RPKI Rsync Delay Modeling

Tech-note: Some responses to Eric's post on the IETF SIDR list

Authors: Kotikalapudi Sriram and Doug Montgomery

November 27, 2012

Eric's post at: http://www.ietf.org/mail-archive/web/sidr/current/msg05346.html

Link to Eric and Danny's tech-note: http://techreports.verisignlabs.com/tr-lookup.cgi?trid=1120005

Hi Eric,

We have read your tech-note and also followed the email thread. Thank you for the study. We have several comments.

A. Consideration of a range of basic rsync fetch time measurements (seconds/object):

Regarding the basic rpki rsync measurement of fetch time (seconds per object), there seems to be more evidence that it could be a lot faster than the measurements you have used from Randy's NANOG presentation. Tim has reaffirmed once again in his most recent post (http://www.ietf.org/mail-archive/web/sidr/current/msg05369.html) that the fetch times seem to be in the range of 10 to 20 ms per object rather than the 628 ms that you've used. Tim mentions that "Randy's numbers may be outdated and represent the totals for our old *flat* rsync repo. This adds a huge amount of latency and setup overhead." Randy also separately mentioned the updated 10 ms/object measurement in one of his responses to your original post.

So we thought it would be good to redo your total RPKI rsync delay numbers considering these lower measurements from Tim/Randy. The resulting table and a plot are shown below:

Table 1: Rsync Delay vs. # RPKI Objects

	Input: Rsync delay per object (seconds/object)		
	Estimated by Eric from Randy's NANOG-56 preso	RIPE (best case - cited by Randy on sidr list)	Tim B.'s number (cited on sidr list)
	0.628	0.01	0.02
# RPKI Objects	Total rsync delay (Hours)		
100	0.0174	0.0003	0.0006
1000	0.1744	0.0028	0.0056
10000	1.744	0.028	0.056
100000	17.44	0.28	0.56
500000	87.22	1.39	2.78
1000000	174.44	2.78	5.56
1492000	260.27	4.14	8.29
1500000	261.67	4.17	8.33
2000000	348.89	5.56	11.11

Eric's estimate

A lot smaller estimates

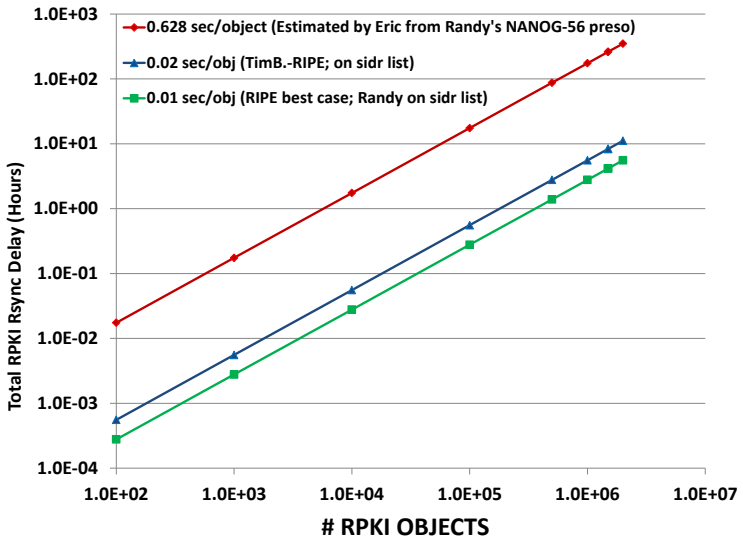

Figure 1: Plot of Rsync Delay vs. # RPKI Objects

As you can see from Table 1, for the same number of total rpki objects (approx. 1.5M) that you have used, the total rpki rsync delays is only about 4 hours and 8 hours, respectively, for 10 ms/object and 20 ms/object, which are 63 and 31 times smaller, respectively, as compared to the 260 hours number in your table. A plot of the data in Table 1 is shown in Figure 1. Note that both axes are logarithmic.

It is also important to note that a running RP fetches only *changed* RPKI objects during each polling instance in normal (steady-state) operation (i.e., whatever the delta happens to be since the previous polling instance). The RP would download *all* RPKI objects rather rarely, e.g., when there is a restart (i.e., failure recovery). So the common fetch delay seen at RPs would be more like seconds or minutes as shown in the lower left corner of Figure 1, corresponding to fetches of 100 to 10,000 changed rpki objects.

B. Estimate of RPKI repositories

It is known that 84% of all ASes are stub, and 16% are non-stub. So about 35,280 are stub ASes and only 6,720 are non-stub. The stub ASes are likely to only run simplex bgpsec and they would likely contract out publication point service to their upstream ISP or a third party (Tim also observed this). Even many non-stub ASes can be expected to do the same. So in the end, we think the total number of repositories (what you call SIAs) will be O(100) or O(1000), much less than the 42,000 you have assumed.

C. Number of router certs:

Observe that per-router certs are not required; certs can be per AS. It remains to be seen what granularity (ranging from per AS to per router) will be used. But, for worse case, if we assume per-router certs in all ASes, then a conservative estimate can be obtained as follows. The stub ASes will have a low average # eBGPSEC routers (say, 2 per AS), and the non-stub ASes will have a relatively larger value for the same (say, 20 per AS). Then the total number of eBGPSEC routers can be estimated at 204,960 (=

20*6720 + 2*35280). You had assumed 420,000 (10 x 42,000). Also, there should be 4 certs per *non-stub* eBGPSEC router; one pair (current and next) for origination prefixes and another pair (current and next) for transit ASes (please see draft-rogaglia-sidr-bgpsec-rollover and draft-sriram-replay-protection-design-discussion). And there should be only 2 certs per *stub* eBGPSEC router since it does not have transit prefixes. That gives us an estimate of 678,720 (=4*20*6720 + 2*2*35280) for the total # router certs. We think this number is conservative (high) but reasonable for now.

D. Time for caches to receive updated RPKI info

Quoting from your original post in this thread:

> in order to ensure that all caches have received updated information

>(such as getting new certificates/CRLs/etc disseminated),

>repositories may have to wait about a month (roughly 32 days by our estimates),

>just for gatherers to reliably pick up a repo's changes.

>Or, a month before a key compromise might get remediated throughout the RPKI system.

We find it hard to understand your assumptions and computation here (including the details about this in your tech-note). Here we should focus on fetching only the delta (changes) since the last fetch. From my table (slide 1), the delta fetch should take only seconds to minutes (also noted above in comment A for fetches of RPKI changes up to 1000 or 10,000 objects). If you factor in time-jittered polling from caches once every z hours, then the delay is of the order of z hours for the caches to fetch updated info from the time the info is available at the publication points. Clearly, Randy's NANOG results (60 to 80 minute delay range) are dominated by the one hour jitter on polling gatherers. It seems that the major contributing factor is the polling interval and not the workload at the publication points.

E. Characterizing the Size and Shape of RPKI (pointer to a NIST study)

Okhee Kim (colleague at NIST) led an effort to characterize the size and shape of a deployed RPKI a couple of years ago. She based it on measurements of address space registrations (inet-num in RPSL, NetHandle in SWIP), AS registrations (aut-num in RPSL, AS-Handle in SWIP) and route object registrations (in IRRs, RADB). The presentation slides of this study can be accessed at:

http://www.nist.gov/itl/antd/upload/RPKI-size-shape-modeling.pdf

This work was done almost two years ago and we haven't had a chance to update the results since then. But you may glance through the slides just to get an idea. There are some interesting analyses and insights reported in the presentation.

We hope that the comments/analysis offered here are helpful.