Requirements for a next-gen RPKI transport

Job Snijders
Fastly / OpenBSD
job@fastly.com

Agenda

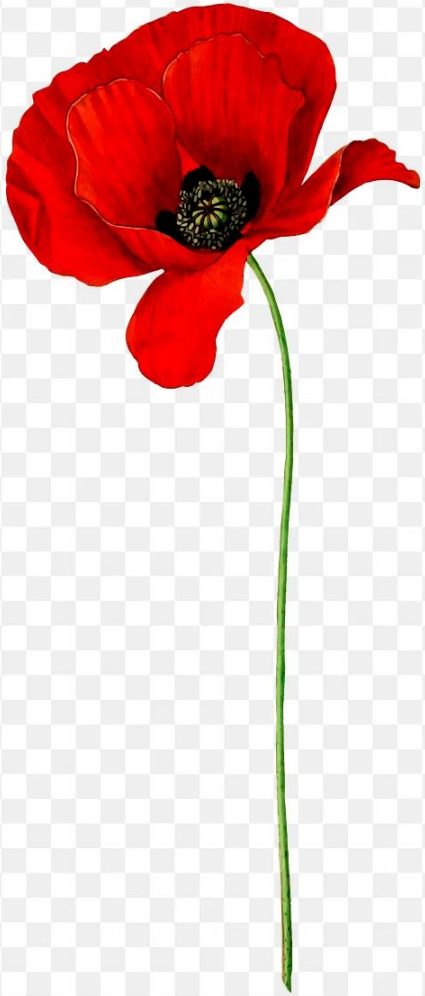
- Introduction
- Proposed path forward
- Questions

Introduction

RPKI data has to be synchronized from the CAs to the RPs

We currently have rsync and RRDP

It seems nobody is truly happy with either approach



What some people like about RRDP

- In *theory* a minimized set of data is transferred towards RP
  - Unchanged UNF is tiny download
  - Contrast with rsync: client & server exchange full file listing
- It runs over HTTPS
  - More choice for libraries
  - CDNs can be used (most CDNs offer HTTP caching layer and not rsync services)
- Server can pre-calculate things, HTTP server serves static content
- Minimal back and forth between client & server
- It is an IETF standardized protocol (no reverse engineering required)

Honestly, it is kinda hard to come up with reasons we like RRDP beyond these 4

But RRDP is not without issues, is it …

Time for honest review

There, I fixed it

RRDP was designed to support scaling in RPKI's asymmetric deployment. It is consistent (in terms of data structures) with the publication protocol [RFC8181] and treats publication events of one or more repository objects as discrete events that can be communicated to Relying Parties. This approach helps to minimize the amount of data that traverses the network and thus helps minimize the amount of time until repository convergence occurs. RRDP also provides a standards-based way to obtain consistent ~~, point-in-time views of a single repository, eliminating a number of consistency-related~~ issues. Finally, this approach allows these discrete events to be communicated as immutable files. This enables Repository Servers to pre-calculate these files only once for all clients, thus limiting the CPU and memory investments required, and enables the use of a caching infrastructure to reduce the load on a Repository Server when a large number of Relying Parties are querying it.

Problems with RRDP

- cARepository state transitions might get smeared across multiple Deltas
  - Properly implementing “Failed Fetch” procedure is fairly complicated
- Requires shared WebPKI Trust Anchors between server & client
- Base64-encoded DER binary blobs stuffed in a XML-document …
- Client don’t have a lightweight mechanism to detect desynchronization
  - Have to download full Snapshot to ascertain full synchronicity
- Easy to provoke thundering herds
  - An issue in an Delta causes a stampede on the Snapshot file
  - RRDP performs poorly in some low-bandwidth situations
  - No ‘resumption’ of download

Things to like about rsync

- No reliance on WebPKI
- Performs *really* well in low-bandwidth situations
- Can ‘resume’ synchronization attempts between runs
  - Eventually you slurp in the full remote side
- Full synchronization is assured every run
  - Whatever the state of the local system is, the remote side helps supplement
  - Server “helps” client by taking part in calculating what needs sending
- Direct access to individual objects
  - You can download 1 specific object, instead of “download everything to get 1 object”

Things not to like about rsync

- No transport security
- cARepository states might be smeared across different synchronization jobs
- A bad client can make the server to waste CPU and Memory
- Not an IETF standardized protocol
  - Fair bit of reverse engineering was required to make things work in openrsync
- Not a service commonly sold by “CDN”-style providers

Things to like about using **both** RRDP & Rsync

Both transports have failure modes, but the failure modes don’t overlap much

One as backup to the other is brilliant from a resilience perspective

Multiple transports / endpoints turned out to be **excellent diversification**:

- IPv4 versus IPv6: routing diversity
- TLS Protected versus Plain-text: transport security became opportunistic feature
- Endpoints hosted in different ASNs: routing diversity
- Different failure modes: XML encoding errors don’t exist in rsync

Things not to like about either RRDP & Rsync

- Neither protocol is as efficient as theoretical limits would suggest
  - RRDP transfers objects “in whole”
  - Rsync transfers the difference between objects, but is not “RPKI application aware”
- No RPKI-aware compression
- No RPKI-aware deduplication

Where does this leave us for the next-gen transport?

Instead of barging on and someone specifying “*tada.wav* here is RRDP v2”...

Let’s *first* work to specify our requirements for the next-gen thing

Requirements documents & the IETF Process

It can feel like “the long way around”... but …

- My intuition is that it’ll be the shorter path in the long run
- Consensus on requirements allows multiple solution proposals to be put forward
- Good way to make sure every stakeholder is heard
  - make inventory of what matters to various folks in the ecosystem
- We now have **A TON** of experience with both RRDP & Rsync

Pre-existing example of “*requirements first, then solution”*:

RFC 6198 “Requirements for the Graceful Shutdown of BGP Sessions”

RFC 8326 “Graceful BGP Session Shutdown”

Potential requirements that come to mind

- Must offer direct access to individual objects?
- Minimize bandwidth consumption?
  - Must have excellent compression & deduplication properties
- Transport agnostic?
  - Should it be able to run over TCP, TLS, SSH, HTTP, HTTPS?
  - Examples: git works over SSH & HTTPS; RTR works over TCP, SSH, TCP-MD5, TCP-AO
- Use DER/ASN.1 as “container” instead of say… Base64/XML
- Synchronization session resume capabilities?
  - HTTP partial downloads
  - Rsync ‘recycles’ the local state
- Efficiency on server side
  - In RRDP you generate a full snapshot for even the tiniest delta, it’s a lot of work
- Auditability / Security: use some form of object security
  - Example: NRTM v4 uses signatures over “snapshots” / “deltas”

Next steps

Myself, Tim Bruijnzeels, and Ben Maddison are working on

draft-spaghetti-sidrops-rpki-transport-requirements-00

Hopefully can publish -00 before the end of the year (2024)

Multiple rounds to get input from:

- CAs operator / implementers
- Publication Point operators / RFC 8181 implementers
- RP implementers
- And really anyone else, this is going to be team effort