angeles, USA
sichangh@usc.edu

Italo Cunha
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
Duke Kunshan University
Kunshan, China
cunha@dcc.ufmg.br

Ethan Katz-Bassett
Columbia University
New York, USA
ethan@ee.columbia.edu

Abstract

The Routing Policy Specification Language (RPSL) enables operators to specify routing policies in public registries. These policies contain information for traffic engineering, troubleshooting routing incidents, and automatically configuring route filters to improve security. RPSL information is also valuable for researchers to better understand the Internet. However, the RPSL's complexities make these policies challenging to interpret programmatically. We introduce RPSLyzer, a tool that can parse and interpret 99.99% of RPSL policies. We use RPSLyzer to characterize the RPSL policies of 78,701 Autonomous Systems (ASes) and verify 779 million BGP routes against these policies. We find RPSL usage varies widely among ASes, identify common RPSL misuses that explain most route verification failures, and offer operators recommendations to improve RPSL usage.

CCS Concepts

• **Networks** → **Network management**; Public Internet.

Keywords

RPSL; IRR; BGP; Interdomain Routing; Routing Policy; Verification

ACM Reference Format:
Sichang He, Italo Cunha, and Ethan Katz-Bassett. 2024. RPSLyzer: Characterization and Verification of Policies in Internet Routing Registries. In *Proceedings of the 2024 ACM Internet Measurement Conference (IMC '24), November 4–6, 2024, Madrid, Spain.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3646547.3689018

1 Introduction

The Routing Policy Specification Language (RPSL) [6, 12, 48] is a mechanism for network operators to specify routing policies. Although RPSL usage is not mandated, Autonomous Systems (ASes) may employ the RPSL to coordinate interdomain routing, troubleshoot incidents, or configure traffic engineering. Some transit providers require or suggest that their customers maintain routes they originate in the RPSL so that they can input them into tools like IRRToolSet [5] or BGPq4 [63] to automatically generate route filters [11, 34, 51]. Some IXPs require member ASes to specify peering policies in the RPSL to use the IXP's route servers [1, 20]. Networks may generate configurations for peering sessions by specifying neighboring networks' border routers, AS number (ASN), and BGP parameters such as route preference (LocalPref) in the RPSL.

In addition to its operational uses, the RPSL contains extensive unique data for researchers. For example, researchers have used RPSL information to identify AS links [26, 27, 59], business relationships [18, 36, 59], peering information [11, 16, 32, 46], stable routes [64], sibling ASes [9], and the semantics of BGP communities [41, 65]. More generally, Internet routes are shaped by routing policies, so understanding policies can unlock a better understanding of the Internet. However, routing policies are often proprietary and opaque. To circumvent this, decades of research focused on exploiting available data sources and developing complex techniques to infer (the impact of) routing policies, *e.g.,* [7, 49, 50]. Despite these efforts, the RPSL, an important data source in which ASes directly report aspects of their policies, remains understudied.

Interpreting the meaning of RPSL policies faces significant challenges due to the language's complexity. The RPSL has recursive syntax, indirect object references, and domain-specific semantics difficult to interpret. None of these challenges is fully overcome by existing tools. For example, BGPq4 [63], a popular up-to-date configuration generation tool, can only resolve single-term RPSL expressions; IRRd [45], a popular RPSL database and query server, at most breaks RPSL down to attribute-value pairs. More advanced yet discontinued tools, like Nemecis [59] and one by Di Battista *et al.* [11], focus on extracting specific high-level information like AS relationships, but lack generality. Finally, since the use of the RPSL is neither mandated nor standardized, its inconsistent use compounds its complexity: operators employ different subsets of the RPSL manually or with rudimentary tools, in different ways, to achieve different goals.

Contributions. To address these challenges, we open source RPSLyzer,[1] the first tool capable of interpreting the full meaning of 99.99% of public RPSL policies (Section 3). Besides parsing the RPSL and interpreting it to verify BGP routes, RPSLyzer exports an intermediate representation (IR) that captures policy semantics, facilitating the development of new tools that analyze the RPSL. We raise two research questions:

How are ASes using the RPSL? We employ RPSLyzer to characterize 78,701 ASes' RPSL usage (Section 4). We find that ASes' RPSL

[1] https://github.com/SichangHe/internet_route_verification

IMC '24, November 4–6, 2024, Madrid, Spain

ACM ISBN 979-8-4007-0592-2/24/11
https://doi.org/10.1145/3646547.3689018

usages vary widely, with 53.2% of ASes not declaring any policies in the RPSL, and the remaining ones describing their policies at widely different levels of detail. We find that most policies in the RPSL are simple and capture customer-provider or peer-to-peer relationships [24].

Do ASes comply with their policies in the RPSL? Using RPSLyzer, we verify if the sequences of ASes in routes observed by BGP collectors match the policies expressed in the RPSL (Section 5). For a large portion of interconnections present in BGP routes (40.4%), we find they cannot be verified using the RPSL due to missing information: ASes often do not declare their policies, and some policies refer to missing RPSL entries. Among the interconnections covered in the RPSL, we observe a high fraction of strict matches (29.3% of all interconnections), and explain most mismatches (19.0% of all interconnections) by six common mistakes we identified, *e.g.,* some transit ASes incorrectly specify that they distribute only their own routes when they mean to distribute both their own routes and routes received from their customers. Based on our findings, we provide recommendations for operators that ease the adoption of the RPSL and increase expressiveness (Sections 4 and 5), like using *route-set*s as a more straightforward way of specifying prefixes accepted on an interconnection.

Our contributions have the potential to catalyze the development of new tools that utilize the RPSL and drive its adoption by operators. Such development could improve the reliability and correctness of interdomain routing by reducing configuration errors that can result in incidents like suboptimal route propagation, outages, route leaks, or prefix hijacks [38, 43, 47, 68]. Research efforts would also benefit from an enhanced understanding of the Internet through increased tooling and RPSL adoption.

2 Routing Policies and the RPSL

We provide a brief overview of the RPSL according to [6, 12], focusing on its routing policy functionalities. The RPSL is object-oriented. It defines *objects*, each with a list of *attributes*. Notable RPSL object classes are the standalone *aut-num*, *route*, and *route6*; as well as *as-set* and *route-set*, which may recursively contain objects of their class. Internet Routing Registries (IRRs) [2, 60] store RPSL objects, and may support nonstandard extensions or lack support for standard objects and attributes.

The *aut-num* object provides an AS's information and enables operators to specify routing policies with its multivalued *import* and *export* attributes. We refer to each *import* or *export* attribute as a *rule*, with the following simplified syntax:

```
import: from <peering> [action <action>;] accept <filter>
export: to <peering> [action <action>;] announce <filter>
```

where each <parameter> has the following meaning:

- ***peering*:** Specifies the set of relationships between routers, commonly as a single ASN like AS X, meaning all BGP sessions between the declaring *aut-num* and AS X.
- ***action*:** Defines modifications applied to route attributes, like setting the LocalPref or attaching a BGP community.
- ***filter*:** Limits the set of IP prefixes to be accepted in *import* or announced in *export* rules.

For example, AS38639's rule below specifies that AS38639 has (one or more) BGP sessions with AS4713, over which it announces routes matching the AS-HANABI *as-set*.

```
export: to AS4713 announce AS-HANABI
```

This example shows a common practice by network operators: specifying *filter*s using an ASN or *as-set*. The RPSL has "predefined set objects" where an ASN defines a set of prefixes given by all *route* and *route6* objects whose *origin* attribute is AS X; an *as-set* defines a set of prefixes its member ASes originate. Alternatively, to include all routes originated by an ASN or *as-set* regardless of prefixes, operators can use an AS-path regular expression (regex), as shown in the compound rule below.

AS14595's rule below highlights RPSL's capability to express complex policies. See Appendix A for more compound examples.

```
% whois -h whois.altdb.net AS14595 | grep | format
mp-import:  afi any.unicast                              #1
                from AS13911                             #2
                accept ANY AND NOT {0.0.0.0/0, ::0/0}; #3
            REFINE
            afi ipv4.unicast                             #4
                from AS13911                             #5
                action pref=200;                         #6
                accept <^AS13911 AS6327+$>;              #7
```

The part of the rule before the *refine* accepts all non-default (#3) unicast IPv4 and IPv6 routes (#1) received from AS13911 (#2). The second part of the rule refines unicast IPv4 routes (#4) accepted by the first part. It only allows routes from AS13911 (#5) whose AS-path matches the AS-path regular expression (regex) in the *filter* (#7). The regex checks that a route is received from AS13911 and originated by AS6327. The rule enhances these routes' preference to 200 (#6), indicating that AS14595 prefers to route traffic towards AS6327's IPv4 prefixes through AS13911 over other potential neighbors. Such destination-specific traffic steering is an example of traffic engineering configurations that can only be captured precisely using compound RPSL rules.

Henceforth, we include the IPv6-compatible *mp-import* and *mp-export* attributes as well as *route6* objects when referring to their IPv4 counterparts (*export*, *import*, and *route*).

3 Parsing the RPSL

RPSLyzer parses all routing-related objects: *aut-num*, *as-set*, *route-set*, *peering-set*, *filter-set*, and *route* (details in Appendix B). For each object type, it decomposes all routing-related attributes, most importantly the *import* and *export* attributes (rules) of *aut-num* objects, into interpretable representations.

Features. RPSLyzer can resolve complex RPSL constructs such as hierarchical *as-set* names, prefix range operators, AS-path regex, and recursive object references. It supports less common features, such as the indirect "members by reference" of *as-set*s and *route-set*s that allow cooperative set maintenance by multiple network operators, and Structured Policies that combine multiple rules using the *except* and *refine* operators [6]. RPSLyzer even accommodates two cases of non-standard but common syntax (Appendix B). To our knowledge, no other tool has similar coverage of RPSL syntax.

Implementation and performance. RPSLyzer converts RPSL objects into an intermediate representation (IR) that captures their meanings. Our open-source Rust library defines the IR as a single

Table 1: IRRs used, grouped and ordered by priority.

IRR	Size (MiB)	*aut-num*	*route*	*import*	*export*
APNIC	929	20,680	988,665	15,615	15,905
AFRINIC	128	2,314	105,835	331	340
ARIN	33	3,047	94,365	6,940	7,359
LACNIC	11	1,847	12,759	0	0
RIPE	5,184	38,573	533,159	368,008	357,317
IDNIC	4	2,276	6,114	3,918	3,938
JPIRR	3	455	14,013	305	307
RADB	487	9,471	1,619,366	12,655	12,834
NTTCOM	176	549	375,836	921	1,016
LEVEL3	66	300	79,152	6,228	5,826
TC	16	4,205	25,333	3,911	3,964
REACH	15	2	20,238	3	3
ALTDB	8	1,680	29,517	3,241	3,143
Total	7,073	78,701	3,367,914	416,312	405,895

data structure with associated methods,[2] and can export it to JSON files for integration with other tools that leverage RPSL information. RPSLyzer parses the 13 IRRs listed in Table 1, totaling 6.9 GiB of data, and exports the IR, all in under five minutes on an Apple M1. This performance is enough to support future RPSL adoption.

4 RPSL Use in the Wild

IRR Datasets. We analyze publicly accessible IRR dumps from June 2023,[3] summarized in Table 1. For RPSL objects defined in multiple IRRs, we first prioritize based on registrar type (authoritative regional and national, RADB, and other databases), then by their sizes, as captured in Table 1.

We lack *aut-num* object definitions for national registrars like KRNIC and NIC.br due to unavailable IRR dumps. Many ASes delegated by ARIN are absent in ARIN's IRR.[4] Our verification remains sound despite the missing *aut-num*s: routes propagating through them are verified as if these ASes have no rules, and references to them from other objects are unaffected. The LACNIC IRR dump does not include any *import* or *export* rules.

RPSL adoption varies widely. Figure 1 shows that 35.2% of *aut-num*s contain no rules, 10.9% define at least 10 rules, and 0.13% (101 *aut-num*s) define over 1000 rules. We find no significant correlation between how many rules an AS defines and how many neighbors, customers, peers, or providers it has in CAIDA's AS-relationship database [15, 46]. This observation holds both for all ASes and when considering only transit ASes with at least 5 customers (not shown). For example, five Tier-1 ASes have zero rules and four have thousands; large cloud and content providers like Microsoft and Cloudflare operate with zero rules while others have very few (Figure 1); several small ASes define hundreds of rules.

Rules are simple. Almost all (98.4%) *peering* definitions comprise a single ASN or *ANY*, which allows imports from (or exports to) the specified AS or any AS, respectively. Most (94.5%) ASes with rules only specify simple *filter*s compatible with BGPq4 [63], a router configuration generator. In our tests, BGPq4 does not support *filter*s

[2]https://docs.rs/route_verification/latest/route_verification/ir/struct.Ir.html
[3]https://www.irr.net/docs/list.html
[4]ARIN operates an authoritative database at whois.arin.net. The use of ARIN's IRR database, operated separately at rr.arin.net, is entirely optional.

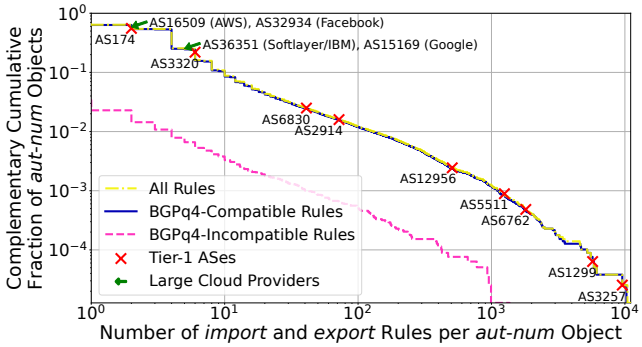

Figure 1: Complementary cumulative distribution function of the number of rules per *aut-num* object. Red crosses show high RPSL adoption variance across Tier-1 ASes; green arrows point to large CDNs and cloud providers operating with few (two or five) rules. Tier-1s and large CDNS with zero rules are not shown.

comprising *filter-set*, AS-path regex, BGP communities, Composite Policy Filters (with *AND*, *OR*, or *NOT*), or Structured Policies (with *refine* or *except*). As Figure 1 shows, the distribution of BGPq4-compatible rules per AS is quantitatively similar to the distribution of all rules. Although most rules are simple, compound rules exist and can improve conciseness or sometimes express policies simple rules cannot, *e.g.,* filtering by AS-paths; they could be more common, usable, and useful with better tooling support. Section 2 and Appendix A describe some examples of compound rules.

Filter definitions are unnecessarily indirect. Table 2 shows that 60.4% of *aut-num* and 31.7% of *as-set* objects are referenced in *filter* definitions. We find that most *filter*s are either an *as-set* (43.4%) or ASN (24.1%). As discussed in Section 2, this common practice relies on *route* objects of the referenced ASes to define the *filter*, thus requiring these (numerous) *route* objects to be kept up-to-date. More importantly, this practice limits each AS to only *one* set of prefixes and does not support, *e.g.,* exporting different sets of prefixes to different neighbors.

We recommend instead employing *route-set* objects for *filter*s, which directly specify sets of prefixes, obviating the need for maintaining *route* objects. *Route-set*s can additionally specify prefix ranges, so a single *route-set* can replace many *route* objects. Moreover, an AS can define multiple *route-set*s, *e.g.,* to selectively advertise different sets of prefixes to different upstreams. Unfortunately, Table 2 shows that a smaller fraction of *route-set*s are actually used compared to *as-set*s, even though both object types have similar modification dates (omitted) and are thus maintained similarly well.

Route objects require management. Our IRR dumps contain 3,904,352 *route* objects, corresponding to 3,367,914 unique prefix-origin pairs and 2,817,344 unique prefixes. This is about 3× more prefixes than in current global BGP tables at least due to (i) the need to create *route* objects for any potential prefix announcements, *e.g.,* temporary announcements for traffic engineering [54, 71], and (ii) accumulation of outdated and unmaintained *route* objects. Among these 2,817,344 prefixes, 697,269 (24.7%) have multiple *route* objects defined, among which 404,901 (58.1%) prefixes have *route*

Table 2: Numbers of objects defined and referenced in rules (overall, in *peering*s, or in *filter*s).

	aut-num	*as-set*	*route-set*	*peering-set*	*filter-set*
Defined	78,701	53,268	24,460	342	203
Referenced					
Overall	52,028	17,789	1,711	64	50
peering	37,595	2,519	—	64	—
filter	47,503	16,891	1,711	—	50

objects with different *origin*s. Furthermore, 469,003 (67.3%) prefixes have *route* objects defined by multiple operators, which is possible as IRRs allow operators to create *route* objects with any prefix and *origin*. This is legitimately used, *e.g.,* when transit providers create *route* objects for customer prefixes, but has also been exploited maliciously [33].

The number of prefixes with multiple *origin*s in *route* objects is 40× larger than the number of multi-origin prefixes observed in BGP dumps [28]. This complicates identifying the legitimate origin AS of a prefix with the RPSL. We bolster our recommendation for *route-set*s, which would not only enhance expressiveness but also lower maintenance burden.

Opaqueness of *as-set*s compromises correctness. Among 53,268 *as-set* objects across all IRRs, 7746 (14.5%) have no members, risking configuration errors if used in *import* or *export* rules. Moreover, 17,430 (32.7%) *as-set*s contain only one member AS, which could replace the whole set. The reserved keyword *ANY* appears in 3 *as-set* definitions, which is likely an error. A few (772, 1.4%) extremely large *as-set*s have more than 10,000 members. Some *as-set*s recursively contain other *as-set*s, forming directed graphs [59]; we find 13,602 such *as-set*s (25.5%), among which 3050 (22.4%) form loops and 3129 (23.0%) have depth 5 or more. Although the RPSL is "readable", manually tracking deeply recursive and potentially cyclic *as-set* definitions is laborious and error-prone, thus analyses greatly benefit from tooling like RPSLyzer.

RPSL errors. RPSLyzer found 663 syntax errors, 12 invalid *as-set* names, and 17 invalid *route-set* names. For example, it found an empty *as-set* named after the RPSL keyword AS-ANY, an anomaly that may disrupt RPSL analysis tools. Common syntax errors include out-of-place text, such as broken comma-separated lists, misplaced comments, invalid RPSL keywords in *import* and *export* rules, or plain typos. Besides syntax errors, data in the RPSL are sometimes outdated or outright incorrect [16]. These issues are compounded by registrars running their own IRR databases, which further complicates maintenance and can lead to inconsistencies across databases. To minimize the impact of inaccuracies in the RPSL, our analyses consider aggregate data from all major IRRs.

5 Verifying AS-Paths

We employ RPSLyzer to verify AS-paths observed in BGP dumps on June 23rd, 2023, from 60 RIPE and RouteViews BGP collectors [3, 4]. Our goal is to characterize how well AS-paths match public routing policies in the IRRs.

For each observed BGP route, we extract the AS-path $\mathcal{A}$ and prefix P, removing prepended ASes. We ignore 0.06% of single-AS routes directly exported by route collector peers as they have no inter-AS links to be verified. We also ignore 0.03% of routes whose AS-paths contain BGP AS-sets, whose use is discouraged [39, 42].

RPSLyzer examines each route $\langle P, \mathcal{A} \rangle$ from the origin AS of $\mathcal{A}$, verifying propagation between each AS pair. Concretely, for AS pair $\langle$Y, X$\rangle$ where AS Y accepts (imports) the route announced (exported) by AS X, we check if AS X has a matching *export* rule and if AS Y has a matching *import* rule. Our implementation queries and interprets the IR to find rules that match $\langle P, \mathcal{A} \rangle$, following the semantics in the RFCs [6, 48]. A rule defined by AS X (or AS Y) matching P's IP version and address type is a *strict match* to route $\langle P, \mathcal{A} \rangle$ if:

(1) AS Y (or AS X) matches the rule's *peering*.
(2) The prefix P and AS-path $\mathcal{A}$ match the rule's *filter*.

When matching *import* (or *export*) rules against a route import (export), we classify the verification status by applying the checks below in order. If there are multiple matches, the "best" rule with the earliest matching check is considered.

Verified. A strict match, as described above.

Skip. The rule is one of the 114 RPSLyzer does not or cannot handle (0.01% of 822,207 rules, see Appendix B). As a comparison point, BGPq4 does not handle 21,463 rules.

Unrecorded. The verification fails due to RPSL objects or rules being missing in the IRR, including (1) absence of *aut-num* object for the AS under check, (2) the *aut-num* object has zero *import* (*export*) rules when checking an import (export), (3) the *filter* refers to an AS that never appears as the *origin* of *route* objects, or (4) the rule refers to an unrecorded *as-set*, *route-set*, *peering-set*, or *filter-set*.

Relaxed. A match when the *filter* is *relaxed*, as detailed in Section 5.1.1.

Safelisted. The relationship between AS X and AS Y is *safelisted*, as detailed in Section 5.1.2.

Unverified. None of the above—a mismatch.

Performance. Verifying the 779.3 million routes in all 60 BGP dumps took 2 h 49 m and less than 2 GiB of RAM on a server with dual EPYC 7763 64-Core processors. RPSLyzer's high throughput allows processing large volumes of BGP updates such as those collected by BGP collectors.

5.1 Special Cases

By manually investigating rule mismatches in the wild, we identified six types of common RPSL misuses. Our checks leverage the business relationship between each pair of ASes. For this purpose, we rely on CAIDA's AS relationship inference database due to its availability and accuracy [46]. RPSLyzer programmatically checks these special cases in the order below.

5.1.1 Relaxed filters. We identify and *relax* checks for three common *filter* misuses that make sense from an operator's viewpoint. When a route matches a rule's *peering* but not its *filter*, we perform the relaxed checks below.

Export Self. More than half (6664, 64.4%) of transit ASes specify themselves as an *export* rule's *filter*. In strict RPSL, this would only allow exporting prefixes the AS itself originates. However, a transit AS X likely intends to export its routes *and* routes received from any of its customers (see Appendix E). Thus, we mark the export as relaxed if the previous AS on the AS-path is a customer of AS X.

Import Customer. Similarly, 3090 (29.8%) transit ASes specify a customer AS C in both an *import* rule's *peering* and *filter*. In strict RPSL, this would only allow importing the prefixes AS C originates. However, transit ASes likely intend to import *any* route received from AS C, including routes originated by AS C's customers. In such cases, we treat the *filter* as *ANY*, allowing any prefix on routes received from AS C.

Missing *route*s. Defining a *filter* with an ASN (or *as-set*) requires keeping the *route* objects originated by the AS (or *as-set*) up-to-date (Section 4). However, we relax this check if the AS is (or the *as-set* contains) the AS-path's origin.

5.1.2 Safelisted Relationships. We *safelist* three common relationships below that explain many policy mismatches.

Only Provider Policies. A few (46, 0.44%) transit ASes only specify rules for their providers, who may have mandated RPSL use to auto-generate BGP filters. Operators of such ASes do not maintain rules for the numerous peers and customers, so we safelist any route such ASes import from a peer or customer.

Tier-1 Peering. Tier-1 networks exchange routes by definition, but not all specify public RPSL policies. Thus, we safelist all route propagation between a pair of Tier-1 ASes.

Uphill Customer-Provider Propagation. Finally, we assume that providers import routes from their customers and that customers rely on their providers to reach the rest of the Internet. Therefore, we safelist any route to propagate *uphill* across a customer-provider link. These cases highlight opportunities where RPSL rules could inform route filters during upstream propagation to curtail route leaks and prefix hijacks [17]. We considered similarly safelisting downhill propagation but decided against it due to its limited utility: both providers and customers typically employ BGP filters that allow any prefix to propagate downhill.

5.2 Verification Results

RPSLyzer verified route imports and exports observed by RIPE RIS and RouteViews BGP collectors [3, 4]. Appendix C details a step-by-step route verification example. To demonstrate the class of analyses RPSLyzer enables, we report the verification statuses at three granularities: per AS, per AS pair, and per BGP route.

Most ASes use the RPSL consistently. Figure 2 shows verification statuses for route imports and exports per AS, ordered on the x-axis by correctness. The majority (61,746, 74.4%) of ASes have all imports and exports with identical statuses, captured in the graph as vertical bars with a single color. We identified 14.2% of ASes with 100% of propagation verified (yellow), 51.6% lacking RPSL information ("unrecorded", green), 0.34% that only use relaxed *filter*s (blue), and 6.9% with only safelisted relationships (red).

RPSL-based route verification is feasible and meaningful. ASes with skipped verifications only constitute 0.03% of ASes (not visible in Figure 2), underscoring RPSLyzer's high coverage of RPSL features. Out of the 54.9% of ASes with unrecorded cases, most can be explained by 27.2% of ASes missing *aut-num* objects and 24.2% of *aut-num*s with no rules. Excluding ASes with skipped or unrecorded cases, we find more ASes with verified (76.3%) or special-cased (62.5%) routes than ASes with unverified routes (23.1%). This high fraction of correctness indicates that operators make mindful use of the RPSL, and the RPSL may help identify routing anomalies and mistakes.

RPSL adoption is a larger concern than its misuse. RPSLyzer finds 25,596 ASes with at least one special-cased import or export (30.9% out of all ASes). Among these ASes, more incorrectly allow customer route exports (994, "export self") than imports (325, "import customer"). This is intuitive as the Internet hierarchy implies more ASes at the edge exporting (potentially incorrect) routes to their providers. However, most of the special cases are due to uphill propagation with no matching rules (23,298 ASes) or missing *route* objects (5181 ASes). These results imply that RPSL misuse is a minor issue, but increased adoption is crucial to increase coverage of Internet routes. Appendix D details unrecorded and special cases.

Verification statuses largely depend on the AS pair. Figure 3 displays the verification status for propagation between every pair of neighboring ASes observed in BGP routes, with distinguished propagation directions. Most AS pairs show either consistent status for all propagated routes (single color) or two statuses in an even split (two colors). An even split happens when each AS has a consistent status different from its neighbor's. For imports, we find 91.7% of AS pairs have a single consistent status; this number is 92% for exports. Thus, although routes with different prefixes and potentially different AS-paths propagate through the same AS pairs, these variations rarely impact route verification.

Most unverified routes traverse undeclared *peering*s. Although Figure 3 shows over half of AS pairs have unverified routes (418,328,

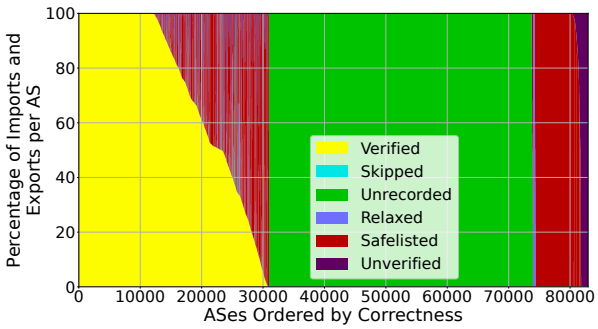

Figure 2: Route verification status for each AS.

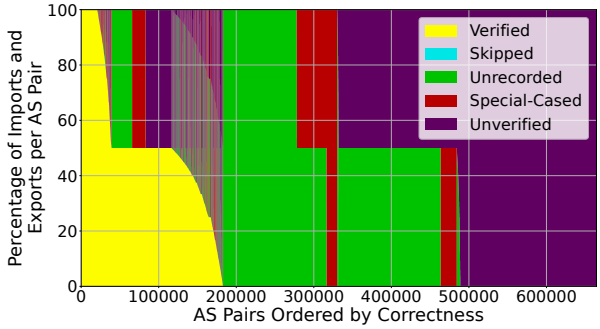

Figure 3: Route verification status for each AS pair.

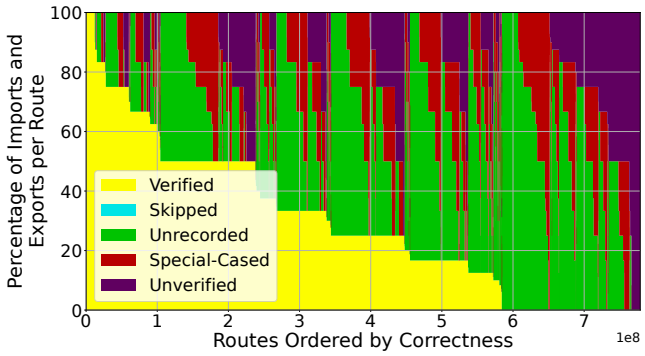

Figure 4: Verification status for all hops in BGP routes.

63.0%), most of them (98.98%) fail verification because the relationship is not declared in the RPSL (*i.e.,* no rules' *peering* covers the other AS in the pair).

Large ASes impact many routes. All ASes have the same weight in Figure 2, but ASes with many neighbors contribute disproportionately when analyzing by AS pairs or routes (Figures 3 and 4). The problem is aggravated because large ASes, including Tier-1s, Tier-2s, and CDNs, often specify few to no rules. This explains the large areas for unverified routes in Figures 3 and 4 compared to Figure 2.

Special cases cover most common RPSL misuses. Given most (99.0%) unverified cases are due to missing *peering* definitions, further relaxing *filter* checks (Section 5.1.1) can hardly increase our coverage. Moreover, since most ASes use simple *peering* definitions (Section 4), rule mismatches due to incorrect *peering*s should be rare, and are likely singular mistakes instead of common misuse of the RPSL.

Verification status is inconsistent within an AS-path. Figure 4 shows the different verification statuses for every import and export in each AS-path. Multiple colors in one bar indicate that verification statuses within an AS-path vary. Only 6.6% of routes have the same status across all hops, captured by having a bar of single color (1.6% verified, 3.0% unrecorded, and 1.6% unverified). Most AS-paths have a mix of two or three statuses, which impedes end-to-end verification and may limit the usefulness of the RPSL, in its current state, for troubleshooting routing anomalies.

We also assess the verification status of the first hop in AS-paths, where filtering is most effective in preventing route leaks and prefix hijacks [17]. Unfortunately, the results are similar (not shown), except that slightly fewer routes are unverified and slightly more are safelisted, since many first hops involve customers exporting routes uphill. Operator efforts on improving RPSL coverage at the edge could complement ongoing RPKI efforts [25, 55] in securing BGP announcements.

6 Related Work

BGP limitations. A large body of research focuses on addressing or alleviating BGP limitations including lack of authentication [8, 29, 44, 68], obliviousness to performance [57, 62, 66, 69], and challenging configuration [13, 22, 47, 52]. Although recommended best practices exist [23], actual deployments are not always compliant [14, 19, 21], and the mechanisms available for supporting interdomain routing coordination are ad-hoc and underdocumented [40, 41, 55, 61]. The RPSL has useful information applicable to all these challenges.

BGP security. Route Origin Validation (ROV) [30, 31, 53, 55] adoption improves BGP resilience against misconfiguration, but it only checks the first AS in the AS-path and provides limited protection against malicious prefix hijacks [17, 58]. The Autonomous System Provider Authorization (ASPA) draft [10] allows operators to securely specify their providers, which can then be used by other networks to filter invalid routes. Our analysis in Section 5 follows this approach using the RPSL instead of ASPA's provider relationships. Although the RPSL lacks the strong authentication provided by ROV and ASPA, it can specify richer intent than both ROV and ASPA, and has uses other than authenticating announcements.

BGP misconfiguration detection. Given the impact of misconfiguration, there have been significant efforts towards verifying BGP configuration (see [13, 67] and references therein). However, such checks overlook generated, semantically wrong BGP configurations. Our work benefits configuration checks and correctness more generally by detecting inconsistencies with the RPSL.

IRR data mining. Several studies have mined specific information from IRRs, e.g., to identify AS links [26, 27], business relationships [18, 36, 59], peering information [11, 16, 32, 46], and other information [9, 41, 64, 65]. The closest to our work are those combining RPSL and BGP dumps to verify route origins [20, 35, 59, 70] and AS paths [60]. These works, however, consider a smaller subset of the RPSL compared to RPSLyzer and are limited to binary validation. RPSLyzer supports richer analyses not only for path validation, but possibly other studies using the RPSL. One challenge shared by these works and ours is that IRRs do not make historical data available, which has been worked around by periodically scraping the IRRs (*e.g.,* [16, 20]).

7 Conclusion

We developed RPSLyzer, a novel tool to interpret RPSL policies. We characterized RPSL policies in public IRRs and discovered substantial potential for improvements in RPSL usage: around half of the ASes have yet to adopt the RPSL for documenting policies, and existing RPSL users can detail their policies further. We recommend operators adopt RPSL *route-set*s to increase policy accuracy and reduce maintenance overhead. We also leveraged RPSLyzer to verify routes on the Internet. We find that the RPSL covers less than half of the interconnections, but routes traversing the covered interconnections often match RPSL policies.

Future work includes the development of further RPSL tooling such as linters, enhancing RPSL validation in IRRs, tracking the evolution of RPSL policy usage over time, classifying ASes by RPSL usage, and possibly amending the language itself to promote expressing clear and correct routing policies. RPSL information can also be applied to longstanding modeling challenges such as AS-relationship inference and identification of sibling ASes, or operational tasks such as traffic engineering.

Acknowledgments

We thank the anonymous reviewers, shepherd, Arvin Ghavidel, and Harsha Madhyastha for their valuable feedback. Sichang He was an undergraduate student at Duke Kunshan University during the development of this work, and would like to thank Boyan Zhang and Zhaoyang He for their support. This work is partially supported by FAPEMIG, FAPESP, CAPES, CNPq, and NSF CNS award #2344761.

References

[1] [n. d.]. AMS-IX Route Servers Documentation. https://www.ams-ix.net/ams/documentation/ams-ix-route-servers.
[2] [n. d.]. Internet Routing Registry. http://www.irr.net.
[3] [n. d.]. RIPE Routing Information Service. http://www.ripe.net/data-tools/stats/ris.
[4] [n. d.]. The University of Oregon Routeviews Project. http://www.routeviews.org.
[5] Cengiz Alaettinoglu, WeeSan Lee, Ramesh Govindan, Rusty Eddy, John Mehringer, Katie Petrusha, Shane Kerr, Hagen Boehm, S.P.Zeidler, Timo Koehler, Håvard Eidnes, Faidon Liambotis, Nick Hilliard, Aliaksei Sheshka, Job Snijders, and Rowan Thorpe. 1994. IRRToolSet: The Internet Routing Registry Toolset. https://github.com/irrtoolset/irrtoolset.
[6] Cengiz Alaettinoglu, Curtis Villamizar, Elise Gerich, David Kessens, David Meyer, Tony Bates, Daniel Karrenberg, and Marten Terpstra. 1999. RFC 2622: Routing Policy Specification Language (RPSL). http://www.ietf.org/rfc/rfc2622.txt.
[7] Ruwaifa Anwar, Haseeb Niaz, David Choffnes, Ítalo Cunha, Phillipa Gill, and Ethan Katz-Bassett. 2015. Investigating Interdomain Routing Policies in the Wild. In *Proceedings of the 2015 Internet Measurement Conference*.
[8] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. 2017. Hijacking Bitcoin: Routing Attacks on Cryptocurrencies. In *IEEE Symposium on Security and Privacy*.
[9] Augusto Arturi, Esteban Carisimo, and Fabian E Bustamante. 2023. *as2org+*: Enriching AS-to-Organization Mappings with PeeringDB. In *Proceedings of International Conference on Passive and Active Network Measurement*.
[10] Alexander Azimov, Eugene Bogomazov, Randy Bush, Keyur Patel, Job Snijders, and Kotikalapudi Sriram. 2024. Draft: BGP AS-PATH Verification Based on Autonomous System Provider Authorization (ASPA) Objects. https://datatracker.ietf.org/doc/html/draft-ietf-sidrops-aspa-verification.
[11] Giuseppe Di Battista, Tiziana Refice, and Massimo Rimondini. 2006. How to extract BGP peering information from the internet routing registry. In *Proceedings of the 2006 SIGCOMM workshop on Mining network data*.
[12] Larry Blunk, Joao Damas, Florent Parent, and Andrei Robachevsky. 2005. RFC 4012: Routing Policy Specification Language next generation (RPSLng). http://www.ietf.org/rfc/rfc4012.txt.
[13] Matt Brown, Ari Fogel, Daniel Halperin, Victor Heorhiadi, Ratul Mahajan, and Todd Millstein. 2023. Lessons from the Evolution of the Batfish Configuration Analysis Tool. In *Proceedings of the ACM SIGCOMM Conference*.
[14] Randy Bush, Olaf Maennel, Matthew Roughan, and Steve Uhlig. 2009. Internet Optometry: Assessing the Broken Glasses in Internet Reachability. In *Proceedings of the ACM Internet Measurement Conference*.
[15] CAIDA. 2013. The CAIDA AS Relationships Dataset, 2023-07-01. https://data.caida.org/datasets/2013-asrank-data-supplement/.
[16] Hyunseok Chang, Ramesh Govindan, Sugih Jamin, Scott J Shenker, and Walter Willinger. 2004. Towards capturing representative AS-level Internet topologies. *Computer Networks* (2004).
[17] Avichai Cohen, Yossi Gilad, Amir Herzberg, and Michael Schapira. 2016. Jumpstarting BGP Security with Path-End Validation. In *Proceedings of the ACM SIGCOMM Conference*.
[18] Rami Cohen and Danny Raz. 2006. The Internet Dark Matter-on the Missing Links in the AS Connectivity Map. In *Proceedings of The IEEE International Conference on Computer Communications*.
[19] Ben Du, Gautam Akiwate, Thomas Krenc, Cecilia Testart, Alexander Marder, Bradley Huffaker, Alex C Snoeren, and Kimberly C Claffy. 2022. IRR Hygiene in the RPKI Era. In *Proceedings of International Conference on Passive and Active Network Measurement*.
[20] Ben Du, Katherine Izhikevich, Sumanth Rao, Guatam Akiwate, Cecilia Testart, Alex C Snoeren, and kc claffy. 2023. IRRegularities in the Internet Routing Registry. In *Proceedings of the ACM Internet Measurement Conference*. 104–110.
[21] Ben Du, Cecilia Testart, Romain Fontugne, Gautam Akiwate, Alex C Snoeren, and Kc Claffy. 2022. Mind Your MANRS: Measuring the MANRS Ecosystem. In *Proceedings of the ACM Internet Measurement Conference*.
[22] Nick Feamster and Hari Balakrishnan. 2005. Detecting BGP Configuration Faults with Static Analysis. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation*.
[23] David Freedman, Brian Foust, Barry Greene, Ben Maddison, Andrei Robachevsky, Job Snijders, and Sander Steffann. 2024. *Mutually Agreed Norms for Routing Security (MANRS) Implementation Guide*. Technical Report. Global Cyber Alliance. https://manrs.org/netops/guide/.
[24] Lixin Gao. 2001. On Inferring Autonomous System Relationships in the Internet. *IEEE/ACM Transactions on networking* 9, 6 (2001), 733–745.
[25] Yossi Gilad, Avichai Cohen, Amir Herzberg, Michael Schapira, and Haya Shulman. 2017. Are We There Yet? On RPKI's Deployment and Security. In *Proceedings of Network and Distributed System Security (NDSS) Symposium*.
[26] Yihua He, Georgos Siganos, Michalis Faloutsos, and Srikanth Krishnamurthy. 2008. Lord of the links: a framework for discovering missing links in the internet topology. *IEEE/ACM Transactions On Networking* (2008).
[27] Yihua He, Georgos Siganos, Michalis Faloutsos, and Srikanth V Krishnamurthy. 2007. A Systematic Framework for Unearthing the Missing Links: Measurements and Impact. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation*.
[28] Tomas Hlavacek, Italo Cunha, Yossi Gilad, Amir Herzberg, Ethan Katz-Bassett, Michael Schapira, and Haya Shulman. 2020. DISCO: Sidestepping RPKI's Deployment Barriers. In *Proceedings of Network and Distributed System Security (NDSS) Symposium*. San Diego, CA.
[29] Thomas Holterbach, Thomas Alfroy, Amreesh Phokeer, Alberto Dainotti, and Cristel Pelsser. 2024. A System to Detect Forged-Origin BGP Hijacks. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation*.
[30] Geoff Huston and George Michaelson. 2012. Validation of Route Origination Using the Resource Certificate Public Key Infrastructure (PKI) and Route Origin Authorizations (ROAs). https://tools.ietf.org/html/rfc6483.
[31] Daniele Iamartino, Cristel Pelsser, and Randy Bush. 2015. Measuring BGP Route Origin Registration and Validation. In *Proceedings of International Conference on Passive and Active Network Measurement*.
[32] Yuchen Jin, Colin Scott, Amogh Dhamdhere, Vasileios Giotsas, Arvind Krishnamurthy, and Scott Shenker. 2019. Stable and Practical AS Relationship Inference with ProbLink. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation*.
[33] Peter Kacherginsky. 2022. *Celer Bridge incident analysis*. Technical Report. Coinbase Engineering. https://www.coinbase.com/en-br/blog/celer-bridge-incident-analysis.
[34] Akmal Khan, Hyunchul Kim, Ted" Taekyoung" Kwon, and Yanghee Choi. 2010. Public internet routing registries (IRR) evolution. In *Proceedings of the 5th International Conference on Future Internet Technologies*.
[35] Akmal Khan, Hyun-chul Kim, Taekyoung Kwon, and Yanghee Choi. 2013. A comparative Study on IP Prefixes and their Origin ASes in BGP and the IRR. *ACM SIGCOMM Computer Communication Review* (2013).
[36] Akmal Khan, Hyun-chul Kim, and Ted" Taekyoung" Kwon. 2020. AS relationships inference from the internet routing registries. In *Proceedings of the SIGCOMM'20 Poster and Demo Sessions*.
[37] Thomas King, Christoph Dietzel, Job Snijders, Gert Doering, and Greg Hankins. 2016. BLACKHOLE Community. https://tools.ietf.org/html/rfc7999.
[38] Maria Konte, Roberto Perdisci, and Nick Feamster. 2015. ASwatch: An AS Reputation System to Expose Bulletproof Hosting ASes. In *Proceedings of the ACM SIGCOMM Conference*.
[39] Warren Kumari Kotikalapudi Sriram. 2011. RFC 6472: Recommendation for Not Using AS_SET and AS_CONFED_ SET in BGP. https://tools.ietf.org/html/rfc6472.
[40] Thomas Krenc, Robert Beverly, and Georgios Smaragdakis. 2021. AS-Level BGP Community Usage Classification. In *Proceedings of the ACM Internet Measurement Conference*.
[41] Thomas Krenc, Matthew Luckie, Alexander Marder, and kc claffy. 2023. Coarse-grained Inference of BGP Community Intent. In *Proceedings of the ACM Internet Measurement Conference*.
[42] Warren Kumari, Kotikalapudi Sriram, Lilia Hannachi, and Jeffrey Haas. 2023. Deprecation of AS_SET and AS_CONFED_SET in BGP. https://datatracker.ietf.org/doc/draft-ietf-idr-deprecate-as-set-confed-set/.
[43] Nate Kushman, Srikanth Kandula, and Dina Katabi. 2007. Can You Hear Me Now?! It Must Be BGP. *ACM SIGCOMM Computer Communication Review* 37, 2 (2007), 75–84.
[44] Weitong Li, Zhexiao Lin, Md Ishtiaq Ashiq, Emile Aben, Romain Fontugne, Amreesh Phokeer, and Taejoong Chung. 2023. RoVista: Measuring and Analyzing the Route Origin Validation (ROV) in RPKI. In *Proceedings of the ACM Internet Measurement Conference*.
[45] NTT Ltd. and Reliably Coded B.V. 2018. IRRd: IRRd version 4. https://github.com/irrdnet/irrd.
[46] Matthew Luckie, Bradley Huffaker, Amogh Dhamdhere, Vasileios Giotsas, and KC Claffy. 2013. AS Relationships, Customer Cones, and Validation. In *Proceedings of the ACM Internet Measurement Conference*.
[47] Ratul Mahajan, David Wetherall, and Tom Anderson. 2002. Understanding BGP Misconfiguration. *ACM SIGCOMM Computer Communication Review*.
[48] David Meyer, Cengiz Alaettinoglu, Carol Orange, Mark R. Prior, and Joachim Schmitz. 1999. RFC2650: Using RPSL in Practice. http://www.ietf.org/rfc/rfc2650.txt.
[49] Wolfgang Mühlbauer, Anja Feldmann, Olaf Maennel, Matthew Roughan, and Steve Uhlig. 2006. Building an AS-topology Model That Captures Route Diversity.

ACM SIGCOMM Computer Communication Review.
[50] Wolfgang Mühlbauer, Steve Uhlig, Bingjie Fu, Mickael Meulle, and Olaf Maennel. 2007. In Search for an Appropriate Granularity to Model Routing Policies. *ACM SIGCOMM Computer Communication Review.*
[51] NTT. 2024. Routing Registry - NTT-GIN. https://www.gin.ntt.net/support-center/policies-procedures/routing-registry/#filter.
[52] Santhosh Prabhu, Kuan Yen Chou, Ali Kheradmand, Brighten Godfrey, and Matthew Caesar. 2020. Plankton: Scalable Network Configuration Verification through Model Checking. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation.*
[53] Andreas Reuter, Randy Bush, Italo Cunha, Ethan Katz-Bassett, Thomas C Schmidt, and Matthias Wählisch. 2018. Towards a Rigorous Methodology for Measuring Adoption of RPKI Route Validation and Filtering. *ACM SIGCOMM Computer Communication Review* 48, 1 (Jan. 2018), 19–27.
[54] ASM Rizvi, Leandro Bertholdo, João Ceron, and John Heidemann. 2022. Anycast Agility: Network Playbooks to Fight DDoS. In *Proceedings of USENIX Security Symposium.*
[55] Nils Rodday, Ítalo Cunha, Randy Bush, Ethan Katz-Bassett, Gabi D Rodosek, Thomas C Schmidt, and Matthias Wählisch. 2021. Revisiting RPKI Route Origin Validation on the Data Plane. In *Proceedings of International Conference on Passive and Active Network Measurement.*
[56] Kevin Rollinson. 2023. GDPR and WHOIS: Here's What You Need to Know. https://umbrella.cisco.com/blog/gdpr-and-whois.
[57] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. 2017. Engineering Egress with Edge Fabric: Steering Oceans of Content to the World. In *Proceedings of the ACM SIGCOMM Conference.*
[58] Pavlos Sermpezis, Vasileios Kotronis, Petros Gigis, Xenofontas Dimitropoulos, Danilo Cicalese, Alistair King, and Alberto Dainotti. 2018. ARTEMIS: Neutralizing BGP Hijacking within a Minute. *IEEE/ACM Transactions on networking* (2018).
[59] Georgos Siganos and Michalis Faloutsos. 2004. Analyzing BGP policies: Methodology and tool. In *Proceedings of The IEEE International Conference on Computer Communications.*
[60] Georgos Siganos and Michalis Faloutsos. 2005. A blueprint for improving the robustness of internet routing.
[61] Brivaldo A. Silva, Paulo Mol, Osvaldo Fonseca, Italo Cunha, Ronaldo A. Ferreira, and Ethan Katz-Bassett. 2022. Automatic Inference of BGP Location Communities. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* (2022).
[62] Rachee Singh, Sharad Agarwal, Matt Calder, and Paramvir Bahl. 2021. Cost-effective Cloud Edge Traffic Engineering with Cascara. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation.*
[63] Alexandre Snarskii. 2007. BGPq4 - BGP Filter generator. https://github.com/bgp/bgpq4.
[64] Kotikapaludi Sriram, Oliver Borchert, Okhee Kim, Patrick Gleichmann, and Doug Montgomery. 2009. A comparative analysis of BGP anomaly detection and robustness algorithms. In *2009 Cybersecurity Applications & Technology Conference for Homeland Security.*
[65] Florian Streibelt, Franziska Lichtblau, Robert Beverly, Anja Feldmann, Cristel Pelsser, Georgios Smaragdakis, and Randy Bush. 2018. BGP Communities: Even More Worms in the Routing Can. In *Proceedings of the ACM Internet Measurement Conference.*
[66] Peng Sun, Laurent Vanbever, and Jennifer Rexford. 2015. Scalable Programmable Inbound Traffic Engineering. In *Proceedings of the ACM SIGCOMM Symposium on Software Defined Networking Research.*
[67] Alan Tang, Ryan Beckett, Steven Benaloh, Karthick Jayaraman, Tejas Patil, Todd Millstein, and George Varghese. 2023. Lightyear: Using Modularity to Scale BGP Control Plane Verification. In *Proceedings of the ACM SIGCOMM Conference.*
[68] Cecilia Testart, Philipp Richter, Alistair King, Alberto Dainotti, and David Clark. 2019. Profiling BGP Serial Hijackers: Capturing Persistent Misbehavior in the Global Routing Table. In *Proceedings of the ACM Internet Measurement Conference.*
[69] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, et al. 2017. Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering. In *Proceedings of the ACM SIGCOMM Conference.*
[70] Man Zeng, Heyang Li, Junyu Lai, and Xiaohong Huang. 2021. A BGP hijacking detection method based on multi-dimensional historical data analysis. In *2021 International Conference on Computer Communication and Artificial Intelligence (CCAI).*
[71] Jiangchen Zhu, Kevin Vermeulen, Italo Cunha, Ethan Katz-Bassett, and Matt Calder. 2022. The Best of Both Worlds: High Availability CDN Routing without Compromising Control. In *Proceedings of the ACM Internet Measurement Conference.*

A Additional RPSL Examples

IRRs provide a number of interfaces to access RPSL objects. The more classical approach is using WHOIS, but RADP and REST APIs are becoming more widely available. Below is a query for a *route* object showing a prefix and the AS expected to originate it:

```
% whois -h whois.radb.net 8.8.8.8 | grep
route:          8.8.8.0/24
origin:         AS15169
descr:          Google
```

The two real RPSL rules that follow illustrate the flexibility of the RPSL and the difficulty in parsing its semantics. The rule below is defined by AS8323:

```
% whois -h whois.ripe.net AS8323 | grep
import: from AS8267:AS-Krakow-1014 action pref=50;
        from AS8267:AS-Krakow-1015 action pref=50;
        accept PeerAS
```

This rule contains two *peering* and *action* definitions, with a single *filter* that applies to both *peering*s. The *filter* definition is the special keyword *PeerAS*, which needs to be interpreted dynamically at runtime as the neighboring AS each route is received from. This rule can be interpreted as: For an AS X in the AS8267:AS-Krakow-1014 or AS8267:AS-Krakow-1015 *as-set*s (*peering*), set the Pref to 50 (*action*), and accept the route only if its destination prefix has a matching *route* object with *origin* set to AS X (*filter*).[5] AS8323 could have set different Pref values for routes received from ASes in each of the *as-set*s. Note that this policy, taken as written, would allow only prefixes with *route* objects with an *origin* set to the neighboring AS X, which is very restrictive. Our relaxed *import customer* filter (Section 5.1.1) would allow any prefixes received from AS X.

The rule below is defined by AS199284:

```
% whois -h whois.ripe.net AS199284 | grep
mp-import: afi any {                                                    #1
    from AS-ANY action community.delete(64628:10, 64628:11, 64628:12,
                                        64628:13, 64628:14, 64628:15,
                                        64628:20, 64628:21, 64628:22);
               accept ANY;
  } REFINE afi any {                                                    #2
    from AS-ANY action pref = 65535; accept community(65535:0);
    from AS-ANY action pref = 65435; accept ANY;
  } REFINE afi any {                                                    #3
    from AS-ANY accept NOT AS199284^+;
  } REFINE afi ipv4 {                                                   #4
    from AS-ANY accept NOT fltr-martian;
  } REFINE afi ipv4 {                                                   #5
    from AS-ANY accept { 0.0.0.0/0^0-24 } AND NOT community(65535:666);
    from AS-ANY accept { 0.0.0.0/0^24-32 } AND community(65535:666);
  } REFINE afi ipv6 {                                                   #6
    from AS-ANY accept { 2000::/3^4-48 } AND NOT community(65535:666);
    from AS-ANY accept { 2000::/3^64-128 } AND community(65535:666);
  } REFINE afi any {                                                    #7
    from AS15725 action community .= { 64628:20 };
                 accept AS-IKS AND <^AS-IKS+$>;
    from AS196714 action community .= { 64628:20 };
                  accept AS-TNETKOM AND <^AS-TNETKOM+$>;
    from AS199284:AS-UP action community .= { 64628:21 };         #8
                        accept ANY;
    from AS35366 action community .= { 64628:22 };
                 accept AS-ISPPRO AND <^AS-ISPPRO+$>;
    from AS20940 action community .= { 64628:22 };
                 accept <^AS-AKAMAI+$>;
    ... several other peers ...
    from AS-ANY action community .= { 64628:22 };                 #9
                 accept PeerAS and <^PeerAS+$>;
  } REFINE afi any {                                                    #10
    from AS-ANY EXCEPT (AS40027 OR AS63293 OR AS65535)
                accept ANY;
  }
```

[5] In the RPSL, Pref is defined as the complement of BGP's LocalPref, *i.e.,* LocalPref ≡ 65535 − Pref [6]. This implies that routes with lower Pref in an RPSL policy are more preferred; while BGP LocalPref is the opposite (routes with higher LocalPref are preferred). We have not investigated this, but suspect many operators may be unaware of this priority inversion and specify rules as if Pref ≡ LocalPref.

This rule starts (#1) by specifying that the AS drops several BGP communities starting with 64628 (a private AS number). These communities are used internally by the AS, *e.g.*, in router configurations (see #7). (#2) The rule then specifies that routes received with BGP community 65535:0 will receive Pref 65535, while other routes will receive Pref 65435. (#3) Routers are configured to not accept external routes for AS199284's own prefixes. (#4) The *fltr-martian* filter indicates that IPv4 reserved, private, or unrouted prefixes, commonly referred to as "martians" or "bogons", are also not accepted. (#5) IPv4 prefixes are accepted up to /24 granularity; however, routes tagged with BGP community 65535:666, standardized to mean that traffic to this prefix should be blackholed [37], are accepted with granularities between /24 and /32. (#6) IPv6 prefixes are handled similarly. (#7) The AS defines specific actions to tag routes received from different peers with BGP communities; we call out that routes from upstreams are tagged with BGP community 64628:21 (#8) and that a catch-all rule allows importing routes from neighbors (#9). (#10) Finally, the rule specifies that routers do not accept routes from Netflix's and Facebook's offnets (AS40027 and AS63293).

B Details on RPSL Handling

AS-Path Regex Matching. To match an AS-path regex $\mathcal{R}$ in a rule's *filter* against an observed AS-path $\mathcal{A}$, we first replace each AS token t_i (a specific ASN, an *as-set*, the *PeerAS* keyword, or a wildcard) in $\mathcal{R}$ with a symbol $\sigma(t_i)$, and generate a *symbolic regex* $\mathcal{R}'$. We convert each ASN n_j in $\mathcal{A}$ to the set $\mathcal{N}_j$ of all symbols that n_j can match, *i.e.,* $\mathcal{N}_j = \{\sigma(t_i) | n_j \text{ matches } \sigma(t_i)\}$. We then generate a set of *symbol strings* from the original AS-path $\mathcal{A}$ by taking the Cartesian product of $\mathcal{N}_j$ for all n_j in $\mathcal{A}$. Finally, if any symbol string matches the symbolic regex $\mathcal{R}'$, we consider the AS-path $\mathcal{A}$ a match to the AS-path regex $\mathcal{R}$.

Nonstandard features. We add support for two cases of nonstandard but common syntax used by operators (4724 times in IRR dumps from June 2023): We allow a *route-set* to be followed by prefix-range operators `^n` and `^n-m`, and apply the range to all prefixes in the set.

Performance. We find matching routes against *filter*s that are *as-set*s to be the slowest operation when verifying BGP routes. As *as-set*s can be defined recursively, we flatten each *as-set* to its member ASes and perform a binary search for the route's prefix over each AS's *route* objects.

Limitations. We leave the handling of 58 rules whose *filter*s contain AS-path regex with ASN ranges (19 rules) or same-pattern unary postfix operators (*e.g.,* ~*, 39 rules) as future work. These constructs are not only rare but mostly used to create *filter*s to drop rare AS-paths containing private ASNs. We note that both cases can be supported by our symbolic approach by treating each ASN range and unary postfix operator as an AS token, similar to ASNs and *as-set*s. We also do not handle two rules containing inline prefix sets followed by range operators.

Although our tool can correctly parse the syntax, we ignore 54 rules with BGP community attributes in their *filter*s. We take this conservative approach because BGP communities may be stripped from BGP routes by intermediate ASes [40], and thus may not be observed by downstream BGP collectors even though it was attached to a route. This characteristic of BGP implementations is outside our control and prevents us from reliably matching rules using BGP communities in *filter*s.

Ignoring these 114 rules may cause the verification of related imports and exports to be *skipped* (Section 5).

C Route Verification Example

For a route with prefix 103.162.114.0/23 and the BGP AS-path ⟨3257 1299 6939 133840 56239 141893⟩, the verification report printout from RPSLyzer is shown below. We omit the numerous remote ASN mismatches of AS3257's rules.

```
BadExport { from: 141893, to: 56239, items: [MatchRemoteAsNum(58552),
    MatchRemoteAsNum(131755)] }
MehImport { from: 141893, to: 56239, items: [MatchRemoteAsNum(55685),
    MatchRemoteAsNum(133840), SpecOtherOnlyProviderPolicies] }
MehExport { from: 56239, to: 133840, items: [MatchRemoteAsNum(55685),
    MatchFilterAsNum(56239, NoOp), MatchFilter, SpecUphill] }
MehImport { from: 56239, to: 133840, items: [MatchRemoteAsNum(55685),
    SpecCustomerOnlyProviderPolicies] }
MehExport { from: 133840, to: 6939, items: [MatchRemoteAsNum(55685),
    SpecUphill] }
OkImport { from: 133840, to: 6939 }
OkExport { from: 6939, to: 1299 }
OkImport { from: 6939, to: 1299 }
UnrecExport { from: 1299, to: 3257, items:
    [UnrecordedAsSet("AS1299:AS-TWELVE99-CUSTOMER-V4"),
        UnrecordedAsSet("AS1299:AS-TWELVE99-PEER-V4")] }
MehImport { from: 1299, to: 3257, items: [MatchRemoteAsNum(12), ...,
    SpecTier1Pair] }.
```

In this example, the export from AS141893 to AS56239 is unverified (BadExport) because of *peering* mismatches: AS141893 defines two export rules, but none of them cover exporting routes to AS56239 (MatchRemoteAsNum):

```
aut-num: AS141893
export: to AS58552 announce AS141893
export: to AS131755 announce AS141893
```

The import by AS56239 from AS141893 is safelisted as an "only provider policies" case (MehImport and SpecOtherOnlyProviderPolicies) because the route mismatches all the rules' *peering*s of AS56239, AS56239 only specifies rules for providers, and AS141893 is classified as a customer of AS56239 in CAIDA's AS-relationship database [46].

Another special case is the export from AS56239 to AS133840, where the route matches the *peering* of AS56239's rule below:

```
aut-num: AS56239
export: to AS133840 announce AS56239
```

However, the route does not match the rule's *filter* because prefix 103.162.114.0/23 does not have a *route* object originated by AS56239. Additionally, AS137296, the only AS in AS56239's customer cone, does not have a *route* object for prefix 103.162.114.0/23, therefore the *filter* does not match even when we apply the "export self" relaxation. Finally, AS56239 is classified as a customer of AS133840, so the route is safelisted as an "uphill propagation" case (MehImport and SpecUphill). The import by AS133840 from AS56239 is also safelisted as "only provider policies", as AS133840 only specifies rules for its providers and AS56239 is classified as a customer.

The export from AS133840 to AS6939 is also safelisted as "uphill propagation", but does not even match the *peering* of any rule defined by AS133840 (MehExport and SpecUphill). The import by AS6939 from AS133840 strictly matches the following rule defined by AS6939 (OkImport):

```
aut-num: AS6939
import: from AS-ANY accept ANY
```

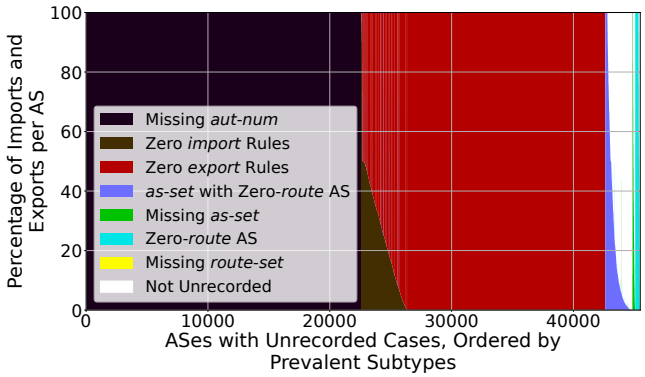

Figure 5: Breakdown of route verification failures due to unrecorded RPSL objects.

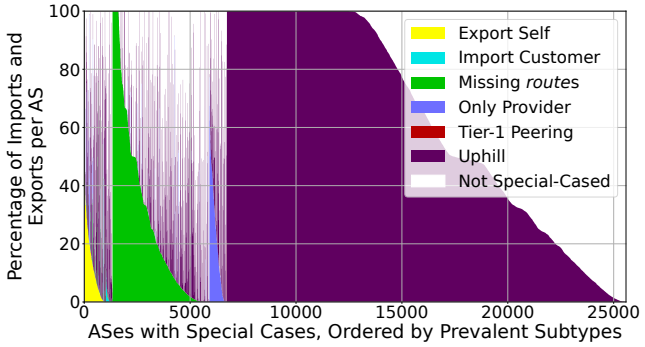

Figure 6: Breakdown of special cases per AS.

The export from AS1299 to AS3257 fails as "unrecorded" because two *as-set*s supposedly covering AS1299's customers and peers are not defined in the IRRs we use (UnrecExport and UnrecordedAsSet). Finally, the import by AS3257 from AS1299 does not match any rule defined by AS3257 and ends up safelisted as propagation across a pair of Tier-1 ASes (MehImport and SpecTier1Pair).

D BGP Route Verification: Unrecorded and Special Cases

Many ASes are missing or do not use the RPSL. Figure 5 zooms in on ASes with routes of the *unrecorded* status, *i.e.*, ASes in the green stripe in Figure 2. For each unrecorded case, we identify what type of information is missing.

Figure 5 shows that the most common unrecorded case is 22,562 ASes not having an *aut-num* object. These ASes have the same color across the *y*-axis because *all* routes traversing them have the unrecorded status by definition. The second most common type is for 20,048 ASes that have zero *import* (or *export*) rules when verifying an import (or export).

Fewer ASes have rules that refer to ASes with no originating *route* objects (zero-*route* ASes, 2706), or set objects (*as-set*, *route-set*, *peering-set*, and *filter-set*) missing in the IRRs (414 total). The white area shown for these ASes indicates that although some of their rules refer to zero-*route* ASes or missing objects, and thus prevent us from verifying some routes, there are other routes matched by other rules and properly checked.

Missing RPSL information explains most special cases. Figure 6 shows the prevalence of different special cases among the 30.9% of ASes with at least one special-cased import or export. More ASes (994, 1.2%) use "export self" compared to "import customer" (325, 0.4%). A significant portion (6.2%) of ASes have missing *route* objects, corroborating our point on the need for maintenance. ASes that have uphill propagation with no matching RPSL rules occupy a large 28.1% of all ASes, much more than the 12.4% of ASes with unverified routes, showing a lack of RPSL adoption for documenting uphill propagation policies.

E Validation of Relaxed Filters

We ran a small survey with operators of ASes whose *aut-num* objects include RPSL rules that we verify using the Export Self and Import Customer relaxations (Section 5.1.1). We extracted all ASes with rules of the following form:

```
import: from <X> import <X>
export: to <peer> export <self>
```

We then tried to find the email address of an operator from the IRRs, but only succeeded for 181 ASes out of 1102, due to identifiable information being removed (*e.g.*, for privacy or regulatory reasons [56]). We emailed these operators one rule from their AS following one of the patterns above, and asked them if the intended meaning was that of (i) the RPSL (strict, no relaxation), (ii) the Export Self or Import Customer relaxations, or (iii) others.

Of the three answers we received, *all* confirmed that the rule's intended meaning was that of the relaxed *filter*s in the Export Self and Import Customer special cases. Although the number of responses was low, the 100% confirmation rate indicates that the relaxed *filter*s may better represent the semantics meant by operators compared to the strict application of the RPSL.

F Ethics

RPSLyzer itself raises no ethical concerns. Although RPSLyzer improves accessibility to information in the RPSL, it does not reveal any additional information not already public.

In our email survey to validate the relaxed filters, we prioritized minimizing potential harm to operators while maximizing the benefits of understanding their intentions. We sourced email addresses publicly available from the IRRs; our emails were succinct to minimize distraction for the operators and included only information from their own RPSL entries; we do not directly report on any response received from operators. We believe that the operators' responses provide valuable insights into their usage of the RPSL, thus contributing significantly to the research.