RPKI Over BitTorrent

Rob Austein <sra@hactrn.net>
Randy Bush <randy@psg.com>
Michael Elkins <Michael.Elkins@sparta.com>
Leif Johansson <leifj@sunet.se>
. . . and a lot of help from our friends

IETF 83
Paris
March 2012

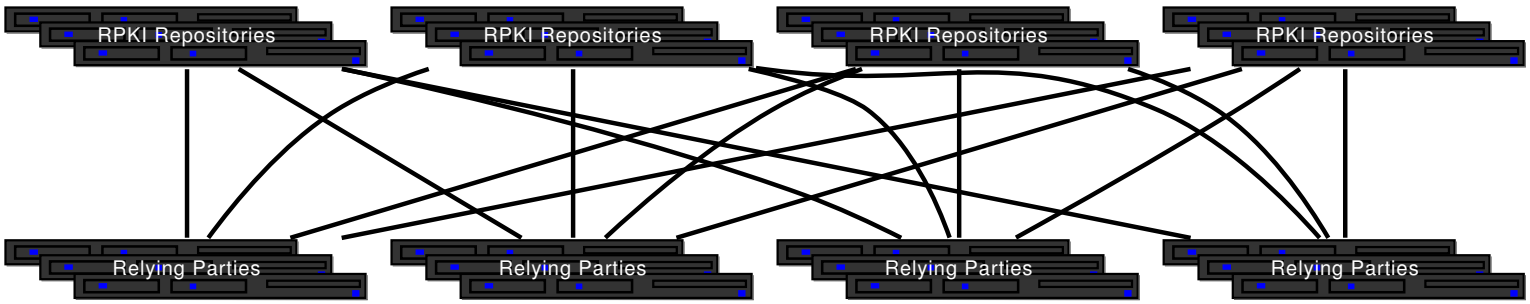
“RPKI Over What?!? You’re Kidding, Right?”

- No, we’re serious about investigating this.
- But this is a report on an experiment in progress, not a proposal to change the SIDR protocol suite.

The Problem We're Trying To Solve

- The RPKI uses rsync as a form of pull-based flooding.
- How efficient this is depends heavily on how the publication repositories are organized.
- In an efficiently organized repository, filesystem hierarchy follows X.509 certificate hierarchy, so that one can pick up significant subtrees with a single rsync connection.
- To date, the RIRs have chosen to deploy flat hierarchies where there is no relationship at all between filesystem hierarchy within the repository and certificate hierarchy.

Simplest Retrieval Topology: Direct Rsync



- Protocols: Rsync.
- Every relying party talks to every repository.

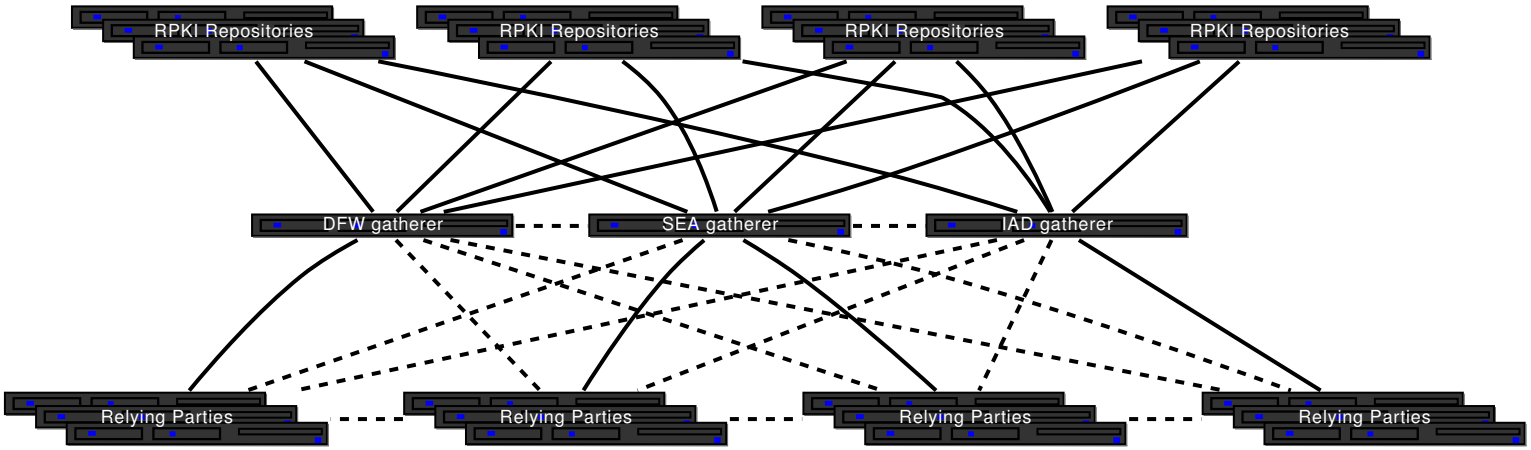
Direct Rsync With Flat Repositories Is Nasty

- The combination of flat repositories and direct rsync means that each relying party is making thousands of connections every hour, just to stay in synch.
- This is not pretty for the relying party.
- Doesn't look so nice for the operators of these big flat repositories either.
- And it's avoidable. . . but the repository operators have to choose to avoid it.
- There are load balancing tricks repository operators can use to spread load here, just as with HTTP and DNS. They may help. We may have to use them in any case.

What Can Relying Parties Do About This?

- Relying parties can (and do) maintain local caches of previously retrieved objects to help ride out failures to connect to repositories.
- Relying parties can reduce load with more complex retrieval schemes.
- We've been pushing a hierarchical retrieval model, in which a relatively small pool of gatherers feed a much larger number of relying parties.

Hierarchical Rsync Topology



- Protocols: Rsync.
- Every gatherer talks to every repository.
- Every relying party talks to at least one gatherer, probably with at least one more gatherer configured as a hot spare, perhaps with other links.
- Mesh can be arbitrarily rich, problem is configuring it.

What About Other Protocols?

- Choosing a different pull-based protocol (HTTP, FTP) wouldn't solve the flat repository problem.
- Conventional pull-based protocols like rsync, HTTP, and FTP aren't really designed for flooding, although they can be made to work.
- There may, however, be an opportunity to take advantage of knowing that our goal is flooding.
- Perhaps we should investigate something that was designed as a flooding protocol?

Flooding Protocols

- NNTP could work, but people tend to run screaming from the room when it's mentioned, it requires a lot of care and feeding, and delay between publication and receipt of data by relying party might be a showstopper.
- Could invent yet another protocol, would rather not.
- So how about BitTorrent?

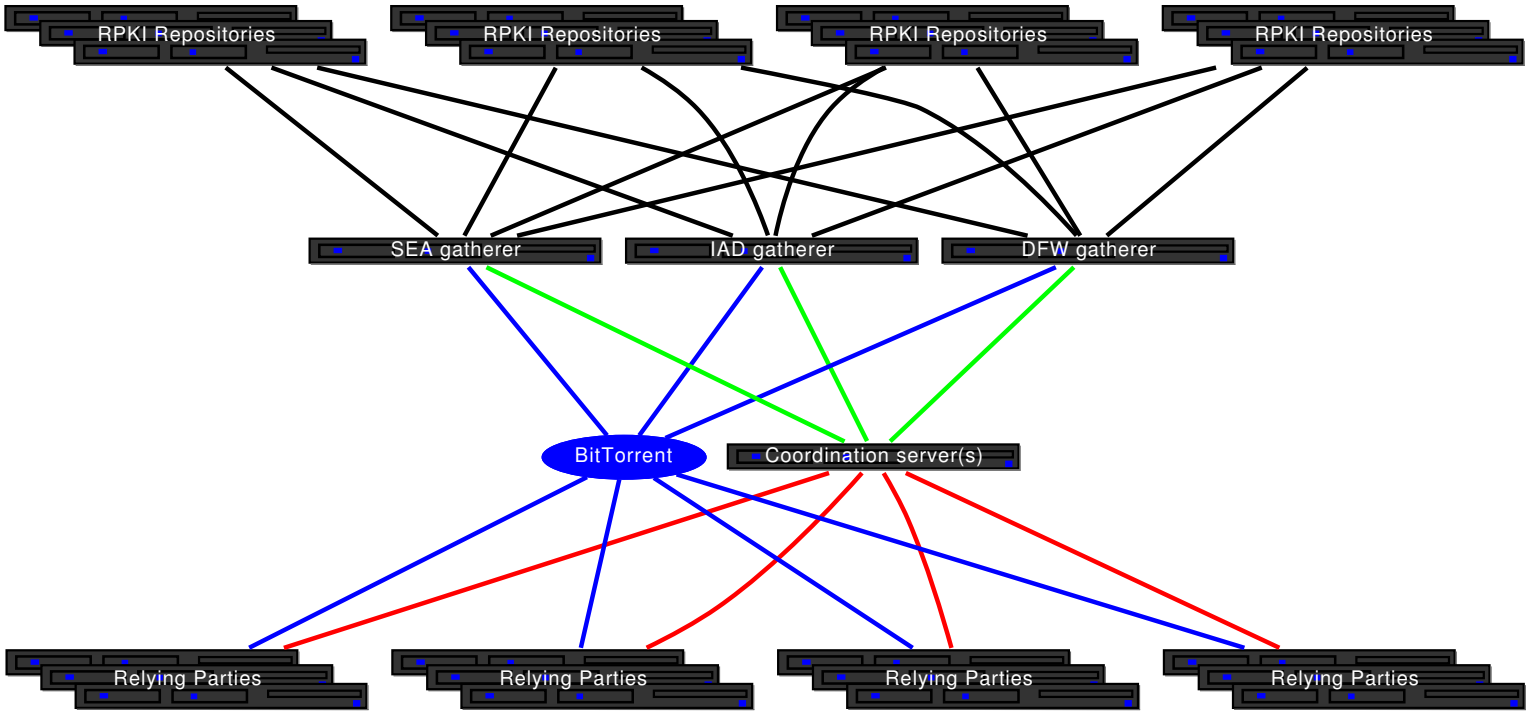
Experiment Goals

1. Use BitTorrent to build a self-organizing flooding network to distribute unauthenticated RPKI data.
2. See if the silly thing works at all.
3. Study and measure behavior to find out whether this is useful.

Notes:

- Relying parties still perform their own RPKI validation, we're just separating that out from RPKI data collection.
- We're really only at stage #2 now, and have only just started stage #3.

BitTorrent Retrieval Topology



- Protocols: Rsync, BitTorrent, SFTP, HTTPS.
- Every gatherer talks to every repository.
- Every gatherer publishes a torrent.
- Every relying party gets one or more torrents from. . . somewhere.

Translating URIs To Filenames

- Somebody asked in Taipei: “How do we know the original publication URI of an object once we’re not retrieving it directly?”
- Tools like GNU wget solved this ages ago: map URI `rsync://example.org/foo/bar.cer` to local filename `example.org/foo/bar.cer`.
- rcynic already does this.
- We already do this for hierarchical rsync.

Operational Overview

- Use BitTorrent to transfer unauthenticated RPKI data as a collection individual files in a torrent. Yes, BitTorrent can do this, with rsync-like behavior (only fetches files that have changed).
- Package `.torrent` file describing the torrent and a "manifest" (not to be confused with a RPKI manifest objects), listing the name of every file in the collection and a SHA-256 hash of each file's content together as a `.zip` file, transfer that `.zip` file via HTTPS.

Creating A Torrent

1. Gatherer runs rcynic with rsync enabled, does normal RPKI tree walk. One output is the `unauthenticated/` directory tree, containing exactly the files we want to redistribute.
2. Gatherer puts copy of `unauthenticated/` tree where BitTorrent engine can find it.
3. Gatherer constructs a `.torrent` file describing the `unauthenticated/` tree and listing the coordination server's BitTorrent tracker.
4. Gatherer constructs manifest listing the name of every file in `unauthenticated/` and a SHA-256 hash of each file's content.
5. Gatherer packages `.torrent` file and manifest together as a `.zip` file, and uploads that to coordination server via SFTP.

Fetching A Torrent

1. Relying party polls coordination server for updated `.zip` file via HTTPS, using HTTP `Last-Modified` header to determine whether the `.zip` file has changed.
2. Upon receiving new `.zip` file, relying party extracts the `.torrent` file and hands that to BitTorrent engine.
3. When BitTorrent engine signals that it has finished receiving the new torrent, relying party checks received files against manifest from the `.zip` file to verify that relying party received what gatherer sent.
4. If everything looks good, relying party runs rcynic with rsync fetching disabled, using the data retrieved via BitTorrent as input to be authenticated.

Other Details

- New `.zip` files are installed with an atomic rename operation.
- Current prototype has only one coordination server, but design should scale up to maybe a dozen, perhaps just making every gatherer also be a coordination server. Above that, we're getting into territory where we'd want to hire experts.
- Each gatherer also mirrors the other gatherers' torrents, so that there will be multiple sources for each torrent. Gatherers are configured for unlimited seeding.
- We're not currently using web seeding, because we haven't needed it yet, but in theory it's simple: gatherer would just make files available via web server too, and include URL when constructing `.torrent` file.

Like “Broadcatching,” But Less Filling

- Overall technique here is similar to “broadcatching,” but without the RSS intermediary.
- Main feature RSS would bring would be ability to discover new feeds automatically. We’re not convinced that’d be a feature, do you really want your relying party software fetching from gatherers it met in an Internet chat room?
- RSS would also complicate installation of new `.zip` files, which is currently dead simple.
- So we left RSS out, because it didn’t add anything we needed. If it turns out we were wrong. . . we’ll add it back.

Software Required: Relying Parties

- RPKI validator (rcynic)
- BitTorrent engine (transmission-daemon)
- Control scripts, HTTPS client (Python 2.7)

Software Required: Data Gatherers

- RPKI gatherer and validator (rcynic, rsync)
- BitTorrent engine (transmission-daemon)
- SSH/SFTP upload client (paramiko)
- Torrent construction tool (mktorrent)
- Control scripts (Python 2.7)

Software Required: Coordination Servers

- HTTPS download server (Apache)
- SSH/SFTP upload server (OpenSSH)
- BitTorrent “tracker” server (opentracker)

Tradeoffs: Caveats

- Most of the following applies equally to hierarchical rsync and to BitTorrent.
- Most of the differences between hierarchical rsync and BitTorrent are a matter of degree rather than of kind.

Tradeoffs: Pros

- Fast, cheap way for relying party to receive data.
- Reduce load on repositories, perhaps dramatically.
- Makes view from multiple gatherers available to relying party, which might be useful.
  - This is still research, so I'm allowed to say that.
- BitTorrent probably scales better than hierarchical rsync—but that's theory, not measurement.

Tradeoffs: Cons

- Relying party is at mercy of gatherers, unless relying party has strategy for falling back to direct fetch.
- Falling back to direct fetch could have its own issues: Provider has bad day, flooding network decays, threshold is crossed. Wham! A bunch of relying parties all fall back to direct fetch at once—which probably makes things worse for provider without fixing anything for the relying parties. Ouch.
- So falling back to direct rsync must be done very carefully, if at all, and it's not immediately obvious how a relying party would decide in which cases falling back is the right thing to do.

Tradeoffs: Cons

- Longer delay between original publication and receipt of data by relying party. Nowhere near as bad as NNTP, but longer than direct rsync.
- More moving parts means more things that can break.
- Some people have issues with BitTorrent because of other things for which it's been used in the past; the person who controls your firewall may be one of those people.

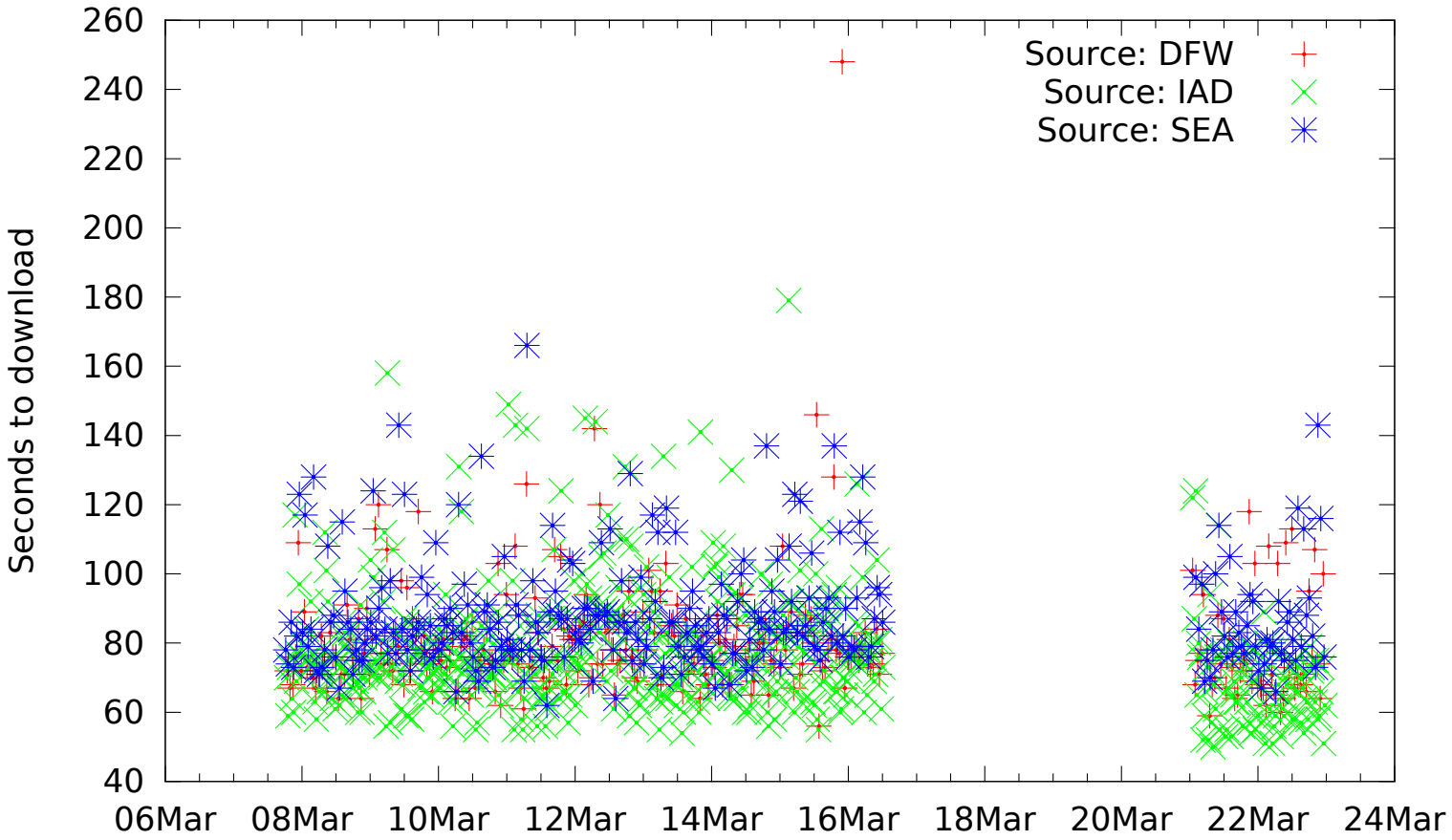
Things We're Measuring, Or Should Be

1. How long does it take relying parties to download?
2. How long does it take relying parties to validate?
3. How fresh are the data relying parties see?

Notes:

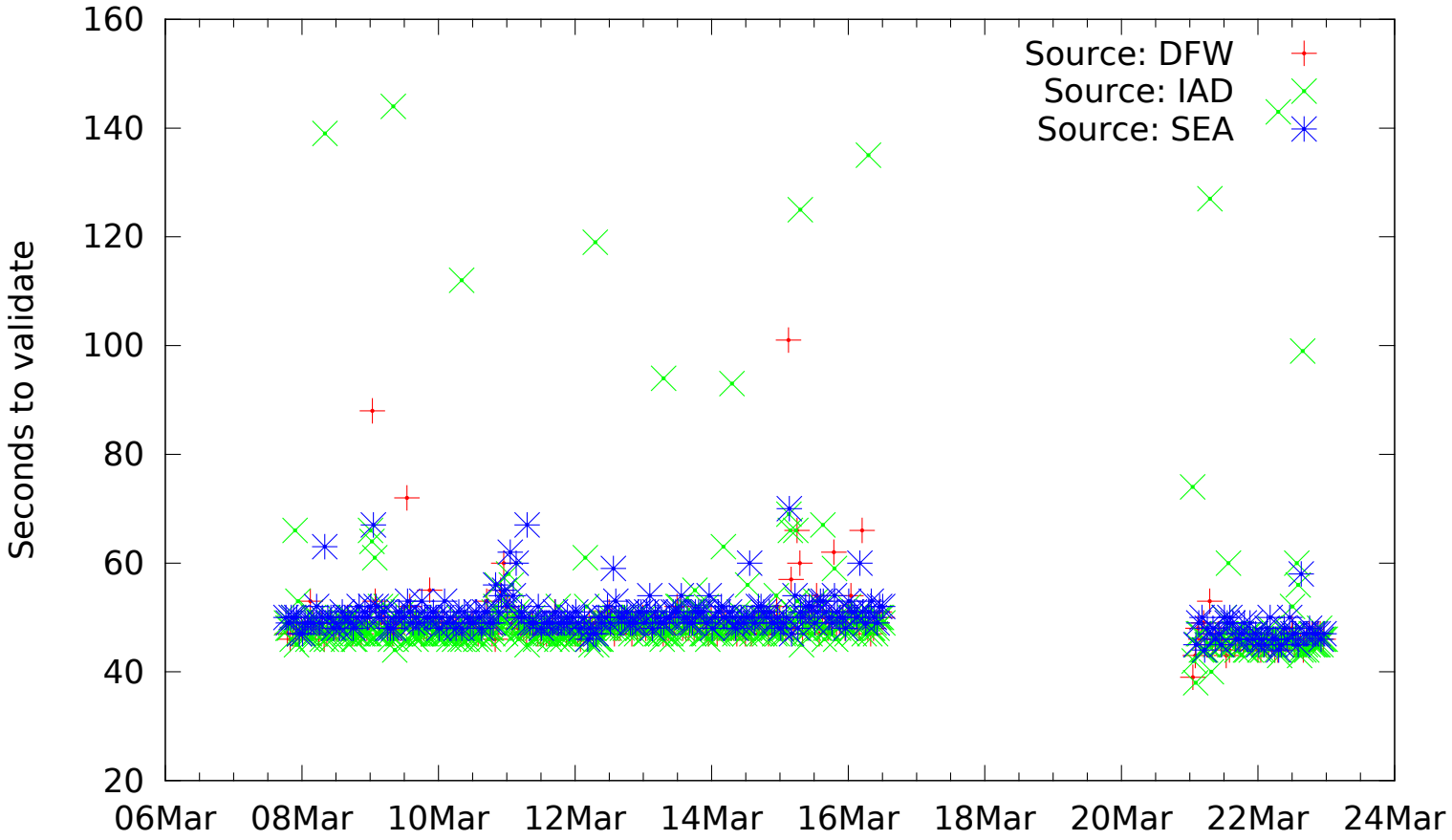
- We're currently tracking #1 and #2.
- We don't have a good handle on #3 yet.
- We don't yet have all the measurements to which we'd need for meaningful comparisons.
- This is proof-of-concept, we don't have a serious testbed running yet.

Time To Download



- Destination is home desk machine, so we suspect much of the noise is just consumer-grade network.
- Outage 17th-21st (fat fingers) shows that recovery after days offline is not significantly different.

Time To Validate



- Spikes are almost certainly environmental: most of them are at 03:00 local time, when an unrelated disk-intensive cron job is running.

Future Work

- Measure data freshness, compare with direct fetch and hierarchical rsync.
- Scale up a bit and invite friends in to play with us.
- Investigate properties of BitTorrent as an alternate mechanism for primary publication by authoritative sources.

Questions?