CCN-KRS: A Key Resolution Service for CCN

Priya Mahadevan[1] Ersin Uzun[1] Spencer Sevilla[2] J. J. Garcia-Luna-Aceves[1,2]
[1]Palo Alto Research Center, Palo Alto, CA 94304
[2]Computer Engineering Department, University of California, Santa Cruz, CA 95064
{priya.mahadevan, ersin.uzun}@parc.com, {spencer, jj}@soe.ucsc.edu

ABSTRACT

A key feature of the Content Centric Networking (CCN) architecture is the requirement for each piece of content to be individually signed by its publisher. Thus, CCN should, in principle, be immune to distributing fake content. However, in practice, the network cannot easily detect and drop fake content as the trust context (i.e., the public keys that need to be trusted for verifying the content signature) is an application-dependent concept. CCN provides mechanisms for consumers to request a piece of content restricted by its signer's public key or the cryptographic digest of the content object to avoid receiving fake content. However, it does not provide any mechanisms to learn this critical information prior to requesting the content.

In this paper, we introduce a scalable Key Resolution Service (KRS) that can securely store and serve security information (e.g., public key certificates of publishers) for a namespace in CCN. We implement KRS as a service for CCN in ndnSIM, a ns-3 module, and discuss and evaluate such a distributed service. We demonstrate the feasibility and scalability of our design via simulations driven by real-traffic traces.

Categories and Subject Descriptors

C.2.0 [**General**]: Security and protection;
C.2.1 [**Network Architecture and Design**]: Network communications;
C.2.6 [**Network Protocols**]: Protocol architecture

General Terms

Security, Design, Performance

Keywords

Information-centric networking; security

ICN'14, September 24–26, 2014, Paris, France.
Copyright 2014 ACM 978-1-4503-3206-4/14/09 ...$15.00.
http://dx.doi.org/10.1145/2660129.2660154 .

1. INTRODUCTION AND MOTIVATION

The way in which users and applications use the Internet has changed dramatically since its early days. The shift from a few thousand people using it to access shared computing resources to billions of people world-wide running a wide range of applications has exposed several limitations of the current Internet design. A number of recent research efforts are focused on designing Information-Centric Networking (ICN) architectures [1] that emphasize efficient and scalable content distribution via named information (or content). Content Centric Networking (CCN) [2], is one of the ICN architectures that has been widely experimented with in the networking community.

One of the main tenets of CCN is to name content, instead of communication end-points. Users interested in a piece of content ask for it directly by name by sending an Interest packet. An Interest packet is forwarded by CCN nodes in the network towards a content source, until it reaches a node that can respond with a matching piece of content (called a Content Object) whose name matches the name stated in the Interest packet. The Content Object is sent back to the requesting user, and can optionally be cached by relay nodes between the node responding to the Interest packet and the consumer that requested it. Hence, a CCN router can respond to an Interest packet it receives with a matching cached Content Object. CCN requires every Content Object to be cryptographically signed by its publisher. Thus, globally addressable and routable content can be authenticated by anyone requesting the content. CCN entities that request content are called *consumers* and they always verify content signatures in order to assert:

- *Integrity* – A valid signature (computed over a content hash) guarantees that the signed content is intact.
- *Origin Authentication* – Anyone can verify whether content originates with its claimed publisher, because a signature is bound to the public key of the signer.
- *Correctness* – A consumer can determine whether delivered content corresponds to what was requested because a signature binds the content name to its payload.

A consumer, having its own trust context and policy, can easily differentiate a genuine piece of content from a fake one (i.e, content signed by unauthorized/untrusted keys) after verifying its signature (and potentially a chain of certificates to reach to a trust anchor). However, it is a challenge for an intermediary node in the network to differentiate a genuine piece of content from a fake one since applications (and the namespaces they operate on) are not likely to adhere to a uniform trust model. It is practically infeasible for a CCN router to maintain an up-to-date global knowledge of such mappings considering that applications themselves are constantly evolving and new applications are being developed everyday. In other words, CCN needs to operate efficiently even in the presence of fake content in router caches.

CCN provides two mechanisms to mitigate the above problem and enforce a consumer's trust preferences at the network layer. While requesting named content through Interest packets, consumers can optionally include which key is acceptable as the signer for the requested content or specify the cryptographic digest of the content [details in Section 2]. The immediate requirement for consumers to use these mechanisms is that they must either (1) have the publisher's public key before issuing an Interest for any Content Object from that publisher, or (2) know the digest of the Content Object before issuing an Interest for it. However, CCN does not provide any solution to this proverbial "chicken-and-egg" problem.

In this paper, we propose a Key Resolution Service (KRS) for CCN. KRS is a service that maps a CCN content name, such as "/parc/papers/krs.pdf", to a set of corresponding security information, such as the public key certificate of the authorized publisher for the namespace and/or the cryptographic digest of the Content Object. By querying KRS before issuing an Interest for a piece of content, consumers can acquire the necessary security context required to activate the trust enforcement mechanisms at the network layer. In other words, KRS allows consumers in CCN to learn and specify the public key of a publisher or the content digest they would trust in an Interest. As long as the routers in the network enforce such restrictions, this technique effectively guarantees delivery of trustworthy content regardless of the number of fake Content Objects carrying the same name that might be cached anywhere in the network.

Given that secure acquisition of keys prior to requesting content is required for CCN to be robust against content-poising attacks [3, 4], our KRS design (in combination with the proposal by Ghali [4] requiring either the publisher public key digest (`PPKD`) or the `content-digest` field be included in every Interest) constitutes the first practical solution to mitigate denial of service attacks via content poisoning/spoofing in CCN.

The rest of the paper is organized as follows: Section 2 gives an overview of the CCN architecture. Section 3 outlines the set of requirements that KRS must meet. Section 4 provides an overview of the KRS architecture and operation. Section 5 describes our implementation and evaluation methodology. Section 6 presents KRS performance results for several different scenarios. Section 7 concludes the paper.

2. CCN OVERVIEW

All communication in CCN is via two distinct types of packets: *Interests* and *Content Objects* (CO). Both of these packets carry a hierarchically structured name that uniquely identifies a piece of content. A consumer interested in a piece of content requests it by sending an Interest packet that carries the name of the desired piece of content. CCN routers use the name in the Interest to forward it towards likely sources of data by performing a longest prefix match on the name in their forwarding table. A CO whose name exactly matches that in the Interest is sent back to the requesting consumer. Only Interests are routed in CCN. COs take the reverse of the path taken by the corresponding Interest packet. Any intermediate router can choose to cache any CO that it receives. If a router receives an Interest packet for a CO that it already has in its cache, it can respond with the matching cached CO. Since CCN consumers can receive content from any node, including from caches of intermediate nodes, it is imperative that they verify the authenticity of the CO they receive.

To verify the authenticity of a CO, a CO contains the following fields, in addition to the name of the content and the data:

- PublisherPublicKeyDigest (PPKD): The digest of the public key required to verify the signature. It is typically a SHA-256 digest.
- KeyLocator: The public key or certificate required to verify the signature.
- Content-digest: The cryptographic digest of the CO.
- Signature: A public-key signature generated by the publisher covering the entire content.

An Interest packet in CCN can also include a PPKD, thus allowing consumers to explicitly specify the publisher's public key digest. If this entry is present in an Interest, a matching CO must have the same digest in its PPKD field and a valid signature. CCN Interests can also include `content-digest` to request a unique CO. To make sure that a consumer only receives content that it would trust (i.e., prevent denial of service attacks by content/cache poisoning), it should issue all its Interests with at least one of the PPKD or `content-digest` restriction set. However, as long as a consumer device/application doesn't come preloaded with that information, it needs a mechanism to be able fetch this security context before sending an Interest for any content.

To solve the above need, we envision a global CCN Key Resolution Service (KRS) that allows consumers to resolve content names to publisher public key and/or the content digest. KRS is loosely analogous to DNS – just as the DNS today allows clients to resolve hostnames to IP addresses, KRS allows consumers to resolve content names to relevant security information.

3. KRS REQUIREMENTS

We envision KRS as a distributed key-value store service, where the key is the content name (or name prefix) and the value is the KRS record that contains the necessary security information required to verify content authenticity. We first describe the requirements that such a global service must satisfy:

Security: KRS itself must be secure. Only valid content publishers must be able to add, delete and modify entries. Consumers and publishers also need to securely obtain root public keys for KRS. When a consumer queries the KRS and receives a response back, the consumer must be able to independently verify the received KRS record.

Scalability: Studies [5, 6] estimated the number of unique webpages to be of the order of 10^{12} in 2011. This number is expected to grow significantly over the next few years. Since we envision KRS to be globally deployed and widely used, it should be able to support and resolve up to 10^{14} content names.

Response Time: KRS must exhibit similar performance as DNS and the response time of KRS to consumer requests should be similar to the response time of DNS.

Flexibility: KRS should be able to resolve a content name to one of: (i) content hash (ii) publisher public key certificate or (iii) public key certificate chain. It must be flexible enough to include any other information that might be deemed necessary in the future.

Seamless Application Support: KRS must operate transparently in a manner similar to DNS today. Rather than requiring CCN applications to programmatically interact with and manage KRS individually, applications should ideally integrate with consumer devices at the system level, similar to how DNS operates in the Internet today.

Discovery: KRS must be discoverable by client devices, just as local DNS servers are discoverable by devices today. For example, when a device joins a WiFi network at a coffee shop, the device has no *a priori* information regarding the network topology, but must still be able to discover the name of a nearby KRS server.

4. KRS DESIGN OVERVIEW

We designed KRS as a global service that runs over CCN to address the requirements described in Section 3. All communication among KRS servers is via CCN Interests and COs. Consumers send requests specifying the name of the content they would like to resolve in the form of an Interest to KRS, and the service returns the KRS record containing the security information for that content name in the form of a CO.

Our KRS design is influenced by that of DNS, given the similarities in requirements between KRS and DNS. We split KRS into two interacting components: *local* KRS servers that receive requests from CCN consumers, and *authoritative* KRS servers that store and manage KRS records. We illustrate the operation of KRS in Figure 1.

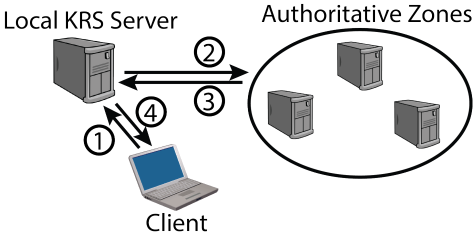

Figure 1: Client Querying KRS

As stated previously, CCN names are hierarchical. We exploit this hierarchy in content names to assign a KRS zone to manage a name prefix. A KRS zone, comprised of one or more authoritative servers, is responsible for storing all KRS records associated with that name prefix.

KRS organizes zones into a hierarchy similar to DNS. We achieve scalability by distributing the storage and management of all associated KRS records to individual KRS zones. An authoritative server in a KRS zone may recruit additional servers at any time to handle the storage and serving of KRS records for the name prefix that it is managing; it can also add a new zone to handle a new subprefix of the content name prefix that it is managing.

Local KRS servers are responsible for receiving and forwarding requests from CCN consumers to the appropriate KRS zone authoritative server. To this end, the main data structure at any KRS server is the "next-hop" table that stores the name and the public key of the KRS service that either stores the KRS record for that content name or knows the name of the next-hop service to whom the KRS request can be forwarded.

Local KRS servers might be discovered by clients either via a secure automatic configuration service (i.e., a CCN equivalent of DHCP with server authentication) or via a predictable (or predetermined) name prefix for the local KRS service such as "/krs." For the latter, CCN nodes in a given network would need to have appropriate entries to forward the Interests to the closest local KRS server.

For example, a client wishing to obtain the security information for the content name `/parc/papers/krs.pdf` can create a CCN Interest and ask for the KRS record for that name using its local KRS service's prefix (`/krs` in this case), resulting in the name `/krs/q=/parc/papers/krs.pdf` in the Interest.[1]

On receiving this Interest, the local KRS server performs a look-up in its next-hop table to determine the name of the KRS zone service that can handle the KRS query. In the absence of an entry corresponding to the zone service name, the local KRS server queries the root KRS service to obtain the globally routable name of the responsible KRS zone service (e.g., `/parc-krs`). The local KRS server uses this globally routable prefix to forward the query over to the service responsible for that KRS zone. Upon receipt of this Interest, the authoritative KRS zone service extracts the content name for which the KRS record has been requested and compares it against the records it stores to retrieve either (1) the final response to the query in the form of a KRS record, (2) the next zone service that the query should be forwarded to, or (3) the message that no such record exists.

In the case of (1) or (3) above, the authoritative server responds to the received Interest by encapsulating the appropriate KRS record as a CCN CO with the same name as in the received Interest. To handle case (2), the KRS server performs a lookup in its next-hop table and creates a query, in the form of an Interest for that KRS zone, which should know more about the namespace that is being queried. This process repeats until either case (1) or case (3) above is performed by an authoritative service. Although we described the process as interactive, we emphasize that all steps in the process can be completed with previous responses to the same query cached either at CCN nodes as COs at the network layer or cached KRS records for popular or recent queries in KRS servers at the application layer.

Once the local KRS server receives a CO encapsulating the requested KRS record, it first decapsulates the record and re-encapsulates it in a new CO that would satisfy the Interest received from the consumer.

Figure 2 depicts the various fields in a KRS record. The name in the KRS record is the name (or name prefix) that is being resolved, and the payload is the security information associated with that name (i.e., a content-hash or public key). Further, each KRS record is individually secured with

[1] Most recent CCNx specification (CCNx 1.0) allows Interests to have a payload. In future implementations, the query part of the name can be carried as payload for shorter names and better overall PIT usage efficiency.

a cryptographic signature and carries a public key certificate or certificate chain for the signing key. A consumer receiving a KRS record would only trust it if the signature on the record is valid and the certificate chain for the signing key anchors at a trusted entity (e.g., a global certificate authority). As stated previously, each KRS record is transmitted as a CCN CO, which includes a signature that can be used to authenticate the KRS service itself and a PPKD field that can be used to limit acceptable responses to an Interest carrying a KRS query to trusted KRS service instances only.

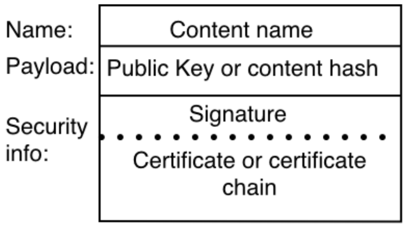

Figure 2: KRS Record

Having provided an overview of the KRS architecture, we next address how we meet each of the requirements listed in Section 3.

4.1 Improving Scalability

Scalability in KRS is attained through the use of a longest-prefix-matching (LPM) algorithm used for lookups in the next-hop table and for retrieving the stored KRS record itself. Using LPM to look up entries in the next-hop table significantly cuts down on KRS recursive referrals, thereby improving KRS response time. LPM also enables the use of "default" records, thereby significantly reducing the burden on KRS to store records for every single content item published.

As an example, a publisher may have one public-private key pair which it uses to sign all content published with the prefix `/parc/papers`. Thus, rather than storing the same key as a separate KRS record for each piece of published content under the prefix `/parc/papers`, KRS simply stores the key once in the authoritative zone `/parc`, with the notation "*" that this record is the final response for any name including the prefix `/parc/papers`. LPM is enabled through the simple designation of a "*" record for a prefix, which indicates that there are no further KRS records below that prefix.

4.2 Improving KRS Performance with Caching

KRS records can be cached in two separate ways: nodes in the underlying CCN network may cache the COs (that encapsulate a KRS record) and return them for subsequently expressed Interests, and KRS servers may *also* themselves cache and return KRS records. To highlight the distinction between these two forms of caching, consider an example where a local KRS server receives a request to resolve the content name "/parc/csl/papers/krs.pdf". In the process of recursively resolving this request, the local KRS server may receive and cache the KRS record for the name prefix "/parc/csl/papers". Subsequently, the local KRS server receives a request to resolve "/parc/csl/papers/paper2.pdf". Though these two requests share the prefix "/parc/csl/papers", CCN caching cannot provide any benefits, since CCN caches use exact matching on names to return a matching CO and the last component of the two prefixes are dissimilar[2]. Instead, KRS servers perform LPM on the requested content name "/parc/csl/papers/paper2.pdf" and respond with the cached KRS record for "/parc/csl/papers", thereby improving performance significantly.

4.3 KRS Forwarding Policies

So far, we have described a recursive forwarding policy employed by authoritative KRS servers, whereby they directly query the next-hop service in charge of the subprefix. An alternative to the recursive technique is the iterative forwarding technique. Here, the server simply responds to the Interest with a CO that contains the globally routable name and the public key of the next-hop service able to process the KRS request. When the requesting server receives this CO, it constructs a new KRS query for the newly learnt KRS service name. The requesting server can also optionally cache this globally routable name and the public key of this KRS service in its next-hop table.

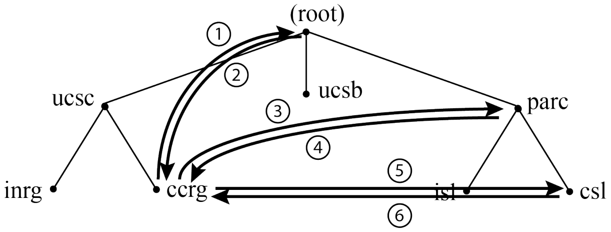

Figure 3: Top-Down Forwarding

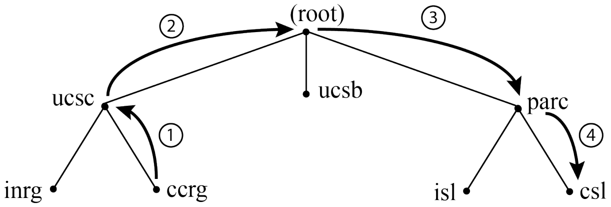

Figure 4: Bottom-Up Forwarding

In DNS, local resolvers come pre-loaded with the addresses of root and top level domain (TLD) servers. This technique bootstraps the resolution process and ensures that a resolver always has an address to send its first query to, even if it has no immediate forwarding information for the query. We refer to this model, illustrated in Figure 3, as *top-down* forwarding. We use an analogous technique in KRS to bootstrap the next-hop table entries at KRS servers. KRS may also employ *bottom-up* forwarding, wherein all local resolvers have a prefix name, and therefore a place in the prefix hierarchy. When a local resolver receives a query for which it

[2] Unlike previous CCN protocol specifications, the current specification CCNx 1.0 requires CCN caches to perform exact matching on names.

has no immediate forwarding information, instead of directly querying the root of the prefix tree, it sends the query to its parent, as illustrated in Figure 4. We examined the tradeoffs between top-down and bottom-up forwarding in [7]. For example, bottom-up forwarding serves to keep local requests more local, yet can potentially result in more referrals, since it introduces more intermediate zones along the path. However, well-designed caching policies help to ameliorate these referrals while simultaneously reducing load on higher-level authoritative servers.

4.4 Bootstrapping and KRS Maintanence

In the previous section, we described how queries are forwarded through the KRS and answered by the authoritative zone. However, before a query is answered, the KRS must be populated with records, and these records must be kept up-to-date. Among the multiple ways of achieving this, we advocate a federated approach similar to the one deployed in DNS today to handle globally routable namespaces and corresponding key registrations. Like in DNS, this registration information can be maintained by the registries, which contract with registrars to provide registration services to the publishers. A publisher can select a designated registrar for the namespace it chooses to own. Unlike DNS, we expect most registrars in KRS to also act as certification authorities that can issue public key certificates to publishers for top level namespaces they own. As in DNS, only the designated registrar may modify or delete information about a KRS record for a namespace and we also envision a global entity, similar to ICANN, that will coordinate the namespace allocations and manage the KRS root servers.

One difference between today's DNS and KRS will be the number of top level zones in the root. We expect this number to be significantly larger for KRS and the contractual arrangements for zone file access may not scale. However, it is expected that DNS will experience the same problem in the near future as the number of global top-level domains increase. The ICANN Zone File Access Advisory Group has published a concept paper [8] with four new access models to alleviate this problem, all of which are equally applicable to the KRS system proposed here. We omit a discussion about these models due to space restrictions in this paper but refer interested readers to [8] for more details.

Similar to DNSSEC [9], each KRS record itself is cryptographically signed for security purposes. However, unlike DNSSEC, KRS records are signed by the publishers that owns the namespace to authenticate the KRS record at the consumer application. Additionally, KRS queries are always answered in the form of CCN COs, and independent from any KRS record they may be carrying, COs are signed by the KRS server that generated it and these signatures are used for authentication at the KRS protocol level.

5. KRS IMPLEMENTATION AND EVALUATION METHODOLOGY

We implemented KRS as an application level service running over ndnSIM [10], which is a ns-3 [11] module. We used simulations and trace-based analysis to evaluate the performance of KRS under a varying range of system parameters.

We first provide an overview of our evaluation methodology. In our ns-3 simulations, we input a realistic Internet topology as the underlying CCN topology with KRS running as an overlay application on this topology. We chose nodes in this topology to represent both local KRS servers and authoritative servers in KRS each zone. As part of the initialization process, KRS records are distributed across respective zones and handled by the designated authoritative server. These records are drawn from representative traffic traces. Client requests that need to be resolved are processed by local KRS servers and matching KRS records are fetched as discussed in Section 4.

At each local KRS server, we quantified the overhead as measured by the number of KRS messages that are required to fetch the matching KRS record. Each referral to the next zone results in two KRS messages – an Interest message and the returning record CO. We also measure the system latency at the local KRS servers as the time required for the local KRS server to obtain the record after it receives the resolution query.

For all the above evaluations, we experimented with a range of KRS specific parameters such as the forwarding scheme used and cache size. All reported KRS latency values represent network latency; we do not include the time required to perform the optional record authenticity verification operation at each KRS server. By default, we used recursive top-down forwarding in all our experiments, except when comparing the benefits of recursive versus iterative forwarding.

Similar to any global distributed system, KRS performance results are dependent on the system and network environment parameters, as well as the workload used to drive the performance evaluation. In the rest of this section, we describe our workload as well as results from our parameter sensitivity analysis that we used in our KRS simulation experiments.

5.1 Representative Workload

As CCN is an emerging network architecture, there are no representative traffic traces currently available. We chose the well-accepted technique of translating HTTP GET requests to CCN Interests [12], and thus ensured that the heirarchy in HTTP URLs directly mapped to the hierarchy in CCN names. Our workload serves two purposes: (i) it determines the records *stored* in KRS and the KRS zone hierarchy and (ii) it represents the set of content names that clients request to be resolved.

We used the IRC trace[3] [13] consisting of roughly five hundred thousand individual HTTP GETs.We removed requests to CDNs and shared hosting services as these URL structures did not translate well to CCN name hierarchies and our resulting trace consisted of approximately 350,000 HTTP GETs. We translated each HTTP GET to a corresponding CCN name by reversing the order of the hostname and replacing each "." with a "/". Thus, the HTTP object "mail.google.com/bob" corresponded to the CCN CO name "/com/google/mail/bob". We aggregated prefixes that span multiple ccTLDs as they all refer to the same CO. Thus, "/au/com/google", "/uk/co/google", and "/com/google" were aggregated to "/com/google".

We plot the distribution of client requests in our traces in Figure 5 and confirm that the requests follow the Zipf distribution, with $alpha = 0.9258$. This result is important

[3] collected at FIX-West over two days of 2009 at Ames Research Center in Mountain View, California

since it implies that a majority of the queries to KRS are for resolving the same content name.

Next, we examined the number of components in each name that needed to be resolved. We plot the distribution of the number of components in each name (black solid line) in Figure 6, with a mean of 7.36 and a standard deviation of 1.68. Since the distribution in Figure 6 is collected over the set of all *names*, it is implicitly weighted by the popularity of the requested COs. We filtered multiple resolution requests for the same CO to show the distribution of unique prefixes in Figure 6 (red solid line), with a mean of 7.48 and a standard deviation of 1.64. This distribution influences the depth of the KRS tree (zone) hierarchy and the number of referrals that need to be performed, thereby impacting KRS performance.

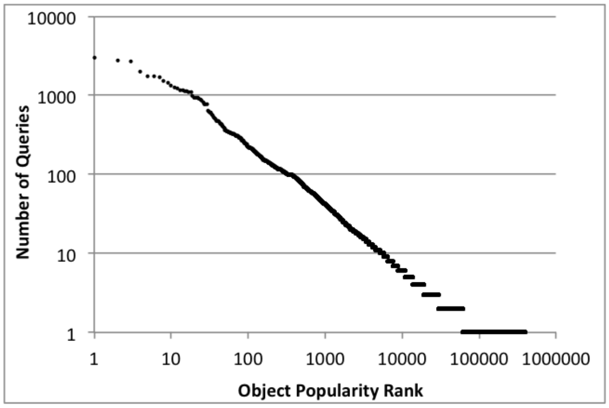

Figure 5: Popularity of COs

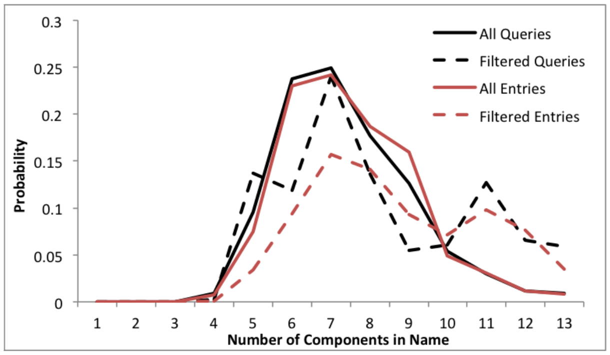

Figure 6: PDF of Name-Components in a Resolution Request

5.1.1 *Domain-based Filtering*

In our trace set, we observed 29932 unique second-level domains (e.g. "google", "yahoo", etc); Figure 7 illustrates the popularity distribution of these domains, with the Y-axis counting all requests for the domain. This distribution is *also* Zipf, with $alpha = 0.8640$ implying that a small number of domains are responsible for a significant portion of the total number of requests. Scalability considerations in the topology we used for running our simulations required us to narrow the request set to include a much reduced number of second-level domains, but at the same time we ensured that the filtered set is still representative of the original requests. We reduced our request set down to the 5 most popular second-level domains ("google", "yahoo", "msn", "friendster", and "cnn"), which resulted in a total of 113,531 requests expressed for 34,834 different COs. This set of domains also provided a diverse spread that captured many different Web use-cases such as search, homepage, social, and news.

To show that this filtered set is still representative of the observed workload, we calculated the probability density function for both the weighted and unweighted name-components in our filtered set. We depict these by the black and red dashed lines respectively in Figure 6. These dashed lines roughly follow the distribution for the entire request set, with one primary difference: the "bump" at 11 components. This bump is explained by the prefix-set distribution of CNN, which employs a much deeper naming tree, both for its articles and the individual HTTP elements referenced within an article. Thus, as input for our KRS experiments, we used a final set of 113,531 requests for 34,834 unique COs.

5.2 Zone Distribution and Colocation

The distribution of zones in KRS is critical to its performance, since it determines the number of authoritative servers that will be needed to power KRS resolution. Note that the distribution of *prefixes* does not necessarily reflect the distribution of *zones*. For example, a query-set consisting of the single prefix
"/parc/csl/papers/krs.pdf" would still require 4 KRS zones: the root zone, "parc", "csl", and "papers".

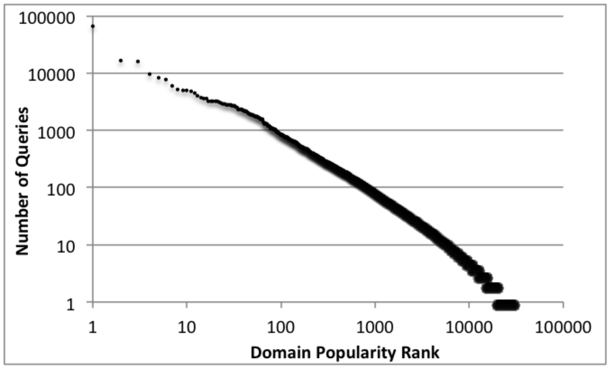

Figure 7: Popularity Of Second-Level Domains

From our filtered request set, we removed all leaf nodes and duplicates to create a set of 38,244 unique KRS zones. As before, we analyzed the number of components for each zone name in our set; we plot the cumulative density function (CDF) for all zones as well as for each second-level domain in Figure 8. While the CDF for Google, MSN, and Yahoo all roughly followed that of the mean CDF for all zones, the CDFs for both Friendster and CNN exhibited different behavior. CNN's CDF was essentially two components higher than the average, but this was consistent with the aforementioned behavior observed in Figures 6. For Friendster, the increase from 5 to 7 is explained by the Friendster name-prefix format, which followed the pattern "/com/friendster/blogs/{username}/{contentname}".

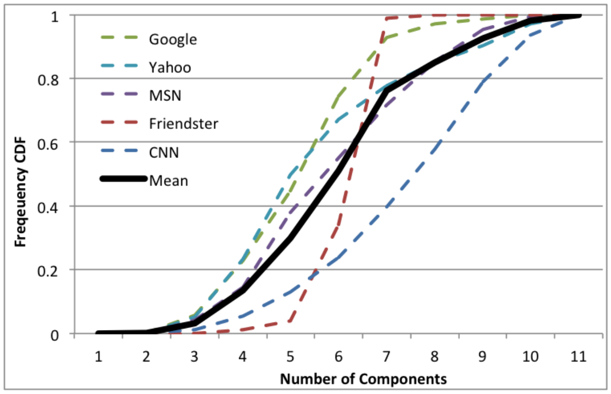

Figure 8: CDF of Name-Components In A Zone

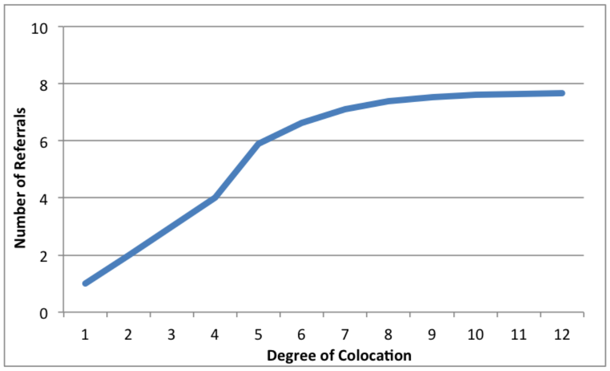

Figure 9: Expected Mean Number of Referrals With Colocation

We present the CDF since it illustrates the percent of zones containing up to a certain number of components. This parameter is important because though our filtered request set requires 38,244 different zones, it is unlikely that each zone will be hosted on a separate server. Rather, we expect that a single server may be responsible for a large number of zones, either through the use of default entries or simply by hosting multiple zone-files; we call this process *zone colocation*. While zone colocation for a domain is the responsibility of the organization owning the domain, it is still an important parameter to consider as it directly impacts the number of referrals that need to be performed as well as the response time to get back a KRS record.

To explore zone colocation, we started by defining a global *zone colocation number* Z which represents a "cap" on how long a zone's prefix may be. For example, when $Z = 3$, the zones "/parc", "/parc/papers", and "/parc/csl" may each be located on separate servers, but the zone "/parc/csl/lab3" may *not* have its own server: rather, it *must* be colocated with "/parc/csl". To analyze the system effects of varying Z, we calculated the number of referrals for our request set for different values of Z. For this calculation, we assumed that every zone that *may* have its own server, *does* have its own server. We explicitly assumed no caching. Thus, if a request has n components, it will result in exactly n referrals, unless $n > Z$, in which case the Zth referral definitively answers the query.

Figure 9 illustrates how we anticipate the mean number of referrals to vary with Z. At low values (e.g. $Z = 2$), the mean number of referrals is close to Z, since almost all requests have $> Z$ components. However, as Z increases, we observed a mean referral count of 7.11 at $Z = 8$ and 7.66 at $Z = 12$. This result highlights the fact that so few requests have a component-length of > 7 that even if colocation is not used, it will not significantly degrade KRS performance. Thus, we conclude that only Z values lower than 7 are useful for improving system performance via limiting referrals.

Building on this result and from Figure 8, we chose Z values for each domain based on the distribution of names. Thus, we chose a value of $Z = 5$ for Google, MSN, and Yahoo, and $Z = 6$ for Friendster and CNN. This roughly corresponded to a value of 0.3 in the CDF for each domain, and reduced our set of 38,244 unique zones down to a manageable 713. Thus, in our simulation topology we have 713 KRS authoritative servers, with each server handling one zone.

5.3 Underlying CCN Topology

To pick the underlying CCN topology for our KRS performance evaluation, we chose a modified version of Rocketfuel's "Verio US" topology [14] consisting of 921 nodes in one connected component. We classified 462 leaf nodes as "client" nodes that connected to 269 "gateway" nodes; the remaining 190 nodes were classified as "backbones". Each node employed a LRU cache eviction policy; we varied the cache size and report the results in Section 5.4. KRS servers run as a service over the nodes in this topology. To ensure that we are observing the performance of the underlying CCN topology itself, we disabled all KRS record caching, and employed a recursive forwarding scheme at every KRS server. For the sake of evaluating only KRS performance, we disregarded the presence of individual clients in our evaluation, and configured local KRS servers to directly request the prefixes from the workload described above.

We assumed that authoritative KRS servers are located at the edge of the network, either at client or gateway nodes. We distributed the 713 authoritative KRS servers across the combined total of 731 client and gateway nodes and assigned similar prefixes (e.g. /com/friendster/blogs and /com/frienster/www) to nodes that were close to each other in the topology.

5.4 CCN Caching

Our goal in this experiment was to determine the cache size to be set in CCN. We started by choosing one node in the above topology to represent a local KRS server that requested every Interest in the aforementioned request-set, with the time between requests randomly generated according to an exponential distribution with $\mu = 1.0$ request/sec. We repeated our experiments for various cache sizes (i.e. the number of elements cached), including cache size 0. We measured both the average KRS messages passed and average latency per request and plot them in Figure 10. With no caching, we observed an average number of 7.66 KRS messages passed (3.83 referrals) per request; this number correlated well with the expected results shown in Figure 9, given the degree of colocation. The mean latency of 1.02 seconds per request was roughly similar to values observed by collecting DNS traffic [15]. We note that a direct com-

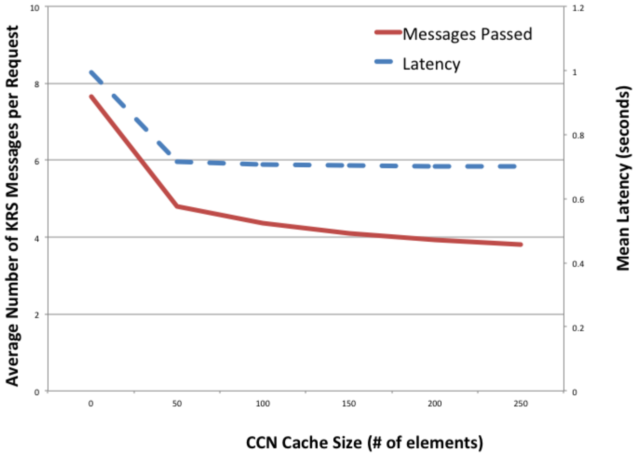

Figure 10: Average KRS messages per request vs CCN cache size

parison between KRS and DNS is not valid because: (1) our values are collected without caching, (2) KRS zone tree being much deeper than that of DNS, results in a much higher number of mean KRS referrals, and (3) the underlying network architecture is completely different. As expected, with caching enabled, we observed an immediate, sharp drop-off, both in latency and average number of KRS messages per request. In accordance with observations about the popularity distributions of COs and their effects on caching [16, 17, 18], we achieved a majority of these benefits even at low cache sizes. Thus, we set CCN cache size to be 100 in the rest of our KRS simulation experiments.

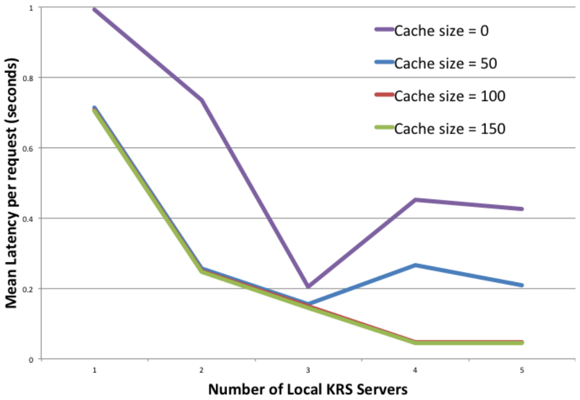

Figure 11: Latency And Number of Local KRS Servers

5.5 Local KRS Servers

Next, we evaluated whether the *number* of local KRS servers issuing KRS requests impacted the average number of KRS messages and latency per request. We repeated the same experiment as above while varying the number of local KRS servers. To account for different topological locations, we ensured that local KRS servers were deployed only at client nodes. Whenever a local KRS server issued a request, it simply chose the first unrequested prefix from the request set. Since our evaluation topology consisted of one individual ISP in the United States, we did not consider client locality, as several studies [19, 20, 21, 22] have shown that client locality is only significant at a coarse-grained level, or where cultural/linguistic boundaries were crossed.

We observed that the average number of KRS messages per request remained unaffected by the number of local KRS servers making requests; this result was expected, since we explicitly disabled all KRS caching. However, we also observed a decrease in mean latency as we increased the number of local KRS servers; this decrease is illustrated for different CCN cache sizes in Figure 11. Notably, for all cache sizes the mean latency did not continue to decrease when we increased the number of local KRS servers beyond 4. We attribute this decrease in latency to two factors: better link-utilization and shorter average distance. However, these benefits are primarily attained with even a small number of local KRS servers, and this explains the lack of change in latency beyond 5 local KRS servers. Thus, for the purpose of our evaluation, we concluded that 4 or 5 is an acceptable value for setting the minimum number of local KRS servers.

6. KRS PERFORMANCE RESULTS

In the previous section, we described our workload and the impact of different system and configuration parameters as well as network characteristics on KRS performance. In this section we evaluated KRS itself, using the configuration parameters chosen in the previous section; we studied KRS behavior under varying KRS-specific parameters such as KRS cache size and forwarding scheme (iterative versus recursive) used.

6.1 KRS Recursion and Caching

Caching and forwarding are fundamentally intertwined and directly impact each other: if caching is disabled, then both the recursive and iterative forwarding schemes result in the same number of referrals and KRS messages. As cache size increases, the forwarding scheme used has growing importance, since it determines which intermediate KRS servers receive and cache entries. To evaluate the impact of these parameters, we enabled KRS caching and examined how KRS cache size impacted average number of KRS messages per request. As discussed previously, we set CCN cache size to 100 elements, and the number of local KRS servers to five (5).

We define a "recursion number" R, where zones with prefix-length $> R$ resolve prefixes recursively, and zones with prefix-length $\leq R$ do so iteratively. We designed this model by observing policies enacted by real-world DNS servers today, where authoritative servers higher in the hierarchy (e.g. root and TLD servers) generally disable recursive resolution [15] to address concerns about performance and security. In all our previous experiments, KRS used recursive forwarding, thus $R = 0$ for those experiments.

In Figure 12, we plot the average number of KRS messages per request as a function of varying KRS cache size (number of elements cached) for different values of R. We note that the *size* of the KRS cache does not have a very strong effect on average messages per request.

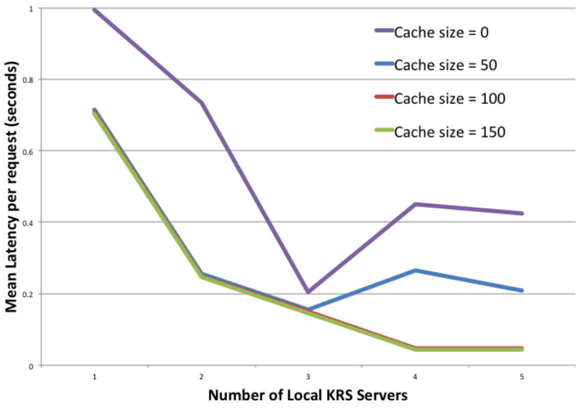

Figure 12: KRS Caching For Varying Values of R

The above result can be explained by considering the results observed in Figure 10, and recalling that this experiment was run with a *CCN* node cache size of 100. As seen from our request workload, KRS requests follow a Zipf distribution implying that a majority of KRS requests are satisfied from CCN's cache. Increasing KRS cache sizes thus does not have any further impact on reducing KRS messages. Furthermore, when $R = 0$, every server is responding recursively. Thus no querying server may learn the name of the second-hop KRS authoritative server and its public key to be cached for future use.

The true strength of longest prefix matching for looking up entries in the next-hop table is not realized and thus there is no reduction in the number of referrals. Effectively, the KRS cache is used only for exact-matching, and the observed performance as we increase the KRS cache size roughly mimics that of increasing the CCN cache size.

We note that increasing the value of R significantly reduces the average number of KRS messages per request as soon as KRS caching is enabled. The benefits are significant for a KRS cache size of 50. However, increasing KRS cache size further does not reduce the average KRS messages much more – the average KRS messages line flattens out with further increase in KRS cache size.

With KRS caching enabled, the true strength of *longest-prefix-matching* is realized and used to reduce intermediate referrals, as described in Section 4.

For example, when $R = 1$, the root server "/" responds iteratively, so local KRS servers may learn the name and the key of the authoritative server handling "/com." They cache this entry in their next-hop table. Thus, even for a small KRS cache size, we quickly see the majority of gained performance benefits, since increasing the KRS cache size further does not reduce any more intermediate referrals. Likewise for both $R = 2$ and $R = 3$: small cache sizes are sufficient to store the set of next-hop server names and their keys, and this caching is responsible for the majority of the performance benefits observed.

6.2 KRS Storage

In this subsection, we discuss the storage requirements for KRS records.

We expect a KRS record to be about 6 KB on average. As described in Section 4, a KRS record consists of the content name, the public key or content hash, the signature, and a certificate or a certificate chain. While CCN names can be arbitrarily long and can have an arbitrary number of components, for efficiency purposes, we expect content names to not exceed 1 KB [2]. Even for a strong 4096-bit signing key, the associated certificate size is about 1.5 KB. Thus, we expect an average KRS record to be approximately 6 KB.

For successful KRS resolution, authoritative zones *must* store the set of KRS entries for which they are responsible. From the colocation methodology described in Section 5, we ran tests on a set consisting of 34,834 unique KRS records. Assuming each record is on average 6 KB in size, storing 34,834 records require approximately 209 MB across the *entire system*. Since this system contains only the five most popular domains, and given that our evaluation shows that these domains do accurately reflect the hierarchy and distribution of name-prefixes, we expect that storing all the KRS records for an organizational domain would require approximately 40 MB on average.

From our experiments in Section 6.1, it is notable that we achieved a majority of KRS performance benefits from caches as small as 50. Even assuming that caches must be two orders of magnitude larger (5000 elements) in order to see these same benefits in a larger topology, storage requirements for caching KRS records at intermediate KRS servers is not a concern.

6.3 Creating, Updating, and Deleting Entries

KRS must also support creating, updating, and deleting records. However, evaluating the performance of such operations is challenging, for several reasons. Today's content delivery protocols and systems largely perform these operations off-line, and as a result it is challenging to find collected data-sets from which we can construct a realistic workload for evaluation, as we did in Section 5.1. We expect the set of create/update/delete entries to not be a significant performance overhead. This expectation comes from the observation that offline configuration (i.e. manually configuring a DNS or HTTP server) are acceptable *only* because such operations happen much more infrequently than content is read.

Additionally, KRS's longest prefix matching look-up algorithm for retrieving KRS records provides significant benefits. Through the use of default entries, KRS provides a powerful mechanism to mitigate the number of create, update, and delete operations. By simply storing the key used to sign every CO under a particular name prefix, no further KRS operations are needed, regardless of how frequently a publisher changes the COs below this prefix.

7. CONCLUSION

We proposed a Key Resolution Service (KRS) for CCN. KRS provides a system for the registration, storage and distribution of security information associated with namespaces in CCN. By taking advantage of the hierarchical naming scheme in CCN, KRS can securely and unambiguously map namespaces in CCN to public keys that are authorized to publish in those namespaces. Additionally, KRS supports a secure mapping between a content name and the cryptographic digest of the Content Object carrying that name.

By analyzing a set of collected real HTTP traces, we estimated the workload for a KRS system and designed realistic experiments to evaluate the scalability of our system under various conditions. Our experiments show that the proposed KRS design is scalable to a global deployment and that small cache sizes at different layers (i.e., at KRS and the CCN layer) significantly benefit KRS performance.

To the best of our knowledge, KRS is the first practical proposal that can fully eliminate the threat of content poisoning attacks in CCN (and its sibling architecture NDN) by enabling consumers to acquire the necessary security information to unambiguously request and receive trustworthy content —regardless of the existence of untrustworthy content in router caches that might share the same name.

8. ACKNOWLEDGMENTS

This research was supported in part by the NSF Future Internet Architecture (FIA) Program Award G015.3707.

9. REFERENCES

[1] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher, and Börje Ohlman. A Survey of Information-Centric Networking. *IEEE Commun. Mag.*, 50(7):26–36, 2012.

[2] PARC. Content centric networking project (ccn). http://www.ccnx.org.

[3] Paolo Gasti, Gene Tsudik, Ersin Uzun, and Lixia Zhang. Dos and ddos in named data networking. In *Computer Communications and Networks (ICCCN), 2013 22nd International Conference on*, pages 1–7. IEEE, 2013.

[4] Cesar Ghali, Gene Tsudik, and Ersin Uzun. Elements of trust in named-data networking. *CoRR*, abs/1402.3332, 2014.

[5] Diego Perino and Matteo Varvello. A reality check for content centric networking. *In Proc. ACM SIGCOMM Workshop ICN*, 2011.

[6] We Knew The Web Was Big... http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html.

[7] Spencer Sevilla, Priya Mahadevan, and JJ Garcia-Luna-Aceves. FERN: A unifying framework for name resolution across heterogeneous architectures. *Proc. IFIP NETWORKING*, 2013.

[8] ICANN Zone File Access Advisory Group. gtld zone file access in the presence of large numbers of tlds. http://www.icann.org/en/topics/new-gtlds/zfa-concept-paper-18feb10-en.pdf, 2010.

[9] S. Weiler and D. Blacka. RFC 6840: Clarifications and Implementation Notes for DNS Security (DNSSEC). *IETF Standard*, 2013.

[10] Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. ndnSIM: NDN simulator for NS-3. Technical Report NDN-0005, NDN, October 2012.

[11] ns-3 network simulator. http://www.nsnam.org/.

[12] Won So, Ashok Narayanan, and David Oran. Named data networking on a router: fast and dos-resistant forwarding with hash tables. In *Proceedings of the ninth ACM/IEEE symposium on Architectures for networking and communications systems*, pages 215–226. IEEE Press, 2013.

[13] IRCache traces. ftp://ircache.net.

[14] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring isp topologies with rocketfuel. *ACM SIGCOMM Computer Communication Review*, 32(4):133–145, 2002.

[15] J. Jung, E. Sit, H Balakrishnan, and R Morris. DNS Performance and the Effectiveness of Caching. *Networking, IEEE/ACM Transactions on*, 10(5):589–603, 2002.

[16] Seyed Kaveh Fayazbakhsh, Yin Lin, Amin Tootoonchian, Ali Ghodsi, Teemu Koponen, Bruce M Maggs, K C Ng, Vyas Sekar, and Scott Shenker. Less Pain, Most of the Gain: Incrementally Deployable ICN. In *Proceedings of SIGCOMM 2013*, page 1. ACM, 2013.

[17] L Breslau, P. Cao, L Fan, G Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. 1:126–134, 1999.

[18] Mark E Crovella and Azer Bestavros. Self-similarity in World Wide Web traffic: evidence and possible causes. *Networking, IEEE/ACM Transactions on*, 5(6):835–846, 1997.

[19] T Chung, J Han, H Lee, and J Kangasharju. Spatial and temporal locality of content in BitTorrent: A measurement study. *Proc. IFIP NETWORKING*, 2013.

[20] John S Otto, Mario A Sánchez, David R Choffnes, Fabián E Bustamante, and Georgos Siganos. On blind mice and the elephant: understanding the network impact of a large distributed system. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 110–121. ACM, 2011.

[21] Michal Kryczka, Ruben Cuevas, Carmen Guerrero, and Arturo Azcorra. Unrevealing the structure of live bittorrent swarms: methodology and analysis. In *Peer-to-Peer Computing (P2P), 2011 IEEE International Conference on*, pages 230–239. IEEE, 2011.

[22] Ruben Cuevas Rumin, Nikolaos Laoutaris, Xiaoyuan Yang, Georgos Siganos, and Pablo Rodriguez. Deep diving into bittorrent locality. In *INFOCOM, 2011 Proceedings IEEE*, pages 963–971. IEEE, 2011.