Privacy-Preserving DNS: Analysis of Broadcast, Range Queries and Mix-based Protection Methods

Hannes Federrath[1], Karl-Peter Fuchs[1], Dominik Herrmann[1],
Christopher Piosecny[2]

[1]Computer Science Department, University of Hamburg, Germany
[2]Dept. of Management Information Systems, University of Regensburg, Germany

Abstract. We propose a dedicated DNS Anonymity Service which protects users' privacy. The design consists of two building blocks: a broadcast scheme for the distribution of a "top list" of DNS hostnames, and low-latency Mixes for requesting the remaining hostnames unobservably. We show that broadcasting the 10,000 most frequently queried hostnames allows zero-latency lookups for over 80 % of DNS queries at reasonable cost. We demonstrate that the performance of the previously proposed Range Queries approach severely suffers from high lookup latencies in a real-world scenario.

1 Introduction

The Domain Name System (DNS), a globally distributed directory service, is mainly used to translate domain names (hostnames) to IP addresses. The bulk of the translation work is offloaded to DNS resolvers, which query the directory service on behalf of users. Unfortunately, the DNS protocol does not account for privacy. In fact, each DNS resolver has easy access to the IP addresses of its users and the domain names they are interested in. The upcoming DNSSEC protocol does not address in any way the confidentiality of DNS traffic, either. In fact, this was a "deliberate design choice" [3].

During the last years a "third-party ecosystem" for DNS services has evolved. Besides the ISPs there are many more providers offering DNS resolvers. The most popular providers are Google Public DNS and OpenDNS.[1] The DNS providers advertize higher availability, protection from phishing and drive-by-downloads, content filtering and higher performance. These services are also used to circumvent DNS-based censorship. The dissemination of alternative DNS servers has increased significantly during the last years according to figures published by OpenDNS: while they received 3 billion requests per day in September 2007[2], this number has increased to 30 billion by 2010[3].

[1] Homepages at http://code.google.com/speed/public-dns/ and http://opendns.com/
[2] http://www.opendns.com/about/announcements/49/
[3] http://blog.opendns.com/2011/01/24/2010-the-numbers-we-saw

The benefits of public DNS servers come at a price: users must give up some privacy. DNS providers have access to all the DNS queries of their users, which may disclose their interests, relations and habits. Recent research results on user session re-identification [21, 29] also suggest that long-term profiling of users may be feasible solely based on the accessed hosts, enabling a malicious DNS resolver to monitor users over long periods of time and at different locations.

Previous research on privacy-enhancing DNS has not resulted in readily available systems so far. In this paper we aim for a practical and usable solution that allows users to access DNS resolvers privately, i. e., issue DNS queries without disclosing the desired hostnames to the DNS provider. As shown in [20] and [15] usability and especially low latency are crucial factors for the acceptance of Privacy Enhancing Technologies. Our solution addresses this challenge by trading in a little amount of additional traffic for significantly lower latencies.

Contributions. Firstly, we propose a DNS Anonymity Service that can improve privacy and performance at the same time through a combination of broadcast and Mixes. Using real-world DNS traffic dumps we demonstrate the practicability of our solution, which offers zero-latency and totally unobservable lookups for up to 80 % of DNS requests. Secondly, we provide an extensive analysis on the performance of the previously proposed Range Queries approach for real-world web traffic, showing that lookup latencies dominate overall performance.

The rest of this paper is structured as follows. In Section 2 we review related work, and we provide an overview of DNS in Section 3. We outline the architecture of our DNS Anonymity Service in Section 4. In Section 5 we present our broadcast scheme for frequently accessed domain names, before we discuss Mixes and Range Queries in Section 6. In Section 7 we present results from our trace-driven simulations before we conclude the paper in Section 8.

2 Related Work

Previous research efforts regarding privacy-preserving access to DNS servers have mainly focused on the concept of “Range Queries”, which achieves privacy by hiding the queries of a client within a set of dummy queries. Zhao et al. [30] propose a random-set Range Query approach using a single DNS resolver. We will provide a detailed description in Section 6.2. Zhao et al. also propose an improved Range Query scheme [31] inspired by Private Information Retrieval [12]. Their improved scheme reduces the required bandwidth, but requires two non-collaborating DNS resolvers running non-standard DNS software. Although the authors suggest their schemes especially for web surfing applications, they fail to demonstrate their practicability using empirical results. In contrast, our study includes a performance evaluation using actual web traffic of a large user group and a concrete implementation of Range Queries. This allows us to assess the real-world performance of Zhao’s Range Query proposal.

Castillo-Perez et al. [8, 9] study privacy issues of DNS in a different context, namely the ENUM protocol and the Object Naming Service (ONS). They propose a variation of the original Range Query scheme published by Zhao et al. in

[30] using multiple DNS resolvers in parallel. They implemented their proposal in order to evaluate its performance. Their results are of limited relevance for our scenario, though, as their evaluation setup does not resemble the effective DNS topology on the Internet.

Lu and Tsudik propose PPDNS [22], a privacy-preserving DNS system, which is also based on Range Queries, but uses a next-generation DNS infrastructure based on distributed hashtables and peer-to-peer technologies. While PPDNS is a promising approach, we do not expect that it will be widely adopted in the near future due to the need for a completely different DNS infrastructure and its high computational complexity, which requires special hardware.

We conclude that there is no readily available, practical solution for web users to protect their DNS queries, and the performance of the proposed Range Query schemes in real-world settings is unknown.

3 Overview of DNS and the Dataset

The Domain Name System is a distributed database which essentially maps domain names to IP adresses. On each client machine there is a *stub resolver*, a software component of the operating system, which relays DNS queries to the local nameserver. Local nameservers, which are also called *caching resolvers*, fetch the requested information *(Resource Records)* from the *authoritative nameservers* or – whenever possible – from their cache. Each Resource Record has a time-to-live (TTL) value, which indicates how long it can be cached by clients and caching resolvers. The original DNS protocol does not contain any security measures safeguarding integrity and privacy of messages. While integrity protection will become available with the adoption of DNSSEC [3], privacy of message contents and protection of the identity of clients are open problems.

3.1 Characteristics of DNS Traffic

To outline the most relevant characteristics of DNS traffic we summarize the main findings of two well-known studies from 2004 [6] and 2001 [19] and verify and complement them with more recent statistics derived from our 2010 dataset.

An important attribute of DNS traffic is its low traffic volume. Brandhorst et al. showed in their study that DNS packets are responsible for only 0.05 % of overall traffic [6]. In our logs the daily DNS traffic per user added up to about 120 KB with 33 KB for requests and 87 KB for replies. The average sizes of request and reply packets were 36 and 102 bytes respectively. However the low bandwidth requirement is not only a consequence of small request and reply sizes. It is also due to the fact that Resource Records can be cached according to their TTL by resolvers and clients. Jung et al. found that 60 % to 80 % of all requests could be answered using client-side caches [19].

We also found numerous *request bursts* in our logs, i. e., clients query for several hostnames with little or no delay between requests. Further analysis revealed two causes: (1) some websites embed media files from multiple domains,

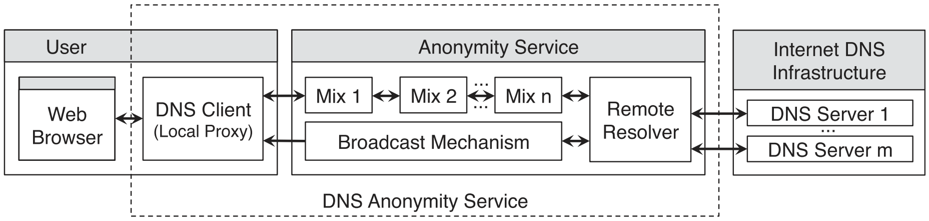

Fig. 1. Overview of the DNS Anonymity Service

and (2) some web browsers pre-resolve all hostnames found in links on a site for performance reasons. Thus, visiting *www.wikipedia.org* may result in up to 150 DNS queries, as each country has its own domain (e. g., *de.wikipedia.org*) linked from the home page.

Another important characteristic of DNS traffic, which is beneficial for the efficiency of caching, is that the popularity of hostnames follows a power-law distribution [19]. A small set of popular hostnames is responsible for the vast majority of all queries, while the "long tail" of remaining hostnames is queried rarely. In the study of Jung et al. 68 % of requests affected the 10 % most popular hosts. We found in our traces that the 10 % most popular hostnames account for 97.7 % of the requests, and the 10,000 most popular hostnames account for 80.2 % of all requests.

3.2 Overview of the Dataset

We cooperated with the computer center of our university to retain a log of all DNS queries from the student housing subnet in pseudonymized form. The log file covers 159 days (from February to July 2010). During that time we observed 9,946,138 distinct hostnames from 4159 users in total. On average there were 2126 users active per day The original log file contains only DNS requests, but lacks information on the replies. Therefore, we issued recursive DNS queries for all hostnames to Google's DNS resolver. We recorded the size of the query and the reply packets as well as the *lookup latency*. We also obtained the TTL values for all hostnames by querying the respective authoritative nameservers. For CNAME Resource Records we followed the trail until an A record was returned and used the minimum of all observed TTL values to obtain the effective TTL. NXDOMAIN replies were handled according to [2].

4 DNS Anonymity Service

The DNS Anonymity Service (cf. Fig. 1) consists of four components, namely a DNS Client, a Mix Cascade, a Broadcast Mechanism and a DNS Resolver ("Remote Resolver"). The DNS Client is installed on the user's computer and acts like a regular DNS resolver towards the users' operating systems. The Remote

Resolver is shared by the clients and looks up DNS entries with the help of the existing DNS infrastructure. Both, DNS Client and Remote Resolver employ caching of replies according to the TTL value to avoid redundant queries. Communication between DNS Client and Remote Resolver is protected by a Mix Cascade (cf. Section 6).

The reason for this design is twofold: On the one hand, we want the whole process of resolving to be transparent for the user. This enables users to keep their usual web browser and additional software. On the other hand, we want to avoid solutions that would require changes to the DNS infrastructure of the Internet.

Attacker model. As the network infrastructure of the Internet does not offer reliable broadcast, we assume that the broadcast messages are distributed consistently to all clients, e. g., by employing Byzantine-fault-tolerant protocols such as [10, 23].

The attacker model for our mix system resembles the attacker models of Tor [14] and JonDonym (formerly AN.ON [5]), two deployed mix-based anonymization systems for low-latency traffic. Specifically, we designed our system to protect against three types of *local attackers*, namely adversaries that control a single (entry or middle) mix or a single communication line (A_1) and adversaries that control an exit mix (A_2) or the DNS resolver (A_3).

We explicitly do not consider a global passive adversary (GPA) with access to all communication lines, as – at least in our web traffic scenario – a GPA can deanonymize users by eavesdropping on HTTP traffic anyway. Initially we set out to also include protection against adversaries controlling both, entry and exit mixes (A_4). The implementation presented in this paper does not protect against such distributed adversaries, though (for reasons explained in Section 6.1). We also do not consider attacks on the integrity of DNS replies by the exit mix or the DNS resolver. Such attacks can be detected by the client once DNSSEC is widely deployed. Finally, we assume that the attacker is computationally bounded and cannot break the cryptographic primitives used.

5 Broadcasting Popular DNS Records

The power-law characteristics of DNS traffic mentioned in Section 3.1 suggest that broadcasting the Resource Records of a small fraction of all hostnames might cover the vast majority of all user traffic. As the replies for the affected queries would be available to users immediately, this solution promises lower latencies than existing non-anonymity providing DNS resolvers. From a security point of view broadcasting is favorable as well since the affected queries can be answered locally with the Resource Records cached in the DNS Client. As a result, resolving of the affected queries becomes unobservable.

Despite the low traffic volume of the DNS protocol, it would be very inefficient and impractical to broadcast all records of the distributed DNS database to all clients due to the large number of registered domains and the long-tailed distribution of query names.[4] Therefore, we suggest a hybrid strategy. Combining broadcast for popular hosts with Mixes for the remaining hosts allows us to find a suitable trade-off between latency and bandwidth usage.

For this purpose, we define a complete ordered list H of all hostnames h_i, sorted by the total number of accesses in descending order. The list is split after θ elements, resulting in two sublists, $\mathrm{TopList}_\theta$ and $\mathrm{LongTail}_\theta$, i. e., $H = \mathrm{TopList}_\theta \cup \mathrm{LongTail}_\theta$ and $H = \mathrm{TopList}_\theta \cap \mathrm{LongTail}_\theta = \emptyset$. $h \in \mathrm{TopList}_\theta$ if $\mathrm{rank}(h) \leq \theta$, otherwise $h \in \mathrm{LongTail}_\theta$.

θ allows us to control the trade-off between latency and bandwidth usage, as it determines the number of hosts to be broadcast. To choose an adequate θ, the interdependent factors "cache hit ratio" (i. e., the percentage of requests affected by broadcast) and "bandwidth requirement" (i. e., the cumulative size of all DNS entries to be broadcast over time) must be considered. While the cache hit ratio is determined by the power-law distribution, bandwidth requirement is limited by the anonymity service's and clients' capacity. To improve the cache hit ratio, further DNS entries must be broadcast, what in turn results in increased bandwidth requirements. A more precise analysis of the interdependencies between both factors is given in Section 7.1.

5.1 Obtaining the Most Popular Hosts

As all the queries for hostnames from the TopList are answered from a local cache in the DNS Client, they are unobservable for the Anonymity Service. Consequently, it is challenging for the Anonymity Service to obtain and maintain the TopList. An obvious approach is to use global web statistics publicly available from companies like Alexa, Com-Score or NetRatings (Strategy 1). They do not represent the usage behavior of varying regionally dependent user groups due to their global focus, though.

A more promising approach is to use the statistics of another DNS resolver located in the same region as the anonymity service (Strategy 2). Opening the DNS Anonymity Service's Remote Resolver for public access (i. e., for users not interested in anonymization) might provide appropriate statistics as well. Alternatively, DNS cache probing [25, 1] can be employed to assemble the TopList.

To fit the TopList as closely as possible to the Anonymity Service's users, rescinding the unobservability property of the broadcast mechanism (i. e., which hosts where queried) is another option (Strategy 3). This should be achieved without revealing which individual user queried which hostnames, of course. With [26], [27] and [7] several well known protocols based on secure multiparty computation techniques exist to solve this challenge, but they suffer from high communicational and computational overhead.

A more pragmatic solution is client-side logging. Users could record the number of requests they were able to save for individual hosts due to broadcast of the

[4] Given an estimate of 205.3 million domain names in Q4/2010 [28] and a size of 50 bytes for each Resource Record, a snapshot of the whole distributed DNS database would amount to more than 9.5 GB.

TopList. If these user statistics were provided to the DNS Anonymity Service in regular intervals (e. g., once a week) via an anonymous channel, the TopList could be kept up to date. While the anonymous channel (e. g., provided by a Mix Cascade) could hide which statistics belong to which users, communication contents (i. e., the statistics themselves) would not be protected, rendering this approach less secure than the protocols mentioned above. As a result, linking user statistics could be possible by means of probabilistical profiling techniques such as [29, 21]. Their impact would be limited though, since linking user statistics with statistics derived from the Mix Cascade used to anonymize hostnames from the long tail is hardly possible, since $\text{TopList} \cap \text{LongTail} = \emptyset$.

5.2 Realization of the Broadcast Mechanism

The broadcast mechanism consists of two parts: first of all, the DNS Anonymity Service must **refresh all entries in the TopList**, since DNS records retrieved from authoritative nameservers expire after a certain time. To this end we use a database containing an entry for each hostname that supplies the corresponding DNS record and a timestamp of its next expiration. A worker thread refreshes the records just before expiration using the Remote Resolver.

Secondly, the **TopList must be distributed to the clients**. Immediately after a client has established a connection to the DNS Anonymity Service, it receives a complete copy of the TopList. As long as the client is connected to the service, it receives a steady stream of incremental updates of the TopList. An update for a record is broadcast only, if the respective record has actually changed since the last update. Since according to [18] DNS records change rarely in comparison to their TTL values, the data volume of incremental updates is supposed to cause only little overhead. Additional reduction in bandwidth can be achieved with compression as pointed out in [18] as well. The broadcast mechanism in our prototype was implemented using TCP/IP unicast, i. e., the DNS Anonymity Service delivers the TopList within a dedicated TCP stream to each connected client. Efficiency could be increased using IP multicast [4].

6 Anonymizing the Long Tail

As outlined in Section 5 we broadcast only a small number of very popular domains in the TopList. Thus, clients need a means to resolve hostnames from the long tail without disclosing them to the resolver. In this paper we study the effectiveness and performance of Mixes and Range Queries for this purpose.

6.1 Mixes

A Mix is a cryptographic technique to enable untraceable communication introduced by David Chaum [11]. The basic idea is to route messages over several independent communication proxies (called Mixes), which hide the communication relationship between senders and receivers. Chaum introduced Mixes for asynchronous applications like electronic voting and e-mail, and he proposed to employ a hybrid cryptosystem using asymmetric and symmetric keys on a per-message basis. Pfitzmann et al. [24] and Goldschlag et al. [17] adapted this concept for real-time protocols that can handle a continuous stream of data with low latency. A client establishes a “channel” which can be used to send multiple consecutive messages through the Mixes. To this end the client establishes shared keys with every Mix using its asymmetric public key. The actual messages are encrypted using fast symmetric ciphers. As all messages transferred within the same channel are linkable, channels have to be switched regularly.

Chaum suggested to repeatedly collect messages until a certain threshold m is reached and only then deliver (flush) all m messages at once in different order to hide the true sender among all present senders. As DNS messages are quite small and most of them are quite similar in size, the application of such an output strategy seems feasible and also promising as it would allow for the construction of a mix system resisting end-to-end attacks. Accordingly, we chose to implement an unbiased, generic mix system instead of building on Tor or JonDonym, both of which are highly optimized for TCP and HTTP traffic and tailored to their respective network topologies. A new development in quite early stage is *ttdnsd*[5], the Tor TCP DNS daemon, which relays DNS queries via TCP to DNS resolvers. Including the official version, without further tuning, in our evaluation would not have allowed for a comparison on fair grounds, though. The then current version 0.7 caused high traffic overhead (queries and replies took up a full 512 byte cell each) and offered poor performance whenever the TCP connection had to be re-established after periods of inactivity.

Security Analysis. The attackers considered in Section 4 may undermine the protection of Mixes in various ways. A_1 may record message sizes of query and reply packets to infer the queried hostnames, exploiting characteristic patterns caused by individual websites. We can thwart this attack by padding packets to a common length. Learning all queried hostnames and being able to link consecutive queries within a channel, A_2 may carry out a user re-identification attack and link consecutive channels. We can decrease the probability of its success by using short-lived channels. While A_3 has access to queries, too, he cannot link queries originating from the same channel. A_2 and A_3 may detect the presence of a certain user based on unique, immediately identifying queries, which is out of the scope of our solution, though. A_4 may correlate timings of incoming and outgoing packets in order to totally deanonymize users. Foiling this attack requires dummy traffic and synchronous batching [24].

Implementation. Our Mix implementation is written in Java using non-blocking I/O operations and the Bouncy Castle crypto provider. In extensive experiments (not reported due to space limitations) we implemented and evaluated several output strategies, e. g., timed and threshold batches with and without

[5] Code repository at https://gitweb.torproject.org/ioerror/ttdnsd.git

dummy traffic, but – even for very small batch sizes – we could not achieve satisfying results in terms of latencies and overhead for any configuration. Thus, we resorted to forward incoming messages immediately, i. e., our mix system does not offer protection against A_4. The implemented channel setup and replay detection resemble JonDonym's mechanisms. Channels are established with an asymmetric cryptosystem (RSA 2048 bit keys) and switched every 60 seconds to limit the information available to A_2.

Requests and reply messages have fixed sizes to address A_1 and are structured as follows: (MAC [16 bytes], length [2], fragmentID [1], payload including padding [s]). For each mix a layer of encryption is applied using a symmetric cipher (AES, 128 bit keys, OFB mode). To find an acceptable trade-off between "message overhead" and the "number of fragmented packets", we analyzed the distribution (weighted by access frequency) of query and reply sizes. We determined $s_{\text{query}} = 57$ bytes and $s_{\text{reply}} = 89$ bytes fitting best. Once DNSSEC is widely deployed, we can change s accordingly.

Our implementation includes several straightforward optimizations, e. g., new channels are established in the background, Mixes use multiple threads to decrypt messages in parallel, and connections between Mixes are multiplexed.

6.2 Range Queries

Various Range Query schemes have been proposed for preserving the privacy of DNS queries (cf. Section 2). Their benefits include a security model that does not rely on the participation of other users and a simple topology, which does not depend on relaying packets over multiple hops. In the following we describe the Range Query scheme evaluated in this paper. It closely resembles the original scheme introduced by Zhao et al. and improved upon by Castillo-Perez et al. In contrast to the PPDNS scheme, which only operates on a DHT-based DNS infrastructure, it is suitable for the DNS infrastructure deployed today.

Each time a client queries a domain name d, it constructs a query set $Q(d)$, of size n, comprised of d and $n-1$ dummy domain names. The client queries the DNS resolver for each of the n names and receives n replies from the server, discarding all but the desired one. Previous work [30, 31, 8] suggests that the client should draw the dummies randomly and independently from a large database of domain names. Assuming that the resolver cannot distinguish the dummies from the desired queries, its chances to correctly guess the desired query are $p = 1/n$. In order to counter intersection attacks mentioned in [9], which can be carried out by an active adversary to uncover the desired hostname, the client uses the same set of dummies for retransmissions of failed queries.

Castillo-Perez et al. and Lu and Tsudik have evaluated the performance of prototypical implementations of their Range Query schemes. Their results are not applicable to our scenario, though, because they assume that all queries can be answered by the DNS resolver immediately, neglecting delays introduced by recursive lookups. In contrast to previous work, we do study the influence of lookup latencies, which may have a significant impact on the overall latencies of Range Queries.

Security Analysis. The security of range queries depends on the resolver being unable to tell apart dummies and actual queries. This assumption is challenged by two traffic analysis attacks that exploit the characteristics of the DNS traffic generated by web browsers, and which have not been studied previously. They allow a malicious resolver to reduce the effective size of the range query whenever consecutive queries are not independent from each other, e. g., for *query bursts*. Firstly, the resolver could mount a *semantic intersection attack* by searching for hostnames known to belong to the same site in consecutive ranges. Instead of randomly and independently sampling dummy hostnames, the ranges must be constructed using plausible sets of hostnames to foil such attacks. Effective protection against intersection attacks is a complex issue, which is outside of the scope of this paper, though, and left open for future work. Instead, we focus on the second traffic analysis attack: the resolver might be able to mount a *timing attack* to identify the dummy replies. If a client issues a range query as an immediate consequence of having processed the desired reply of a previous range query (e. g., when downloading embedded images served from various web servers), the secondary query may reach the resolver before all the replies belonging to the primary query have been received from the upstream DNS servers or sent to the client. Thus, the resolver may deduce that all the pending replies of the previous query are likely dummies. An active adversary could also maliciously send out the replies in a trickle to increase the effectiveness of the attack.

Implementation. For the purpose of evaluation we built a DNS Range Query client in Java. The client bundles up the Range Query into a single package, which is compressed using the zlib library, and sends it to the server component over a TCP socket. The server component resolves all queries in parallel using the Remote Resolver and returns them to the client. We have implemented two alternative strategies that aim to foil the *timing attack* mentioned above. With the *StallDesiredReply* strategy the client waits until all replies of a range query have been received and only then returns the desired answer to the caller. The client can also employ the *DelayConsecutiveQuery* strategy, i. e., return the desired reply to the caller immediately once it is available, but hold back consecutive range queries issued before the still-pending query has been fully processed. We will evaluate the two strategies in Section 7.3.

7 Evaluation

7.1 Broadcasting the TopList

In this section we evaluate the broadcast mechanism regarding cache hit ratio and required bandwidth, taking into consideration the interdependencies between both factors as outlined in Section 5. Our main goal is to quantify the trade-off between latency and bandwidth usage in order to choose an adequate θ. For our simulations we selected a 24h-sample from our dataset. As we focus on DNS queries issued by web browsers in this paper, we selected only type A queries (2,591,240 requests, i. e., 95.7 % of the sample).

Cache Hit Ratio. In a first experiment we examine the suitability of different sources the TopList can be obtained from. We use three different TopLists matching the scenarios described in Section 5. For evaluating Strategy 1 (the use of global web statistics) we derived domain names from the Alexa top one million hostlist. To mitigate the problems in terms of precision with this list, we retrieved the contained web sites using an automated Firefox script. The occuring DNS requests were recorded and combined to a new list with a cut-off after θ elements *(Global TopList)*. To analyze the second strategy we use the most popular hosts obtained from a proxy server used by 50 German schools *(Same Region TopList)*. For the third strategy we have determined the most popular hosts from our DNS dataset to simulate a top list matching user behavior perfectly *(Optimal TopList)*.

In the following we discuss the results obtained through our simulations with $\theta = 10000$. As expected the highest hit rate (83.94 %) was achieved with the *Optimal TopList*. Quite surprisingly the *Same Region TopList* provides comparable results (68.72 %). With only 41.32 % the *Global TopList* performs worst, as expected. Surprisingly, hit rates can be further improved using a client-side cache, which saves 15.72 % of requests. Apparently the caching strategies of user-side stub resolvers fail to exploit the full potential of caching.

In a second experiment we analyze the influence of θ on the cache hit ratio. We used the *Optimal TopList* with varying values for θ between 100 and 100,000. Hit rates from client-side caches were included in the simulation. With a TopList of 100 hosts a hit ratio of 40.02 % was achieved. For $\theta = 1000$ and $\theta = 10000$ the TopList can satisfy 63.94 % and 83.94 % of requests, respectively. Raising θ above 10,000 leads to minor improvements only. At $\theta = 100000$ a hit ratio of 94.54 % was achieved.

Required Bandwidth. We implemented the broadcast mechanism to measure its bandwidth requirements for varying values of θ using our *Optimal TopList*. Therefore, we set up a local instance of the BIND nameserver. We configured BIND to resemble the behavior of typical third-party resolvers by enabling the *minimal-responses* configuration directive. For experimental purposes we disabled BIND's internal cache and configured it to forward all queries immediately to the authoritative nameservers.

We analyze the traffic requirements separately from the perspective of the DNS Anonymity Service, which must continuously refresh its database, and from the perspective of a client of the service.

Traffic for refreshing the TopList. The traffic volume caused by refreshing the TopList database amounts to the traffic caused by DNS requests and replies issued by the DNS Anonymity Service whenever an entry expires. The traffic volume is independent of the number of clients. The daily traffic and the average number of queries per second are shown in Fig. 2 for varying values of θ. The figure indicates that the cost per additional hostname is constant up to $\theta = 2000$ and decreases slightly from there onwards. The daily traffic volume required for

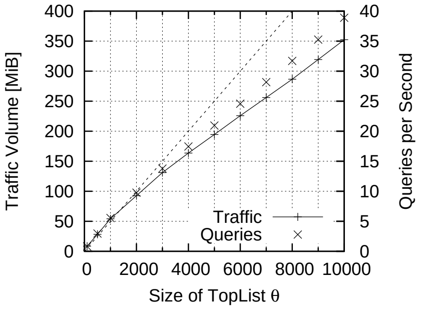

Fig. 2. Refreshing the TopList

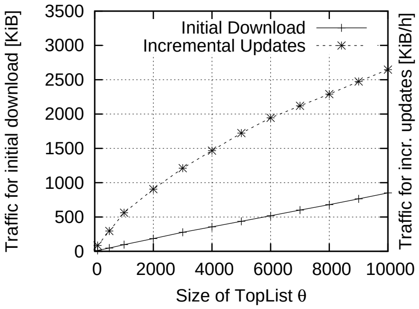

Fig. 3. Distribution to Clients

refreshing a TopList with $\theta = 10000$ is approximately 352.37 MB. On average the DNS Anonymity Service will have to issue 38.89 queries per second to keep all hostnames up to date. The majority of queries (and therefore traffic) pertains to a small fraction of hosts with TTL=60 (1,733 of 10,000 hostnames). In future work we will study optimizations such as enforcing a minimum TTL>60 seconds for all broadcast hostnames and advanced caching methods for round robin DNS replies that are used by many popular web sites for load balancing. Advanced schemes, which make authoritative servers push revocations or update notifications to resolvers, are also promising.

Traffic for distributing the TopList to the clients. The distribution of the TopList database to clients consists of two parts. Whenever a client connects to the DNS Anonymity Service, it receives a full copy of the TopList. After that it receives a steady stream of incremental updates. Using our measurement setup we determined the traffic volume for the initial download of the TopList and for receiving the incremental updates. The results are shown in Fig. 3. The initial download of the TopList amounts to 850 KB for $\theta = 10000$. On average the incremental updates cost 2.58 MB per hour and client (62.02 MB per day and client), which can be streamed with a bandwidth of less than 0.8 KB/s to each client. For 2000 connected clients, broadcasting the TopList consumes a bandwidth of 1.44 MB/s.

Further measurements indicate that the amount of traffic can be reduced considerably by compression. Using the zlib library, we reduced the initial download size on average by almost two thirds (290 KB for $\theta = 10000$), while the volume of the incremental updates was cut by roughly 40 % (to 1.5 MB/h). This finding matches the results in [18].

7.2 Trace-Driven Simulations

We evaluate our implementations of Mixes and Range Queries using trace-driven simulations. This approach allows us to study the effectiveness and performance under different loads induced by real users in a controlled environment. In each experiment we replay actual traffic from the log files in real-time to obtain statistics regarding bandwidth and latency.

In a pretest we found that experimental results stabilize already after a very short time. Thus, we randomly selected 10 chunks from the log file, each containing the traffic of a continuous two-hour period. For ease of exposition we will only provide results for one sample. We repeated the experiments with the remaining samples and validated the results presented in this section. The selected sample contains the DNS queries of 2082 users issued on April 20th, 2010 between 7.00 pm and 9.00 pm. Again, we selected queries of type A only. The resulting log file contains 465,435 requests for 193,133 distinct hostnames.

To allow for a fine-grained analysis of the latencies introduced by our system, we decided to start out neglecting network latencies and congestion, as both are known to dominate overall latencies in practical mix systems. Therefore, the first set of experiments was carried out in a local 1000 Mbit network (cf. Section 7.3). We dedicated a second set of experiments to the analysis of network latencies and congestion to study the expected real-world performance (cf. Section 7.4).

For both sets of experiments, we have implemented a DNS traffic simulator, which instrumented a number of DNS client processes (one per simulated user) according to the recorded traffic from the log file. The traffic simulator and the DNS client processes were running on a single machine. The Remote Resolver artificially delayed queries according to the *lookup latency* τ_l (see below) recorded in our dataset. For the evaluation of the Mix system we set up three Mix nodes, a common configuration also used by JonDonym and Tor, on three dedicated machines with a single DNS resolver on the last machine. Range Queries were evaluated using a DNS resolver with a thread pool of 1,500 workers running on a single machine. All machines were equipped with an Intel Core Duo 2.8 GHz CPU and 4 GB of RAM.

7.3 Performance Comparison of Mixes and Range Queries

In the following we provide the results of the trace-driven simulations for various configurations (first set of experiments). Client-side caches were enabled.

Reported Latencies. We model the *user-perceived latency* as $\tau = \tau_c + \tau_p + \tau_l$, i. e., it consists of the *client network latency* τ_c between the user's machine and the Anonymity Service, the *processing latency* τ_p within the Anonymity Service and the *lookup latency* τ_l for resolving the query at the Remote Resolver. In our experiments we determine the user-perceived latency by measuring the *difference between the time when the client sends the query and the time it receives the corresponding reply*. The reported latency values refer only to the queries that are relayed to the server component, i. e., local cache hits and requests for hostnames contained in the TopList are not included for clarity reasons. Including them would bring down the reported figures to 0 for most experiments. In fact user-perceived latency is 0 seconds for the majority of queries, if the TopList is enabled, of course (cf. Section 7.1).

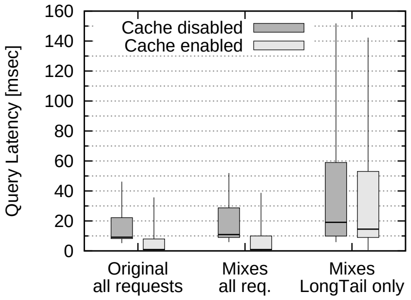

Fig. 4. Latency for Mixes

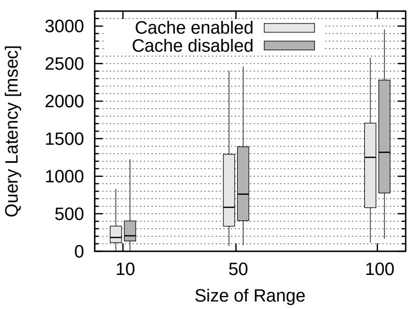

Fig. 5. Latency for Range Queries

Mixes. The results for four configurations using Mixes are shown in Fig. 4. Each boxplot shows the minimum latency, the percentile for 25 % the median and the percentiles for 75 % and 90 %. The baseline configuration ("Original, cache disabled") shows the user-perceived latency, i. e., the distribution of τ_l, without our techniques. The median is 9.2 ms, and the 90 % percentile is 46.2 ms.

The configuration "Mixes all requests" consists of Mixes only (no TopList broadcasting, no caching on the Remote Resolver). The median increases slightly to 10.9 ms, and 90 % of the queries were answered within 52 ms.

Enabling the shared cache on the Remote Resolver ("Cache enabled") brings down latencies significantly: 75 % of the queries are answered within 10 ms. About 60 % of the requests scored cache hits in the Remote Resolver, which matches the findings in [19]. We found the majority of cache hits to be scored by hosts contained in the TopList. Therefore, cache effectiveness is expected to decrease once the TopList is enabled.

The configuration "Mixes LongTail only" shows the latencies observed for the hostnames contained in the LongTail set, i. e., for the queries remaining if the client has access to a TopList ($\theta = 10000$). Latencies are higher in this configuration as the average latency for hostnames from the TopList is considerably smaller (35.25 ms) than for all hostnames on overall (79.74 ms). As expected effectiveness of the cache on the Remote Resolver is limited in this scenario.

The results show that the overhead introduced by the cryptographic operations carried out by Mixes is small. User-perceived latencies will mainly depend on network latencies between clients and Mixes as well as on congestion effects. We study their influence in Section 7.4.

Range Queries. We also measured latency of Range Queries with range sizes $n = 10,\ 50,\ 100$ in our environment. Again, we neglect network delays for now.

The DNS Client creates ranges by randomly drawing dummies without replacement from the set of 193,133 hostnames contained in our dataset. This limitation is artificially introduced by the nature of our trace-driven simulation: we need to know τ_l for every hostname – and also for all possible dummy hostnames. In reality the dummies should be drawn from a much larger set, ideally from the set of all currently active hostnames on the Internet.

The overhead introduced by our Range Query implementation is negligible for isolated queries: we observed that τ_l of the desired query remained virtually unaffected for range sizes between 10 and 1000. In the following, we focus our analysis on the performance impact of the two strategies to counter the timing attack described in Section 6.2. Fig. 5 shows the perceived latency with the optimal TopList ($\theta = 10000$) and the *StallDesiredReply* strategy enabled. We observe that latencies are much higher than τ_l of the desired replies. Even for $n = 10$ 50 % of the requests take longer than 206 ms. For $n = 50$ and $n = 100$ the median is well above 500 ms and 1200 ms, respectively. Not a single Range Query could be fully answered from the cache in our experiments. The performance impact of this strategy is due to holding back the desired reply until the slowest (dummy) reply has been received by the client.

Chances of at least one slow query to be included are very high when they are drawn randomly from the whole population. Given a population of N hostnames containing αN "slow hostnames" with $\tau_l > T$, the probability that a randomly assembled range query of size n does contain at least one slow hostname can be obtained using the hypergeometric distribution:

$$P_T^n = P(X > 0) = 1 - P(X = 0) = 1 - \frac{\binom{\alpha N}{0}\binom{(1-\alpha)N}{n}}{\binom{N}{n}} = 1 - \frac{\binom{(1-\alpha)N}{n}}{\binom{N}{n}}$$

For $T = 200\text{ms}$ we obtained $\alpha = 7\,\%$, i. e., $P_{200\text{ms}}^{10} \approx 0.516$ and $P_{200\text{ms}}^{100} \approx 0.999$, which explains the poor performance of the *StallDesiredReply* strategy. We expected the *DelayConsecutiveQuery* strategy, which achieves low latencies for singular queries at the cost of only delaying closely following queries, to achieve lower latencies on overall. The results indicate otherwise, though: even for small ranges ($n = 10$, cache on Remote Resolver disabled) the median is already 407 ms (90 % percentile: 10.45 s). Further analysis suggests that about 50 % of queries have to wait for their predecessors. In conclusion we find that, while a basic range query scheme may be fast, obscuring timing patterns of web browsers comes at a considerable cost.

7.4 Real World Latencies

In this section we present the results of the second set of experiments that aim at assessing the real world performance of our system by taking into account network latencies. We extend the experimental setup of the previous section by deploying WANem[6] delay boxes between network nodes, which are capable of simulating network latencies and congestion. To parameterize the delay boxes realistically we have cooperated with the JonDonym project, which kindly provided us with common network parameters derived from mixes actually deployed in their cascades across Europe (RTT between mixes: 20 ms; bandwidth of mixes:

[6] http://wanem.sourceforge.net/

Table 1. Effects of network delays and congestion on performance of Mix Cascade. The results for our trace show user-perceived latency ($\tau_c + \tau_p + \tau_l$), while the results for various synthetic loads with the given constant query rate include $\tau_c + \tau_p$ only.

percentile	our trace	synthetic	100	500	1000	2000	3000	4000	5000
50 %	171		139	139	141	245	342	527	1389
90 %	274		140	144	168	341	580	1544	7783

100 Mbit). For client delay boxes we used a latency of 40 ms, which reflects the widespread RTT of about 80 ms for common ADSL connections.

The resulting user-perceived latencies (τ) are shown in Table 1 for our trace-driven simulations with 2082 users (107 $^{\text{queries}}/_{\text{s}}$ on average): in comparison to the measurements without delay boxes latencies for the 50 % (171 ms) and 75 % (274 ms) percentiles increase by 160 ms and 222 ms, respectively. Given that the sum of simulated network latencies is 120 ms, congestion effects are barely visible for this load. Total latencies are still low enough for practical usage.

To study the effects of congestion we induced synthetic traffic, i. e., we sent a constant number of queries per second (qps) to the cascade and measured latencies for different query rates (cf. Table 1). Until 1000 qps, little to no congestion is visible. Between 1000 qps and 3000 qps latencies start to increase noticeably with the load, although some users may find them still acceptable for practical usage. Above 3000 qps, effects of congestion are apparently dominating the performance of the cascade: latencies become unacceptably high.

A straightforward and scalable solution to prevent congestion in practice is to deploy multiple redundant mix cascades. As this would lead to splitted anonymity groups and may have a negative impact on privacy, the adoption of *Free Routes* or intermediate solutions like expander graphs [13] or stratified networks [16] would be worth consideration. As our current implementation is capable of handling a rather high number of participants already, we leave the study of further topologies to future work.

7.5 Traffic Overhead

We measure the traffic overhead by comparing the size of the original query and reply packets in our DNS dataset ("Original") with the traffic volume for Mixes and Range Queries. The resulting overhead is shown in Table 2. Due to message padding and multiple layers of encryption our mix system increases traffic by 99 %. This is considerably less than the overhead for Range Queries.

Interestingly, the overhead for Range Queries is not as high as expected. Without compression, traffic increases by only 583 % instead of the expected 900 % for $n = 10$ (not shown in table). This can be explained by the fact that dummies are drawn uniformly from the set of all hostnames. While the (by access frequency) weighted average size of the DNS replies issued by our users is 102

Table 2. Traffic overhead relative to the original traffic volume for the two hour trace (compression enabled for Range Queries and refreshing the TopList)

	Original		Mix		RQ 10		RQ 50		RQ 100	
	A	B	A	B	A	B	A	B	A	B
No TopList	1	–	1.99	–	4.14	–	14.05	–	23.53	–
$\theta = 10000$	0.15	105.5	0.32	105.7	0.80	106.1	2.81	108.1	5.08	110.4
$\theta = 1000$	0.36	23.34	0.55	23.53	1.81	24.79	6.67	29.66	12.42	35.40
$\theta = 100$	0.63	4.10	0.83	4.30	3.01	6.48	11.10	14.57	20.66	24.13

bytes, the average unweighted reply size is only 72 bytes. Even with compression the overhead is still 314 % for $n = 10$, though.

The table also indicates the traffic savings gained by the TopList for various values of θ. The columns labelled with “A” depict the overhead when traffic needed for refreshing the TopList is neglected, while the “B”-columns incorporate this traffic. It is apparent from the numbers, that the overhead caused by refreshing the TopList significantly outweighs the remainder. We want to stress that the absolute traffic volume is still manageable, though: on average each user transferred 3245 KB (including 3096 KB for refreshing the TopList with $\theta = 10000$) in the Mix configuration within the two hours of simulation.

8 Conclusion

We proposed a DNS Anonymity Service, which combines broadcast with Mixes in order to trade in traffic volume for low latencies, which are critical for DNS. Our proposal exploits the power-law characteristics of DNS traffic to offer improvements on both, privacy and performance, at the same time. We found that broadcasting a small fraction of all hostnames enables unobservability for a large share of user traffic. Moreover, the broadcasted DNS responses are available to users immediately, i. e., faster than with common non-anonymous third-party DNS resolvers. This property of the DNS Anonymity Service may serve as an effective incentive to foster its adoption. Our broadcast component can also be used by conventional DNS providers, who want to offer superior performance.

Moreover, we have evaluated the applicability of Range Queries and Mixes for anonymizing the remaining hostnames with real traffic traces. We found that Range Queries offer poor performance, if dummy hosts are randomly drawn from a large set of hostnames and protection against timing attacks is desired, while Mixes do not introduce considerable delays apart from network latencies. Regarding privacy, a definitive comparison of the two systems is difficult to obtain, due to their different topology and techniques. The security of Mixes and their limitations is well-understood, enabling us to build a practical low-latency system. In light of attacks that exploit semantic interdependencies of queries, the security of Range Queries for web traffic seems much more fragile. It depends on a good source for dummy hostnames as well as a secure range construction scheme, both of which being fertile areas for future work.

Acknowledgements

We thank the anonymous reviewers, Jaideep Vaidya and Benedikt Westermann for their critical feedback. This work has been partially sponsored and supported by the European Regional Development Fund (ERDF).

References

[1] Akcan, H., Suel, T., Brönnimann, H.: Geographic Web Usage Estimation By Monitoring DNS Caches. In: Proceedings of the first international workshop on Location and the web. LOCWEB '08, vol. 300, pp. 85–92. ACM, New York, NY, USA (2008)

[2] Andrews, M.: Negative Caching of DNS Queries (DNS NCACHE). RFC 2308 (1998)

[3] Arends, R., Austein, R., Larson, M., Massey, D., Rose, S.: DNS Security Introduction and Requirements. RFC 4033 (2005)

[4] Armstrong, S., Freier, A., Marzullo, K.: Multicast Transport Protocol. RFC 1301 (1992)

[5] Berthold, O., Federrath, H., Köpsell, S.: Web MIXes: a system for anonymous and unobservable Internet access. In: International workshop on Designing privacy enhancing technologies: design issues in anonymity and unobservability. pp. 115–129. Springer-Verlag, New York, USA (2001)

[6] Brandhorst, C., Pras, A.: DNS: A Statistical Analysis of Name Server Traffic at Local Network-to-Internet Connections. EUNICE 2005: Networks and Applications Towards a Ubiquitously Connected World pp. 255–270 (2006)

[7] Burkhart, M., Dimitropoulos, X.: Fast Privacy–Preserving Top–k Queries using Secret Sharing. In: Proceedings of 19th International Conference on Computer Communications and Networks (ICCCN). pp. 1–7. IEEE (2010)

[8] Castillo-Perez, S., García-Alfaro, J.: Anonymous Resolution of DNS Queries. In: OTM 2008: On the Move to Meaningful Internet Systems, Monterrey, Mexico, Proceedings, Part II. pp. 987–1000. Springer-Verlag, LNCS 5332 (2008)

[9] Castillo-Perez, S., García-Alfaro, J.: Evaluation of Two Privacy–Preserving Protocols for the DNS. In: Proceedings of the Sixth International Conference on Information Technology: New Generations. pp. 411–416. IEEE Computer Society, Washington, DC, USA (2009)

[10] Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: Proceedings of the third symposium on Operating systems design and implementation. pp. 173–186. OSDI '99, USENIX Association, Berkeley, CA, USA (1999)

[11] Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM 24(2), pp. 84–90 (1981)

[12] Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private Information Retrieval. In: Proceedings of the 36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin. pp. 41–50. IEEE Computer Society (1995)

[13] Danezis, G.: Mix-networks with Restricted Routes. In: Proceedings of Privacy Enhancing Technologies workshop (PET 2003). pp. 1–17. Springer-Verlag, LNCS 2760 (2003)

[14] Dingledine, R., Mathewson, N., Syverson, P.F.: Tor: The Second–Generation Onion Router. In: Proceedings of the 13th USENIX Security Symposium. pp. 303–320. USENIX, Berkeley (2004)

[15] Dingledine, R., Serjantov, A., Syverson, P.F.: Blending Different Latency Traffic with Alpha–mixing. In: Proceedings of Privacy Enhancing Technologies (PET 2006). pp. 245–257. Springer-Verlag, LNCS 4258 (2006)

[16] Dingledine, R., Shmatikov, V., Syverson, P.: Synchronous batching: From cascades to free routes. In: Proceedings of Privacy Enhancing Technologies workshop (PET 2004). pp. 186–206. Springer-Verlag, LNCS 2424 (2004)

[17] Goldschlag, D., Reed, M., Syverson, P.: Onion routing. Communications of the ACM 42(2), pp. 39–41 (1999)

[18] Handley, M., Greenhalgh, A.: The case for pushing DNS. In: ACM Workshop on Hot Topics in Networking (Hotnets) (2005)

[19] Jung, J., Sit, E., Balakrishnan, H., Morris, R.: DNS Performance and the Effectiveness of Caching. IEEE/ACM Transactions on Networking (TON) 10(5), pp. 589–603 (2002)

[20] Köpsell, S.: Low Latency Anonymous Communication – How long are users willing to wait? In: Emerging Trends in Information and Communication Security: International Conference, ETRICS 2006, Freiburg, Germany. pp. 221–237. Springer-Verlag, LNCS 3995, Berlin, Heidelberg (2006)

[21] Kumpošt, M., Matyáš, V.: User Profiling and Re–identification: Case of University–Wide Network Analysis. In: TrustBus '09: Proceedings of the 6th International Conference on Trust, Privacy and Security in Digital Business. pp. 1–10. Springer-Verlag, Berlin, Heidelberg (2009)

[22] Lu, Y., Tsudik, G.: Towards Plugging Privacy Leaks in the Domain Name System. In: Proceedings of the Tenth International Conference on Peer–to–Peer Computing (P2P). pp. 1–10. IEEE, IEEE (2010)

[23] Pease, M., Shostak, R., Lamport, L.: Reaching Agreement in the Presence of Faults. J. ACM 27, 228–234 (1980)

[24] Pfitzmann, A., Pfitzmann, B., Waidner, M.: ISDN-MIXes: Untraceable Communication with Very Small Bandwidth Overhead. In: Proc. GI/ITG-Conference "Kommunikation in Verteilten Systemen" (Communication in Distributed Systems). pp. 451–463 (1991)

[25] Rajab, M.A., Monrose, F., Provos, N.: Peeking Through the Cloud: Client Density Estimation via DNS Cache Probing. ACM Trans. Internet Technol. 10, 9:1–9:21 (2010)

[26] Vaidya, J., Clifton, C.: Privacy–Preserving Top–k Queries. In: Proceedings of the 21st International Conference on Data Engineering (ICDE). pp. 545–546. IEEE Computer Society (2005)

[27] Vaidya, J., Clifton, C.: Privacy–Preserving Kth Element Score over Vertically Partitioned Data. IEEE Trans. Knowl. Data Eng. 21(2), pp. 253–258 (2009)

[28] Verisign Inc.: The Domain Name Industry Brief. http://verisigninc.com/assets/domain-name-report-feb-2011.pdf (February 2011)

[29] Yang, Y.C.: Web user behavioral profiling for user identification. Decision Support Systems 49, 261–271 (2010)

[30] Zhao, F., Hori, Y., Sakurai, K.: Analysis of Privacy Disclosure in DNS Query. In: Proceedings of the 2007 International Conference on Multimedia and Ubiquitous Engineering (MUE 2007). pp. 952–957. IEEE Computer Society (2007)

[31] Zhao, F., Hori, Y., Sakurai, K.: Two–Servers PIR Based DNS Query Scheme with Privacy–Preserving. In: Proceedings of the The 2007 International Conference on Intelligent Pervasive Computing. pp. 299–302. IEEE Computer Society (2007)