Sizing Estimates for a Fully Deployed RPKI

[Verisign Labs Technical Report #1120005 version 2]

Eric Osterweil
Verisign Labs
eosterweil@verisign.com

Terry Manderson
ICANN
terry.manderson@icann.org

Russ White
Verisign Labs
rwhite@verisign.com

Danny McPherson
Verisign Labs
dmcpherson@verisign.com

1 Introduction

As the Resource Public Key Infrastructure (RPKI) [12] has begun its early deployment trials, estimating how large a global RPKI deployment must be to certify today's Internet has become increasingly relevant. In this document, we present a "back-of-the-envelope" style evaluation of the number of objects required to fully deploy a global RPKI in today's Internet, the performance characteristics of the RPKI under potential usage models (i.e., the degree of "hosting" employed, cache replication strategies, etc.), and how deployments of the RPKI under these different deployment models might scale. This work is intended to be a first stab at addressing these questions, and the authors welcome feedback, corrections, comments, etc. Note this document does not try to estimate the increasing size of the RPKI system due to massive new connectivity expected as IPv6 grows. With these caveats in mind, we begin from the perspective of a thought exercise, and pose the following questions:

- What would be the size of a fully deployed RPKI in today's Internet?
- What is the complexity of finding/learning/gathering a fully deployed RPKI?
- How long might such a gathering process actually take?

To answer these questions, we begin by estimating the size of the sum of the RPKI's various objects contained across the global set of repositories. These object types (described in [12]) are: CA certificates, Route Origin Authorizations (ROAs), Manifests, Certificate Revocation Lists (CRLs), and ghostbuster records. Objects like the ROA and ghostbuster record contain an End Entity (EE) certificate in their ASN.1 structure. Since each certificate is maintained in a repository, the caching system must download all objects from all repositories before validation computation can begin. While there are currently 5 RIRs running repositories, it is not clear if that will always be the case (the so-called "hosted" approach), if each Autonomous System (AS) ultimately ends up running its own repository, or some approach in between these two extremes eventually becomes the normal deployment model. To account for each of these cases, our analysis considers a range of cases where there are: 5 (the current set of RIRs), 10, 100, 1000, 10000, 42000 (the number of Autonomous System Numbers, ASNs, routed today) repositories.

What we find below is that with a lower-bound estimate of about 601,337 objects (assuming no BGPsec deployment) and 2,601,377 objects when RPKI is used to support a global BGPsec deployment. From these numbers, a globally deployed RPKI (in the worst cases) would likely take between 15 days to over 30 days to synchronize the local cache at every Relying Party (RP) in the world. While the methodology and measurements in this document are meant to be clear and concise, they are simply submitted as candidates of what a globally deployed RPKI *might* look like. Corrections are welcome, and alternate methodologies and measurements are very much encouraged. This work is intended to be a living document, which should evolve as operational experience is gained with the RPKI.

c^{CA}	A CA certificate
c^{EE}	An End Entity certificate
m	A manifest object
$o^M = (M, C^{EE})$	A manifest object with its EE cert
crl	A Certificate Revocation List (CRL)
$o^{CRL} = (CRL, C^{EE})$	A CRL object with its EE cert
gb	A Ghost Busters record
$o^{GB} = (GB, C^{EE})$	A Ghost Busters record and its EE cert
roa	A Route Origin Authorization (ROA)
$o^{ROA} = (ROA, C^{EE})$	A ROA and its EE cert
ca	A Certificate Authority (not to be confused with a CA certificate)
rir	A Regional Internet Registry CA
org	An organization in the routing system (an ISP, an enterprise, a service provider, etc.)
RIR	The set of all RIRs
ORG	The set of all organizations responsible to routing in the Internet (ISPs, enterprises, service providers, etc.)

Table 1: Object definitions

2 Methodology

The methodology used here is to examine the types of objects that must be generated to create a complete RPKI, then to estimate the size or number of each of these objects, and finally to combine these two estimates to provide a projected RPKI size in terms of individual replicated objects. Each section below represents an individual part of this process.

2.1 Object Definitions

In our candidate methodology, we enumerate the different types of objects according to the naming scheme in Table 1. One important clarification is that the RPKI architecture requires *all* objects to have verifying *End Entity* (EE) Certificates associated with them. However, as an optimization, objects are always transferred with their EE certs, as if they were one object. For example, one would not retrieve a *crl* object alone. Rather, one would transfer an o^{CRL} tuple. While this reduces transfer costs, the cryptographic material in the c^{EE} certificate in o^{CRL} has potentially *very* different operational complexities associated with it. Certificates' private portions may need to be managed in HSMs, the creation of new ROAs may (or may not) require new cryptographic certificate generation, etc. Thus, it is critical to account for their separate natures in different operational contexts.

According to the architecture of the RPKI, each *ca* must contain certain constituent objects: an organization-level CA cert, a CRL tuple, and a ghost busters tuple:

$$ca_i = \{c_i^{CA}, o_i^{CRL}, o_i^{GB}\}$$

This set *may* include a manifest object o^M, if it is hosted in a hierarchical repository (i.e. not a flat repo), or in its own separate repository.

An organization must manage its own *ca* and the allocation resource objects and its routing objects. That is:

$$org_i = \{ca_i, \{o_1^{ROA}, o_2^{ROA}, \ldots o_n^{ROA}\}, \{c_{rtr1}^{EE}, c_{rtr2}^{EE}, \ldots, c_{rtrm}^{EE}\}\}$$

We generalize ca_i set to be called $InfaObjs$, and the set of allocation objects (ROA tuples) to be called $AllocObjs$ and the set of routing keys (EE certs for routers) to be called $RoutingObjs$, and (finally) the repository provisioning/structure dictates if an organization needs its own o^M object (standalone or hierarchical repositories need this, flat structures do not) and we call this set (that might be empty) $StructObjs$. That makes $org_i = \{InfaObjs, AllocObjs_i, RoutingObjs_i, StructObjs_i\}$. At this point we note the cardinalities of these sets. Each organization will normally have a 1:1 cardinality with its $InfaObjs$, 1:n cardinality with its $AllocObjs$, and 1:m cardinality with its $RoutingObjs$.

In addition, each *rir* is currently planning to run 5 *ca*s internally (one for each of the other RIRs, and one for ICANN):

$$rir_a = \{ca^{RIR}_{ICANN}, ca^{RIR}_b, ca^{RIR}_c, ca^{RIR}_d, ca^{RIR}_e\}$$

The rationale here is to certify transfer blocks between RIRs, and to account for ERX/transfers (legacy numbers). A detailed explanation is beyond the scope of this sizing document.

With a few simple assumptions, we can model the expected size of a global deployment of RPKI from the above definitions. If we consider the total *downloadable* object count to be the set of all RIRs and all the objects needed to attest to all allocations and routers in the Internet today, we arrive at the following expression.

$$O^{Total} = \{RIR, ORG\}$$

With the above expansion of *rir*, we can expand O^{Total} as follows:

$$O^{Total} = \{\bigcup_{i=1}^{5} (ca^{RIR}_{ICANN}, ca^{RIR}_{i+1}, ca^{RIR}_{i+2}, ca^{RIR}_{i+3}, ca^{RIR}_{i+4}), ORG\}$$

Next, we make the observation that since each object in the RPKI is intended to come from a single root (i.e. no object can have multiple roots in its cryptographic delegation chain), then the intersection between *any* two object sets (infrastructure, allocation, or routing objects) will necessarily be $\emptyset$. For example, consider two organizations A and B. When comparing the infrastructure object sets for these two organizations, we will always be able to say $InfraObjs_A \cap InfraObjs_B = \emptyset$. This observation has the powerful side effect that it allows us to describe the aggregate sets that amalgamate all sets of each type of object into corresponding supersets:

$$\widehat{InfraObjs} = \bigcup_{i=1}^{|ORG|} InfraObjs_i$$

$$\widehat{AllocObjs} = \bigcup_{i=1}^{|ORG|} AllocObjs_i$$

$$\widehat{RoutingObjs} = \bigcup_{i=1}^{|ORG|} RoutingObjs_i$$

$$\widehat{StructObjs} = \bigcup_{i=1}^{|ORG|} StructObjs_i$$

These amalgamations allow us to further expand O^{Total} as follows:

$$O^{Total} = \{\bigcup_{i=1}^{5} (ca^{RIR}_{ICANN}, ca^{RIR}_{i+1}, ca^{RIR}_{i+2}, ca^{RIR}_{i+3}, ca^{RIR}_{i+4}), \widehat{InfraObjs}, \widehat{AllocObjs}, \widehat{RoutingObjs}, \widehat{StructObjs}\}$$

Next, consider the relative sizes of these sets as they would be in a fully deployed RPKI system, today. To do this, we consider the size of what's deployed in today's Internet, today: 5 RIRs, 42,000 ASNs, and 431,000 Prefixes. A reported estimate of the number of eBGP speaking routers is 1 million [6] (this makes roughly $\frac{1{,}000{,}000}{42{,}000} = 23.81$ eBGP routers per AS).[1]

We note that the number of prefixes here does not include *dark prefixes*, or rather publicly usable address blocks allocated for private use. Whether or not these prefixes are ever advertised on the public Internet, they are still often used in private peering arrangements, and the organizations that own them would certainly still want certificates generated and distributed for these blocks. Currently, neither the literature nor conventional operational wisdom describe ways to measure or estimate the number of dark prefixes that currently exist but which are not advertised. However, we could take 2^{16} as the probable number of address allocations that will ultimately have certificates distributed through any given origin authentication system.

When a new organization receives an allocation from an RIR, this will result in three necessary downloadable objects: $\{c^{CA}, o^{GB}, o^{CRL}\}$, and depending on the repository provisioning/structure it may have o^M. In addition,

[1] While the authors appreciate the estimation, we are still investigating a more systematic method of estimating this number.

Set	Derivation	Count
$\|RIR\|$	$(5 \times 4) * 5$	100
$\|\widehat{InfraObjs}\|$	$(20,246 + 2,025 + 61,505) \times 3$	251,328
$\|\widehat{AllocObjs}\|$		273,592
$\|\widehat{RoutingObjs}\|$	$42,000 \times 23.8$	2,000,040
$\|\widehat{StructObjs}\|$	$\|ORG\| = 20,246 + 2,025 + 61,505$	83,776

Table 2: Approximation of object set sizes

at least one additional o^{ROA} will be needed for each RIR that an organization gets an allocation from (IP or ASN). That means each $InfraObjs$ set will account for *at least* three objects. We, therefore, make the general estimate that the number of organizations today is the sum of RIR members plus the number of ERX legacy blocks (as those must be represented separately, even for existing organizations) plus the number of suballocations *from* RIR members times three. More formally:

$$|\widehat{InfraObjs}| = (|RIR^{mem}| + |ERX| + |SubAlloc|) \times 3$$

Where RIR^{mem} is the set of all members of all RIRs, ERX is the set of all legacy ERX transfers, and $SubAlloc$ is the set of all suballocations. This addition clearly does not include any sub-suballocations. Indeed, in [10], on slides 12 and 13, the authors show measurements that (in particular) LACNIC has over an order of magnitude *more* sub-suballocations than suballocations. We, therefore, treat this measure as a pronounced lower bound. Today there are $3,370 + 4,277 + 2,705 + 1,276 + 8,618 = 20,246$ members of RIRs [4, 5, 3, 11, 14], since the number of ERX allocations is hard to know[2] we estimate it at 10% of the RIR members 2,025, and with 61,505 suballocations [9], we get a total count of $|\widehat{InfraObjs}| = 20,246 + 2,025 + 61,505 = 83,776$.

2.2 Estimating the Number of Objects

Next we consider $\widehat{AllocObjs}$. There are many cases in which an organization will need multiple o^{ROA} objects. For example, each routed allocation needs at least one ROA, and also if an organization wants to announce a prefix from multiple origins (MOAS), etc. It is for this reason that $\widehat{AllocObjs}$ will be loosely related to the way in which allocations are routed. We use this intuition to estimate that the number of allocation objects could roughly match an optimized view of the routing system today (i.e. a view in which announcements were maximally aggregated). For this, we use the origin report [9]. This site can be used to collapse the routing table entries down to the smallest set possible, and results in 273,592, or $\widehat{AllocObjs} = 273,592$. We note that in [10], on slide 9, the authors calculate that RIPE alone would result in 100,000 ROA objects. Rather than use that inflated estimate and invent an arbitrary multiplier (and to avoid any potential systemic bias), we choose the smaller and more systematic measure that is based on the routing table.

As mentioned earlier, the total number of eBGP speaking routers throughout the Internet was estimated as roughly 23.8 per ASN. We conservatively double this (to represent standby keys) as advised by a lead on the BGPsec design team [6]. This leads to $\widehat{RoutingObjs} = 2,000,000$.

Finally, the set of $\widehat{StructObjs}$ depends entirely on the RPKI deployment. If there are just a few flat repositories, then $\widehat{StructObjs}$ would be very small. If, on the other hand, every organization were to run their own repository, this number would grow to $|\widehat{StructObjs}| = |ORG|$. As a result, we use this as an independent variable in Section 3 when calculating object counts.

With the above, Table 2 maps our object types to size estimates based on the size of the Internet today. Therefore, we estimate that

$$O^{Total} = 100 + 251,328 + 273,592 + 2,000,040 + 83,776 + |\widehat{StructObjs}| = 2,608,836 + |\widehat{StructObjs}|$$

[2] depends on how many of an RIR's members have historical resources or resources transferred to them from another RIR.

Repository	Avg sync time	Avg num objects	Avg seconds / object
"An ISP"	3.14	3	1.05
"RIR"	945.96	234	4.04
ARIN Pilot	83.42	131.46	0.635
APNIC	1006.95	1405	0.717
RIPE	2269.75	3800	0.597
RIPE	2065.41	3785	0.546
RIPE	2004.8	4066.32	0.493
"rpki101.fra2.de.euro-transit.net"	4.61	5	0.922
APNIC	944.56	1469.14	0.643
AfriNIC	75.45	82	0.920
Aggregate	**9404.05**	**14980.92**	**0.628**

Table 3: Repo sync stats: Average: 0.628 seconds/object

Number of Repositories ($\|\widehat{StructObjs}\|$)	Number of Objects (no router c^{EE})	Time to Gather	Days to Gather
5	608,801	382,330	4.42512
10	608,806	382,336	4.42519
100	608,896	382,449	4.42650
1,000	609,796	383,580	4.43958
10,000	618,796	394,884	4.57042
42,000	650,796	435,076	5.03560

Table 4: Gathering times as the number of repositories grows (without router certs).

3 Evaluation

While 2.6 million objects could be considered to be a large corpus (especially as a lower bound estimate), it will certainly grow whenever any keys are being rolled over and changed, it will necessarily almost double during cryptographic algorithm rollovers, and it doesn't include those objects needed for the actual allocation hierarchy. So, again, we submit that this is a lower bound.

Next, to estimate the amount of *time* it takes for caches to actually gather a fully deployed RPKI (at today's Internet's scale), let's examine the algorithm. We consider the set of interconnected repositories to be a directed graph, in which the set of vertices V are discovered by gathering (i.e. an repository contains certificates who have SIA and AIA URI pointers to potentially external repositories), and to traverse all of it, we use a depth-first search approach. The complexity of this approach is well known to be: $O(|V| + |E|)$, where V is the set of vertices, and E is the set of edges.

In the RPKI, V is the set of repositories, and E is the set of unique pairs of vertices in V that come from certificates in each repository and which point to external SIA or AIA URIs (for example, certificates chaining to their CAs). From this understanding, we need to estimate how long it would actually take to follow this algorithm (with the above set of objects) in order for a cache to discover all of the repositories in the Internet (on each running). To map this complexity to actual deployment numbers, we consider the presentation [1]. In this presentation, we see 10 different performance graphs for RPKI repositories. These graphs each explicitly show the average number of objects and the average sync time. Table 3 lists these values and the computed aggregate statistics across all of them.

With these numbers, Table 4 lists estimates for how long it would take to download from a varying number of repositories containing objects for the Internet, if RPKI were *fully* deployed, today. We note that in our evaluation, we should have modeled a latency factor for traversing edges in E (for example, DNS takes time, timeouts, etc.). However, to continue to focus on a lower bound, we ignored the E cost in our calculations.

Figure 1 illustrates that as the number of repositories grows, so too does the sync time for the system. In addition to these numbers, [1] appears to analyze the caching behavior of a large ISP with an internal hierarchical caching system. This would mean that after a master cache (for a single ISP) has gathered all of the objects

Number of Repositories ($\|\widehat{StructObjs}\|$)	Number of Objects (with router c^{EE})	Time to Gather	Days to Gather
5	2,608,841	1,638,355	18.96245
10	2,608,846	1,638,362	18.96252
100	2,608,936	1,638,475	18.96383
1,000	2,609,836	1,639,605	18.97691
10,000	2,618,836	1,650,909	19.10774
42,000	2,650,836	1,691,101	19.57293

Table 5: Gathering times as the number of repositories grows (*with router certs*).

Figure 1: Between 4.3 and 5 *days* to sync (19 to 20 with router certs). That means that operators have to count on 9 to 10 *days* (or 38 to 40 with router certs) after making a change that they want RPs to pick up (twice the polling period).

from across the global RPKI, it must begin replicating this to other internal "tiers" of caches (inside the ISP). Therefore, for such ISPs, we must also add the numbers from [1] to this (as internal caches must replicate). In [1], the replication time was between 70 and 80 *minutes*. However, while the authors describe their internal caching testbed as being, "fairly large scale," their simulated RPKI deployment only had 14,000 objects (vs. our estimated 2.6 million, in Table 4). As a result, one might worry that with strictly linear scaling, this replication time for internal caches could become $70 \times \frac{2,650,836}{14,000} = 13,254.18$ more minutes (or 9.204292 more days). This could make estimates inflate to $19.51868 + 9.178375 = 28.697055$ *days*. Admittedly, each ISP may choose different caching strategies, but this hierarchical model provides a lot of operational robustness, though it complicates convergence time significantly. One final note, the above analysis does not add in time for potential system failures, latency as load increases in repository systems, etc.

3.1 Deadline Crawling

A repository system that relies on RPs having a consistent/fresh view of *all* objects in the entire global repository before validation can be assured begins with an architectural handicap. Recall that in the RPKI, caches must fully enumerate, and known, and support O^{Total} objects, spread out across $|\widehat{StructObjs}|$ repositories. If there

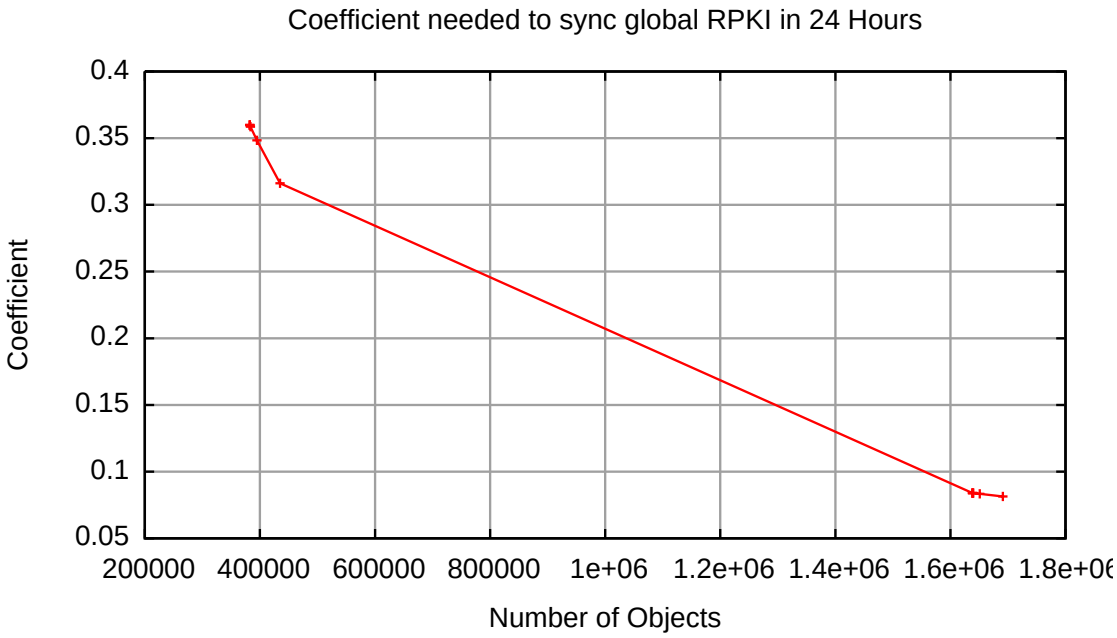
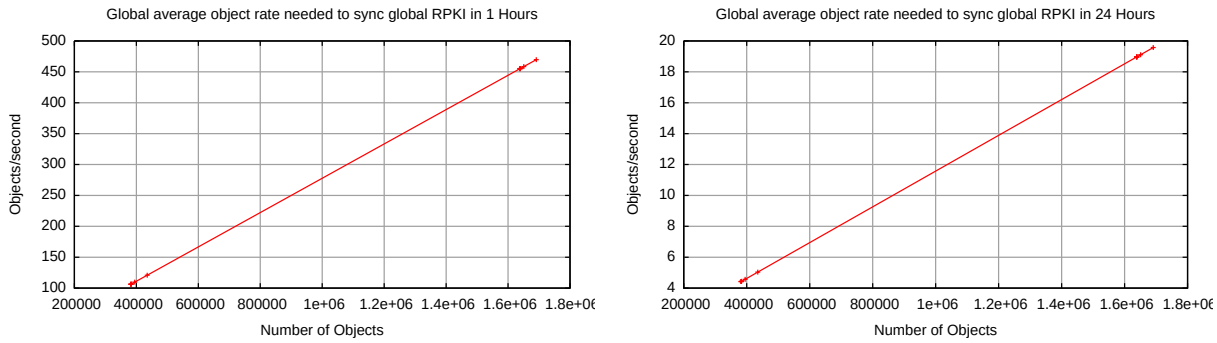


Figure 2: This Figure illustrates the coefficient needed to accelerate the downloading of RPKI objects as the repository grows.



(a) This Figure illustrates the required average object/second rate required to meet a global deadline of 1 hour.

(b) This Figure illustrates the required average object/second rate required to meet a global deadline of 1 day.

Figure 3: This Figure illustrates the the required average object/second rate required to meet a global deadlines.

becomes a requirement to provide a certain degree of freshness (i.e. caches must be able to learn new objects in less than t seconds), then we must use the above formulation to drive requirements on the synchronization time.

That is, if there are $|O^{Total}|$ objects in the RPKI, then if our deadline is t, we can determine how long synchronizing them must take. The above calculations assume a linear fetch time as object counts grow. On the other hand, if we consider a deadline of t, and time coefficient of $z(t, n)$ (where n is an object count, and t is the deadline) for the objects/second synchronization rate r, then we can cast the overall synchronization time as:

$$t = (|O^{Total}| \times r) \times z(t, |O^{Total}|)$$

If we fix our synchronization time (deadline), we can solve for $z(t, n)$ as:

$$z(t, |O^{Total}|) = \frac{t}{|O^{Total}| \times r}$$

Thus, if we (for example) want to set a deadline of one day (86,400 seconds), we would need a coefficient of: $\frac{86{,}400}{2{,}650{,}836 \times 0.628} = 0.0519$. Using the measurements from Tables 4 and 5, Figure 2 shows the value of the coefficient needed to meet the deadline of *one day*. That is, what the sublinear scaling would need to look like in order to have a global system pick up changes in one day. Based on caching, this could actually be double, but for simplicity, we leave the measurements as presented.

Switching this around, if the goal is to have a full synchronization of the global RPKI in one hour (such as might be needed by DDoS service providers [13]), then Figure 3(a) shows the global average transfer rate required for a deadline of 1 hour. Figure 3(b) shows the global average transfer rate required for a deadline of 1 day.

4 Discussion

So far, we have dealt with the size of the database of objects required to build a complete RPKI, if the system is in a stable state. Many factors will require regular updates to the objects discussed so far, including key rollover and duplication of objects for systemic resiliency. The rate at which key rollovers occur within the system is undetermined at this time, so it is difficult to know precisely how key rollovers will impact system scaling. Duplication of information [8] for system resiliency, however, is likely to be fairly common, and therefore worth investigating.

5 Conclusion

The calculations in this document, while just candidate representations of RPKI, project a very long lower-bound on the potential time needed for RP caches to gather all objects in the RPKI. Furthermore, when repositories in the RPKI undergo key rollovers, repositories *and caches* must store old and new objects for about a month (the polling period + potential cache replication times) before they can safely expunge them. This, in turn, will directly inflate the above numbers, and cause longer delays.

For example, if an algorithm is rolled over [7], the system will have roughly twice as many objects as it normally does, because every signed object in the entire global RPKI will necessarily have to be signed with an outdated algorithm, and then again with a new one. Moreover, this state is envisioned to persist for on the order of "months or years" at a time. This may almost double the sync time, and cause the RPKI to be even *less* responsiveness to key rollovers (possibly almost twice the 30 day estimate).

Also, if rsync scales sublinearly, then it is likely that the actual time needed to sync will grow beyond these conservative estimates. Again, even if rsync periodically has to restart because objects being transferred have changed [2], this underestimate will get worse. In addition to this, active CRLs will *not* (by design) include expired data. Therefore, as soon as operators (following advice in existing drafts) begin to use expired data, Relying Parties (RPs) will necessarily be forced to keep track of old data everywhere (thus increasing the object count, again).

However, as we saw in Section 3, if we consider deadlines for global synchronization, we find that object synchronization rates must be *superlinear* (not sublinear). That is, the system must get faster as it gets more loaded. What is worrisome about these findings is that even the modest goal of 24-hour synchronization times would require a sublinear scaling coefficient (and thereby superlinear performance), which would need to scale roughly two orders of magnitude to meet today's scale. One reason to begin considering deadline scheduling was illustrated during mail list discussions [13].

The participants of this email thread outlined the need to have tangible deadlines for the routing system to be able to reflect changes in routed origins. The explanation focused on the service provider DDoS market, in which new customers that are being impacted by DDoS attacks may ask service providers to protect them through new advertisements. Today, this can be done is minutes (and normal routing can be returned to customers in minutes too), and that is a business-case deadline.

Prior to this document, only publications such as [1] reported scaling measurement. While this presentation (and its relative incarnations) purport to be "fairly large scale" they appear to have estimated object counts roughly two orders of magnitude too small. We mention this simply to motivate the importance of having a living document such as this one. Finally, this analysis does not even take into account churn or systemic dependencies. Such issues should be discussed in similar (but separate) technical notes.

References

[1] RPKI Propagation. In *NANOG 56*, October 2012. http://www.nanog.org/meetings/nanog56/abstracts.php?pt=MjAwMCZuYW5vZzU2&nm=nanog56.

[2] Validation of RPKI objects using a local cache. In *IETF 85, SIDR WG*, November 2012. http://www.ietf.org/proceedings/85/slides/slides-85-sidr-7.pdf.

[3] AfriNIC. AfriNIC Membership Stats. http://www.apnic.net/__data/assets/pdf_file/0006/52557/2012-09-20-ecminutes.pdf.

[4] ANIC. APNIC EC Meeting Minutes. http://www.apnic.net/__data/assets/pdf_file/0006/52557/2012-09-20-ecminutes.pdf.

[5] ARIN. Membership. https://www.arin.net/about_us/membership/index.html.

[6] R. Bush. [sidr] Scaling properties of caching in a globally deployed RPKI / BGPSEC system. http://www.ietf.org/mail-archive/web/sidr/current/msg05351.html.

[7] R. Gagliano, S. Kent, and S. Turner. Algorithm agility procedure for rpki, draft-ietf-sidr-algorithm-agility.

[8] R. Gagliano, T. Manderson, and C. Martinez. Multiple repository publication points support in the resource public key infrastructure (rpki) draft-rogaglia-sidr-multiple-publication-points-01, October 2012.

[9] ICANN. Orign Report. http://stats.research.icann.org/bgp/cidr-map/origin-map.bgp.20121129.1200.html.

[10] O. Kim, K. Sriram, O. Borchert, and D. Montgomery. Characterization of the "size and shape" of static rpki. December 2010. http://www.nist.gov/itl/antd/upload/RPKI-size-shape-modeling.pdf.

[11] LACNIC. LACNIC Membership. http://www.lacnic.net/documents/10834/17013/LACNIC-listadeMiembros_11-2012.pdf/ed420a71-bfcc-41ca-9d32-5270727a0ea9.

[12] M. Lepinski and S. Kent. An Infrastructure to Support Secure Internet Routing. RFC 6480, Feb. 2012.

[13] D. McPherson. [sidr] Scaling properties of caching in a globally deployed RPKI / BGPSEC system. http://www.ietf.org/mail-archive/web/sidr/current/msg05394.html.

[14] R. NCC. RIPE NCC Statistics. https://labs.ripe.net/statistics/?tags=lirs.

