This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2007 proceedings.

A Proxy View of Quality of Domain Name Service

Lihua Yuan
ECE, UC Davis
lyuan@ece.ucdavis.edu

Krishna Kant
Intel Corporation, OR
krishna.kant@intel.com

Prasant Mohapatra
CS, UC Davis
prasant@cs.ucdavis.edu

Chen-Nee Chuah
ECE, UC Davis
chuah@ece.ucdavis.edu

Abstract—**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* or QoDNS, let alone a thorough evaluation of it. This paper 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. After validating our model against simulation results, we extend it to study the impact of DNS cache poisoning attacks and evaluate various DNS proposals with respect to the QoDNS metrics. In particular, we compare the performance of two newly proposed DNS security solutions: one based on cryptography and one using collaborative overlays.**

I. Introduction

The Domain Name System (DNS) is one of the most valuable components in the Internet infrastructure. Almost all applications, e.g. 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 information about mail exchange, serve as public-key infrastructure, and provide dynamic load distribution. Given this, the notion of *quality of Domain Name Service* (QoDNS) is an important one, but currently there are no metrics to quantify it, let alone evaluate it thoroughly.

In this paper, we define such a concept based on *accuracy*, *availability*, *latency*, and *overhead* of DNS service. 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. Since a successful DNS lookup is required before a connection attempt, the latency incurred will be observed by every application, and in many cases, contributes a significant portion of it [1].

The DNS consists of a hierarchy of authoritative servers (Section II-A) and a large number of proxy servers that cache the resource records, thereby making the lookup more efficient and scalable. We propose an analytical model that captures this structure and provides estimates for QoDNS metrics.

A major area of current concern regarding DNS relates to *DNS cache poisoning* attacks [2], where incorrect or malicious records are injected into the DNS proxy. This not only results in a potential denial of service (due to wrong IP address provided) but can also be exploited as a foothold for more harmful sub-attacks such as Man-In-The-Middle attacks [3] or large-scale phishing attacks known as *pharming* [4]. The current DNS protocol makes it surprisingly easy to inject poisoned resource records and does not require compromising the proxy servers themselves. Furthermore, the hierarchical structure of DNS propagates the poison down the tree and thereby can affect a large number of clients, as witnessed in March 2005 [5]. Thus, capturing the impact of poisoning is a crucial attribute for a useful QoDNS metric.

This work is partly supported by Intel and NSF NeTS NBD-0520320.

Currently, there are two classes of solutions proposed for thwarting cache poisoning attacks in the DNS: (1) those based on cryptography, e.g. public-key based DNS Security Extension (DNSSEC) [6] and its symmetric-key based variant, SK-DNSSEC [7], and (2) those based on peer-to-peer cooperative checking, e.g. Domain Name Cross referencing (DoX) [8]. We shall apply our QoDNS metric to evaluate both of these alternatives.

The contributions of this paper are:

- We define a set of metrics to comprehensively evaluate and compare QoDNS (Sec. III).
- We present an analytical model of DNS proxies for quantitative evaluation of QoDNS (Sec. IV). Our model can be used to determine the tradeoffs of key parameters like TTL. In particular, we use it to study the impact of the increasing popularity of *Dynamic DNS* (Sec. V).
- We use our model to compare existing solutions to thwart DNS cache poisoning and provide insight into which solution might be most appropriate in what situations.

The structure of the paper is as follows. Section II discusses DNS structure and related work. In section III, we motivate the definition of QoDNS, and in section IV we present analysis of basic DNS from QoDNS perspective. Section V presents some numerical results on standard DNS based on the model. Section VI then discusses analysis of some DNS variants that attempt to thwart DNS poisoning attacks. Finally, section VII concludes the paper and discusses future work.

II. Background and Related Work

A. DNS Structure

DNS is a hierarchically organized, distributed database starting with records for top level domains (e.g., .com or .edu) and extending down to a few levels. The fully-qualified domain name, e.g. www.intel.com, describes a complete path from the root node to this node. Each domain is associated with a set of *resource records* (RRset), which contain information about this domain or referrals to its sub-domains.

A DNS proxy can find the record for a node n by querying the DNS server(s) listed in the Name Server (NS) record of its parent node $A^1(n)$. Since the RRset for the root node is always available on every proxy as the *root hint* file, a DNS proxy can find the RRset of any domain by starting from the root, and recursively following the referrals. A DNS proxy caches all such RRsets for the period of their TTL. The TTL of a node is assigned by its authoritative server. To improve performance and reduce overhead, the proxy searches its cache for the *best-matching* record (BMR) and starts the recursive lookup from there instead of the root node every time.

Figure 1 presents a simple state machine for various states of a record (or RRset). Initially, the record is in *un-cached* state indicated as U. The arrival of a query causes the RRset 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). Finally, if the query to AS by the proxy returns a poisoned record, the record state becomes erroneous (E). Since the proxy server itself has not been compromised, it can flush a poison record when its TTL expires. However, the malicious attacker can set TTL to the largest possible value (e.g., 7 days). This paper assumes poisons are permanent (i.e., dotted arc does not exist in the state machine).

A name lookup through the 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 to node n results in a cache miss, the proxy needs to obtain the NS record of n from the parent node, $A^1(n)$, to contact the AS. This is considered as an *induced query*. An induced query to node $A^1(n)$, if it results again in a cache miss, induces a query to its own parent node, $A^2(n)$. This process iterates until a hit happens. The first ancestor node with cached information provides the best-matching record (BMR). 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 node n. Records of nodes on the path from the BMR to n will be *cached* during this process.

Because of the hierarchical nature of DNS, a poisoned record can percolate down as illustrated in Figure 2. Assume 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 (*E*) status of .au. The TTL at each level could be set to a large value, thereby making the poison both long lasting and widespread. Such a scenario was observed on March 2005 [5] when the .com entry was poisoned. This propagation mechanism offers an amplification effect to the injected poison and makes high-level nodes especially attractive to malicious attackers.

B. Related Work

Jung et al. [9] 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 [10, 11]. 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*, i.e. such hierarchy dependency exists among DNS objects themselves contained in the same cache.

Kolkman [12] measured the effects of deploying DNSSEC on CPU, memory, and bandwidth consumption on authoritative servers (as compared to standard DNS). Curtmola [13] 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.

TABLE I
SUMMARY OF SYMBOLS

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

III. QoDNS Metrics

In the following subsection, we provide a formal definition of QoDNS based on these key aspects: accuracy, availability and latency/overhead.

A. Accuracy of DNS Resolutions

A DNS client may receive inaccurate mappings for two main reasons: (1) obsolete and (2) 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 set to be fairly large and there is rarely a capacity based eviction of the records. This situation tends to aggravate the problem of obsolete records, leading to failed communication attempts.

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, any query from lower levels of the hierarchy will propagate them further (to a much larger set of nodes).

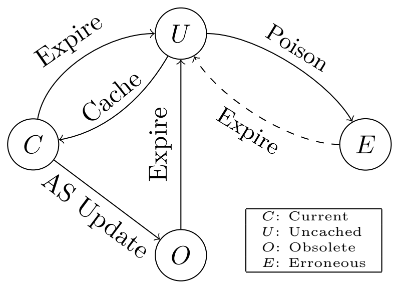

Fig. 1. State Transition Diagram

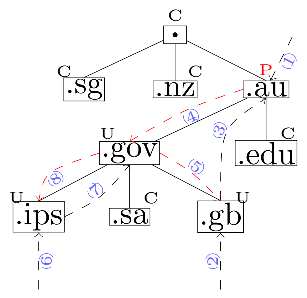

Fig. 2. Illustration of Poison Propagation

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 can be considered more dangerous since they can be exploited for further, more serious attacks.

B. 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 [14]. However, caching can also negatively affect the availability. In particular, if the desired record is uncached, and the BMR that the proxy finds in its 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 here if the RRset (that provides the name to IP mapping to the client) is obsolete, this situation will result in a *application failure* when the client attempts to use the IP address. This situation is covered by the accuracy aspect (see section III-A), and is *not* an availability issue for the DNS. In both cases, the TTL expiry and subsequent retrieval of updated record from AS 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 paper. 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.

C. 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 [1]. 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, e.g., 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 some specialized circumstances.

D. Definition of QoDNS

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

$\mathcal{A}_O$	Probability that the service provides an obsolete resource record.
$\mathcal{A}_P$	Probability that the service provides a poison resource record. If the proxy is not under malicious attack, this metric is always zero.
$\mathcal{V}_O$	Overall unavailability of DNS service under the assumption of perfect node level availability (i.e., we exclude the impact of node failures, as discussed earlier).
$\mathcal{L}_N$	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.
$\mathcal{O}_M$	Network overhead of DNS service, measured as average number of packets send and received per query. We make the simplified assumption that all queries and responses can fit into one packet of the same size.

IV. Analysis of Standard DNS

A. Recursive Model for Query Arrival Characteristics

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, i.e. TTL values and arrival characteristics for leaf nodes.

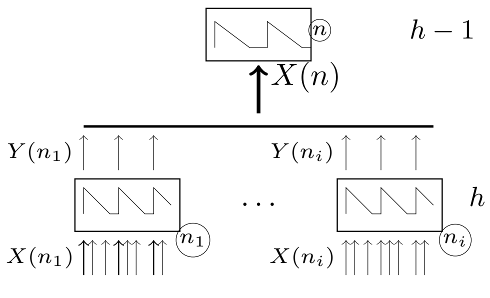

Fig. 3. Bottom-Up Recursion Model

Figure 3 shows how a query to node n results in a cache miss if n is not cached. This will cause node n 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 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.

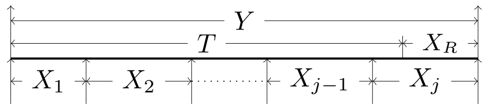

Fig. 4. Relationship Between Arrival and Miss Process

Let N denote the maximum number of inter-arrival times that can fit 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.

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

Let X_R denote the remaining time before the next arrival after TTL expiration. Then $Y = T + X_R$. Therefore, $f_Y(y)$ 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 *non-leaf* node n, the arrival process is not known directly. Assume a non-leaf 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, i.e.,

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

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:

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

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

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

B. Computational Issues and Estimation

The calculations proposed above 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 [15].

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

C. 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 $M(n)$, one can find the stationary state property of node n, i.e. the probability of a node begin observed to be *cached*, *obsolete*, or *uncached*.

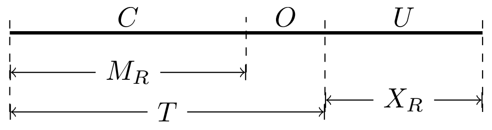

Fig. 5. Stationary State Properties

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 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.

D. Evaluation of Performance 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 VI-A.

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:

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

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

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

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 being obsolete:

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

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:

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

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:

$$\begin{aligned}\mathcal{O}_M(n) &= \sum_{i=1}^{h} Ret \cdot \mathbf{P}\{s(A^i(n)) = O \mid B^i(n)\}, \\ &\quad + \sum_{i=1}^{h} i \cdot \mathbf{P}\{s(A^i(n)) = C \mid B^i(n)\}, \qquad (14) \\ \mathcal{O}_M &= \sum_{n \in \text{leaf}} W(n) \cdot \mathcal{O}_M(n). \qquad (15)\end{aligned}$$

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,

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

E. Validation of Analytic Results

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

It can be noted that the stationary state probabilities in Equations (7-9) are fundamental to our analysis. As a result, it is important to see how these probabilities compare against our simulations, which emulate the actual DNS tree behavior with random query arrivals following a Poisson process. Figure 6 shows that the analysis and simulation results 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.

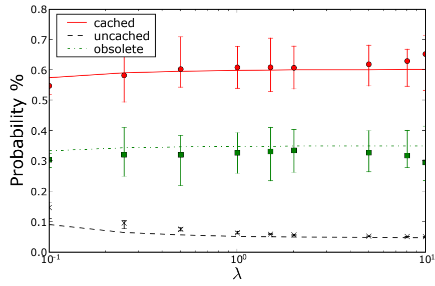

Fig. 6. Validation of Stationary State Properties

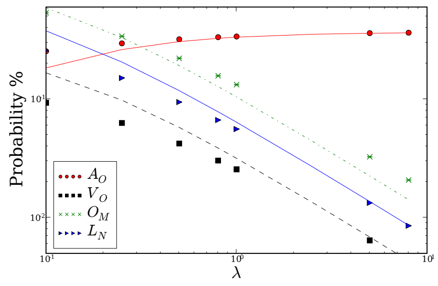

Fig. 7. Validation of QoDNS metrics

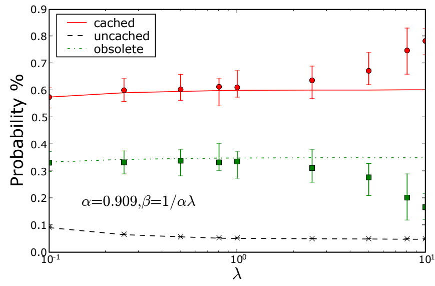

Fig. 8. Validation on Gamma Distribution

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 Equations 10-16 (assuming steady-state) provide good estimate for the values observed from the actual simulations.

The above 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 w/ 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.

After verifying the correctness and applicability of our analytical model, we use it to study the performance of standard and dynamic DNS in the following section.

V. Study of Standard DNS

We construct the DNS hierarchy based on the following structures: $D(H, N)$, a deterministic DNS tree of H-level with every non-leaf node has exactly N children; $R(H, A, B)$, a random H-level tree with the non-leaf nodes have i children, where i is 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. [16] shows that, similar to Web objects, domain name objects also follows Zipf distribution.

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

A. Impact of TTL

The TTL values of RRsets presents 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* [17]. For each unique node (including leaf node, www.intel.com. and non-leaf node e.g. 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, e.g. 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.

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.

B. Is Dynamic DNS Bad?

Dynamic DNS (DDNS) [18] 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, 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,

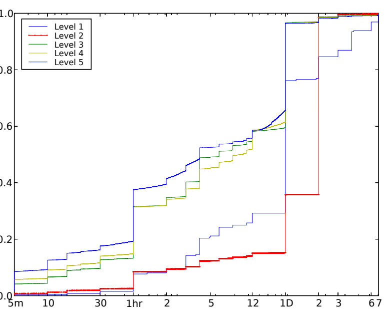

Fig. 9. Distribution of TTL Settings on the Internet

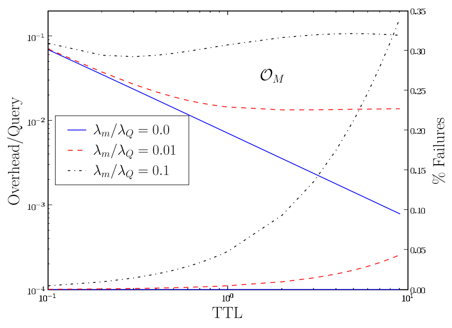

Fig. 10. Impact of TTL Settings

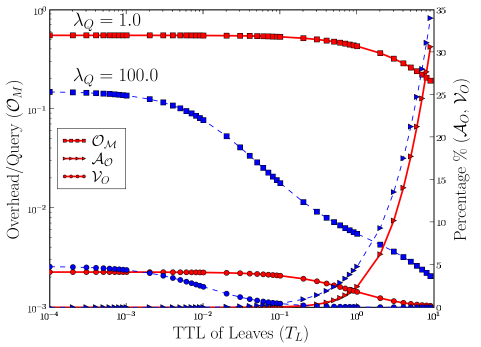

Fig. 11. Impact of Dynamic DNS

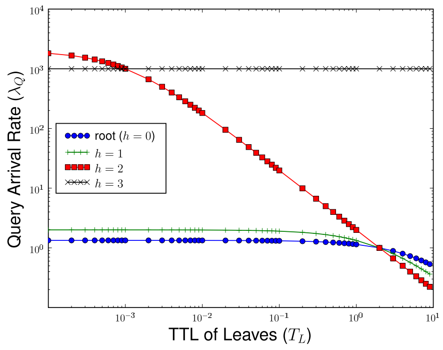

Fig. 12. Traffic at Different Layers

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 significantly smaller than T_{NL}, further reducing T_L does not increase overhead significantly.

Figure 12 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.

VI. Study of Cache Poisoning and Solutions

Next we extend the analytical model to analyze the cache poisoning propagation in the current DNS. We then compare some proposals – namely DNSSEC and DoX– designed to deal with these attacks.

A. Poison Propagation in Regular DNS

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 (Equation 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 Equation 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 (Equation 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.

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

B. DNSSEC

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 RRset 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.

1) Accuracy and Availability: On guaranteeing the authenticity of a record, DNSSEC offers the same strength as the public-key cryptography scheme it uses. 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 paper. We assume that DNSSEC can prevent cache poison from happening perfectly.

Even though DNSSEC guarantees the authenticity of a record, it does not prevent or reduce 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. As a result, DNSSEC will observe the same probability of serving obsolete records to clients ($\mathcal{A}_O$) or the availability of DNS records.

2) Overhead and Latency: The additional overhead introduced by DNSSEC mainly comes from zone inflation caused by cryptography signatures and related keys (RRISG, DNSKEY and DS RRs). RFC 3226 [19] 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 standard DNS 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 [20], 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, which is about 60% over standard DNS.

The discussed zone inflation is, however, not expected to significantly affect the latency observed by end client. This is because the latency is dominated by RTT. Although transmission time increases proportional to the data size, it constitutes a negligible part to the total latency. On the other hand, processing latency of DNSSEC could be significantly larger than standard DNS.

C. DoX

Domain Name Cross Referencing (DoX) [8] is proposed as an alternative solution to DNS cache poisoning attacks. Instead of relying on cryptography for record authentication, proxy servers in DoX form a peer-to-peer network (a $DoX(N, P)$ network has P proxies and each peering with N peers) to verify the accuracy of DNS records. In addition, DoX peers augment proxies with a *verification cache* to store verified results so that re-verification is only needed when updates happen. Furthermore, once a record update is verified, the peer propagates such update information to other peers so that the entire network is synchronized to the authoritative server.

After a record is retrieved through standard DNS, the DoX peer compares it against its local verification cache and two possible scenarios could result. First, if the record remains unchanged, the response can be served to clients immediately. Second, if there is a modification, proxy A sends a verification request to N peers. As long as at least one peer is not poisoned, DoX can detect the poison and recover from the attack by flushing the cache. If a verifying peer agrees that an update is valid, it will propagate it to other peers so that every peer in the network can "catch up" and update to the current version. Consequently, an obsolete record will be updated as soon as any one of the peers notices the change.

1) Accuracy and Availability: 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 this node in either *obsolete* or *uncached* status. Therefore, in a $DoX(N, P)$ network, the probability of node n being obsolete in a proxy cache is

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

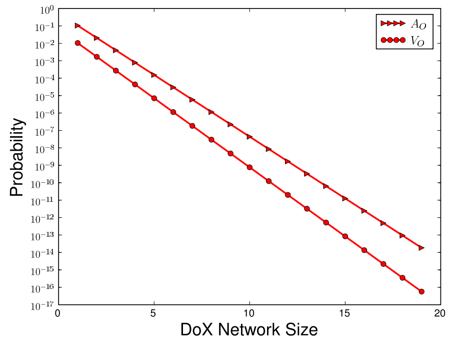

Fig. 13. DoX on Improving Accuracy and Availability

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 13. 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.

2) Overhead: DoX peers incur additional messaging overhead when verifying a record update with peers. 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 P other peers. Assume the modification and query arrival process are Poisson with rate λ_M and λ_Q, respectively. The per-query additional overhead incurred by DoX is:

$$DoX^O = 2 \times P \times \frac{\lambda_M}{\lambda_Q}\,. \qquad (21)$$

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.

3) Latency: 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

$$DoX^L = \frac{H \times RTT}{N} \cdot \frac{\lambda_M}{\lambda_Q}\,. \qquad (22)$$

D. Comparison Between DNSSEC and DoX

Based on the above 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 overhead. For DoX, this depends on the modification process. If $\lambda_M \approx \frac{1}{10}\lambda_Q$, 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 smaller.
- $\mathcal{L}_N$: DNSSEC does not increase the latency directly. DoX, although will not increase the average latency significantly, might increase the worst case latency, which is undesirable.

VII. Conclusions and Future Work

In this paper, we proposed the notion of *Quality of Domain Name Service* (QoDNS) as a comprehensive metric for evaluating the existing DNS service and its proposed variants. The QoDNS metric includes four crucial aspects of DNS, namely, accuracy, availability, latency and overhead. We showed how these parameters can be evaluated analytically for regular DNS and two variants, namely DNSSEC and DoX. The analysis provides insights into how various DNS parameters affect QoDNS components. An important aspect of our work is the analysis of cache poisoning propagation in the hierarchical DNS structure, which is becoming an increasing concern.

The work in this paper can be extended along several vectors. First, the analysis needs to be extended to non-Poisson query arrival process and to the delegation structure of DNS. Both of these are difficult, and approximations will likely have to be explored. Second, the analysis can be extended to other DNS proposals including those based on peer-to-peer technology (e.g., Overlook [21], DDNS [22], CoDNS [23], CoDoNS [24]). Third, a more thorough evaluation of QoDNS via direct measurements would provide further insights into the performance of various DNS variants.

References

[1] A. S. Hughes and J. Touch, "Cross-domain cache cooperation for small clients," in *Proc. Network Storage Symposium*, 1999.

[2] S. M. Bellovin, "Using the domain name system for system break-ins," in *Proc. 5th USENIX Security Symposium*, 1995.

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

[4] Netcraft Ltd., "DNS poisoning scam raises wariness of 'pharming'," http://news.netcraft.com/archives/2005/03/07/dns_poisoning_scam_raises_wariness_of_pharming.html, 2005.

[5] K. Haugsness and the ISC Incident Handlers, "DNS cache poisoning detailed analysis report version 2," http://isc.sans.org/presentations/dnspoisoning.php, 2005.

[6] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS Security Introduction and Requirements," RFC 4033, 2005.

[7] G. Ateniese and S. Mangard, "A new approach to DNS security (DNSSEC)," in *Proc. 8th ACM Conference on Cmputer and Communications Security*, 2001.

[8] L. Yuan, K. Kant, P. Mohapatra, and C.-N. Chuah, "DoX: A peer-to-peer antidote for DNS cache poisoning attacks," in *Proc. IEEE ICC*, 2006.

[9] J. Jung, A. W. Berger, and H. Balakrishnan, "Modeling TTL-based Internet Caches," in *Proc. IEEE INFOCOM*, 2003.

[10] H. Che, Y. Tung, and Z. Wang, "Hierarchical web caching systems: Modeling, design and experimental results," *IEEE JSAC*, 2002.

[11] Y. T. Hou, J. Pan, B. Li, and S. S. Panwar, "On expiration-based hierarchical caching systems," *IEEE JSAC*, 2004.

[12] O. Kolkman, "Measuring the resource requirements of DNSSEC," RIPE NCC / NLnet Labs, Tech. Rep. ripe-352, Sep 2005.

[13] R. Curtmola, A. D. Sorbo, and G. Ateniese, "On the performance and analysis of dns security extensions," in *Proceedings of CANS*, 2005.

[14] R. Naraine, "Massive ddos attack hit dns root servers," http://www.internetnews.com/dev-news/article.php/1486981, Oct 2002.

[15] D. Bertsekas and R. Gallager, *Data Networks*, 2nd ed. Prentice-Hall, 1992.

[16] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, "DNS performance and the effectiveness of caching," *IEEE/ACM Transaction on Networking*, vol. 10, no. 5, 2002.

[17] "dmoz - open directory project," http://www.dmoz.org.

[18] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)," RFC 2136, 1997.

[19] O. Gudmundsson, "DNSSEC and IPv6 A6 aware server/resolver message size requirements," RFC 3226, 2001.

[20] D. Eastlake 3rd, "RSA/SHA-1 SIGs and RSA KEYs in the Domain Name System (DNS)," RFC 3110, 2001.

[21] M. Theimer and M. B. Jones, "Overlook: Scalable name service on an overlay network," in *Proc. 22nd ICDCS*, 2002.

[22] R. Cox, A. Muthitacharoen, and R. T. Morris, "Serving DNS using a peer-to-peer lookup service," in *Proc. IPTPS*, 2002.

[23] K. Park, V. Pai, L. Peterson, and Z. Wang, "CoDNS: Improving DNS performance and reliability via cooperative lookups," in *Proc. 6th Symposium on Operating Systems Design and Implementation*, 2004.

[24] V. Ramasubramanian and E. G. Sirer, "The design and implementation of a next generation name service for the internet," in *Proc. SIGCOMM*, Portland, Oregon, 2004.