LLUAD: Low-Latency User-Anonymized DNS

Philip Sjösvärd
Networked Systems Security group
KTH Royal Institute of Technology
Stockholm, Sweden
sjosv@kth.se

Hongyu Jin
Networked Systems Security group
KTH Royal Institute of Technology
Stockholm, Sweden
hongyuj@kth.se

Panos Papadimitratos
Networked Systems Security group
KTH Royal Institute of Technology
Stockholm, Sweden
papadim@kth.se

Abstract

The Domain Name System (DNS) is involved in practically all web activity, translating easy-to-remember domain names into Internet Protocol (IP) addresses. Due to its central role on the Internet, DNS exposes user web activity in detail. The privacy challenge is honest-but-curious DNS servers/resolvers providing the translation/lookup service. In particular, with the majority of DNS queries handled by public DNS resolvers, the organizations running them can track, collect, and analyze massive user activity data. Existing solutions that encrypt DNS traffic between clients and resolvers are insufficient, as the resolver itself is the privacy threat. While DNS query relays separate duties among multiple entities, to limit the data accessible by each entity, they cannot prevent colluding entities from sharing user traffic logs. To achieve near-zero-trust DNS privacy compatible with the existing DNS infrastructure, we propose `LLUAD`: it locally stores a `Popularity List`, the most popular DNS records, on user devices, formed in a privacy-preserving manner based on user interests. In this way, `LLUAD` can both improve privacy and reduce access times to web content. The `Popularity List` is proactively retrieved from a (curious) public server that continually updates and refreshes the records based on user popularity votes, while efficiently broadcasting record updates/changes to adhere to aggressive load-balancing schemes (i.e., name servers actively load-balancing user connections by changing record IP addresses). User votes are anonymized using a novel, efficient, and highly scalable client-driven `Voting Mix Network` – with packet lengths independent of the number of hops, centrally enforced limit on number of votes cast per user, and robustness against poor client participation – to ensure a geographically relevant and correctly/securely instantiated `Popularity List`. We find that with a 25 000 entries long `Popularity List`, `LLUAD` provides both privacy-preserving and high performance DNS: this is due to the instant local (and anonymous) resolution of around 94 % of queries based on the `Popularity List`, with the few remaining queries using other privacy-preserving, but latency-costly, alternatives, such as querying a public resolver over a public anonymous network, e.g., Tor. Beyond strong DNS privacy and low average lookup latency, `LLUAD` maintains network traffic overhead on par with widely deployed secure DNS protocols, with a memory/storage overhead of less than 2 MB.

CCS Concepts

• **Networks** → **Network privacy and anonymity**; *Naming and addressing*.

Keywords

Anonymous DNS resolutions, Honest-but-curious DNS, Popularity list, Anonymous voting, Client-based mix network

1 Introduction

The Domain Name System (DNS) is a key component of the Internet, translating domain names into Internet Protocol (IP) addresses and other types of related information [5, 12, 28]. DNS resolvers respond to client queries with the sought information/translation, obtaining it from an internal cache, or by recursively querying a distributed hierarchical network of name servers to retrieve the desired DNS record from an authoritative name server [28]. To prevent external entities from manipulating DNS responses or observing user queries, existing secure DNS protocols enable resolvers to authenticate name server responses using Domain Name System Security Extensions (DNSSEC) [30], and encrypt user-to-resolver traffic [14, 23, 24]. However, these efforts fail to protect user privacy: user DNS queries, directly reflecting their web activity – revealing detailed history of visited domains, frequency of Internet activity, used Internet services, etc. – remain exposed to the resolvers [12]. This is particularly concerning given the rising popularity of public DNS resolvers; public resolvers account for nearly 60 % of global DNS name server traffic, with Google DNS accounting for approximately 30 % [4]. These popular public resolvers, often hosted by large corporations [4, 5, 19], providing fast and reliable resolutions, raise concerns about the collection and potential misuse of sensitive user data. Notably, DNS provides particularly reliable data on user-accessed web resources, unlike the potentially unreliable inferences made, for example, by Internet Service Providers (ISPs) based on accessed IP addresses. That is, several domains can be hosted on the same IP address(es): see the widespread usage of Content Delivery Networks (CDNs).

DNS queries can be anonymized by introducing a relay server to separate the queried information from the sender identity [10, 27], revealing the content of the query only to the resolver and the sender IP address only to the relay. However, the separated information can be trivially unified if the two entities collude. In other words, relay server solutions rely on trust, without any further guarantees for privacy. Anonymous networks (e.g., Tor [13]) provide a higher level of user anonymity at the expense of significant latency [10, 25, 34] to DNS query resolutions. To improve privacy without increasing latency, i.e., without sacrificing user experience, proposed solutions include: chaff queries, i.e., generating decoy traffic to obscure user queries/interests [18, 37], or pre-downloading a list of the most popular DNS records to anonymously resolve affected queries locally [18]. While the former suffers from high bandwidth overhead and potentially poor chaff quality [21, 22], the latter is a promising concept, as it eliminates exposure to the

resolver for the majority of queries. However, several key questions remain: How do we instantiate and maintain the relevance of such a list of popular DNS records based on user interests/needs in a privacy-preserving manner? How do we achieve efficient incremental updates to the client-cached records, as they continuously change due to aggressive DNS load-balancing schemes? How do we design a more efficient format of the list to reduce its memory/storage overhead? And fundamentally in this context, how can we enable users to efficiently, securely, and anonymously vote on the content of such a list of popular DNS records?

To effectively address these challenges, we propose *Low-Latency User-Anonymized DNS* (LLUAD). LLUAD provides two methods to resolve DNS queries. First, the user attempts to resolve them locally using a pre-fetched `Popularity List` of frequently resolved DNS records, maintained and incrementally updated by a public untrusted (honest-but-curious) server. In the event the resolution fails, i.e., the query response is unavailable in the `Popularity List`, the user will resort to querying external DNS resolvers using other privacy-focused protocols, such as public mix networks (e.g., Tor). To effectively instantiate LLUAD, we contribute:

(1) A novel client-based `Voting Mix Network` where clients can *efficiently* and *anonymously* vote on geographically relevant DNS records, with packet lengths independent of the number of hops, and a centrally enforced maximum number of votes cast per user.
(2) A scheme to *pre-fetch* and rotate the answers of actively load-balanced records to enable highly efficient incremental updates to the `Popularity List`.
(3) Methods for reducing the memory/storage overhead of the `Popularity List`.

In the rest of this paper, which is based/expands on the early-stage ideas and preliminary results we presented in [33]: Section 2 provides an overview of the DNS along with privacy work relevant to the adversarial model and the problem definition in Section 3. Our scheme, LLUAD, is detailed in Section 4. We evaluate key metrics of LLUAD performance in Section 5, before concluding in Section 6.

2 Background and Related Work

Domain Name System (DNS): DNS records are distributed across a hierarchical structure of independently managed DNS zones [6, 28], allowing organizations to host and manage their own domain names. Each zone includes name servers that respond to queries for its authorized domains. The hierarchical structure of DNS allows any zone to be found starting from a few root name servers. Users can traverse down the hierarchy of domains and subdomains by recursively querying name servers to find an authoritative one that holds the desired record [28]. For example, to resolve the Fully Qualifying Domain Name (FQDN) *www.example.com.*, a user starts at the root zone (denoted by the trailing dot), then moves to the *com* zone, and finally to the *example.com* zone, where the records for *www.example.com* are found.

Recursive DNS Resolvers: This need for multiple queries encourages the use of recursive DNS resolvers, which handle the repeated querying process and cache results to reduce latency and load on name servers. Public DNS resolvers, e.g., hosted by Google (8.8.8.8 and 8.8.4.4) [19] and Cloudflare (1.1.1.1 and 1.0.0.1) [5], have become the standard for DNS resolutions [4] due to their convenience, low latency, and reliability. However, this means that users must trust these public resolvers, as they gain access to their DNS queries [12].

DNS Load-Balancing: DNS is inherently suitable for load-balancing client connections across servers. For instance, a DNS query response can include multiple IP addresses corresponding to different servers hosting the requested resource [7]. By alternating the order of these addresses for each client query, DNS resolvers effectively distribute connection requests among the available servers. Moreover, name servers can actively redirect users based on real-time server loads to *actively* load-balance connections [7].

Encrypted DNS Queries: Since DNS queries are typically unencrypted, a recent push has been to develop protocols enabling encryption between users/clients and DNS resolvers. The first major protocol was DNSCrypt [14], publicly implemented in a DNS resolver in 2011 [35]. In 2016, a new protocol called DNS over TLS (DoT) [24] was proposed. Unlike DNSCrypt, which uses a more proprietary encryption set-up supporting both UDP and TCP, DoT uses TLS. Another protocol, DNS over HTTPS (DoH), emerged in 2018 [23] and has seen wider adoption [2]. DoH adds HTTP on top of TLS, simplifying the protocol's interface and improving privacy by making DNS queries less distinguishable from regular HTTPS traffic.

DNS Relay Servers: However, encryption does not protect user privacy against curious DNS resolvers. Anonymized DNSCrypt [10] and Oblivious DoH (ODoH) [27] were introduced to improve privacy by routing the encrypted queries through a relay that masks the origin (IP address) from the DNS resolver. Due to end-to-end encryption between the client and the resolver, the relay cannot observe the queried resources. However, there is no guarantee that the relay and resolver do not collude to undermine privacy. To defend their privacy against curious colluding relay servers, users can send queries over the Tor network [11, 31] using DoH over Tor (DoHoT), to anonymously query resolvers. However, Tor can significantly affect the user experience [10, 34], introducing around 400 ms to over 1 s of latency per DNS query (see Section 5.4); especially considering that websites typically trigger multiple DNS queries [12, 22].

Chaff Queries: To avoid the latency incurred by mix networks (e.g., Tor), one could create an artificial anonymity set by introducing decoy DNS queries (chaff traffic) to obscure user queries [18, 37]. However, generating plausible and realistic decoys is difficult [21, 22]: to ensure not only consistency between the probability of selecting a specific chaff domain/record and its expected popularity, but also that external entities cannot trivially correlate subsequent queries to reveal the genuine ones, given that websites typically trigger a pattern of multiple DNS queries [22]. Even if we presume realistic chaff query distributions and that obscuring query patterns are achievable, a vast pool of chaff domains to draw from would be needed. For instance, the database in [36] tracks billions of global domains and subdomains, resulting in significant overhead for storing a sufficient number of domains. Moreover, the communication overhead scales linearly with the anonymity set size; emulating N artificial users results in N times greater computational and network overhead for Internet services and DNS infrastructure.

Private Information Retrieval (PIR): A more efficient solution is for users to directly, yet anonymously, resolve DNS queries using Private Information Retrieval (PIR). In PIR, users query either multiple non-colluding servers for parts of the answer, or a single server with a database supporting such anonymous queries by fulfilling specific computational requirements [16, 20]. Thus, users can anonymously query the contents of the database without resorting to the trivial solution of downloading it in its entirety, an otherwise ideal solution if not for the overhead. However, a database of a large number of DNS records has a small size (in bytes) [18], thus the ideal solution to download the entire database is not only feasible, but arguably a promising alternative.

Pushing the Most Popular DNS Records: Distributing a list of the top 10 000 most frequently resolved records [18] offers low-latency and privacy-preserving resolution for the majority of DNS queries, while the remaining queries to DNS servers are anonymized with a mix network [18]. The list provider is responsible for continually refreshing/re-querying the records whenever their Time-To-Lives (TTLs) expire, broadcasting any found updates (e.g., changes to IP addresses) to users [18].

Although [18] discusses several strategies for generating and maintaining such a list, only a brief qualitative analysis is provided. Notably, [18] do not provide a rigorous solution for initiating and updating the list based on the interests of its users. Ensuring the list is relevant for the specific user base is important, as indicated by the difference in hit ratio between the optimal, regional, and global list in [18] (achieving 83.9 %, 68.7 %, and 41.3 %, respectively). However, to ensure relevance, user interests/queries need to be regularly (if not continuously) securely and anonymously sampled. Furthermore, it is of great interest to reduce the overhead of such a scheme. In particular, reducing the significant network bandwidth overhead of incremental updates seen in [18], caused by name servers that constantly change the IP addresses of DNS records to actively balance server loads.

Sphinx Mix Network: Mix networks anonymize packet origins by mixing packets from multiple senders within a network of geographically distributed relay nodes [9, 29]. This disassociates the incoming packets from the outgoing ones, making it difficult to trace their origins. To send packets, users encrypt them several times using different keys that only a predetermined path of specific nodes in the network can decrypt, ensuring that they are sufficiently disassociated from their origin.

A notable example relevant to this work is the Sphinx mix network, where each receiving node decrypts the message using a symmetric key derived using Elliptic-curve Diffie-Hellman between its private key and a (singular) public key element included in each message [9]. Instead of, for example, the Tor circuit-switched approach requiring recursive pre-established tunnels (tunnel to the third node is established through the tunnel to the second node, through the one to the first node, etc.) [13], the Sphinx mix network operates on a packet-switched model, with each message containing sufficient information to perform a key exchange. The Sphinx mix network has the ability to blind the public key element in each message to ensure that it does not remain static throughout the mix network path, in which case it would trivially expose the traveled path. LLUAD leverages the blinding technique in its `Voting Mix Network`, as detailed in Section 4.2.3.

3 Problem Statement and System Model

We aim to minimize Internet user activity exposed to DNS system entities, which may collude and exchange information to link DNS queries to their senders' IP addresses. Motivated by the shortcomings of existing DNS privacy solutions, we aim for low communication overhead, preserving the lightweight nature of the DNS protocol, and especially low latency, an important aspect of the Internet/Web user experience. Similarly, the computational and memory/storage overhead of the scheme should be practically imperceptible by the user of a web browser, considering that this is the primary use case of such a privacy scheme. In summary, a significant set of often conflicting requirements that LLUAD must fulfill.

System and Adversary Model: There are three different types of system entities: users/clients (primarily assumed to be devices from which users browse the web), a public LLUAD server central to our proposal, and existing servers within the public DNS infrastructure (public resolvers, name servers, query relays, etc.). Figure 1 illustrates the roles of these entities as a user obtains the `Popularity List`, explained in detail in Section 4.

The public LLUAD server and the DNS servers/infrastructure, as well as ISPs, are considered honest-but-curious. Honest-but-curious entities provide their advertised/expected services, ensuring that the DNS protocol functions effectively and satisfies users. However, these entites are also curious about user activity in terms of DNS-resolved domains and records. They can drop or alter packets without denying DNS (e.g., occasionally dropped queries do not generally hinder DNS functionality). In contrast, users/clients are viewed not only as curious, but also as potentially malicious. Thus, clients might have malicious intentions to disrupt the protocol availability, by injecting, altering, or dropping packets. System entities may collude with each other to achieve their goals, assuming that it does not interfere with their overarching goals (e.g., an honest-but-curious server will not collude with a malicious client that is actively disrupting its service).

4 Our Scheme

Central to our scheme is a `Popularity List` locally stored on user devices, containing the $N_{popular}$ most popular DNS records in a specific area, geographic, or otherwise representative of its users. The size of $N_{popular}$ can vary per client based on available computational resources. This list acts as a local DNS resolver, either on the user device or within their personal network. Local DNS caches, such as those of web browsers, function as usual. That is, they are always prioritized over querying a resolver (including the `Popularity List`). DNS queries not resolvable by the `Popularity List` (the requested record is not in it) fall back to LLUAD-external privacy-preserving methods that maintain a high level of privacy, e.g., Tor, but at the cost of higher latency, offset by the primary reliance on the local `Popularity List`. The format of the `Popularity List` is detailed in Section 4.1.

The LLUAD server, as illustrated in Figure 1, manages and distributes the `Popularity List`. Clients connect to the server and download the latest version of the list (step (1) in Figure 1). The LLUAD server continuously re-queries the public DNS infrastructure (name servers or public DNS resolvers) as the TTLs of records in the

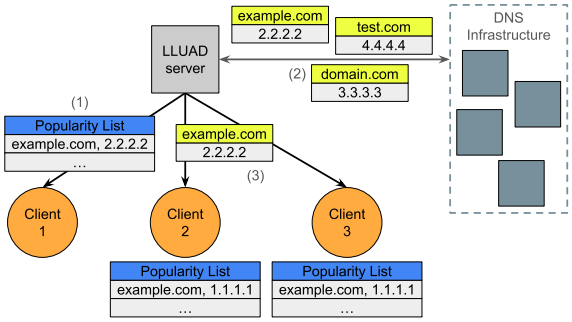

Figure 1: Example of a client downloading the `Popularity List` (1), the `LLUAD` server re-querying the list's records as their TTLs continuously expire (2), and the `LLUAD` server pushing a discovered record update to connected clients (3).

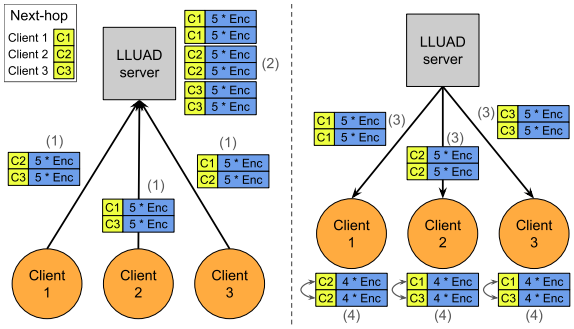

Figure 2: Illustration of the `LLUAD` server relaying batches of encrypted `Votes` between clients that decrypt the top encryption layer and shuffle their order.

list expire (step (2)), broadcasting any found incremental updates to connected clients (step (3)). The `LLUAD` server and its accompanying `Popularity List` emulate the role of a public DNS resolver. For example, name servers returning different results based on client-location would do so based on the location of the `LLUAD` server, much like they would with an intermediate public DNS resolver. Functionality such as the EDNS(0) Client Subnet Extension, which allows resolvers to forward the subnet of the querying user to allow the authoritative name server to return a geographically appropriate response [8], is not supported due to its implications on privacy.

The relevance of the records in the `Popularity List` is ensured by users (connected clients) continually casting anonymous `Votes` on records to include in the list. These `Votes`, reflecting user queries during the last $t_{refresh}$ seconds (saving resolved records as potential `Votes`), are submitted simultaneously by all clients at set $t_{refresh}$ intervals through a `Voting Mix Network` (detailed in Section 4.2), optimized for our `LLUAD` scheme, allowing the `LLUAD` server to periodically refresh the `Popularity List`. The `Votes` only specify which DNS records to include, not their corresponding translations since the `LLUAD` server performs the lookups. This prevents any potentially malicious client from poisoning records in the list.

As shown in Figure 2, our `Voting Mix Network` does not use third-party servers to mix `Votes`. Instead, connected clients act as mix nodes that collectively anonymize the `Votes`. Another distinct feature of the `Voting Mix Network` is that the `LLUAD` server relays packets (`Votes`) between mix nodes (clients), a more efficient and implementable approach (see Section 4.2) compared to nodes directly communicating. The process of the `Voting Mix Network`, as illustrated by the enumerated steps in Figure 2, is as follows:

(1) All clients submit their `Votes` simultaneously to the `LLUAD` server. The `Votes` are iteratively encrypted using the keys of, in this example, five other mix nodes (clients) in the network, meaning they need to be received and processed by a sequence of five predetermined mix nodes (not counting the intermediate hops via the `LLUAD` server) before their content is revealed.
(2) The `LLUAD` server sorts the received `Votes` into batches based on their next-hop destination.
(3) The `LLUAD` server redistributes the batches of `Votes` to their corresponding client mix nodes.
(4) After receiving a batch of `Votes`, each client decrypts each top layer of encryptions and randomly shuffles the order of their `Votes`, obscuring their origins among each other.

The process is repeated until all `Votes` are fully decrypted, allowing the server to evaluate the current popularity of records and update the `Popularity List` accordingly. The voting mechanism is not latency sensitive. Therefore, the system can accommodate longer delays between each shuffling round to support users with high latency.

To mitigate Sybil attacks, where attackers connect with a large number of identities to obtain greater influence on the `Popularity List` or undermine the anonymization of the `Voting Mix Network`, users must register in advance to an external Public Key Infrastructure (PKI) and authenticate with their obtained credentials (certificate) when connecting to the `LLUAD` server. Note that, similarly to anonymous networks, e.g., Tor, where ISPs can trivially observe users connecting to Tor entry nodes, `LLUAD` does not anonymize the identity of connected users to the `LLUAD` server, but rather their activity. The registration process can require a phone number, etc., to thwart large-scale Sybil attacks. To ensure the `LLUAD` server does not perform similar attacks, users can validate the identity of peer clients via the certificates obtained from the external PKI during the registration. Specifically, these certificates are used to verify the public keys of other clients in the `Voting Mix Network` (detailed in Section 4.2.1). Privacy can be further improved by using ephemeral pseudonymous certificates (used for vehicular communication settings [3, 26]) to hide user/device identifiable information. [1]

4.1 Construction of the Popularity List

The `Popularity List` stored by clients contains only the domain, record type, and answer of the top $N_{popular}$ DNS records, excluding the root domain. Thus, redundant data such as DNS class and TTLs are omitted (the latter is tracked by the `LLUAD` server, which keeps the records up-to-date). Apart from the top $N_{popular}$ records,

[1] The ramifications for different approaches of registering users and distributing public keys/certificates cannot be fully addressed in this paper and are generally orthogonal to the privacy and quality of DNS resolutions. Thus, we will not elaborate further.

the Popularity List also includes additional records that at least one of the top records redirects to using so-called CNAME records (records that redirect/point to another domain). The Popularity List uses a hierarchical structure to minimize its overall size, avoiding duplicate data such as repeating the Top-Level Domain (TLD) ".com" after each domain entry. For instance, the Popularity List in Figure 3a contains four ".com" domains, yet only stores "com" once, as all four domains, due to their position in the tree, implicitly belong to the TLD. Each record answer in the tree (indicated in blue in Figure 3a) is compacted, only containing the record type identifier and the corresponding answer (e.g., an IP address). Users query the local Popularity List service using the DNS protocol, where the service searches the structure and responds with a correctly formatted DNS response. If the sought record is not found in the local list, the query is forwarded to an external privacy-preserving but latency-costly DNS service (see Section 4.3).

To resolve *mail.internal.example.com* in Figure 3a, the service starts at *com* and goes directly to the *example* section. Since *www* is not the correct subdomain, the first subdomain is skipped, finding the sought *mail.internal*. In this example, *mail* and *internal* are combined to reduce computational and memory overhead, as the latter has no answers and only one subdomain (*mail*) directly below it. CNAME records do not store their referenced domain in cleartext. Instead, they contain pointers to relevant domain labels in the Popularity List to reduce overhead. For example, the CNAME record of *www.example.com* in Figure 3a could redirect to *example.com* by pointing to the headers of *com* and *example*.

4.1.1 Actively Load-Balanced Records. As described in Section 2, DNS records can be actively load-balanced to dynamically distribute connections across servers. These records often have TTLs as short as 30 s and change with almost every query. This would result in substantial network traffic as the LLUAD server would broadcast each incremental update to the Popularity List. To reduce network overhead, clients store all potential answers for all actively load-balanced domains in a shared Load-Balancing Pool at the end of the Popularity List (e.g., all IP address answers observed by the server as it re-queries these records), as illustrated in Figure 3b, omitting duplicate answers. The affected entries in the Popularity List tree structure point to the byte location of their corresponding answer in the pool.

Prefetching all potential answers allows the LLUAD server to efficiently update a record answer in the Popularity List by sending two integers: the number of the Popularity List entry to update (counting only actively load-balanced ones) and how many answers the pool pointer should be offset by, rather than sending the full record and corresponding answer, substantially reducing communication overhead. Non-load-balanced domains store their answers directly in the tree structure of the Popularity List, as exemplified by *example.com* in Figure 3b. The answers of actively load-balanced domains are added in alphabetical order to the Load-Balancing Pool based on (i) domain name, (ii) numerical record type, and (iii) answer content, omitting those already added.

To reduce memory/storage and bandwidth overhead, clients receive only one randomly selected answer per record type and domain from the LLUAD server when multiple answers are available. For example, if a DNS response (from an authoritative name server)

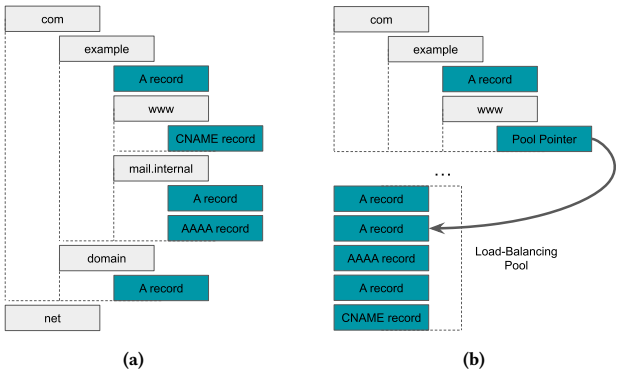

Figure 3: (a) Example of a Popularity List, (b) with an appended Load-Balancing Pool containing all discovered answers of all actively load-balanced records.

contains three different A record answers (IPv4 addresses), the LLUAD server only distributes one randomly chosen address to each client. This preserves passive load balancing (assuming answers are distributed uniformly) while reducing overhead. Since there is only one active/stored answer per record, incremental updates to actively load-balanced records only require offsetting a single pool pointer instead of multiple, further minimizing network overhead.

To avoid the otherwise large overhead of packet headers, multiple incremental updates to actively load-balanced records are concatenated into a single transmitted message, sent at most once per minute by default. As a result, records with a TTL shorter than one minute may not be fully adhered to. However, the Chromium DNS cache (the basis of most web browsers) caches records for exactly one minute, regardless of the specified TTL [1, 17]. Thus, enforcing a one-minute minimum TTL has no practical impact on the DNS protocol.

4.1.2 Number of Votes to Contribute. Two primary parameters control the number of Votes each client can/will contribute per voting round, determining the communication and computational overhead, and any client's influence on the Popularity List. Moreover, the difficulty of reconstructing individual user profiles from the collected pool of Votes increases if users refrain from voting on most of their queries performed during the last $t_{refresh}$ period. The configurable parameters are:

(1) **Voting-rate:** The probability that a client will submit any performed query as a Vote to the LLUAD server.
(2) **Maximum Votes per voting-round:** The upper limit of number of cast Votes per $t_{refresh}$ period per client.

Clients do not include how many times they have queried the record of each Vote, as it is difficult to enforce the correctness of said value. Furthermore, this way, the Popularity List reflects the popularity of DNS records of a general population rather than the niche interests of highly active individual users. Although the LLUAD scheme does not prevent clients from maliciously submitting multiple Votes for the same DNS record, the impact of such behavior is thwarted by restricting the maximum number of Votes

per client and refresh period (parameter (2) above). The details of how our scheme limits the number of anonymous Votes per user are given in Section 4.2.

4.1.3 Impact of Each Voting Round. The user-cast Votes determine which $N_{popular}$ records are included in the Popularity List. Each record is ranked according to its weighted sum of received Votes from each voting round (at the end of each $t_{refresh}$ period). The weighted sum of the latest m-th system-wide voting round is

$$N_{weighted_m} = a \cdot N_{occurences_m} + (1 - a) \cdot N_{weighted_{m-1}}, \quad (1)$$

where a is the weight for the latest voting round, $N_{occurences_m}$ is the number of Votes received for the record during voting round m, and $N_{weighted_{m-1}}$ is the weighted sum from all previous rounds. Initially, $N_{weighted_0} = 0$. The weight a is empirically chosen to maximize the Popularity List hit ratio, while balancing the volume of updates and the responsiveness to changes in popularity.

After updating record rankings at the end of a $t_{refresh}$ period, the LLUAD server sends messages to connected clients to inform which records/answers to remove/add to the Popularity List and its Load-Balancing Pool to update them accordingly. Messages removing records/answers are compacted by referencing domains in the same manner as Popularity List CNAME entries.

4.2 Mix Network for Vote Anonymization

The simplest way for clients to vote anonymously would be through a public anonymous network (e.g., Tor). However, such third-party networks would prevent the LLUAD server from limiting the number of Votes cast per user, allowing attacks on the voting system, diluting and effectively poisoning the Popularity List with unpopular records, to lower its hit ratio. Furthermore, to maintain unlinkability between Votes, each user would have to establish a new circuit (series of tunnels) for every Vote, making this approach inefficient. We address the above issues with our specialized Voting Mix Network.

Objectives: The Voting Mix Network must ensure that individual Votes are anonymous and that no one, including the LLUAD server, can link multiple Votes to the same (anonymous) user profile. Furthermore, LLUAD should not rely on volunteering third-party hosted mix node servers, or assume full trustworthiness of users. Additionally, it must function independently of firewalls and Network Address Translations (NATs) and impose minimal bandwidth overhead, comparable to the lightweight DNS protocol. Recall also that LLUAD needs to restrict the number of Votes per user (to safeguard the Popularity List).

Having all clients simultaneously submit their Votes to the Voting Mix Network results in the maximum achievable anonymity set of potential packet/Vote origins. Additionally, each user Vote is sent through a unique Voting Mix Network path (a unique pseudo-random sequence of client mix nodes) to reduce Vote linkability between Votes from the same user. A compact packet format is used to minimize the bandwidth overhead typically associated with mix networks. Specifically, the lengths of the Vote packets are independent of the number of mix node hops and do not require pre-established tunnels. To ensure minimum leakage of user information, the number of Votes cast (per voting round) by each client is obscured. In particular, clients with fewer Votes than the maximum allowed limit will submit empty ones to the Voting Mix Network to fulfill the quota.

By routing all packets through the LLUAD server, the system improves efficiency by reusing existing connections, bypassing the need for clients to continuously open new connections to distribute the Votes, all with unique next-hop addresses. Reusing the server connection also avoids firewall and NAT implications. In addition, multiple Votes can be concatenated in the same IP packet to reduce the overhead of packet headers. Lastly, the server can enforce a limit on the number of Votes sent by each client, as it controls all flows in the Voting Mix Network. The synchronization of mixing/shuffling rounds prevents the server from analyzing the packet flows further, thwarting any risks of tracing the origin of Votes, an otherwise considerable issue with a single entity observing all mix node flows in a mix network.

4.2.1 Packet Format. Each Vote passes through a set of client mix nodes and exits at the LLUAD server. The number of these mix nodes, $N_{shuffle}$, determines the number of rounds that Votes are shuffled. However, the number of stacked encryptions is $N_{shuffle} + 1$, as the final encryption is removed by the LLUAD server. Each of the $N_{shuffle} + 1$ target destinations has index $i = \{1, 2, ..., N_{shuffle} + 1\}$.

Each Vote sent through the Voting Mix Network consists of three components: a public key element, p_i, a next-hop hash, h_i, and a payload, d_i. As indicated by the index of each component, all fields are transformed at each traversed node to maintain the integrity and anonymity of the Vote. If each element did not change, it would be trivial for an observing party (such as the LLUAD server) to map incoming and outgoing Votes at each shuffling client and deduce the traveled path, thus rendering the packet mixing obsolete. $Node_i$ derives p_{i+1}, h_{i+1}, and d_{i+1} from p_i, h_i, and d_i and sends them to the LLUAD server, which forwards them to $Node_{i+1}$.

Public key element: a 32-byte elliptic-curve public key element [9] used by a receiving $Node_i$ to derive, when performing Diffie-Hellman with its private key, the symmetric key s_i used for the current top encryption layer of a Vote packet.

Next-hop hash: a 16-byte field functioning as an address, mapping to the identity of the current next-hop client mix node using a function similar to Distributed Hash Tables (DHTs). For example, the hash value modulo the number of client mix nodes, resulting in h_i to map to the address of $Node_i$, h_{i+1} to $Node_{i+1}$, etc.

Payload: a 32-byte field containing the record the user is voting on, the first byte being a general flag field mentioned in Section 4.2.4. The original payload d is iteratively encrypted with symmetric keys $\{s_{N_{shuffle}+1}, s_{N_{shuffle}}, ..., s_i, s_{i-1}, ..., s_1\}$ (in order), where the original sender and $Node_i$ share the symmetric key s_i ($Node_{N_{shuffle}+1}$ is the LLUAD server). Thus, the payload d_i received by $Node_i$ is encrypted with $\{s_{N_{shuffle}+1}, s_{N_{shuffle}}, ..., s_{i+1}, s_i\}$.

4.2.2 Next-Hop Hash. The next-hop address of each node in a mix network path is typically appended at each encryption stage, which can be later obtained by removing the top encryption layer. However, appending multiple addresses would add noticable overhead to the LLUAD voting system. Instead, each Vote packet uses a fixed-length pseudo-random hash that maps to a next-hop address and is rehashed after each mix node to reflect the next destination. $Node_i$

recalculates the received hash h_i with

$$h_{i+1} = Hash(h_i, s_i, t_{timestamp}), \quad (2)$$

where $t_{timestamp}$ is a system-wide variable representing the current voting round, ensuring a unique next-hop sequence, and h_1 is initialized as a random value. While $t_{timestamp}$ and h_i are known/visible to observers, s_i is only known by the original sender and $Node_i$. Thus, only the original sender and $Node_i$ can create h_{i+1}, where only the sender can derive the entire sequence of mix nodes with all required symmetric keys $\{s_1, s_2, ..., s_{N_{shuffle}+1}\}$.

The next-hop hash does not explicitly indicate when the packet should exit the `Voting Mix Network`. However, the LLUAD server that orchestrates the shuffling rounds can trivially track the current round number and update the `Popularity List` with the `Votes` in the final round. Note that consequently, all clients must follow the same system-wide $N_{shuffle}$.

4.2.3 Blinding the Public Key Element. Ensuring that all fields of the `Vote` packets continuously change each hop is generally straightforward. For instance, the next-hop hash is automatically refreshed into a new seemingly random value, and so is the payload after each decryption. However, the public key element p_i used to derive s_i remains inherently static. Thus, LLUAD uses a technique found in the Sphinx mix network [9], where the public key element is blinded at each hop, transforming it into another public key element. The transformation involves the shared key, s_i, which means that only the original sender and the specific mix node know the mapping between the incoming and the outgoing public key elements (p_i and p_{i+1}). $Node_i$ blinds p_i with

$$p_{i+1} = Blind(p_i, b_i), \quad (3)$$

where

$$b_i = Hash(p_i, s_i) \quad (4)$$

is the blinding factor applied by $Node_i$. The symmetric key s_i shared between the original sender and $Node_i$ is derived by the latter using Diffie-Hellman (DH):

$$s_i = DH(p_i, k_{priv_i}), \quad (5)$$

where k_{priv_i} is the private key of $Node_i$. Much like how the original sender can create the entire chain of next-hop hashes, h_i, for all $i = \{1, 2, ..., N_{shuffle} + 1\}$ during the creation of the mix network packet, they can also generate p_i for all i. Table 1 shows all the steps for generating the original `Vote` packet, where the final step is to iteratively encrypt the payload, d, with all derived symmetric keys.

4.2.4 Payload Format. Each `Vote` includes the type and domain of the corresponding record, where the latter can vary significantly in length. Since length is a distinct feature of individual packets, the `Votes` would be traceable throughout the `Voting Mix Network`. Thus, all packets are padded to a uniform length. A fixed number of bytes is allocated for the record type and domain name; specifically, the 31 last bytes of the payload d. If the combination of record type and domain exceeds this length, it is hashed to fit within the limit. The LLUAD server can interpret the hash if it has previous knowledge of the record.

While the server can build a database of known DNS records and their corresponding hashes over time, it must be able to identify records on their first encounter. The server could use an external service that provides practically all public domains [36]. Nevertheless, to allow LLUAD to function independently, the server can request assistance from its clients. The server can broadcast a message to indirectly request the client who submitted the hash to anonymously provide the record in cleartext during the next voting round. Anonymity/Unlinkability is maintained as the client splits the cleartext record across several `Vote` packets, using the leading flag and a random ID number occupying the first two bytes of the 31 in the payload to help the server reconstruct the otherwise disassociated messages as they exit the `Voting Mix Network`.

4.2.5 Tracking Other Clients. Each client must pre-fetch the public keys of all clients in the paths of its `Votes`. Since mix node paths are randomly selected using the pseudorandom next-hop hashes (can at most re-roll the sequence by selecting a new initial next-hop hash h_1 to build the sequence from), this requires clients to store the public keys of all other clients in the system, leading to significant storage overhead in the case of many users. To reduce public key storage overhead, the shuffling role can be assigned to a subset of randomly chosen users, potentially rotating the assigned roles over time to distribute overhead and trust in any one entity. The additional overhead on these users is not of major concern, as the overhead of shuffling `Votes` is only a small part of the total overhead of LLUAD. To counteract collusion and ensure the integrity of the `Voting Mix Network`, the LLUAD server must not dictate the assignment of these users. Note that since the LLUAD server redirects `Votes` based on next-hop hashes, clients do not need to store the IP addresses of other clients.

As user devices could go offline at any moment, all participants must know which clients (assigned the shuffling role) are active and available to mix `Votes`. Thus, the LLUAD server initiates each voting round by broadcasting a message containing a bit stream with each bit representing the availability/online status of each client. The first bit implies whether the first shuffling client (first public key) should be considered, the second bit the second client (second public key), etc. This also allows users to assess the current size of the `Voting Mix Network` anonymity set.

4.2.6 Vote Acknowledgments. While all `Votes` can be anonymously contributed to the server through the `Voting Mix Network`, malicious actors could covertly perform several attacks. For example, a client participating in shuffling `Votes` could add, remove, or replace them. Although the LLUAD server can observe discrepancies in the number of sent and received `Votes`, thus revealing clients adding or dropping `Votes`, a client could still corrupt or replace `Votes` to influence the `Popularity List`. To detect whether a `Vote` was corrupted or replaced, the LLUAD server sends an authenticated acknowledgment for each received `Vote`, allowing users to confirm that the server correctly received each of their `Votes`, and be notified of any malicious behavior within each corresponding `Voting Mix Network` path. Users may report detected misbehavior (reporting the selected path of clients) to the LLUAD server, which can identify the malicious client by correlating sufficiently many reports. The acknowledgments are anonymously sent to the original senders by shuffling them in synchronized rounds similar to the `Votes`. Specifically, all acknowledgments traverse the reverse path of their corresponding `Vote`, with each acknowledgment sent

Table 1: Calculated elements, in order left to right and top to bottom, to create a `Voting Mix Network` `Vote` packet.

Step	(1) Next-hop hash	(2) Public key	(3) Symmetric key	(4) Blinding
1	$h_1 = Random\ hash\ value$	$p_1 = Gen_{public}(k_{priv_0})$	$s_1 = DH(k_{priv_0}, k_{pub_1})$	$b_1 = Hash(p_1, s_1)$
2	$h_2 = Hash(h_1, s_1, t_{timestamp})$	$p_2 = Blind(p_1, b_1)$	$s_2 = DH(k_{priv_0}, k_{pub_2}, b_1)$	$b_2 = Hash(p_2, s_2)$
		...		
$i\ (\leq N_{shuffle})$	$h_i = Hash(h_{i-1}, s_{i-1}, t_{timestamp})$	$p_i = Blind(p_{i-1}, b_{i-1})$	$s_i = DH(k_{priv_0}, k_{pub_i}, b_1, ..., b_{i-1})$	$b_i = Hash(p_i, s_i)$
		...		
$N_{shuffle} + 1$	$d_1 = Enc(Enc(...(Enc(Enc(d, s_{N_{shuffle}+1}), s_{N_{shuffle}}), ...), s_2), s_1)$			

to the immediate predecessor of the corresponding `Vote`. The acknowledgment payload, created by the `LLUAD` server (replacing d) is:

$$r = Hash(d, s_{N_{shuffle}+1}, t_{timestamp}), \tag{6}$$

where $s_{N_{shuffle}+1}$ is the symmetric key shared between the original sender and the `LLUAD` server. To reverse the path through the mix network, messages are sent as follows, where the receiving node encrypts the payload with s_i instead of decrypting it:

(1) $Node_i$ receives a `Vote` packet containing p_{i+1}, h_{i+1}, and r_{i+1} from $Node_{i+1}$ via the `LLUAD` server.
(2) $Node_i$ encrypts r_{i+1} with s_i to generate r_i.
(3) $Node_i$ replaces p_{i+1} and h_{i+1} with p_i and h_i (saved and used as flow identifiers from the original `Vote` shuffling rounds).
(4) $Node_i$ sends p_i, h_i, and r_i to the `LLUAD` server, forwarding them to $Node_{i-1}$, which repeats the process (by tracking the `Vote` shuffling flows, the server can map h_i to $Node_{i-1}$).

When the original sender receives r_1 from $Node_1$, it can decrypt all $N_{shuffle} + 1$ encryptions, as it possesses all symmetric keys. Subsequently, using Equation (6), it can authenticate that r is the acknowledgment of the originally transmitted d. Similarly to the `LLUAD` server in Section 4.2.2, clients can trivially track when the acknowledgments should exit the `Voting Mix Network`.

In each acknowledgment shuffling round, each shuffling client expects a set of acknowledgments responding to the `Votes` it processed during the corresponding `Vote` shuffling round. Thus, the shuffling clients can generate dummy cover acknowledgments to replace any missing ones (potentially dropped by a malicious party) to ensure that no entity is able to drop the acknowledgment of a specific `Vote` and observe who does not receive it. The cover acknowledgments (generated in place of missing ones) are generated by appending random bytes to p_i and h_i in place of r_i. Thus, the original sender is notified of potential misbehavior as they could not authenticate the acknowledgment due to r being random bytes.

4.3 Resolving Records Outside the List

Records not found in the `Popularity List` are resolved externally using what we term a *fallback DNS protocol*. The chosen fallback DNS protocol depends on the desired performance and privacy. We consider the following configurations, with (4) and (5) being the most relevant:

(1) **Unencrypted DNS:** No protection for queries outside the `Popularity List`, but the overall exposure is reduced as most queries are resolved locally by the `Popularity List`. We use Cloudflare, 1.0.0.1/1.1.1.1.
(2) **DoH:** Adds encryption to external queries to protect against external eavesdroppers. We use Cloudflare, 1.0.0.1/1.1.1.1.
(3) **DNSCrypt with rotating resolvers:** Continuously changing resolvers to reduce exposure to any individual one. We randomly query one of the DNSCrypt resolvers in [15].
(4) **Anonymized DNSCrypt with rotating relay/resolvers:** Reduces the exposure to any resolver-relay pair, where a large number of resolver-relay pairs can be rotated between [15]. We randomly query one of the DNSCrypt resolvers in [15], each with a permanently assigned random relay.
(5) **DoHoT:** Lessens the risk of colluding relays and resolvers by forwarding queries through Tor, at the cost of high latency. We use a standard configuration of three nodes for Tor and Cloudflare's 1.0.0.1/1.1.1.1 as the resolver.

5 `LLUAD` Evaluation

In this section, we quantitatively evaluate `LLUAD`. We evaluate the hit ratio of the `Popularity List` in Section 5.2, the rate at which DNS queries are exposed to external entities in Section 5.3, and the scheme's performance overhead, i.e., latency, memory/storage, and network bandwidth in Section 5.4, 5.5, and 5.6, respectively. We compare the results with widely deployed DNS privacy schemes. Due to space limitations, arguments for the security and unlinkability/anonymity of `LLUAD` are embedded in its presentation in Section 4, providing the reasoning behind each corresponding design choice.

5.1 Dataset

We simulate real-time operation using a dataset that contains real-world DNS queries and responses from a campus network with up to 4000 concurrent active hosts [32]. Only DNS packets querying A or AAAA records (IPv4 and IPv6 addresses) are considered. Packets with malformed responses, server errors, and non-compliant FQDNs are excluded, as are retransmissions whenever they affect cache hit ratios. 172.31.3.121 serves as the main resolver for the network, indicated by its interaction with DNSSEC-related record types (used by resolvers to authenticate DNS data [30]). 172.31.3.121 communicates solely with 172.31.1.6, which serves thousands of clients. To avoid counting the same packet on multiple links or missing queries responded to by the DNS cache of an earlier router, only traffic between clients and 172.31.1.6 is considered.

5.2 Popularity List Hit Ratio

5.2.1 Initialized List. Figure 5 shows the measured hit ratio of the `Popularity List` for different list sizes: the percentage of user

queries in the dataset the list could resolve, as each DNS query is sequentially processed, simulating a real-world application of LLUAD. Shown is the mean hit ratio for each hour of the last nine out of ten days of the dataset, with each user (IP address) casting up to 10 `Votes` per hour ($t_{refresh} = 3600$), with a voting rate of 0.3 (see Section 4.1.2). For the first 18 hours, all user queries are considered `Votes` to generate the initial `Popularity List` (*fast start* mode). The weight of the latest voting round (see Equation (1)) is set to $a = 0.1$. The values (default from here on) were selected to balance the hit ratio, privacy, and communication and computational overhead. The dashed plots show the mean hit ratio from day five and onward when no `Votes` other than from the first 18 hours have been contributed to the `Popularity List`, simulating a sudden long-term stop in `Votes`.

The hit ratio quickly plateaus as the list size increases, achieving over 90 % with $N_{popular} = 10\,000$. Interestingly, $N_{popular} = 100$ manages about 40 %, demonstrating the high popularity of a small subset of DNS records. For optimal performance, a list size of at least 10 000 is recommended, with 25 000 considered the default value, achieving a mean daily hit ratio of around 94.4 %, which means 94.4 % of user DNS lookups are performed locally. Increasing the list size further to 100 000 only marginally improves the hit ratio. The marginal drop in the hit ratio as no `Votes` are received after the first 18 hours, resulting in a `Popularity List` that is between four and nine days out of date, shows that LLUAD is robust against Denial-of-Service (DoS) attacks on the voting system orchestrated by the `Voting Mix Network` (e.g., many clients dropping `Votes`).

5.2.2 During Initialization with Few Users. Figure 6 shows the mean hit ratio and mean number of records in the `Popularity List` per day as it is instantiated with only a handful of participating users (during each voting round, only the `Votes` of a randomly selected subset of clients in the dataset is considered). Compared is with *fast start* (users vote for all their performed DNS resolutions) and without.

The `Popularity List` achieves a high hit ratio quickly even in cases where there are only a handful of users. The scheme achieves a mean hit ratio of around 66 % during the second day with `Votes` from a maximum of ten users per hour (no *fast start*). Around 81 % is achieved if the number of voters increases to a mere 50, matching that of 10 voters with *fast start*. In other words, the initialization of the `Popularity List` remains effective despite a minimal pool of participating users.

5.3 Exposure/Loss of Privacy

Figure 4 shows the *exposure rate* of user DNS queries, i.e., the probability that a user-performed resolution can be connected to its user origin by a DNS resolver or the LLUAD server (due to encryption, these are the only respective entities with direct access to the content of queries). Compared are different protocols, considering varying collusion rates between anonymizing relays (relay in Anonymized DNSCrypt, Tor nodes, other clients in LLUAD, etc.) and the resolver/LLUAD server. Different fallback protocols and varying numbers of system users (voters) are compared for LLUAD while mixing 10 `Votes` in each of the 10 shuffling rounds for a list of size 25 000.

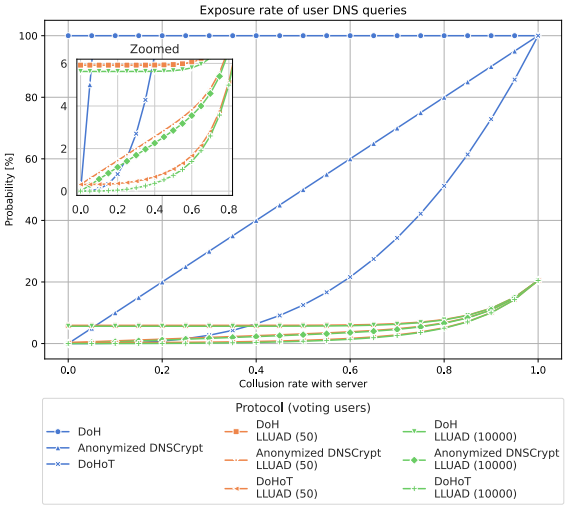

Figure 4: The probability that a DNS resolver or the LLUAD server can connect a user-performed DNS resolution to its IP address origin for different protocols and collusion rates between entities and the DNS resolver/LLUAD server. The DNS protocols on their own are plotted in blue, and are, when used as a fallback protocol for LLUAD using 50 and 10 000 voters, plotted in orange and green, respectively.

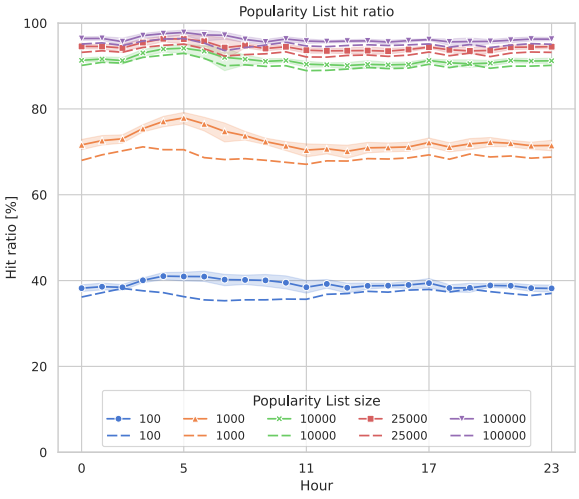

Figure 5: The `Popularity List` hit ratio during each hour of the day for various $N_{popular}$ values with $t_{refresh} = 3600$, where the dashed plots simulate a long-term stop of all `Votes`.

Due to the lack of anonymizing relays, DoH offers no protection against curious DNS resolvers on its own, resulting in all queries being exposed. Anonymized DNSCrypt, with its single relay, can obscure user queries as long as the two entities are not colluding. DoHoT further reduces the exposure rate by using three separate relays. We have here assumed that no external observer (e.g., an ISP) performs volume or timing attacks on observed network traffic and that the pool of users (the anonymity set) is large enough

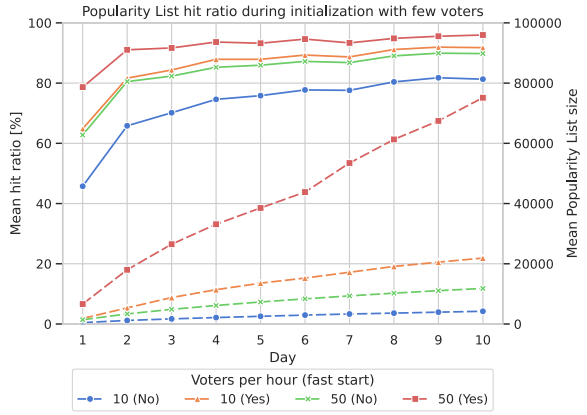

Figure 6: Mean hit ratio (non-dashed) and mean $N_{popular}$ (dashed) per day for a `Popularity List` initialized by only a few participating voting users.

that it can be approximated as infinite. LLUAD, without any such assumptions for its `Voting Mix Network`, significantly reduces the exposure rate in environments with colluding entities. The exposure rate for LLUAD includes that caused by `Votes`, as it is assumed that the number of voting users is small enough to affect privacy, hence LLUAD with 50 voters slightly increasing the exposure rate of Anonymized DNSCrypt and DoHoT as the collusion rate approaches zero in Figure 4. Nevertheless, the exposure rate with 50 voting users is surprisingly similar to the, in practice, perfect setup of 10 000, implying that LLUAD is robust against poor user participation.

As expected, DoHoT as the fallback protocol results in the lowest exposure, with Anonymized DNSCrypt a close second. The generally low exposure rate of LLUAD is largely due to only around 5.6 % of queries being resolved externally and around 15.7 % being cast as `Votes` on average. The result is that LLUAD achieves an exposure rate of around 20.5 % in scenarios where all nodes are colluding, providing the user with a higher guarantee of privacy compared to widely deployed alternatives. The exposure rate can be further reduced by decreasing the frequency of voting rounds (increasing $t_{refresh}$). The impact on the hit ratio should be minimal as indicated in Figure 5, where a `Popularity List` four to nine days out of date has a hit ratio only a few percentage points lower than one refreshed hourly.

User Activity Status: The scheme reveals the activity status of users assigned to shuffling `Votes` by broadcasting their status to all users at the start of voting rounds. In addition, the server is trivially aware of which users are currently connected. However, specific activity levels (e.g., DNS queries per hour) are not revealed, mitigating most privacy concerns, as ISPs can obtain significantly more information regarding users' current activity rates.

5.4 Lookup Latency

Figure 7 shows the mean DNS lookup latency of LLUAD, using a `Popularity List` of size 25 000 to resolve most DNS records. We compare the results of multiple different fallback DNS protocols to resolve queries/records outside the `Popularity List`, as introduced in Section 4.3. The mean latency of these DNS protocols on their own is displayed on the left and right sides of the figure to clearly show the significant improvements with the `Popularity List`. Our latency measurements (measured using Wireshark/Tshark) were obtained by sampling 1000 lookup times based on the hit ratio from Figure 5. For each sampled hit, a random record within the `Popularity List` (implemented in C) is resolved in a UTM Kali Linux Virtual Machine (VM) on a Macbook Pro M1 Pro (four allocated CPU cores). For each miss, the same setup randomly queries one of the 500 most popular records in the dataset [32] externally using a 2.4 GHz Wi-Fi 6 connection, including the overhead of first querying the `Popularity List` on the *loopback* interface (but missing). Any domain in the 2016 dataset that is now an NXDOMAIN (non-existent domain) was not considered for the top 500. In Figure 7, "New" includes opening a new TCP/TLS connection to the resolver, compared to "Established".

Using a `Popularity List` of size 25 000 significantly decreases the mean latency across all DNS protocols. It is especially noticeable for new DoHoT connections, where the mean latency drops from over 1 s down to between 80 ms and 90 ms, greatly improving the viability of the protocol. However, one should note the variance in lookup latency for such a `Popularity List` configuration, as most lookups are below 1 ms with a few exceeding 1 s.

5.5 Memory/Storage Overhead

5.5.1 Popularity List. Figure 9a shows the mean size in bytes of the `Popularity List` for varying values of $N_{popular}$ during all hours of the last nine days in the dataset [32]. The impact of flattening CNAME chains (excluding the intermediate redirect domains) reveal modest but generally inessential gains. Overall, the `Popularity List` scales well as $N_{popular}$ increases, where a 150 % increase in size (10 000 to 25 000) results in only a 60.5 % increase in byte size (no CNAME flattening). Considering that the main use case of the LLUAD scheme is to improve the privacy of web browsing, the memory overhead of the `Popularity List` is negligible. For comparison, Google Chrome self-reports a memory usage of around 30 MB to 45 MB for a single tab of the Google Search landing page in Windows 11. Meanwhile, the YouTube homepage reported between 250 MB and 500 MB.

5.5.2 Public Keys. For a system with 10 000 shuffling users, the storage overhead is 0.32 MB (32 B per key), noticeably less than the `Popularity List` of around 1.2 MB (25 000 records without CNAME flattening in Figure 9a). The origins of these keys are verified through client certificates obtained during registration (see Section 4). We assume clients are not required to permanently store other clients' certificates locally. For example, the certificates might be publicly accessible on the LLUAD server, allowing clients to (optionally) fetch them to verify public keys at any point to reduce storage requirements.

5.6 Network Bandwidth Overhead

All network bandwidth measurements are performed in an Ubuntu 22.04 VirtualBox VM, where all client-to-LLUAD server communications are sent over TLS using the Python *ssl* library. Client-to-LLUAD server communications are simulated over the *loopback* interface with a Maximum Transmission Unit (MTU) of 1500.

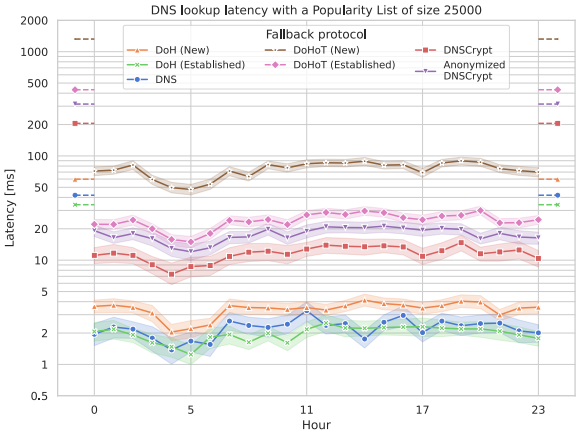

Figure 7: Mean LLUAD DNS lookup latency per hour using various fallback DNS protocols. Mean latency without LLUAD shown on the left and right ends (below 0 and above 23).

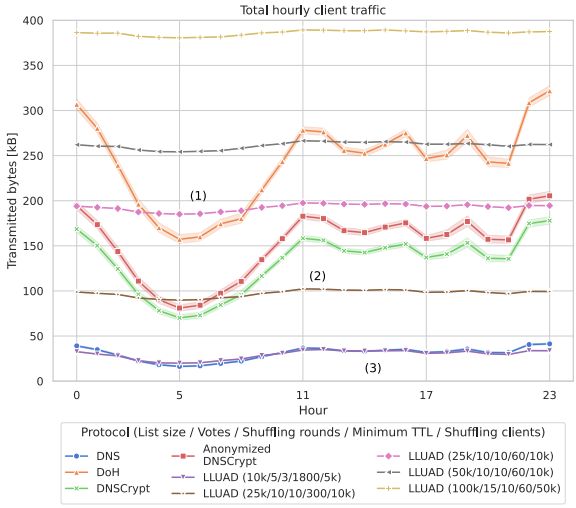

Figure 8: Total number of bytes sent and received per hour by a client for different DNS schemes, where LLUAD utilizes Anonymized DNSCrypt as its fallback protocol.

Figure 9: (a) Popularity List size in bytes as a function of $N_{popular}$, with and without CNAME flattening. (b) Total sent and received bytes to download the Popularity List (without CNAME flattening) over TLS, with and without compression.

5.6.1 Downloading the Popularity List. Figure 9b shows the mean number of transmitted bytes to download the Popularity List for each hour of the last nine days in the dataset [32]. Compared is also the overhead when compressing the list using the Python *zlib* library. Unsurprisingly, without compression, the number of bytes is similar to the size of the Popularity List, as seen in Figure 9a. Compressing the Popularity List significantly reduces the number of transmitted bytes, reducing it by around half.

5.6.2 Continuous Client Traffic. Figure 8 shows the mean total hourly bandwidth overhead for each user/client (i.e., source IP address that generates at least one DNS query during the hour) in the dataset [32] for various DNS configurations (excluding the initial download of the Popularity List for LLUAD). The LLUAD configurations use Anonymized DNSCrypt with randomized resolver-relay pairs as its fallback protocol. Included for LLUAD is: (i) the overhead of receiving incremental updates to the Popularity List (includes incremental updates to actively load-balanced records and replacing records due to changes in popularity, with compression applied), (ii) broadcasting the availability of shuffling clients, (iii) all Vote shuffling rounds, and (iv) falling back to Anonymized DNSCrypt according to the hit ratio in Figure 5. The frequency that DNS answers change (causing the incremental updates) is extrapolated from measurements on the corresponding real-world domains by continuously re-querying a large subset of the domains in [32] for an hour (only basing the measurements on domains that have not become NXDOMAINs since 2016). The bandwidth overhead of DNS queries is estimated using the mean overhead of resolving the top 500 most popular records from Section 5.4. The TCP-based protocols DoH and DoHoT use a 30 s connection keepalive.

The overhead of LLUAD using Anonymized DNSCrypt as fallback is similar to that of existing DNS protocols. Specifically, the default configuration (25 000 records, 10 Votes per user and hour, 10 shuffling rounds, 60 s minimum TTL for actively load-balanced records, and 10 000 shuffling clients) is comparable to standalone Anonymized DNSCrypt, outperforming DoH (see (1) in Figure 8). With a minimum TTL of 300 s, LLUAD outperforms DNSCrypt (see (2)). A lighter configuration (10 000 records, 5 Votes, 3 shuffling rounds, 30 min TTL, and 5000 mixing clients) matches the traditional DNS (see (3)) while improving privacy and reducing latency at the cost of less than 1 MB (10 000 records and 5000 public keys) in memory/storage.

5.6.3 Continuous Server Traffic. Figure 10 shows the mean network bandwidth usage of the LLUAD server for the scheme's default configuration for various $N_{popular}$. Included is the overhead of continuously re-querying public DNS resolvers using DoH to ensure records in the Popularity List are up-to-date, and all data transmitted between the server and all its currently connected clients (excluding the initial client downloads of the Popularity List). The overhead is considered insignificant, with only around 25 Mbit/s for the default configuration ($N_{popular}$ = 25 000) with

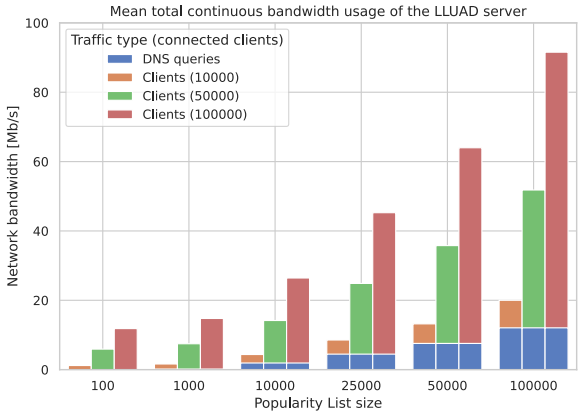

Figure 10: Mean continuous bandwidth usage of the LLUAD server, including: (i) re-querying DNS records in the `Popularity List` as their TTLs expire, and (ii) all data continuously transmitted to/from connected users (10 `Votes` per user, 10 shuffling rounds, 60 s minimum TTL, and 10 000 mixing clients).

50 000 currently connected users. Even the most extreme configuration with $N_{popular}$ = 100 000 and 100 000 active users does not surpass 100 Mbit/s.

5.7 Comparison to Previous Work

Table 2 compares LLUAD with the, to our knowledge, only conceptually directly comparable related work in [18], where applicable. The hit ratio of LLUAD is generally in line with the *Optimal TopList* from [18], which is not implementable in real-world scenarios as it is constructed from future user queries. Although the hit ratio of LLUAD is slightly higher than the *Optimal TopList* for medium-sized lists, the key takeaway is not the improvement itself, as much of it may be due to differences in the DNS dataset, but rather that LLUAD is comparable to an optimally instantiated `Popularity List`.

For a list of size 10 000, LLUAD incurs a similar network overhead for downloading the `Popularity List` without compression as [18] (LLUAD uses TLS, whereas [18] only uses TCP), suggesting comparable memory and storage requirements of the list, despite our `Popularity List` including a `Load-Balancing Pool`. The increased efficiency of our list is also shown by the comparably higher overhead with compression enabled. Although the `Load-Balancing Pool` adds some memory/storage overhead to the list, it plays an important role in LLUAD reducing the network overhead of incremental updates. The improvement in network overhead of our LLUAD scheme is approximately 95.5 % compared to [18] for $N_{popular}$ = 10 000 with compression, where we use a negligible 60 s minimum TTL. These results are especially significant considering only 39 queries per second was required in [18] to continuously refresh the TTLs of 10 000 records, compared to 188 today, indicating a significant decrease in average DNS record TTL and a potential increase in required incremental updates.

Table 2: Comparison between LLUAD and the results of [18] from 2011, where *(comp.)* indicates whether compression is used.

	$N_{popular}$	LLUAD	[18]
List mean hit ratio	100	39.2 %	40 %
	1000	72.6 %	63.9 %
	10 000	91.4 %	83.9 %
	25 000	94.4 %	
	100 000	96.2 %	94.5 %
Downloading the list	10 000	836.6 kB	850 kB
	10 000 (comp.)	426.1 kB	290 kB
Incremental updates	10 000	74.1 kB/h	2580 kB/h
	10 000 (comp.)	67.5 kB/h	1500 kB/h
	25 000 (comp.)	139.9 kB/h	

6 Conclusion

LLUAD using DoHoT achieves near zero-trust DNS privacy, guaranteeing a high level of DNS privacy despite large-scale collusion between anonymizing entities/relays. Moreover, the scheme greatly improves the mean latency of its paired DNS protocol, allowing DoHoT using Tor to be on par with traditional DNS over port 53. We also find that LLUAD does not incur any additional network overhead to the paired DNS protocol, only introducing a modest memory/storage overhead of less than 2 MB. LLUAD achieves this uncompromising combination of attributes, while remaining compatible with existing DNS infrastructure, by pre-fetching a local `Popularity List` of the most frequently resolved records, securely instantiated and continually updated based on anonymous user votes. LLUAD contributes a `Voting Mix Network` – a highly scalable self-sufficient mix network design, not requiring third-party hosted mix servers and achieving packet lengths independent of the number of traversed mix nodes: allowing users to anonymously and efficiently cast a centrally limited/enforced maximum number of votes. LLUAD also reduces the network bandwidth overhead required to incrementally update the `Popularity List` by over 95 % by contributing a `Load-Balancing Pool` system.

Acknowledgments

This work was supported by the Swedish Research Council (VR).

References

[1] Daniel Aleksandersen. 2019. *A survey of DNS caching and TTL in end-user client software.* Ctrl blog. https://www.ctrl.blog/entry/dns-client-ttl.html
[2] APNIC Labs 2024. *Encrypted DNS World Map.* APNIC Labs. https://stats.labs.apnic.net/edns?s=0
[3] Norbert Bißmeyer, Jonathan Petit, and Kpatcha M. Bayarou. 2013. Copra: Conditional pseudonym resolution algorithm in VANETs. In *IEEE/IFIP WONS.* 9–16.
[4] CircleID 2023. *Analysis of 7.5 Trillion DNS Queries Reveals Public Resolvers Dominate the Internet.* CircleID. https://circleid.com/posts/20230316-analysis-of-7.5-trillion-dns-queries-reveals-public-resolvers-dominate-the-internet
[5] Cloudflare 2024. *What is 1.1.1.1?* Cloudflare. https://www.cloudflare.com/learning/dns/what-is-dns/
[6] Cloudflare 2024. *What is a DNS zone?* Cloudflare. https://www.cloudflare.com/learning/dns/glossary/dns-zone/
[7] Cloudflare 2024. *What is DNS-based load balancing?* Cloudflare. https://www.cloudflare.com/learning/performance/what-is-dns-load-balancing/
[8] Carlo Contavalli, Wilmer van der Gaast, David C Lawrence, and Warren Kumari. 2016. Client Subnet in DNS Queries. RFC 7871. https://doi.org/10.17487/RFC7871

[9] George Danezis and Ian Goldberg. 2009. Sphinx: A compact and provably secure mix format. In *2009 30th IEEE Symposium on Security and Privacy*. IEEE, 269–282.

[10] Frank Denis. 2023. *Anonymized DNSCrypt specification*. GitHub. https://github.com/DNSCrypt/dnscrypt-protocol/blob/master/ANONYMIZED-DNSCRYPT.txt

[11] Frank Denis. 2024. *DNSCrypt/dnscrypt-proxy*. GitHub. https://github.com/DNSCrypt/dnscrypt-proxy

[12] Sara Dickinson, Benno Overeinder, Roland van Rijswijk-Deij, and Allison Mankin. 2020. Recommendations for DNS Privacy Service Operators. RFC 8932. https://doi.org/10.17487/RFC8932

[13] Roger Dingledine, Nick Mathewson, Paul F Syverson, et al. 2004. Tor: The second-generation onion router.. In *USENIX security symposium*, Vol. 4. 303–320.

[14] DNSCrypt 2024. *DNSCrypt version 2 protocol specification*. DNSCrypt. https://dnscrypt.info/protocol/

[15] DNSCrypt 2024. *List of public DoH and DNSCrypt servers*. DNSCrypt. https://dnscrypt.info/public-servers/

[16] Natnatee Dokmai, L. Jean Camp, and Ryan Henry. 2022. Assisted Private Information Retrieval. Cryptology ePrint Archive, Paper 2022/1082. https://eprint.iacr.org/2022/1082

[17] ericlaw. 2022. *Chromium's DNS Cache*. text/plain. https://textslashplain.com/2022/03/31/chromiums-dns-cache/

[18] Hannes Federrath, Karl-Peter Fuchs, Dominik Herrmann, and Christopher Piosecny. 2011. Privacy-Preserving DNS: Analysis of Broadcast, Range Queries and Mix-Based Protection Methods. In *ESORICS*. Berlin, Heidelberg, 665–683.

[19] Google 2024. *Public DNS | Google for Developers*. Google. https://developers.google.com/speed/public-dns/docs/using

[20] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. 2023. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 3889–3905. https://www.usenix.org/conference/usenixsecurity23/presentation/henzinger

[21] Dominik Herrmann, Christian Banse, and Hannes Federrath. 2013. Behavior-based tracking: Exploiting characteristic patterns in DNS traffic. *Computers & Security* 39 (2013), 17–33. https://doi.org/10.1016/j.cose.2013.03.012 27th IFIP International Information Security Conference.

[22] Dominik Herrmann, Max Maaß, and Hannes Federrath. 2014. Evaluating the Security of a DNS Query Obfuscation Scheme for Private Web Surfing. In *ICT Systems Security and Privacy Protection*, Nora Cuppens-Boulahia, Frédéric Cuppens, Sushil Jajodia, Anas Abou El Kalam, and Thierry Sans (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 205–219.

[23] Paul E. Hoffman and Patrick McManus. 2018. DNS Queries over HTTPS (DoH). RFC 8484. https://doi.org/10.17487/RFC8484

[24] Zi Hu, Liang Zhu, John Heidemann, Allison Mankin, Duane Wessels, and Paul E. Hoffman. 2016. Specification for DNS over Transport Layer Security (TLS). RFC 7858. https://doi.org/10.17487/RFC7858

[25] I2P 2024. *I2P Network Performance: Speed, Connections and Resource Management*. I2P. https://geti2p.net/en/about/performance

[26] Mohammad Khodaei, Hongyu Jin, and Panagiotis Papadimitratos. 2018. SECMACE: Scalable and Robust Identity and Credential Management Infrastructure in Vehicular Communication Systems. *IEEE Transactions on Intelligent Transportation Systems* 19, 5 (2018), 1430–1444.

[27] Eric Kinnear, Patrick McManus, Tommy Pauly, Tanya Verma, and Christopher A. Wood. 2022. Oblivious DNS over HTTPS. RFC 9230. https://doi.org/10.17487/RFC9230

[28] P. Mockapetris. 1987. Domain names - concepts and facilities. RFC 1034. https://doi.org/10.17487/RFC1034

[29] Ania M Piotrowska. 2020. *Low-latency mix networks for anonymous communication*. Ph. D. Dissertation. UCL (University College London).

[30] Scott Rose, Matt Larson, Dan Massey, Rob Austein, and Roy Arends. 2005. DNS Security Introduction and Requirements. RFC 4033. https://doi.org/10.17487/RFC4033

[31] Mahrud Sayrafi. 2018. *Introducing DNS Resolver for Tor*. Cloudflare. https://blog.cloudflare.com/welcome-hidden-resolver/

[32] Manmeet Singh, Maninder Singh, and Sanmeet Kaur. 2019. TI-2016 DNS dataset. https://doi.org/10.21227/9ync-vv09

[33] Philip Sjösvärd, Hongyu Jin, and Panos Papadimitratos. 2025. DNS in the Time of Curiosity: A Tale of Collaborative User Privacy Protection. Springer. Twenty-ninth International Workshop on Security Protocols, to appear..

[34] The Tor Project 2024. *Performance - Tor Metrics*. The Tor Project. https://metrics.torproject.org/onionperf-latencies.html

[35] David Ulevitch. 2011. *DNSCrypt – Critical, fundamental, and about time*. Cisco Umbrella. https://web.archive.org/web/20200701221715/https://umbrella.cisco.com/blog/dnscrypt-critical-fundamental-and-about-time

[36] WhoisXML 2024. *Domain & IP Data Intelligence for Greater Enterprise Security*. WhoisXML. https://www.whoisxmlapi.com/

[37] Fangming Zhao, Yoshiaki Hori, and Kouichi Sakurai. 2007. Analysis of Privacy Disclosure in DNS Query. In *2007 International Conference on Multimedia and Ubiquitous Engineering (MUE'07)*. 952–957. https://doi.org/10.1109/MUE.2007.84