TrickleDNS: Bootstrapping DNS Security using Social Trust

Sriram Sankararaman[1], Jay Chen[2], Lakshminarayanan Subramanian[2], Venugopalan Ramasubramanian[3]
Harvard University[1], New York University[2], Microsoft Research [3]
Email: sankararaman@gmail.com, {jchen,lakshmi}@cs.nyu.edu, rama@microsoft.com

***Abstract*—This paper presents TrickleDNS, a decentralized system for proactive dissemination of DNS data. Unlike prior solutions, which depend on the complete deployment of DNSSEC standard to preserve data integrity, TrickleDNS offers an incrementally deployable solution with a probabilistic guarantee on data integrity that becomes stronger as the adoption of DNSSEC increases. TrickleDNS provides resilience from data corruption attacks and denial of service attacks, including sybil attacks, using three key steps. First, TrickleDNS organizes participating nameservers into a well-connected peer-to-peer Secure Network of Nameservers (SNN) using two types of trust links: (a) strongly trusted social relationships across DNS servers (which exist today); (b) random yet constrained weak trust links between DNS servers, which it introduces. The SNN allows nameservers in the network to reliably broadcast their public-keys to each other without relying on a centralized PKI. Second, TrickleDNS *reliably binds domains* to their authoritative name servers through independent verification by multiple, randomly chosen peers within the SNN. Finally, TrickleDNS servers *proactively disseminate* self-certified versions of DNS records to provide faster performance, better availability, and improved security.**

I. Introduction

The Domain Name System (DNS) forms a critical component of the Internet infrastructure by providing the essential service of host name to IP address resolution. Internet users and providers of web-based services implicitly assume and rely on its correct operation, constant availability, and fast response times. However, DNS as operated today is susceptible to a wide range of attacks that can affect the integrity of name lookup and the availability of the DNS. Most prominently, malicious elements can hijack domain traffic by intercepting DNS requests and propagating bogus address mappings or make the domain unavailable by launching DoS attacks. In the past few years several massive DDoS attack have been launched on the root servers and specific domains to disrupt name resolution.

To improve DNS availability and performance, several research systems have proposed replacing or augmenting the DNS hierarchical name resolution process with a cooperative, peer-to-peer approach. These proposals rely on peer nodes as backup resolvers when the primary resolver fails (CoDNS [16]), or employ a full-fledged peer-to-peer overlay (Chord [22] and Pastry [20]) for routing DNS queries (DDNS [6], Overlook [24]), or advocate proactive dissemination of DNS records to servers organized in a peer-to-peer overlay (CoDoNS [18], Handley and Greenhalgh [9]). Even though such decentralized systems based on proactive dissemination can provide faster resolution of queries and resilience from failures and DoS attacks, their P2P approach inherently entails a large trusted computing base and makes them vulnerable to data integrity attacks. To preserve data integrity, the above systems typically invoke DNSSEC, which is unfortunately not supported by most domains including top-level domains despite many years of effort.

978-1-4673-0298-2/12/$31.00 © 2012 IEEE

This paper presents TrickleDNS, a hybrid system for proactive dissemination of DNS records. The primary objective of TrickleDNS is security while providing incremental deployability. TrickleDNS organizes participating servers into a distributed overlay network similar to existing P2P DNS systems [18], [6], and proactively pushes DNS records in a cooperative manner. However, TrickleDNS uses a decentralized security framework to protect against attacks instead of relying on DNSSEC or a centralized PKI. Thus leaf domains that are currently waiting for their parent domains to deploy DNSSEC can instead join the TrickleDNS network to securely disseminate their data. TrickleDNS provides probabilistic security guarantees for data integrity and DoS resilience; its guarantees become incrementally stronger when augmented by DNSSEC and as DNSSEC's deployment widens. TrickleDNS achieves these properties using the following mechanisms:

Limiting Sybil Identities: TrickleDNS leverages the SybilLimit protocol [26] to limit the number of sybil nodes ($O(\log n)$ per attack edge) allowed to participate in the system.

Decentralized Key Distribution: TrickleDNS establishes a Secure Nameserver Network *(SNN)* to facilitate fully decentralized public key distribution. The SNN allows TrickleDNS to distribute public keys even in the presence of a sizeable number of compromised servers ($O(\frac{n}{\log n})$ in an n server network).

Reliable Name Binding: TrickleDNS binds domains to their authoritative nameservers through independent verification by multiple, randomly-chosen servers in the network.

Proactive Dissemination of DNS Data: Finally, TrickleDNS servers push-out DNS records signed by their own public keys on the secure network in order to improve lookup performance and availability.

II. Background and Related Work

The Domain Name System (DNS) is a general-purpose database for mapping names from a globally unique name space to data resources associated with a name. It uses a hierarchical name space partitioned into

non-overlapping regions called *domains*. For example, *foo.bar.com* is a sub-domain of *bar.com*, which in turn is a sub-domain of the top-level domain *com*, which is under the global root domain. Resources for names within a domain are served by a set of nodes called the *authoritative name servers*. In addition to network addresses for host names, DNS resources, called *records*, could also include names of authoritative DNS servers, names of mail servers, or any small-sized data associated with the domain.

DNS uses a delegation based architecture for name resolution [13], [14]. A DNS *resolver* resolves names by following a chain of authoritative servers, starting from the root, followed by the top-level domain name servers, down to the servers of the queried domain. For example, the name *www.foo.bar.com* is resolved by following the authoritative severs of the parent domains *com*, *bar.com*, and *foo.bar.com*. DNS lookups could take a long time to follow the chain of servers in the hierarchy. To improve the lookup latency, DNS resolvers aggressively cache responses. Clients are typically configured with one or more resolvers from their local domain, through which they access DNS.

Several measurement studies have identified limitations in the performance and reliability of DNS. The multi-step, iterative process for query resolution adds critical latency to DNS lookups [10], [25], while progapagation of updates is delayed until records are expunged after the expiry of a predetermined lifetime (TTL), preventing fast relocation of services during emergencies. More criticallly, the low redundancy in nameservers leads to limited tolerance of failures and attacks; 80% of domain names are known to be served by just two name servers, while 32% of domain names have all name servers behind the same network gateway [15], [18]. Finally, the hierarchical structure make the root and top-level domains a frequent target of denial of service attacks [4], [5].

Several researchers have proposed to augment the DNS lookup process through alternative, complementary systems that provide better performance and resilience.

Centralized Solutions: Deegan *et al.* [7] propose to serve the entire DNS data from a single, centrally-managed repository [7]. While centrally managed systems can provide good performance and availability just like the DNS root servers today, they still problematically require trust to be placed on a single, centralized entity.

Peer-to-Peer DNS Solutions: Many proposals use the advantages of a peer-to-peer system for improving failure resilience, make query resolution faster, and proactively propagate updates. For instance, CoDNS [16] is a client-driven solution that uses a backup-set of resolvers from peer domains if the primary resolvers from the local domain are slow or unavailable. It uses a weak form of security based on majority consensus provided by ConfiDNS [17] to alleviate the risk of being misled by a malicious or compromised peer resolver.

Other peer-to-peer solutions use full-fledged overlay networks or Distributed Hash Tables (DHTs) for query resolution. DDNS [6] implements legacy DNS functionalities on top of Chord [22], while Overlook [24] is a new name service layered on top of the Pastry [20]. CoDoNS [18] provides low latency query resolution and update propagation through proactive, analysis-driven caching on DHTs, complementing other benefits of DHTs. These systems spread the responsibility of serving a domain uniformly across participating servers and efficiently transfer this responsibility to other servers during failures. Similarly, Handley et al. [9] propose an architecture to proactively push DNS records using a peer-to-peer overlay for high performance and failure resilience.

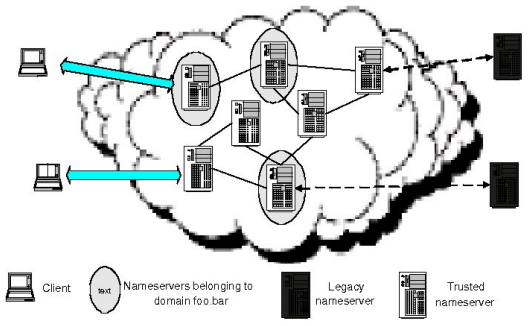

Fig. 1. TrickleDNS Architecture: A cloud of TrickleDNS nameservers provide reliable DNS lookup service to clients while interacting with legacy DNS servers in the background.

Presently, these proposals rely on DNSSEC [1], the prevalent security standard for DNS, to preserve data integrity.

DNSSEC: DNSSEC uses public-key cryptography to generate certificates and verify authenticity. Each record belonging to a domain has a certificate signed by the domain's private key, while its public key is disseminated through DNS as a key record. In order to prove ownership of its name space, the domain obtains a certificate from its parent domain, which consists of its key record signed by the parent's private key. Essentially, DNSSEC associates each domain with a chain of certificates signed by the centralized root of its parent domains. DNS clients are seeded with the public key of the root domain and can verify the integrity of records by following the chain of certificates.

Unfortunately, the acceptance of DNSSEC has been remarkably poor. Only a few top level domains (notably *.se*, and *.gov* until Dec 2009) support DNSSEC. Consequently, many domains that wish to secure itself are unable to use DNSSEC unless their parent domains adopt DNSSEC. Proposals to use alternative centralized certification authority such as OpenSSL (PKIs) (Fetzer et al. [8]) suffer from similar limitations.

III. TrickleDNS Overview

TrickleDNS is a cooperative peer-to-peer network of authoritative name servers of participating domains. A domain can join TrickleDNS even if its parent is not a participant. Each participating nameserver, called a *trusted nameserver* (TN), has two types of trust links with other participating TNs within TrickleDNS. Collectively, these TNs form a distribution network called the *Secure Nameserver Network (SNN)* (Figure 1) The two types of trust links within the SNN form two logical networks:

Social trust network: Two TNs that have a pre-established "social" relationship may establish a *social trust link* between themselves through an out-of-band channel. Social relationships between nameservers exist in the current

DNS for various reasons. (a) Nameservers belonging to the same administrative domain have a default trust relationship. (b) All authoritative nameservers serving a domain have a trust relationship between them. Since, some authoritative nameservers are often chosen from different domains for improving failure resilience (for example, *cornell.edu* has an authoritative nameserver in *cs.rochester.edu* domain [19]), the resulting relationships form a more elaborate trust network, transitively [19]. (c) Finally, nameservers of a domain have implicit relationships with the nameservers of the parent domain, which are trusted by default.

The above social trust network helps to prevent a large number malicious hosts from joining the TrickleDNS SNN. TrickleDNS applies the SybilLimit protocol on the social network to perform admission control on the set of accepted TNs within TrickleDNS. The SybilLimit protocol guarantees that the number of Sybil nodes that get accepted into the TrickleDNS network is bounded by O(log n) per attack edge where n is the number of nodes (not including Sybil nodes) and an attack edge is a social trust link between a compromised node and the network of honest nodes.

Reliable communication network: Given that the social trust network may not form a densely connected graph, TrickleDNS builds a more densely connected network for reliable communication. TrickleDNS TNs use the reliable communication network to distribute their public keys (via multiple dis-joint paths) and verify the authenticity of other TN's public keys in a completely decentralized manner. Our reliable communication protocol can tolerate up to $O(n/\log n)$ adversarial nodes within the reliable communication network using a constrained-randomization technique to establish links in the network, which we describe later.

TrickleDNS also uses this reliable communication network to proactively disseminate useful DNS records (NS-records and A-records) of participating domains to all servers in a secure and verifiable manner. Each TN pushes the DNS records of the domain it represents and additionally, also forwards the records it receives from its neighbors. The records are self-verifying and carry public-key-cryptographic signatures. This proactive push-based protocol provides better availability, lower query resolution times, and faster update propagation.

Name resolution in TrickleDNS: TrickleDNS supports the same query interface as existing DNS and interacts with non-participating nameservers through standard DNS protocols. A client can resolve a name, say *foo.bar*, using TrickleDNS as follows: First, a client's DNS servers are set to point to one or more trusted nameserver in the TrickleDNS network. These may be the nameservers of the client's local domain itself or other open-access TrickleDNS nameservers. A TrickleDNS namesever then handles the query for *foo.bar* in the following way: 1) If foo.bar is a participating domain, then the nameserver uses its locally cached NS and glue A records for the domain's authoritative nameservers, queries them, and fetches a signed response records from one of them. The TN can then verify the integrity of the returned data since it has the domain's public key. 2) If foo.bar is not a participating domain, then the nameserver executes a regular DNS recursive resolution process starting with the authoritative nameserver of the immediate participating parent domain.

IV. TrickleDNS Design

In this section, we describe the details of the TrickleDNS design.

A. Admission Control

TrickleDNS is boot-strapped with an initial set of name servers which act as bootstrap nodes.

The authenticity of a new domain that wishes to join TrickleDNS is validated using its social relationships with other participating domains. In the TrickleDNS network, the new domain's name servers have trusted links with name servers of the related domains. TrickleDNS then invokes the SybilLimit protocol to validate name servers in its network.

SybilLimit is a completely decentralized protocol that enables any honest node V (called the verifier) to decide whether or not to accept another node S (called the suspect). It assumes that the number of edges connecting the honest region (the region containing all the honest nodes) and the sybil region (the region containing all the sybil nodes), *attack edges* is small (even though the number of sybil nodes could be huge) since social relationships between DNS domains require human negotiations. While in the original SybilLimit protocol every node acts as its own verifier, in TrickleDNS only the bootstrap nodes are mandated to act as the verifier nodes in the protocol and maintain consistency via majority consensus.

A node is deemed to be *accepted* into the TrickleDNS network if a majority of bootstrap servers accept the node using the SybilLimit protocol. The SybilLimit protocol guarantees that if the honest region of the social network is fast mixing, then any honest verifier node can discover $(1-\epsilon)n$ honest nodes and at most learns $O(\log n)$ Sybil nodes per attack edge where ϵ is a very small quantity [26]. In Section VI, we show that the existing social trust network among DNS servers satisfies this fast mixing property. Using this protocol, we can guarantee that the bootstrap nodes can admit almost all honest nodes while admitting very few Sybil nodes. This protocol effectively prevents large botnets from easily infiltrating the TrickleDNS network.

B. Decentralized Key Distribution

Each Trusted Nameserver (TN) s generates a private-public key pair $(s.k, s.K)$ independently. We call the tuple $s.kid = (s.id, s.K, s.seq)$ the *keyed identity* of s, where $s.id$ is the id for s and $s.seq$ is a sequence number used to mark the latest public key. We chose the identifier of a TN to be a collision-resistent function of its IP address since IP address is the unit of identification of a nameserver in DNS. We discuss the implications of this choice for the identifier in Section V.

The immediate goal of the key distribution process is to ensure that each TN correctly learns the keyed identity of all other TNs.

1) Reliable Communication Network: A reliable way to distribute keyed identities to all TNs in the absence of a centralized certification authority is by forming a well-connected distribution network—a network with sufficient independent paths so that information communicated between TNs is resilient to malicious or compromised servers. It is well known that a network with at least $2k+1$ independent, vertex-disjoint paths between each pair of TNs can tolerate up to k malicious servers [23]. Two paths are vertex-disjoint if no intermediate server appears on both the paths.

Prior work [23] shows that random, peer-to-peer networks, where each server is connected to a fixed number D of other randomly chosen servers, provides an efficient way of building well-connected networks. More precisely, such a random network of neighbor degree D is guaranteed to have at least D vertex-disjoint paths between any two servers with high probability. However, an unconstrained network, where participating TNs have the freedom to chose any other TN as their neighbor, has limited attack resilience. An attacker can connect a set of colluding servers as as neighbors of a targeted nameserver s and can feed bogus data to s. To form such a network for reliable communication, TrickleDNS constrains the neighbors of a TN s through the use of *consistent hashing* [12]. One way to use consistent hashing is to have an identifier $s.id$ for each TN s drawn from a circular key space. Then, the TN s can pick its i^{th} neighbor as that TN whose identifier is closest to $h(s.id|i)$ clockwise on the key space, where the operation $|$ refers to concatenation and h is a collision resistent hash function whose range is the same circular key space. The neighbors of s, thus chosen are termed its *broadcast neighbors*.

Choosing broadcast neighbors through consistent hashing enhances the attack resilience in two ways: First, an attacker needs to compromise or introduce a large number of servers in the system before it can control a sufficient number of the broadcast neighbors of a targeted nameserver. Second, any attempt by the attacker to fake a broadcast neighbor relationship with a targeted nameserver will be discovered by other TNs since consistent hashing provides easily-verifiable, deterministic neighbor relationships. A network constrained in this manner increases the attack resilience of reliable communication from a constant number k of malicious servers, shown in [23] to a sizeable fraction of the total network ($O(n/\log n)$).

Theorem 1: Reliable communication can be achieved with high probability between any pair of non-malicious nameservers in the presence of $O(\frac{n}{\log n})$ malicious servers provided the number of broadcast neighbors $D \geq \alpha \log n$ for some $\alpha > 6 \ln 2 \sim 4.15$ and the paths used for reliable communication are of length at most $\log n$.

A proof is presented in our technical report [21]. In other words, if every TN has a minimum number of broadcast neighbors, the SNN will contain sufficient number of short vertex-disjoint paths between every pair of TNs such that a majority of these paths do not have any adversarial nodes with high probability.

2) Key Distribution and Verification: Once the reliable communications network is setup, key distribution follows the same protocol described in [23]. We give a brief overview of this protocol for completeness.

State: Each TN s stores an *identity graph* G_s with distinct keyed identities it learns and the neighbor relationships between them. Note that there could be more than one keyed identity for the same server either because the server generated a new public key with a higher sequence number or because a malicious server created a fake keyed identity for it. The identity graph enables s to verify the authenticity of a keyed identity $t.kid$ by checking whether there are at least $\lceil \frac{\mathcal{B}}{2} \rceil$ vertex-disjoint paths between s and t in G_s. If two keyed identities for the same server t pass this check, then s accepts the keyed identity with the greater sequence number. One way to perform the disjoint-path check is by running a standard Max-Flow algorithm.

Protocol: TNs exchange their keyed identity through the broadcast of *signed path vector* messages to their broadcast neighbors. A TN s sends a signed path vector (SPV) $spv[(s.id, s.K, s.seq), s.b_i.id]_{s.k}$ to its broadcast neighbor $s.b_i$. The SPV contains the keyed identity of the sending TN and the identity of the receiving broadcast neighbor; the whole path is signed using the TN s's private key. Any other TN can verify that the SPV has not been corrupted using the embedded public key.

The receiving server $r = s.b_i$ then extends the SPV by adding its own keyed identity and the keyed identity of the broadcast neighbor to which the SPV will be forwarded, signs the extended SPV $spv[spv[(s.id, s.K, s.seq), s.b_i.id]_{s.k}, (r.id, r.K, r.seq), r.b_j.id]_{r.k}$ with its private key, and propagates it. We denote an SPV that traverses servers $s_1, \ldots, s_n$ as $SPV = spv[(s_1, \ldots, s_n)]$.

A receiving TN r rejects an SPV under three conditions: First, the SPV has bogus link relationships; that is, the link relationships do not obey the consistent-hashing-based neighbor selection rules. Second, the SPV contains no new link information about keyed identities. And, finally, the SPV does not verify itself; that is, some signature in the SPV does not match the corresponding public key. If the SPV passes these checks, it is scheduled to be further propagated to the broadcast neighbors.

Overhead Analysis: We can analyze the message overhead of the Reliable Key Broadcast in the steady state where all TNs have learned a stable topology and a new TN joins the system. The new TN s creates $\mathcal{B}$ new neighbor relationships. Since a TN does not forward an SPV if it does not contain any new neighbor relationships, each TN forwards at most one SPV for each new neighbor of s to each of its neighbors. As a result, a TN must perform $O(\mathcal{B}^2)$ verifications and transmit $O(\mathcal{B})$ SPVs for each new TN joining the system.

The overhead may be higher if several TNs join the system simultaneously, but we expect large simultaneous joins to be rare in practice. More seriously, a malicious server could induce the exchange and verification of a large number of SPVs by creating fake server identities and neighbor relationships leading to a DoS attack on the system. Rate limiting on the number of SPVs accepted by a TN from each neighbor eliminates this risk of a DoS attack.

C. SNN Maintenance

This section details how TrickleDNS accepts new TNs into the system, handles failure and leaving of existing TNs, and revocation or replacement of public keys.

Join: A new nameserver s uses the TrickleDNS join protocol where the bootstrap nodes have to approve every new node using majority consensus after running the SybilLimit protocol. Once admitted, the node obtains the set of reliable dissemination links by contacting a few existing TNs called *bootstrap servers*. The bootstrap servers return to s their current identity graphs. s constructs its identity graph G_s by applying majority consensus; that is, it accepts a keyed identity if a majority of the bootstrap servers know about that keyed identity. s then identifies its broadcast neighbors from the list of TN identities it has and initiates the broadcast of its keyed identity.

Failures and Departures: TrickleDNS employs a heart-beat protocol to detect failures. Each TN s periodically broadcasts a signed *keep alive* including its keyed identity. A TN removes a keyed identity from its identity graph if it fails to receive a few consecutive *keep alives* for that keyed identity.

Key Renewal: Finally, a TN might want to revoke its current public key and start using a new pair of private-public keys. Key revocation and renewal in TrickleDNS is trivial and happens when the TN initiates a SPV broadcast with a new keyed identity with the new public key and a higher sequence number.

Note that the above processes of joins, failures, and key renewals might create temporary inconsistencies in the identity graph; few keyed identities might represent departed or failed nodes or older keys while new keys and node identities may not have been included yet. We intend to tolerate these inconsistencies simply as part of the path disjointedness check where they may cause false positives. A slight increase in the neighbor degree, and thereby the network connectivity, is sufficient to alleviate the effect of false positives.

D. Reliable Name Binding

The purpose of Reliable Name Binding is to tightly couple TrickleDNS with the current DNS. The end-goal of TrickleDNS is to be a safety net for the existing DNS as opposed to setting up a completely new namespace. Hence, if a TN s within TrickleDNS claims to be the authoritative nameserver for the domain *foo.bar*, then the property we require from reliable name binding is that: an external client doing a name-lookup through TrickleDNS for *foo.bar* should be redirected to s only if s's claim is genuine.

Each participating domain D chooses a private-public key pair $(D.k, D.K)$ independently and creates a *domain keyed identity* $D.kid = (D.name, D.K, D.seq)$ for itself. A participating authoritative nameserver s of the domain then broadcasts the domain keyed identity on the SNN by signing it with its private key. For its message to be accepted, the TN needs to prove to the system that it is indeed one of the authoritative nameservers of the domain.

TrickleDNS generated this *proof of authority* through independent verification by randomly chosen peer TNs called *certifying servers*. Each TN s belonging to domain D connects to $\mathcal{C}$ certifying servers in a similar manner as it chooses its broadcast servers, that is, constrained random selection through consistent hashing. The i^{th} certifying server of s is that server whose identifier is closest to SHA-1$(s.ip_{0\ldots b-1}|i)$ clockwise on the consistent-hashing key space.

Note that the different authoritative nameservers of a domain will be mapped to the same certifying servers if their IP addresses are from the same block. This reduces the verification load on the certifying servers, especially when malicious servers request verifications for false claims. Moreover, malicious adversaries only owning a small number of IP address blocks cannot launch DoS attacks by requesting authority certificates.

A certifying server c checks whether a TN s is authoritative for a domain D using the DNS hierarchy or TrickleDNS itself if the parent domain is part of TrickleDNS. c perfoms the check as follows: 1) if c reliably knows the public key of the parent domain already it looks for a cached NS records and glue A records signed by the parent indicating that s is authoritative for D. 2) otherwise, c performs a complete DNS lookup identifying the parent domain's nameservers and fetching the NS and glue A records from them. It then checks that a majority of the parent domain's nameservers acknowledge that s is authoritative for the claimed domain.[1].

The i^{th} certifying server c_i for a TN s provides a signed *authority certificate* $cert[c_i.id, s.id, D.id, expiry\_time]_{c_i.k}$ to s attesting its authority over the domain D, which s broadcasts on the SNN. Any server can verify the certificates using the public key of the certifying server. A server accepts that s is authoritative for D if it has at least $\lceil \frac{\mathcal{C}}{2} \rceil$ valid authority certificates.

In order to overcome the certification mechanism, an adversary needs to compromise a majority certifying servers of a domain. Similar to key distribution, randomization makes it difficult to succeed as shown in the following theorem:

Theorem 2: Every non-malicious nameserver can be reliably bound to its domain with high probability in the presence of $f * n$ malicious servers, where f is an upper bound on the fraction of adversarial servers in an n-servers system, provided $\mathcal{C} \geq \beta \log n$, where $\beta = \frac{16(1-f)}{(1-2f)^2} \ln 2$.

A proof of this theorem is presented in our technical report [21]. An authority certificates may need to be revoked since a nameserver currently authoritative for a domain may not be authoritative forever. The certificates have an *expiry_time* after which it is not accepted. We expect a certificate's lifetime to be of the order of days so that the certificate generation and propagation overhead is low and loose-synchronization of clock across servers is sufficient to enforce certificate expiry. However, TrickleDNS permits a certifying server to explicitly revoke a certificate by broadcasting a signed *revocation* $revoke[c_i.id, s.id, D.id, expiry\_time]_{c_i.k}$ throughout the system if required.

In the described certification process, trust is placed on the current DNS hierarchy. While this is a compromise as DNS

[1]Looking for agreement might generate false positives because sometimes DNS servers respond with different set of records depending on the location of the server or the client. However, this is not a serious problem for NS records which are seldom generated dynamically.

is not secure in the first place, we believe it is practical for three reasons. First, it reduces the large trusted computing base (TCB) involved in peer-to-peer DNS alternatives to the much smaller TCB along the DNS hierarchy. Second, higher-level, parent domains typically tend to have better redundancy than low-level domains as we show in Section VI. Finally, in the absence of DNSSEC or an alternative central certification authority, an approach based on independent verification and consensus provides some resilience against compromises in current DNS.

E. Pushing DNS Records

The reliable distribution of domain and server keys facilitates the rest of data dissemination in TrickleDNS. Authoritative nameservers of a domain D broadcast DNS records signed by the domain key followed by their individual server keys.

TrickleDNS proactively propagates only a certain set of critical records that include the delegation or NS records used to identify the authoritative namservers of a domain, the corresponding glue (A) records that provide the IP address of the nameservers, and the start of authority or SOA records. Consequently, a TN typically needs to perform a single lookup to resolve a DNS query for a participating domain (a domain redirection through a CNAME response, however, might require additional lookups). We could avoid this lookup by proactively disseminating every record associated with a domain, but this considerably increases the bandwidth overhead on the TNs diminishing their incentive to participate. Alternatively, we could enable proactive dissemination for popular records (mail servers, web servers, etc.). Doing so requires additional mechanisms to ensure that participating domains do not overload the system.

DNS records propagated on TrickleDNS are not stored forever at each server, but only as long as the *time-to-live* on the DNS record indicates. This ensures that bindings that are no-longer invalid (for example, an expired sub-domain) or that are signed by old keys are expunged from the system. However, if a domain changes a record (for instance, the IP address of a nameserver), it can still proactively broadcast the new record and thereby avoid the long update propagation delay of current DNS.

Finally, dynamically generated DNS records (for example, DHCP addresses) require the private key to be stored in memory to sign records online posing a risk of key compromise. This problem can be mitigated by isolating the signing process and running it on a node that is protected within a firewall, only communicates on restricted ports, and does not also run the name server.

V. Security of TrickleDNS

In this section we provide a top-down summary of the security properties of TrickleDNS and briefly discuss their implications.

A fully-decentralized, peer-to-peer solution to DNS such as TrickleDNS faces at least two types of attacks from malicious adversaries:

1. **Server Compromises:** An adversary might take advantage of a software vulnerability and compromise one or more participating nameservers. While TrickleDNS cannot eliminate software vulnerabilities or prevent compromises, it makes a successful domain hijack from compromised TrickleDNS servers incredibly difficult. In order to successfully hijack a domain, the adversary either needs to compromise a majority of the certification servers, called a *certification attack*, or a sufficient number of servers in the paths between the domain's authoritative nameservers and other TNs, called a *path attack*.

TrickleDNS achieves this high resilience against certification and path attacks through randomization, that is, distributing the vulnerable servers uniformly in the overlay. The success of certification attack depends on the probability of finding a majority vulnerable servers among certification servers, a value that can be made very low by increasing $\mathcal{C}$ as required. Similarly, the probability of success of a path attack can be made as low as desired by increasing the neighbor degree $\mathcal{B}$ as required.

Of course, an adversary can also hijack a domain by compromising one or more of its parent domain nameservers (*parental attack*). If some of the parent domains are part of TrickleDNS, then the above security analysis also applies to them as well. Otherwise, this dependence of current DNS hierarchy is an unavoidable risk inherent in the DNS protocol.

2. **Identity Attacks:** An adversary could increase its chances of succeeding in a path attack or certification attack by artificially increasing the number of malicious servers in the system or breaking randomization by controlling the identity of the server. A rich and powerful adversary might be able to launch an attack using a large network of compromised hosts called a Botnet. TrickleDNS limits the number of compromised hosts that can enter the system by leveraging SybilLimit and the social trust links that already exist implicitly in the DNS.

TrickleDNS is secure against infiltration by Botnets due to a combination of two factors. First, by using the SybilLimit protocol on the social network, TrickleDNS can guarantee that the total number of Sybil identities is bounded by $\log n$ identities per attack edge in the social network. Second, the reliable communication protocol for disseminating public keys is resilient in the face of upto $O(n/\log n)$ adversarial nodes. If γ is the number of attack edges that an attacker owns in the social network, then the number of Sybil identities in TrickleDNS is bounded by $\gamma \log n << n/\log n$. In practice, γ is a constant since it is indicative of the number of real authoritative name servers compromised by a botnet. In smaller networks, the probabilistic security guarantee can be strengthened at the cost of increased communication, by increasing the degree d of the number reliable communication links of each node. To achieve perfect reliable communication without relying on any randomness, we require $d \geq 2\gamma \log n + 1$ which is feasible in small networks.

Another, more subtle way to launch an identity attack is to attack the IP layer by a) spoofing IP addresses, b) IP hijacking, or c) man-in-the-middle attacks. TrickleDNS is immune to IP spoofing because it performs two-way communication using TCP. IP hijacking by compromising Internet routing, however, could be dangerous; if the hijack is partial it is likely to have less impact as TrickleDNS connects each server to several others randomly distributed in the Internet. On the other hand,

a complete hijack can be treated as a compromised IP address or IP address block; the above analysis for server compromises holds for IP hijacks as well. Man-in-the-middle attacks have a similar impact as hijacked IP addresses and can be treated as a server compromise.

Finally, we expect that a participating domain interested in its own security will take the necessary measures (ie. apply patches) to secure its own nameservers. TrickleDNS does not protect a domain from compromises to its own nameservers. Its goal instead is to protect a participating domain from vulnerabilities in other, less-secure domains.

VI. Evaluation

In this section, we evaluate the security properties and the performance of TrickleDNS.

A. Security Analysis

While the theoretical results in Section III showed that TrickleDNS can handle $O(n/\log n)$ adversarial servers, this bound is asymptotic and may not completely reveal the effectiveness of TrickleDNS in real-world settings. In this section, we evaluate the resilience of TrickleDNS to the path, certification, parental, and Sybil attacks defined above through simulations.

We simulated random topologies to represent TrickleDNS networks. To generate a topology of n servers, we assigned a random IP address to each node and connected it to $\mathcal{B} = \log n$ other nodes according to the rules of Section III. To analyze security attacks, we modeled a global adversary that controls a random fraction f of servers uniformly distributed in the topology. These servers are assumed to collude to cause maximum damage to the system.

1) Resilience from Path Attacks: First, we evaluate the resilience of TrickleDNS from path attacks. Recall that for a correct dissemination of a key from a source node to a destination node, at least a majority of vertex-disjoint paths between the pair of nodes should be void of compromised servers. We define a *good path* in the generated topology as a path from the source to the destination passing only through non-compromised nodes. For reliable communication, the number of good paths must be at least $\lceil \frac{1}{2} \log n \rceil$ since the neighbor degree is $2 \log n$. We call such a pair of servers a *reliably-communicating pair*.

Figure 2 shows the CDF of the number of good paths between node pairs for a system with $n = 65536$ servers of which $f = 5\%$ are compromised and controlled by an adversary. For this scenario, each nodes has 16 neighbors and an expected 16 disjoint paths with every other node. Thus any pair of nodes with 9 or more disjoint paths is a reliably communicating pair.

In Figure 2, at least 99% of node pairs are able to communicate reliably. This may appear to be not high enough, but note that we are looking at a worst case scenario of 5% of malicious nodes all colluding together. In practice, we'd expect much fewer compromises at any given time and even if there are many compromised servers for them not to all collude together. Moreover, we can easily increase the neighbor degree if required to handle a greater number of compromised servers.

Figure 3 shows the tolerance to path attacks for different values of malicious fraction f and network size n. These numbers are an average of 10 runs. We observe a high extend of reliable communication happens for all network sizes shown. However, as expected, there is a critical point of f beyond which the fraction of reliably communicating pairs drastically drops.

2) Resilience from Certification Attacks: Next we evaluate the resilience of TrickleDNS to certification attacks. Figure 4 shows the CDF of the fraction of non-malicious servers in the certifying server set of a TN. Here, we set the number of certifying servers for each domain to be $\log n$ and the number of servers in the system to be $n = 65536$. Recall that a TN succumbs to a certification attack if a majority of certification servers are compromised. Figure 4 shows that even in the presence of $f = 5\%$ compromised servers in the system, the probability of a genuine server succumbing to the attack is very small (less than 10^{-5}). Even at a high fraction of 20% compromised servers, only about 1% of genuine servers succumb to the attack.

3) Resilience from Parental Attacks: To understand the impact of attacks to legacy DNS servers on TrickleDNS, we analysed a snapshost of inter-domain parent-child relationships in DNS as it existed on July 22, 2004. This snapshot was generated from a study done at Cornell University [19] and contains $166,771$ distinct name servers that contribute towards resolution of $597,196$ distinct domains.

Figure 5 shows the CDF of the number of attacks against the authoritative nameservers of the parent domain that the majority-consensus approach can tolerate. In general, it is $\lfloor \frac{a-1}{2} \rfloor$ for compromises and $a - 1$ for DoS attacks, where a is the number of distinct authoritative nameservers involved. The key observation from Figure 5 is that close to 90% of the domains able to tolerate at least 6 compromises in its parent's nameservers. This is surprising because, in general, more than 80% of domains have only two nameservers [15], [18]. The surprisingly good resilience to attacks at parents comes from the DNS hierarchy being flat and a large number of domains (even though not secure on their own) fall directly under the more secure top-level domains. The sharp increase in parental resilience at 6 corresponds to the large number of *.com* domains that are served by the thirteen *.com* name servers.[2] Finally, Figure 5 also shows that the resistance to DoS attacks for parent domains is good.

4) Resilience from Sybil Attacks: For the purposes of evaluating the effectiveness of using the SybilLimit protocol on the DNS topology, we constructed a graph of DNS nameserver "trust" relationships and measure expander graph properties of this graph. We show that the number of new neighbors discovered via vertex expansion increases exponentially and therefore the graph is fast-mixing, implying that the SybilLimit protocol guarantees will apply.

To construct the DNS trust graph, we used the same DNS relationships dataset obtained from Cornell [19] we used previously. We represent the nameservers as vertices and undirected

[2] In reality, there are more than thirteen *.com* nameservers behind the thirteen published IP addresses. In this analysis, we just count parents by the number of distinct NS records returned.

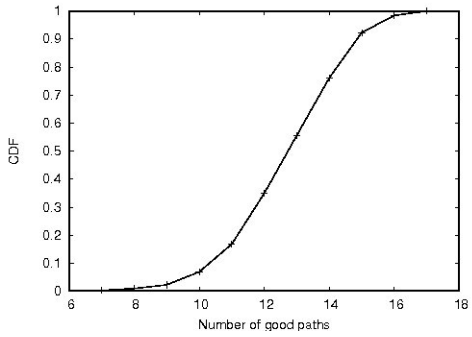

Fig. 2. CDF of good paths for 65536 servers and malicious fraction = 5 %

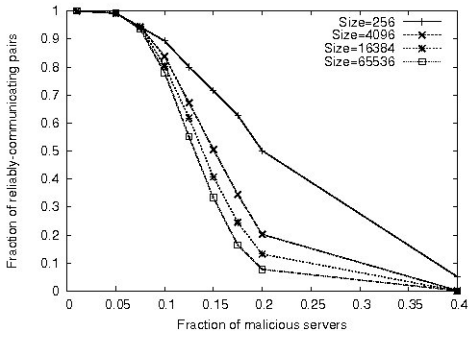

Fig. 3. Reliably communicating pairs for different values of system sizes and malicious fractions.

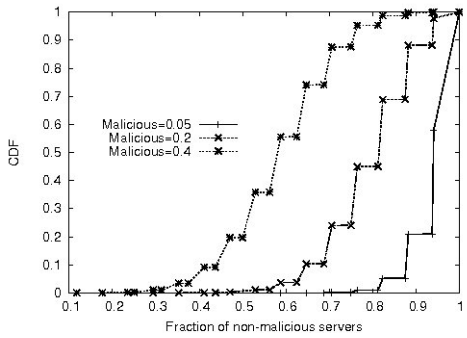

Fig. 4. **CDF of the fraction of non-malicious servers in the certifying server set.**

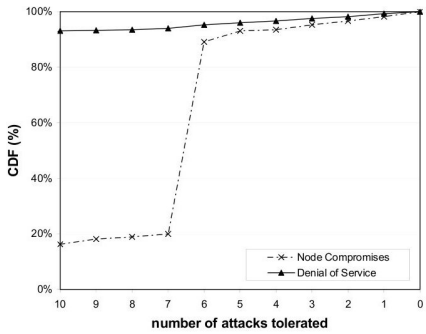

Fig. 5. **CDF of the number of node compromises and DoS attacks that can be tolerated by existing domains.**

edges as the trust relationships between all authoritative nameservers of a domain. To this, we also add edges representing parent-child dependencies, that is, all nameservers belonging to domain foo.bar.edu have edges to all authoritative name servers of *bar.edu*.

We found that the above graph including the parent-child trust links was nearly completely connected — 166,758 servers out of 166,772 (the disconnected server names may simply be artifacts of the crawling methodology). Without including the parent-child trust links, the graph was extremely disconnected with 70588 components.

We perform a vertex expansion by running a breadth first search (BFS) from the server nodes of a randomly selected domain in our trust graph and count the number of unique neighbors at each iteration. Figure 6 shows the min/max/avg number of new neighbors discovered per hop from 1000 randomly selected seed domains. We can see that the diameter of the trust graph is only 12 hops with the trust links from the DNS hierarchy indicating a high degree of connectedness in the graph which is the property that we want to show. In contrast, without the DNS hierarchy edges the graph diameter is of the subcomponent selected reaches 32 hops. We conclude that SybilLimit would maintain its guarantees for the authoritative nameserver trust topology if the DNS heirarchy's trust relationships were included in the graph.

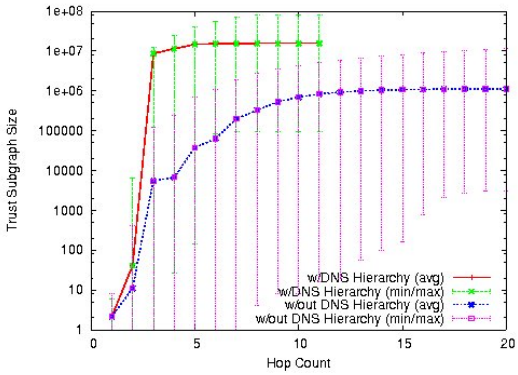

Fig. 6. **Neighbors per hop in DNS trust network**

Summary The analysis in this section tells us that i) the reliable communication mechanism and the certification mechanism can tolerate a significant malicious presence, ii) that using the parental nameservers to verify the authority of a nameserver before it joins TrickleDNS is a reasonable approach, and iii) existing trust relationships between domains can be leveraged on to mitigate Sybil attacks. These properties have been achieved with $\log n$ broadcast and certifying servers. These system parameters can be increased by a constant factor with little added overhead so that the path and certification resilience can be improved. Similarly, resilience to Sybil attacks

can be improved by encouraging domains to form more trusted links. On the other hand, the parental resilience is not easy to improve without having the parental domain actually join the TrickleDNS network.

B. Performance Analysis

In this section, we describe experiments to measure the performance of TrickleDNS and compare it with legacy DNS. Here, we are interested in quantifying the following three metrics: 1) lookup latency of DNS queries in TrickleDNS, 2) time taken to push DNS records between name servers, and 3) memory and bandwidth overhead of pushing DNS records.

For benchmarking performance, we implemented the TN functionality based on the djbdns [3] codebase. The mechanisms for reliable communication described in Section III are implemented as a reliable communication toolkit and exported to the TrickleDNS implementation codebase. Our evaluation used 1024-bit RSA keys for authentication. The TN implementation is layered on top of the toolkit. Each TN acts as a caching DNS nameserver that can support the operations and optimizations present in current DNS.

We deployed our implementation of the TN on the 62-node PSI cluster [2]. To compare the lookup performance of TrickleDNS with legacy DNS, each TN acts as an authoritative nameserver. Queries are sent to a randomly chosen TN. The TN tries to answer the query from its cache; otherwise it queries legacy DNS and reflects the response record. Whenever a TN gets a new record, it pushes it to the other servers. This gives us an estimate of the overhead of pushing DNS records. Queries are generated from a portion of the real workload collected by Jung *et al.* [11] at MIT with 281,943 total queries to 47,320 distinct domains.

In our testbed, we executed 60 TN instances on different nodes with each instance having an average degree of 6. Each TN instance reliably discovered the other instances in the network and then pushes nameserver records to the these instances.

1) Lookup Characteristics: We first compare the lookup latencies in TrickleDNS and the legacy DNS. From the discussion on the lookup process in Section III, we notice that the time to perform a lookup is dependent on whether the client's local DNS server is part of the TrickleDNS. Given a target domain D, if the local DNS server is part of the TrickleDNS, it can respond to a query with the NS and glue records of an authoritative name server of D. Otherwise, the local server reflects the query to a TN. To enhance security, the local server may simultaneously queries different TNs and wait for a majority consensus.

The first experiment deals with the scenario where the local DNS server is a TN. In this case, Figure 7 shows the distribution of query latency for TrickleDNS and legacy DNS. Only queries which were not answered from the local cache were included in the measurement. We measure the latencies of those queries for which the TNs are already populated with the target records. Otherwise, the query latency would include the latency of fetching the record via legacy DNS. The Figure shows that the median latency of TrickleDNS is a factor of 10 lower than legacy DNS. Note that the latencies shown in both cases are simply the latencies for fetching the authoritative NA and glue A record. The DNS clients then have to contact the authoritative server for the target A record which will incur an additional delay irrespective of whether legacy DNS or TrickleDNS is used.

In the second experiment, we consider the case in which the client's local DNS server is not part of TrickleDNS. The local server then reflects the query to a set of TNs and takes a majority vote on their responses. In our experiments, the resolver contacts 3 TNs for redundancy. The latencies for this experiment is shown in Figure 8.

We consider three scenarios corresponding to no node in the network being malicious, and 1 and 5 nodes being malicious respectively. A malicious node when contacted simply allows the query to time out. The first observation is that, in the median case, queries take almost the same time as the case where only one node is contacted. Further, the addition of a single malicious node does not affect the latencies because the servers are chosen uniformly at random for each query. With 5 malicious nodes, the median latency only increases by about 8%. With a larger network size, a single malicious node will have a smaller influence on the query latency.

To summarize, for domains that are part of TrickleDNS, queries are answered much faster than legacy DNS both in the cases where the local DNS server is a TN and when it is not (and must send out redundant queries).

2) Overhead of Pushing DNS records: Proactive dissemination of keys and DNS records in TrickleDNS does incur additional overhead. Within our experimental setup, we computed two quantities: (a) the time incurred by a new TN to reliably broadcast its key when it joins the network; (b) the time incurred by a new TN to propagate a DNS record in a secure manner. As expected, we found out that the update time for reliable broadcast keys was much higher than the time to broadcast DNS records. The 90^{th} percentile of these two quantities were 4.5s and 180 ms respectively. While reliable broadcast of keys is relatively expensive due to the need for path-vector signature computation and verification operations, this operation is relatively infrequent (on the order of days) since the underlying topology is not very dynamic. Once key distribution within the SNN is accomplished, subsequent propagation and verification of DNS records incurs very low overhead.

Based on our experiments, we found out that the system profiles for the TN instances indicate that TrickleDNS incurs low bandwidth and memory overhead. The bandwidth exchanged between TNs is 1.35 KBps of which only 12% of the bytes are used in reliable communication. This represents a very small bandwidth overhead. The net memory usage of each TN was roughly 9.356 MB of memory at the end of the trace with 6804 records being stored. The state maintained and propagated per domain by our implementation is less than 10 KB. In addition, our implementation can verify the validity of roughly $40,000$ signatures every second.

Recent work by Handley *et al.* [9] shows that roughly 0.5% of domains change name servers and about 0.1% of domains expire every day. Extrapolating to the entire DNS, they claim that roughly $420,000$ domains change nameservers

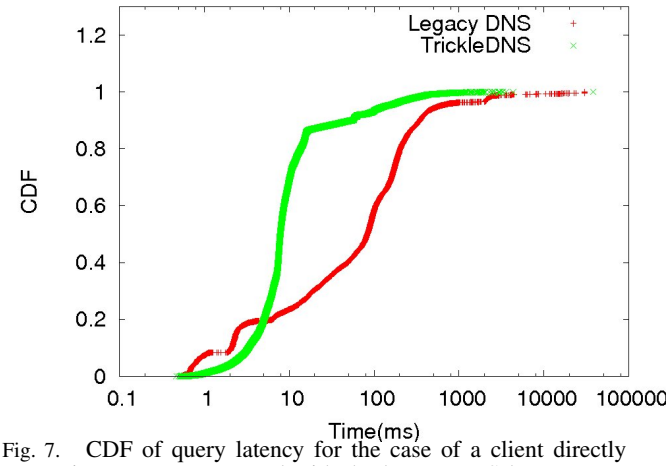

Fig. 7. CDF of query latency for the case of a client directly contacting an TNs compared with the legacy DNS latency.

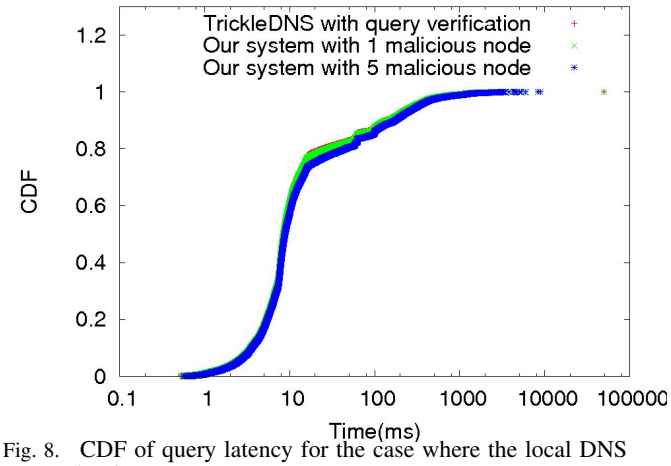

Fig. 8. CDF of query latency for the case where the local DNS server that is not an TN.

and $100,000$ domains expire everyday. For the entire DNS this translates to an update rate of 1.6 Kbps [9], a rate that can very easily be handled by our system.

In summary, the overhead of pushing name server records is reasonably small in terms of the amount of state maintained, bandwidth requirements, memory requirement and processing overhead.

VII. Conclusions

This paper presented TrickleDNS, a peer-to-peer proactive dissemination system for DNS. TrickleDNS is a safety net for DNS and is meant to act as a stopgap to secure DNS until DNSSEC is eventually fully adopted. The primary contribution of TrickleDNS is to be secure against malicious attacks that may attempt to corrupt or hijack DNS records. To this end, TrickleDNS builds a robust overlay network that can tolerate commonly-encountered attacks, while providing low lookup latency, fast update propagation, and improved failure resilience.

References

[1] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Protocol Modifications for the Domain Name System Security Extensions. RFC 4035, March 2005.

[2] Berkeley Millennium Cluster. http://www.millennium.berkeley.edu/PSI/index.html.

[3] D. Bernstein. djbdns. http://cr.yp.to/djbdns/notes.html.

[4] N. Brownlee, kc claffy, and E. Nemeth. DNS Measurements at a Root Server. In *Proc. of IEEE GlobeCom*, San Antonio, TX, November 2001.

[5] N. Brownlee, kc Claffy, and E. Nemeth. DNS Root/gTLD Performance Measurements. In *Proc. of Usenix Systems Administration Conference*, San Diego, CA, December 2001.

[6] R. Cox, A. Mutitacharoen, and R. Morris. Serving DNS using a Peer-to-Peer Lookup Service. In *Proc. of IPTPS*, Cambridge, MA, March 2002.

[7] T. Deegan, J. Crowcroft, and A. Warfield. The main name system: an exercise in centralized computing. *SIGCOMM Comput. Commun. Rev.*, 35(5):5–14, 2005.

[8] C. Fetzer, G. Pfeifer, and T. Jim. Enhancing dns security using the ssl trust infrastructure. In *Proceeding of the IEEE WORDS 2005*, 2005.

[9] M. Handley and A. Greenhalgh. The Case for Pushing DNS. In *Proc. of HotNets*, November 2005.

[10] C. Huitema and S. Weerahandi. Internet measurements: The rising tide and the DNS Snag. In *Proc. of ITC Specialist Seminar on Internet Traffic Measurement and Modeling*, Monterey, CA, September 2000.

[11] J. Jung, E. Sit, H. Balakrishnan, and R. Morris. Dns performance and the effectiveness of caching. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 153–167, New York, NY, USA, 2001. ACM Press.

[12] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigraphy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *Proc. of ACM Symposium on Theory of Computing*, El Paso, TX, April 1997.

[13] P. Mockapetris. Domain Names: Concepts and Facilities. Request for Comments 1034, November 1987.

[14] P. Mockapetris. Domain Names: Implementation and Specification. Request for Comments 1035, November 1987.

[15] V. Pappas, Z. Xu, S. Lu, D. Massey, A. Terzis, and L. Zhang. Impact of Configuration Errors on DNS Robustness. In *Proc. of ACM SIGCOMM*, Portland, OR, August 2004.

[16] K. Park, V. Pai, and L. Peterson. CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups. In *Proc. of Symposium on Operating Systems Design and Implementation*, San Francisco, CA, December 2004.

[17] L. Poole and V. S. Pai. ConfiDNS: Leveraging scale and history to improve DNS security. In *Proceedings of WORLDS 2006*, Seattle, WA, Nov 2006.

[18] V. Ramasubramanian and E. G. Sirer. The Design and Implementation of a Next Generation Name Service for the Internet. In *Proc. of ACM SIGCOMM*, Portland, OR, August 2004.

[19] V. Ramasubramanian and E. G. Sirer. Perils of Transitive Trust in the Domain Name System. In *Proc. of ACM IMC*, Berkeley, CA, October 2005.

[20] A. Rowstorn and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-scale Peer-to-Peer Systems. In *Proc. of IFIP/ACM International Conference on Distributed Systems Platforms*, Heidelberg, Germany, November 2001.

[21] S. Sankararaman, J. Chen, V. Ramasubramanian, and L. Subramanian. Trickledns: Bootstrapping dns security using social trust. In *Technical Report. Available at http://www.cs.nyu.edu/~lakshmi/tricklednsreport.pdf*. Computer Science Department, New York University, 2011.

[22] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. of ACM SIGCOMM*, San Diego, CA, August 2001.

[23] L. Subramanian. Decentralized security mechanisms for routing protocols. Ph.D. thesis, University of California, Berkeley.

[24] M. Theimer and M. B. Jones. Overlook: Scalable name service on an overlay network. In *ICDCS '02*, page 52, Washington, DC, USA, 2002. IEEE Computer Society.

[25] C. E. Wills and H. Shang. The Contribution of DNS Lookup Costs to Web Object Retrieval. Technical Report TR-00-12, Worcester Polytechnic Institute, July 2000.

[26] H. Yu, P. Gibbons, M. Kaminsky, and F. Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. In *IEEE Symposium on Security and Privacy, 2008. SP 2008*, pages 3–17, 2008.