Reliable Resource Searching in P2P Networks*

Michael T. Goodrich[1], Jonathan Z. Sun[2],
Roberto Tamassia[3], and Nikos Triandopoulos[3,4]

[1] Dept. of Computer Science, U. California, Irvine, USA
[2] School of Computing, Univ. of Southern Mississippi, USA
[3] Dept. of Computer Science, Brown University, USA
[4] Dept. of Computer Science, Boston University, USA

Abstract. We study the problem of securely searching for resources in p2p networks where a constant fraction of the peers may act maliciously. We present two novel hashing-based schemes that can be employed to reliably support *resource location* and *content retrieval* queries, limiting the ability of adversarial nodes to carry out attacks. Our schemes achieve scalability and load balancing and have small authentication overhead. In particular, for a network with n peers, resources are securely located with $O(\log^2 n)$ messages and content from a collection of m data items is securely retrieved with $O(\log n \log m)$ messages.

Key words: peer-to-peer, overlay networks, distributed hash tables, one-way hash functions, digital signatures

1 Introduction

An overlay peer-to-peer (p2p) network is a distributed structure imposed on a set of machines, called nodes or peers, for sharing data and computing resources. A p2p network can achieve load balancing and scalability by allowing peers to efficiently join and leave the network and users to efficiently store and retrieve data content. Data storage is typically supported by realizing a *distributed hash table* (DHT) that exports a basic `put`/`get` API. A data item can be inserted into the DHT with a `put` operation under a key and can be retrieved from the DHT with a `get` operation given its key. At a lower level, any resource is mapped to some peer that is responsible for this resource and can be efficiently located.

In this paper, we study the problem of *verifying* the resource searching functionality in a p2p network in the presence of faulty or malicious nodes. While faulty nodes are trouble enough, adversarial p2p nodes—a considerable threat since most p2p systems do not impose any restrictions on membership in the network—can be especially troublesome. For example, a coalition of adversarial nodes may wish to degrade the network performance by falsifying responses to redirection queries during resource location. Alternatively, nodes responsible for

* Work supported in part by NSF grants 0713046, 0713403, and 0724806, the RISCS Center at Boston University and the Center for Geometric Computing at Brown University.

some stored data may respond with content that appears to be a file of interest, but is in fact of degraded quality, virus infected, or outdated. Even more insidiously, adversaries may collude to systematically misdirect queries to a "parallel" p2p network that has invalid content. Finally, a group of nodes may mimic normal behavior and only aim at taking control over a small set of target items, by maliciously subverting resource locations or falsifying content retrievals.

To defend against such attacks, we are interested in designing techniques that protect the integrity of *resource location* at the network level and of *content retrieval* at the application level. That is, we wish to authenticate the p2p routing paths (followed by the distributed location process) as well as the p2p content (returned through a `get` operation over a DHT). With respect to routing protection, we aim at defending against *shunting* attacks, where adversaries misdirect queries and updates to malicious nodes. With respect to content protection, we would like to detect *content forgery*, where invalid data is returned, and *replay attacks*, where out-of-date content is retrieved.

We consider an adversary that inserts corrupted machines into the system and controls their behavior, stored content and routing information. However, we assume that at all times the adversary controls a constant fraction of the participating peers—$\frac{1}{4}$ in our case, which is reasonable for any large-scale p2p network. Thus, we do not consider denial-of-service attacks. Our goal is to design *crypto-enhanced* schemes that employ lightweight cryptographic primitives, such as collision-resistant hashing. When malicious behavior is sporadic or selective, we additionally wish that if no attack is in place, our verification mechanisms asymptotically incur no extra overhead. To the best of our knowledge no existing work has this *mode-adaptability* property for crypto-based secure routing. Applying standard techniques for content authentication results in solutions that do not achieve either scalability or load balancing. For instance, storing signed items results in a linear-size overhead as all items must be resigned to prevent reply attacks. Also, hash-tree based schemes introduce "hot-spots" in the system, as nodes storing hash values close to the root are more heavily accessed.

We assume the existence of a PKI where the public keys of the users (not necessarily peers of the p2p network) publishing data into the DHT are known to all parties. In addition, for an owner of a data collection of size m we assume the availability of a public and reliable storage of size $O(\log m)$ (e.g., a web page managed by the owner). Thus far, existing works have addressed the two problems in isolation, solely providing either route or content verification.

Related work. Early work on *secure routing* [3] in p2p networks tolerates certain attacks by assigning verifiable identifiers to network nodes. Numerous DHTs (e.g., [14, 17]) have been shown to tolerate random failures. Other schemes (e.g., [10, 16]) have been designed to deter misbehavior in adversarial models, and in particular some schemes (e.g., [2, 5, 6]) use *quorum-based* approaches, where regular network nodes correspond to large random blocks of machines and faulty behavior is prevented through majority-voting. Schemes using *redundancy* in searching (e.g., [9, 11]) have also been proposed to tolerate random Byzantine behaviors. Most p2p storage systems (e.g., [4, 15]) support content authentica-

Table 1. Qualitative comparison of our authentication schemes with other approaches.

path verification	secure routing	quorum-based	redundancy	**scheme 1**
attack-resistant	•	•	•	•
crypto-enhanced	–	–	–	•
mode-adaptable	•	–	–	•

content verification	self-certified	sign-all	DMT	**scheme 2**
dynamic	–	•	•	•
replay-safe	N/A	–	•	•
load-balanced	–	•	–	•

tion using *"sign-all"* techniques where data items are each individually signed. Signature amortization (i.e., signing a single digest) is used in some systems by storing the so-called *self-certified* data [7], but only for large individual items in static data sets. A *distributed Merkle tree (DMT)* is presented in [18]; realizing a p2p extension of Merkle's hash tree, this scheme lacks load-balancing.

Our contributions. We present two new authentication schemes for efficiently and securely verifying resource location and content retrieval operations, respectively. Our schemes are based on corresponding novel *hashing schemes*, which constitute extensions of hash trees to general directed acyclic graphs (DAGs).

Our first authentication scheme, called *skip-DHT*, is based on *skip graphs* [1]. Its hashing scheme embeds a set of DAGs in a skip graph so that source-to-sink paths in each DAG correspond to search paths in the skip graph. Our construction efficiently authenticates all possible search paths that can be used for resource location. By combining this scheme with *quorum-based* techniques (e.g., [2]), an n-node skip-DHT supports resource locations that are cryptographically verifiable with $O(\log n)$ messages in $O(\log n)$ time, has near-optimal query complexity in the absence of faulty nodes and is attack-resistant in the presence of a constant fraction $\leq \frac{1}{4}$ of adversarial nodes using $O(\log^2 n)$ messages.

Our second authentication scheme is a middleware component that can operate on top of *any* DHT to efficiently verify `put`/`get` operations on a data set of size m owned by a given data source. We define a hashing scheme of high expansion over the data items and we store this structure in the DHT so that retrieved content can be associated with many equivalent verification hash paths, linking each item to one of $O(\log m)$ publicly available digests that are signed by the data source. As paths can be retrieved with uniform workload, we obtain a distributed implementation of Merkle's hash tree where load balancing is preserved. Using certain algorithmic techniques, item insertions and delayed item deletions can be supported with $O(\log n \log m)$ amortized time and communication overheads.

Combined together, our two schemes yield a new distributed hash table with certain unique features: (*i*) it is the first DHT to provide cryptographic security guarantees for both resource location and content retrieval; (*ii*) it has near optimal searching performance in the absence of adversarial nodes; (*iii*) it achieves both scalability and load balancing. Table 1 compares our work with previous approaches for path and content verification.

2 Resource location authentication

In this section, we describe a new attack-resistant DHT that is based on the structure induced by skip graphs [1]. To authenticate the search paths in the DHT, we design a hashing scheme that is defined over its graph structure.

A skip graph on a set K of keys is a distributed structure that supports operation $\mathsf{suc}(k)$, returning the smallest key $k' \in K$ such that $k' \geq k$. Although designed for the purpose of supporting order-based queries, skip graphs provide a natural method for support $\mathsf{put}/\mathsf{get}$ operations. The structure of a skip graph can be viewed as a distributed extension of skip lists [13]. Both skip lists and skip graphs consist of a set of increasingly sparse, sorted, doubly-linked lists ordered by levels starting at level 0, where membership of a particular key $k \in K$ in a list at level i is determined by the first i bits of a potentially infinite sequence of random bits associated with k, referred to as the *membership vector of* k, and denoted by $m(k)$. We denote the first i bits of $m(k)$ by $m(k)|i$. In the case of skip lists, level i has only one list, for each i, which contains all keys k such that $m(k)|i = 1^i$, i.e., all keys whose first i coin flips all came up heads. As this leads to a bottleneck at the single node present in the uppermost list, skip *graphs* have 2^i lists at level i, which we will index from 0 to $2^i - 1$. Key k belongs to the jth list of level i if and only if $m(k)|i$ corresponds to the binary representation of j. Hence, each key is present in one list of every level until it eventually becomes the only member of a singleton list. The set of all lists to which a particular key k belongs meets the definition of a skip list, with membership in level i determined by comparison to $m(k)|i$ rather than to 1^i. We refer to such a skip list as the skip list defined by $m(k)$ and denote it by $SL(m(k))$ or $SL(k)$.

To search for the successor of k' we begin from the sparsest list at, say, level x and traverse the list by pointers to the right as far as possible without moving to a node whose key is greater than the key sought; we proceed downward to the list at level $x - 1$ in the same way, until level 0 is reached, where we move rightward to get the result. In a distributed setting, we map each graph node at level i in to a network node, so that each node stores only four links: pointers L and R for the network addresses of the machines assigned to the nodes of the list immediately before and after the given node, and pointers U and D for the machines responsible for the same key at levels $i+1$ and $i-1$. To search for the successor of key k', a machine mapped to key k performs a search in the skip list $SL(k)$ defined by $m(k)$. If $k < k'$, a rightward search is performed beginning at the top-most list of $SL(k)$, which we refer to as the root of k and denote by $r(k)$. This root list must contain key k and hence can be reached by following the U pointers of the lower level nodes of k. Based on the analysis in [13] it can be shown that queries in a skip graph take $O(\log n)$ time and messages.

Search-path hashing scheme. We present a hashing scheme consisting of a collection of DAGs embedded in the skip graph so that the authenticity of any root-to-leaf search path may be verified by the querier. Overall, our hashing scheme is an extension of the one used in [8]. Let h be a cryptographic collision-resistant hash function, and let $h(a, b) \triangleq h(h(a)\|h(b))$. Let v be a level-i node in a skip list $SL(k)$, with neighboring nodes $w = R(v)$ and $u = D(v)$, and denote

by $d(v)$ the digest of v. Node w is called a *plateau* node if its key does not appear at level $i+1$ in $SL(k)$ or a *tower* node otherwise. Then skip list $SL(k)$ is hashed as follows: If v is at level 0, then $d(v) = h(ID(v), ID(w))$ if w is a tower node or $d(v) = h(ID(v), d(w))$ if w is a plateau node; if v is not at level 0, then $d(v) = d(u)$ if w is a tower node or $d(v) = h(d(u), d(w))$ if w is a plateau node. All these digests can be computed efficiently.

Lemma 1. *If L is a distributed skip list with n nodes with a longest search path of size H and where each node maps to a machine, then the digests of L can be computed by respective nodes in H rounds using at most three messages per node (sent plus received) and $n-1$ messages in total.*

Given the true digest of a root node digest in the above hashing scheme, a querier is able to verify the value returned by the node at the end of the search path, since every search in a skip graph is also a search in a skip list. Thus, if the digests of every skip list in the skip graph were computed and stored at those nodes, the searches could be verified given the true value of the root node's digest. On the surface, this seems an unsatisfactory solution, as nodes are present in as many as $\frac{n}{\log n}$ different skip lists, and hence would seem to need to store an equally large number of digests. However, consider a level-i list L in a skip graph corresponding to membership prefix s. Suppose the above hashing scheme is applied to two skip lists, which have membership prefix sb, $b \in 0, 1$. Then the digests of all nodes in L are identical between the two skip lists. Therefore, it turns out that each node takes on only two distinct digest values, one for those skip lists in which the node is a plateau and one for those in which it is not.

Lemma 2. *A skip graph can be hashed to authenticate the search path of each membership query, with each node maintaining two digests. The two digest values at each node can be computed using $O(1)$ messages per node and $O(n)$ messages for all nodes. With high probability, this process takes $O(\log n)$ rounds.*

Quorum-based extension. To make this path authentication scheme resilient to shunting attacks, we need to satisfy the following requirements: 1) the digests are computed correctly at each node and passed correctly to neighboring nodes, and 2) the querier machine knows the true value of the root node's digest. We observe that the only machines that begin a query from a particular root node v are those machines assigned to v or some node directly below v. As such, the hashing algorithm should pass each root digest down to the nodes directly beneath it. We therefore need a message-passing scheme that is resistant to adversarial tampering in order to satisfy these both requirements.

We consider the quorum-based extension (e.g., as in [2]) of our skip-graph, where each skip-graph node corresponds to a *supernode* consisting of $\Theta(\log n)$ machines. Members of a supernode are completely connected, forming a clique, edges between supernodes correspond to complete sets of edges between their members, and data mapped to a supernode is stored by all of its members. Doing so increases the degree of each machine and the number of stored keys by a factor of $O(\log n)$. For a constant fraction of adversarial nodes, if it is guaranteed

that each supernode contains a random subset of machines, then, with high probability, every supernode will consist of a majority of honest machines.

In this redundant DHT, reliable search and update operations can be performed using a voting scheme in which each step in the traversal of the skip graph is verified by requesting the correct local answer from every machine in the current supernode. Also, when a search reaches the supernode responsible for the key sought, all of its members are polled to determine what data, if any, matches the search key. Our scheme employs this polling-based search whenever the path-verification protocol indicates an error in the resource location execution. Furthermore, we use voting-based computations to ensure that the digests are correctly reported to neighboring supernodes and the root digests are correctly reported to the supernodes beneath the root. This increases the asymptotic message costs by a factor of $\log^2 n$, as each node-to-node communication because hash updates now uses $O(\log^2 n)$ messages.

Updates. As described, our authentication scheme supports secure resource locations in a static collection of keys, since the digests of all root nodes need to change when a key is added or removed from the skip graph. We regain efficient support for updates as follows. We assume that the fraction of bad nodes is less than $\frac{1}{4}$ and use the construction of [2] to assign node identifiers from the interval $[0, 1)$ to the machines in such a way that w.h.p. every interval of length $\frac{c \log n}{n}$ contains a $\frac{3}{4}$-majority-good supernode of $\Theta(\log n)$ machines. We construct a skip graph whose keys consist of the node identifiers of the smallest member of each supernode. Data items are stored to supernodes through a pseudo-random hash function mapping arbitrary strings to $[0, 1)$ and then to the closest supernode identifier. Thus, exact searches for data items are supported by searching for the hash of the desired key. We refer to this structure as a *skip-DHT*.

We use the skip-graph hashing scheme to certify query results in the skip-DHT in a way that avoids recomputing the digests after every key or machine update. We use the machine identifiers in a supernode as the data that will be hashed as the digest of leaf nodes. Therefore, data updates no longer must yield an update in the root digests, since supernode membership is verified instead—we do not need to know the current list of machines in a supernode to have confidence in the query results. Instead, we rely upon knowing that the majority of the remaining original machines can be trusted. This is true as long as less than $\frac{1}{2}$ of the original nodes have left the network.

Overall, when a resource location query is executed, a *non-redundant* search is carried out given the query resource key—that is, a pointer to a single, arbitrary member of the next node in the search path is requested from a (single, arbitrary) member of the current node. When a destination machine is reached that claims to be a member of the supernode nearest the search key, it must provide a list of the identifiers of the original members of the supernode to which it belongs. The querier then computes the hash of this list and checks it against the verification path. This verifies the successor supernode in the skip list, thus the correct resource location. Each of these steps requires $O(\log n)$ messages.

Theorem 1. *An n-node skip-DHT satisfies the following properties:* (1) *In the absence of faulty nodes, verifiable exact-match queries are executed with $O(\log n)$ messages in $O(\log n)$ time;* (2) *In the presence of a constant fraction of adversarial nodes, queries are answered and securely verified with $O(\log^2 n)$ messages in $O(\log n)$ time;* (3) *The hashing scheme adds only a constant number of messages to amortized bandwidth usage for adding and removing machines.*

3 Content retrieval authentication

In this section, we study the problem of authenticating content at the application level through the put/get core functionality of any DHT. Our goal is to design a distributed scheme that verifies that data items claimed to have been added by a data source were really put in the DHT by this entity and have not been modified by malicious nodes. We wish this scheme to achieve *load balancing*, that is, to evenly distribute the workload related to authentication across the network nodes. We consider a standard query model where an underlying DHT stores key-value pairs of the type (k, x), each added through operation $\mathsf{put}(k, x)$, where keys are unique identifiers and values are associated with keys. We assume that the DHT supports operation $\mathsf{get}(k)$, which returns the value associated with key k, with $O(\log n)$ expected time and message costs.

For simplicity, we assume that a single data source is storing items in the system; for more data sources, we make use multiple invocations of our scheme. We assume that the public key of each data source storing data in the DHT is known to any entity querying the DHT. Also, we assume the availability of some *public reliable storage* that is associated to a given data source and that can be easily accessed and updated independently of the underlying DHT. The size of this information is only *logarithmic* in the number of data items published by the source. In practice, this assumption is easily implementable through a web service that posts to a web-site a small amount of data regarding a data source.

Load-balanced hashing scheme. Our data structure achieves signature amortization by applying a hashing scheme over the data items stored in the DHT. The main idea in our construction is to use a hashing scheme G of high expansion rate, namely with a structure that resembles the FFT computation graph or a butterfly network, such that for any data item, there exist many equivalent verification paths. We distribute DAG G to the network nodes of the underlying DHT by appropriately indexing the digests and storing them as special data items. We preserve the structure of the hashing scheme G in the DHT as follows: the network node storing the digest of node v in G also stores the keys under which the digests of the immediate successors and predecessors of v in G are stored in the DHT. We then *randomize* the generation of the verification paths to achieve a uniform workload over the visited network nodes.

We describe our hashing scheme G for m data items and its embedding into an n-node DHT. For simplicity and without loss of generality, we assume that $m = 2^k$. The nodes of G are partitioned into $k + 1$ levels, each having m nodes.

The nodes at level 0 are source DAG nodes, each associated with a data item. Each of the nodes at one of the remaining levels has two predecessors nodes at the previous level. The edges in G are defined so that the nodes at level k are the roots of m perfect binary trees over the data set. More formally, let us number the nodes on each level and denote with $v_{i,j}$ the j-th node of G on level i, $i = 0, \ldots, k$, $j = 0, \ldots, m-1$. For $i > 0$, node $v_{i,j}$ has two incoming edges from nodes $v_{i-1,j}$ and $v_{i-1,j+\delta(i,j)}$, where $\delta(i,j) = (-1)^{\lfloor j/2^{i-1} \rfloor} 2^{i-1}$. Let h be a cryptographic collision-resistant hash function. For $i = 0$, we set $d(v_{i,j}) = h(k||x)$, where (k,x) is the data item associated with $v_{i,j}$. For $i > 0$, we set $d(v_{i,j}) = h(d(v_{i-1,j}) \;||\; d(v_{i-1,j+\delta(i,j)}))$. By symmetry, the nodes of G at level i store 2^{k-i} distinct digests. The data source signs the single digest stored at nodes of level k and makes it available as public information. Then, each DAG node $v_{i,j}$ is indexed by a unique identifier $id_{i,j}$, where in particular node $v_{0,j}$ that is associated with data item (k,x) is indexed by k, and is inserted in the DHT as a special data item, using $id_{i,j}$ as the key and the digest and identifiers of its predecessors and successors in G ($O(1)$ information) as the value.

Query and verification. We now describe how `get` operations are handled. We begin by performing a query according to the underlying DHT structure (e.g., as discussed in the previous section). Given that data item (k,x) stored at network node W is located by the DHT, node W initiates a randomized process for generating a verification path for (k,x). Namely, W flips a coin to determine which of its two parents at level 1 (next node in the path) to contact next (through a resource location operation, first). In general, a network node V at level j randomly chooses the next network node (to be contacted while forming the verification path) independently and with probability $\frac{1}{2}$. Thus, any query results in a verification path of length $O(\log m)$, using $O(\log m)$ location operations, with $O(\log m \log n)$ computation and communication costs. Through the randomized search process, every verification path for a fixed data item is actually an independent and identically distributed random variable and no hot-spots are created while accessing the authentication structure. The verification path is returned by the DHT and given this, one can authenticate the answer of operation `get` by processing the digests contained in the path, verifying the publicly available signed digest and checking their consistency. The total storage required is $O(m \log m)$; that is, assuming perfect mapping functions from keys to network nodes (usually through a cryptographic hash function), the storage is logarithmic in n per network node, when $m = O(n)$—i.e., still optimal, since most DHTs use routing tables of logarithmic size. Using a caching technique as in [18], we can further improve the creation of the verification paths.

Updates. To support updates, we modify the scheme described above using a dynamization technique due to Overmars [12], which allows to transform a static data structure into a corresponding dynamic structure. The idea is to partition a data set of size m into sequence of $O(\log m)$ *blocks*, where the size of each block is twice the size of the previous block, and to completely rebuilt blocks after updates, as necessary. We apply this technique to support insertions of data items with new keys. Let D be a data set of size m and let $(b_k, b_{k-1}, \ldots, b_1, b_0)_2$

be number m written in binary, with $b_k = 1$. Note that items in D are not assumed to be sorted. We partition D into $\lfloor \log m \rfloor + 1$ blocks $B_0, B_1, \ldots, B_k$, each a subset of D, according to the weights of the bits of m, i.e., $|B_i| = b_i \cdot 2^i$. Let then $G(i)$, $0 \leq i \leq k$, denote the hashing DAG described in previous section that is built for the items of block B_i. DAG $G(i)$ has $b_i \cdot 2^i \cdot i$ nodes. DAGs $G(0), G(1), \ldots, G(k)$ are used separately as authentication structures: that is, for $i = 0, \ldots, k$, if $b_i = 1$, the source signs the top-level digest h_i of DAG $G(i)$ and each $G(i)$ is distributed over the network nodes as before. For any queried data item in block B_i, the corresponding verification path in $G(i)$ is retrieved using $O(i)$ location operations. Thus, $O(\log m)$ signed time-stamped digests (one for each block) are made available as public information.

We perform insertions of data items through operations `put` as follows. Let i be the smallest i such that $b_i = 0$ or $i = k+1$ if no such i exists. To insert an item x into D, we merge DAGs $G(0), G(1), \ldots, G(i-1)$ to create DAG $G(i)$ for the new block $B_i = B_0 \cup \ldots \cup B_{i-1} \cup x$. Note that $|B_i| = 1 + \sum_{j=0}^{i-1} 2^j = 2^i$. The insertion of a data item into a set of size m stored into a DHT of size n takes $O(\log m \log n)$ expected amortized time. Accordingly, we update the public information: the data source creates new fresh time-stamps and re-signs the publicly available digests. This occurs for all blocks after every update of a block, independently of whether or not the corresponding block structure has been altered in the most recent update. Thus, at any point in time, we maintain $O(\log m)$ fresh signed digests as public information. At asymptotically no additional cost and using similar ideas with the verification of queries, the data source can verify the correctness of an operation `put` performed by the DHT: any change in the hashing scheme is checked for consistency with the $O(\log m)$ signed digests.

We can also support *delayed deletions*, defined in our context as item removals that do not actually occur on-line, but instead occur at some future time and during the insertion of new items. Asymptotically, these deletions incur no additional communication or computational cost. In particular, we schedule the deletion of an item in block B_i during the construction phase of a new DAG $G(j)$, $j > i$, where j depends on the exact state of the authentication structure. This deletion procedure requires minor modifications to the above insertion algorithm. Replay attacks are eliminated by having the data source S performing controlled delayed deletions of items before they are replaced by new items in the system. Moreover, using delayed deletions, our structure supports data item expiration and content revocation: we remove expired or revoked items during the construction of some particular new DAG $G(j)$. In this case, our structure has the following important *self-correction* property that limits the window of opportunity for replay attacks: any expired or revoked item is automatically removed from the structure the first time that the corresponding block containing the item is restructured (rebuilt). Thus, the system supports item expiration/revocation in the sense that no old item can stay forever in the system; in particular, no item can be more than $m/2$ steps old, where m is the current number of items, and depending in the exact application, items can be scheduled to leave the storage system such that no replay-attacks can be launched by the DHT.

Theorem 2. *Given an n-node DHT where resource location has $O(\log n)$ expected time and message cost, there exists a distributed authentication scheme for verifying content from an m-item data set such that:* (1) *The scheme uses $O(\log m)$ public reliable storage and $O(m \log m)$ distributed storage;* (2) *Retrieved content is verified in $O(\log m)$ time with one signature and proofs of $O(\log m)$ size computed with $O(\log n \log m)$ expected time and message cost and with uniform workload over the DHT nodes;* (3) *Data-item insertions have $O(\log n \log m)$ expected amortized time and message cost;* (4) *The scheme is resilient to content forgery and replay attacks and supports delayed data-item deletions.*

References

1. J. Aspnes and G. Shah. Skip graphs. *SODA* 384–393, ACM 2003.
2. B. Awerbuch and C. Scheideler. Towards a scalable and robust DHT. *SPAA* 318–327, ACM 2006.
3. M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured P2P overlay networks. *OSDI* 299–314, ACM 2002.
4. F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. *SOSP* 202–215, ACM 2001.
5. A. Fiat and J. Saia. Censorship resistant peer-to-peer content addressable networks. *SODA* 94–103, ACM 2002.
6. A. Fiat, J. Saia, and M. Young. Making Chord robust to Byzantine attacks. *ESA* 803–814, Springer 2005.
7. K. Fu, M. F. Kaashoek, and D. Mazieres. Fast and secure distributed read-only file system. *Transactions on Computer Systems*, 20(1):1–24, ACM 2002.
8. M. T. Goodrich, R. Tamassia, and A. Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. *DISCEX* 02:1068, IEEE 2001.
9. A. Kapadia and N. Triandopoulos. Halo: High assurance locate for distributed hash tables. *NDSS* 61–79, Internet Society 2008.
10. K. Kothapalli and C. Scheideler. Supervised peer-to-peer systems. *I-SPAN* 188–193, IEEE 2005.
11. A. Nambiar and M. Wright. Salsa: a structured approach to large-scale anonymity. *CCS* 17–26, ACM 2006.
12. M. H. Overmars. *The Design of Dynamic Data Structures*, *LNCS* 156, Springer 1983.
13. W. Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.
14. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. *SIGCOMM* 161–172, ACM 2001.
15. S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: A public DHT service and its uses. *SIGCOMM* 73–84, ACM 2005.
16. J. Saia, A. Fiat, S. D. Gribble, A. R. Karlin, and S. Saroiu. Dynamically fault-tolerant content addressable networks. *IPTPS* 270–279, Springer 2002.
17. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable P2P lookup service for Internet applications. *SIGCOMM* 149–160, 2001.
18. R. Tamassia and N. Triandopoulos. Efficient content authentication in peer-to-peer networks. *ACNS* 354–372, Springer 2007.