# RPKI rsync Download Delay Modeling

**Steve Kent and Kotikalapudi Sriram**

**Email: kent@bbn.com ; ksriram@nist.gov**

March 11, 2013

IETF-86 SIDR WG Meeting, Orlando, FL

The authors wish to thank David Mandelberg, Andrew Chi, and Rob Austein for sharing RPKI rsync measurement data. Thanks are also due to Chris Morrow, Doug Montgomery, Sandy Murphy, and Randy Bush for comments and suggestions.

## Executive Summary

- **New measured values of rsync speed (ms/object) for the hierarchic-structured rpki.ripe.net are much lower:**
  - 12 ms/object (BBN) and 17 ms/object (Austein) (see [1][2][3])
  - These basic per unit measurements are 52X and 35X lower, respectively, than what was measured in the past (avg. 628 ms/object) (see [4] for the previous measurement data; [5] reported an analysis using the data in [4]).
- **In the model presented here, the rsync download delays have been estimated for realistic ranges of # repositories, #RPKI objects, and ms/object measure.**
- We have also included parallel fetching by each relying party (RP) from multiple repositories.
  - Parallel fetching lowers the total (system wide) delay to perform rsync. This is a realistic assumption, and has been implemented and tested (by BBN).
- The effects of session setup overheads have also been included explicitly in the model.
- **Based on the results of this modeling, we project global rsync delays, as seen by each RP doing an incremental fetch, to be in the range of single digit minutes to tens of minutes.**
  - Even lower values are possible with additional optimizations to current prototype and production RPKI systems.

## How Many Publication Points

| | | Publication Points |
|---|---:|---:|
| # RIRs | 5 | 5 |
| # ASes | 43,000 | |
| % of ASes with a single allocation source | 90% | |
| % of ASes with multiple (2) allocation sources | 10% | |
| Avg. # of sources from which an AS has allocations | 1.1 | |
| % Stub AS | 84% | |
| % non-stub AS | 16% | |
| # Stub Ases | 36,120 | 39,732 |
| # Non-stub ASes | 6,880 | 7568 |
| Total # Publication Points | | 47305 |

Citation for the (84%,16%) stub, non-stub AS data (measurements by Randy Bush):
See slide 15 in [7].

# Estimation of # RPKI Objects

| Object type | # Objects per Pub Point<br>Stub AS | <br>non-Stub AS | <br>RIR | Grand Total |
|---|---|---|---|---|
| CA cert | 1 | 7 | 1515 | |
| CRL | 1 | 1 | 1 | |
| Manifest | 1 | 1 | 1 | |
| Ghostbuster Record | 1 | 1 | 1 | |
| ROA | 1 | 1 | 1 | |
| Router certs | 2 | 10 | 0 | |
| Totals (per AS or RIR) | 7 | 21 | 1,519 | |
| Totals (multiply line above by corresponding # Pub Points): | | | | |
| Without router certs | 198,660 | 83,248 | 7,593 | 289,501 |
| With router certs | 278,124 | 158,928 | 7,593 | 444,645 |

* Note 1

** Note 2

* Note 1: Each non-stub AS pub point carries 6 CA certs (on average) of stub ASes, and each RIR carries 1514 CA certs (on average) of non-stub ASes, see slide 3.

** Note 2: Router certs: One pair (current + next) per stub AS; 5 pairs per non-stub AS (non-stubs have 5 zones of trust on average in their AS).

Path validation (i.e., with router certs)

Origin validation only (i.e., no router certs)

## Frequency of Changes in RPKI Objects

| | Change frequency | | |
|---|---|---|---|
| Object type | Stub AS | non-Stub AS | RIR |
| CA cert | annually | annually | annually |
| CRL | daily | daily | daily |
| Manifest | daily | daily | daily |
| Ghostbuster Record | annually | annually | annually |
| ROA | annually | annually | annually |
| Router certs | daily or annually | daily or annually | n/a |

If periodic key rollover is used for replay protection then router certs change every 24 hours or so.

# Rate of Changes in RPKI Objects when Router Certs Rollover <u>Annually</u>

These numbers do not include newly added pub points, just changes to existing pub points.

365 days per year

| Object type | Changes per day at each Pub Point | | | Grand Total Changes (per Day) |
|---|---|---|---|---|
| | Stub AS | non-Stub AS | RIR | |
| CA cert | 0.0027 | 0.0192 | 4.1496 | |
| CRL | 1.0000 | 1.0000 | 1.0000 | |
| Manifest | 1.0000 | 1.0000 | 1.0000 | |
| Ghostbuster Record | 0.0027 | 0.0027 | 0.0027 | |
| ROA | 0.0027 | 0.0027 | 0.0027 | |
| Router certs | 0.0055 | 0.0274 | 0.0000 | |
| Totals (per AS or RIR) | 2.0137 | 2.0521 | 6.1551 | |
| Totals (multiply line above by corresponding # Pub Points): | | | | |
| Without router certs | 79,791 | 15,323 | 31 | 95,144 |
| With router certs | 80,008 | 15,530 | 31 | 95,569 |

Path validation (router certs rollover <u>annually</u>)

Origin validation only

# Rate of Changes in RPKI Objects when Router Certs Rollover Daily

| Object type | Changes per day at each Pub Point | | | Grand Total Changes (per Day) |
|---|---|---|---|---|
| | Stub AS | non-Stub AS | RIR | |
| CA cert | 0.0027 | 0.0192 | 4.1496 | |
| CRL | 1.0000 | 1.0000 | 1.0000 | |
| Manifest | 1.0000 | 1.0000 | 1.0000 | |
| Ghostbuster Record | 0.0027 | 0.0027 | 0.0027 | |
| ROA | 0.0027 | 0.0027 | 0.0027 | |
| Router certs | 2.0000 | 10.0000 | 0.0000 | |
| Totals (per AS or RIR) | 4.0082 | 12.0247 | 6.1551 | |
| Totals (multiply line above by corresponding # Pub Points): | 159,255 | 91,003 | 31 | 250,288 |

These numbers do not include newly added pub points, just changes to existing pub points.

365 days per year

Path validation (router certs rollover daily)

# # RPKI Objects Downloaded in Each of Five Scenarios

| **Scenarios** | #Objects downloaded per instance |
|---|---:|
| All objects rsync download (w/o Rtr certs) | 289,501 |
| All objects rsync download (with Rtr certs) | 444,645 |
| Download changes only at 24-hr polling intervals (w/o router certs) | 95,144 |
| Download changes only at 24-hr polling intervals (with router certs - rollover annually) | 95,569 |
| Download changes only at 24-hr polling intervals (with router certs - rollover daily) | 250,288 |

# Rsync Delay Measurements

## Measurements for rpki.ripe.net with hierarchic repository structure

| | Measurement (RIPE Heirarchical) | BBN's | Rob Austein's | |
|---|---|---|---|---|
| R | # RPKI Objects | 4,898 | 4,690 | |
| X | Total time to sync (sec) | 59.04 | 80.78 | sec |
| S = X*1000/R | sync time per object (ms/obj) | 12.05 | 17.22 | ms |
| C | rsync session setup/teardown overhead (sec per session) | 1.22 | 0.5 | sec |
| | # rsync sessions per visit to a TA repository | 3 | 3 | |
| | # rsync sessions per visit to a non-TA repository | 1 | 1 | |

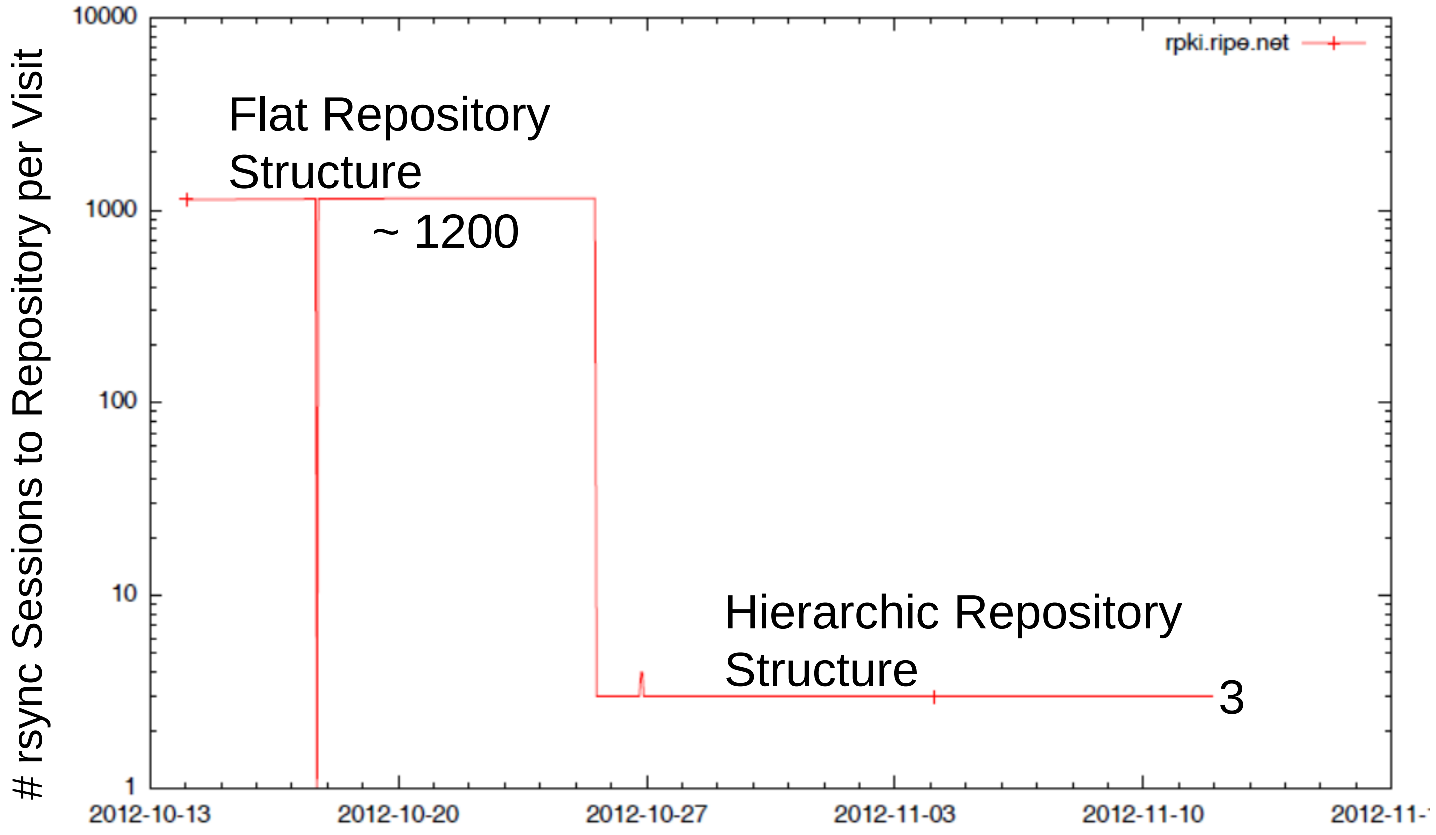
- X includes all components of delay: (1) rsync session setup/teardown overheads (negligible because only 3 sessions are involved), (2) rsync processing plus network delays (propagation, TCP flow control, etc.).
- The 17.22 ms ms/object measurement is 36X lower than the average 628 ms/object number used in [5], where it was derived from measurement data with flat repository structure from [4].

Citations:

1. Austein's measurements: see slide 24 in [1]; also see slides 6, 10 in [2].; also see slides 10, 11 here.
2. BBN's measurements: Private communication [3].

# # rsync Sessions to Repository

## rpki.ripe.net

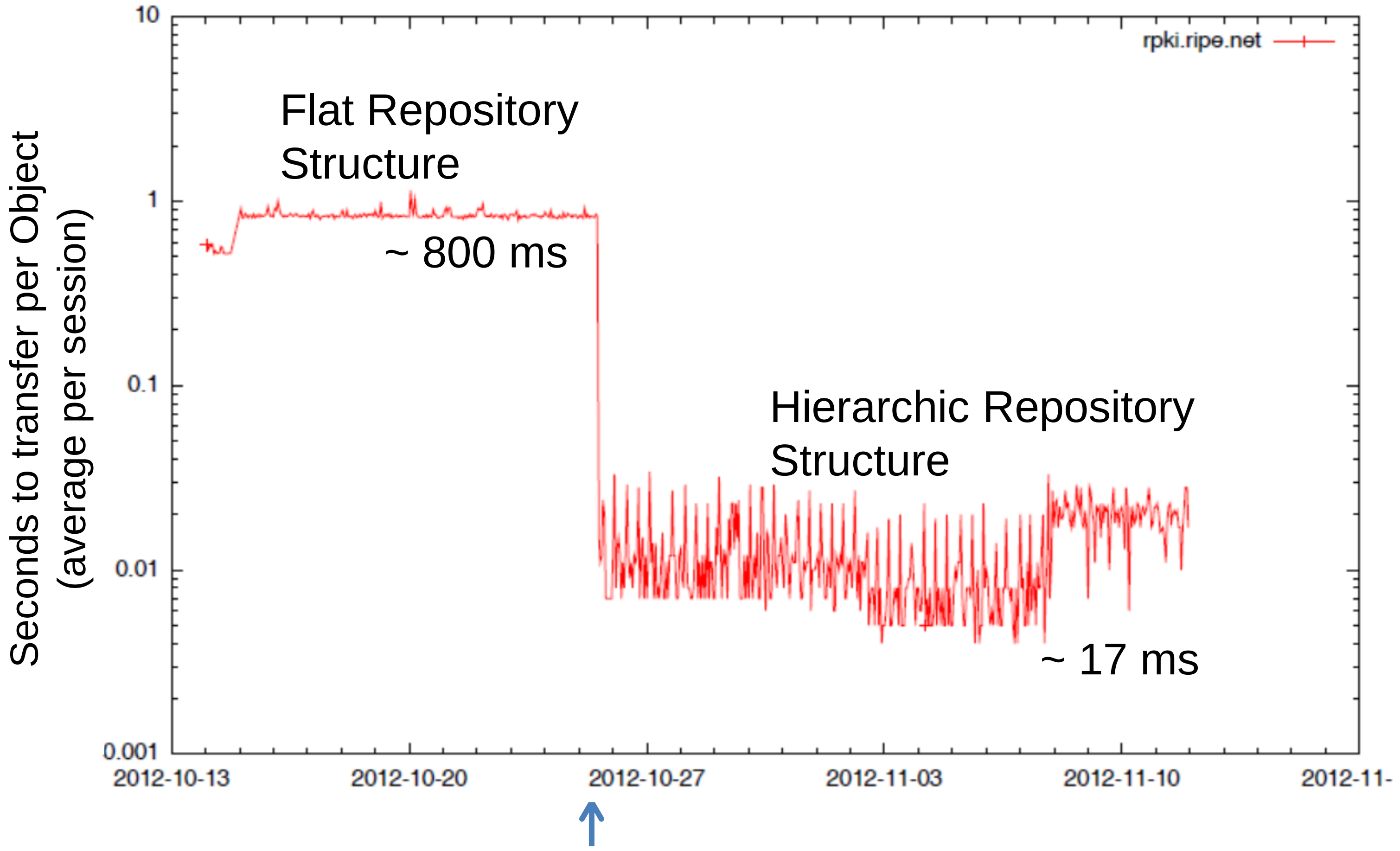


RIPE transitions from Flat to Hierarchic Repository Structure

Citation: Austein's measurements – see slides 6 in [2].

# Seconds/Object

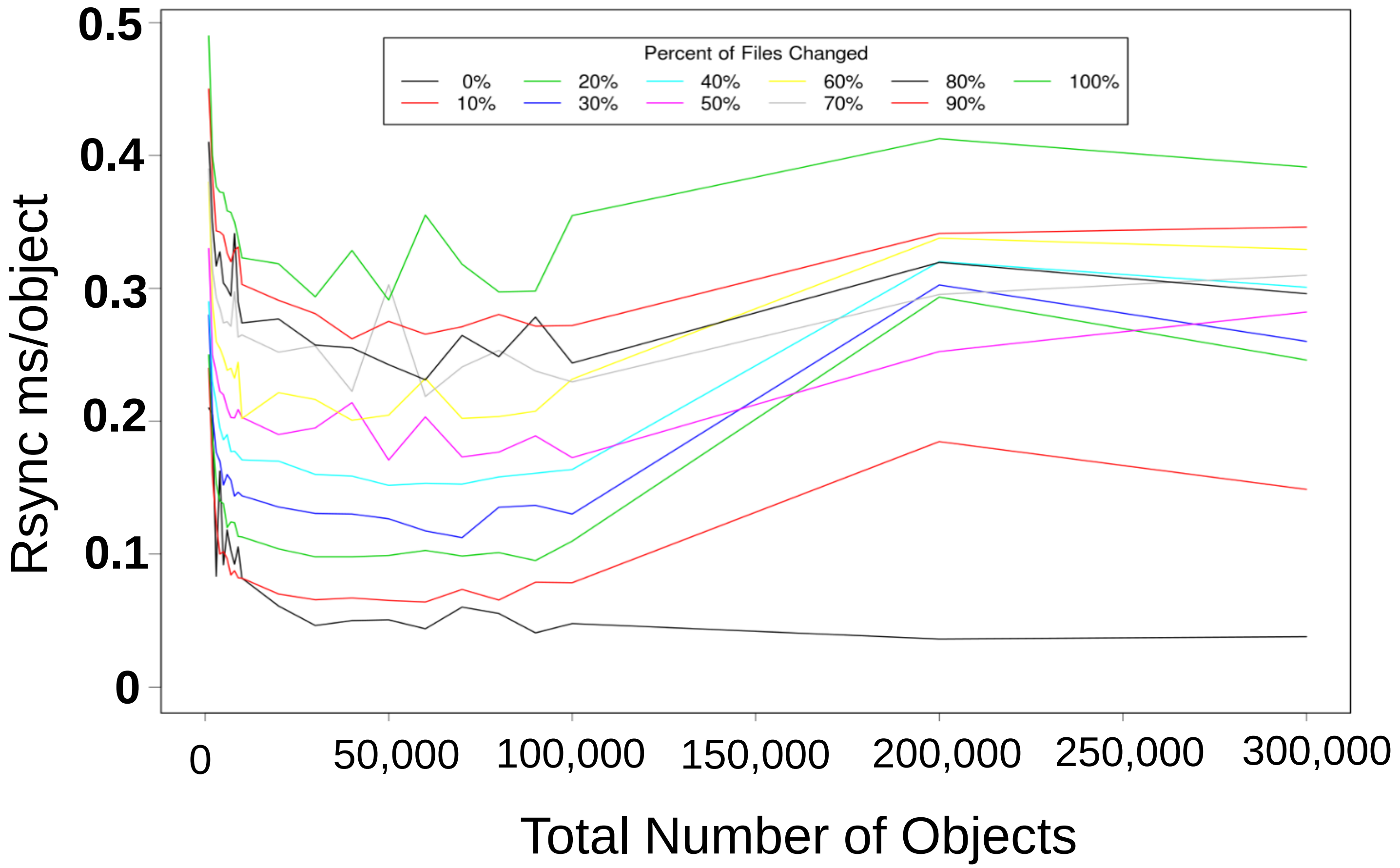
## rpki.ripe.net



RIPE transitions from Flat to Hierarchic Repository Structure

Citation: Austein's measurements – see slides 10 in [2].

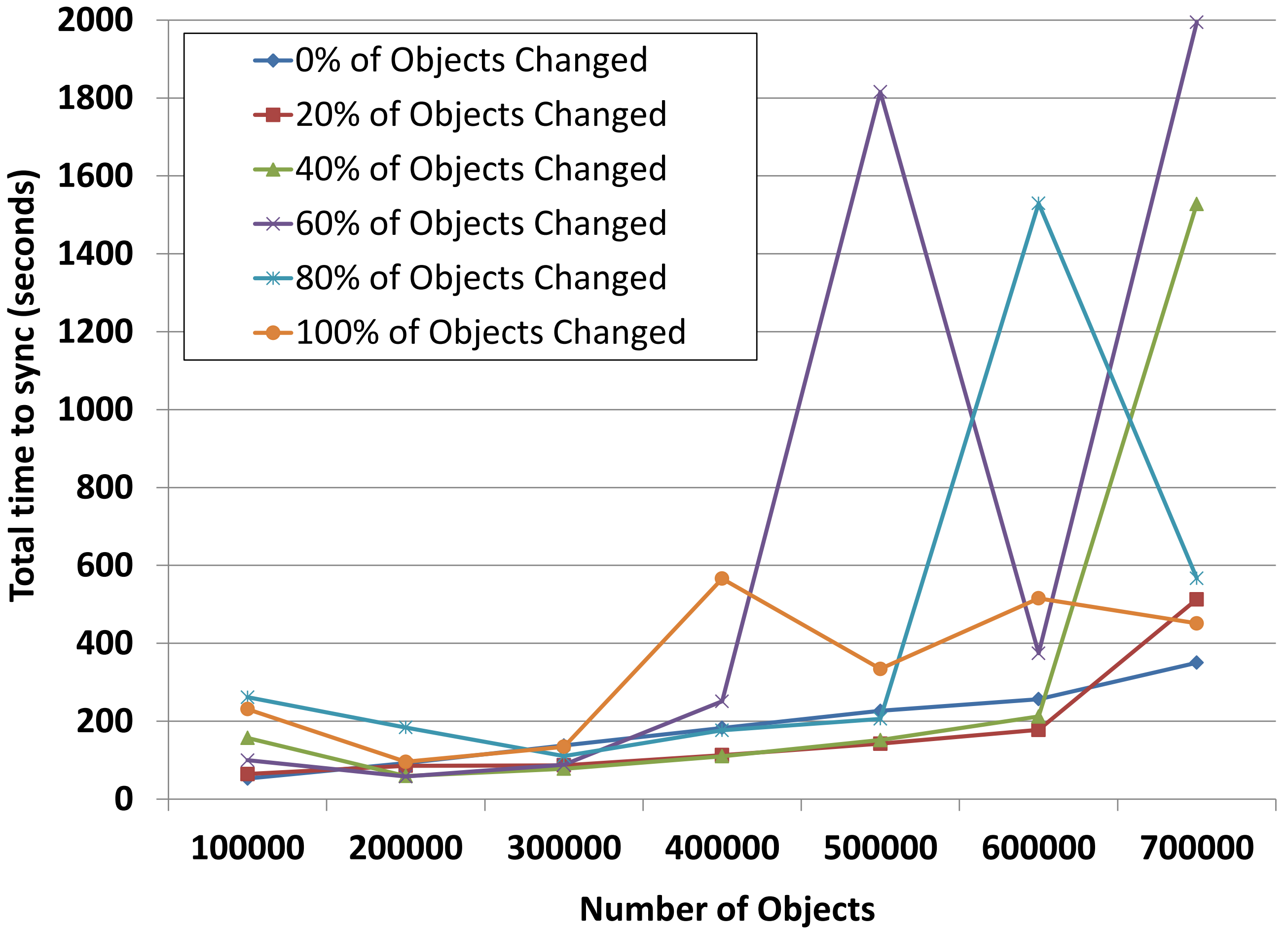
## Measurements with Synthesized RPKI Objects: ms/Object (w/o Network Delays)



- Client and repository running on two virtual machines on the same computer (Intel Core i7 CPU at 2.2 GHz)
- So no network delays are involved

Source: Measurement data from D. Mandelberg BBN [4].

# Measurements with Synthesized RPKI Objects: Measured rsync Delay (with Network Delays)

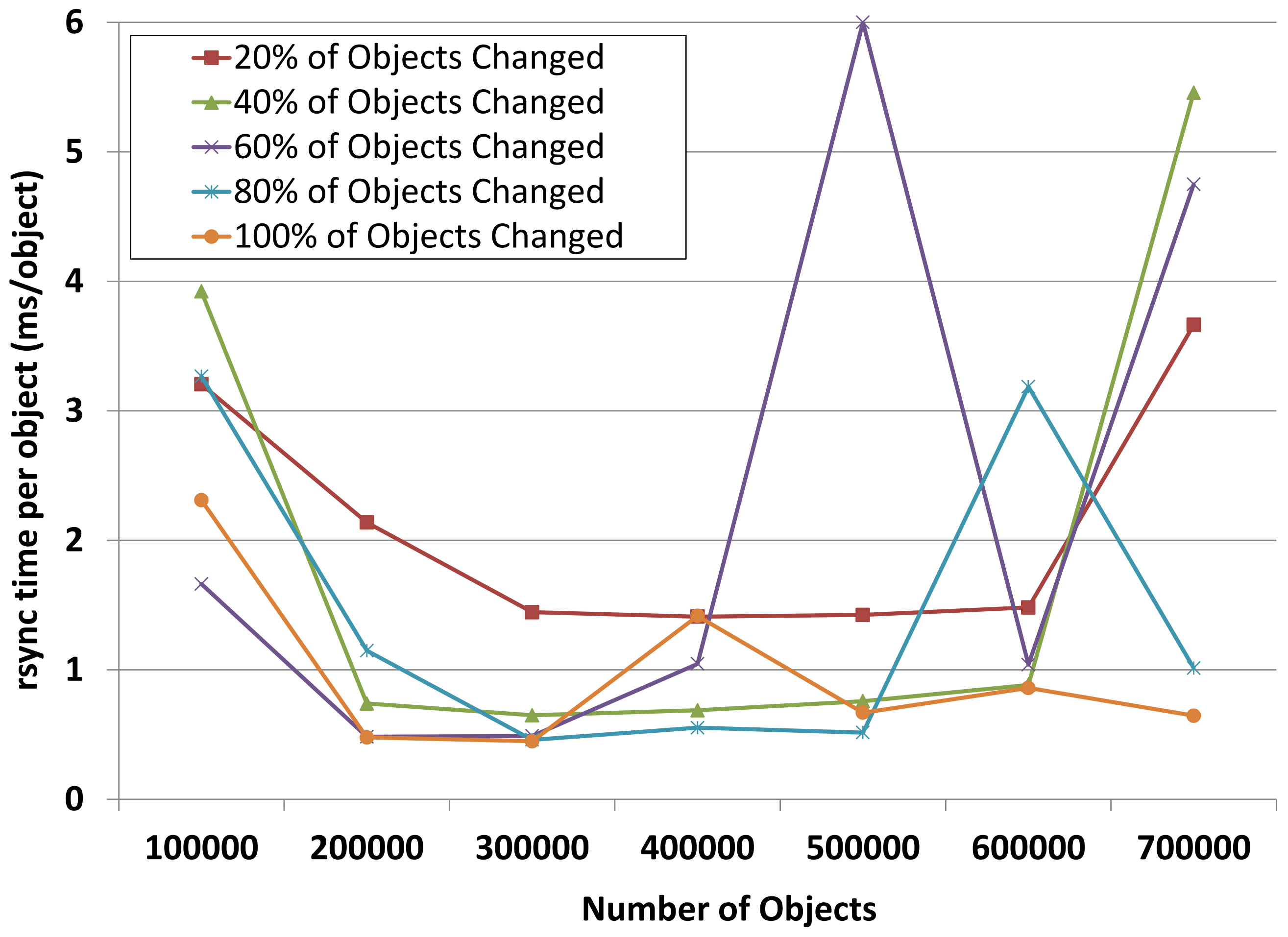


TCP receive window scaling enabled by default in Linux kernels

- Client: VM in Boston, MA with two 2.4GHz, 2GB RAM
- Server: Amazon EC2 m1.large instance in Tokyo (two cores, 7.4GB ram)

Source: Measurement data from D. Mandelberg BBN [4].

# Measurements with Synthesized RPKI Objects: ms/Object (with Network Delays)



**Avg. ms/obj = 2.15 ms**

TCP receive window scaling enabled by default in Linux kernels

- Client: VM in Boston, MA with two 2.4GHz, 2GB RAM
- Server: Amazon EC2 m1.large instance in Tokyo (two cores, 7.4GB ram)

Source: Measurement data from D. Mandelberg BBN [4].

# Other Modeling Parameters

| **Repostories:** | |
|---|---|
| R = # repositories (range considered) | 500 - 7,000 |

- All stub ASes are expected to outsource CA/repository operation
- Some of the non-stub ASes will also outsource
- A fraction of non-stub ASes will operate CAs/repositories
- Total # non-sub ASes = 6880
- This study varies the # repositories from 500 to 7000

| **Parallel fetching:** | |
|---|---|
| F = # parallel fetches (RP simultaneously fetching from different repositories) | 5 |

- RP can run even 20 to 30 rsync fetch threads simultaneously to different repositories
- We conservatively assume 5X speedup due to parallel fetches

# Rsync Download Delay Components

Connection setup overhead:

$$D_c = \frac{R * C}{F} \quad \text{ms}$$

rsync processing, network:

$$D_r = \frac{N * T}{F} \quad \text{ms}$$

$F$ = # parallel fetches from RP to repositories
$R$ = # repositories
$C$ = session setup time per visit to repository (ms)
$N$ = # RPKI objects to be downloaded
$T$ = measured ms/obj **actual** rsync download time

Assumption: *R* > *F*

Total rsync download delay, in minutes:

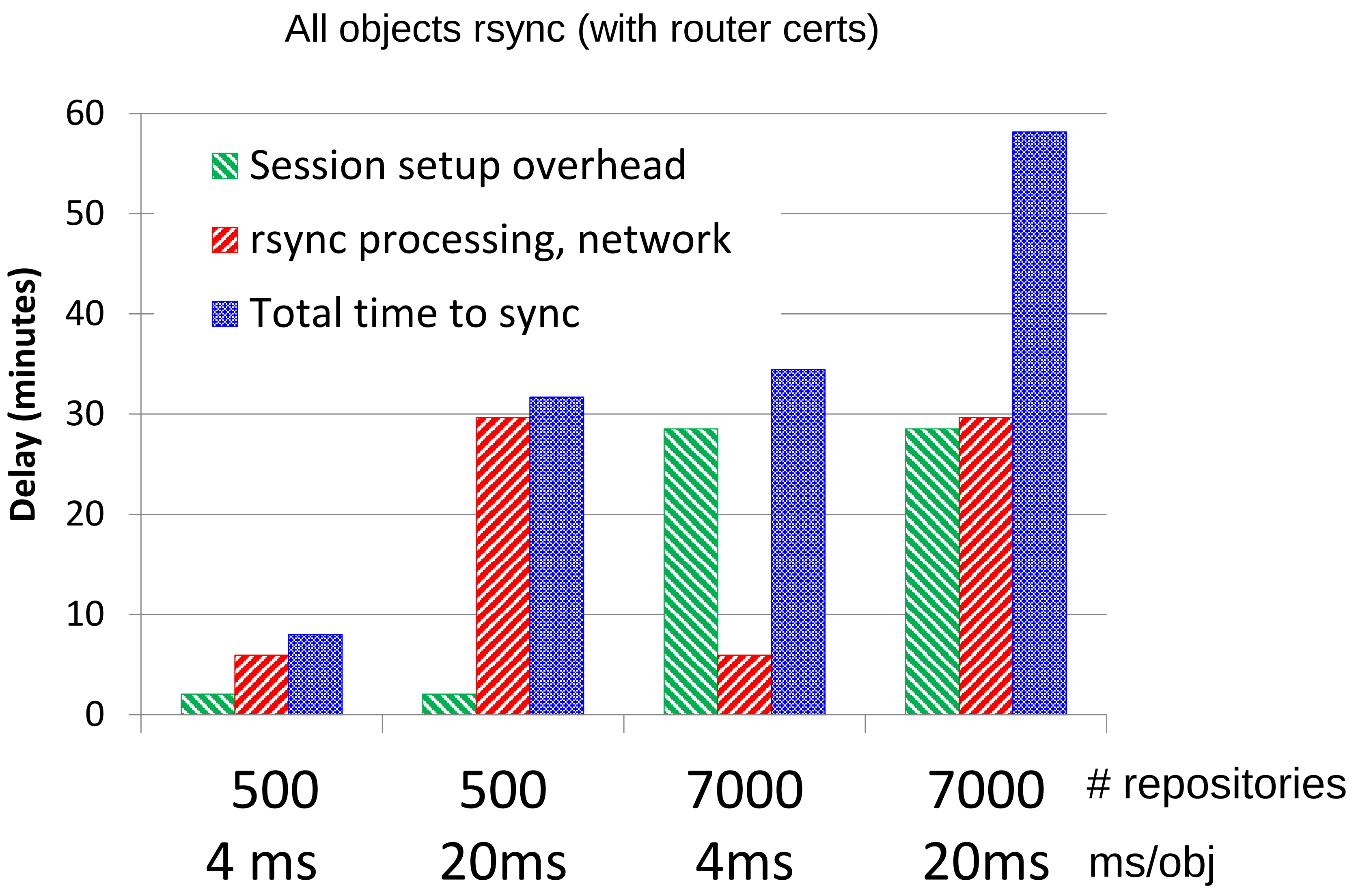
$$D = (D_c + Dr)/60{,}000$$

See slides 8, 9 and 13 for the values of the parameters used in our modeling:
We choose *C* = 1220 ms, conservatively (slide 9).
*F* = 5 (slide 13).
*R* is varied from 500 to 7000 (slide 13).
*T* is varies from 4 ms to 20 ms (see slides 9,15).
*N* varies based on the download scenario (slide 8).

## Future Envisioned Speedup Factors

- We already have measurements of 12 ms/object and 17 ms/object (including network delays) from BBN and R. Austein (slide 9).
- A significant part of the above ms/object numbers may be the network delays (between the US and the Netherlands) – inferred from info on slide 12.
- Hence, with repository mirroring (by RIRs, LIRs, major ISPs) in the future, the ms/object number may come down significantly.
- Faster processors and parallel processing using multiple cores may also provide further speedup.
- The next slide considers a range of ms/object values (excluding session setup/teardown overhead) from 4 ms/obj to 20 ms/obj
- Session setup/teardown delays are included separately in our modeling as per eqns. on slide 14.

# Time to sync: Components



## Effect of Polling Interval Value **(1/2)**

- When the polling interval is reduced, say, from 24 hrs to 16 or 8 hrs, then the number of changed objects during each visit will be reduced accordingly by a factor of 2/3 or 1/3, respectively, relative to the corresponding 24-hr-polling numbers (see Slide 8).
- Hence, for the smaller polling intervals, the rsync processing plus network-delay part of the Total Time to Sync goes down.
- But the # repositories to be visited remains unchanged.
- So the session overhead part of the Total Time to Sync remains unchanged and also becomes more dominant (at high # repositories)
- See numerical results on the next slide.

# Effect of Polling Interval Value **(2/2)**

ms/obj: *T* = 20 ms (the high end of the range)
# repositories: ***R* = 7000** (the high end of the range)

| | Total time to sync (minutes) | | |
|---|---|---|---|
| Polling interval (hours) | Download changes only at polling intervals (w/o router certs) | Download changes only at polling intervals (with router certs - rollover annually) | Download changes only at polling intervals (with router certs - rollover daily) |
| 8 | 30.63 | 30.64 | 34.08 |
| 16 | 32.74 | 32.76 | 39.64 |
| 24 | 34.86 | 34.88 | 45.20 |
| | | | |
| # objects | 95,144 | 95,569 | 250,288 |

Note: When # repositories is large, the session setup time is the biggest component; so the variations for different polling intervals are not too pronounced.

ms/obj: *T* = 20 ms (the high end of the range)
# repositories: ***R* = 500** (the low end of the range)

| | Total time to sync (minutes) | | |
|---|---|---|---|
| Polling interval (hours) | Download changes only at polling intervals (w/o router certs) | Download changes only at polling intervals (with router certs - rollover annually) | Download changes only at polling intervals (with router certs - rollover daily) |
| 8 | 4.15 | 4.16 | 7.60 |
| 16 | 6.27 | 6.28 | 13.16 |
| 24 | 8.38 | 8.41 | 18.72 |
| | | | |
| # objects | 95,144 | 95,569 | 250,288 |

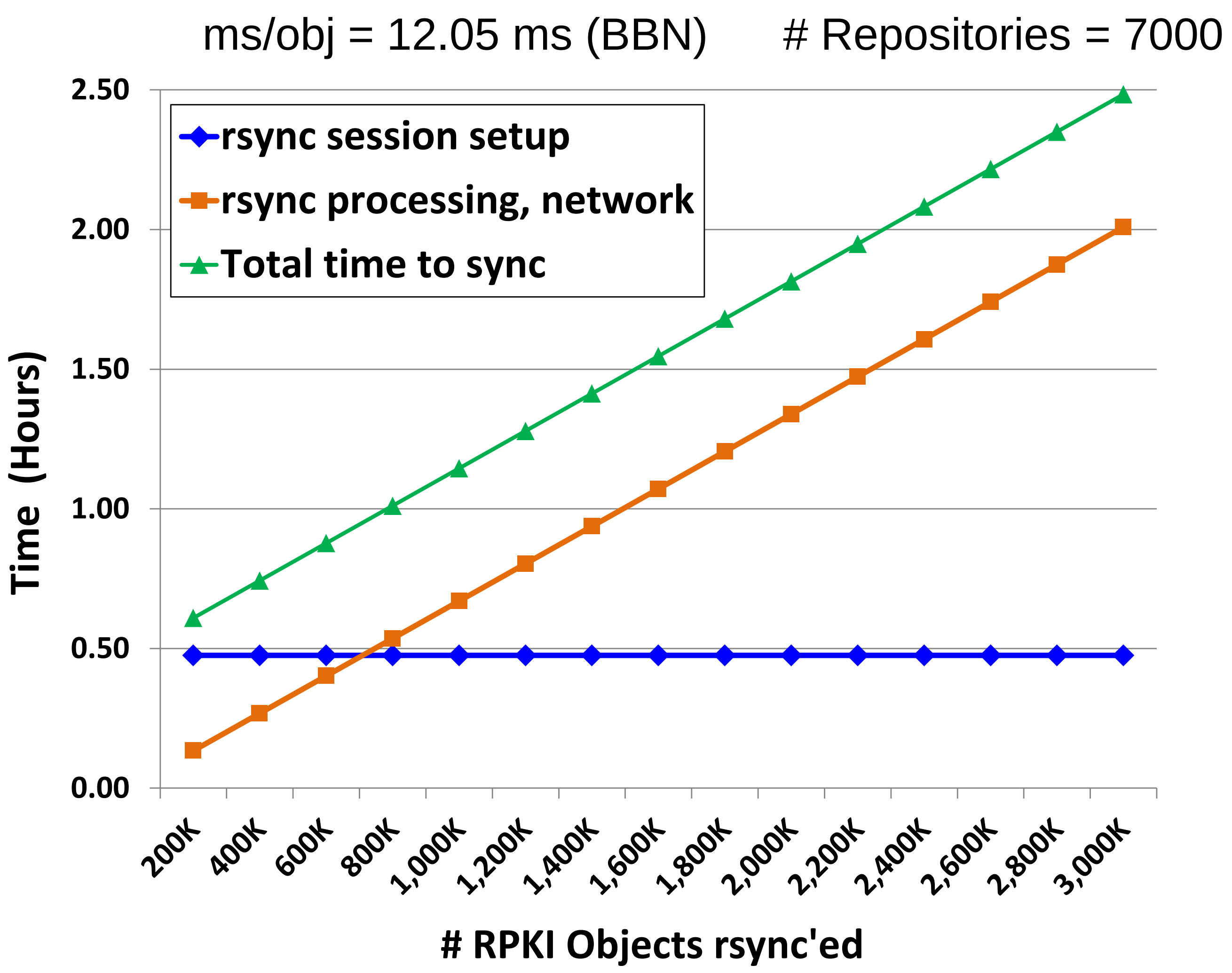
But here they are pronounced but doesn’t matter because the delay numbers here are generally much smaller.

## Different Estimates of # RPKI Objects

| | Estimated Total # RPKI Objects |
|---|---:|
| This Effort | 444,645 |
| Eric Osterweil et al. [5] | 2,651,000 |
| Tim Bruijnzeels et al. [8] | 3,000,000 |

- Other estimates [5][8] of # RPKI objects are large primarily due the assumption that there will be 1 Million BGPSEC routers worldwide and each will have its distinct pair of router certs (keys)!
- The key assumption in [5][8] is that each BGPSEC router will have a distinct router key. But it is unlikely; instead each non-stub AS will have a few security zones and router keys will likely be shared within each zone.
- Anyway, we vary the # RPKI objects up to 3,000,000 (see next slide)

# Total Time to Sync: Vary # RPKI Objects



- This is download time for rsync’ing ALL objects

## Conclusions

- **New measured values of rsync speed (ms/object) for the hierarchic-structured rpki.ripe.net are much lower:**
  - 12 ms/object (BBN) and 17 ms/object (Austein) (see [1][2][3])
  - These basic per unit measurements are 52X and 35X lower, respectively, than what was measured in the past (avg. 628 ms/object) (see [4][5]).
- Global rsync download delays have been estimated for **realistic ranges of # repositories, # RPKI objects, and ms/object measure.**
- The effects of session setup overheads and parallel fetching have been included explicitly in the model.
- **Based on the results of this modeling, we project global rsync delays, as seen by each RP doing an incremental fetch, to be in the range of single digit minutes to tens of minutes.**
- Even lower values are possible with additional optimizations to current prototype and production RPKI systems.

## Backup material

# Total Time to Sync: Vary ms/obj

Keep # repositories fixed: *R* = 7000 (the high end of the range)

| | Total time to sync (minutes) | | | | |
|---|---|---|---|---|---|
| Rsync delay parameter (ms/obj) | All object rsync download (w/o Rtr certs) | All object rsync download (with Rtr certs) | Download changes only at 24-hr polling intervals (w/o router certs) | Download changes only at 24-hr polling intervals (with router certs - rollover annually) | Download changes only at 24-hr polling intervals (with router certs - rollover daily) |
| 4 | 32.37 | 34.44 | 29.78 | 29.79 | 31.85 |
| 6 | 34.30 | 37.41 | 30.42 | 30.42 | 33.52 |
| 8 | 36.23 | 40.37 | 31.05 | 31.06 | 35.19 |
| 10 | 38.16 | 43.33 | 31.68 | 31.70 | 36.86 |
| 12 | 40.09 | 46.30 | 32.32 | 32.34 | 38.52 |
| 14 | 42.02 | 49.26 | 32.95 | 32.97 | 40.19 |
| 16 | 43.95 | 52.23 | 33.59 | 33.61 | 41.86 |
| 18 | 45.88 | 55.19 | 34.22 | 34.25 | 43.53 |
| 20 | 47.81 | 58.16 | 34.86 | 34.88 | 45.20 |
| | | | | | |
| # objects | 289,501 | 444,645 | 95,144 | 95,569 | 250,288 |

# Total Time to Sync: Vary ms/obj

Keep # repositories fixed: *R* = 500 (the low end of the range)

| | Total time to sync (minutes) | | | | |
|---|---|---|---|---|---|
| Rsync delay parameter (ms/obj) | All object rsync download (w/o Rtr certs) | All object rsync download (with Rtr certs) | Download changes only at 24-hr polling intervals (w/o router certs) | Download changes only at 24-hr polling intervals (with router certs - rollover annually) | Download changes only at 24-hr polling intervals (with router certs - rollover daily) |
| 4 | 5.90 | 7.97 | 3.31 | 3.31 | 5.37 |
| 6 | 7.83 | 10.93 | 3.94 | 3.95 | 7.04 |
| 8 | 9.76 | 13.89 | 4.57 | 4.59 | 8.71 |
| 10 | 11.69 | 16.86 | 5.21 | 5.22 | 10.38 |
| 12 | 13.62 | 19.82 | 5.84 | 5.86 | 12.05 |
| 14 | 15.55 | 22.79 | 6.48 | 6.50 | 13.72 |
| 16 | 17.48 | 25.75 | 7.11 | 7.13 | 15.39 |
| 18 | 19.41 | 28.72 | 7.75 | 7.77 | 17.05 |
| 20 | 21.34 | 31.68 | 8.38 | 8.41 | 18.72 |
| | | | | | |
| # objects | 289,501 | 444,645 | 95,144 | 95,569 | 250,288 |

## Total Time to Sync: Vary # Repositories

Keep ms/obj fixed: *T* = 20 ms (the high end of the range)

| | Total time to sync (minutes) | | | | |
|---|---|---|---|---|---|
| # repositories | All object rsync download (w/o Rtr certs) | All object rsync download (with Rtr certs) | Download changes only at 24-hr polling intervals (w/o router certs) | Download changes only at 24-hr polling intervals (with router certs - rollover annually) | Download changes only at 24-hr polling intervals (with router certs - rollover daily) |
| 500 | 21.34 | 31.68 | 8.38 | 8.41 | 18.72 |
| 1000 | 23.37 | 33.72 | 10.42 | 10.44 | 20.76 |
| 1500 | 25.41 | 35.75 | 12.45 | 12.48 | 22.80 |
| 2000 | 27.45 | 37.79 | 14.49 | 14.52 | 24.83 |
| 2500 | 29.48 | 39.83 | 16.53 | 16.55 | 26.87 |
| 3000 | 31.52 | 41.86 | 18.56 | 18.59 | 28.91 |
| 3500 | 33.56 | 43.90 | 20.60 | 20.63 | 30.94 |
| 4000 | 35.59 | 45.94 | 22.64 | 22.66 | 32.98 |
| 4500 | 37.63 | 47.97 | 24.67 | 24.70 | 35.02 |
| 5000 | 39.67 | 50.01 | 26.71 | 26.74 | 37.05 |
| 5500 | 41.70 | 52.05 | 28.75 | 28.77 | 39.09 |
| 6000 | 43.74 | 54.08 | 30.78 | 30.81 | 41.13 |
| 6500 | 45.78 | 56.12 | 32.82 | 32.85 | 43.16 |
| 7000 | 47.81 | 58.16 | 34.86 | 34.88 | 45.20 |
| | | | | | |
| # objects | 289,501 | 444,645 | 95,144 | 95,569 | 250,288 |

## Total Time to Sync: Vary # Repositories

Keep ms/obj fixed: *T* = 4 ms (the low end of the range)

| | Total time to sync (minutes) | | | | |
|---|---|---|---|---|---|
| # repositories | All object rsync download (w/o Rtr certs) | All object rsync download (with Rtr certs) | Download changes only at 24-hr polling intervals (w/o router certs) | Download changes only at 24-hr polling intervals (with router certs - rollover annually) | Download changes only at 24-hr polling intervals (with router certs - rollover daily) |
| 500 | 5.90 | 7.97 | 3.31 | 3.31 | 5.37 |
| 1000 | 7.93 | 10.00 | 5.34 | 5.35 | 7.41 |
| 1500 | 9.97 | 12.04 | 7.38 | 7.38 | 9.45 |
| 2000 | 12.01 | 14.08 | 9.42 | 9.42 | 11.48 |
| 2500 | 14.04 | 16.11 | 11.45 | 11.46 | 13.52 |
| 3000 | 16.08 | 18.15 | 13.49 | 13.49 | 15.56 |
| 3500 | 18.12 | 20.19 | 15.53 | 15.53 | 17.59 |
| 4000 | 20.15 | 22.22 | 17.56 | 17.57 | 19.63 |
| 4500 | 22.19 | 24.26 | 19.60 | 19.60 | 21.67 |
| 5000 | 24.23 | 26.30 | 21.64 | 21.64 | 23.70 |
| 5500 | 26.26 | 28.33 | 23.67 | 23.68 | 25.74 |
| 6000 | 28.30 | 30.37 | 25.71 | 25.71 | 27.78 |
| 6500 | 30.34 | 32.41 | 27.75 | 27.75 | 29.81 |
| 7000 | 32.37 | 34.44 | 29.78 | 29.79 | 31.85 |
| | | | | | |
| # objects | 289,501 | 444,645 | 95,144 | 95,569 | 250,288 |

## References

1. R. Austein, R. Bush, and M. Elkins, "A few months in the life of an RPKI validator," Presentation slides, January 2013. (See measurement data on slide 24 corresponding to the hierarchical repository structure at rpki.ripe.net) http://www.hactrn.net/opaque/a-few-months-in-the-life-of-an-rpki-validator-2013-01-05.pdf
2. R. Austein, R. Bush, and M. Elkins, "RIPE goes hierarchical," Presentation slides, January 2013. (See slides 6 and 10.) http://www.hactrn.net/presentations/2013-01-15.ripe-goes-hierarchical.pdf
3. A. Chi and D. Mandelberg (BBN), Private communication, February 2013.
4. R. Bush, "Measuring RPKI repositories," NANOG-56, October 2012. http://www.nanog.org/meetings/nanog56/presentations/Monday/mon.general.bush.26.pdf
5. E. Osterweil, T. Manderson, and R. White, "Sizing Estimates for a Fully Deployed RPKI," Verisign Labs Technical Report #1120005 version 2, December 2012. http://techreports.verisignlabs.com/tr-lookup.cgi?trid=1120005&rev=2
6. A. Chi, "Update on RPKI validator testing," IETF 85, November 2012. http://www.ietf.org/proceedings/85/slides/slides-85-sidr-11.pdf
7. K. Sriram and Randy Bush, "Estimating CPU Cost of BGPSEC on a Router," Presented at the RIPE 63, October 2011, (see slide 15), http://ripe63.ripe.net/presentations/127-111102.ripe-crypto-cost.pdf
8. T. Bruijnzeels, O. Muravskiy, and B. Weber, "RPKI Repository Analysis and Requirements," IETF Draft, February 2012, http://datatracker.ietf.org/doc/draft-tbruijnzeels-sidr-repo-analysis/