Title

A Proxy View of Quality of Domain Name Service, Poisoning Attacks and Survival Strategies

Journal

ACM Transactions on Internet Technology, 12(3)

ISSN

1533-5399

Authors

Yuan, Lihua
Chen, Chao-Chih
Mohapatra, Prasant
et al.

Publication Date

2013-05-01

DOI

10.1145/2461321.2461324

Peer reviewed

A Proxy View of Quality of Domain Name Service, Poisoning Attacks and Survival Strategies

LIHUA YUAN, CHAO-CHIH CHEN, PRASANT MOHAPATRA, and CHEN-NEE CHUAH,
University of California, Davis
KRISHNA KANT, Intel

The Domain Name System (DNS) provides a critical service for the Internet – mapping of user-friendly domain names to their respective IP addresses. Yet, there is no standard set of metrics quantifying the *Quality of Domain Name Service* (QoDNS), let alone a thorough evaluation of it. This article attempts to fill this gap from the perspective of a DNS proxy/cache, which is the bridge between clients and authoritative servers. We present an analytical model of DNS proxy operations that offers insights into the design tradeoffs of DNS infrastructure and the selection of critical DNS parameters.

Due to the critical role DNS proxies play in QoDNS, they are the focus of attacks including cache poisoning attack. We extend the analytical model to study DNS cache poisoning attacks and their impact on QoDNS metrics. This analytical study prompts us to present Domain Name Cross-Referencing (DoX), a peer-to-peer systems for DNS proxies to cooperatively defend cache poisoning attacks. Based on QoDNS, we compare DoX with the cryptography-based DNS Security Extension (DNSSEC) to understand their relative merits.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems; C.4 [**Performance of Systems**]: *Performance attributes; reliability, availability, and serviceability*

General Terms: Reliability, Security

Additional Key Words and Phrases: DNS, QoDNS, proxy, cache, poisoning

ACM Reference Format:
Yuan, L., Chen, C.-C., Mohapatra, P., Chuah, C.-N., and Kant, K. 2013. A proxy view of quality of domain name service, poisoning attacks and survival strategies. *ACM Trans. Internet Technol.* 12, 3, Article 9 (May 2013), 26 pages.
DOI:http://dx.doi.org/10.1145/2461321.2461324

1. INTRODUCTION

The Domain Name System (DNS) is one of the most critical components in the Internet infrastructure. Almost all applications, for example, HTTP, email and FTP, rely on DNS to resolve human-friendly domain names to the corresponding machine-friendly IP address prior to establishing connections. DNS is also tasked to distribute

This work was partially supported by Intel and NSF NeTS NBD-0520320.
Parts of this article appeared as "DoX: A peer-to-peer antidote for DNS cache poisoning attacks," in *Proceedings of the IEEE International Conference on Communications, 2006 (ICC'06)*, 2345–2350, and as "A proxy view of quality of domain name service," in *Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM'07)*, 321–329.
Authors' addresses: L. Yuan, Microsoft Corporation, 1 Microsoft Way, Redmond, WA 98052.; C.-N. Chen and P. Mohapatra, 2063 Kemper, Computer Science Department, University of California at Davis, One Shields Avenue, Davis, CA 95616-5294; email: {cchchen, mohapatra}@cs.ucdavis.edu; C.-N. Chuah, 3125 Kemper, Electrical & Computer Engineering Department, University of California at Davis, One Shields Avenue, Davis, CA 95616-5294; email: chuah@ece.udavis.edu; K. Kant, Research Professor, Center of Secure Information Systems, George Mason University, Fairfax, VA 22030; email: kkant@gmu.edu.

DOI:http://dx.doi.org/10.1145/2461321.2461324

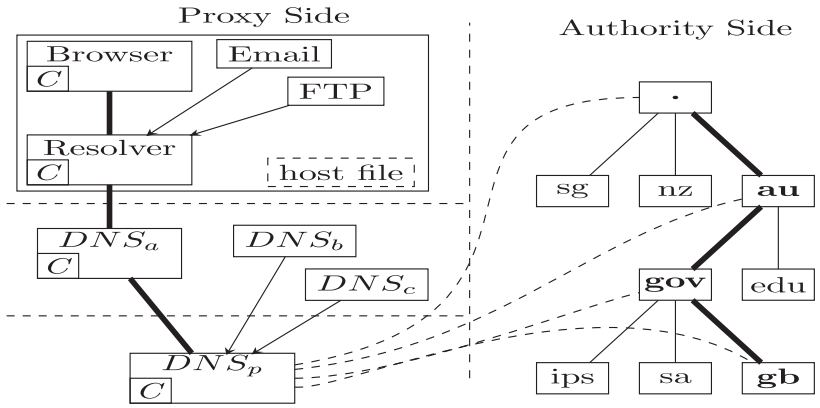


Fig. 1. DNS architecture.

information about mail exchange, serve as public-key infrastructure, and provide dynamic load distribution. Therefore, the notion of *quality of Domain Name Service* (QoDNS) is important to a large range of users and applications on the Internet. However, there is currently no standard set of metrics quantifying QoDNS, let alone a thorough evaluation of it.

In this article, we define a comprehensive set of QoDNS metrics, which includes *accuracy*, *availability*, *latency* and *overhead* (Section 2). Accuracy and availability refer to the ability to supply up to date and correct DNS records to the client so that it can connect to and only to the desired site. The latency incurred by DNS lookup can contribute a signification portion of the overall latency observed by applications in some cases [Hughes and Touch 1999]. The messaging overhead of DNS lookups at proxy and higher level authoritative servers might affect the scalability of DNS servers and impact Internet infrastructure substantially.

Analyzing and studying QoDNS is however a challenging task because of the complex architecture of DNS. As illustrated in Figure 1, DNS is a distributed database served by a large number of *authoritative servers*. These servers are organized in a hierarchical fashion like an inverted tree. Each server provides the records of its own domain and referral information to servers at next lower level, which holds information of the subdomain. Such a hierarchical organization enables any client to start from the root servers, whose location is fixed and known to everyone, and iteratively follow the referral to find the corresponding record of any domain name. For scalability and performance reasons, DNS uses proxies (also called *caching-only name servers*) to relay queries and responses between clients and servers. DNS proxies cache replicas of records so that subsequent queries for the same record can be answered locally. The hierarchical organization of DNS also make it possible for proxies to improve lookup performance by starting an iterative lookup from the best-matching record (BMR) instead of the root server. Being the bridge between clients and servers, proxies and their caches are key components in determining the QoDNS. To this end, this article propose a proxy-centric view of DNS.

The fundamental blocks in DNS, DNS records, are typically small in size and the expiration of Time-To-Live (TTL) timer is the primary mechanism flushing records. Second, DNS proxies must explicitly consider the hierarchy of domain names, the delegation between a domain and its subdomains, and the best-matching algorithm proxies used to search its cache. In Section 3, we propose a hierarchical model of DNS cache, which captures the essential operation of DNS proxies. Using this model,

administrators can study the impact of various parameters (such as TTL) on QoDNS, determine quantitatively the trade-offs associated with the parameters. One can also use our model to study schemes such as dynamic DNS. Such QoDNS-aware study enables administrators to make informed decisions to deploy or tune the DNS to achieve better performance.

In addition to the system administrators, various attacks can affect QoDNS in many ways as well. Among them, *DNS cache poisoning*[1] attack [Bellovin 1995], billed as "the Achilles' heel of the Internet" by some [Evers 2005], has raised the greatest concern because of its significant impact on accuracy. DNS cache poisoning refers to the cases where the cache of a proxy server is injected with incorrect records. Consequently, clients get incorrect, and probably malicious, replies that will lead connection attempts to the false IP addresses. This is often used as a foothold for escalating to more harmful sub-attacks, for example, Man-In-The-Middle attacks [Green 2005] or large-scale phishing attacks known as *pharming* [Netcraft, Ltd. 2005]. Cache poisoning attack to DNS is very different from such attacks to Web cache because poison can, as witnessed in the incident of March 2005 [Haugsness 2005], propagate through its hierarchical structure and affects far more records than those original poisoned. In Section 5, we extend our model to characterize poisoning attacks to DNS cache. Our extended model offers a sound explanation to the poison propagation mechanism and a tool to analyze the impact of poison attack on QoDNS.

There are several proposed defense mechanisms to protect DNS from cache poisoning attacks. In addition to piecemeal patches for specific vulnerabilities, past work focused on applying cryptography techniques to authenticate DNS records. Among them, DNS Security Extensions (DNSSEC) [Arends et al. 2005], which is based on the public-key infrastructure (PKI), has attracted most efforts. Although DNSSEC was proposed 15 years ago, its current deployment is very limited [Liu 2007]. We believe the following factors have contributed to the stalemate of DNSSEC. First, an effective DNSSEC implementation requires a global cooperation to form the chain of trust, and past experience have shown that deployment requiring global cooperation at different level is difficult. Second, the success of DNSSEC requires modifications to the DNS servers at every level, the availability of security-aware resolver for every operating system, and complex key management. Considering the fact that many DNS servers are currently running with known vulnerabilities, one probably cannot expect the complete deployment of DNSSEC in a short period of time.

In Section 6, we propose a proxy-centric solution to DNS cache poisoning called *domain name cross-referencing* (DoX). Our main observation is that DNS records are expected to be consistent at different proxies (except for a few cases that we will discuss in Section 6). However, cache poisoning is a local attack which affects individual proxies. It is difficult for attackers to compromise several DNS proxies simultaneously with the same false data. Therefore, DoX propose to groups peers together to compare their DNS results and detect inconsistencies as a waring sign of poisoning attacks. In Section 7, we compare DNSSEC and DoX based on QoDNS to assist administrators in choosing an ideal solution.

The contributions of this article are the following.

— We define a set of metrics to comprehensively evaluate and compare QoDNS (Section 2).
— We present an analytical model of DNS proxies for quantitative evaluation of QoDNS (Section 3). Our model can be used to study the trade-offs of key

[1]Also referred to as *name spoofing*, *hijacking*, or *cache pollution*.

Table I. Summary of Symbols

Symb.	Description	Symb.	Description
n	A *DNS node**	$T(n)$	TTL value of n
n_i	ith child of n	$A^i(n)$	i-level ancestor of n
$n_{i,j}$	jth child of n_i	H	Height of the DNS tree†
$s(n)$	State of n	$X(n)$	Query arrival process of n
$h(n)$	Level of n	$Y(n)$	Query miss process of n
$W(n)$	Weight of n	$M(n)$	Modification process of n
$B^i(n)$	A query to n is best-matched to $A^i(n)$		

*DNS node in the context of this paper denotes domains in the DNS system.
†Height of the delegation tree. Note that we assumed a simplified model where no delegation across TLDs occurs.

configuration parameters like TTL or the impact of the new scheme like *Dynamic DNS* (Section 4).

— We extend our model to study cache poisoning attack (Section 5) and propose DoX, a peer-to-peer solution that requires no modification to the current DNS architecture (Section 6). We further compare DNSSEC and DoX in terms of QoDNS (Section 7) so that administrators can decide which solution might be most appropriate in their specific situations.

2. QODNS METRICS

Table I presents a summary of symbols used in the following part of this article.

This section presents a formal definition of QoDNS based on these key user-observable aspects: accuracy, availability, latency and overhead.

2.1. Accuracy of DNS Resolutions

A DNS client may receive inaccurate mappings for two main reasons: obsolete and poisoned (or tampered) records. DNS uses a "weak consistency" model wherein a cached resource record is not flushed from the proxy cache until the TTL expires, even if its master copy at the authoritative service is updated in the mean time. As a result, clients may receive obsolete (and incorrect) records. Since DNS records are typically small, storage limitation is normally not an issue (unlike web caches). Consequently, TTLs are generally fairly large and capacity-triggered eviction of the records are rare. This situation tends to aggravate the problem of obsolete records when such problem arises, leading to failed communication attempts.

As we will see in Section 5, poison records could be injected into DNS cache by exploiting software or protocol vulnerabilities. The poisoned records could use a large TTL to increase its lifetime. Furthermore, queries from lower levels of the hierarchy will propagate them further.

While both poison records and obsolete records are *inaccurate*, we must differentiate them since their causes and consequence are quite different. In particular, poison records are more dangerous since they can be exploited for further, more serious attacks.

2.2. Availability of DNS

DNS is a mission-critical system and it is important to ensure the availability of its data to any querying client, even under failures or malicious attacks. The distributed nature of DNS coupled with caching by numerous local proxies help make DNS service

quite robust. For example, the data replication provided by DNS proxies helped the DNS infrastructure survive the massive scale Distributed Denial-of-Service (DoS) attack in October 2002 [Naraine 2002]. However, caching can also negatively affect the availability. In particular, if the desired record is uncached, and the BMR in cache is obsolete, we have a *referral failure* and the DNS resolution cannot continue unless the DNS client is configured with other DNS proxies that can return accurate information.

It is important to note that if a record (that provides the name to IP mapping to the client) is obsolete, it will cause an *application failure* when the client attempts to use the IP address. This situation is covered by the accuracy aspect (see Section 2.1), and is not an availability issue for the DNS. In both cases, the TTL expiry and subsequent retrieval of updated record from AS (authoritative sever) will fix the problem without any administrative intervention.

Although overall availability of the DNS service (defined as the fraction of time the service remains responsive) is a crucial parameter, we only concentrate on the obsolete referral aspect of it in this article. The reason for this choice is that traditional availability enhancement techniques (e.g., hardware redundancy, hot swap, data replication, etc.) are well studied in the literature.

2.3. DNS Lookup Latency and Overhead

A DNS lookup precedes almost every connection attempt. Consequently, the latency incurred by DNS lookup will affect the overall user-experienced latency. Early measurement results show that DNS lookup latency forms a significant part of Web latency [Hughes and Touch 1999]. There are two components to overall latency.

(1) *Network Latency*. This comes into play primarily on misses from proxy or the DNS server hierarchy. The client-to-proxy query/response latency is usually on a local LAN and thus not significant. The iterative queries through the DNS hierarchy could add substantial latency.
(2) *Processing Latency*. This includes all aspects of local processing at a server. This latency is typically small but could become significant in special cases, for example, when using public key cryptography (e.g., DNSSEC) or as a result of substantial imbalance in load distribution. We ignore processing latency in our model.

Closely related to latency is the notion of *overhead*, which too can be described in terms of network and processing components. In particular, the *network overhead* refers to the network traffic resulting from a given scheme. The WAN traffic is of primary interest here and results from cache misses at various levels of the DNS hierarchy. The computational overhead can be measured using some suitable CPU related metric (e.g., instructions/sec, cycles/sec, etc.). As with latency, computational overhead is likely to be relevant only in specific circumstances.

2.4. Definition of QoDNS

Based on the previous discussion, we define QoDNS as a quintuplet $< \mathcal{A}_O, \mathcal{A}_P, \mathcal{V}_O, \mathcal{L}_N, \mathcal{O}_M >$ where the following holds.

$\mathcal{A}_O$ is the probability that the service provides an obsolete resource record.
$\mathcal{A}_P$ is the probability that the service provides a poison resource record. If the proxy and its ancestors are not under malicious attack, this metric is always zero.
$\mathcal{V}_O$ is the overall unavailability of DNS service under the assumption of perfect node level availability (i.e., excluding the impact of node failures).

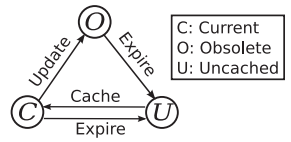


Fig. 2. State transition diagram.

$\mathcal{L}_N$ is the overall query-response latency for DNS service. Except in cases where computation latency may be substantial. This metric is dominated by the number of round-trip communications over the Internet.

$\mathcal{O}_M$ is the network overhead of DNS service, measured as average number of packets send and received per query. We make the simplified assumption that queries and responses can fit into one packet of the same size.

3. ANALYSIS OF STANDARD DNS

This section presents an analytical model for DNS proxies and their caches under normal operation.

3.1. Modeling of a Single Record

Figure 2 presents a simple state machine for various states of a record. Initially, the record is in *uncached* state indicated as U. The arrival of a query causes the record to be cached (indicate by state C). The caching holds for the TTL duration and then reverts to U state. If the authoritative server (AS) of the record updates it while the record is in cached state, the cached copy becomes *obsolete* (O). These three states is enough to describe a single record if the proxy is operating normally.

3.2. Recursive Model for Query Arrivals

A name lookup through a proxy server essentially consists of two steps: best-matching and referral-following. We model the best-matching algorithm as queries climbing up the DNS tree. When a query on node n results in a cache miss, the proxy needs to obtain the NS record of n from the parent, $A^1(n)$, to contact the AS. This is considered as an *induced query*. An induced query to $A^1(n)$, if it results again in a cache miss, induces a query to its own parent, $A^2(n)$. This process iterates until a hit to a cached ancestor node, that is, the BMR, happens. If the BMR is *current*, the referral-following algorithm climbs back down level-by-level by querying the servers listed in the NS records, until it reaches n. Records of nodes on the path from the BMR to n will be *cached* during this process.

DNS queries typically arrive for leaf nodes and queries to non-leaf nodes are dominated by induced queries. While query arrival characteristics for leaf nodes are easy to measure at proxy servers, query characteristics for non-leaf nodes depend on the child nodes, their TTL values, and their query characteristics. Therefore, we propose a recursive model to derive the query arrival characteristics of non-leaf nodes in the DNS hierarchy based on information readily available, that is, TTL values and arrival characteristics for leaf nodes.

Figure 3 shows how a query to node n results in a cache miss if n is not cached. This will cause proxies to hold the new cached information for TTL and the next miss happens for the first query arriving after the TTL expires. Therefore, the characteristics of the cache miss process, $Y(n)$, can be determined if the query arrival process, $X(n)$, and the TTL value is known. A cache miss at a node leads to an induced query at its parent node. For a non-leaf node n, $X(n)$ is the superposition of the cache miss process

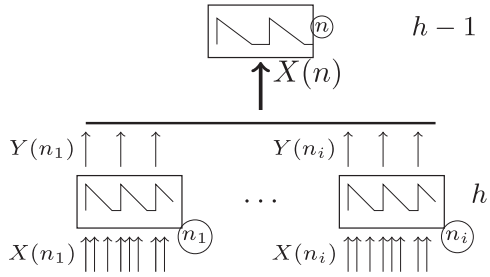


Fig. 3. Bottom-up recursion model.

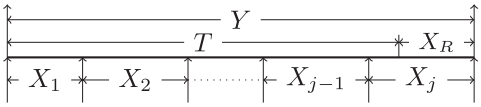


Fig. 4. Relationship between arrival and miss process.

of all its child nodes. Following such a model, we can recursively determine the query arrival process of every non-leaf node from bottom-up.

Let N denote the maximum number of inter-arrival times that can fit with the period T. Let X^j denote the sum of j inter-arrival time of X ($X^j = \sum_{i=1}^{j} X_i$). As depicted in Figure 4, the inter-miss time Y is given by:

$$f_Y(y) = \sum_{j=1}^{\infty} \mathbf{P}\{N = j\} f_{X^j}(y) \,. \tag{1}$$

By definition, $\mathbf{P}\{N < j\} = \mathbf{P}\{X^j > T\}$. Therefore, $\mathbf{P}\{N = j\} = \mathbf{P}\{X^{j+1} > T\} - \mathbf{P}\{X^j > T\}$. Thus, for a given value of T, $\mathbf{P}\{N = j\}$ can be computed recursively up to a certain limit, and then approximated analytically via Gaussian distribution. Furthermore, the Laplace transform of $f_Y(y)$, which is denoted as $\phi_Y(s)$, can be determined as in Eq. (2). From here, estimating the first few moments of Y is straightforward though tedious.

$$\phi_Y(s) = \sum_{j=1}^{\infty} \mathbf{P}\{N = j\} \, [\phi_X(s)]^j \,. \tag{2}$$

Let X_R denote the remaining time before the next arrival after TTL expiration (i.e., the residual time). Then $Y = T + X_R$. $f_Y(y)$ can be expressed as $f_Y(y) = f_{X_R}(y - T)$, which is the TTL-shifted version of f_{X_R}. If the arrival process is Poisson, both X and X_R have the same exponential distribution and determining the distribution of Y is straightforward.

For a *nonleaf* node n, the arrival process is not known directly. Assume a nonleaf node n with K children, and denote n_i the ith child of n ($1 \le i \le K$). Let $X(n_i)$ denote the time to next query to n given that the last query came from its ith child. Let $Y(n_i)$ be the miss process from the ith child and $Y_R(n_i)$ be the corresponding residual time. Clearly, $X(n_i)$ is the minimum of inter-query time of child i and the residual times (Y_R) of all other nodes, that is,

$$\mathbf{P}\{X(n_i) > y\} = \mathbf{P}\{Y(n_i) > y\} \prod_{\substack{j=1 \\ j \neq i}}^{K} \mathbf{P}\{Y_R(n_j) > y\} \,. \tag{3}$$

Based on Eq. (3), and considering that there are K children, the collective arrival process to the parent node n, $X(n)$, has the following inter-arrival time distribution:

$$\mathbf{P}\{X(n) > y\} = \sum_{i=1}^{K} \frac{\mathbf{P}\{Y(n_i) > y\} \cdot \lambda_i}{\lambda} \prod_{\substack{j=1 \\ j \neq i}}^{K} \mathbf{P}\{Y_R(n_j) > y\},$$

where $\lambda_i = \dfrac{1}{\mathbf{E}[Y(n_i)]}$ and $\lambda = \sum_{i=1}^{K} \lambda_i$. To express this in transform space, products turn into convolutions,

$$1 - \phi_{X(n)}(s) = (1 - \phi_{Y_k}(s)) \otimes \bigotimes_{\substack{j=1 \\ n \neq k}}^{K} \frac{1 - \phi_{Y_R(n_j)}(s)}{s}. \tag{4}$$

3.3. Computational Issues and Estimation

The calculations proposed previously are cumbersome since they require actual distributions instead of just the moments. Furthermore, operating in the transform space is ruled out because of the required convolutions. Finally, it is not possible to do recursion over multiple levels unless $X(A^1(n))$ has the same form as $X(n)$. Although one could evaluate all equations numerically, a symbolic computation is more efficient and insightful.

If X is a Poisson process with average rate of λ, then X_R is the residual time with the same distribution as X because of the memoryless property of Poisson process. The density function of Y is the TTL-shifted version of f_X, as in Eq. (5) in which $u()$ is the step function.

$$f_Y(y) = f_X(y - T) = u(y - T)\, \lambda e^{-\lambda (y - T)}. \tag{5}$$

Note that this query miss process at one node has the same characteristics as the output of an $M/D/1/1$ queue. If the parent node has a large number of child nodes, the collective arrival process can be estimated by a Poisson process with a mean equals to the sum of all arrivals (Eq. (6)). This is based on the similar observation as Jackson's Theorem [Bertsekas and Gallager 1992].

$$\lambda_{X(n)} = \sum_{i=1}^{K} \frac{1}{T(n_i) + \frac{1}{\lambda_{X(n_i)}}}. \tag{6}$$

3.4. Stationary State Properties

In the earlier parts of this section, we presented a method to derive the query arrival process ($X(n)$) of any node n in the DNS hierarchy given the query arrival processes of leaf nodes. Coupled with additional knowledge about the TTL value $T(n)$ and the modification process[2] $M(n)$, one can find the stationary state property of node n, i.e., the probability of a node being observed to be *cached*, *obsolete*, or *uncached*.

We study the stationary state distribution of a node based on the observation on any given store-and-flush cycle, as depicted in Figure 5. The expected length of a store-and-flush cycle is $T + \mathbf{E}[X_R]$, where X_R is the remaining time of the arrival of next query when the TTL expires. The probabilities of finding node n in states U, C, or O can be derived rather easily by assuming that the modification process M, and consequently

[2]Modification process is a probabilistic distribution on how often the records are modified.

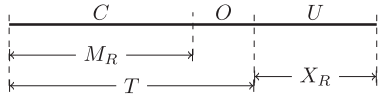


Fig. 5. Stationary state properties.

the residual time of the modification M_R, is independent of the query arrival process (details omitted due to space limitation). The results are:

$$\mathbf{P}\{s(n) = U\} = \frac{\mathbf{E}[X_R]}{T + \mathbf{E}[X_R]}\,, \tag{7}$$

$$\mathbf{P}\{s(n) = C\} = \frac{\mathbf{E}[M_R] \cdot \mathbf{P}\{M_R \le T\}}{T + \mathbf{E}[X_R]}\,, \tag{8}$$

$$\mathbf{P}\{s(n) = O\} = 1 - \mathbf{P}\{s(n) = U\} - \mathbf{P}\{s(n) = C\}\,. \tag{9}$$

The main difficulty in these equations is the estimation of X_R, which is not the normal residual life, and can be quite difficult to compute. For Poisson arrivals, X_R, of course, has the same distribution as the inter-arrival time, and the equations can be evaluated easily.

3.5. Evaluation of QoDNS Metrics

If a proxy server is not under attack, $\mathcal{A}_P$ is always zero. Therefore, we defer the discussion on $\mathcal{A}_P$ to Section 5.

Before we proceed to determine the QoDNS metrics, it is important to note that all of the QoDNS metrics are closely related to the state of the best-matching record. A query to n will be best-matched to its i-th level higher ancestor if itself and all its ancestors before $A^i(n)$ are uncached. Therefore, we define $B^i(n)$ as the event that a query to node n is best matched to its i-level higher ancestor. The probability of observing such event is:

$$\mathbf{P}\{B^i(n)\} = \mathbf{P}\{s(A^i(n)) \neq U\} \cdot \prod_{j=0}^{i-1} \mathbf{P}\{s(A^j(n)) = U\}\,. \tag{10}$$

The overall probability of lookup results can be determined by a weighted-average of the previous equations among all leaf nodes. We define the weight of a leaf node $W(n)$ as the ratio of queries coming to this node:

$$W(n) = \frac{\lambda_{X(n)}}{\sum_{i \in \text{leaf nodes}} \lambda_{X(i)}}\,. \tag{11}$$

3.5.1. Accuracy. If a queried node is present in cache but is obsolete, an obsolete record will be served to client. Therefore, the $\mathcal{A}_O$ metric can be determined as the weighted average of the probability of leaf nodes' record being obsolete:

$$\mathcal{A}_O = \sum_{n \in \text{leaf}} W(n) \cdot \mathbf{P}\{s(n) = O\}\,. \tag{12}$$

3.5.2. Unavailability. If the best-matching record (BMR) is obsolete, the proxy will not be able to follow the delegation to retrieve the queried node, resulting in a lookup failure. Hence, $\mathcal{V}_O$ can be evaluated as the weighted average of the probability of hitting an obsolete BMR:

$$\mathcal{V}_O = \sum_{n \in \text{leaf}} W(n) \cdot \sum_{i=1}^{h} \mathbf{P}\{s(A^i(n)) = O \mid B^i(n)\}\,. \tag{13}$$

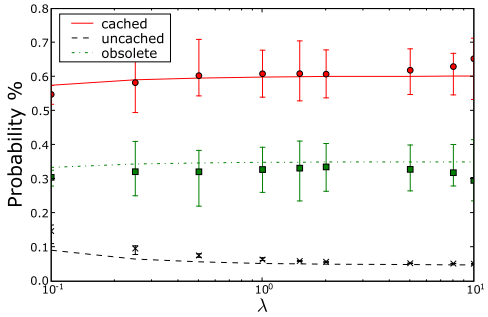


Fig. 6. Validation of stationary state properties. Line plots are derived from analysis; point plots with range are simulation results.

3.5.3. Overhead. If the BMR contains a *current* record, the overhead incurred by the query is proportional to the number of external lookups generated. However, if the BMR is *obsolete*, the query to the remote server will timeout and the proxy server will retransmit for *Ret* times. The actual number retransmits depends on the implementation. We assume $Ret = 3$, which is the default of BIND. Equation 14 presents the average overhead incurred by queries to n. The weighted average of $\mathcal{O}_M(n)$ is the average per-query overhead observed by the proxy:

$$\begin{aligned}\mathcal{O}_M(n) &= \sum_{i=1}^{h} Ret \cdot \mathbf{P}\{s(A^i(n)) = O \mid B^i(n)\}\,, \\ &\quad + \sum_{i=1}^{h} i \cdot \mathbf{P}\{s(A^i(n)) = C \mid B^i(n)\}\,, \qquad (14)\end{aligned}$$

$$\mathcal{O}_M = \sum_{n \in \text{leaf}} W(n) \cdot \mathcal{O}_M(n)\,. \qquad (15)$$

3.5.4. Latency. The latency incurred by successful lookups (lookup failures have been accounted in $\mathcal{V}_O$) is proportional to the number of external servers the proxy needs to contact to fulfill the query. Therefore,

$$\mathcal{L}_N = \sum_{n \in \text{leaf}} W(n) \cdot i \cdot RTT \cdot \mathbf{P}\{s(A^i(n)) = C \mid B^i(n)\}\,. \qquad (16)$$

3.6. Validation of Analytical Results

In this section, we validate our analytic results obtained previously against simulations that faithfully mimic DNS lookup mechanisms.

One might note that the stationary state probabilities in Eqs. (7)–(9) are fundamental to our analysis. It is important to see how these probabilities compare against simulations, which emulate the actual DNS tree behavior with random query arrivals following a Poisson process. Figure 6 shows that the analysis (line plot) and simulation results (point plot with ranges) agree well in all cases. The x-axis plots the query arrival rate and y-axis shows the probabilities of three events: cached, uncached, and obsolete. The midpoints and ranges shown are for 100 simulation runs in each case.

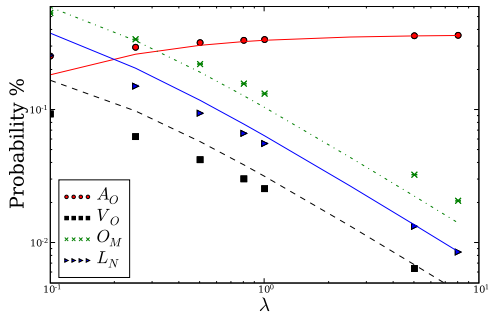


Fig. 7. Validation of QoDNS metrics. Line plots are derived from analysis; point plots with range are simulation results.

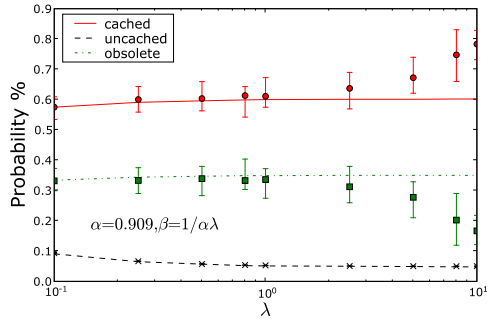


Fig. 8. Validation on gamma distribution. Line plots are derived from analysis; point plots with range are simulation results.

Figure 7 compares the QoDNS metrics obtained from the analytical models and simulations as a function of arrival rate. The continuous lines are generated by the analytical models and the points show the average values from the simulations. Figure 7 verifies that the estimated QoDNS values using Eqs. (10)–(16) (assuming steady-state) provide good estimate for the values observed from the actual simulations.

These results are based on Poisson arrival process. In reality, the arrival process may not be quite Poisson, but the analysis of this case becomes quite challenging. On the other hand, the superposition of a large number of non-Poisson processes should approach Poisson. This calls for a robustness check of the analytic results based on Poisson assumptions. Figure 8 plots analytical results based on Poisson arrivals against simulation results with non-Poisson arrivals as a function of arrival rate. We chose the simulation inter-arrival time as gamma distributed with shape parameter (α) of 0.909. Since errors are likely to be larger when coefficient of variation (CVAR) exceeds 1, we chose $\alpha < 1$ (CVAR $=1/\sqrt{\alpha}$ for gamma distribution). Figure 8 shows that the agreement remains very good except when λ (arrival rate) becomes large. This is to be expected – if the number of superimposed processes is held constant, an increase in arrival rates of individual processes will move the superimposed process away from the Poisson process.

4. STUDY ON STANDARD DNS

We construct the DNS hierarchy based on the following structures: $D(H,N)$, a deterministic DNS tree of H-level with every nonleaf node has exactly N children; $R(H,A,B)$, a random H-level tree with the nonleaf nodes have i children, where i is

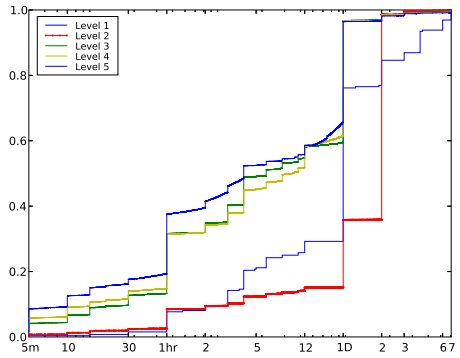


Fig. 9. Distribution of TTL settings on the Internet.

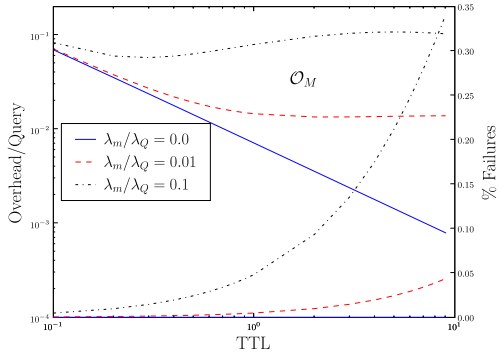


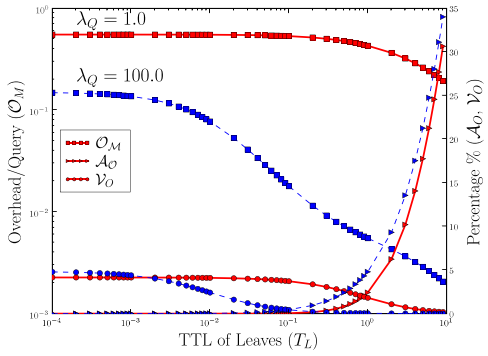
Fig. 10. Impact of TTL settings. TTL is normalized.

uniformly distributed between A and B. Queries arrive at the DNS proxy following a Poisson process with rate λ. The popularity of the leaf nodes follows one of the following models: $E(\lambda)$, all leaf nodes are equally popular and queries arrivals are independent, identically distributed among all leaf nodes. This is the most straightforward case. Another model is $Z(\gamma, \lambda)$: the popularity of leaf nodes follows a Zipf distribution with parameter γ. A recent survey by Jung et al. [2002] shows that, similar to Web objects, domain name objects also follows Zipf distribution.

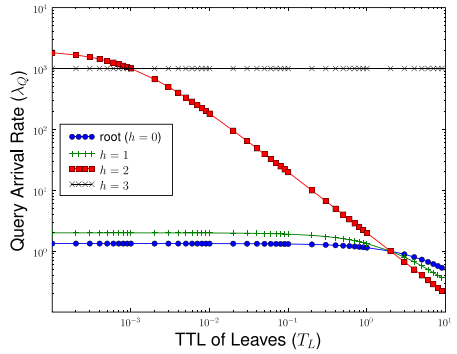
Due to space limitation, the following discussions is based on $D(3, 20)$ with equal popularity, and queries arrives with Poisson parameter λ_Q. Our results are similarly applicable to other cases.

4.1. Impact of TTL

The TTL values of records present one important design option for domain administrators. To understand the realistic TTL settings of the Internet, we collected 2.7 million unique FQDN based on the directory listing at *dmoz.org* [DMOZ]. For each unique node (including leaf node, `www.intel.com.` and nonleaf node, for example, `intel.com.` and `com.`), we found its authoritative servers and queried them directly for its TTL value. From Figure 9, several easy-to-remember numbers, for example, one, two hours or days, dominate the choice of TTL values. This suggests that administrators may not have attempted to optimize the TTL settings for their operational conditions.



(a) Tradeoff Involving DDNS.



(b) Traffic at Different Layers.

Fig. 11. Impact of DDNS. TTL is normalized.

In Figure 10, we vary the TTL value of all nodes in the DNS tree and study the impact on various QoDNS metrics. When there are no modifications, increasing the TTL value reduces lookup overhead and does not cause any failures. However, with moderate modifications ($\lambda_m = 0.01$), a large TTL will increase DNS failures significantly. A lookup failure caused by obsolete referrals ($\mathcal{O}_R$) can cause more overhead than a cache miss. When the modifications are frequent ($\lambda_m = 0.1$), the additional overhead caused by lookup failures could outweigh the reduced overhead by larger TTL value. Therefore, an administrator should choose the TTL carefully based on the tradeoff between overhead and lookup failures.

4.2. Is Dynamic DNS Bad?

Dynamic DNS (DDNS) [Vixie et al. 1997] is a server-side mechanism to alias a dynamic IP address to a static host name. In order to cope with the large modification rate, the server has to set a significantly smaller TTL value (in the order of minutes) so that proxies will not cache the records too long and replied obsolete records to clients. This lead to a natural question on whether the increasing deployment of DDNS will cause significant amount of overhead to the proxies and the servers.

In Figure 11(a), we vary the TTL values of leaf nodes while keeping the TTL value of non-leaf nodes constant. This imitates the behavior of deploying DDNS for end-host IP address but name server themselves are stable. One can notice that if the TTL value of leaf nodes (T_L) are unduly large, it can significantly increase the amount of obsolete records served to clients. Therefore, the administrators have to set a smaller T_L in order to ensure accuracy ($\mathcal{A}_O$). When T_L are comparable with the TTL value of

none-leave nodes (T_{NL}), the overhead grows with the reduced T_L. However, when T_L is much smaller than T_{NL}, further reducing T_L does not increase overhead significantly.

Figure 11(b) looks at message overhead for nodes at different layers. The message overhead for layer 3 nodes increases significantly if a smaller T_L is used. It confirms that administrators do need to plan their server/bandwidth capacity when deploying DDNS. However, going upwards in the hierarchy, the impact of a smaller T_L diminishes. The effect of deploying DDNS is local and has a limited global impact.

5. CACHE POISONING AND ITS MODELING

While QoDNS can be affected by unintentional events such as latency, specific and target attacks such as DNS cache poisoning can also affect QoDNS. In this section, we will introduce a model of DNS cache poisoning to understand its behavior. We will then propose a solution in Section 6 to combact DNS cache poisoning and show how QoDNS can be improved by avoiding cache poisoning.

5.1. DNS Cache Poisoning

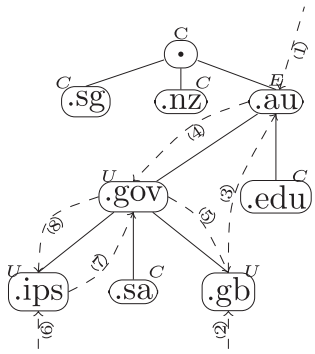
DNS cache poisoning is often used as a foothold for escalating to more harmful subattacks. The attacker can redirect the querier to an IP address which (1) is nonexistent, thus causing Denial-of-Service (DOS) to the querier, (2) is a malicious site that drops Malware/Spyware [Haugsness 2005; Symantec Corporation 2004], or (3) is a masquerade server for man-in-the-middle (MITM) attacks [Green 2005], which includes large scale *phishing* attacks known as *pharming* [Netcraft, Ltd. 2005] and hijacking of emails or SSL sessions. All these attacks were observed in the Mar 2005 incident [Haugsness 2005].

In this section, we first analyze the mechanisms that lead the poison into a cache. Based on these analysis, we extend the model presented Section 3 to consider cache poisoning attacks.

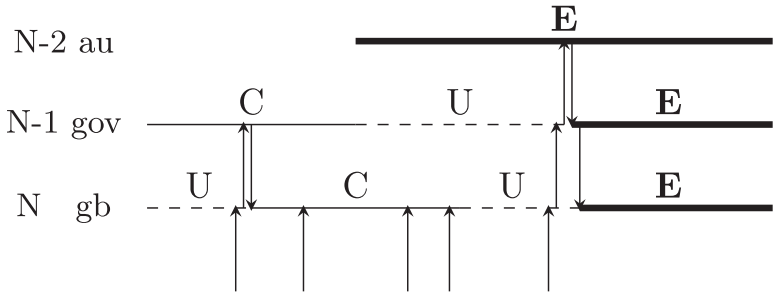
5.2. Analysis of Poisoning Attacks

5.2.1. Poisoning Individual Records. Vulnerabilities in DNS make it possible to inject poisoned records without compromising the entire system. One could attach malicious records as additional information to a legitimate reply and some server implementations will cache the additional records [Microsoft 2005; Schuba 1993] without necessary checks. DNS is UDP-based and the transaction ID (and sometimes source port number) is the sole form of authentication for a DNS reply. Unfortunately, for many implementations, guessing the right transaction ID is possible [Stewart 2003]. The probability of guessing the right transaction ID could be much higher due to weaknesses in the random number generators, use of multiple queries and the consequent birthday paradox [Stewart 2003].

5.2.2. Poison Inheritance. Once a poison is injected, it can spread within the cache or to its clients through normal query and response mechanism. In Figure 12, we model the DNS name space as an inverted tree and queries as "hits" to the leaf nodes. Upon receiving a query hit, the DNS server searches its cache for the *closest match* record and starts a recursive lookup from there. A poisoned closest match can misdirect the query to a malicious name server that returns poisoned response, causing poison to propagate in the cache. In Figure 12, a query for `gb.gov.au` is best matched to `.au`, which is poisoned. The subsequent recursive queries cause gov.au and `gb.gov.au` to inherit the poisoned (P) status of `.au`. The same scenario was observed on March 2005 [Haugsness 2005] when the `.com` entry was poisoned and subsequently many subdomains under `.com`.



(a) Inherited poison.



(b) Hit-through model.

Fig. 12. Cache evolution of poison inheritance.

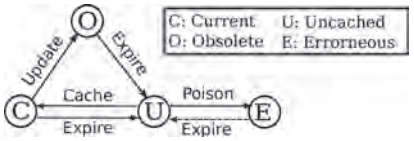


Fig. 13. State transition diagram with poison.

5.2.3. Poisoning the Entire Cache. The DNS server, resolver and client programs are also subject to common issues like buffer overrun and vulnerabilities of the operating system. These include buffer overrun vulnerabilities of early versions 4 and 8 of BIND (Berkely Internet Name Demon) and malwares that modify the host file of a browser or a end-host. In such cases, one must assume the entire cache is poisoned. Additionally, these poisons are persistent since they cannot be flushed out by the TTL mechanism. This article does not attempt to address this type of vulnerabilities are not specific to DNS and are beyond the scope of this article.

5.3. Model of Single Record Under Poisoing Attack

Figure 13 introduce an *erroneous (E)* state to model the state when a record is compromised during cache poisoning attacks. Since cache poisoning attacks do not need to compromised proxy servers, an erroneous records can be flushed when its TTL expires. However, the malicious attacker can set TTL to the largest possible value (e.g., 7 days). We believe such long duration of erroneous records is unacceptable for mission-critical systems like DNS, and assumes poisons are permanent (i.e., dotted arc does not exist in the state machine).

5.4. Modeling of Poison Propagation

Assume $A^1(n)$, the parent node of n, is poisoned at a random time. Denote $\nabla(A^1(n), n)$ the time taken for the poison to propagate from $A^1(n)$ to n. The time of the next query miss of n, which will be resolved by $A^1(n)$, is the time when n gets poisoned. Therefore

$$\nabla(A^1(n), n) = Y_R(n)\,, \tag{17}$$

in which $Y_R(n)$ is the residual time of $Y(n)$.

If the poison is injected at j-level higher parents $A^j(n)$, the time it takes for the poison to reach n is simply the sum of propagation time at each level (Eq. (18)). Note that $A^0(n) = n$ by definition. The distribution of this time $\nabla(A^j(n), n)$ can be found as the convolution shown in Eq. (19),

$$\nabla(A^j(n), n) = \sum_{i=0}^{j-1} \nabla(A^{i+1}(n), A^i(n))\,. \tag{18}$$

$$f_{\nabla(A^j(n),n)}(t) = \bigotimes_{i=0}^{j-1} f_{\nabla(A^{i+1}(n),A^i(n))}(t)\,. \tag{19}$$

Once a leaf node is poisoned, it will serve poison records to clients. With the poison propagating to more leaf nodes over time, the probability of serving poison records ($\mathcal{A}_P$) increases (Eq. (20), F is the cumulative distribution function correspond to f). After a sufficiently long period, all queries to the sub-tree rooted at the poison nodes will receive poison responses.

$$\mathcal{A}_P(t) = \sum_{n \in leaf} W(n) \times F_{\nabla(A^j(n),n)}(t)\,. \tag{20}$$

6. DOMAIN NAME CROSS-REFERENCING (DOX)

If a DNS lookup is affected by a poisoned cache, the response must be different from what dictated by the authoritative server. Naturally, one could verify a record with its authoritative server, assuming the record about the authoritative server is correct. However, verifying every record with the authoritative server is not a viable approach since it renders caching ineffective and hence the DNS unscalable. In this section, we propose a domain name cross-referencing (DoX) system in which peers collaboratively monitor their DNS lookup results. We design a DoX peer to intercept communication between a DNS query and its response. DoX peers perform consistency checks with each other and consult the authoritative server only if suspicious records are noticed.

In this section, we first discuss the other few scenarios that might cause inconsistencies among DNS records at peers (Section 6.1) and how we can handle them. Then we introduce two methods of constructing a P2P network for DoX to facilitate large-scale consistency check (Section 6.2). In Section 6.3, we introduce a new concept of verification cache (vCache) aiming to reduce the overhead of peer verification. Section 6.4 presents the actual DoX algorithm for detecting cache poisoning.

6.1. DNS Inconsistencies

Although DNS records should be consistent with entries dictated by the authoritative servers, there are reasons other than cache poisoning that lead to inconsistencies among clients.

6.1.1. Stale Cache. DNS cache uses a TTL-based weak consistency mechanism that could store and return stale records. This can be resolved if peers ignore the cache and *reload* the record from authoritative servers. As a side effect, DoX also detects inconsistencies caused by modifications and resolves them.

6.1.2. Load Distribution. DNS servers commonly use *round robin* load distribution which rotates the order of the records mapped to the same domain name. Consequently, for a domain name that has three addresses $A1, A2, A3$ mapped to it, one peer could receive $A2, A3, A1$ while the other could receive $A3, A1, A2$. DoX uses *set* comparison for consistency check to handle the permuted records.

6.1.3. Multiple Views. Some modern DNS servers, for example, BIND 9.2, can provide *multiple views* of the zone data. DNS servers using multiple views can return different records based on the IP address of clients. This is often used to separate internal clients from external ones, offering the latter a limited, probably modified, view into the internal networks. Consequently, two peers receiving different *views* will not be able to resolve the inconsistency by querying the authoritative server. DoX avoids raising false alarms in this case by verifying a *record update* instead of a record itself. If the view assignment is static, the older version of the record can serve as an indicator to whether the two peers are under the same view.

6.1.4. Dynamic Mapping. Some content distribution networks (CDNs) use DNS-based redirection for performance improvement and load balancing. Clients could be assigned to servers that are closer or less-loaded, based on the current status. DoX introduces the concept of verification channels to avoid checking names served by CDNs.

6.2. DoX Network as Verification Channel

The entire DNS name space is enormously large. Each DNS server is *interested* in resolving only a very small subset of the name space. If two peers both want to ensure the accuracy of a particular record, the consistency check between them is mutually beneficial. Otherwise, the consistency check is helpful to only one peer and presents only overhead to the other. In addition, if two peers share common queries, it will be more likely they can verify for each other based on local cache. To reduce the overhead and latency of verifications, we propose to construct a *verification channel* to a group of peers that share common interests. A verification channel could be defined in one of the following ways.

Topic Channel. Topic channels use a similar concept as mailing lists or IRC channels by defining *suitable topics* in a certain channel. A topic refers to a set of the DNS names, the correctness of which peers wish to collectively guard. It can be all subdomains under a certain domain, for example, *intel.com* or all domain names listed under a yahoo directory, for example, the “Government/Military/” directory.

Community Channel. A community channel exploits the likelihood that peers from the same community will have more common interests, or at a bare minimum, are willing to bear the overhead for each other. A typical community is “everybody in the company”.

A verification channel is described by a *tracker file* and is coordinated by a *tracker*, which is a daemon program that tracks participating peers and announces them to each other. A peer joins a channel by opening the tracker file and connecting to the tracker for bootstrapping. Every peer in a channel are assigned k peers randomly so that the peers form a k-connected network. We leave it to the tracker administrator to setup the actual definition of a channel, and what channels to exclude (e.g.,

if it is served by CDN),[3] and peers choose the suitable channel based on their own interest.

Using the k-connected network, the peer-perceived network can be managed by upperbounding k; then, regardless of the overall network size, each node is connected to a bounded number of peers. The k peers can also be randomly permuted to minimize the chance of consistently forming the P2P network with malicious peers.

6.3. Verification Cache (vCache)

If a record has been verified earlier and remains unchanged, it is unnecessary to be verified again, even though it might have expired from the local cache and therefore was obtained through DNS lookup again. For this reason, we augment every peer with a separate *verification cache (vCache)* to store previously verified results. A vCache is similar to a normal DNS cache but stores records for an unlimited amount of time. It flushes the old records only if the memory limit is reached.

A natural consequence of using vCache is that a peer will only verify a record with the DoX network if (1) the record does not exist in the vCache or (2) the peer observes a record update in which the version stored in vCache (R^v) is different from the version obtained through standard DNS lookup (R^d). We denote the older version of the record as R_o, the newer version as R_n and the transition as $R_o \rightarrow R_n$. The first case is considered as a special case where $R_o = None$.

6.4. DoX Consistency Check Algorithm

Algorithm 1: *DoXCheck*(Q).

```
1  if R^d = R^v then
2      return OK;
3  else
4      R_o ← R^v, R_n ← R^d;
5      Send R_o → R_n to k peers;
6      Wait for first m results;
7      if #Disagree = 0 then
8          return OK;
9      else
10         R^a ← Authoritative_Server_Lookup (Q) if R^a ≠ R_n then
11             Poison Detected;
12             else if R^a = R_n and #Agree > threshold then
13                 return OK;
14             else
15                 return WARNING;
```

Algorithm 1 describes the DoX checking algorithm. The version stored in vCache (R^v) was verified earlier and the current version obtained through DNS lookup (R^d) might be a newer version. If $R^v \neq R^d$, the peer constructs a verification request in the form of $< Q, R_o \rightarrow R_n >$. The purpose of including R_o in the verification request is to help determine if the verifying and requesting peers are receiving different

[3] With the prevalence of CDN, excluding sites served by CDN might be impractical. In this case, administrators can set up CDN based on likelihood of peers being directed to the same servers. This will increase peers' chance of successfully verifying their records.

views, which we elaborate shortly. A verification request is sent to n remote peers simultaneously but the requester only wait for the first m ($m < n$) responses to guard against peer failures and reduce latency. If the local record is poisoned, there will be "Disagrees" as long as at least one peer is not. (The remote peers decide if they agree based on the algorithm we will describe shortly.) In this case, the local peer will use an "iteration-only" lookup to obtain an authoritative copy (R^a) directly from the authoritative server. In the case of local poisoning, R^a will not be consistent with R_n. If $R^a = R_n$ and a sufficient number of peers "Agree" with the verification request, we consider the verification a success since there might be malicious peers. Otherwise, we raise a warning.

A peer will perform consistency check upon receiving a verification request in the form of $< Q, R_o \rightarrow R_n >$. It replies to the requesting peer with a verification response in the form of <Decision, Info> where the decision can be one of "Agree", "Disagree" or "DiffView" and *Info* contains additional information about this check.

The verifying peer first checks R_n with its local vCache version (R^v). If they are consistent, the verifying peer can "Agree" without further checks since it must have verified this earlier. Otherwise, the verifying peer request an authoritative copy (R^a) from the authoritative server and perform the following checks.

If $R^v = R_o$, the two peers had consistent history record.

— $R^a = R_n$ (*Agree*): The verifying peer observes the same update from the authoritative server. The reason it did not observe this update earlier is likely a stale record in its local DNS cache. Therefore, this verification request actually helps the verifying peer to update the stale record faster than its TTL expiration.
A verifying peer forwards a record update to other peers for preemptive update and verification if it agrees with the requesting peer. Consequently, a record update is made known to every peer in a DoX network after the first peer notices the update. This can significantly reduce obsolete records and failures caused by them. To avoid global synchronization, a peer set the TTL to be a random value between 1 and the original TTL.
— $R^a \neq R_n$ (*Disagree*): The verifying peer does not observe the same update. In particular, if $R^a = R_o$, the verifying peer does not observe a change at all. The requesting peer is most likely *poisoned*.

If $R^v \neq R_o$, the two peers did not have consistent history record. If $R^a = R_n$ (Agree), the authoritative server is probably merging the two peers into the same view. Even if $R^a \neq R_n$ (DiffView), it is possible that they are assigned different *views* by the authoritative server and the verifying peer.

If $R_o = None$ or $R^v = None$, either the requesting or the verifying peer does not have this record in its vCache. The verifying peer agrees if $R^a = R_n$ and disagrees otherwise. This case can be avoided using a safe startup phase to initialize vCache.

7. STUDY ON CACHE POISONING AND SOLUTIONS

To study the performance of DoX, we implemented an event-based simulator for the DNS infrastructure and the proposed DoX scheme. The name space is based on the 2.7 million names listed on *dmoz.org* [DMOZ]. We queried the authoritative server of individual names to find TTL values and found that several popular ones, for example, 1 hour, 2 hours, 1 day, and 2 days, dominate. Queries arrive with a Poisson arrival process of rate λ_q to leaf nodes and the popularity of names follows a Zipf-distribution. Every node in the namespace has equal opportunity to be modified by their respective administrators. Collectively, modifications follow a Poisson arrival of rate λ_m. The root

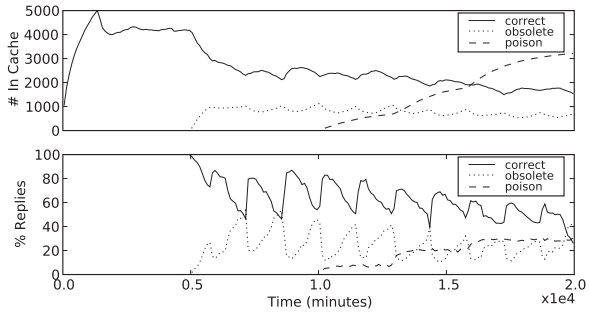


Fig. 14. Evolution of a standard DNS cache. The diurnal pattern arises from TTL expiration.

node is excluded from being modified or poisoned since every DNS server has a *root hint* file and record about the root servers has not changed for many years.

7.1. Evolution of A Standard DNS Cache Under Attack

Figure 14 presents the evolution of a standard DNS cache under various conditions. Ideally, one hopes more records are cached correctly so that queries can be answered locally. Arrivals of queries cause records to be cached but the TTL expiration mechanism also flushes records. Since all records have a TTL expiration, a certain arrival rate will only keep a certain number of records in the DNS cache. This is observed in Figure 14 before $t = 5000$. Starting from $t = 5000$, modifications to DNS records are introduced. Consequently, certain cached records become obsolete and affects the accuracy.

Poison is injected at time 10000 for `.com`, at 12500 for `.net` and at 15000 for `.org` respectively. Poisons do not start to propagate or cause poisoned records being returned for a small period. This is because a typical application does not query `.com` directly. Instead, they query for leaf nodes like `www.cnn.com` and if `www.cnn.com` or `cnn.com` is cached, the recursive lookup will start from the best-match instead of `.com`. However, once queries starts to reach the poisoned nodes, it propagate down and spreads in the cache.

To inflict maximum damage, malicious attackers are likely to set a very large TTL value for the poisoned record. Although most DNS servers have a maximum allowable TTL value (default is 7 days for BIND), it is significantly longer than the TTL values of most DNS records. Consequently, we observe a steady increase of poisoned cache over time until all names under the `.com` (or `.net` or `.org`) branch is poisoned. Because of this poison propagation mechanism, the attacker need to succeed only once to affect a large number of users.

7.2. DoX for Poison Detection and Removal

Figure 15 presents the evolution of DoX-protected DNS cache based on a strict strategy in which every answer obtained through DNS lookup is checked before it is relayed to the querier. As long as there is at least one verifying peer that is not poisoned, the requesting peer will be able to detect the poison and remove it by flushing. In Figure 15, poison is injected at time 2000 for `.com`, at 4000 for `.net` and at 6000 for `.org`. In the top figure, the injected poison can stay in the cache for a while until a query *hits* it. However, it is detected by DoX and removed immediately if it causes any poisoned response to be returned. Since DoX stops poison from propagating, we do not observe poisoned records in the cache or poisoned replies as in Figure 14. In addition, the percentage of correctly cached records is significantly increased.

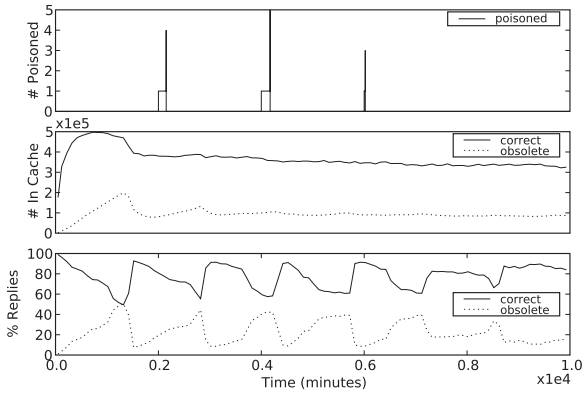


Fig. 15. Poison detection and removal. The diurnal pattern arises from TTL expiration.

7.3. DoX for Improving Cache Consistency

In DoX, a verifying peer will forward a valid record update to other peers in addition to sending the confirmation to the requesting peer. Consequently, all peers in the DoX network will update to the current record as soon as one peer notices the update, independent of the DoX topology. Figure 16(a) shows that the average number of obsolete records decreases when the size of the DoX network increases. Peers in a large DoX network will observe negligible amount of obsolete records.

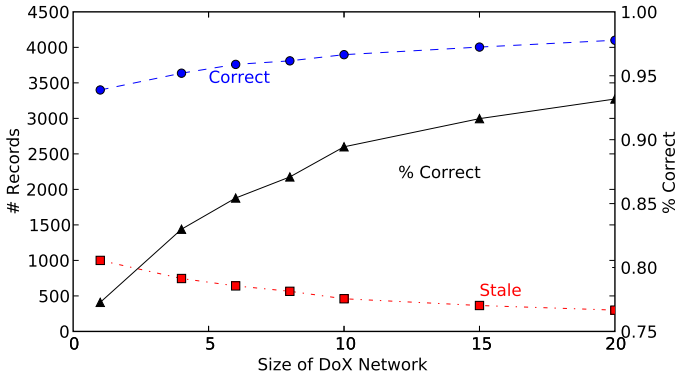
7.4. Comparison of DNSSEC and DoX

DNS Security Extensions (DNSSEC) propose to use public-key cryptography to authenticate resource records in order to prevent cache poisoning attacks. DNSSEC augments DNS with a few additional RR types so that a zone can be cryptographically signed by authoritative servers and verified resolvers. A signed zone includes a RRSIG RR which is the signature signed by the private key of the zone owner. The DNSKEY RR contains the corresponding public key, which can be used to authenticate the records against the RRSIG. To verify that the DNSKEY itself is not tempered, the Delegation Signer (DS) RR at the parent node contains the public key to verify the DNSKEY RR at child nodes. Given that resolvers can retrieve the DNSKEY of the root node using some out-of-band mechanism, there is an authentication chain starting from the trusted root node to the RRSet of interest.

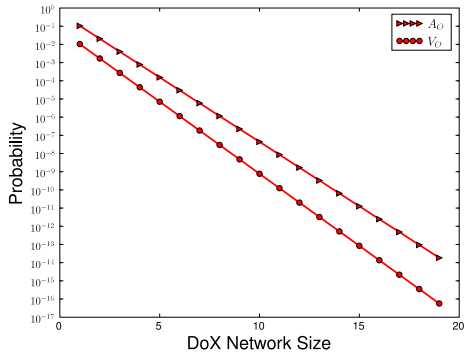
7.4.1. Accuracy and Availability. DNSSEC guarantees the authenticity of a record with at same strength as public-key cryptography used. However, key management remains a challenging research issue for DNSSEC. The administrators need to keep the private keys online if they were to regenerate the signatures periodically. They also need to roll over to new keys periodically to prevent the key being broken. However, the security issues of DNSSEC are beyond the scope of this article. We assume that DNSSEC can prevent cache poison perfectly.

Even though DNSSEC guarantees the authenticity of a record, it does not reduce the number of inaccurate obsolete records. This is because DNSSEC uses the same TTL-based cache expiration mechanisms as standard DNS and hence does not improve cache consistency. Consequently, DNSSEC have the same probability of serving obsolete records to clients ($\mathcal{A}_O$) or the availability of DNS records.

The DoX network detects a modification (and updates to it) as soon as any one peer detects it. Denote $DoX^O(n)$ the probability for a DoX peer to have node n in *obsolete* status. This happens only if (1) it is obsolete in local cache and (2) all other peers have



(a) Cache Consistency.



(b) Accuracy and Availability.

Fig. 16. Impact of DoX on QoDNS.

this node in either *obsolete* or *uncached* status. Therefore, in a $DoX(M,N)$ network, M being the voting threshold and N the size of the P2P network, the probability of node n being obsolete in a proxy cache is

$$\mathbf{P}\{s^{DoX}(n) = O\} = \mathbf{P}\{s(n) = O\} \cdot (1 - \mathbf{P}\{s(n) = C\})^{N-1}.$$

One can see that the probability for a cached record in DoX being obsolete is significantly lower than standard DNS. When N is large, a DoX network will observe almost no obsolete records, thus achieving strong cache consistency. Consequently, the accuracy and availability can be improved. This is confirmed in Figure 16(b). With the increasing size of DoX network size, the probability of sending obsolete records to clients or having lookup failures due to obsolete records becomes negligible.

7.4.2. Overhead and Latency. The additional overhead introduced by DNSSEC mainly comes from zone inflation caused by cryptography signatures and related keys (RRSIG, DNSKEY and DS RRs). Gudmundsson [2001] suggests that these signatures, depending on the cryptography algorithm used, range from about 80 octets to 800 octets, with most signatures below 200 octets. A practical bound for the overhead is that UDP (most common transport-layer protocol) allows messages with at most 512 octets while the DNSSEC standard recommends a limit of 4000 octets.

To determine the additional overhead of deploying DNSSEC requires the knowledge of the size of original DNS data and the cryptography algorithm used. We assume the original DNS data is 512 octets (upper bound) and RSA/SHA-1 [Eastlake 2001], which is the only mandatory algorithm defined. We further assume that the keys are 1024-bit

long. In this case, the RRSIG, DNSKEY, and DS RRs jointly introduces at least about 300 octets overhead. This is about 60% additional overhead than standard DNS.

Zone inflation is however not expected to significantly affect the latency observed by end clients. This is because the latency is dominated by RTT. Transmission time contributes a negligible part to the total latency even though it increases proportional to the data size. On the other hand, processing latency of DNSSEC could be significantly larger than standard DNS.

DoX peers incur additional messaging overhead when verifying with peers for a record update. For each update observed or notified by other peers, a peer send/receive a verification request (in the form of $< old \rightarrow new >$) to/from M other peers. Note that this messaging overhead is incurred only if there are modifications. Equation (21) gives the per-query additional overhead incurred by DoX.

$$DoX^O = 2 \times M \times \frac{\lambda_M}{\lambda_Q}. \tag{21}$$

Since the verification cache of DoX peers are flushed only if storage limit is reached, DoX could use all the storage available. However, this does not impose a practical concern because (1) DoX can gracefully recover from flushed verification cache and (2) storage does not pose a major constraint for DNS cache as compared to Web cache, in which object size are significantly larger.

DoX incurs additional latency only for the first peer that notices the record update. In the worst case, a verifying peer needs to traverse the entire delegation chain (H levels) to verify a request. The upper bound of additional latency incurred by DoX is:

$$DoX^L = \frac{H \times RTT}{N} \cdot \frac{\lambda_m}{\lambda_Q}. \tag{22}$$

Based on this discussion, we can compare DNSSEC and DoX in terms of QoDNS so that an informed decision can be made by administrators.

— $\mathcal{A}_P$. Both DNSSEC and DoX can reduce the probability of sending out poison records to zero, but based on different assumptions. DNSSEC assumes safe key management while DoX assumes the correctness of at least one peer.
— $\mathcal{A}_O$ and $\mathcal{V}_O$. DNSSEC does not improve on these two metrics while DoX can significantly reduce them.
— $\mathcal{O}_M$. DNSSEC will incur at least 60% additional the overhead. For DoX, this depends on the modification process. If the arrival rate of modifications is 1/10 of that of queries, and with a DoX network using node degree of 3, the overhead is comparable with DNSSEC. But we expect that the arrival rate of modifications is significantly lower than that.
— $\mathcal{L}_N$. DNSSEC does not increase the latency directly. DoX will not increase the average latency significantly but might increase the worst case latency.

8. RELATED WORK

Jung et al. [2003] present a mathematical model for TTL-based caching. They model query arrival with a renewal process and derive cache hit/miss ratios. However, their model ignores the hierarchical dependencies among objects in DNS cache and implicitly assumes the objects stored in the cache are independent of one another. Therefore, their model is insufficient for understanding many aspects of DNS cache including the impact of lower level TTL at higher levels, or the propagation of cache poisoning down the hierarchy.

There are several works on modeling hierarchical caches [Che et al. 2002; Hou et al. 2004]. They consider a layered deployment of Web proxies, each with its own cache.

Lower-layer proxies, upon a cache miss, will forward the query to higher-layer ones. Thus, cached objects in lower-layer caches is synchronized with that in high-layer parent. In the case of DNS, the hierarchical dependency is *internal*, that is, such hierarchy dependency exists among DNS objects themselves contained in the same cache.

Kolkman [2005] measures the effects of deploying DNSSEC on CPU, memory, and bandwidth consumption on authoritative servers (as compared to standard DNS). Curtmola et al. [2005] compared two DNSSEC variants – the IETF version using public key cryptography and another using symmetric key. Their approaches are to deploy the compared schemes on a testbed and compare the performance.

Cao and Liu [1998] study several approaches to achieve strong cache consistency for the web, including adaptive TTL, polling-every-time, and proactive server invalidation. A server invalidation scheme can also be used for improving DNS cache consistency; however, the main difficulty is that the DNS server would have to maintain states for known clients. Invalidation does not protect against cache poisoning.

Cohen and Kaplan [2001] proposed proactive caching of DNS records in which they use renewal policies to refresh selected cache entries by issuing unsolicited queries. In addition, they propose a *simultaneous-validation* (SV) in which the end-host will use an expired DNS entry to connect to the web server, but the content will be served only if the DNS entry is validated by the SV queries. Their focus is on improving cache hit rate.

Several recent work, including Overlook [Theimer and Jones 2002], DDNS [Cox et al. 2002], CoDNS [Park et al. 2004] and CoDoNS [Ramasubramanian and Sirer 2004], proposed to use structured overlays to provide DNS lookup services. Exploiting the scalability and reliability of the underlying overlay network, these new DNS protocols achieve faster lookups and fewer failures. However, none of these works addresses the accuracy of DNS lookups. In fact, blindly relying on untrustworthy peers for DNS lookups could be dangerous.

Poole and Pai [2006] proposed a P2P system that can detect poisoned DNS caches. However, the proposed system does not provide detail on how to avoid false positives due to practical DNS implementations poison cache identification (e.g., Multiple view). Wong and Nikander [2010] proposed a new architecture to secure DNS, basing the architecture on the idea of decoupling identity and location; using certificate look-up to establish authority, and a dns-like service to discover location. Dagon's presentation [Dagon 2008] and Alexiou et al.'s [2010] analytical work discussed the danger in DNS cache poisoning, with Dagon [2008] referring to some recent RFCs to combat the attack, and Alexiou et al. [2010] studied the Kaminsky attack in depth. Suggestion in all works require implementation change at the server side, in which the client is dependent on the adoption rate of the implementation change. DoX on the other hand can be implemented on the client side, allowing the clients to improve their security independent of the servers' architectural or implementation updates.

9. CONCLUSION

This article proposed to evaluate the quality of Domain Name Service (QoDNS) as a whole picture instead of improving on one aspect and ignoring the others. We first presented a comprehensive set of metrics, including accuracy, availability, latency and overhead, to evaluate QoDNS. DNS proxy is one of the most important component on improving QoDNS. However, early modeling and analysis on Web proxy is insufficient for DNS proxy because of the hierarchical naming and delegation structure, and the consequent best-matching algorithm and referral-following algorithm. We presented a hierarchical analytical model for DNS proxy, which can assist the understanding of DNS and can be used to study the tradeoffs among of critical parameters like TTL or understand the impact of dynamic DNS. Being on the critical path between DNS

clients and servers, DNS proxies and their caches are often targets of attacks. We extended our model to analyze the poison propagation in DNS proxy under such attacks. We went on to propose a novel P2P-based solution for cache poisoning attack, which requires only cooperation between some DNS proxies and does not require changes to the current DNS infrastructure. We performed QoDNS-based comparison between DNSSEC and DoX to understand their relative merits.

REFERENCES

Alexiou, N., Basagiannis, S., Katsaros, P., Dashpande, T., and Smolka, S. 2010. Formal analysis of the Kaminsky DNS cache-poisoning attack using probabilistic model checking. In *Proceedings of the IEEE 12th International Symposium on High-Assurance Systems Engineering (HASE)*. 94–103.

Arends, R., Austein, R., Larson, M., Massey, D., and Rose, S. 2005. DNS security introduction and requirements – RFC 4033.

Bellovin, S. M. 1995. Using the domain name system for system break-ins. In *Proceedings of the USENIX Security Symposium*.

Bertsekas, D. P. and Gallager, R. 1992. *Data Networks* 2nd Ed. Prentice-Hall.

Cao, P. and Liu, C. 1998. Maintaining strong cache consistency in the world wide web. *IEEE Trans. Comput. 47*, 4, 445–457.

Che, H., Tung, Y., and Wang, Z. 2002. Hierarchical web caching systems: Modeling, design and experimental results. *IEEE J. Sel. Areas Comm. 20*, 7, 1305–1314.

Cohen, E. and Kaplan, H. 2001. Proactive caching of dns records: Addressing a performance bottleneck. In *Proceedings of the Symposium on Applications and the Internet*. 85–94.

Cox, R., Muthitacharoen, A., and Morris, R. 2002. Serving DNS using a peer-to-peer lookup service. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'01) (Revised Papers)*. Springer-Verlag, 155–165.

Curtmola, R., Sorbo, A. D., and Ateniese, G. 2005. On the performance and analysis of DNS security extensions. In *Proceedings of the International Conference on Cryptology & Network Security*.

Dagon, D. 2008. DNS poisoning: Developments, attacks and research directions. In *Proceedings of the 17th USENIX Syposium*.

DMOZ. DMOZ – Open directory project. www.dmoz.org.

Eastlake III, D. E. 2001. RSA/SHA-1 SIGs and RSA KEYs in the Domain Name System (DNS) – RFC 3110.

Evers, J. 2005. DNS servers: An Internet's Achille's heel. http://news.cnet.com/DNS-servers–an-Internet-Achilles-heel/2100-7349_3-5816061.html.

Green, I. 2005. DNS spoofing by the man in the middle. http://www.sans.org/rr/whitepapers/dns/1567.php.

Gudmundsson, O. 2001. DNSSEC and IPv6 aware server/resolver message size requirements – RFC 3226.

Haugsness, K. 2005. DNS poisoning scam raises wariness of 'pharming'. http://isc.sans.org/presentations/dnspoisoning.php.

Hou, Y., Pan, J., Li, B., and Panwar, S. 2004. On expiration-based hierarchical caching systems. *IEEE J. Select. Areas Commun. 22*, 1, 134–150.

Hughes, A. S. and Touch, J. 1999. Cross-domain cache cooperation for small clients. In *Proceedings of Network Storage Syposium*.

Jung, J., Sit, E., Balakrishnan, H., and Morris, R. 2002. DNS performance and the effectiveness of caching. *IEEE/ACM Trans. Netw. 10*, 5, 589–603.

Jung, J., Berger, A. W., and Balakrishnan, H. 2003. Modeling TTL-based Internet caches. In *Proceedings of the IEEE INFOCOM*.

Kolkman, O. 2005. Measuring the resource requirements of DNSSEC. Tech. rep. RIPE-352, RIPE NCC/NLnet Labs. Sept.

Liu, C. 2007. Handicapping new DNS extensions and applications. http://www.onlamp.com/pub/a/onlamp/2007/01/11/dns-extensions.html.

Microsoft. 2005. Description of the DNS server secure cache against pollution setting. http://support.microsoft.com/kb/316786/EN-US/.

Naraine, R. 2002. Massive DDOs attack hit DNS root servers. http://www.internetnews.com/devnews/article.php/1486981.

Netcraft, Ltd., N. 2005. DNS poisoning scam raises wariness of 'pharming'. http://news.netcraft.com/archives/2005/03/07/dns_poisoning_scam_raises_%wariness_of_pharming.html.

Park, K., Pai, V. S., Peterson, L., and Wang, Z. 2004. CoDNS: improving DNS performance and reliability via cooperative lookups. In *Proceedings of the 6th Conference on Operating Systems Design & Implementation (OSDI'04)*. USENIX Association, Berkeley, CA, 14–14.

Poole, L. and Pai, V. S. 2006. Confidns: Leveraging scale and history to improve dns security. In *Proceedings of WORLDS*. Vol. 6.

Ramasubramanian, V. and Sirer, E. G. 2004. The design and implementation of a next generation name service for the Internet. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'04)*. ACM, New York, 331–342.

Schuba, C. 1993. Addressing weakness in the Domain Name System protocol. M.S. thesis, Purdue University.

Stewart, J. 2003. DNS cache poisoning – the next generation. http://www.securityfocus.com/guest/17905.

Symantec Corporation 2004. Symantec gateway security products DNS cache poisoning vulnerability. http://securityresponse.symantec.com/avcenter/security/Content/2004.06.%21.html.

Theimer, M. and Jones, M. B. 2002. Overlook: Scalable name service on an overlay network. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*. 52–61.

Vixie, P., Thomson, S., Rekhter, Y., and Bound, J. 1997. Dynamic updates in the Domain Name System (DNS UPDATE) – RFC 2136.

Wong, W. and Nikander, P. 2010. Secure naming in information-centric networks. In *Proceedings of the Re-Architecting the Internet Workshop (ReARCH'10)*. ACM, New York, 12:1–12:6.