

Towards Plugging Privacy Leaks in the Domain Name System

Yanbin Lu and Gene Tsudik
Computer Science Department, University of California, Irvine
{yanbinl, gene.tsudik}@uci.edu

Abstract—Privacy leaks are an unfortunate and an integral part of the current Internet domain name resolution. Each DNS query generated by a user reveals – to one or more DNS servers – the origin and the target of that query. Over time, users’ communication (e.g., browsing) patterns might become exposed to entities with little or no trust. Current DNS privacy leaks stem from fundamental features of DNS and are not easily fixable by simple patches. Moreover, privacy issues have been overlooked by DNS security efforts (such as DNSSEC) and are thus likely to propagate into future versions of DNS.

In order to mitigate privacy issues in DNS, this paper proposes a Privacy-Preserving DNS (PPDNS), that offers privacy during domain name resolution. PPDNS is based on distributed hash tables (DHTs), an alternative naming infrastructure, and computational private information retrieval (cPIR), an advanced cryptographic construct. PPDNS takes advantage of the DHT index structure to provide name resolution query privacy, while leveraging cPIR to reduce communication overhead for bandwidth-sensitive clients. Our analysis shows that PPDNS is a viable approach for obtaining a reasonably high level of privacy for name resolution queries. PPDNS also serves as a demonstration of blending advanced systems techniques with their cryptographic counterparts.

I. INTRODUCTION

The Internet Domain Name System (DNS) [19] is a global hierarchically distributed database. It translates human-readable hostnames into numerical identifiers associated with network-layer IP interface addresses for the purpose of locating (and routing to) individual hosts. Given its importance, much effort has been put into the next generation of secure DNS, referred to as DNSSEC [17]. The main purpose of DNSSEC is to authenticate the origin, and guarantee the integrity of, DNS records. Notably, DNSSEC explicitly rules out data disclosure threats [8].

It is easy to see that today’s DNS provides no privacy. All DNS messages are transmitted in the clear. A malicious local name server that delegates DNS queries for all clients in its administrative domain can easily learn all origin-target information. It also knows each target’s query volume for its administrative domain. Each name server in the DNS zone hierarchy – from the root to the authoritative server for the target queried by a local name server on behalf of its client – learns both the target of the query and query volume for that target. Although caching at the local name server hides the exact volume from other name servers in the hierarchy, authoritative servers can still infer expected query volume based on the distribution of queries’ arrivals.

Leaks of query target identity can be abused for censorship purposes. For example, an ISP can forbid access to a host by simply dropping all queries for that host-name at the local name server controlled by the ISP. This approach, known as DNS filtering, is effectively employed by the Great Firewall of China [2] to filter out websites considered to be a threat by the state. Also it is part of the upcoming German national ISP filtering plan and similar efforts by Australian and New Zealand governments are already partially deployed. Although this is not the only way for an ISP to block a host, it is more effective than other approaches. First, a host-name is easier to memorize than an IP address; hence, most people prefer to use host-names. Second, IP blocking may not work if a host can have a large pool of IP addresses and frequently changes the mapping between its name and an IP address unknown to the ISP. If there were a perfect scheme for privately resolving host-names, the only conclusion the ISP would draw is that there is traffic between the client and an unknown IP address¹. Of course, the ISP can use reverse DNS to retrieve the host-name corresponding to the unknown IP address. However, a host can simply configure its PTR record to some random name and prevent the ISP from learning anything useful. Another approach is for the ISP to install a proxy bound to all sensitive host names, hoping to catch all traffic directed to them. However, we note that DNSSEC can effectively prevent this.

Leaks of target query volume can be used for surveillance purposes. For example, if a DNS server observes a sudden spike in the number of DNS queries for a given host-name, it can alert authorities to check whether suspicious or offensive information on that host is causing its increased popularity. This is a common surveillance skill used in countries such as China to identify possibly sensitive websites.

Leaks of relative target query volume (that might be inferred from target query volume) can be abused for commercial purposes. For example, suppose that the adversary measures relative popularity of any two domain names registered under a certain registrar. The adversary might then “steal” a popular domain name when its owner fails to renew it. It might seem that the chance is quite low for a popular domain owner to forget to register its domain name. However, history shows that even industry giants like Microsoft sometime forget to renew their domain names [4]. Many registrars have been making

¹Assuming IPsec [17] can be used to hide all information above IP layer.

it more difficult for such theft to occur by auto-renewals or by parking previously registered domains for a grace period. However, if the adversary happens to be the registrar itself, it can reserve any popular constituent domains and sell them later at higher prices.

Moreover, information leaks during DNS query pose risk to applications that rely on DNS as an underlying mechanism. For example, ENUM [12] (telephone number mapping) system uses special DNS record types to translate a telephone number into a Uniform Resource Identifier from VoIP providers and to other Internet-based services, such as E-mail. Exposure of a DNS query reveals one’s daily contact list, which is considered to be a serious privacy leak.

This paper focuses on adversaries located at DNS servers. Clearly, a malicious DNS server poses some very serious threats. In this paper, we limit the scope of threats based on the assumption that an adversary does not wish to expose itself too early by behaving too aggressively. Although aforementioned information leaks might be exploited through other means, for adversaries at DNS servers, learning information via DNS is straightforward. This paper focuses on addressing target-related DNS query privacy problems by proposing a privacy-preserving DNS (PPDNS). PPDNS combines a distributed hash table (DHT) with a computational private information retrieval (cPIR) [13] technique. Basically, PPDNS offers a novel way to do range query, i.e. querying a range of domain names instead of only one each time. Owing to consistent identifiers offered by DHT, the same range query can be posed for each identifier inside the range, even in the presence of an active adversary, thereby hiding the actual query target. Moreover, we can also fix the same range query for each identifier inside the range among all clients, thereby hiding the volume and relative volume for a query target. Furthermore, each DHT node can treat each range as a small database and take advantage of cPIR that allows a client to retrieve a record from a server (without revealing which record is being retrieved) with less than $\mathcal{O}(N)$ communication overhead; where N is the number of database records.

Contributions of this work are as follows: (1) we investigate and categorize DNS privacy leaks, (2) we propose a novel range query technique that mitigates them, (3) we design a flexible and parallelizable framework for incorporating cPIR at the server side, and, (4) our security analysis shows that PPDNS offers appreciably better privacy than prior work.

The rest of this paper is organized as follows: Sec. II overviews DNS. Next, Sec. III defines our adversary model. Sec. IV analyzes existing approaches and discusses why they are insecure. Then, Sec. V introduces some preliminaries (before presenting PPDNS). The architecture of PPDNS is presented in Sec. VI and its security analysis follows in Sec. VII. Sec. VIII discusses PPDNS performance and Sec. IX concludes the paper.

II. DNS OVERVIEW

The DNS [19], [20] namespace has a tree structure where each node, except the root, has a non-empty label. Each do-

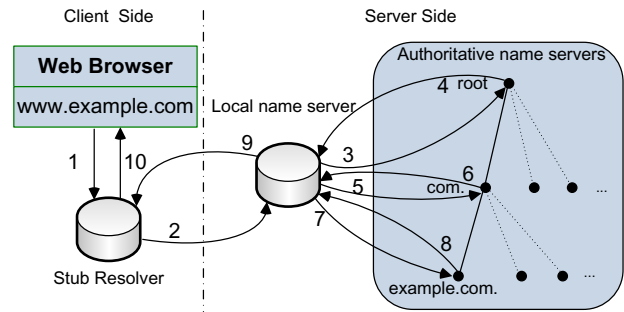


Fig. 1. Current DNS Infrastructure: ten-step resolution of domain name “www.example.com”.

main is a node in the namespace tree, and bottom-up concatenation of nodes’ labels delimited by dots (“.”), creates a fully qualified domain name. For instance, `www.example.com` is a fully qualified DNS name of a node `www` with the parent – `example`, grandparent – `com`, and great-grandparent – DNS root.

Nodes are grouped into zones. The apex of a zone is called the *start of authority* (SOA) and bottom edges are called *delegation points* if other zones exist below them, and *leaf nodes*, otherwise. Zones are served by *authoritative name servers* that are either *primary*, if the zone data comes to them from the outside of DNS, or *secondary*, if their zone data comes to them from primary servers via a zone transfer procedure. Authoritative name servers manage all name information in the domain, keep track of authoritative name servers of sub-domains rooted at their domain; they are administered by namespace operators.

Every node stores *resource records* (RRs) containing information associated with a domain name. An RR can map a host name to an IP address, vice versa, or serve a variety of other purposes. Each RR has a name, class, type, TTL and data. An *RRset* is a list of all records matching a given domain name and resource type. A nameserver returns an RRset as a response to a query.

DNS initiators on host machines are called *stub resolvers*; they have no caches of their own and do not directly interact with the zone hierarchy. They pose basic queries to *local name servers*, also known as recursive resolvers, within their own administrative domain. The local name server is usually designated by the ISP and only provides service to hosts within its administrative domain. It accepts recursive queries from stub resolvers, contacts a chain of authoritative name servers inside the zone hierarchy to locate a DNS RRset, and answers the stub resolver. Because pursuing a chain of delegations to resolve a query can incur significant delays, the local name server caches the result and answers subsequent queries for the same target using the cache, until a TTL (assigned by the authoritative nameserver) expires.

In an example in Fig. 1 (that assumes no caching), resolution of `www.example.com` involves the following: (1) User enters the domain name into the web browser, that consults its local stub resolver, (2) The stub resolver sends the query to its local name server. (3) The local name server consults

with the root server – the authority for the empty label. (4) The root server refers the local name server to “com.” zone authoritative server. (5) The local name server consults this server. (6) The “com.” zone authoritative server delegates the local name server to “example.com” zone’s authoritative server. (7) The local name server consults the latter. (8) “example.com” zone’s authoritative server returns the host address record for “www.example.com”. (9) Finally, the local name server caches the resource record and returns the result to the stub resolver. (10) The stub resolver returns the IP address to the web browser, that launches its connection to the target IP address.

III. THREAT ANALYSIS

In this section we discuss DNS privacy threats and the adversarial model.

A. DNS Privacy Leaks

We consider the following DNS privacy leaks:

- 1) query source-target association
- 2) query source identity
- 3) source query volume, i.e., number of DNS queries issued by a given source.
- 4) relative source query volume, i.e., the difference between query volumes for two given sources.
- 5) query target identity
- 6) target query volume, i.e., number of DNS queries issued for a given target.
- 7) relative target query volume, i.e., the difference between query volumes for two given targets.

Of course, DNS clients may choose to use a general-purpose anonymization service, such as TOR [11] to hide the query source. This obviates problems (1)-(4). However, source anonymity afforded by tools like TOR does nothing to address target-specific leaks (5)-(7) that serve as the focus of this paper.

B. Adversary Model

An adversary can be an insider or an outsider. The latter does not compromise any name servers and only eavesdrops on messages to learn target-related information in DNS queries. We are not concerned with such attacks since standard security tools, such as SSL/TLS and/or IPsec [17], are effective against them. Insiders are malicious local or authoritative domain name servers. Although an insider can be trivially addressed by letting clients running their own recursive resolvers, we still need to analyze the malicious local name server’s advantage, given the fact that few clients are willing to sacrifice performance gained from caching at the local name server. Insiders can be further classified into:

- 1) Passive (Honest-but-Curious) Insider - only listens to queries sent to it but responds honestly.
- 2) Active Insider - may drop, forge or manipulate DNS responses.

We further assume that malicious name servers under the same namespace operator’s control may collude.

C. Adversarial Advantage

We now formally define the adversarial advantage \mathcal{A} . We begin with some definitions: $\eta_t(d)$ denotes the average number of queries for name d within time interval t . We give $\eta_t(d)$ different meanings based on where it is measured. At a local name server, $\eta_t(d)$ represents the number of queries only from its local administrative domains, while, at an authoritative name server, $\eta_t(d)$ represents the number of queries from the entire Internet.

The advantage of a specific \mathcal{A} in terms of each privacy leak in Sec. III-A is defined as:

- 1) $\text{Adv}_1(d)$: the probability of \mathcal{A} learning the target identity for a query against domain name d . (Refer to Sec. III-A.(5))
- 2) $\text{Adv}_2(d, t)$: the probability of \mathcal{A} learning $\eta_t(d)$ for a given domain name d . (Refer to Sec. III-A.(6))
- 3) $\text{Adv}_3(d_1, d_2, t)$: the probability of \mathcal{A} learning $\eta_t(d_1) - \eta_t(d_2)$ for two given domain names d_1 and d_2 . (Refer to Sec. III-A.(7))

In the following, we may omit the parameter and use $\text{Adv}_1, \text{Adv}_2, \text{Adv}_3$ for simplicity.

IV. CURRENT PRIVACY THREATS

It is easy to see that, in current DNS, all three advantages defined in Sec. III-C are 1 for their respective insider adversaries. In this section, we discuss existing approaches that attempt to mitigate the situation.

A. General-Purpose Anonymity Service

TOR [11] uses the onion routing technique to hide traffic source. In other words, Tor employs cryptography in a multi-layered manner to detach the relationship between source and target. Traffic from the source goes through several Tor nodes in cyphertext until reaching the exit. Viewed from the destination, the traffic appears to originate at the Tor exit node. As mentioned earlier, it is an effective approach for hiding the source/target relationships, query sources, source query volume and relative source query volume. However it does nothing to address target-related leaks.

B. Random-Set Query

Zhao, *et al.* [26] and Castillo-Perez, *et al.* [9] propose a random-set query approach. Specifically, each time a client queries a domain name d , it constructs a query set $R(d)$, of size m , comprising of d (*real target*) and $m - 1$ randomly picked names (*confusing targets*). The source then queries each of the m names. This method has the advantage of easy implementation and requiring no changes to the current DNS infrastructure. However, there are some notable drawbacks.

With respect to $\text{Adv}_1(d)$, both [26] and [9] claim that their approaches can achieve $\frac{1}{m}$. However, this is not always true in the face of an active malicious local name server that can force a client in its administrative domain to re-launch a random-set query to the same target. A local name server can achieve this (while remaining undetected) by simply dropping requests from a local client within a short period

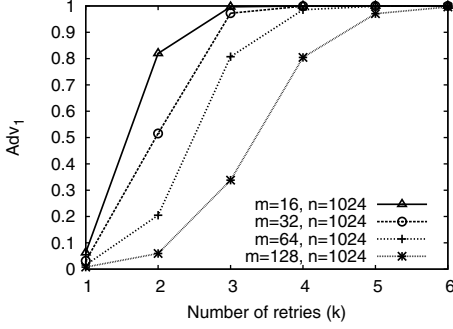


Fig. 2. Random-set query: Adv_1 increases dramatically with the number of requests for the same target.

of time. Let $R_i(d)$ denote the random set of the i -th query to target d . If the local DNS server can get k random set queries for the same target, $R_1(d), \dots, R_k(d)$, then, since $d \in \cap_{i=1}^k R_i(d)$, $\text{Adv}_1 = \frac{1}{|\cap_{i=1}^k R_i(d)|}$ instead of $\frac{1}{m}$. Let S denote the pool of domain names where the client picks both the real target and the confusing targets. Let n denote the size of S . For each name $d' \neq d \in S$, the probability of d' appearing in $R_i(d)$ is $\frac{m-1}{n-1}$. Therefore, the probability of $d' \in \cap_{i=1}^k R_i(d)$ is $(\frac{m-1}{n-1})^k$ assuming that confusing targets are independent for each $R_i(d)$. Thus, the average size of the intersection $\cap_{i=1}^k R_i(d)$ is $\frac{(m-1)^k}{(n-1)^{k-1}} + 1$. Fig. 2 gives an example of the relationship between Adv_1 and k . It shows that, as k increases, Adv_1 increases dramatically, especially, when $\frac{m}{n}$ is small.

As far as $\text{Adv}_2(d, t)$, \mathcal{A} at the local name server still has a full advantage. Assuming that \mathcal{A} (within interval t) witnessed n domain names being queried. \mathcal{A} counts the number of queries from its administrative domain to d_i in interval t as $q_t(d_i), \forall i \in [1, n]$, which includes both queries for d_i as real targets and queries for d_i as confusing targets. Let $\sigma = \sum_{i=1}^n q_t(d_i)$. Then $\sum_{i=1}^n \eta_t(d_i) = \frac{\sigma}{m}$ where $\eta_t(d_i)$ is the number of queries which query d_i as a real target. For each $R(d_j)$, its confusing targets have probability $\frac{m-1}{n-1}$ of hitting d_i for $i \neq j$. Therefore, we have:

$$\begin{aligned} q_t(d_i) &= \eta_t(d_i) + \frac{m-1}{n-1} \cdot \sum_{j \neq i} \eta_t(d_j) \\ &= \eta_t(d_i) + \frac{m-1}{n-1} \cdot \left(\frac{\sigma}{m} - \eta_t(d_i) \right) \end{aligned}$$

Solving this equation yields $\eta_t(d_i) = \frac{q_t(d_i)(n-1)}{(n-m)} - \frac{\sigma(m-1)}{m(n-m)}$.

We also observe that $\text{Adv}_3(d_a, d_b, t) = 1$ for both malicious local name servers and authoritative name servers. It is trivial for \mathcal{A} at a local name server to compute $\eta_t(d_a) - \eta_t(d_b)$, since it already knows $\eta_t(d_a)$ and $\eta_t(d_b)$ ($\text{Adv}_2 = 1$). If \mathcal{A} is an authoritative server, it does not learn $\eta_t(d_i)$, since it does not know σ and n . However, $\eta_t(d_a) - \eta_t(d_b) = \frac{(n-1)(q_t(d_a) - q_t(d_b))}{(n-m)} \approx q_t(d_a) - q_t(d_b)$, given $n \gg m$. Thus, \mathcal{A} can also easily determine $\eta_t(d_a) - \eta_t(d_b)$.

C. Fixed-Set Query

Instead of choosing $m-1$ confusing targets randomly, each client can fix them for each real target d . This can be achieved efficiently through a pseudo-random function with a seed only known by the client. This way, each query for the same target ends up with the same set and $\text{Adv}_1 = \frac{1}{m}$. However, this approach lacks robustness. If the adversary learns the mapping of a target d from one client, all future requests for d from that client are known to the adversary since the mapping is fixed. Moreover, for a malicious authoritative name server, $\text{Adv}_3(d_a, d_b, t)$ is still 1. This is because it is hard to synchronize the mapping function between different clients; therefore, the fixed sets chosen for the same target by different clients are still independent. If at time t , the number of requests to d_a or d_b from the same client is negligible with respect to the total volume, then $q_t(d_a) - q_t(d_b)$ yields a good estimate of $\eta_t(d_a) - \eta_t(d_b)$, as discussed in Sec. IV-B.

D. Combining Tor and Random Set Query

If we combine Random-Set Query with TOR, the adversary at the local name server cannot force the same client to relaunch a random-set query to the same target. This approach effectively reduces Adv_1 to $\frac{1}{m}$ for adversaries at local name servers. However, this privacy gain is not free. To maintain perfect unlinkability, the client needs to create a new TOR circuit, which can take more than 4 seconds [21], for each DNS query. This is very high overhead for a client that only wants to hide its query target. Also, if the number of TOR users in a local domain is small, Adv_2 and Adv_3 for adversaries at local name servers remain quite high. Moreover, even with many TOR users, Adv_3 for adversaries at authoritative servers is not affected, since they are the ultimate “sink” for all queries.

E. DNS Caching

DNS caching only moves the overhead of DNS resolution from authoritative name servers to local name servers. Thus, as pertains to \mathcal{A} at a local name server, caching does not offer any privacy benefits.

With caching, if \mathcal{A} is located at an authoritative name server, it cannot learn the exact query volume for a target. However, it can still compute the average target query volume. To show this, we first need to model the arrival process of DNS request. Poisson processes are widely used to model events if times between consecutive events are independent random variables and the number of events in one interval is independent of these of other intervals. Previous work [22] suggests that session inter-arrivals can be reasonably approximated by an exponential distribution, and that has been supported in studies on DNS caching [15]. We thus consider it reasonable to model DNS requests’ arrival events as a Poisson process.

We assume the arrival process of DNS queries to target d from a local administrative domain l conforms to Poisson distribution with arrival rate $\lambda_l(d)$. Let $tll(d)$ denote the TTL for d . After a client queries its local name server for d , the local name server caches the response (from authoritative

name server) for time $tll(d)$. After seeing a query against d from a local name server from domain l , the adversary knows that, within the next period of length $tll(d)$, it will see no further queries from the same name server. However, because of Poisson distribution, \mathcal{A} knows the average number of queries against d in the following $tll(d)$ time is $\lambda_l(d) \cdot tll(d)$. Given a longer time period $t > tll(d)$, \mathcal{A} can count the number of queries, $k_l(d)$, against d from the local name server. Then, it can estimate the total number of queries occurring in time t as: $\lambda_l(d) \cdot tll(d) \cdot k_l(d) + k_l(d)$. Again, due to Poisson distribution, the following equation holds:

$$\lambda_l(d) \cdot t = \lambda_l(d) \cdot tll(d) \cdot k_l(d) + k_l(d)$$

Solving it yields $\lambda_l(d) = \frac{k_l(d)}{t - tll(d) \cdot k_l(d)}$ and $t \cdot \lambda_l(d) = \frac{tk_l(d)}{t - tll(d) \cdot k_l(d)}$ as the average number of queries from that local administrative domain l . Finally, summing up the average number of queries from all local domains, \mathcal{A} obtains $\eta_t(d) = \sum_l \frac{tk_l(d)}{t - tll(d) \cdot k_l(d)}$.

If \mathcal{A} is an active insider at an authoritative name server, it can also set the TTL to a negligible value, since it has the right to do so. Then, \mathcal{A} retains the same advantage as discussed in previous sections.

F. Information-Theoretic Private Information Retrieval

Information-theoretic PIR is a cryptographic technique that allows a client to retrieve a record from multiple non-colluding servers, each having a copy of the database, without either server learning which record the client wants to retrieve. Zhao, et al. [27] use the two-server version of information-theoretic PIR to achieve DNS query privacy. In reality, servers with duplicate databases are usually the primary and secondary servers for the same zone. These servers can collude with according to the assumptions in Sec. III-B, since they are under control of the same namespace operator. Thus, we do not consider information-theoretic PIR to be suitable for preventing leaks in DNS queries.

V. BUILDING BLOCKS

In this section, we introduce the building blocks used in our system.

A. DHT-based DNS

Distributed hash tables (DHTs) are a class of decentralized distributed systems that provide lookup service similar to hash tables: (key, value) pairs are stored in the DHT, and any participating node can efficiently retrieve the value associated with a given key. There has been plenty of research on flat-structure DHT-based DNS [10], [23]. In fact, some results have already seen trial deployment on the Internet. DHT-based DNS methods use a flat structure to replace the current hierarchical DNS organization. Basically, a DHT-based DNS is a peer-to-peer system where both nodes and objects have randomly assigned identifiers from the same circular space. We use CoDoNS [23] as an example and describe its architecture in detail.

CoDoNS is based on prefix-matching DHT. It designates the node with the identifier closest to the consistent hash of a domain name as the *home node* for that domain name. The home node stores a permanent copy of resource records owned by that domain name and manages their replication. If the home node fails, the next closest node in the identifier space automatically becomes the new home node.

CoDoNS supports DNSSEC by separating authentication of data from service of that data. Every namespace operator has a public-private key pair: the private key is used to digitally sign DNS records managed by that operator, and the corresponding public key is certified by a signature from a domain higher in the hierarchy. This process creates a chain of certificates, terminating at a small number of well-known public keys for globally trusted authorities. The signature and the public key are stored in CoDoNS as resource records of type **sig** and **key**, respectively. Clients verify authenticity of a resource record by fetching these types of records. CoDoNS servers cache certificates along with resource records to help clients in validating records.

CoDoNS decouples namespace management from physical locations of name servers in the network. Namespace operators do not need to participate in data serving. Their responsibility is only to sell certificates of names they account for to nameowners. Nameowners later introduce these names into CoDoNS. CoDoNS servers authenticate nameowners directly through certificates provided for every insertion, deletion and update. CoDoNS is agnostic about the hierarchical structure of namespace, while namespace operators are agnostic about which CoDoNS servers are serving names.

Some common DNS operations are not specified by CoDoNS, but can be easily combined into any DHT-based DNS. For example, to support wildcard querying, a DHT node (when finding no matches for a specific subdomain name, e.g. “random.example.com”) can redirect this request to a node responsible for “*.example.com”, where there is a record for default mapping.

B. Computational Private Information Retrieval (cPIR)

Computation PIR (cPIR) [18] refers to cryptographic techniques for privately retrieving database records. cPIR can be viewed as a game between a user and a database where the latter holds some public data and the former wishes to retrieve the i th record without revealing its interest. The communication complexity must be strictly smaller than the database size. Currently, the best cPIR technique is by Gentry and Ramzan (GR cPIR) [13] with communication complexity $\mathcal{O}(k+d)$, where $k \leq \log n$ is a security parameter that depends on the database size n and d is the bit-length of the retrieved database block. In contrast to information-theoretic PIR, cPIR does not require multiple non-colluding replicated servers and is more suitable for DNS scenario.

VI. PPDNS ARCHITECTURE

We now describe Privacy-Preserving DNS (PPDNS) that mitigates target-related DNS privacy issues. PPDNS is built

upon a DHT-based naming infrastructure. It takes advantage of DHT’s index to implement fixed-range routing. It also offers bandwidth-limited clients the option of using cPIR to reduce communication overhead.

A. Why DHT-based DNS?

Having examined existing approaches, we conclude that current DNS cannot be amended to offer privacy due to its fundamental design features. Therefore, we must consider new approaches that would lend themselves to better privacy. We believe that DHT-based DNS is a viable candidate for a privacy-preserving domain name system.

There are two privacy advantages of DHT-based DNS. First, it uses consistent hashing for object location that enables techniques such as fixed range query and cPIR. Second, the flat and balanced storage structure gives equal power to adversaries compromising different servers. This is in contrast to adversaries in current DNS that attain more power by compromising higher-level DNS servers or servers with more records.

B. Why cPIR?

Modern residential ISPs can offer up to 6Mbps download speeds; thus, most clients do not worry about extra communication overhead due to range queries. However, for GPRS [3] users in developing countries, the maximum currently achievable speed is between 56 and 114kbps, depending on the distance to the base station. If we assume the user issues one range query with $m = 128$ every 4 seconds, the bandwidth amounts to 16kbps for regular DNS (512 bytes per response) and 64kbps for DNSSEC (2048 bytes per response), which consumes a lot of the channel’s capacity. This is unacceptable for most GPRS users. Moreover, many cell-phone users’ data plans are based on traffic volume, especially, when roaming in foreign countries. These factors motivate the use of cPIR, which significantly decreases bandwidth consumption.

C. PPDNS protocol

We begin by introducing our definitions and notation. All clients and PPDNS nodes share a hash function that maps any domain name into a circular space $[0, N - 1]$ where N is the upper-bound of the hash space. We use SHA1 [7] as the hash function, and thus, $N = 2^{160}$. We say that an identifier is *empty* if there are no domain names attached to it. We refer to identifier space density as ρ – ratio of number of non-empty identifiers over total number of identifiers. Each PPDNS node can compute ρ individually, due to the uniformity of the hash function. We define query range for a *target identifier* as a set of contiguous identifiers, including the target. We refer to identifiers in the query range – except the target – as *confusing identifiers*. Each client has a security parameter m representing the number of non-empty identifiers it expects in the query range.

Each client gets an estimate of ρ from its local name server and periodically updates it as a moving average of its previous estimate and newly learnt ρ values. A client who wants to

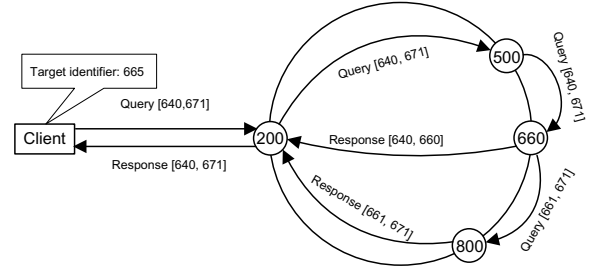


Fig. 3. Range Query in DHT-based DNS

resolve a target domain name first hashes it into an identifier i^* . Then, it determines the number of non-empty identifiers, m , in its query range. The actual range size is computed as 2^s where $s = \lceil \log_2 \frac{m}{\rho} \rceil$. We expect most clients to have the same m and ρ . However, we allow clients to have different m -s and have some estimation error on ρ . The reason for the client to select its range as a power of 2 is to accommodate these differences. Finally, the query range for target identifier i^* is:

$$Q(i^*) = \left[\left\lfloor \frac{i^*}{2^s} \right\rfloor \cdot 2^s, \left(\left\lfloor \frac{i^*}{2^s} \right\rfloor + 1 \right) \cdot 2^s - 1 \right] \quad (1)$$

Theorem 7.1 states that, if the query range is formed in the above manner, then the query range generated for each identifier is fixed and the same query range is posed for each identifier inside this range. Theorem 7.2 states that, even for clients with different m and ρ values, the intersection of their query ranges is lower-bounded by the smallest query range among them.

Next, the client constructs a range query, that only includes the start and end identifiers. It is initially sent to the local PPDNS node that replies immediately if it has a cached copy of the queried range. Otherwise, it routes the query towards its destination node, which accounts for identifiers inside the range, by following the underlying DHT protocol. If any intermediate PPDNS nodes happen to have a cache of the queried range, they respond directly to the local PPDNS node and the query stops. Otherwise, they pass the query to the next intermediate node, until it reaches the destination.

Range splitting may occur and multiple sub-range queries may be generated at intermediate PPDNS nodes. A PPDNS node that has complete records for a sub-range query returns all records (with identifiers inside the subrange) to the local PPDNS server. The local PPDNS server waits until all responses arrive and then delegates them to the client. The local PPDNS server also caches the whole range of responses for the minimum TTL among all responses in the range. Since all clients in the same administrative domain get ρ from the same local name server, they come up with the same range size, as long as m is also the same. Therefore, by caching the whole range served to one client, the local PPDNS server can also respond to later queries from other clients.

As an example, consider Chord [25] circular space shown in Fig. 3, where $\rho = \frac{1}{2}$, $m = 16$, $N = 1024$. Suppose the client

queries a domain name with hash identifier 665. It constructs a range query of [640, 671] (according to Eq. (1)) to its local PPDNS server (node 200). Node 200 forwards the query to node 500, by looking at its finger table. Node 500, in turn, forwards the query to node 660, which is the home node for range [640, 660]. Node 660 first splits the range into [640, 660] and [661, 671]. Then, it replies with records corresponding to range [640, 660] to the local PPDNS server and forwards the sub-range query [661, 671] to node 800, which later replies to the local name server with records corresponding to range [661, 671]. The local name server finally replies to the client.

D. cPIR support

To take advantage of cPIR, the client needs to view a query range as a small database. An output of a secure hash function is typically on the order of hundreds of bits (e.g. 160 for SHA-1). The total number of domain names with respect to the total number of identifiers in the hash space is quite low, i.e., ρ is very small. Given the low density of the hash space, simply using the hash identifier as the index into the database for cPIR would work. However, it would also cause very high computation and bandwidth overhead, since cPIR overhead is proportional to the number of database entries.

One way to tackle this problem is for the client to divide the range (database) into n_{pir} equal-size blocks and assign a consecutive index to each. Then, the client uses cPIR to retrieve the block which contains the desired identifier's record. Because the hash function is near-uniform, the average number of non-empty records in each block is m/n_{pir} . This way, cPIR overhead is proportional to m , instead of $\frac{m}{\rho}$.

Currently, we only employ cPIR between the client and the local PPDNS server. If cPIR was also used between PPDNS servers, the range cache would be no longer possible. We observe through simulation in Sec. VIII-C that the cache captures more than 50% of all range queries. Thus, giving up the cache actually counteracts cPIR benefits. Moreover, due to the high capacity of backbone links, backbone traffic resulting from a range query is very limited (less than 1% in our simulations). Therefore, we see no reason to use cPIR between PPDNS servers.

E. Other Ideas

To facilitate incremental deployment, the client relies on current DNS to resolve queries for records not in PPDNS and explicitly inserts them into the system. To attain privacy against adversaries in both DNS and PPDNS, the client first queries PPDNS with the range formed by Eq. 1. Once it realizes the target domain name's RR set is not within the reply, the client uses the same range query in current DNS. When the client learns the records for target domain name, it asks a random PPDNS node (e.g., chosen by random walk [14]) to insert the target into the system. DNSSEC is employed for preventing poisoning attacks.

VII. SECURITY ANALYSIS

In this section, we analyze the security of PPDNS in terms of adversarial advantage described in Section III-C.

First, we state two theorems dealing with range selection in Eq. 1.

Theorem 7.1: $\forall s \in \mathbf{Z}$ such that 2^s divides N , the query range generated by Eq. (1) for any identifier is fixed and the same query range can be posed for any identifier inside this range.

Proof: Suppose $N = k \cdot 2^s$. Then, for any identifier $i \in [0, N - 1]$, there exists one $j \in [0, k - 1]$ such that $i \in [j \cdot 2^s, (j + 1) \cdot 2^s - 1]$. Moreover, $\forall i \in [j \cdot 2^s, (j + 1) \cdot 2^s - 1]$, the range generated by Eq. (1) for i is $[j \cdot 2^s, (j + 1) \cdot 2^s - 1]$. ■

Theorem 7.2: The lower bound of the size of the intersection for all query ranges generated for a common identifier by different clients with different m and ρ is the smallest $2^{\lceil \log_2 \frac{m}{\rho} \rceil}$.

Proof: We need to show that, if two ranges: $\gamma_1 = [k_1 \cdot 2^{s_1}, (k_1 + 1) \cdot 2^{s_1} - 1]$ and $\gamma_2 = [k_2 \cdot 2^{s_2}, (k_2 + 1) \cdot 2^{s_2} - 1]$ where $s_2 > s_1, k_1, k_2, s_1, s_2 \in \mathbf{Z}^+$, overlap, then $\gamma_1 \subseteq \gamma_2$.

If $k_2 \cdot 2^{s_2} \leq k_1 \cdot 2^{s_1} < (k_2 + 1) \cdot 2^{s_2} - 1$, then $k_1 < (k_2 + 1)2^{s_2 - s_1}$. Since both sides are integers, we have $k_1 \leq (k_2 + 1)2^{s_2 - s_1} - 1$. Therefore, $k_2 \cdot 2^{s_2} < (k_1 + 1) \cdot 2^{s_1} - 1 \leq (k_2 + 1) \cdot 2^{s_2} - 1$. Similarly, we can prove that if $k_2 \cdot 2^{s_2} < (k_1 + 1) \cdot 2^{s_1} - 1 \leq (k_2 + 1) \cdot 2^{s_2} - 1$, then $k_2 \cdot 2^{s_2} \leq k_1 \cdot 2^{s_1} < (k_2 + 1) \cdot 2^{s_2} - 1$.

Thus, if one end of γ_1 falls inside γ_2 , so does the other. Then, the theorem holds. ■

We expect that most clients will have the same security parameter m . As shown in Sec. VIII-A, densities estimated at different PPDNS nodes are very close. Coupled with Theorem 7.2, this means that the intersection size should be reasonably close to $\frac{m}{\rho}$. Even if one client uses a very small m , it only affects Adv_1 as pertains to its own query; this has limited effect on Adv_2 and Adv_3 , as long as the number of such clients are small.

We now use the model of Sec. III to analyze PPDNS security. We assume that all clients use the same m and ρ . Recall that any PPDNS server can be both a local name server and an authoritative name server. We neglect the security analysis for intermediate routing PPDNS nodes, since their advantage is captured by the authoritative name server.

A passive \mathcal{A} at a local PPDNS server receives range queries from, and provides responses to, clients in its administrative domain. By examining each range query response it learns all domain names within inside the range. Therefore, $\text{Adv}_1 = \frac{1}{m}$.

As far as target volume, suppose that V is the total volume for target d 's query range $Q(d)$ from the local administrative domain. Then, the average volume for d should be $p \cdot V$, where p is the probability of d being contained within each range query $Q(d)$. If we assume \mathcal{A} has no prior knowledge of p , any value between 0 and 1 is equally likely, from \mathcal{A} 's perspective. Therefore, the target volume can be any value between 0 and V and $\text{Adv}_2 = \frac{1}{V+1}$.

In terms of relative target volume, given two domain names d_a and d_b , suppose that total query volume for $Q(d_a)$ and $Q(d_b)$ is V_a and V_b , respectively. Then, in \mathcal{A} 's view, $\eta_t(d_a) - \eta_t(d_b)$ can be any value between $-V_b$ and V_a .

Therefore, $\mathbf{Adv}_3 = \frac{1}{V_a + V_b + 1}$. Note that \mathbf{Adv}_2 and \mathbf{Adv}_3 only relies on volume and has nothing to do with query range m . In particular, if $V = 0$ (resp. $V_a = V_b = 0$), $\mathbf{Adv}_2 = 1$ (resp. $\mathbf{Adv}_3 = 1$).

An active \mathcal{A} at a local PPDNS server can drop a response without being detected. However, this does not give it more advantage, since the same range query is always issued for the same target, as shown in Theorem 7.1. \mathcal{A} has no incentive to spread false density information to clients in its administrative domain, since doing so can be detected by checking the density difference with other PPDNS nodes. \mathcal{A} also has no motivation to forge or manipulate responses since DNSSEC can effectively prevent this. Therefore, the advantage for an active malicious local DNS server is the same as that for its passive counterpart.

A passive \mathcal{A} at an authoritative PPDNS server learns whether an identifier in its charge is empty, since $\mathbf{Adv}_1 = \frac{1}{m}$. The advantage of guessing target query volume is again $\mathbf{Adv}_2 = \frac{1}{V+1}$, where V is the total volume to the target identifier’s query range. And, the advantage of guessing relative target query volume is $\mathbf{Adv}_3 = \frac{1}{V_1 + V_2 + 1}$ if V_1 , where V_2 is the volume to two given domain names’ query range.

An active \mathcal{A} at an authoritative PPDNS server cannot change the TTL or manipulate RRset, unlike an authoritative name server in today’s DNS, since all records are signed by different namespace operators and the authoritative PPDNS server is just a host for domain names for which it does not have corresponding signing keys. Thus, its advantage is the same as that of a passive adversary.

VIII. PERFORMANCE EVALUATION

We now evaluate the performance of PPDNS components. We start by considering the accuracy of computing density ρ , which plays an important role in determining query range. Then, we analyze GR cPIR scheme, to see how much bandwidth it saves. Finally, we assess PPDNS performance under realistic backbone environment through simulation.

A. Density Estimation Accuracy

Density ρ estimated at each PPDNS node plays an important role in determining the query range in Eq. 1. Recall that clients in the same administrative domain have the same estimated ρ , since they all get it from the local PPDNS server. However, there is a security threat if ρ values estimated at different PPDNS nodes vary significantly. This is because: (1) large deviation in ρ allows the local adversary to cheat clients by spreading larger ρ (which might not be detected) and (2) by intersecting query ranges, \mathcal{A} may gain higher advantage, since the intersection size is the smallest query range for a specific target identifier.

SHA1 is well known for maintaining uniformity across its range. However, due to the importance of ρ , we need to assure that SHA1 maintains high uniformity in terms of domain names as input. To test this, we retrieved all second-level domain names ending with “.com” (80,044,181 in total, from

VeriSign [6] in July 2009) prefixed each with “www.”, hashed with SHA1 and stored these results at PPDNS nodes.

Fig. 4 shows the density deviation with respect to the number of PPDNS nodes. Errorbars show quantiles 5, 50 and 95 related to the set of density estimation made at each node. As we can see, deviation density estimation increases with the number of nodes. This is expected, since, with more nodes, the sample space at each node becomes smaller.

To measure range deviation, we set m to be 1024 for all nodes and check the deviation of $s = \lceil \log_2 \frac{m}{\rho} \rceil$ which is the logarithm of the final range size to the base of two. We choose a large m here because larger m tends to generate larger error. Fig. 5 shows the range deviation. The errorbars here capture the minimum, median and maximum points. As we can see, until the number of nodes reach 10000, the s estimated at each node is exactly the same. Even when the number of nodes is equal to 10000, the difference of the maximum/minimum s with the median s is only 1. We expect in the real world, the number of domain names will be much bigger and the error will be even smaller.

B. GR cPIR Performance

Due to space limitations, we only consider communication overhead of GR cPIR. For other benchmark results, please refer to the full paper². Fig. 6 shows the compression ratio under different combination of l and n , where l is block length and n is the total number of blocks in the database. As this figure shows, the larger l or n is, the higher the compression ratio.

C. Simulation Results

We now examine PPDNS performance by simulation using NS-3 [5] and show that it can efficiently perform name lookup while incurring reasonable cost, over the backbone network.

We compare average delay, maximum link utilization and cache hit ratio of PPDNS with standard single query and random-set query performed in DHT-based DNS. We also consider range split ratio in PPDNS.

1) *Setup*: We collected a DNS trace at one of uci.edu’s name servers between June and July 2009. We excluded non-legitimate names by querying each distinct name in the trace, which resulted in 1,980,623 distinct legitimate fully-qualified names. For each legitimate name, its TTL and response size were recorded. We then fed each simulator node to NS-3 with one day’s DNS trace, from 12:00pm to 10:00pm. We choose Chord [25] as the underlying DHT for PPDNS. Each simulator node identifier was randomly selected from the hash space.

We simulated PPDNS with backbone topologies, since we expect PPDNS to be deployed at domain level. We used real, inferred, and synthetic topologies. For real ones, we used Abilene and Geant backbone topologies. We also used topologies inferred by the Rocketfuel project [24]. In addition, we generated synthetic topologies using BRITE [1] topology generator. All real and inferred topologies were on the level of Point-of-Presences (PoPs). Table I summarizes the topologies.

²available at <http://arxiv.org/abs/0910.2472>

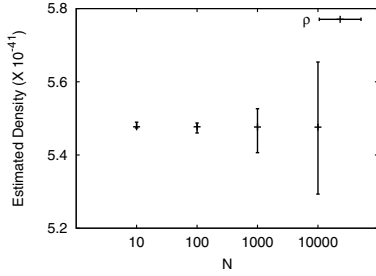


Fig. 4. Density deviation

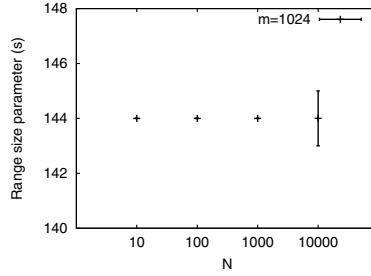


Fig. 5. Range deviation

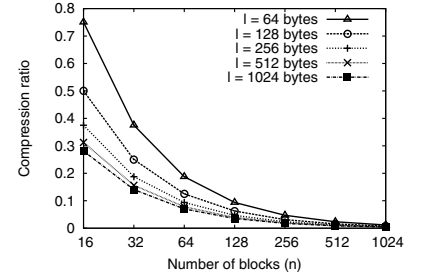


Fig. 6. Compression Ratio

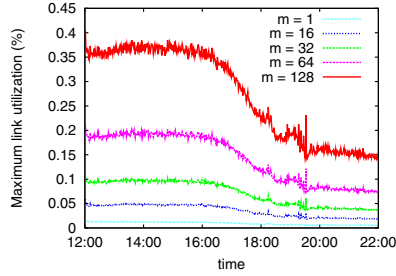


Fig. 7. Maximum link utilization for our range query

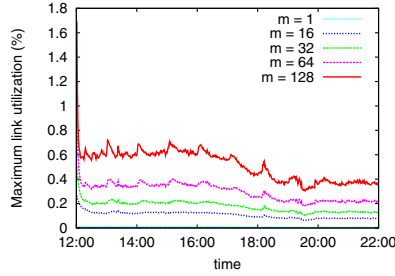


Fig. 8. Maximum link utilization for random-set query

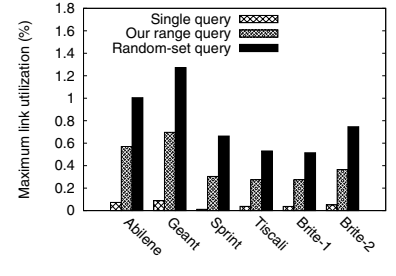


Fig. 9. Comparison of average maximum link utilization among all topologies

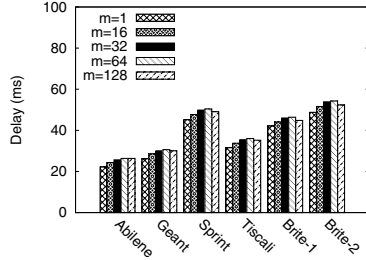


Fig. 10. Average query delay for our range query

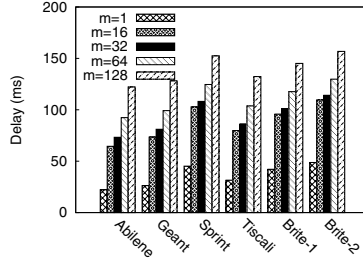


Fig. 11. Average query delay for random-set query

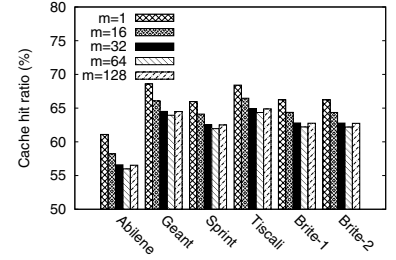


Fig. 12. Cache hit ratio for our range query

Topology	Type	# of Nodes	# of Links
Abilene	R	11	28
Geant	R	23	74
Sprint	I	52	168
Tiscali	I	41	174
Brite-1	A	50	342
Brite-2	A	50	386

TABLE I
SUMMARY OF THE TOPOLOGIES USED IN OUR SIMULATION.
TYPES R, I, AND A MEAN Real, Inferred, AND Random
TOPOLOGIES RESPECTIVELY.

For real topologies, we used actual link capacity in our analysis. For Rocketfuel and random topologies, link capacities were set according to the two-tier model [16], where each link is either OC48 (i.e., 2.48Gbps) or OC192 (i.e., 10Gbps). Specifically, links connecting to the top 20% PoPs with the highest node degrees have higher capacity (10Gbps), while other links have lower capacity (2.48 Gbps). In addition, we set each link initial utilization to be at 60% to simulate medium-loaded network. We set propagation delay of the link with

the smallest weight to 10ms and all the other links' delays – proportional to their weights.

2) *Maximum link utilization*: Fig. 7 and Fig. 8 compares the variation of the maximum link utilization with time in the case of the random topology for our range query and random-set query. The line marked with $m = 1$ means single query. We omit other topologies here because they all show the same trend. The link utilization we show here excludes the link's initial load and is purely contributed by DNS queries. As we can see, the maximum link utilization almost doubles as the range size doubles. Our range-query scheme sees less maximum link utilization than random-set query.

To further compare the link utilization between our range query and random-set query, Fig. 9 shows the maximum link utilization averaged over time for different topologies where m is fixed at 128. We can see that our range query is much better than random query in terms of maximum link utilization. This is due to the fact that the local PPDNS node has a high chance of caching the whole range if there is a previous range query to any target inside this range. While as to random-set query,

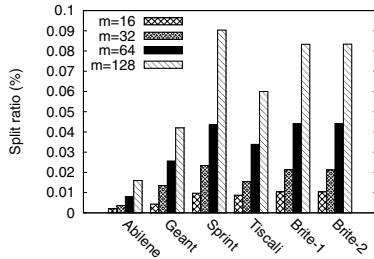


Fig. 13. Range split ratio for our range query

even if the same target has been queried before, there is little chance PPDNS can cache all responses for every name inside the random set of a new query for the same target since the random set chosen each time is random.

3) *Average Delay*: The delay calculated here is caused purely by backbone network, not including the delay between end host and local PPDNS node. Fig. 10 shows the average delay for our range query. As we can see, the average delay in different cases is almost the same as a single query (e.g. $m = 1$). This again is due to our range query's better capability of being cached. We measure the delay for a random set query as the delay between when the first request is sent and when the last response is received for one query set. From Fig. 11, we can see the average delay for random-set query, is much higher than that for the single query and our range query.

4) *Cache hit ratio*: Fig. 12 shows the cache hit ratio for our range query. We can see that it decreases at first, as query range size increases and increases later. This is because the TTL for a range is decided by the smallest TTL of all domain names' TTL within the range. By increasing the range, the smallest TTL has a chance to decrease and therefore the hit ratio decreases as well. However, further increasing range size also helps cache more results which increases the probability of the range being hit. That is why cache hit ratio at $m = 128$ is higher than that at $m = 64$. Also, we can see that the cache hit ratio is over 50% for our range query in all cases. In contrast, the cache hit ratio for random-set query is zero throughout our analysis and we do not show it here.

5) *Range split ratio*: Finally, we measure the split ratio for the range query. Range split can cause extra response delay, since local PPDNS needs to wait for all split-ranges' responses to arrive before answering its client. Fig. 13 shows the percentage of queries which are split in our study. In all cases we examined, the split ratio is below 0.1%. And it is linearly proportional to query range size. We only use 1,980,623 legitimate domain names in our simulation. If we increase the number of available domain names (up the density), the split ratio should decrease accordingly. In practice, we expect the total number of legitimate domain names to be 100 times larger than we recorded. Even with a range size of 10,000, the split ratio should remain below 0.1%.

IX. CONCLUSION

We proposed a new Privacy-Preserving DNS (PPDNS). PPDNS is constructed using a DHT and employs a range query

mechanisms that takes advantage of DHT index structure to offer privacy. PPDNS also incorporates a flexible cPIR framework that can help clients reduce their communication overhead.

Our analysis showed that PPDNS significantly improves privacy for name queries. Performance evaluation demonstrates that PPDNS incurs reasonable communication overhead for backbone links. It also shows that PPDNS retains the efficient part of the traditional DNS—high cache hit ratio, and incurs much lower latency than random-set query.

REFERENCES

- [1] BRITE Topology Generator. <http://www.cs.bu.edu/brite>.
- [2] Empirical analysis of internet filtering in china. <http://cyber.law.harvard.edu/filtering/china/appendix-tech.html>.
- [3] General packet radio service. http://en.wikipedia.org/wiki/General_Packet_Radio_Service.
- [4] Microsoft forgets to renew hotmail.co.uk domain. http://www.theregister.co.uk/2003/11/06/microsoft_forgets_to_renew_hotmail/.
- [5] The Network Simulator 3. <http://www.nsnam.org/>.
- [6] VeriSign. <http://www.verisign.com/>.
- [7] *Secure Hash Standard*. National Institute of Standards and Technology, Washington, 2002. Federal Information Processing Standard 180-2.
- [8] D. Atkins and R. Austein. Threat analysis of the domain name system (dns). *RFC 3833*, August 2004.
- [9] S. Castillo-Perez and J. Garcia-Alfaro. Anonymous resolution of dns queries. In *OTM*. Springer, 2008.
- [10] R. Cox, A. Muthitacharoen, and R. Morris. Serving dns using a peer-to-peer lookup service. *Peer-to-Peer Systems*, 2002.
- [11] R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *SSYM*, Berkeley, CA, USA, 2004.
- [12] P. Faltstrom and M. Mealling. Dynamic delegation discovery system (ddds) application (enum). *RFC 3761*, April 2004.
- [13] C. Gentry and Z. Ramzan. Single-database private information retrieval with constant communication rate. In *Automata, Languages and Programming*. Springer, 2005.
- [14] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. In *INFOCOM*. IEEE, 2004.
- [15] J. Jung, A. W. Berger, and H. Balakrishnan. Modeling TTL-based Internet Caches. In *INFOCOM*. IEEE, 2003.
- [16] S. Kandula, D. Katabi, B. Davie, and A. Charny. Walking the tightrope: responsive yet stable traffic engineering. *SIGCOMM Comput. Commun. Rev.*, 35(4):253–264, 2005.
- [17] S. Kent and K. a. Seo. Security architecture for the internet protocol. *RFC 4033*, March 2005.
- [18] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *FOCS*. IEEE, 1997.
- [19] P. Mockapetris. Domain names - concepts and facilities. *RFC 1034*, December 1987.
- [20] P. Mockapetris. Domain names - implementation and specification. *RFC 1035*, December 1987.
- [21] A. Panchenko, L. Pimenidis, and J. Renner. Performance analysis of anonymous communication channels provided by tor. In *ARES*. IEEE, 2008.
- [22] V. Paxson and S. Floyd. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3:226–244, 1995.
- [23] V. Ramasubramanian and E. G. Sirer. The design and implementation of a next generation name service for the internet. In *SIGCOMM*. ACM, 2004.
- [24] N. T. Spring, R. Mahajan, D. Wetherall, and T. E. Anderson. Measuring ISP topologies with rocketfuel. *IEEE/ACM Trans. Netw.*, 12(1):2–16, 2004.
- [25] I. Stoica, R. Morris, D. Karger, F. M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*. ACM, 2001.
- [26] F. Zhao, Y. Hori, and K. Sakurai. Analysis of privacy disclosure in dns query. In *MUE*. IEEE, 2007.
- [27] F. Zhao, Y. Hori, and K. Sakurai. Two-servers pir based dns query scheme with privacy-preserving. In *IPC*. IEEE, 2007.