DNS in the Time of Curiosity: A Tale of Collaborative User Privacy Protection

Philip Sjösvärd, Hongyu Jin, and Panos Papadimitratos

KTH Royal Institute of Technology, Stockholm, Sweden
Networked Systems Security Group
https://www.eecs.kth.se/nss
{sjosv,hongyuj,papadim}@kth.se

Abstract. The Domain Name System (DNS) is central to all Internet user activity, resolving accessed domain names into Internet Protocol (IP) addresses. As a result, curious DNS resolvers can learn everything about Internet users' interests. Public DNS resolvers are rising in popularity, offering low-latency resolution, high reliability, privacy-preserving policies, and support for encrypted DNS queries. However, client-resolver traffic encryption, increasingly deployed to protect users from eavesdroppers, does not protect users against curious resolvers. Similarly, privacy-preserving policies are based solely on written commitments and do not provide technical safeguards. Although DNS query relay schemes can separate duties to limit data accessible by each entity, they cannot prevent colluding entities from sharing user traffic logs. Thus, a key challenge remains: organizations operating public DNS resolvers, accounting for the majority of DNS resolutions, can potentially collect and analyze massive volumes of Internet user activity data. With DNS infrastructure that cannot be fully trusted, can we safeguard user privacy? We answer positively and advocate for a user-driven approach to reduce exposure to DNS services. We will discuss key ideas of the proposal, which aims to achieve a high level of privacy without sacrificing performance: maintaining low latency, network bandwidth, memory/storage overhead, and computational overhead.

Keywords: DNS · Privacy · Security

1 Introduction

The Domain Name System (DNS) translates domain names into information necessary for the functionality of other protocols, most commonly Internet Protocol (IP) addresses [1],[6]. DNS resolvers respond to client DNS queries with a locally cached answer/translation or by querying an authoritative name server, located by recursively querying a distributed hierarchy of name servers [1],[6]. To prevent manipulation of name server responses, resolvers can authenticate them using Domain Name System Security Extensions (DNSSEC) [18]. Meanwhile, user-resolver queries can be encrypted using, for example, DNS over TLS (DoT) [15], DNS over HTTPS (DoH) [14], or DNSCrypt [3]. However, user privacy is

not protected this way; DNS queries still reveal user web browsing activity and thus user interests to resolvers [9].

Granted, resolvers with a strong reputation for privacy and transparency can be chosen, especially if they support encryption of client DNS queries. Users could even try to rotate among multiple such resolvers to reduce their exposure to any one entity. Nevertheless, privacy policies (e.g., do not log specific user information), when in place, must be trusted, as users can hardly verify them alone. More importantly, public resolvers centralize DNS query handling: public resolvers account for nearly 60 % of global name server DNS traffic [2]; one of them, i.e., Google DNS, account for approximately 30 % [2]. Generally speaking, individual organizations handling potentially massive amounts of user data raise concerns about the potential misuse of sensitive user information. Thus, large-scale adoption of public resolvers, despite the services they offer, notably fast and reliable resolutions, is a *double-edged* sword.

DNS queries can be anonymized by introducing a relay server that separates the sender identity from the query content [17],[8]. In such a setup, the resolver only sees the query content, and the relay only sees the sender IP address. However, this approach has limitations, as a colluding relay and resolver can easily re-associate the separated information, undermining user privacy.

To achieve stronger anonymity, anonymous networks such as Tor [10],[19] offer a solution, although they come with the drawback of significantly increased latency [5],[4],[8]. To improve privacy without negatively impacting the user experience, one approach is to use chaff queries: decoy traffic designed to obscure real queries [21],[11]. However, this method is subject to high bandwidth usage and the risk of poor decoy quality [13],[12]. A more promising alternative is to pre-download a list of popular, frequently resolved DNS records – entirely avoiding exposure to external resolvers for the majority of queries, resolving them locally with minimal latency – and resolving any remaining unpopular ones by querying an external resolver using a mix network [11]. To ensure that the answers, e.g., IP addresses, of the included records are up-to-date, users remain connected to the server that provided the list to receive incremental updates [11]. The server continuously re-queries the records as their Time-To-Lives (TTLs) expires, broadcasting any found changes (e.g., changes in IP addresses) to connected users. However, key questions remain: How do we instantiate such a list of popular records according to user interests in a privacy-preserving and secure manner? How do we ensure the relevance of the list as user interests change over time? How do we mitigate the excessive communication overhead from incrementally updating the list due to name servers employing aggressive DNS load-balancing schemes that continuously change record answers? How do we reduce the memory/storage overhead of such a list of popular records?

Our proposal is based on a two-fold approach for resolving DNS queries. First, users attempt to locally resolve queries using a pre-fetched `Popularity List` containing the most popular DNS records. Naturally, the list should reflect the interests of its users, without undermining their privacy. Additionally, one cannot assume that all users are benign when aggregating their interests.

Our proposal is to have clients/users contribute in a privacy-preserving manner to the `Popularity List` creation using a specialized `Voting Mix Network`. The `Popularity List` is maintained and distributed to clients/users by a not trusted, honest-but-curious public server. If a client cannot resolve a query using the `Popularity List`, they can query an external DNS resolver using, for example, a public anonymous network (e.g., Tor).

Using cached data to improve privacy is not a novel concept. Instead, our contribution is the development of methods to anonymously and securely infer user interests to construct a relevant cache (i.e., the `Popularity List`) and to efficiently distribute incremental updates to it. By designing an efficient voting mechanism and scheme to broadcast large volumes of incremental record updates to clients, DNS privacy can be achieved with low latency and network bandwidth overhead.

2 System and Adversary Model

The system consists of three different entity types, ranging from *honest-but-curious* – entities interested in obtaining user information but not in interrupting the DNS protocol – to curious and potentially malicious ones. These entities may collude with each other to achieve their goals. The entities are:

- **A public server:** Central to our proposal, this server maintains and distributes the `Popularity List` and its incremental updates. It is assumed to be honest-but-curious, meaning it correctly provides its advertised services to ensure the functionality of the DNS protocol, but remains interested in tracking user activities (in particular, client-resolved DNS records). The server may occasionally drop or alter packets, as irregular occurrences do not interrupt the general functionality of the DNS.
- **Existing DNS and Internet infrastructure:** These include public resolvers, name servers, Internet Service Provider (ISP) network infrastructure, etc. These entities are honest-but-curious, similar to the central public server.
- **Users/Clients:** They are considered curious and potentially malicious. They may attempt to disrupt the availability of the DNS protocol or infer user information by altering, dropping, or injecting packets.

3 Problem Statement

Our goal is to minimize the exposure of Internet user activity to DNS services, which may collude and share traffic logs to link DNS queries to their user (e.g., IP address) origin. Unlike widely deployed privacy solutions that generally have a linear trade-off between privacy and performance, we aim to achieve privacy comparable to anonymous networks while maintaining inconsequential overhead, i.e., latency, bandwidth, computational, and memory/storage overhead. All while accounting for the ever-changing nature of DNS record data.

4 Our Scheme

The `Popularity List` central to our scheme contains the $N_{popular}$ most popular DNS records, as well as any intermediate CNAME records (records referencing/pointing to another domain) found among those $N_{popular}$. The list is maintained by a central public server (the public server in Section 2) and distributed to all clients/users within the system, which in turn can use it to resolve most DNS queries locally and anonymously. Given the finite size of the `Popularity List`, DNS records not found in it are resolved using other privacy-preserving schemes, such as anonymous networks (e.g., Tor). Within the context of the `Popularity List`, we call such schemes *fallback DNS protocols*. The key feature of the `Popularity List` is not necessarily to provide privacy per se, but instead to improve the user experience of otherwise costly privacy-enhancing schemes (e.g., Tor).

The `Popularity List` memory and storage overhead is generally small due to the long-tail distribution of DNS record popularity, with a small number of records accounting for the majority of queries [11]. We mirror the hierarchical tree structure of domain names to further reduce the `Popularity List` size, compared to storing domain-answer pairs line by line. For example, accumulating all *.com* domains under a single section/header allows *example.com* to be represented by *example*, effectively compressing domain names.

Given that DNS records can change at any point (e.g., changes in IP addresses), the central public server ensures that the `Popularity List` is up to date. That is, any DNS record whose TTL expires is immediately re-queried by the server to receive an up-to-date response from a DNS resolver or name server. If the new response differs from the previous one, the server will broadcast the incremental update to all connected clients. To reduce network overhead, we accumulate several updates and send them in the same message, enforcing only a small minimum TTL (e.g., 60 s) inconsequential to the DNS protocol.

4.1 Efficient Broadcast of Incremental Updates

DNS load-balancing schemes are widely used to distribute users across servers by continuously changing the IP addresses of DNS records. We find that these records often have TTLs as short as 30 s, resulting in high network usage to incrementally update the `Popularity List`, as is the case in [11]. To reduce the amount of information transmitted in each incremental update, our `Popularity List` maintains a pool of answers (typically IP addresses) that at some point have been relevant for at least one record in the list. Instead of transmitting the new answer in its entirety, incremental updates can instead reference/point to the relevant answer in the already cached pool of potential answers by referencing its order of appearance relative to the current one. Similarly, the record whose answer should be updated can be identified by its order of appearance in the `Popularity List`, significantly reducing the incurred network overhead of incremental updates.

4.2 Voting on the Contents of the Popularity List

The set of DNS records included in the `Popularity List` is continuously updated according to anonymous `Votes` submitted by users via a specialized `Voting Mix Network`. These `Votes`, which reflect user queries in the last $t_{refresh}$ seconds, are submitted simultaneously by all clients at set $t_{refresh}$ intervals, allowing our server to periodically refresh the list according to the interests of its users. Any resulting `Popularity List` updates are broadcasted by the server to all connected clients.

Unlike traditional mix networks, the `Voting Mix Network` does not utilize third-party servers for mixing its packets (i.e., `Votes`). Instead, connected clients act as mix nodes, collectively anonymizing the `Votes`. Furthermore, our server also relays mix network packets between each mix node (client) for a more efficient and implementable approach compared to direct node-to-node communication, avoiding firewalls and Network Address Translations (NATs). But it also does so to monitor incoming and outgoing `Votes` at each mix node, to ensure that no entity submits more `Votes` than allowed or injects additional ones. In a traditional mix network, a central node monitoring all connections would undermine its ability to anonymize packets due to possible timing attacks. However, the `Voting Mix Network` is built around all clients casting their `Votes` at the same time and mixing them in distinct *shuffling* rounds, alleviating any such concerns. The overarching process steps are:

1. Each client iteratively encrypts each of their `Votes` using different sets of public keys corresponding to a random set of mix nodes (other clients) in the `Voting Mix Network`. Every `Vote` now has a designated (unique) path through the `Voting Mix Network`.
2. The server receives all initial `Votes` from all clients, sorts them into batches based on their next-hop destination, and redistributes them.
3. Upon receiving its batch of `Votes`, each client decrypts the top layer of encryptions using their private key and randomly shuffles the order of the `Votes`.
4. The server then collects all shuffled batches from all clients and redistributes them, repeating the process.
5. The process continues until all `Votes` are fully decrypted and no longer are forwarded, allowing the `Popularity List` to be updated with the now anonymized `Votes`.

To prevent Sybil attacks, where attackers use a large number of identities/devices to gain increased influence over the `Popularity List` and/or undermine the anonymity of the `Voting Mix Network`, users must register in advance, e.g., with a phone number, and authenticate using acquired credentials upon connecting to the server. It is important to note that we, much like anonymous networks (e.g., Tor), anonymize user activity, not their identity. For example, a user's ISP can observe that they are connected to a Tor entry node.

Furthermore, to ensure that the server does not perform similar attacks by impersonating a large set of clients, users can validate the identity of peer clients

using certificates obtained from an external Public Key Infrastructure (PKI) during registration. Specifically, these certificates are used to verify the public keys of clients participating in the `Voting Mix Network`. To hide user identities, one may use ephemeral pseudonymous client certificates [7],[16].

5 Preliminary Results

To simulate a real-world application of our scheme, we use a dataset of DNS traffic collected from a campus network over ten days [20], which peaks at around 4000 active hosts [20]. The `Popularity List` is instantiated by the queries of the first day of the dataset, and then updated once an hour through a communal voting round using the `Voting Mix Network` ($t_{refresh} = 3600\,\text{s}$). For each hourly voting round, clients cast `Votes` for up to ten unique DNS records, where each `Vote` is based on the DNS queries the client/user resolved during the last $t_{refresh}$. Each user DNS query has a 30 % probability of generating a `Vote`. The voted-on records are ranked based on their weighted number of occurrences, w_m,

$$w_m = 0.1n_m + 0.9w_{m-1}\,, \tag{1}$$

during voting round m, based on the number of received `Votes`, n_m.

Table 1 shows the mean hourly hit ratio of the `Popularity List` during the last nine days of the dataset. The high hit ratio implies that the vast majority of user DNS queries are resolved locally by the `Popularity List`, effectively substantially reducing: the mean latency of the fallback DNS protocol compared to using it as the sole DNS protocol, and the number of queries exposed to the DNS resolver. These results are comparable to the 83.9 % and 94.5 % seen in [11] for a list of size 10 000 and 100 000, respectively. [1]

Table 1: Mean hourly hit ratio of our `Popularity List`.

$N_{popular}$	**Hit ratio**
10 000	91.4 %
25 000	94.4 %
100 000	96.2 %

For our scheme, we use the standard Tor configuration (three nodes) as the fallback protocol for resolving queries not in the `Popularity List`. Figure 1 shows the *exposure rate* of our scheme (using $N_{popular} = 25\,000$ and ten `Voting Mix Network` shuffling rounds per voting round) with Tor as fallback, comparing it with widely deployed DNS anonymization methods. The exposure rate is

[1] The compared *Optimal TopList* from [11] is not implementable in real-world scenarios, as it is constructed from future user queries; users whose queries it cannot reliably sample from in general, as there is no voting (or equivalent) mechanism.

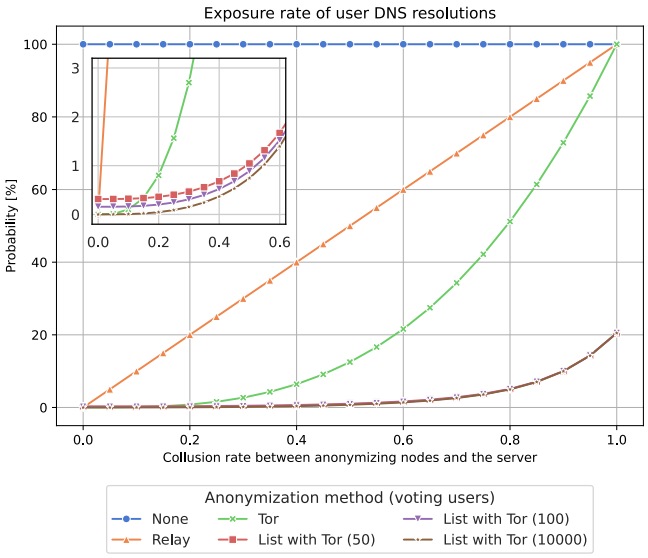

Fig. 1: Exposure rate – the probability that a DNS resolver or our public server correctly guesses the origin of a user-performed DNS query – for various collusion rates between the DNS resolver/public server and other nodes. Tor uses three nodes.

calculated as the probability that a DNS resolver or our public server correctly guesses/infers the origin of a user-performed DNS query, including the potential exposure of converting them to `Votes`; unlike the likely large anonymity set of users for widely deployed schemes, the pool of voters could potentially be small enough to have privacy implications. Compared are various collusion rates between the resolver/public server and anonymizing relays (e.g., single DNS query relay, Tor nodes, other clients, etc.). Overall, our scheme achieves a low exposure rate, even at high collusion rates. The scheme manages to improve the exposure rate, and, importantly, also the mean latency of Tor, as 94.4 % of queries are locally cached by the `Popularity List`. The exception is a slight increase in the exposure rate for low collusion rates due to the potentially limited pool of voting users, allowing `Votes` to be associated with the current pool of users. Interestingly, the performance of 50 and 10 000 voters is comparable, indicating a high level of privacy despite potential cases of poor participation in the voting system.

Figure 2 shows the mean client network bandwidth overhead for incrementally updating the `Popularity List` when, for example, the IP addresses of its DNS records change, for various enforced minimum record TTLs (reflecting the time between transmitted incremental updates). As shown, the overhead of incremental updates is minimal. For comparison, [11] achieves around 1500 kB/h (3.3 kbit/s) for a list of size 10 000: 20 times that of the 67.5 kB/h (0.15 kbit/s)

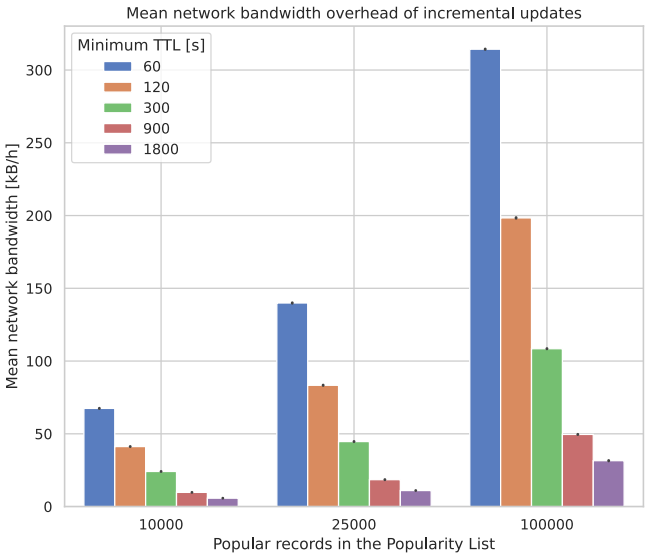

Fig. 2: Mean client network bandwidth overhead for incrementally updating the `Popularity List`, applying compression (Python's *zlib* library) whenever applicable, for various $N_{popular}$ and minimum record TTLs.

our scheme achieves with the 60 s minimum TTL configuration. These improvements are achieved despite, according to preliminary testing, DNS records likely changing more frequently today compared to 2011 due to noticeable reductions in average DNS record TTLs.

6 Conclusion

We consider preparedness to safeguard user privacy based on user collaboration, to avoid relying on the assumption that DNS resolvers and other Internet infrastructure are not curious. The objective is to ensure low-latency, low-bandwidth, and reliable DNS query resolution with strong user privacy protection. We propose a user-driven `Voting Mix Network`, allowing users to anonymously vote on the contents of a securely instantiated, pre-downloaded `Popularity List` of frequently resolved DNS records; a list with an answer cache mechanism to enable efficient incremental updates. In this way, user exposure to public resolvers is minimized while remaining efficient and compatible with the current DNS ecosystem.

7 Acknowledgments

This work was supported in parts by the Strategic Research Foundation (SSF).

References

1. Domain names - concepts and facilities. RFC 1034 (Nov 1987). https://doi.org/10.17487/RFC1034, https://www.rfc-editor.org/info/rfc1034
2. Analysis of 7.5 Trillion DNS Queries Reveals Public Resolvers Dominate the Internet (Mar 2023), https://circleid.com/posts/20230316-analysis-of-7.5-trillion-dns-queries-reveals-public-resolvers-dominate-the-internet
3. Dnscrypt version 2 protocol specification (Feb 2024), https://dnscrypt.info/protocol/
4. I2P Network Performance: Speed, Connections and Resource Management (Apr 2024), https://geti2p.net/en/about/performance
5. Performance - Tor Metrics (Apr 2024), https://metrics.torproject.org/onionperf-latencies.html
6. What is 1.1.1.1? (Apr 2024), https://www.cloudflare.com/learning/dns/what-is-dns/
7. Bißmeyer, N., Petit, J., Bayarou, K.M.: Copra: Conditional pseudonym resolution algorithm in vanets. In: IEEE/IFIP WONS. pp. 9–16 (2013)
8. Denis, F.: Anonymized DNSCrypt specification (Mar 2023), https://github.com/DNSCrypt/dnscrypt-protocol/blob/master/ANONYMIZED-DNSCRYPT.txt
9. Dickinson, S., Overeinder, B., van Rijswijk-Deij, R., Mankin, A.: Recommendations for DNS Privacy Service Operators. RFC 8932 (Oct 2020). https://doi.org/10.17487/RFC8932, https://www.rfc-editor.org/info/rfc8932
10. Dingledine, R., Mathewson, N., Syverson, P.F., et al.: Tor: The second-generation onion router. In: USENIX security symposium. vol. 4, pp. 303–320 (2004)
11. Federrath, H., Fuchs, K.P., Herrmann, D., Piosecny, C.: Privacy-preserving dns: Analysis of broadcast, range queries and mix-based protection methods. In: ESORICS. pp. 665–683. Berlin, Heidelberg (2011)
12. Herrmann, D., Banse, C., Federrath, H.: Behavior-based tracking: Exploiting characteristic patterns in dns traffic. Computers & Security **39**, 17–33 (2013). https://doi.org/https://doi.org/10.1016/j.cose.2013.03.012, https://www.sciencedirect.com/science/article/pii/S0167404813000576, 27th IFIP International Information Security Conference
13. Herrmann, D., Maaß, M., Federrath, H.: Evaluating the security of a dns query obfuscation scheme for private web surfing. In: Cuppens-Boulahia, N., Cuppens, F., Jajodia, S., Abou El Kalam, A., Sans, T. (eds.) ICT Systems Security and Privacy Protection. pp. 205–219. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
14. Hoffman, P.E., McManus, P.: DNS Queries over HTTPS (DoH). RFC 8484 (Oct 2018). https://doi.org/10.17487/RFC8484, https://www.rfc-editor.org/info/rfc8484
15. Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., Hoffman, P.E.: Specification for DNS over Transport Layer Security (TLS). RFC 7858 (May 2016). https://doi.org/10.17487/RFC7858, https://www.rfc-editor.org/info/rfc7858
16. Khodaei, M., Jin, H., Papadimitratos, P.: Secmace: Scalable and robust identity and credential management infrastructure in vehicular communication systems. IEEE Transactions on Intelligent Transportation Systems **19**(5), 1430–1444 (2018)

17. Kinnear, E., McManus, P., Pauly, T., Verma, T., Wood, C.A.: Oblivious DNS over HTTPS. RFC 9230 (Jun 2022). https://doi.org/10.17487/RFC9230, https://www.rfc-editor.org/info/rfc9230
18. Rose, S., Larson, M., Massey, D., Austein, R., Arends, R.: DNS Security Introduction and Requirements. RFC 4033 (Mar 2005). https://doi.org/10.17487/RFC4033, https://www.rfc-editor.org/info/rfc4033
19. Sayrafi, M.: Introducing DNS Resolver for Tor (Jun 2018), https://blog.cloudflare.com/welcome-hidden-resolver/
20. Singh, M., Singh, M., Kaur, S.: Ti-2016 dns dataset (2019). https://doi.org/10.21227/9ync-vv09, https://dx.doi.org/10.21227/9ync-vv09
21. Zhao, F., Hori, Y., Sakurai, K.: Analysis of privacy disclosure in dns query. In: 2007 International Conference on Multimedia and Ubiquitous Engineering (MUE'07). pp. 952–957 (April 2007). https://doi.org/10.1109/MUE.2007.84