Fern : An updatable authenticated dictionary suitable for distributed caching

Eric Freudenthal, David Herrera, Steve Gutstein, Ryan Spring, and Luc Longpré
{efreudenthal, daherrera, smgutstein, rcspring, longpre}@utep.edu

University of Texas at El Paso
Computer Science Technical Report #UTEP-CS-06-45
December 2006

Abstract

Fern is an updatable cryptographically authenticated dictionary developed to propagate identification and authorization information within and among distributed systems. Conventional authenticated dictionaries permit authorization information to be disseminated by untrusted proxies, however these proxies must maintain full duplicates of the dictionary structure. In contrast, Fern incrementally distributes components of its dictionary as required to satisfy client requests and thus is suitable for deployments where clients are likely to require only a small fraction of a dictionary's contents and connectivity may be limited

When dictionary components must be obtained remotely, the latency of lookup and validation operations is dominated by communication time. This latency can be reduced through the exploitation of locality-sensitive caching of dictionary components. Fern dictionary's components are suitable for caching and distribution via autonomic scalable locality-aware Content Distribution Networks (CDNs) and therefore can provide these properties without requiring the provisioning of a dedicated distribution infrastructure,

Others have proposed the construction of incrementally distributed authenticated dictionaries that utilize either trees that dynamically re-balance or skiplists. The structural changes that result from tree rebalancing can reduce the effectiveness of caching. Skiplists do not require balancing and thus are more amenable to caching. However a client lookup from a skiplist-based dictionary must sequentially transfer two-to-three times as many components as a client of a dictionary based on self-balancing trees. In both cases, these transfers are *necessarily serialized*, and thus skiplists will incur proportionally increased latency.

Fern's dictionary structure utilizes a novel randomized trie that has the desirable characteristics of both of these approaches. While Fern's algorithm is far simpler than self-balancing trees, a Fern trie will have similarly short (average and expected worst case) path lengths, and thus requires that clients obtain approximately the same number of vertices. Furthermore, like skiplists, Fern's trie *does not require rebalancing* and thus is similarly amenable to caching.

A prototype implementation of Fern has been constructed that utilizes the CoralCDN scalable, locality-aware, and autonomic content distribution network. We provide an informal analysis of bandwidth requirements for the Fern authenticated dictionary that agrees with experimental results. We are not aware of other implemented systems with similar properties or comparable analysis of such systems' performance and bandwidth requirements.

Finally, the potential integration of Fern within the CDN on which it relies could yield symbiotic benefits. The indexing infrastructure for autonomic CDNs such as Coral are vulnerable to disruption by malicious participants. Therefore, a CDN's integrity could be guarded against malicious interference through the dissemination of up-to-date authorization information provided by Fern. In a complementary manner, a CDN so fortified by Fern could potentially provide more reliable content distribution service to Fern and thus also improve Fern's availability and performance.

1 Introduction

Reliable and consistent dissemination and revocation of access permissions and identity mappings are persistent problems in distributed systems. Authorization information in distributed environment is frequently communicated through independent credentials or on-line queries, each authenticated by a distinct public-key computation. Fern utilizes a complementary approach in which widely-trusted originators authorization information incrementally disseminate an efficiently searchable dictionary containing a large set of authorization information.

Commonly used approaches to consistent dissemination of trust and authorization information include on-line validation services such as OCSP[15] and the dissemination of time-limited public-key based certificates[15, 4, 18, 11].

These approaches share a high computational cost for public-key cryptography. They also reflect a range of scalability, timeliness, and connectivity characteristics. On one extreme, on-line status verification protocols provide per-transaction validation of access permissions when sufficient computational and communication bandwidth to trusted online servers can be assured. Credentials with extended lifetimes reduce the bandwidth and computational power required to convey information needed for authorization decisions by effectively relaxing consistency. However, a window of TOCTTOU[1] vulnerability is created if no mechanism is incorporated to disseminate and respond to premature revocation.

Updatable online authenticated dictionaries[19, 10, 9, 2] can provide a complementary source of authorization information based upon integrity-guaranteed name-to-value mappings stored within an indexed or searchable data structures. These mappings may be used to provide evidence of authorization or revocation[2]. Mechanisms to detect tampering are embedded within these dictionaries in a manner that permits their distribution via untrusted proxies to security-sensitive clients who can efficiently validate mappings (or the lack of mappings) stored within them.

A feature of authenticated dictionaries is that proxies and security-sensitive clients can incrementally obtain portions of their search structure as required to answer queries. These partial copies of the dictionary's structure can be cached and therefore utilized to answer future queries that share common components.

Modifications to mappings stored within an authenticated dictionary typically cause changes to dictionary's search structures. Since components of an authenticated dictionary may be cached, it is desirable if these updates are localized. Thus, algorithms that rely on structural modification to achieve good performance (e.g. self-balancing trees) can be problematic.

Our prototype of Fern utilizes a randomized prefix-trie that has expected $O(\log n)$ complexity for all update, lookup, and validation operations. This complexity is asymptotically equivalent to other authenticated dictionaries. However, we argue that, given the high latency of inter-host communication in distributed systems, *the constants do matter* and Fern's expected cost is comparable to the most aggressive ealgorithms based upon self-balancing trees and far lower than skip-lists. Furthermore, Fern's algorithm is far simpler than self-balancing trees and does not require rotation.

While others have proposed designs for authenticated dictionaries whose contents are distributed via an autonomic content distribution network (CDN), we are unaware of any other implemented systems. We present measurements collected from our prototype implementation that utilize the CoralCDN self-organizing and locality-aware content distribution network.

Finally, we believe that we are the first to consider the impact of cacheability upon the cost of monitoring a set of mappings stored within an authenticated dictionary that exploits caching. In Section 4.1, we provide an informal analysis that agrees with observed behavior.

2 Related Work

On-line authenticated dictionaries disseminate the contents of a database containing name:value mappings to network-connected clients. The *originator* of an authenticated dictionary embeds a set of immutable

[1] Time-of-check-to-time-of-use (see [7])

[2] E.g. that the person known to the dictionary's originator as "Alice" is associated with public key K_A, or that certificate number N has been revoked.

name:value mappings within a search structure[3]. Like other authenticated dictionaries, Fern exploits the structural coposition of cryptographic hashes (see Merkle[14]) to amortize the cost of a single public-key crytpographic operation that serves as a witness to entire dictionary's integrity. This facilitates the scalable distribution of these mappings by untrusted *publishers* whose clients request and validate copies of mappings stored in these dictionaries by querying any publisher claiming to have a current copy.

Due to their potential to autonomically scale and respond to dynamic changes in connectivity, we are particularly interested in authenticated dictionary structures that are suitable for dissemination by peer-to-peer content distribution networks. Our prototype of Fern utilizes the CoralCDN[5] locality-aware content-distribution network.

It is desirable for authenticated dictionaries to be structured in a manner that permits (1) publishers to determine the integrity of nodes they distribute and (2) clients to determine the validity of name:value mappings without requiring transfer and/or storage of the entire dictionary structure.

A common approach is to store name:value mappings within a search-DAG (directed acyclic graph) where each internal vertex contains cryptographic hashes of its children and thus achieves the integrity properties of a Merkle-tree[14]. As such, all internal vertices contain hashes of their children and each hash contains the hashes of its children. Thus, vertex hashes can transitively witness the integrity of all descendant vertices. A path consisting of vertices from the DAG's root to a vertex containing the referenced mapping serves as a *Validation Object* (VO)[12] that witnesses the mapping's integrity. The number of nodes in a VO is bounded by the DAG's depth, which is typically logarithmic in the number of mappings stored within the dictionary.

While cryptographic hashes can be used to demonstrate that (portions of) a previously composed data structure has not been corrupted, other mechanisms are typically used to disseminate identifiers for and hashes of updated structures. The identifier permits the dictionary's originator to specify the *root* of a dictionary structure, and the cryptographic hash provides a witness to the dictionary's integrity. Fern conjoins search-structure identifers and their hashes into a single *node identifier* (NID) and the root's NID is embedded within a *root certificate* that both references and authenticates the current root of a Fern dictionary. Like other authenticated dictionaries, the integrity of a Fern root certificate is witnessed by a digital signature generated using asymmetric cryptography.

The typical application of an online authenticated dictionary has a single (exclusive) writer and multiple (concurrent) clients and thus has semantics analogous to the CREW (concurrent-read exclusive-write) shared memory storage model. However, due to an authenticated dictionary's pervasive use of hash composition, it is not sufficient to modify a name:value mapping *in place*. As a result, authenticated dictionaries are typically immutable and thus provide semantics similar to WORM (write-once, read-many) storage devices (e.g. CD-ROMs and PROMs). In order to support online updates, originators typically distribute (1) replacement dictionaries and/or (2) a continuous stream of updates that must be applied to the previously distributed full dictionary. The first approach can require the periodic dissemination of a large data structure that can be problematic for bandwidth-limited deployments. The second approach requires that publishers obtain *all* updates since the current dictionary was published and the transmission of *all* missed updates after a prolonged disconnection can be be expensive. Furthermore, this full-replication approach will distribute portions of the dictionary that may not be referenced by a particular publisher's clients.

In contrast, Fern achieves CREW semantics by approximating in-place updates. Vertices whose contents have changed are *replaced* and disseminated *on demand*. Since vertex contents include cryptographic hashes whose value is dependent on their descendants, each updated mapping will typically result in the replacement and dissemination of all $O(\log_2 n)$ vertices on the path between the root certificate and the vertex containing the mutated mapping.

Similar techniques have been explored by Goodrich, Tamassia, et al.[10, 8, 19]. These approaches utilize either skiplists or self-balancing search trees. However, skiplists require that clients transfer two-to-three times more vertices than Fern's trie. Self-balancing search tress require complicated algorithms and the structural changes that result from tree rotations reduce the utility of caches, and thus can result in the replacement and transfer of more tree vertices.

[3]Fern can also implement a common variant of authenticated dictionary used to communicate certificate revocation lists that disseminates sets of names (e.g. certificate serial numbers) with empty values. Their function is analogous to the word lists frequently used to arbitrate disputes in the popular parlor game Scrabble.

3 Approach

A Fern dictionary's internal structure is a binary Merkle-trie with path compression. Like the similarly authenticated authorization framework suggested by Tamassia et. al. in [19], vertices from Fern's Merkle-trie are suitable for distribution by peer-to-peer (P2P) Content Distribution Networks (CDNs) to clients who construct and maintain VOs for mappings on which they depend.

As illustrated in Figure 1, Fern's trie is a binary prefix tree whose internal vertices with no branching are eliminated. Name:value mappings are stored in leaf vertices whose search key is equal to the name's SHA-1 hash. Following the model of a Merkle-tree[14], internal nodes also contain SHA-1 hashes of each immediate descendant (these hashes are not indicated in this figure). The incorporation of these hashes permits clients who possess valid copies of internal vertices to validate the integrity of their descendant's. The current root and its hash is stored in a *root certificate* signed with the originator's public key using GPG's DSA cipher.

By creating a Fern dictionary, an originator serves as a *Certification Authority* for clients who depend on mappings stored within it. A Fern verification object (VO) for a particular name corresponds to a search path containing its mapping. If the name is stored within the trie, it will terminate in a leaf containing its mapping. Otherwise, it will contain a search path that demonstrates that the referenced name is not in the trie.

Our initial implementation of Fern publishes through the CoralCDN self-organizing and locality-aware content distribution network. Coral serves as a scalable autonomic proxy-cache for frequently-referenced vertices. Clients obtain vertices required to construct or update VOs from coral and thus avoid transferring the entire trie structure. This helps to minimize network congestion that arises from distributing vertices. Furthermore, client caching is exploited, thus eliminating the need to obtain the same vertex multiple times.

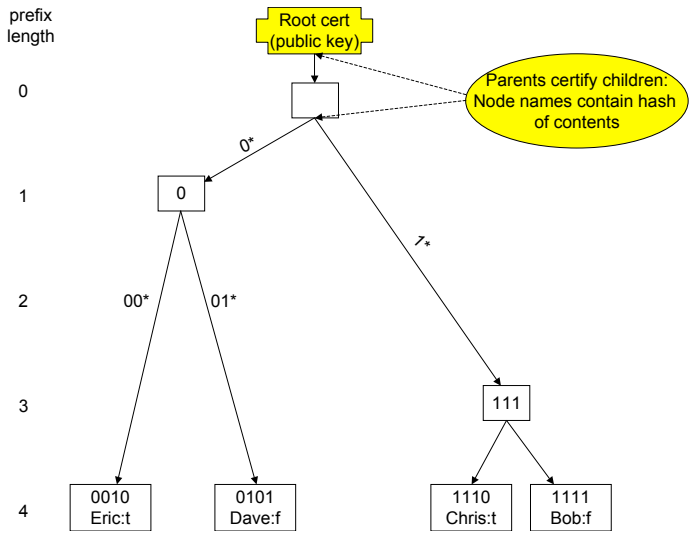

Figure 1: A Fern trie for 4-bit keys containing leaves (0010, 0101, 1110, and 1111). The leaves contain mappings of names to groups. Internal nodes are labeled by their corresponding key prefix and edges denote the bits that they match.

3.1 Authentication with Fern

Figure 1 depicts a simplified example of the structure of a Fern trie. In contrast to Fern, which uses 160-bit SHA-1 hash digests as search keys, the simplified trie depicted in Figure 1 only uses four-bit search keys. Name:value mappings are stored in leaf vertices with search-keys corresponding to the name's hash digest. In this figure, the name "Bob" (mapped to f) is depicted as having a hash equal to 1111.

Routing decisions within this trie are made one bit at a time, in decreasing order of significance. Each internal vertex is labeled with the search prefix corresponding to the common prefix shared by all its descendants. Vertices within tries that have no branching are collapsed as is illustrated by the absence of vertices corresponding to the prefixes 1, 10, 11, and 110.

A client with interest in an Id's mapping stored within Fern obtains the set of trie nodes that comprise the path from the root to the desired leaf. As described in Section 5.1, leaves within Fern's trie are generally at a depth of $1.1 \log_2 n$ where n is the number of mappings stored within the trie.

A leaf vertex with search key k stores a name:value mapping where k is the name's SHA-1 cryptographic hash. Each internal vertex of a Fern authorization trie contains node identifiers (NIDs) of its children. NIDs are used as a locator by clients requesting particular nodes. Like SFS-RO's[6] self-certifying pathnames, Fern's NIDs include cryptographic hashes of the referenced node's contents and are used to verify the integrity of a nodes obtained from insecure channels.

As illustrated at the top of Figure 1, the Fern CA periodically publishes a signed, time-limited *root certificate* using asymmetric cryptography. Fern clients know the CA's asymmetric public-key identity and use it to determine the integrity of root certificates. Every internal node of a Fern trie contains NIDs of its immediate descendants, and NIDs contain cryptographic hashes of the nodes they reference. Thus the full path of trie nodes from a root certificate to a leaf node can be validated.

A VO for a referenced identifier, I, contains all Fern-nodes between the Fern-root and I. A full certification path can be included within the payload of a single message transmitted from one Fern client to another. Alternatively, since each internal node of a Fern-trie contains both search guides and the NIDs of immediate children, a Fern client can obtain a certification path for I by obtaining intermediate nodes as it traverses the path from the root toward the leaf node containing I.

A search-path that comprises a complete VO can be transmitted directly between clients, or instead be constructed by a client that recursively searches for the trie leaf containing the required name:value mapping.

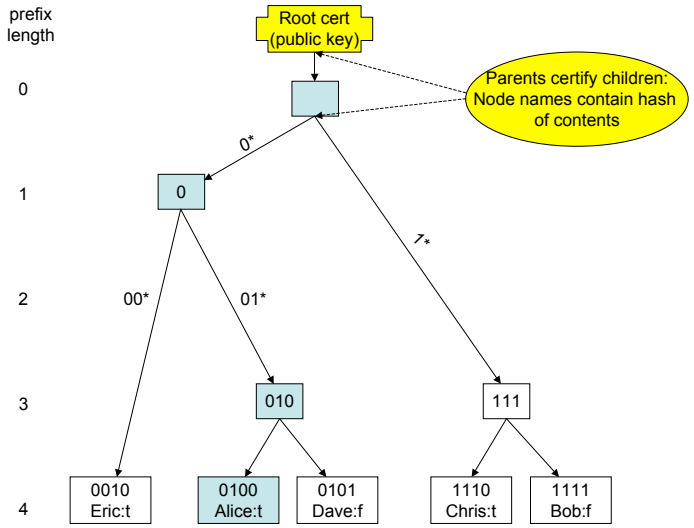

Figure 2: Inserting node 0100 requires inserting a new internal node 010 and updating all ancestor nodes (shaded) along the path.

3.2 Trie Insertion

Figure 2 illustrates modifications the the trie of Figure 1 after the insertion of a mapping for "Alice" to "t". Observe that this insertion also required the creation of a new internal vertex corresponding to prefix "010." Since each internal vertex contains the cryptographic hash of its children, all of Alice's ancestors will

be updated as is indicated by shading in this figure. Since a new NID will be used to identify each of these updated vertices, they will be distributed by the CDN as replacement rather than as updated values.

Note that clients will be unaware of any updates until the next time a root certificate is issued that references the replacement root. Our prototype implementation of Fern generates a new root certificate every five minutes, and thus updates are grouped into five minute epochs.

3.3 Interactions Between Clients

Assume Bob and Alice are motivated to enter into a trust-sensitive relationship for which their Fern VOs can serve as supporting evidence. It is likely to be most efficient for them to exchange their complete VOs at the time the relationship is established.

Since this relationship may be prolonged, Alice will be motivated to monitor Bob's authorization and thus will need to update his VO every time the trie root is modified. Since a prolonged relationship may extend beyond the current root certificate's lifetime, additional vertices may be required to continue their mutual authorization. Our implementation of Fern automatically obtains vertices needed to update VOs representing such continuing relationships from the CDN and thus exploits the CDN's ability to scalability distribute online content in a locality-sensitive manner.

4 Comparison with Other Cacheable Authenticated Dictionaries

Several designs for authenticated dictionaries with equivalent asymptotic complexity have been proposed. In this section, we examine their *expected* cost and argue that *the constants matter* because latency of authenticated dictionary operations that rely on distributed caches to distribute vertices is heavily dependent on the latency of *necessarily serialized communication*.

As discussed in Section 3.3, we observe that the determination and maintenance of authorization are important operations for clients of authenticated dictionaries. In both cases, a client's challenge is to repeatedly obtain and validate the subset of a dictionary's current vertices needed to maintain VOs that authorize relationships on which the client relies.

A VO for a particular mapping stored within an authenticated dictionary can be quickly generated if all needed vertices are available from a nearby dictionary. However, if the dictionary is large, connectivity is limited, or worse connectivity varies dynamically, it may be impractical to distribute the entire dictionary to proxies with good connectivity to clients.

Fern uses an alternative approach of distributing a dictionary's data structures incrementally through the use of an autonomic content distribution infrastructure (such as a DHT). In such a deployments, each successive vertex in a VO's transitive chain must be obtained sequentially.

As observed by Tammasia and Triandopolos, the expected time to acquire a dictionary node stored within a DHT composed of P hosts will require $O(\log P)$ pairwise communications that must be serialized. Since each pairwise communication can take dozens to hundreds of milliseconds, communication latency is likely to dominate the time required to assemble a VO.

Since the computational cost related to construct and verify a VO is small, the time required by a client to assumable or update a VO will be dependent on the the number of nodes that this client must obtain from the content distribution infrastructure.

Tamassia, Goodrich, Winsboro, and others identified skiplists[17] and various self-balancing trees[16, 3] as having the following desirable properties for the underlying data structures used for cacheable and updatable authenticated dictionaries.

- Logarithmic expected leaf node depth with narrow variance. *This ensures that the number of vertices within a VO is logarithmic in the number of words stored within the dictionary*
- Localization of tree update operations. *Rotation and other tree-rebalancing operations disrupt the tree structure and thus require replacement (and transmission) of tree vertices that might not otherwise be affected by a particular tree update operation.*

A Fern VO for a word stored at depth d contains d dictionary vertices. As described in Section 5.1, our initial evaluations of Fern indicate that, within a very narrow variance, the expected value for d is $1.1 \log_2 n$,

for reasonably large n. Furthermore, as described in Section 5.1, update operations associated with a word stored within a leaf of a Fern trie are localized to that leaf and its (approximately $1.1 \log_2 n$) ancestors.

In [10], Goodrich, Tammasia, et al. identify skiplists as suitable structures for constructing authenticated dictionaries in a distributed environment. While skiplists share Fern's desirable property of not requiring rebalancing, we observe that a client must collect a greater number of vertices when assembling a skiplist-based VO than a Fern VO: The expected number of skiplist vertices that must be examined when assembling a VO from a dictionary containing n words is $3 \log_2 n$, and the expected number of vertices within a skiplist that will be modified when an entry is updated is $2 \log_2 n$. As described in Section 4.1, Fern's expected value of $1.1 \log_2 n$ for the number of nodes to update compares very favorably with skiplists.

In [19], Tamassia and Triandopolos identify red-black and BB[α] as suitable dictionary structures for constructing authenticated dictionaries whose vertices are, like Fern, distributed by an autonomic DHT-based cache. These self-balancing trees utilize rotations to guarantee that the search-structure remains approximately balanced. While these rotations are infrequent, they will increase the number of vertices that are replaced and thus increase network communication required to update VOs not associated with the updated mapping.

The advantage that Fern's Merkle-trie achieves over self-balancing trees and skip lists arises from the ability of a good hash function (e.g. SHA-1) to evenly distribute a set of elements over an ordered range which is large compared to the number of elements being distributed. Each data element is hashed to the range 0 ... 2^{160}, where each value corresponds to a *possible* leaf of Fern's Merkle-trie. The specific values to which data elements are hashed become the *actual* leaves of Fern's Merkle-trie. It is because of the nature of the hashing function that the Merkle-trie is created in a well-balanced manner, without any dynamic interventions. This, massive virtual hash table structure is usable because it allocates memory for nodes in a lazy fashion. No node is allocated unless it is an *actual* leaf or if both of its children are either actual leaves or ancestors of actual leaves. Thus, as is common in tries, some edges will be compressed and further reduces the memory required for this trie, and reduces the number of vertices within a VO.

4.1 Expected number of refresh queries

The hash value of a Merkle-tries's leaf and all of its ancestors changes whenever the leaf's contents are updated. In this section, we consider the number of vertices that must be obtained by a client who is monitoring mappings stored within a set of $w = |W|$ *watched* leaves within a dictionary storing a total of n mappings in the event that $u = |U|$ leaves are *updated*.

Let $c = |U \cap W|$ be the number of vertices in W whose mappings have changed.

Below, we informally demonstrate how the cases where $w \leq u$ and $u \leq w$ are reducible to each other. Throughout both analyses, we will be making the approximation of a well balanced trie at all levels. As our empirical results show, this is a good approximation, especially as w, u and n become large.

4.2 Case 1: $w \leq u$

As illustrated in Figure 3, our analysis divides the trie into three regions. The upper region, which extends to depth $\lfloor \log_2 w \rfloor$ has approximately w vertices at its (approximate) lower edge. We assume that our hash function has uniformly distributed keys, and thus assume that most of these (approximately) w vertices are ancestors to the w leaves mapping members of W. Since $u \geq w$, it is likely that most of these (approximately) w vertices members are also ancestors of the members of $|U|$. Thus our client is likely to require updates for all (approximately) $2w - 1$ vertices in the upper region.

The middle region of this trie extends to depth $\lfloor \log_2 u \rfloor$. Like our analysis of the upper region, it is likely that most vertices above depth $log_2 u$ contain nodes that are ancestors of the u updated leaves and thus most vertices in this region are likely to have been updated. However, only w paths of length $\lfloor \log_2(u/w) \rfloor$ through this middle region are likely to be ancestors of vertices being *watched*,and thus only $w\lfloor \log_2(u/w) \rfloor$ of these vertices are likely to be needed by the client.

The lower region extends down from (approximately) level $\lfloor \log_2 u \rfloor$ .and thus has approximately $\lceil \log(n/u) \rceil$ levels. Since we assume that clients previously cached unchanged vertices leading to members of W, we expect that clients' searches will terminate near to the top of this region for most of the $w - c$ paths that do

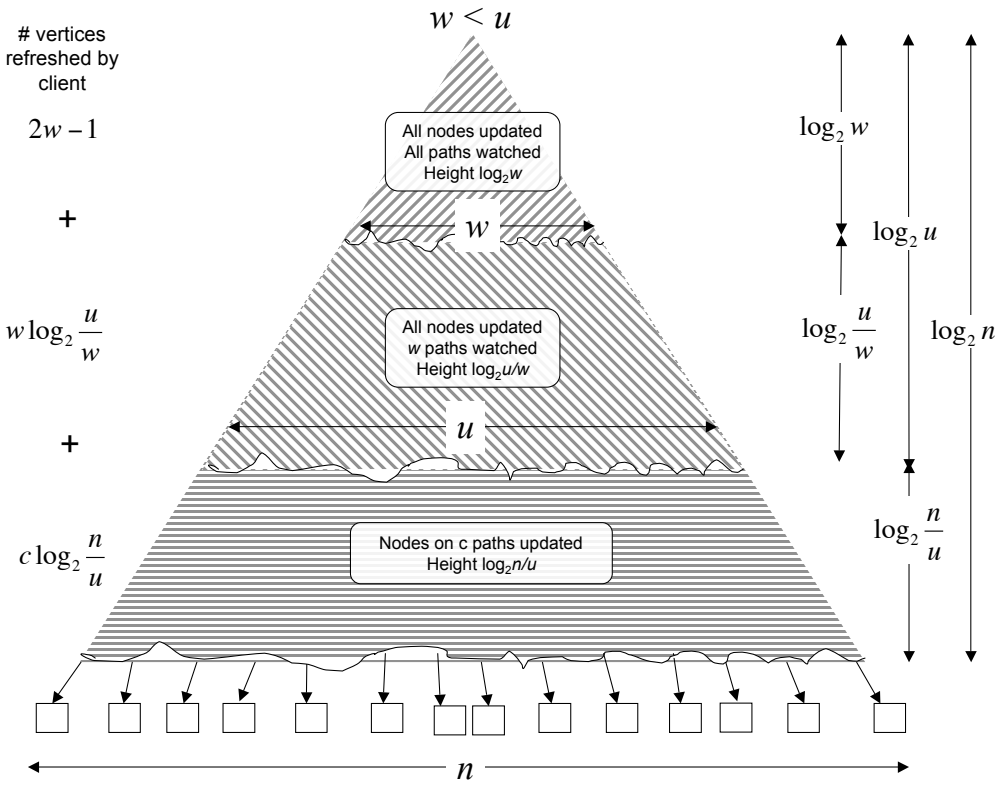

Figure 3: Illustration depicting our analysis of the number of nodes that need to be updated when the number of watched data entries is smaller than the number of updated data entries

not lead to modified leaves. However, the approximately $c\lceil \log_2(n/u) \rceil$ vertices along the c paths to leaves that are both watched and updated will be obtained by clients.

Thus we expect that approximately $(2w-1) + w\lfloor \log_2(u/w) \rfloor + c\lceil \log_2(n/u) \rceil$ nodes that will require updating. In practice, the shift from one section to the next may not occur exactly at the same level on every path, but these small variations do not significantly not change our estimates.

4.3 Case 2: $u \leq w$

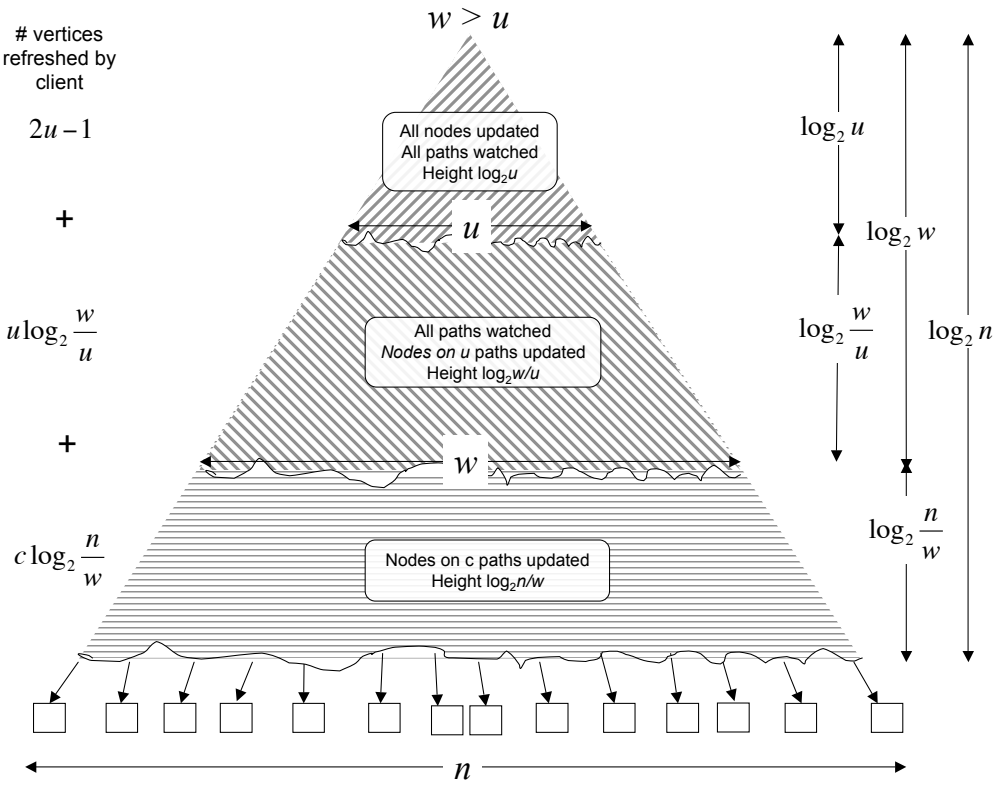

Figure 4: Illustration depicting our analysis of the number of nodes that need to be updated when the number of watched data entries is greater than the number of updated data entries

As with Case 1, we anticipate that most of the $2u-1$ vertices in the upper region will be ancestors to both watched and updated leaves, and thus will be obtained by clients.

Similarly, we anticipate that while most vertices in the middle region (with a lower edge containing w vertices) will be watched, only u paths will continue through to the bottom of this section of height $\lfloor \log_2 w/u \rfloor$ towards updated leaves, and that the other $w-u$ paths will terminate near its upper boundary. Thus, approximately $\lfloor u \log_2 w/u \rfloor$ of its vertices will be updated and therefore obtained by clients.

Thus w paths through the lower region will be watched. However, only c of them will lead to leaves in U and thus we expect that approximately $c\lfloor \log_2 n/w \rfloor$ vertices in this region will be obtained by clients.

Finally using the same arguments, approximately $(2u-1) + w\lfloor \log_2(w/u) \rfloor + c\lceil \log_2(n/w) \rceil$ vertices will be obtained by clients. They approach approach the same value for the case $w = u$, which also corresponds to the situation where the middle term of both expressions becomes 0.

We have a formal proof that for the case $c = 0$ and $w = 1$ the expected number of node updates is $\log_2 u \pm o(\log_2 u)$ that will be released as a tech report prior to the publication of this paper.

As stated above, the expected number of nodes a skiplist based dictionary would need to update is $2\log_2 n$, for the case $c = u = w = 1$. For this case, our expression reduces to $1 + \log_2(n)$. Once again, we see Fern achieves a significant advantage when one observes the constants within the asymptotic expressions.

5 Evaluation

We conducted both small-scale online evaluation experiments to determine gross runtime characteristics and offline simulations to examine behavior for larger-scale systems.

5.1 Offline Experiments

In order to measure the efficacy of searches using Fern, we ran several experiments using sets containing between 2^6 and 2^{18} names consisting of the decimal representation of sequentially ordered numbers and measured the distributions of the search depths obtained. We relied upon the randomization provided by the SHA-1 hash function to distribute the names throughout the branches of the Fern trie. Path compression within the trie further reduced the resultant search depths. An optimal tree, which contained n names, would have each name located in a leaf at depth $\lceil \log_2 \rceil$

In the graphs associated with these experiments, we normalize depths by dividing them by $\log_2 n$, thus providing a meaningful metric for how well the Fern trie has placed a given leaf. When the normalized depth of a leaf is 1, that leaf is optimally placed. Since this is a binary tree, leaves placed at depths that are better than optimal result in more leaves being placed at sub-optimal depths.

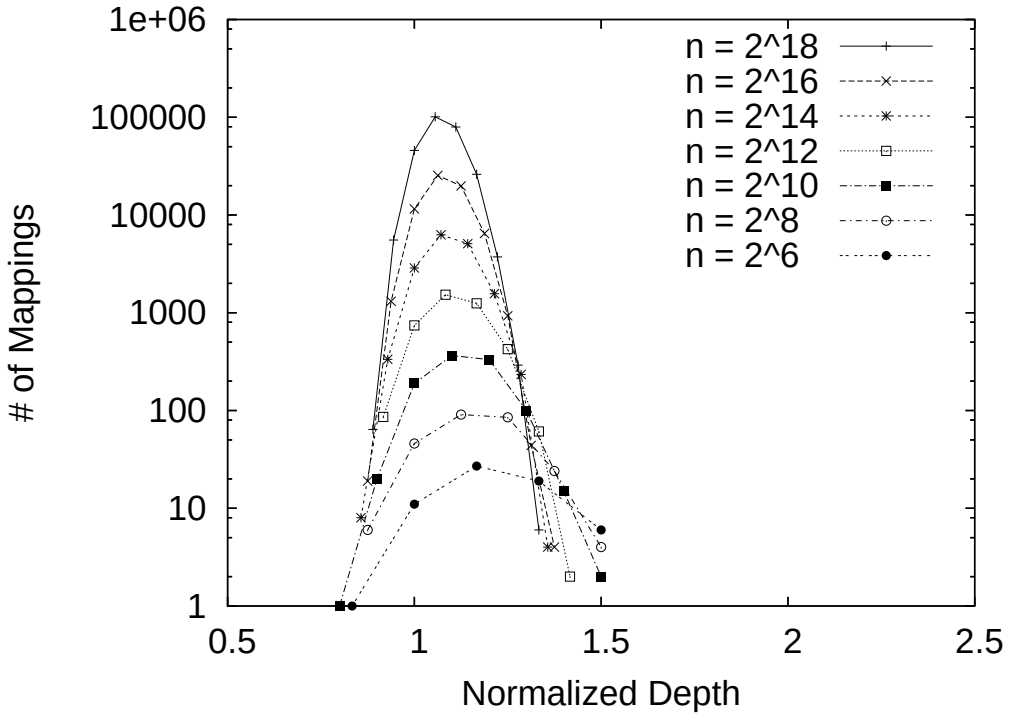

Figure 5: Normalized histogram trie leaf depths for various values of n, where n = total number of trie leaves

Figure 5 is a histogram of the normalized depths of the leaves from each of the sets we placed in the Fern

trie. As can be seen, for each set of names, the histogram peaks at slightly above a normalized depth of 1 and no histogram has any leaves with a normalized depth greater than 1.5.

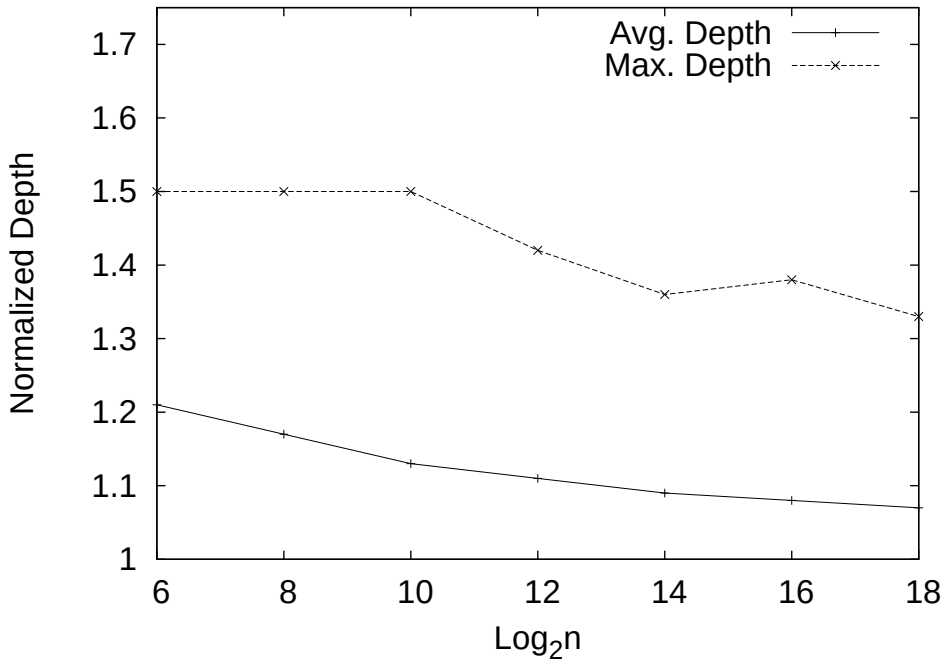

Figure 6: Comparison of worst case and average case normalized depths for various values of $\log_2 n$, where n = total number of trie leaves

To better illustrate the average and worst case results we obtained, we have included Figure 6, which shows how the average and worst case normalized depths vary with the size of the set of names we insert into the trie. In this figure, we see that for small sets of names, the average normalized depth is only slightly worse than 1.2 (it is in fact 1.21), whereas for larger, more typical sets, the average normalized depth drops below 1.1 (about 1.07). Both these numbers compare favorably with results obtained with trees that actively balance themselves. Furthermore, in our worst-case results (i.e. the leaves with the greatest normalized depths), the normalized depth is never worse than 1.5, whereas for larger sets the worst-case normalized depth is only slightly worse than 1.3 (i.e. 1.33). These numbers are also comparable with those obtained by more complex tree-balancing algorithms.

The forward edges in the trie provide path compression because they represent more than one bit in a conventional binary tree. Although they are not the major factor in providing short search paths, they do make a significant contribution eliminating long paths. In Figure 7 we repeat the earlier experiments, only this time without benefiting from forward edges. The expected longest path length is $2log_2 n$. As compared to Figure 5, we can see that the histograms are shifted to the right and are slightly broader, with longest paths approximating the expected length of $2\log_2 n$. Figure 8 compares the average normalized depth for the leaves of each of our experimental sets with and without path compression. Here it can be seen that the use of path compression reduces average search depth by approximately 5% for larger sets and 15% for smaller sets.

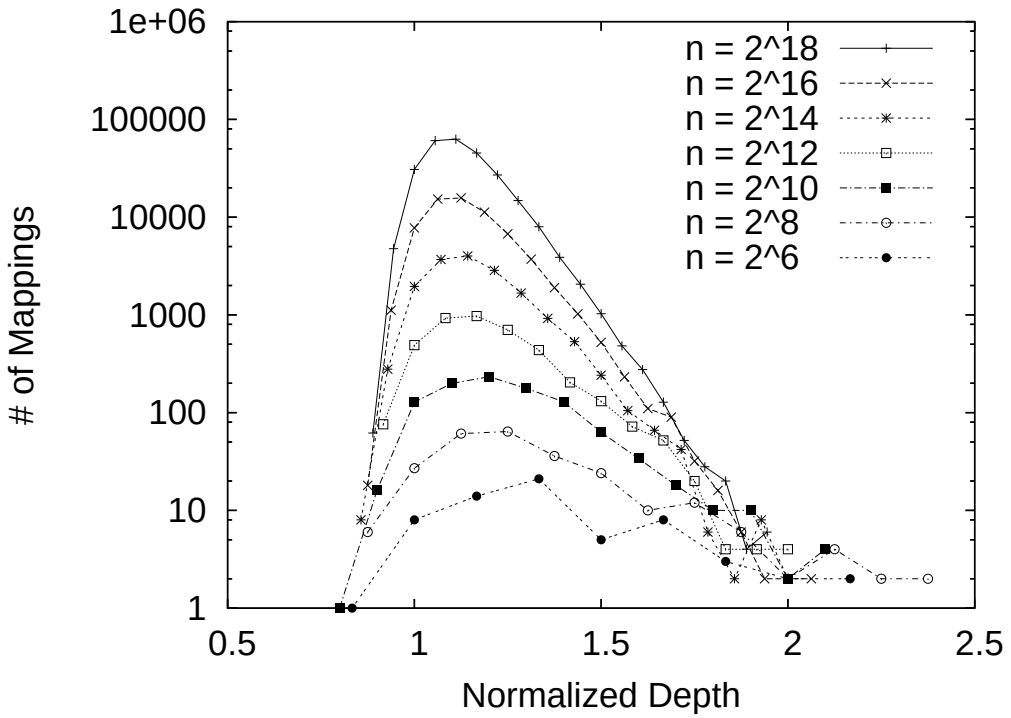

Figure 7: Normalized histogram of trie leaf depths without path compression for various values of n, where n = total number of trie leaves

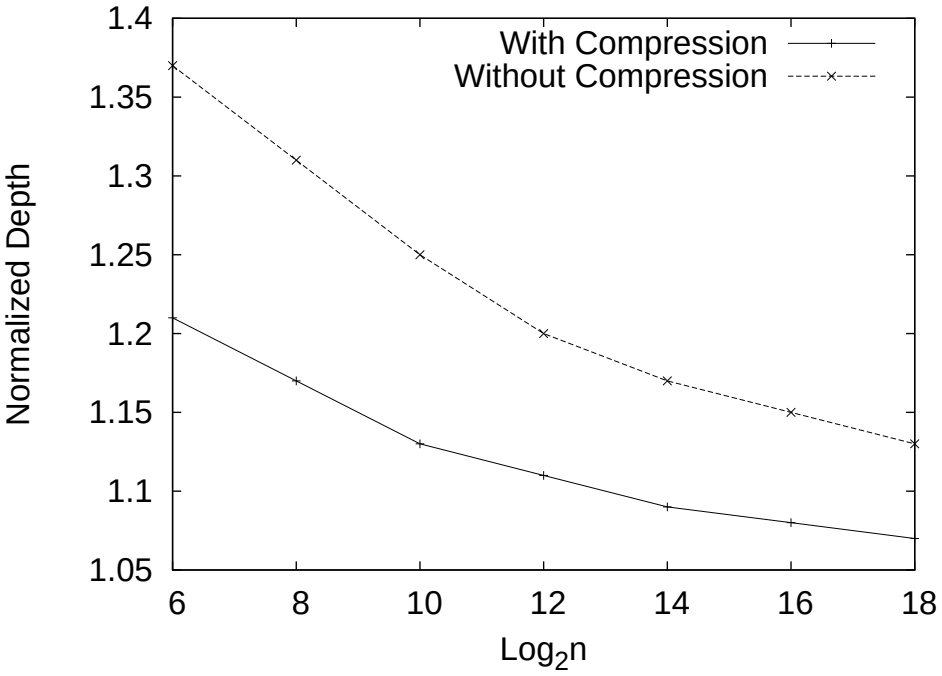

Figure 8: Comparison of worst case and average case normalized depths for various values of $\log_2 n$ without path compression, where n = total number of trie leaves

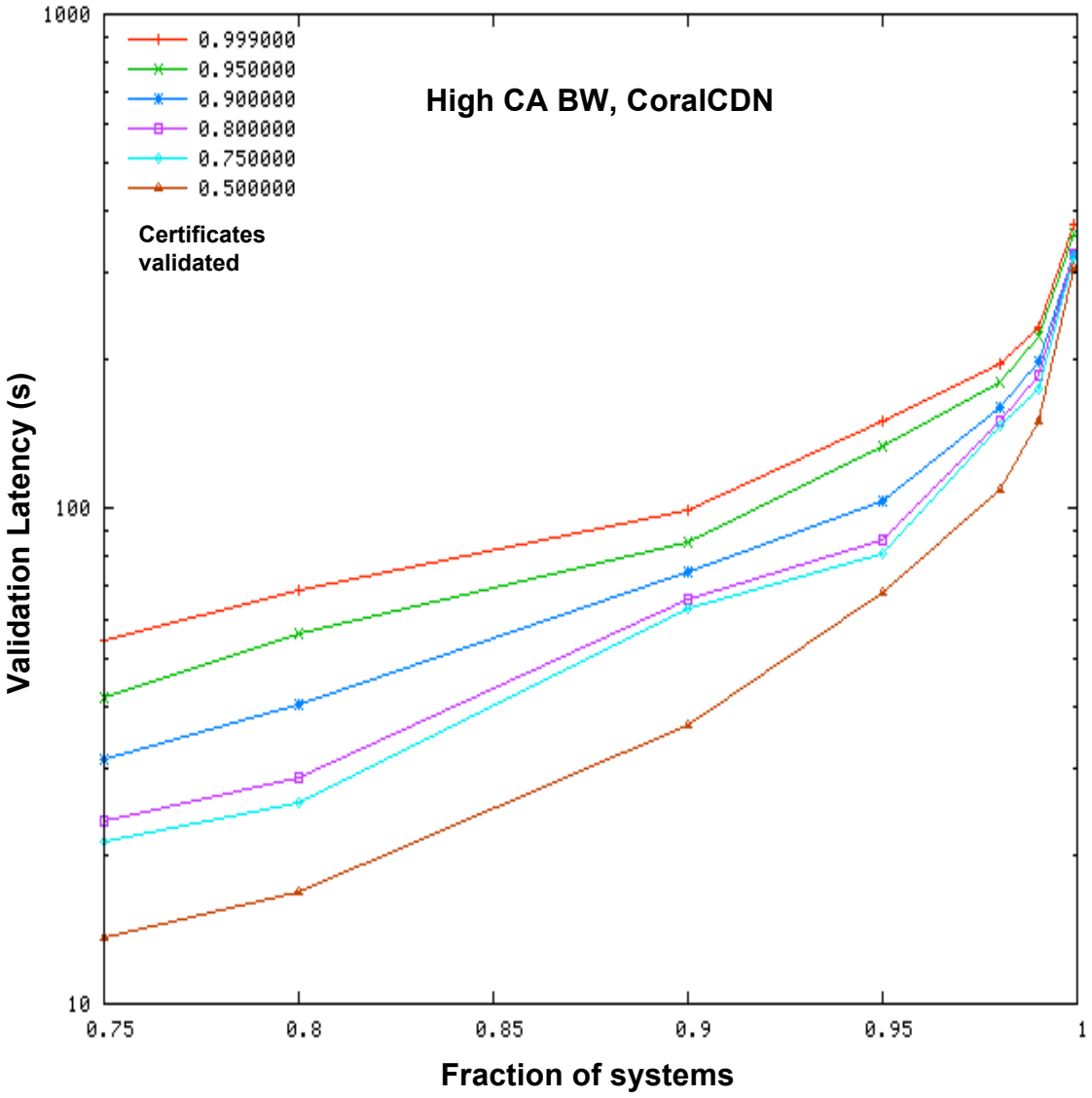

Figure 9: In-vivo experiment upon Planetlab. Originator has high-bandwidth connectivity.

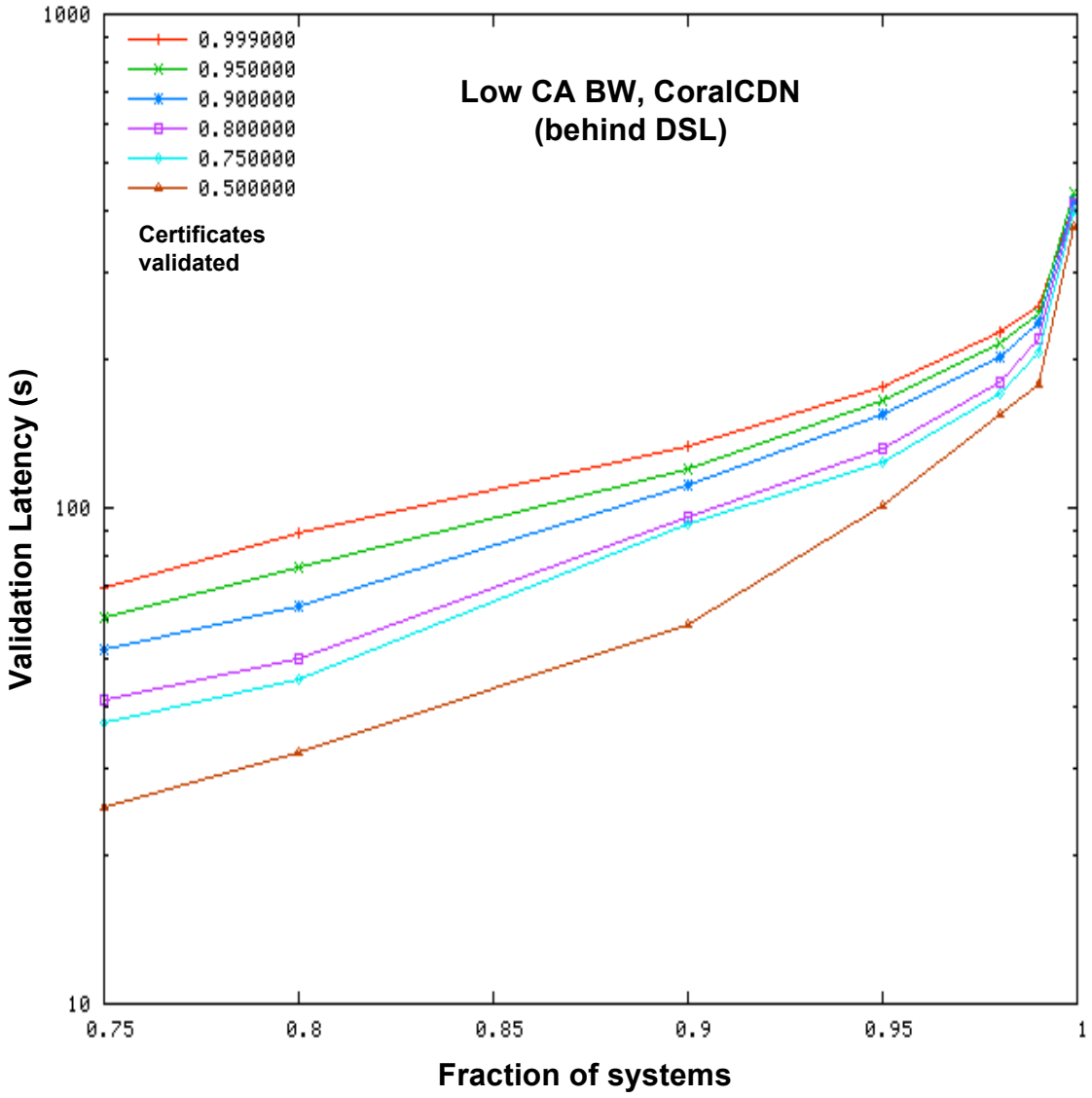

Figure 10: In-vivo experiments upon Planetlab. Originator has limited bandwidth.

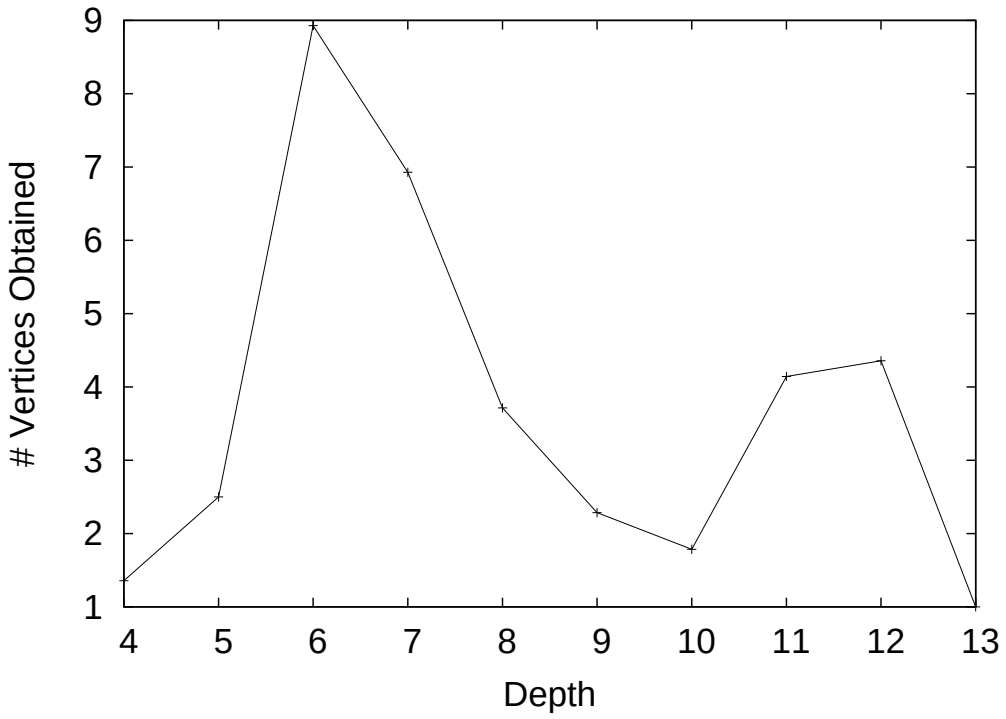

Figure 11: Average Depth of Refresh Queries. As predicted by the analytical model, refresh queries tend to terminate around depths $\log_2 u = 7$ and $\log n = 11$.

5.2 Online Experiments

We have begun evaluation of Fern upon the Planetlab[1] global distributed testbed. Figures 9 and 10 illustrate two experiments with approximately 700 hosts, each monitoring approximately sixty mappings chosen with distributions that mimic routing tables in a Kademlia[13] distributed hash table. Each time the dictionary was updated, two hundred randomly selected mappings were changed. Each plot in these graphs depicts the fraction of mappings obtained by each host. The vertical axis represents latency in seconds, the horizontal axis represents the fraction of systems. In the left chart, all hosts are connected via the open Internet. The right chart simulates a network partitioned by a bandwidth-limited link. The originator's server has severely limited bandwidth to any of the clients. Nonetheless, this experiments slightly increased overall latency despite the limited bandwidth indicates that the CDN successfully handled the large offered client load.

Figure 11 provides anecdotal evidence supporting our analytical model of refresh queries presented in Section 4.1. In this experiment, $k = 2058$, $w = 37$, $u = 130$. Clients are requesting a Pseudo-Kademlia distribution, plots are average for clients who obtained $c = 8$ updated nodes during this experiment. In these experiments, the average number of vertices obtained by clients is 154. Our analytical approximation prediction (Case 1) is 146. Surprisingly, this observed average number of refreshed vertices is within 2% of the modeled value.[4]

6 Synopsis

Fern is an updatable cryptographically authenticated dictionary. Unlike conventional approaches that require provisioning of proxies that maintain full duplicates of the dictionary structure at untrusted proxies, Fern utilizes an scalable and locality-aware autonomic content distribution network that incrementally distributes portions of the dictionary as required by clients.

Fern's use of an autonomic CDN makes it suitable for dynamic and Peer-to-Peer environments where dedicated resources may not be available. Furthermore, Fern can also be used to fortify the same CDN on which it relies against disruption from unauthorized systems.

A range of tree structures have been proposed for implementing updatable dictionary structures. While most of these algorithms have similar asymptotic behavior, the number of vertices that must be transferred to a client incrementally looking up a mapping stored within authenticated dictionary structures varies dramatically. Since the communication latency related to transferring nodes to clients will dominate the latency of client operations, these constants *do* matter. Our experiments indicate that Fern's randomized trie requires that substantially fewer vertices be transferred to clients than skiplists, and a comparable number with more complicated self-balancing trees.

Furthermore clients may need to monitor the stability of mappings (not) stored within an authenticated dictionary. Since Fern achieves an approximately balanced structure without rebalancing, clients can efficiently utilize caching of previously obtained vertices, thus minimizing the number of vertices that must be transferred when a dictionary update is distributed.

Initial experiments conducted upon the Planetlab[1] global distributed testbed indicate that the CoralCDN provides similar performance to a large number of clients who have a limited bandwidth to a Fern originator. In addition, our analysis of the number of vertices that must be transferred to a client monitoring a set of mappings stored within a Fern trie approximates observed operation.

Acknowledgment

This work was supported in part by the U.S. Army Research Laboratory, Survivability/Lethality Analysis Directorate, Information and Electronic Protection Division, Information Warfare Branch.

[4] We have not yet performed a sensitivity study on this result, but will do so prior to the author notification date and include it in the final version of this report.

References

[1] *Planetlab: An open platform for developing, deploying, and accessing planetary-scale services.* http://planet-lab.org.

[2] Anagnostopoulos, A., Goodrich, M. T., and Tamassia, R. Persistent authenticated dictionaries and their applications. In *Lecture Notes in Computer Science, Proc. Information Security Conference (ISC)* (2001), vol. 2200, Springer-Verlag, pp. 373–393.

[3] Bayer, R. Symmetric binary b-trees: Data structure and maintenance algorithms. *Acta Inf. 1* (1972), 290–306.

[4] Ellison, C. M., Frantz, B., Lampson, B., Rivest, R., Thomas, B. M., and Ylonen, T. SPKI Certificate Theory. IETF RFC 2693, 1998.

[5] Freedman, M., Freudenthal, E., and Mazieres, D. Democratizing content publication with coral. In *Proc. NSDI 2004* (2004).

[6] Fu, K., Kaashoek, M. F., and Mazières, D. Fast and secure distributed read-only file system. *Computer Systems 20*, 1 (2002), 1–24.

[7] Garfinkel, S., and Spafford, G. *Practical UNIX and Internet Security.* O'Reilly and Associates, Inc., 1996.

[8] Goodrich, M., and Tamassia, R. Efficient authenticated dictionaries with skip lists and commutative hashing, 2000.

[9] Goodrich, M., Tammassia, R., and Schwerin, A. Implementation of an authenticated dictionary with skip lists and commutative hashing. In *DARPA Information Survivability Conference and Exposition II (DISCEX II)* (2001).

[10] Goodrich, M. T., Shin, M., Tamassia, R., and Winsborough, W. H. Authenticated dictionaries for fresh attribute credentials. In *iTrust* (2003), P. Nixon and S. Terzis, Eds., vol. 2692 of *Lecture Notes in Computer Science*, Springer, pp. 332–347.

[11] Li, N., Mitchell, J. C., and Winsborough, W. H. Design of a role-based trust management framework. In *Proc. 2002 IEEE Symposium on Security and Privacy* (May 2002), IEEE Computer Society Press, pp. 114–130.

[12] Martel, C., Nuckolls, G., Devanbu, P., Gertz, M., Kwong, A., and Stubblebine, S. A general model for authenticated data structures, 2001.

[13] Maymounkov, P., and Mazieres, D. Kademlia: A peer-to-peer information system based on the xor metric. In *Proc. IPTPS02* (2002).

[14] Merkle, R. C. A certified digital signature. In *Advances in Cryptology – CRYPTO '89*, G. Brassard, Ed., vol. 435. Springer-Verlag, 1990, pp. 218–238.

[15] Myers, M., Ankney, R., Malpani, A., Galperin, S., and Adams, C. X.509 Internet Public Key Infrastructure Online Certicate Status Protocol. IETF RFC 2560, 1996.

[16] Nievergelt, J., and Reingold, E. M. Binary search trees of bounded balance. *SIAM J. Comp. 2* (1973), 33–43.

[17] Pugh, W. Skip lists: A probabilistic alternative to balanced trees. In *Workshop on Algorithms and Data Structures* (1989), pp. 437–449.

[18] Rivest, R. L., and Lampson, B. SDSI – A simple distributed security infrastructure. In *Proc. of CRYPTO'96* (1996).

[19] Tamassia, R., and Triandopoulos, N. Efficient content authentication over distributed hash tables. Tech. rep., Brown University, 2005.