Low latency RPKI validation

...

Mikhail Puzanov
RIPE NCC

RPKI validation in a nutshell

- RPKI Relying Party software (validators) download crypto-objects from a set of repositories for each Trust Anchor
- RPs validate RPKI tree(s) and produce payloads (VRPs, VAPs, etc.)
- The payloads end up in routers directly using RTR protocol or indirectly via API or RTR proxy
- Repeat the above in some form

What do we mean by latency

- Get payloads (VRPs, etc.) from repositories to the router ASAP
  - "Propagation to Relying Party represents the most time-consuming step observed in ROA processing" [1]
  - We assume that it is an improvement to reduce this time
- Do not get delayed or blocked by unresponsive or misconfigured repositories
  - Multiple papers about RPKI validators getting delayed, completely blocked or crashed [2], [3], etc.
- All the ideas in this talk are implemented in rpki-prover validator
  - https://github.com/lolepezy/rpki-prover

Handling repositories: basics

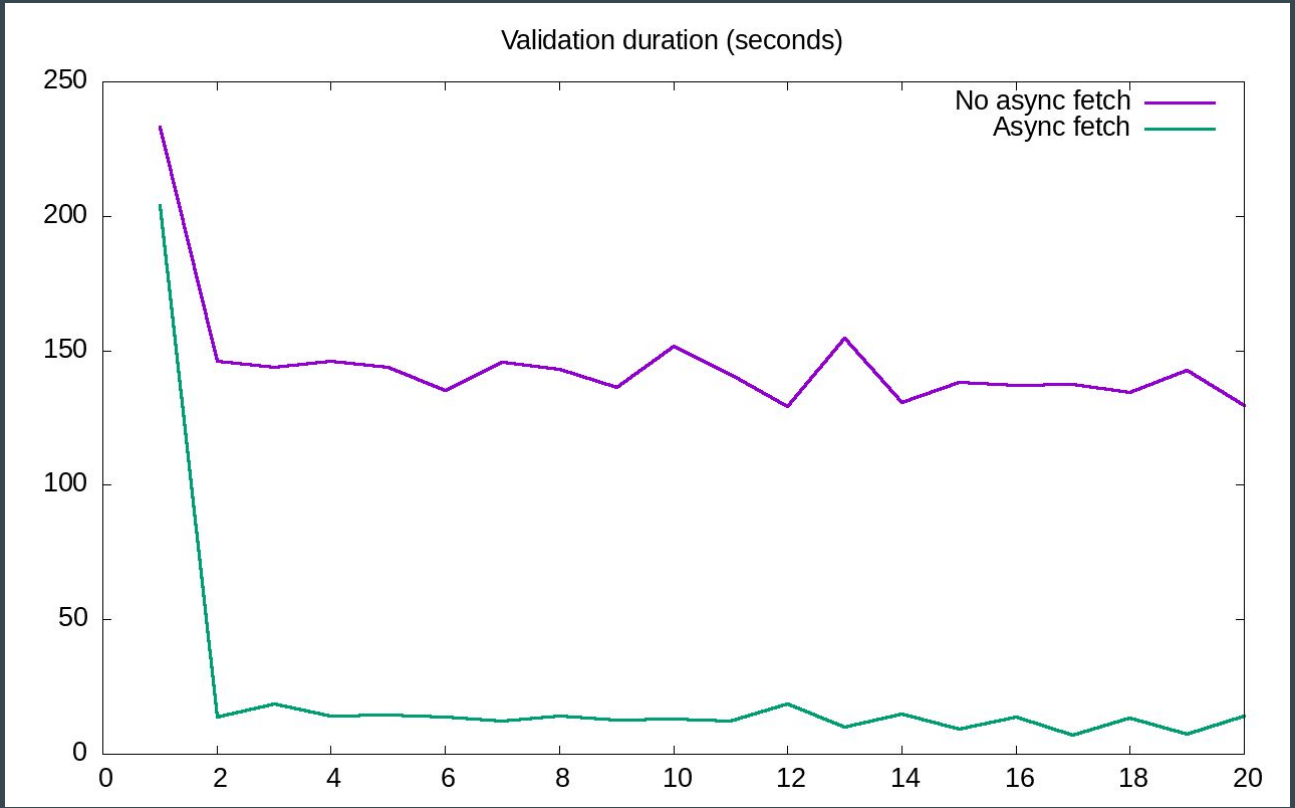
- Run every repository fetch in a separate process constrained in
  - Time (clock and CPU)
  - Memory
- Run multiple fetches in parallel
  - Run a fetch if there are less than N of them already running
  - Or it's been waiting for more than M seconds
  - Always make some progress

Handling bad repositories

- Some repositories time-out: validation is blocked on fetching
- Fetch can be synchronous or asynchronous to validation
- A repository is marked "synchronous" when
  - Seen for the first time
  - After a successful fetch within N seconds
  - No RRDP —> rsync fall-back happened

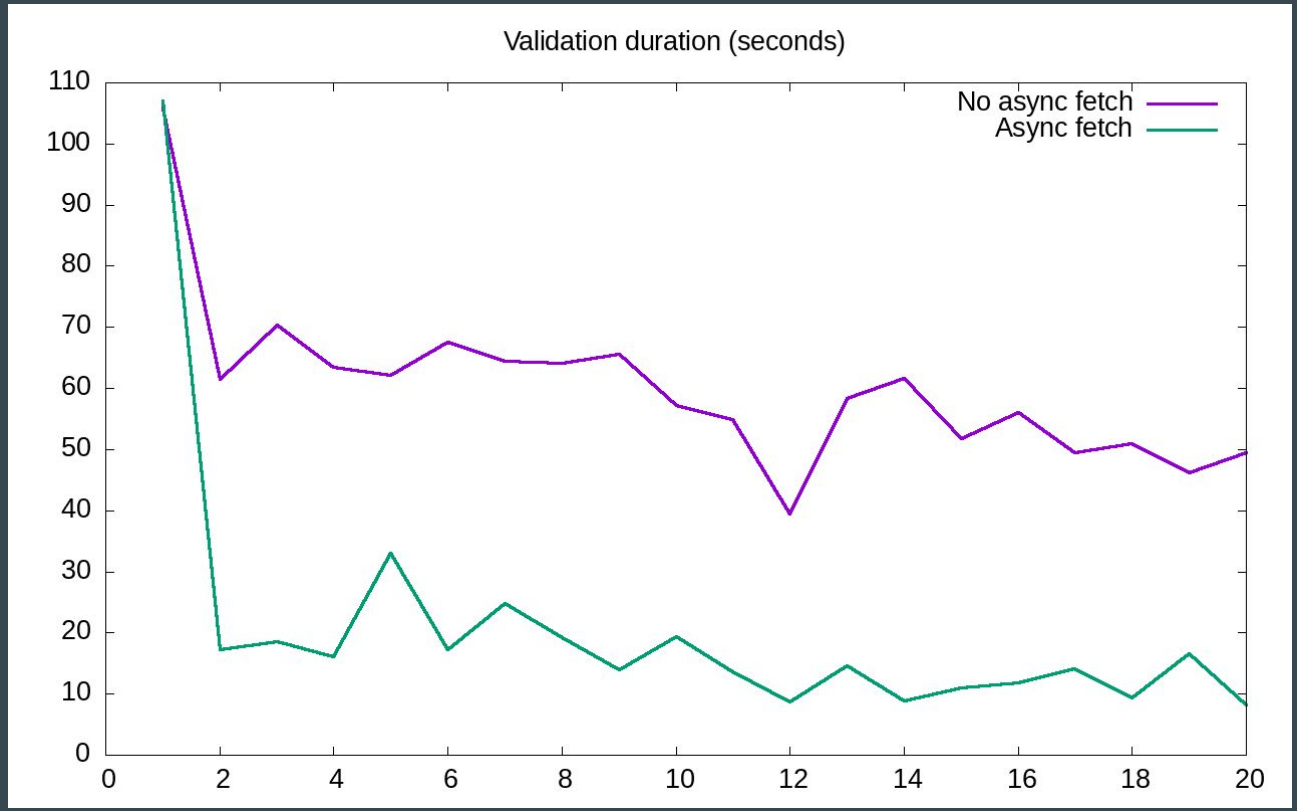
Handling bad repositories

RRDP timeout 120s
5 fetchers max
7 TAs (5 RIRs, 2 TA0)



Handling bad repositories

RRDP timeout 30s
5 fetchers max
7 TAs (5 RIRs, 2 TA0)



Handling bad repositories

- Set timeouts per synchronous repository based on how much it took to download it previous time(s)
  - The faster a repository is the less we are going to wait for it next time
  - No big delays from a suddenly broken synchronous repository

Reducing delays by RP

- Revalidate more often
  - It is expensive: it takes clock and CPU time (10s of seconds to minutes)
  - More frequent RRDP requests
- Reduce CPU usage per validation
- Avoid unnecessary RRDP requests

Reduce CPU usage per tree validation

- Incremental tree validation
  - Validate fully only newly downloaded objects
  - Be smart about which manifest children to revalidate
  - For already validated objects only re-check validity time
- Complexity: $O(V_{full} \times N_{objects})$ becomes $O(V_{full} \times N_{updates} + V_{short} \times N_{objects})$

Reduce CPU usage per tree validation

- Pro: about 9-10 times less CPU usage for tree validation
- Cons:
  - Complexity, git diff is (+3337, -1910)
  - Does not currently support validation reconsidered (RFC 8360)

Less RRDP requests

- Adaptive refresh intervals per repository
  - If more than 1 delta, reduce fetch interval
  - If there are no updates, increase fetch interval
  - Don’t make it less than 1 minute or more than 10 minutes
- Pro: about 40% less RRDP fetches
- Cons: complexity, higher latency for infrequently updated repositories
- It is a tradeoff between latency and redundant requests

Less RRDP requests

- For a typical run
  - 3-5 repositories converge to 1 minute interval (depending on time of the day)
  - ~ 65 repositories converge to 10 minutes interval
  - ~ 7 settle somewhere in between

Conclusion

- Pretty simple rules significantly improve resiliency to delay and blocking, still some low-hanging fruits there
- It is possible to implement a cheaper and a more future-proof tree validation algorithm but it introduces complexity
- It seems to make sense to adjust update intervals for RRDP repositories dynamically (also **ETags** are not supported universally, it's a shame)
- rpki-prover releases 0.9.x includes all these features, try it

References

1. Romain Fontugne, Amreesh Phokeer, Cristel Pelsser, Kevin Vermeulen & Randy Bush. RPKI Time-of-Flight: Tracking Delays in the Management, Control, and Data Planes
2. Tomas Hlavacek, Philipp Jeitner, Donika Mirdita, Haya Shulman and Michael Waidner, Stalloris: RPKI Downgrade Attack
3. Koen van Hove, Jeroen van der Ham and Roland van Rijswijk-Deij, rpkiller: Threat Analysis from an RPKI Relying Party Perspective

Questions?