# RPKI Syncing: Delay in Relying Party Synchronization

Khwaja Zubair Sediqi
*Max Planck Institute for Informatics*
*Saarland University*
Saarbrücken, Germany
zsediqi@mpi-inf.mpg.de

Romain Fontugne
*IIJ Research Laboratory*
Tokyo, Japan
romain@iij.ad.jp

Amreesh Phokeer
*Internet Society*
Port-Louis, Mauritius
phokeer@isoc.org

Massimiliano Stucchi
*Glevia GmbH*
Brüttisellen, Switzerland
stucchi@glevia.com

Massimo Candela
*NTT Data*
Barneveld, Netherlands
massimo@ntt.net

Anja Feldmann
*Max Planck Institute for Informatics*
Saarbrücken, Germany
anja@mpi-inf.mpg.de

***Abstract*—The Resource Public Key Infrastructure (RPKI) enhances routing security by providing cryptographically verifiable objects containing the Autonomous System (AS) numbers authorized to originate IP address ranges. Relying Party (RP) software performs RPKI synchronization by downloading, validating, and processing data from RPKI Publication Points (PPs) that are then provided to the routers to decide on accepting or rejecting BGP announcements. While RPKI is gaining importance, synchronization delays can compromise BGP reactivity, causing ASes to accept outdated or invalid announcements and potentially leading to service disruptions.**

**We study the causes of delay in RPKI synchronization by examining the key characteristics of the RPKI system, such as Route Origin Authorization (ROA) structures, RPKI operational modes, certificate chains, and the networking delays to access PPs. Our findings reveal that bundling multiple prefixes into a single ROA reduces validation delays by up to threefold. Depending on the ROA structure and publication infrastructure of delegated Certificate Authorities (CAs), they can reduce RP synchronization delays or inflate them by up to 90%. Some regions worldwide observe delays of a few milliseconds and others of several hundred milliseconds in accessing RPKI resources. Every additional 100 milliseconds of Round Trip Time (RTT) can delay the RP synchronization by up to 25 extra seconds.**

***Index Terms*—RPKI, BGP, Synchronization, Delay, Security**

## I. Introduction

The Resource Public Key Infrastructure (RPKI) [1]–[3] is designed to improve Border Gateway Protocol (BGP) security by providing a cryptographically verifiable association between the IP prefixes and Autonomous Systems (ASes) that are authorized to advertise these prefixes. These associations are recorded in Route Origin Authorization (ROA) objects and are signed by a Certificate Authority (CA). Each of the five Regional Internet Registries (RIRs) operates a Trust Anchor (TA) and a Publication Point (PP) to host the RPKI objects.

Network operators use Relying Party (RP) software to download the RPKI data from a globally distributed set of Publication Points or repositories, cryptographically verify the ROAs, and provide the Validated RoA Payloads (VRPs). We refer to all these steps as the RPKI or Relying Party synchronization process. The updated set of VRPs from the RPKI synchronization process is then provided to the RPKI-enabled BGP-speaking routers to help them filter out the illegitimate routes originated by rogue ASes from incoming BGP announcements.

Network operators have documented operational issues such as long delays for changes in RPKI to take effect [4]–[6]. The time it takes for an announcement to propagate globally in BGP is fast [7]–[9]. However, changes in RPKI are slow and might lead to scenarios where BGP accepts outdated or invalid announcements, potentially leading to service disruptions.

RPKI delay has three main components: (1) the *publication delay*, which is the time required for a change in RPKI data to be publicly visible in Publication Points, (2) the *synchronization delay*, which is the time Relying Party software will take to fetch, validate and update its list of Validated ROA Payloads, and (3) the *BGP refresh interval*, which is the time interval at which RPKI-enabled routers refresh their list of VRPs perform Route Origin Validation (ROV) and update their Forwarding Information Base (FIB) [1].

Fontugne *et al.* [10] analyzed the end-to-end delay by observing the reaction time of BGP after changing ROA information at the five RIRs. More precisely, they show that the highest cause of delay between RPKI and BGP comes from the synchronization of RP software but they provide little details on why this step is slow.

The RP synchronization delay is composed of three main elements: (1) RPKI data downloading time, (2) RPKI data validation time, and (3) the time it takes to produce the list of VRPs. The first part requires communication between RP software and Publication Points.

As RPKI is gaining importance, and its adoption is growing [11], the download and process time of RPs is also expected to increase over time due to an increasing number of CAs, ROAs, and PPs incurring more delays. The increasing fetch

978-3-903176-74-4 ©2025 IFIP

time for retrieving a full snapshot of the RPKI tree may hamper further deployment. In this paper, we study the dynamics of fetching the RPKI tree from the PP hierarchy, examine the cryptographic verification of ROAs, and propose optimization techniques to enhance the RP synchronization delay. Our contributions are as follows:

**In-depth analysis of RPKI synchronization delay:** We analyze each TA's structure of publishing ROAs and the impact of mode of operation (hosted vs delegated) in RPKI data publication. Our results show that consolidating several prefixes of an AS into a single ROA can reduce the cryptographic verification time by a factor of up to three. While the fetching and validation of ROAs are the main delay factors in RP synchronization, we further identify that 7% of VRPs in delegated PPs accounts for up to 90% synchronization delays for one of the TAs (see Section IV).

**Large-scale latency measurement:** We perform large-scale active measurements using more than 700 RIPE Atlas anchors from 91 countries around the globe to identify the accessing delay to RPKI PPs. Most PPs are accessible within 50 milliseconds of Round Trip Time (RTT) from their primary serving regions but require hundreds of milliseconds from other regions. Accessing RPKI PPs using IPv4 or IPv6 has a similar delay, but the RPKI Resource Delta Protocol (RRDP) deployment of PPs is accessible faster than PPs using the rsync protocol. We show that for every additional RTT of 100 milliseconds, RPKI synchronization increases up to 25 seconds (see Section V).

We identify the root causes of RP synchronization delay and discuss possible optimization techniques for PP maintainers, RP software developers, and RP synchronization modes of operation.

## II. Background/RPKI Overview

The Resource Public Key Infrastructure [3] is a security system that uses a chain of X.509 certificates to validate the authenticity of routing announcements made by ASes on the Internet. RPKI aims to prevent malicious or accidental routing incidents due to prefix mis-origination, i.e., when an attacker falsely advertises itself as a specific IP address block holder, which can result in redirecting or blackholing Internet traffic intended for the legitimate holder. However, RPKI does not mitigate all BGP attacks (e.g., path injection). Figure 7 in the Appendix A presents an overview of the various building blocks of the RPKI system that we explain through the action statements below.

**Certificates**: RPKI follows the IANA prefix allocation hierarchy. Each RIR operates a self-signed certificate, called a Trust Anchor, for the Internet resources they manage.

When RIRs allocate resources to Local Internet Registries (LIRs) e.g., an Internet Service Provider (ISP) or to National Internet Registries (NIRs) like TWNIC or NIC.br, the TA issues a child certificate with the member resources [Action ❶]. This certificate (and the associated private key) is used to create special attestations called Route Origin Authorizations.

**Modes of operation**: NIRs or LIRs that have been issued resource certificates can choose between two modes of operation: (1) The *Hosted mode*, where their RPKI environment (RPKI objects, private key material, and PP) is managed entirely by the RIR. This is currently the most used mode of operation, or (2) the *Delegated mode*, where an LIR operates its own RPKI environment including the management of a PP. Some RIRs provide Publication as a Service (PaaS), where delegated CAs can publish their objects in their RIR's PP.

**Route Origin Authorization (ROA)**: Once the members are issued a certificate that proves their holdership of allocated resources, they can issue a signed attestation called a ROA to authorize an AS to advertise their prefixes [Action ❷]. The ROAs are then published in repositories called Publication Points, which are operated by either the RIR or by the resource holder itself in the case of the delegated mode [Action ❸].

**Publication Points**: Each Certificate Authority in the RPKI hierarchy needs to publish their objects (certificates, ROAs, manifest files, and Certificate Revocation Lists (CRLs)) in a publicly accessible repository, called a Publication Point. Anyone can operate a PP, but in practice, each RIR operates a PP to publish the RPKI objects of their own CA and those of the child CAs (if the latter is operating in 'hosted mode'). These objects are then fetched using either the RRDP[12] or rsync protocol and validated by Relying Party software.

**Route Origin Validation (ROV)**: [Action ❹] RPs periodically fetch and validate the RPKI objects from the PPs, and they generate a set of VRPs. VRPs contain a set of prefixes (with prefix and max length) and the associated authorized ASes. Validation follows a top-down approach, starting with the TAs. RP software usually comes bundled with the RIR's Trust Anchor Locators (TALs), where each TAL contains the root certificate's Uniform Resource Identifier (URI). In the case of delegated mode, the parent CA will point to the child CA's repository to build the chain of trust, allowing the RPs to fetch and validate the RPKI objects from the delegated CAs' PPs [Action ❺]. Using the *rpki-rtr* protocol, the list of VRPs is periodically pushed to the RPKI-enabled routers [13]. Using this information, incoming routing announcements in the Routing Information Base (RIB) are tagged as either Valid, Invalid or Unknown. Based on the routing policy, routers will accept or drop incoming routing announcements before sending them to their Forwarding Information Base (FIB) [Action ❻].

## III. Related Work

Osterweil *et al*. [14] simulated a "fully deployed" RPKI and showed that it would take between 15 to 30 days to synchronize the local caches at RPs around the world. The number of objects was calculated based on RIRs' allocation trends with a one-to-one mapping between allocation and a ROA. However, since 2012, there have been a few improvements to RPKI, namely the standardization of the RPKI Delta Protocol [12], the use of Content Delivery Networks (CDNs) for repository hosting [15], bundling ROAs and other network optimizations that helped reduce the fetching time.

Measuring RPs in the wild, Kristoff *et al*. [16] looked at the *timeliness* of RPs which is based on their configured refresh time intervals that would determine how many times an RP would synchronize with the PP. Routinator, a RP software, has a refresh interval of 10min [17]. For rpki-client, another RP software, the manual page states an hourly run using crontab [18]. To date, there are no defined standards on the refresh time interval and each RP comes with its own default time settings.

Hlavacek *et al*. [19] analyzed the retrieval interval, the cache management algorithms and the frequency at which the different RPs refresh the cache by contacting the publication server. They also looked at the reachability of PPs by varying network configurations such as enabling/disabling DNSSEC. Hlavacek *et al*. [20] proposed an attack scenario whereby an attacker can inject packet loss at specific time intervals and stall RPs by creating very long delegation chains. This would prevent RPs from fetching ROAs and other RPKI data from PPs, forcing the expiry of cached objects. While their attack scenario is hypothetical, increasing the depth of certificates impacts the synchronization time, as we see in Section IV-D. The possibility for malicious repositories that could stall the RP synchronization process was studied by Hlavacek *et al*. [21], and the sort-and-limit algorithm is proposed to prevent attacks and repository failures. van Hove *et al*. [22] studied a threat model where the attacker controls a CA and PP, and discovered that attackers might have the potential to disrupt RP software.

RPKI MIRO [23] was introduced to monitor and inspect RPKI objects. Researchers investigated RPKI usage by assessing the deployment of RPKI and ROV [24]–[28] and explored the RP software vulnerabilities, security concerns, and RFC compliance [29]–[31].

To reduce RP synchronization time Li *et al*. [32] proposed a bit-map encoding scheme to compress the total size of ROA payloads in RRDP by 26.6%. Using a 10 Mbps link, they showed a reduction of 41.3% for the currently used max-length approach and 50.4% for the minimal ROA approach [33]. While the compression algorithm proposed can reduce validation time, it remains theoretical as it requires adoption by all RPs, which is impractical.

The closest study to ours is by Fontugne *et al*. [10] which looked at the end-to-end delay between the creation or removal of a ROA and the corresponding changes observed in BGP. They announced prefixes assigned from all five RIRs and created/deleted ROAs to change the RPKI status of the announcements. Delay metrics were recorded at several steps of the RPKI lifecycle, namely at user creation, ROA signing, publication, and BGP update. They found that RIRs usually publish new RPKI information within five minutes, except APNIC, which is ten minutes slower.

However, their work does not investigate the detailed causes for RP's synchronization delay. We address this gap by providing an in-depth analysis of the delay factors in the RP synchronization process.

## IV. Measuring RP Synchronization Delay

RP synchronization delay can occur in several parts of the process, including network delay of downloading data from RPKI Publication Points, cryptographic verification at the client by following the certificate chain, and production of VRPs. In this section, we thoroughly analyze delay factors that impact the RPKI validation process and try to identify potential delay areas for optimization.

### A. Experimental Setup

To measure the Relying Party software synchronization delay, we use dedicated hardware located in Germany, running Debian 11 with an AMD 8-core processor, maximum speed 2.1GHz each, 16 threads, 31 GiB of RAM, and 2.0 GiB of Swap memory. We connect the machine directly to the Internet with no firewall in between via one Gbps link. We performed a similar analysis from a second vantage point in Japan using a dedicated Virtual Machine (VM) and obtained similar results. In this paper, we present results of dedicated hardware only, which is less biased by the system's underlying infrastructure compared to the VM.

**Relying Party Software:** We use *rpki-client* version 9.3 [34], and *Routinator* version 0.14.0 [35] as the actively maintained, popular in the RPKI community, and commonly used RP software in practice [16]. Using two different RP implementations, we identify implementation-specific delays and separate them from RPKI infrastructure-caused delays. Our goal is to understand the bottlenecks in provisioning RPKI objects and their impact on delay.

We use the default configuration for both RP software to produce a list of VRPs and always run only one instance of one software at a time during the experiment to avoid any competition in terms of network and CPU resources.

In practice, network operators start by pulling and validating the entire RPKI tree and then perform synchronization of local RPKI cached data by fetching updates only. We refer to the first case as the Cold cache and the second case as the Warm cache. To thoroughly analyze the RPKI synchronization delay, we also introduce a third case we call Hot cache RPKI synchronization. We define the Cold, Warm, and Hot modes for RPKI cached data as follows:

**Cold cache**: We manually delete all the RPKI cached objects from our system and then start the RPKI synchronization with an empty cache. Clearing the cache before running the RP software allows us to measure the entire time RPs take to fetch data, process it, and produce the list of VRPs. For RP operations in Cold mode, we select the one-hour interval aligned with rpki-client man pages [18].

**Warm cache:** RPs reuse the previously downloaded RPKI data. However, they still have to synchronize with RPKI repositories to update to the latest data, run validation, and produce the VRPs. The cached RPKI data in our experiment is around 10 minutes old, aligned with Routinator's default policy [17], for each Warm mode RP synchronization.

**Hot cache:** RPs process the cached data only and do not download any data from RPKI repositories. While Hot mode is

not used for operational settings, we use this mode to exclude the network delay and provide the exact delay required to perform cryptographic verification or validation of RPKI data and produce the final output set of VRPs.

As we report on timing values, we consider the impact of CPU cache on our analysis. A full or partial existence of RPKI data in RAM, swap/virtual memory, or buffer memory, which we refer to collectively as the CPU cache memory, may influence our analysis. To handle the CPU cache memory impact, for each of the RP operation modes, we consider two states of CPU cache retained or True where a '(T)' is appended to the name of RP software like *Routinator (T)* or *rpki-client (T)*, and the CPU cache cleaned or False with '(F)' appended to RP name, for example, *Routinator (F)* and *rpki-client (F)* in the rest of this paper. For each mode of RPKI cache in Cold, Warm, and Hot, and every status of CPU cache being True or False, we perform 100 iterations of each RP software synchronization for 15 consecutive days. We run the same configuration for the two RPs as close in time as possible, but only one instance at a time. We collect each experiment's detailed logs and output, constituting this section's main dataset.

### B. Synchronization Delay of Relying Party Software

We define the RP synchronization delay as the time from the start to the end of one RPKI validation process where RP software contacts RPKI Publication Points, downloads data, performs cryptographic verification of data, and produces a set of VRPs (excluding synchronization with the router via RTR). The Hot RPKI cache mode does not have any network involvement, and the RP synchronization delay reflects only the time required to cryptographically verify the RPKI data from the system's local cache and produce VRPs. We use the terms RP synchronization and RPKI validation interchangeably throughout this paper.

In Figure 1, Routinator has the highest synchronization delay in Cold mode with a median value of around 490 seconds for both cases of CPU cache True and False. For rpki-client (T) and (F) in Cold mode, the median delay is around 190 seconds. Using Routinator in Cold mode has a major page fault rate of around 455k. When RP software accesses a memory page that is not in physical RAM, a major page fault occurs, and the operating system then fetches the page from the disk. Because accessing disk storage is slower than RAM, major page faults create delays for Routinator. For rpki-client, we identify zero to 20 major page faults. In Cold mode, neither RP is sensitive to CPU cache status as all the data should be downloaded from the Internet.

For the Warm mode, Routinator (T) has a median RP synchronization delay of around 200 seconds and 210 seconds when the CPU cache is False. rpki-client (T) and (F) have a median value of around 140 seconds and 240 seconds, respectively. We believe the increase in synchronization delay for rpki-client (F) is due to high read from the file system, showing rpki-client is more sensitive to the CPU cache than Routinator. As expected, the Warm mode has a lower RP

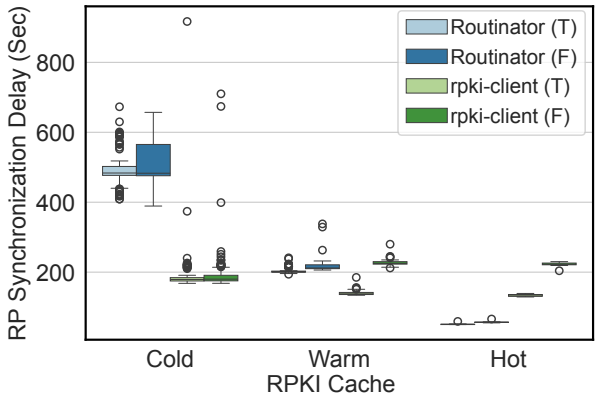


Fig. 1. Boxes show the RPs synchronization delay for Cold, Warm, and Hot RPKI cache modes. The order of the boxes matches the legend.

synchronization delay compared to the Cold mode because less data has to be downloaded.

We see the highest outlier of RP synchronization delay for rpki-client in Cold mode because it has a default timeout of 900 seconds to switch from RRDP to rsync protocol [18]. We identify the *rpki.cnnic.cn* repository containing 1,218 VRPs (1,117 IPv4, 101 IPv6) registered for 173 ASes as the problematic PP causing the highest outlier. To avoid delays caused by outliers, skipping the problematic PPs, especially when the local copy exists, might be an option. Stalling the RPKI synchronization process by a PP is a vulnerability also reported in other studies [22][36].

Routinator (T) and (F) have the lowest median delay of 40 and 50 seconds, respectively, in Hot mode. The rpki-client (T) has a higher median delay of more than 100 seconds and more than 200 seconds when the CPU cache is False or rpki-client (F). The Hot mode indicates that Routinator has a faster cryptographic verification of the RPKI data than rpki-client. However, the rpki-client has a better synchronization time when the network is involved in the RP synchronization process. Cold mode is write-intensive, with nearly 4 million write operations to the file system. Inversely, the Warm and Hot modes are read-intensive, with more than 4 million read operations for Routinator (F) and rpki-client (F) only when the data is unavailable in the CPU cache.

We run the same set of experiments using a memory-mapped file system for the RPKI data storage to understand the potential impact of read and write to the disk file system on RP synchronization. Using a memory-mapped file system mitigates the impact of the system's CPU cache and further reduces the RP synchronization delay by up to 8%. This is because the read/write from and to memory-mapped file systems are faster than the disk-mapped file system.

For Routinator, the significant delay difference between Hot mode and the other modes indicates that fetching RPKI data from the Internet is the main contributor to synchronization delay. In contrast, the rpki-client exhibits a different behavior of taking more time to process the cryptographic verification of ROAs during synchronization. The use of RPKI cache

is more efficient for Routinator, reducing synchronization delay to nearly 200 seconds compared to not using cached data, specifically in settings where a refresh occurs every 10 minutes. However, the RPKI cache has a few seconds of improvement for the rpki-client only when the CPU cache is retained. The rsync protocol implementations with rpki-client using openrsync and Routinator using the system-provided rsync might also contribute to variations in synchronization delay in Cold mode [34], [35].

Routinator has the fastest cryptographic verification of RPKI data and is more efficient in handling empty CPU cache status, but suffers from a high number of page faults, causing the highest RP synchronization for the software in Cold mode. In contrast, rpki-client takes nearly twice as much time as Routinator for the cryptographic verification of the data, but has efficient network handling and lower synchronization delay than Routinator in Cold mode with almost no page fault. It is also more prone to the system's CPU cache state. We think combining Routinator's fast cryptographic verification and robust handling of CPU cache with rpki-client's efficient network handling in an RP can significantly reduce RP synchronization, and running them on a memory-mapped file system can even further optimize the RPKI synchronization process.

Typically, Routinator runs as a daemon, and rpki-client runs by cron, in Warm mode, and stores its cache state on disk; this means these RPs will only run in Cold mode once after first installation on a system, or after an explicit wipe of the on-disk RPKI cache by an operator. In normal operations, and because RPs typically run in Warm mode, rpki-client and Routinator will have a very similar synchronization delay in day-to-day performance.

### C. Impact of ROA Composition

We want to understand whether the delay for cryptographic verification of ROAs for all five RIRs is the same or different. Measuring each RIR's RPKI tree individually provides us with a clearer picture of the performance bottlenecks at each RIR. For this purpose, we measure the RP synchronization delay in Hot cache mode for the entire RPKI tree of each RIR (i.e., the TA and all its delegated repositories). The Hot mode is not biased by our machine's location, as only the RPKI cached data is analyzed, and network delay is not involved. Since each TA's RPKI tree maintains a vastly different number of ROAs and VRPs, we normalize the measured delay per VRP and the average number of VRPs per ROA.

Figure 2 (a) illustrates a scatter plot of the average number of VRPs produced per second on the x-axis and the average number of VRPs per ROA object on the y-axis for each RIR using Routinator and rpki-client. The results show that the RPKI tree of RIPE and APNIC has the fastest processing time per VRP, more than 10k and 9k VRPs per second, respectively, using Routinator or more than 3k using rpki-client. The rest of the TAs have less than 3k VRPs per second processing time using rpki-client and less than 6k using Routinator. AFRINIC has the lowest number of 2.3k VRPs per second using Routinator, and LACNIC has the lowest number of 1.8K VRPs using rpki-client. Cryptographic verification of ROAS for the same amount of VRPs from ARIN, LACNIC, and AFRINIC RPKI tree requires at least twice as much time as APNIC and RIPE using Routinator. The VRP production rate, number of VRPs per second, and the difference between Routinator and rpki-client are due to Routinator being fast in Hot mode and even faster when the data amount is higher, as shown for AFRINIC and ARIN. This difference between RIRs is mainly explained by the bundling of multiple prefixes in a single ROA object, a common practice for RIPE and APNIC but not the other RIRs. ROAs from ARIN, AFRINIC, and LACNIC RPKI trees produce, on average, around one VRP. In contrast, the RIPE and APNIC RPKI tree ROAs contain, on average, 3.6 and 5.6 VRPs. Consolidating several prefixes for one authorized AS into a single ROA object reduces the number of cryptographic operations required per VRP and, hence, significantly reduces the delay observed for processing RPKI data.

Figure 2 (b) displays the distribution of prefixes per AS extracted from VRPs for each RIR. Around 60% to 85% of ASes have more than one IP prefix per AS in each TA's RPKI tree. In the range of 10% to 18% of ASes have more than ten prefixes per AS, with Amazon AS16509 having the highest number of 10,879 IP prefixes in RPKI data.

All ASes with more than one prefix registered in RPKI are candidates for bundling their multiple prefixes into a single ROA object. We discuss the potential side-effects of bundling in Section VI.

### D. Impact of Certificate Chain Depth

In the RPKI, certificates are organized in a hierarchical mode. The topmost certificates in RPKI are the RIRs' Trust Anchors (root certificates) (Depth 1). For different operational reasons, RIRs can create intermediate child certificates to manage their CA, AFRINIC and LACNIC use depth of 2, and the rest of the RIRs use a depth of 3 for this purpose. They also create certificates (member certificates) for or to National Internet Registries or Local Internet Registries for organizations to which resources are assigned (Depth 4). LIRs and NIRs can issue End-Entity certificates (Depth 5), which are used to sign ROAs. LIRs and NIRs can sub-allocate resources, this adds another level to the depth of the certificate.

RP synchronization needs a new TCP connection to be established for each Publication Point. This process involves numerous handshakes, making it crucial to ensure fast downloads of RPKI snapshots or updates from these delegated CAs. We study the impact of the depth of the certificate chain or delegated Certificate Authorities on RP synchronization delay. For this particular measurement, we use only Routinator as it is not possible to measure CA depth using rpki-client. Routinator has a configurable chain depth to mitigate certain denial-of-service attacks [22] with a default CA depth value 32. We use this feature Routinator to perform RPKI synchronization in Cold mode with CA depths of 2 to 6 for each RIR's RPKI tree, as other depths contain no ROAs and 50 iterations at each depth and for every RIR.

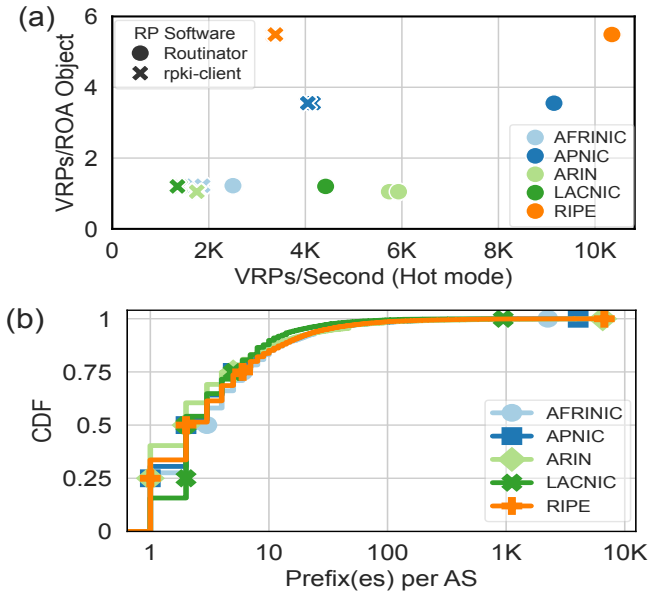


Fig. 2. For each RIR's RPKI tree (a) shows the average rate of RPKI data validation and (b) shows the distribution of IP prefix per AS.

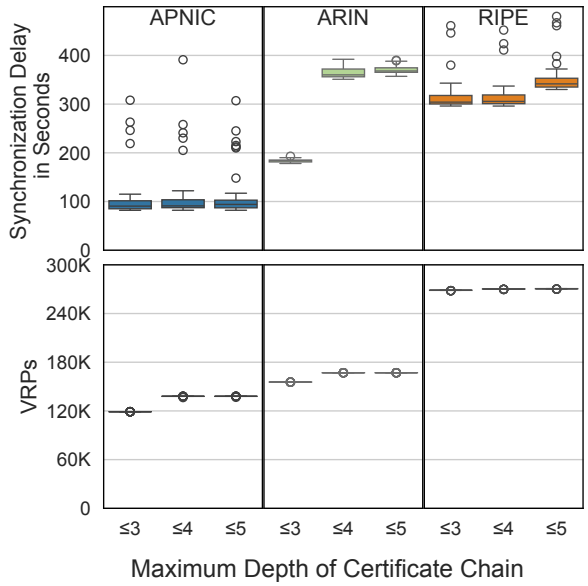


Fig. 3. Impact of CA depth on TA's RP synchronization delay (top plot) and their contribution of VRPs (bottom plot) using Routinator in Cold mode.

In Figure 3 the x-axis shows the maximum depth of the certificate chain of trust (i.e., a maximum depth of 4 includes all levels up to 4). The y-axis represents the delay in seconds (top subplot) and the number of VRPs (bottom subplot). We show the RP synchronization delay for each CA depth for APNIC, ARIN, and RIPE in the top subplot of Figure 3 and the corresponding number of VRPs at each depth in the bottom subplot. We exclude AFRINIC and LACNIC from this analysis because they take less than 100 milliseconds, not affecting the total delay of RP synchronization. Moreover, AFRINIC has no CA depth, and LACNIC has only one additional depth than TA, and only for the mentioned RIRs, we observe data at a CA depth of 2. For APNIC, the number of VRPs are increased from 119k at a CA depth of 3 to 137k at a CA depth of 5, shown in the bottom subplot of Figure 3. However, the CA depth impact on RP synchronization delay is minimal for APNIC, with all depths having less than 100 seconds of delay. The *rpkica.twnic.tw* PP at CA depths of 3 and 4, and *rpki.sub.apnic.net* PP causing outliers with increased RP synchronization delay from 200 to nearly 400 seconds. Delegated CAs of APNIC contribute 13% additional VRPs and add only around 2% extra delay, which means they have a positive impact and help RP software validate more data in relatively less time.

The maximum certificate chain depth clearly impacts RP synchronization delays for ARIN. We notice a median of 184 seconds RP synchronization delay at a CA depth of 3, which increased to a median of 360 seconds at a depth of 4 and is slightly higher at the CA depth of 5. Processing 7% of VRPs from ARIN's delegated CAs cause more than 90% delay for the entire ARIN RPKI resources in Cold mode.

Moving deep from CA depth of 3 to 4 for RIPE CA depth, the RP synchronization delay has medians of 304 and 305 seconds, which is in line with the slight increase of VRPs from 268k to 270k, as shown in the bottom subplot. However, the delegated CAs at a depth of 5 that contributes nearly 320 VRPs increase the RP synchronization from 304 to 330 seconds. This means around 1% of delegated RPKI data causes up to 10% additional delay for RIPE.

These delays reflect the fact that a small number of ROAs from delegated CAs may inflate the delay observed for the whole RPKI tree of a TA. While delegated CAs hosted on cloud infrastructure positively impact the RP synchronization for APNIC, a few specific delegated CAs cannot always serve RPKI data using RRDP and cause RPs to hang for hundreds of seconds on them, which can slow down the entire RP synchronization process. Analyzing the RP synchronization logs, we find that sometimes repositories cannot serve the RPKI data using RRDP, they time out, and then fall back to rsync. This timeout and fallback to rsync is the common issue for those repositories creating outliers with high RP synchronization.

RPs download and process delegated CA objects after the initial delay incurred in fetching and processing TA data at the top of the certificate chain. Any slowdown in the certificate chain's depth for delegated CAs can significantly prolong the entire RPKI synchronization process. Therefore, the quick and efficient operation of delegated CAs is even more important. Maintainers of particular delegated CAs may take action to ensure smooth and fast serving of RPKI data to avoid hanging of RP synchronization process on their repositories. Fast operation of delegated CAs is crucial in RPKI synchronization, and even more in the future as their number may increase.

## V. RPKI Delay Measurement using RTT

RPKI data contains nearly 560k small objects stored in distributed repositories or Publication Points worldwide. Downloading RPKI data requires exchanging around 140k packets between the RP software and RPKI Publication Points. Therefore, understanding the Round Trip Time to access RPKI resources or PPs from around the globe is crucial to the RP synchronization delay.

### A. Measurement Design and Data

We perform an active measurement using all the RIPE Atlas anchors by running a series of TCP-based traceroutes toward all the RPKI Publication Points for two weeks.

We use around 700 RIPE Atlas anchors located in 91 countries for our experiment because they are usually hosted in data centers or in well-provisioned locations like universities. Hence, the delay results observed from RIPE Atlas anchors from a particular network are arguably a good approximation of the delay experienced by Relying Party software in that network for the RPKI synchronization process.

We use a list of 69 PP hostnames for both RRDP and rsync PPs extracted from the Authority Information Access (AIA) field of all RPKI certificates of each TA. We mimic the two protocols by running TCP-based traceroute (both rsync and RRDP are using TCP), on port 443 for RRDP and port 873 for rsync measurements. We could use ping, but ICMP might be handled/prioritized differently by network operators than real traffic. Moreover, we anticipated that path-level information might become useful for extended analyses or follow-up studies. The PP hostnames are resolved using the anchor's local DNS. We probe each PP hostname via IPv4 and IPv6 (three times per address) every five hours for two weeks—ensuring coverage of all hours and weekdays. We use data of traceroutes reaching the correct destination IP. These include 3,893,594 traceroutes to RRDP PP hostnames and 3,149,005 to rsync PP hostnames.

### B. Measuring Delay of RPKI Publication Points

The violin plots in Figure 4 show the distribution of RTT delay for accessing RPKI rsync and RRDP PPs using IPv4 and IPv6 from each region globally. The x-axis shows the continent, the y-axis shows the RTT delay in milliseconds in log scale, the width of the violin reflects the relative density of data, and the dotted horizontal lines, within each violin, represent the data's quartiles.

We observe a similar distribution of RTT delay for both IP versions, with IPv6 having a slightly lower delay toward RRDP Publication Points for some regions. Overall, the violin plots show a significant peak above 100 milliseconds for most of the continent, except for skewed distribution data points, which are lower than 100 ($10^2$) milliseconds for Europe and North America, indicating a high concentration of data points. Generally, RRDP PPs are accessible faster than the rsync PPs globally.

Nearly 75% of data points from Africa and Oceania observe an RTT delay of over 300 milliseconds accessing RRDP and rsync PPs, and this number is around 50% for accessing RRDP PPs. RIR's PPs have the lowest RTT delay in their primary serving region, like AFRINIC PP being fast in African countries only and LACNIC being fast in South America and users accessing them from other regions experiencing a relatively high RTT of more than 300 milliseconds on average.

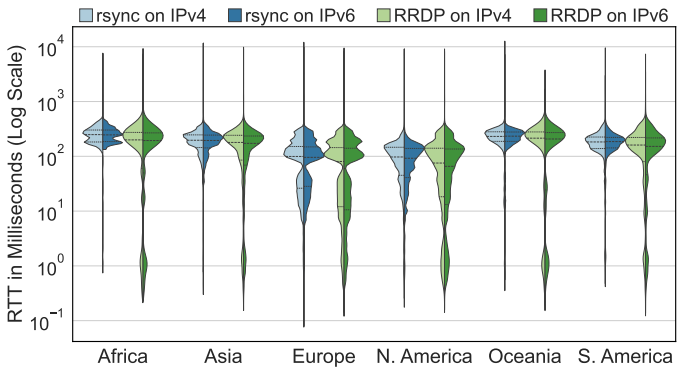


Fig. 4. Violin plots showing the distribution of IPv4 and IPv6 RTT delay (in milliseconds, log scale) to RPKI rsync and RRDP Publication Point hostnames.

Europe and North America generally have the fastest access to RPKI resources, possibly due to most PPs being hosted in these regions. Africa, Oceania, and South American countries observe the highest RTT delay in accessing RPKI resources.

Eight of the RRDP PPs, including *rrdp.apnic.net, rrdp.ripe.net*, all using content distribution networks (CDNs) like Cloudflare, Akamai, Amazon, and Google, have lower than 10 ($10^1$) millisecond RTT delay for all regions.

### C. RTT Delay to RIRs' RPKI Resources

We use IPv4-only data and select RRDP-only repositories because they are the primary PPs for RP synchronization. If RRDP fails, the RP switches to the rsync protocol. The RTT delay in accessing RPKI PPs from IPv4 and IPv6 are similar. Europe and North America have faster access to RPKI resources, and RRDP PPs have lower delays than the rsync PPs (see Section V-B for details).

We measure the RTT delay in accessing each RIR's TA Publication Point and its delegated Publication Points from every continent. The violin plots in Figure 5 exhibit the RTT delay distribution for accessing RRDP PPs of each RIR's RPKI tree globally.

The shapes of violin plots reflect that for most of the cases (with few exceptions), each RIR's Publication Points, including the delegated ones, have a relatively lower accessing RTT in their primary serving continent and a high RTT delay from other regions. For example, the AFRINIC, ARIN, LACNIC, and RIPE have, on average, an RTT delay range of less than one to nearly 90 milliseconds for their primary serving continents, respectively, and a 100 to nearly 700 milliseconds RTT delay for other regions. The prime reason for this difference is that the PPs are hosted in those regions. We

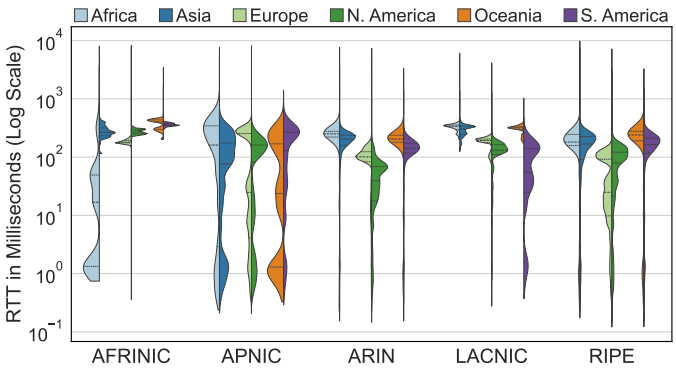


Fig. 5. Violin plots showing the distribution of RTT (in milliseconds, log scale, IPv4) in accessing RPKI RRDP Publication Point hostnames of each RIR's RPKI tree.

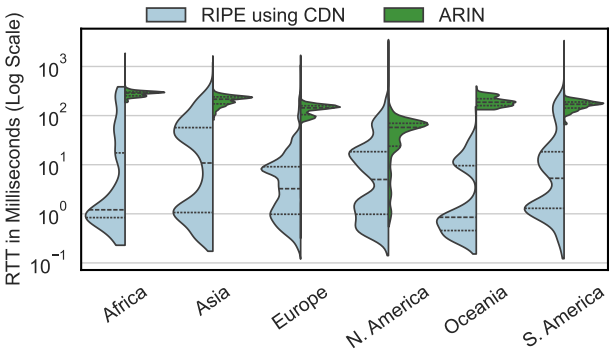


Fig. 6. Violin plot showing the distribution of RTT (in milliseconds, log scale) to RPKI Publication Point hosted on CDN vs. None-CDN, from continents.

identify that a few of APNIC's delegated PPs use international cloud infrastructure. For example, *rpki.owl.net*, originally from Vanuatu islands of Australia, and *rrdp.rp.ki* using xTom[37] and Misaka Network [38] cloud infrastructures, respectively, that has data centers in Oceania, North America, and Europe causing the lower bump of high-density RTT smaller than 10 milliseconds for these regions.

AFRINIC PP has more than 200 milliseconds RTT delay for other continents, except Africa, with Europe being slightly lower than the rest. Europe also has the lowest RTT delay in accessing APNIC RPKI tree Publication Points, with North America and Asia being the second and third with the lowest RTT. After North America, which has the lowest RTT delay to ARIN's RPKI tree PPs, Europe and South America are second and third with the lowest RTT delay. Africa observes a high RTT of 300 to 800 milliseconds when accessing ARIN's PPs. While Asia, Africa, and Oceania observe a high RTT delay in the range of 400 milliseconds to 700 milliseconds, the vast majority of probes from Europe and North America have RTT delay in the range of 100 milliseconds to 500 milliseconds RTT toward LACNIC PPs. The RIPE PPs have the second lowest RTT, the first being Europe, for North America, with nearly 50% RTT delay being below 100 milliseconds. Asia, Africa, and South America have a similar pattern of RTT delay, with more than 100 milliseconds on average for nearly 75% of the data. Oceania has the highest range of RTT delay of above 200 milliseconds towards RIPE PPs for more than 50% of the data.

For both APNIC and RIPE RPKI PPs, we observe densities of data with RTT of less than 10 milliseconds for other regions apart from their primary serving zone. Using the IP2ASN [39] and BGPtools [40], we identify CDN service providers that serve the fastest PPs in RPKI.

**Help of CDN:** To quantify the added benefits of serving PPs from a CDN, we compare IPv4 RTTs for the two largest RIR PPs: RIPE's CDN-hosted PP *rrdp.ripe.net* and ARIN's *rrdp.arin.net* self-hosted PP. Here, we consider only RRDP hostnames since rsync cannot be served by CDNs. Figure 6 illustrates the continent-level distribution RTT delay for accessing RIPE and ARIN's PPs. ARIN's PP has, on average, a high RTT delay of more than 100 milliseconds, peaking at 300 milliseconds, for more than 90% of the data from all continents except North America. Accessing RIPE PP has a much lower RTT, peaking at lower than 10 milliseconds, with some around 1 milliseconds, for all the continents. A few networks with high RTT delays shown on the upper part of the violin plot might be due to the slow local infrastructure at a particular time or the impact of routing policies leading traffic to a CDN location with a high RTT. a high RTT. More than 75% of the RTT data on accessing ARIN and RIPE PPs from regions around the world, apart from their primary serving regions, show 100 milliseconds RTT delay for ARIN PP and less than 10 milliseconds RTT delay for RIPE PP. This indicates RIPE PP is accessible ten times faster for 75% of the cases.

**Artificial Delay:** To measure the impact of RTT delay on RP synchronization, we artificially add RTT delay for all the egress packets from the measurement machine in the network path and re-run the RP software to synchronize with RPKI. For Cold mode, every additional 100 milliseconds of RTT adds around 25 seconds and 15 seconds extra delay, respectively, to the RP synchronization process of Routinator and rpki-client. For the Warm mode, every 100 seconds of extra RTT delay adds around 5 seconds for Routinator and 3 seconds for rpki-client.

## VI. Discussion

The reason for the RPKI synchronization delay varies based on the selection of RP software. Routinator has the faster cryptographic verification of RPKI objects, and rpki-client has efficient network handling in RPKI synchronization. This difference between the implementation of RP software indicates a potential room for optimization for Routinator to handle the network activities efficiently and for the rpki-client to perform the cryptographic verification process faster. As RPKI synchronization requires a high number of read and write operations, using a memory-mapped file system can further optimize the delay.

RIPE and APNIC consolidate multiple prefixes into a single ROA object, which is cheaper regarding resource consumption (CPU load, disk space) and more beneficial during the RP synchronization process than one prefix per ROA. Conversely, ARIN, AFRINIC, and LACNIC have a simpler management of ROA objects of more or less one prefix per ROA. However, using a single ROA for multiple prefixes complicates ROA management, for example the expiration or change for a single prefix requires to reissue the ROA for all prefixes.

In terms of management, using ROA with a single IP prefix is simpler and thus recommended [41], [42]. One of the RIRs acknowledged aggregating multiple prefixes for the same ASN on a single ROA, which needs a complex system that has to carefully track data changes and reissue ROA objects proactively without risking IP prefix invalidation. Adding a new prefix to RPKI means that the CA should maintain a state for the existing ROAs, revoke the ROA that should be updated, and create a new one with the new prefix, a time-consuming task for CAs but beneficial for RP synchronization.

The historical trend of registering prefixes to the RPKI [43], reported growth of IPv6 prefixes in global routing table [44], usage of small CIDR sizes or hyperspecific prefixes [45], with some prefixes used by multiple origin ASes [46] results in a higher number of VRPs coming in the RPKI ecosystem. Delegated RPKI permits a distributed and flexible architecture but may introduce complexity and potential delays in RPKI synchronization. Despite the current and relatively low numbers of delegated repositories, our findings reveal that RPs hang on specific delegated Publication Points for hundreds of seconds, making the entire RPKI synchronization process slow. The two studied RPs implement timeout mechanisms to skip unresponsive Publication Points, yet as the RPKI infrastructure and the number of RPKI objects (e.g., ROAs) is growing, we expect a larger number of delegated CAs; hence, quick synchronization will become even more challenging.

The current Publication Points deployment of RPKI data lacks a worldwide quick-to-access setup, with most of the PPs being accessible fast only in their primary serving regions. Serving the RPKI data from a worldwide, easy-to-access infrastructure like a CDN will help reduce the network delay in the RPKI synchronization process. We observe that RRDP PPs with cachable data on CDN are quicker to access globally than the PP used for rsync, which is not cachable on CDN. Ideally, serving all the RPKI data using RRDP might be faster. However, in practice, we encounter cases where a Publication Point can not serve data using RRDP, and they switch to the rsync protocol, which has known vulnerabilities [47] and slows the RP synchronization.

The rapid changes in RPKI software and RP implementations make RPKI a challenging and moving target for this study. We found two Publication Points with no IPv6 addresses, resulting in a disparate number of ROAs when validating from an IPv6 RP; however, the problem was resolved within a few months. Introduction of new object types like AS Provider Authorization, ASPA [48], Signed Prefix List [49], improved validation procedure [50], are examples of the rapid changes in the RPKI landscape. Thus, we believe RPKI synchronization delay may change over time and needs constant attention from the networking community.

We acknowledge that conducting RP synchronization from diverse locations with additional RP software would provide us with broader coverage of how networks experience RP synchronization delay. Despite limitations, our approach is suitable, and our analysis identifies the delay factors in the RP synchronization process and provides valuable insights on how to improve them.

## VII. Conclusion

Using two RP software, we thoroughly analyze the elements that induce RP synchronization delay, including the structure of ROAs, the impact of delegation in RPKI resource hosting, and the latency to Publication Points. Network delay and the cryptographic verification delay of RPKI data are the two main elements of delay in RPKI synchronization. Routinator is fast in cryptographic verification and robust in handling empty CPU cache, but suffers from many page faults during data retrieval from the Internet. Therefore, it has the highest delay of nearly 500 seconds in Cold mode and the lowest delay of 40 seconds in Hot mode. rpki-client efficiently uses the network with almost no page faults, but it takes double the time of Routinator for cryptographic verification of ROAs and is more prone to the system's CPU cache state. Consequently, it has around 190 seconds, less than half of the Routinator delay in Cold mode, and more than double of Routinator in Hot mode, and a similar delay in Warm mode. In normal operations, RPs typically run in Warm mode, so rpki-client and Routinator will have a very similar synchronization delay in day-to-day performance.

Bundling several prefixes of an AS into a single ROA can reduce the synchronization time by a factor of three, and is possible for more than 60% of ASes in RPKI. Delegated CAs hosted on a fast-to-access infrastructure positively impact the RPKI validation rate, but for ARIN, 7% of VRPs from delegated CAs cause more than 90% delay.

We highlight regional disparities in accessing RPKI PPs, with Europe and North America experiencing lower latencies and other regions significantly higher latencies in accessing RPKI resources. The current PP deployment delays reveal that each RIR's RPKI tree is accessible relatively faster in their primary serving regions and slower from other regions. Our results show that CDN usage can reduce the PP accessing time by up to 10 times in some regions.

We quantify that every additional 100 milliseconds of extra delay in the network path will increase the RP synchronization by 25 seconds for Routinator and 10 seconds for rpki-client in Cold cache mode, and almost half of the values in Warm mode.

## Acknowledgment

We thank the anonymous reviewers of our paper for their valuable feedback. We also thank the Internet Society for their partial financial support of the research under the MANRS research fellowship program.

## REFERENCES


[1] R. Bush and R. Austein, *The Resource Public Key Infrastructure (RPKI) to Router Protocol, Version 1*, RFC 8210, Sep. 2017. [Online]. Available: https://www.rfc-editor.org/info/rfc8210.

[2] C. Lynn, S. Kent, and K. Seo, *X. 509 extensions for IP addresses and AS identifiers*, 2004.

[3] M. Lepinski and S. Kent, *RFC 6480: an infrastructure to support secure Internet routing*, 2012.

[4] M. Candela, "RPKI Automation and Monitoring," in *LACNIC 35*, 2021.

[5] M. Candela, "NTT's RPKI Deployment Update," in *RIPE 82*, 2021.

[6] M. Candela, "A one-year review of RPKI operations," in *RIPE 84*, 2022.

[7] G. Huston, *BGP Updates in 2024*, 2025. [Online]. Available: https://blog.apnic.net/2025/01/07/bgp-updates-in-2024/.

[8] B. Zhang, D. Massey, and L. Zhang, "Destination reachability and BGP convergence time [border gateway routing protocol]," in *IEEE Global Telecommunications Conference, 2004. GLOBECOM'04.*, 2004.

[9] A. García-Martínez and M. Bagnulo, "Measuring bgp route propagation times," *IEEE Communications Letters*, vol. 23, no. 12, 2019.

[10] R. Fontugne, A. Phokeer, C. Pelsser, K. Vermeulen, and R. Bush, "RPKI Time-of-Flight: Tracking Delays in the Management, Control, and Data Planes," in *PAM*, 2023.

[11] National Institute of Standards and Technology (NIST), *RPKI Monitor*, 2025. [Online]. Available: https://rpki-monitor.antd.nist.gov.

[12] T. Bruijnzeels, O. Muravskiy, B. Weber, and R. Austein, *RFC 8182: The RPKI Repository Delta Protocol (RRDP)*, 2017.

[13] R. Bush and R. Austein, *RFC 6810: The Resource Public Key Infrastructure (RPKI) to Router Protocol*, 2013.

[14] E. Osterweil, T. Manderson, R. White, and D. McPherson, "Sizing estimates for a fully deployed rpki," Verisign Labs, Technical Report 1120005, 2012. [Online]. Available: https://cs.gmu.edu/~eoster/doc/sizing.pdf.

[15] F. V. Silveira, *RPKI Repositories and the RIPE Database in the Cloud*, 2021. [Online]. Available: https://labs.ripe.net/author/felipe_victolla_silveira/rpki-repositories-and-the-ripe-database-in-the-cloud/.

[16] J. Kristoff, R. Bush, C. Kanich, G. Michaelson, A. Phokeer, T. C. Schmidt, and M. Wählisch, "On measuring rpki relying parties," in *ACM IMC*, 2020.

[17] N. Labs, *Routinator manual*, https://routinator.docs.nlnetlabs.nl/en/stable/manual-page.html, 2024.

[18] O. Project, *rpki-client Manual*, https://man.openbsd.org/rpki-client, 2024.

[19] T. Hlavacek, P. Jeitner, D. Mirdita, H. Shulman, and M. Waidner, "Behind the scenes of rpki," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022.

[20] T. Hlavacek, P. Jeitner, D. Mirdita, H. Shulman, and M. Waidner, "Stalloris: RPKI Downgrade Attack," in *USENIX Security*, 2022.

[21] T. Hlavacek, P. Jeitner, D. Mirdita, H. Shulman, and M. Waidner, "Beyond limits: How to disable validators in secure networks," in *ACM SIGCOMM*, 2023.

[22] K. van Hove, J. van der Ham-de Vos, and R. van Rijswijk-Deij, "rpkiller: Threat Analysis of the BGP Resource Public Key Infrastructure," *Digital Threats: Research and Practice*, vol. 4, no. 4, 2023.

[23] A. Reuter, M. Wählisch, and T. C. Schmidt, "RPKI MIRO: Monitoring and Inspection of RPKI Objects," *ACM SIGCOMM CCR*, vol. 45, no. 4, 2015.

[24] A. Reuter, R. Bush, I. Cunha, E. Katz-Bassett, T. C. Schmidt, and M. Wählisch, "Towards a rigorous methodology for measuring adoption of RPKI route validation and filtering," *ACM SIGCOMM CCR*, vol. 48, no. 1, 2018.

[25] T. Chung, E. Aben, T. Bruijnzeels, B. Chandrasekaran, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, R. v. Rijswijk-Deij, J. Rula, *et al.*, "RPKI is coming of age: A longitudinal study of RPKI deployment and invalid route origins," in *ACM IMC*, 2019.

[26] C. Testart, P. Richter, A. King, A. Dainotti, and D. Clark, "To Filter or Not to Filter: Measuring the Benefits of Registering in the RPKI Today," in *PAM*, 2020.

[27] W. Li, Z. Lin, M. I. Ashiq, E. Aben, R. Fontugne, A. Phokeer, and T. Chung, "RoVista: Measuring and Analyzing the Route Origin Validation (ROV) in RPKI," in *ACM IMC*, 2023.

[28] N. Rodday, Í. Cunha, R. Bush, E. Katz-Bassett, G. D. Rodosek, T. C. Schmidt, and M. Wählisch, "The Resource Public Key Infrastructure (RPKI): A Survey on Measurements and Future Prospects," *IEEE Transactions on Network and Service Management*, vol. 21, no. 2, 2023.

[29] D. Mirdita, H. Schulmann, and M. Waidner, *SoK: An Introspective Analysis of RPKI Security*, 2024. eprint: arXiv:2408.12359. [Online]. Available: https://arxiv.org/abs/2408.12359.

[30] D. Mirdita, H. Schulmann, N. Vogel, and M. Waidner, *The CURE to vulnerabilities in RPKI validation*, 2023.

[31] H. Schulmann, N. Vogel, and M. Waidner, *RPKI: Not Perfect But Good Enough*, 2024. [Online]. Available: https://arxiv.org/abs/2409.14518.

[32] Y. Li, H. Zou, Y. Chen, Y. Xu, Z. Ma, D. Ma, Y. Hu, and G. Xie, "The Hanging ROA: A Secure and Scalable Encoding Scheme for Route Origin Authorization," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, 2022.

[33] Y. Gilad, O. Sagga, and S. Goldberg, "Maxlength Considered Harmful to the RPKI," in *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, 2017.

[34] D. Kristaps, J. Claudio, S. Job, d. R. Theo, T. Sebastian Benoit, and Buehler, *rpki-client*, Available at https://www.rpki-client.org, 2023.

[35] N. Labs, *Routinator 3000*, Available at https://www.nlnetlabs.nl/projects/rpki/routinator/, 2023.

[36] J. Frieß, D. Mirdita, H. Schulmann, and M. Waidner, "Byzantine-Secure Relying Party for Resilient RPKI," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024.

[37] xTom, *Xtom facilities*, 2025. [Online]. Available: https://xtom.com/facilities/.

[38] Misaka Network, Inc., *Misaka network*. [Online]. Available: https://www.misaka.io.

[39] I. H. Report, *Ip2asn*, 2025. [Online]. Available: https://github.com/InternetHealthReport/ip2asn.

[40] B. Tools, *BGP Tools*, 2025. [Online]. Available: https://bgp.tools.

[41] Z. Yan, R. Bush, G. Geng, T. de Kock, and J. Yao, *Avoiding Route Origin Authorizations (ROAs) Containing Multiple IP Prefixes*, RFC 9455, Aug. 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9455.

[42] Z. Lai, Z. Yan, G. Geng, and H. Nakazato, "Issuance Policies of Route Origin Authorization with a Single Prefix and Multiple Prefixes: A Comparative Analysis," *International Journal of Advanced Computer Science and Applications*, vol. 15, no. 3, 2024. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2024.01503116.

[43] National Institute of Standards and Technology (NIST), *RPKI Monitor*, https://rpki-monitor.antd.nist.gov, 2025. (visited on 01/16/2025).

[44] Potaroo, *BGP Routing Table Analysis*, https://bgp.potaroo.net/v6/as2.0/index.html, 2024.

[45] K. Z. Sediqi, L. Prehn, and O. Gasser, "Hyper-specific prefixes: Gotta enjoy the little things in interdomain routing," *ACM SIGCOMM CCR*, vol. 52, no. 2, 2022.

[46] K. Z. Sediqi, A. Feldmann, and O. Gasser, "Live long and prosper: Analyzing long-lived moas prefixes in bgp," in *IEEE TMA*, 2023.

[47] Computer Incident Response Center Luxembourg (CIRCL), *Vulnerability bundle: D938dc28-6877-40db-ad5f-25f3051288e6*, 2025. [Online]. Available: https://vulnerability.circl.lu/bundle/d938dc28-6877-40db-ad5f-25f3051288e6.

[48] A. Azimov, E. Bogomazov, R. Bush, K. Patel, J. Snijders, and K. Sriram, "BGP AS_PATH Verification Based on Autonomous System Provider Authorization (ASPA) Objects," Internet Engineering Task Force, Internet-Draft draft-ietf-sidrops-aspa-verification-19, 2024. [Online]. Available: https://www.ietf.org/archive/id/draft-ietf-sidrops-aspa-verification-19.html.

[49] J. Snijders and G. Huston, *A profile for Signed Prefix Lists for Use in the Resource Public Key Infrastructure (RPKI)*, https://datatracker.ietf.org/doc/draft-ietf-sidrops-rpki-prefixlist/, 2024.

[50] J. Snijders and B. Maddison, *RPKI Validation Re-reconsidered*, https://datatracker.ietf.org/doc/draft-spaghetti-sidrops-rpki-validation-update-03/, 2023.

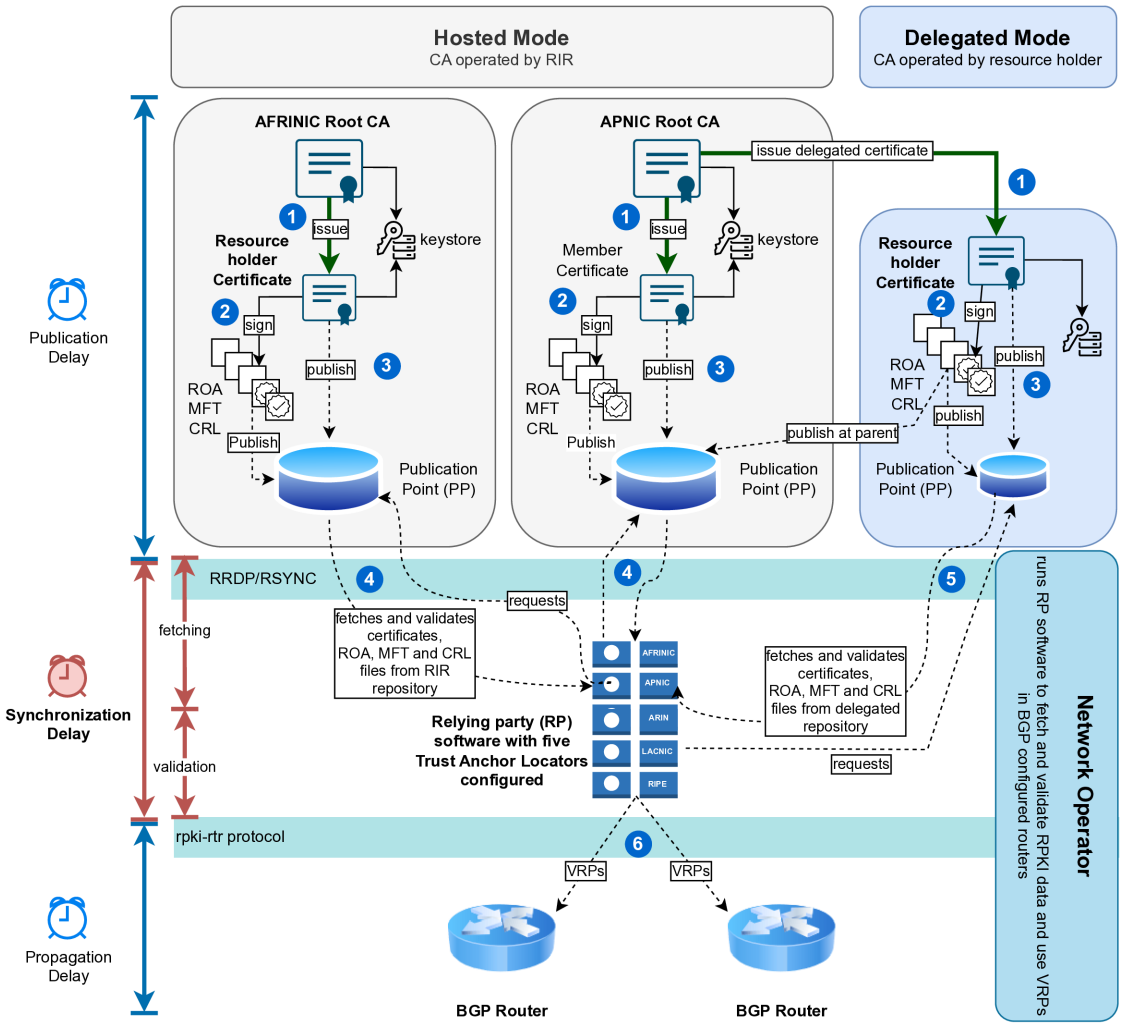


Fig. 7. RPKI components and modes of operation. The red part highlights the focus of this paper.

## APPENDIX
## ETHICS

This work does not raise any ethical issues.

### A. RPKI Validation Rate at CA Depth

Figure 8 illustrates the RPKI data validation rate or the number of VRPs validated per second at each CA depth. For APNIC, delegated CAs positively impact RP synchronization and increase the VRP validation rate from 1300 VRPs to almost 1500 VRPs per second. For ARIN, the delegated CAs reduce the RPKI validation by almost half, from 800 VRPS per second at a CA depth of 3 to around 450 VRPs at depths of 4 and 5. The RIPE delegated CA at depth reduces the validation rate of VRPs from 800 to nearly 750 VRPs per second. The impact of outliers, as explained in Figure 3, reducing the validation rate of RPKI data is visible here as well.

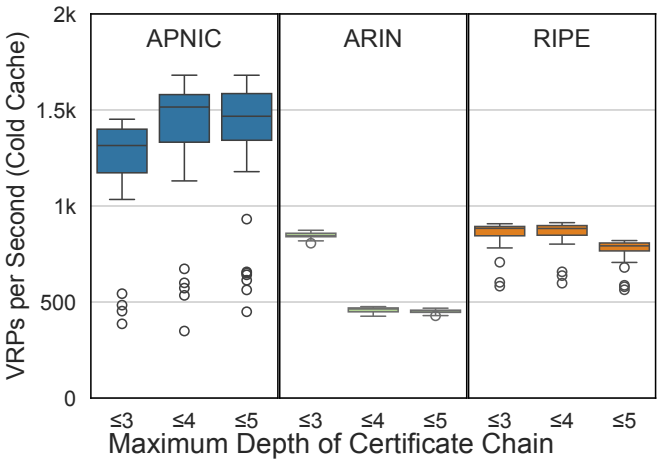


Fig. 8. RP synchronization rate of VRPs per second at various Certificate chain depths of TAs.