RTRlib: An Open-Source Library in C for RPKI-based Prefix Origin Validation

Matthias Wählisch
Freie Universität Berlin
waehlisch@ieee.o.rg

Fabian Holler
Hamburg University of Applied Sciences
mail@fholler.de

Thomas C. Schmidt
Hamburg University of Applied Sciences
t.schmidt@ieee.org

Jochen H. Schiller
Freie Universität Berlin
jochen.schiller@fu-berlin.de

Abstract

A major step towards secure Internet backbone routing started with the deployment of the Resource Public Key Infrastructure (RPKI). It allows for the cryptographic strong binding of an IP prefix and autonomous systems that are legitimate to originate this prefix. A fundamental design choice of RPKI-based prefix origin validation is the avoidance of cryptographic load at BGP routers. Cryptographic verifications will be performed only by cache servers, which deliver valid AS/prefix mappings to the RPKI-enabled BGP router using the RPKI/RTR protocol.

In this paper, we give first insights into the additional system load introduced by RPKI at BGP routers. For this purpose, we design and implement a highly efficient C library of the RPKI/RTR router part and the prefix origin validation scheme. It fetches and stores validated prefix origin data from an RTR-cache and performs origin verification of prefixes as obtained from BGP updates. We measure a relatively small overhead of origin validation on commodity hardware (5% more RAM than required for full BGP table support, 0.41% load in case of $\approx$ 92,000 prefix updates per minute), which meets real-world requirements of today.

1 Introduction

The current Internet backbone is quite threatened by misconfigurations as well as intended attacks such as prefix hijacks. Both lead to disturbances on the BGP layer. A common threat is the incorrect announcement of the origin autonomous system (AS). Several prominent examples are known like the recent (April 2010) cause of redirection of 15% US traffic to China due to incorrectly announced IP prefixes [1, 243 et seqq.].

Securing BGP has been discussed in the research community since more than one decade [2]. Current efforts of the Secure-Inter Domain (SIDR) working group within the IETF lie in the standardization of a set of protocols to enhance the security of BGP. They focus on solving two problems: enable a router (a) to verify that a BGP update did originate at an authorized AS and (b) to verify that the AS path within the BGP update corresponds to the route traversed. Even though the latter is far from global deployment, first steps have been performed to tackle the first challenge.

An integral part for securing BGP is the Resource Public Key Infrastructure (RPKI) [3]. The RPKI is a robust security framework. It consists of a distributed repository that stores certificates and Route Origin Authorization (ROAs). ROAs provide a secure binding between an IP prefix and an AS that is allowed to originate that prefix. For a nice overview on this topic we refer to [4]. RPKI-enabled routers do not store ROAs itself but only the validated content of these (ROA) authorities. The validation of ROAs will be performed by trusted cache servers, which will be deployed at the network operator site. The RPKI/Router (or short RPKI/RTR) protocol [5] defines a standard mechanism to maintain the exchange of the prefix/origin AS mapping between the cache server and routers. In combination with a BGP prefix origin validation scheme [6] a router is able to verify received BGP updates without suffering from cryptographic complexity.

In this paper, we study the overhead introduced by RPKI-based prefix origin validation at routers. Current implementations of the RPKI/RTR protocol do not allow for detailed analysis as they are either closed source or have not been developed for real deployment. To achieve our goal, we carefully designed and implemented a lightweight C library that implements (a) the client part of the RPKI/Router protocol and (b) prefix origin validation. This library, called RTRlib, provides functions to establish a connection to a single or multiple trusted caches and to determine the correctness of a prefix/origin AS mapping. We apply this library in experiments and real-world measurements and give first insights into the load behaviour of RPKI-enabled commodity routers. Based on our analysis we show that the overhead of RPKI-based origin validation at BGP routers is negligible for current and increasing update rates.

Built as a library RTRlib can be included in various application scenarios beyond this work. It is useful for researchers, network operators, and for developers of routing software. Researchers may use the library to get a better understanding of prefix origin validation and identify improvements (e.g., performance evaluation). Network operators may apply the RTRlib to develop monitoring tools (e.g., to detect RPKI misconfiguration in 'real-time'). Developers can integrate the RTRlib into a BGP daemon to extend their implementation towards RPKI. We argue that in particular the research community benefits from an implementation which is suitable for both, experiments and real-world deployment as this allows for comparable and realistic analysis.

The remainder of this paper is structured as follows. Section 2 presents the design and architecture of the RTRlib. Section 3 analyses characteristic performance aspects. In Section 4 we give a short overview about related work. We conclude in Section 5.

2 RTRlib

For RPKI-based prefix origin verification, the RPKI/RTR protocol needs to be implemented on routers. The RTRlib assembles the required functions as an external independent library, which simplifies code reuse. Existing BGP daemons can be extended by simply integrating the library or parts of it. We provide a Quagga extension, which allows for RFC-compliant prefix origin validation. The same code base may also be used to build tools for researchers or ISPs. All source code of the RTRlib project is released under GNU LGPL. In addition to the basic protocol functions, the source code package includes command line tools to get quickly in touch with RPKI/RTR protocol and debug origin validation. The RTRlib is available at `http://rpki.realmv6.org/`.

2.1 Design

Our implementation of the RPKI/RTR protocol considers the subsequent design goals:

Broad system integration The library shall run on different system environments and thus minimize dependencies on specific operating system calls and third party tools. We implement the library in C because several BGP daemons are written in C (e.g., BIRD, Quagga) or C++ (e.g., XORP). In contrast to C++, C-functions can be invoked or adopted into other C/C++ programs without modifications. To facilitate the smooth migration to a variety of operating systems, our library is based on POSIX interfaces.

Interoperability The implementation shall be able to exchange data with existing and upcoming RTR cache servers. The presented RPKI/RTR library implements the latest version of the protocol specification. We performed interop tests with the RPKI/RTR cache servers available [7], which helped to reveal errors on both sites.

Extensibility The library shall be easy to modify for supporting upcoming protocol changes and extensions as well as specific user demands. To support this requirement, the library consists of separate components with high cohesion and low coupling. Sufficient abstraction allows easy exchange of the modules.

Efficiency BGP routers are confronted with high BGP update rates and must currently store more than 400,000 IP prefixes. With respect to memory and CPU consumptions, the RPKI/RTR implementation shall be prepared to handle these data even though not all prefixes are part of the RPKI at the moment. In addition, the implementation should minimize the internal overhead. We designed our library to scale very well with current and upcoming BGP data. It does not require specific hardware, but runs on commodity devices and thus supports an easy migration to RPKI origin validation without introducing additional costs.

2.2 Architecture

Our implementation follows a flexible design. The software architecture includes different layers to simplify the extension or an exchange of individual parts as shown in Figure 1. The lowest layer of the architecture is built by the *transport sockets*. They allow for the implementation of different transport channels that provide a common interface to exchange PDUs with the cache (i.e., the RPKI/RTR server). The current version of RTRlib supports unprotected TCP and SSH.

On top of the transport layer, the *RTR socket* uses a transport socket for RTR-specific data exchange with the RPKI/RTR server. The RTR socket implements the RPKI/RTR protocol, i.e., fetches validation records and stores them in a prefix table data structure.

The *prefix validation table* stores validated prefix origin data. This abstract data structure provides a common interface to add and delete entries as well as to verify a specific prefix. Its implementation is crucial as the data structure stores all prefixes received from the cache

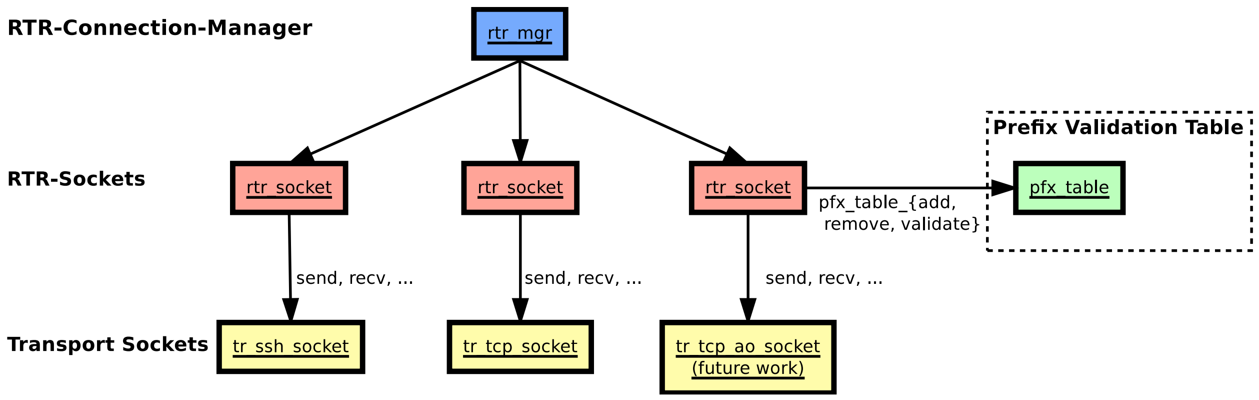


Figure 1: Software architecture of RTRlib

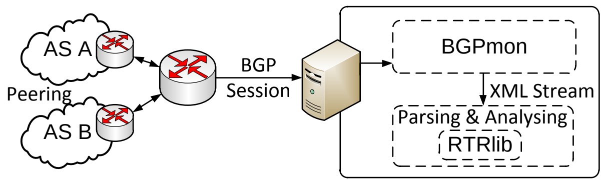


Figure 2: Principle monitoring setup: Prefix origin validation without changig BGP router firmware

servers (i.e., low memory overhead required) and is responsible to perform prefix lookup for the BGP updates (i.e., find validated IP prefixes very fast). Our library implements a Longest Prefix First Search Tree (LPFST) [8], but can be extended to other data structures. In contrast to common data structures for IP prefix lookup such as Tries or Patricia, the LPFST needs fewer memory access and exhibits lower memory overhead [8].

Internally, the RTRlib uses two separate prefix validation tables, one for IPv4 records and one for IPv6 records. This makes tree operations (insert, delete, find) more efficient as the height per tree is lower in contrast to a combined IPv4/v6 tree. The appropriate prefix validation table will be chosen according to the IP version.

On top of the modular architecture, the *RTR connection manager* maintains connections to multiple RTR servers. This includes failover mechanisms. It represents the main interface for users of the library.

2.3 Use Cases

Online experiments: Changing the firmware of a deployed BGP router for research experiments is an unpopular option. To emulate prefix origin validation of dedicated BGP peers in real-time, we propose a mirroring concept (cf., Figure 2). The measurement node deploys BGPmon [9] and the RTRlib. BGPmon allows for the establishment of (unidirectional) peering sessions and transforms the received BGP update data into an XML stream. This XML stream can be parsed with very low additional costs. After extracting prefix, netmask, and origin AS the update data will be validated by the RTRlib. An operator needs only to add an additional peering session at the BGP router. Further configuration changes are not required. As a real-world use-case we started the deployment of this setup at one of Germany's largest Internet Exchange Points to provide IXP members with most current analysis of their prefixes.

Offline experiments: The design of the RTRlib allows for easy embedding of the library into Python or Perl scripts, which are part of the common toolchain in analysing BGP dump data.

3 Performance Analysis

In this section, we analyse the runtime performance and the scaling behaviour of RPKI-based origin validation at the system level based on our reference implementation. The experiments have been conducted on commodity hardware, i.e., a dual AMD Opteron 280 processor (2.4 GHz) and 8 GB RAM with Linux Ubuntu (2.6.32-33 kernel) as underlying operating system.

We explicitly note that any comparison with other implementations of the RPKI/RTR router part would be unfair because they either are not designed for real-time purposes (rpki.net) or do not allow for RPKI-specific system measurements (Cisco/Juniper).

3.1 Memory Consumption

The memory consumption of RTRlib mainly depends on the number of prefixes inserted into the prefix validation table. Considering a 64 bit architecture with 8 bytes per pointer, a single record within the prefix validation table consumes 78 bytes in our implementation of the longest prefix first search tree. Padding bytes, which maybe be

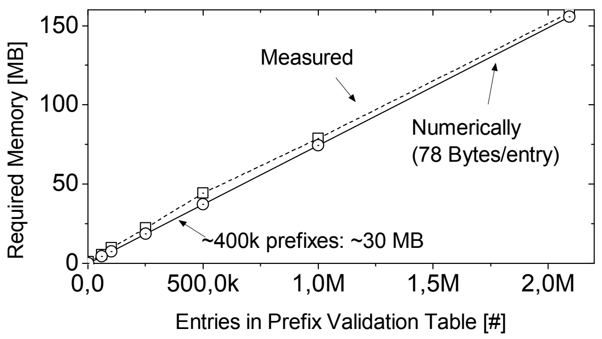


Figure 3: Memory consumption of the RTRlib

inserted by the compiler, are omitted in this calculation. Note, a common BGP route entry requires between 100 and 200 bytes [10].

To measure the memory required on a real system, we added randomly generated prefixes to the prefix validation table. Figure 3 displays the scaling behaviour for different table sizes. The overall memory consumption scales linearly as expected. ROAs for all $\approx$ 400,000 active IP prefixes included in current BGP routing tables would result in additional $\approx$ 30 MB of RAM for an RPKI/RTR-enabled router. Cisco suggests to equip their devices with 512 MB of RAM for storing a complete global BGP routing table [11]. Thus a full RPKI validation table would lead to a 5% increase of RAM—a relatively small overhead. In particular, BGP services implemented on commodity hardware, where several gigabytes of RAM are common, should not suffer from RPKI requirements.

3.2 CPU Consumption

The processing overhead of RPKI/RTR on the router is dominated by the complexity that results from update and lookup operations on the data structure holding the valid ROA information. Update operations on the prefix validation table are triggered by new, modified, or deleted ROAs, whereas lookups follow BGP updates.

A fast **initial creation of the prefix validation table** (e.g., after booting the router) is necessary to start immediately with origin validation of BGP updates. With the first connection to the RPKI/RTR cache server, the complete set of valid ROA data must be (a) transferred to the router and (b) added to the prefix tree. Performance of the first part depends on the network topology, i.e., the distance between router and cache server. The second part is only related to the local system. We keep our setup simple, abstract from (a), and focus on (b). Note, the latency introduced by the cache-router communication is an additive term to the overall delay.

In this experiment we read different sized sets of ROA

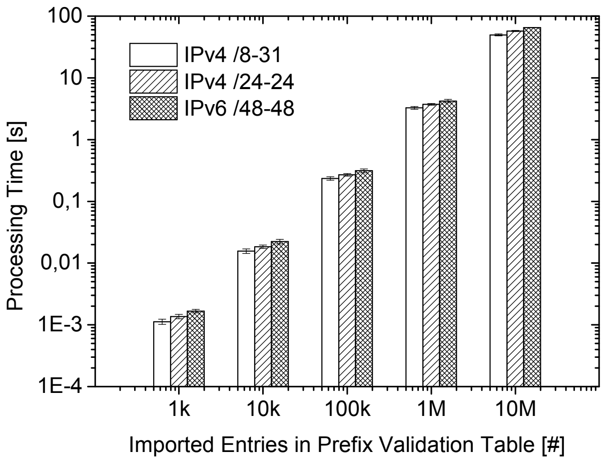


Figure 4: Delay to load a bulk of ROA data into the BGP router for different mask lengths

data from a file and measure the import delay introduced by the RTRlib. Figure 4 visualizes the required time for prefixes with different (minimal and maximal) mask lengths. Inserting 1,000,000 prefix/AS mappings into an empty prefix table needs $\approx$ 4 seconds. Overall, the performance does neither depend on the IP version nor on the ROA configuration. Applying a non-linear fit shows a complexity of $O(n \cdot \log(n))$, with n describes the number of entries.

From a router perspective, the RPKI deployment state is measured by the amount of **additional lookups**, i.e., the ratio of valid (or invalid) versus not found IP prefixes. Once origin verification is enabled, even a prefix that is not part of the RPKI requires a lookup within the prefix validation table. In the worst case, the complete height of the prefix tree must be traversed. In the following, we analyze the CPU load for a varying mixture of validation outcome to quantify overhead based on different states of deployment.

The performance of tree data structures correlates with the tree shape, which is influenced by the inserted data. A future ROA prefix distribution could be derived from the current Internet backbone routing table. However, the Internet itself is a continuously evolving structure, which makes predictions quite hard [12]. In proceeding this way, we evaluate the CPU load by randomly generating 100,000 ROA IPv4 prefixes with a fixed minimum and maximum /24-netmask. The AS number of the authorized origin AS is randomly generated, as well. We consider ratios per validation state of 0%, 25%, 50%, 75%, and 100%. For each combination of all possible validation outcomes (e.g., 25% valid, 50% invalid, and 25% not found BGP prefix announcements), we create a total of 2,000 IP prefixes that match the required states. We emulate a very high BGP update rate of 2,000 verifi-

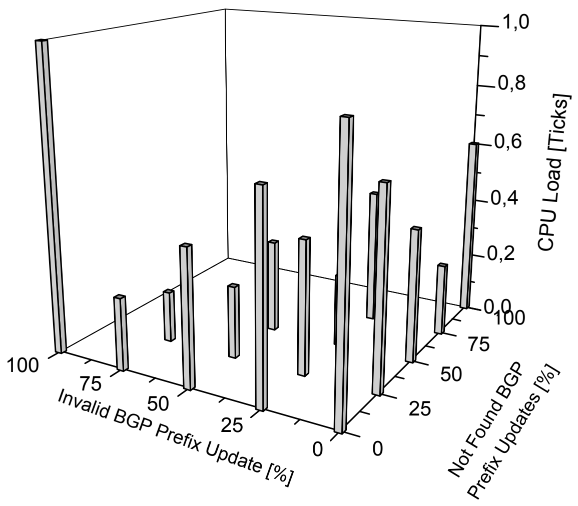
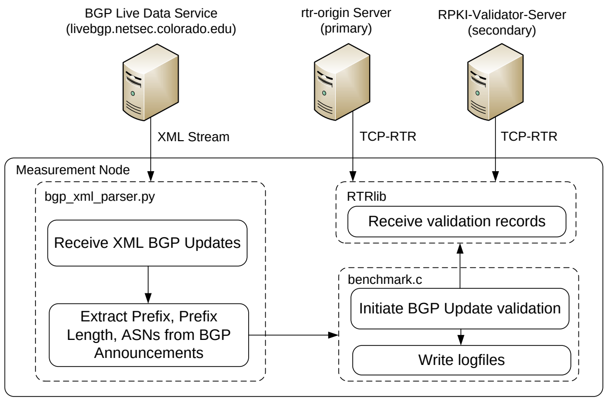


Figure 5: CPU overhead for varying validation outcomes



BGP Live Data Service (livebgp.netsec.colorado.edu)
rtr-origin Server (primary)
RPKI-Validator-Server (secondary)
XML Stream
TCP-RTR
TCP-RTR
Measurement Node
bgp_xml_parser.py
Receive XML BGP Updates
Extract Prefix, Prefix Length, ASNs from BGP Announcements
RTRlib
Receive validation records
benchmark.c
Initiate BGP Update validation
Write logfiles

Figure 6: Real-world measurement setup

cations per second.

The average CPU load is shown in Figure 5. It ranges between 0.17 and 1.0 clock ticks (or jiffies). The CPU load fluctuates $\ll$ 1 tick. It is worth noting that our measurement resolution is bound to the predefined timer interrupt handler of the system, which is 10 ms; and we keep system default configuration to avoid side effects. Many events require less time than covered by the jiffy resolution. This fine-grained scale (maximum 1 jiffy) makes it difficult to predict the impact of the validation state on the system performance in more detail, i.e., is the value close to 10 ms or significantly less. However, we emphasize two observations. First, there is a dependency of the ROA validation mix on the lookup costs. Second, this complexity is—under normal conditions—insignificant and thus does not affect BGP routers. Details on this are part of our future work.

In a second step, we quantify the **CPU load introduced by current ROA data and real BGP updates**.

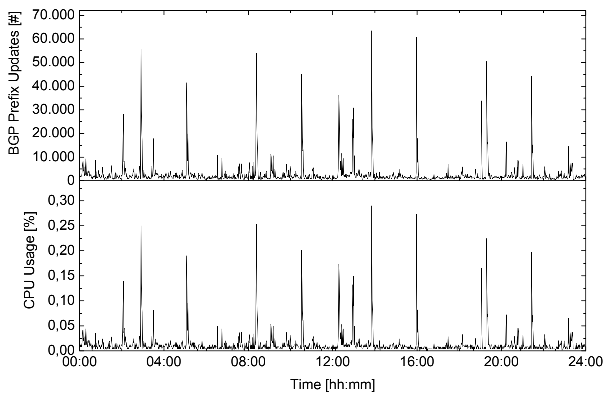


Figure 7: Characteristic CPU load for prefix validation and the number of received prefixes for January 5, 2012

To receive live BGP data, we connect to the BGPmon live stream [13], which provides real-time BGP updates (cf., Figure 6). The BGPmon projects maintains nine direct peerings as well as indirect peerings via three collectors of the RouteViews project. The indirect peering includes routing updates from more than 100 peers such as AOL, Hurricane Electric, and AT&T. The amount of peering relations allows us to experience live BGP Update rates similar to those seen at a larger ISP. We run this setup since January 2012.

The CPU load correlates with the number of prefix validations (i.e., the BGP update rate). Figure 7 visualizes both measurements per minute for January 5, 2012. All other days show the same qualitative behaviour. During the measurement period, we observed a maximum of 92,549 prefix announcements per minute and a maximum CPU load of 0.41%. The average CPU load per day was 0.02% with a standard deviation of 0.04%.

3.3 Future Research Directions

Attacks on RPKI-enabled routers An legitimate owner of an IP prefix may create intentionally AS/prefix mappings that are cryptographically valid but misses a counterpart in the BGP system. Note, there are no restrictions on creating a huge number of ROAs. Without any pre-filtering by the cache server, this will unnecessarily allocate memory at BGP routers and can be misused by an attacker. Another attack is the creation of ROA/BGP data that harms the router with respect to its processing capacities. Our analysis revealed a validation overhead that slightly depends on the result of the BGP update verification (i.e., valid, invalid, or not found). Even though this is not significant in current deployment, an attacker may exploit this dependency.

In this paper we argue for the need of further research that analyzes the vulnerability of RPKI-enabled routers.

The processing complexity is mainly controlled by the concrete data structure in use. The RTRlib allows for a flexible exchange of the embedded data structure, which makes future evaluation of this topic less complicated.

Modeling cache server dynamic The daily load on RPKI-enabled routers is strongly bound to the frequency of creating and updating ROA data as well as the BGP update rate. BGP routing dynamic has been analyzed in great detail during the last years (e.g., [14], [15], [16]). One might argue that the ROA dynamic implicitly follows BGP announcements, as a new IP prefix requires a valid ROA in a full RPKI deployment. Such a correlation is not obvious. ROAs can be created in advance for backup situations or missed due to misconfiguration, for example.

We considered extreme cases as well as real world BGP and ROA streams, which help to anticipate a general performance space. However, at the moment it is too early to finally judge the typical load signature, because the RPKI deployment is still in vivid progress. We expect models that describe the dynamic of ROA updates when the data within the RPKI repositories is more stable.[1]

Cache server placement The total delay for adding ROA data to the BGP router depends also on the placement of the cache server within the network topology relatively to the BGP router. There are two reasons to place the cache server in the vicinity of the router. First, topological closeness allows to omit prefix origin validation without introducing vulnerability. Secondly, low latencies reduce the transmission time to update the prefix origin validation table. On the other hand, operating multiple primary cache servers increases maintenance costs.

Our analysis showed that a router is able to create the prefix validation table fast. The RTRlib loads one million entries in approximately four seconds. For n entries, the asymptotic overhead is $O(n \cdot \log(n))$. Combining network latency and processing costs may help to adjust the placement question.

Online monitoring to reveal prefix hijacks A recent, preliminary study of the RPKI [17] analyzed the validation outcome of BGP updates in detail. The authors found that a surprisingly large number of updates are invalids according to the RPKI validation process. Most of them are very likely the result of misconfigurations of the RPKI data. Separating those misconfigurations from real prefix hijacks is a challenge that needs to be addressed in the future. We consider the RTRlib as a helpful tool that complies with the real-time demands caused by the corresponding research questions.

[1]For an early discussion of this topic, we refer to the very recent email thread at `http://www.ietf.org/mail-archive/web/sidr/current/msg05346.html`, November 2012.

4 Related Work

Support for the RPKI/RTR cache is available and deployed, the number of RPKI/RTR clients, though, is limited [7]. Cisco and Juniper recently implemented RPKI-based prefix origin validation in their firmware. These implementations are not fully open-source. More importantly, measuring system behaviour on a fine-grained base is difficult up to impossible in those closed environments.

The BGP–SRx framework [18] provides RPKI security extension for the open-source BGP daemon Quagga. This extension does not comply with the current IETF spec as not only the validation of ROAs but also the validation of the BGP updates is delegated to a specific SRx server. Furthermore, this software is not intended to be used in production systems. To the best of our knowledge, rpki.net/rtr-origin [19] is the only open-source implementation of the RPKI/RTR client part in addition to the RTRlib. However, rtr-origin was explicitly developed in Python to debug the protocol and not designed for real-time operations or router integration.

The Regional Internet Registries provide looking glasses to monitor the creation of ROAs [20] and to request the validation outcome of prefix origination [21], [22]. It is worth noting that the looking glasses are based on 6 hours dumps of external BGP data and thus do not provide real-time validation but off-line testing.

5 Conclusion and Outlook

In January 2011, practical steps have been implemented towards securing the Internet backbone routing. With the deployment of the Resource Public Key Infrastructure (RPKI) by the Regional Internet Registries (e.g., ARIN, RIPE) owners of IP prefixes are able to attest autonomous systems the legitimate announcement of these prefixes. Based on this information BGP routers may perform prefix origin validation and thus prevent prefix hijacking.

Supporting the research community by appropriate tools and testbeds to increase the understanding of secure Internet backbone routing is an ongoing effort (e.g., [23]). In this paper we introduced RTRlib, a very efficient open-source implementation of the required functions for prefix origin validation. Our implementation is built as modular C library making it applicable for real routers and online monitoring systems as well as script-based evaluations. We hope that the research community will benefit from an implementation that is tailored for real-world deployment, experiments, and measurements as this may help to increase reproducibility.

Using this reference implementation in C, we presented a first load analysis of prefix origin validation at

BGP commodity hardware. The overall system scales surprisingly well even under very high BGP update rates. Including all presently announced IP prefixes into the RPKI would lead to a 5% increase of RAM compared to current vendor suggestions for memory resources.

In future work, we will elaborate our public online monitoring system based on RTRlib, which is fed by a BGP live stream and provides information about prefix origin validation outcome. We will refine our Quagga integration to complement the evaluation part. We will extend our preliminary work [17] on the distinction of RPKI-misconfiguration from prefix hijacks.

Acknowledgements

We would like to thank the SIDR community for very valuable discussions. We thank Rob Austein, Tim Bruijnzeels (RIPE NCC) and Andrew Chi (BBN) for fruitful interoperability tests with their implementations of the RPKI/RTR cache server. Olaf Maennel is gratefully acknowledged for discussions on BGP routing table modeling and adjusting the question on CPU performance.

This work is partially supported by the German Federal Ministry of Education and Research within the project Peeroskop (`http://peeroskop.realmv6.org`).

References

[1] D. M. Slane, C. Bartholomew *et al.*, "2010 Report to Congress," U.S.–China Economic and Security Review Commission, Annual Report, November 2010. [Online]. Available: http://www.uscc.gov/annual_report/2010/annual_report_full_10.pdf

[2] K. Butler, T. Farley, P. McDaniel, and J. Rexford, "A Survey of BGP Security Issues and Solutions," *Proc. of the IEEE*, vol. 98, no. 1, pp. 100–122, January 2010.

[3] M. Lepinski and S. Kent, "An Infrastructure to Support Secure Internet Routing," IETF, RFC 6480, February 2012.

[4] G. Huston, "Resource Certification," *The Internet Protocol Journal*, vol. 12, no. 1, pp. 13–26, March 2009.

[5] R. Bush and R. Austein, "The Resource Public Key Infrastructure (RPKI) to Router Protocol," IETF, RFC 6810, January 2013.

[6] P. Mohapatra, J. Scudder, D. Ward, R. Bush, and R. Austein, "BGP Prefix Origin Validation," IETF, RFC 6811, January 2013.

[7] R. Bush, R. Austein, K. Patel, H. Gredler, and M. Waehlisch, "RPKI Router Implementation Report," IETF, Internet-Draft – work in progress 01, January 2012.

[8] L.-C. Wuu, T.-J. Liu, and K.-M. Chen, "A longest prefix first search tree for IP lookup," *Computer Networks*, vol. 51, no. 12, pp. 3354–3367, August 2007.

[9] H. Yan, R. Oliveira, K. Burnett, D. Matthews, L. Zhang, and D. Massey, "BGPmon: A real-time, scalable, extensible monitoring system," in *Proc. of the 2009 Cybersecurity Applications & Technology Conference for Homeland Security (CATCH'09)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 212–223.

[10] K. Dooley and I. Brown, *Cisco IOS Cookbook*, 2nd ed. Sebastopol, USA: O'Reilly, 2006.

[11] Cisco, "BGP: Frequently Asked Questions," http://www.cisco.com/image/gif/paws/5816/bgpfaq_5816.pdf, 2012.

[12] M. Roughan, W. Willinger, O. Maennel, D. Perouli, and R. Bush, "10 Lessons from 10 Years of Measuring and Modeling the Internet's Autonomous Systems," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1810–1821, 2011.

[13] D. Massey, H. Yan, M. Strizhov *et al.*, "BGPmon. Next generation BGP Monitor," http://bgpmon.netsec.colostate.edu/, 2012.

[14] A. Feldmann, O. Maennel, Z. M. Mao, A. Berger, and B. Maggs, "Locating Internet Routing Instabilities," in *Proc. of SIGCOMM*. New York, NY, USA: ACM, 2004, pp. 205–218.

[15] J. Li, M. Guidero, Z. Wu, E. Purpus, and T. Ehrenkranz, "BGP Routing Dynamics Revisited," *ACM SIGCOMM CCR*, vol. 37, no. 2, pp. 5–16, 2012.

[16] A. Elmokashfi, A. Kvalbein, and C. Dovrolis, "BGP Churn Evolution: A Perspective from the Core," *IEEE/ACM Trans. Netw.*, vol. 20, no. 2, pp. 571–584, 2012.

[17] M. Wählisch, O. Maennel, and T. C. Schmidt, "Towards Detecting BGP Route Hijacking using the RPKI," in *Proc. of ACM SIGCOMM, Poster Session*. New York: ACM, August 2012, pp. 103–104. [Online]. Available: http://conferences.sigcomm.org/sigcomm/2012/paper/sigcomm/p103.pdf

[18] NIST, “BGP Secure Routing Extension (BGP-SRx),” http://www-x.antd.nist.gov/bgpsrx/, 2012.

[19] R. Austein, “rpki.net,” http://subvert-rpki.hactrn.net/trunk/, 2012.

[20] “RIPE NCC. RIR Trust Anchor Statistics,” https://www.ripe.net/lir-services/resource-management/certification/rir-trust-anchor-statistics, 2012.

[21] RIPE NCC, “Making Better Routing Decisions Through RPKI Validation ,” http://www.ripe.net/lir-services/resource-management/certification/making-better-routing-decisions-through-rpki-validation, 2012.

[22] L. labs, “Origin Validation Looking Glass,” http://www.labs.lacnic.net/rpkitools/looking_glass/, 2012.

[23] O. Maennel, I. Phillips, D. Perouli, R. Bush, R. Austein, and A. Jaboldinov, “Towards a Framework for Evaluating BGP Security,” in *Proc. of the 5th Workshop on Cyber Security Experimentation and Test*. Berkeley, CA, USA: USENIX Association, 2012.