Hide and Seek: Revisiting DNS-based User Tracking

Deliang Chang*§, Joann Qiongna Chen†§, Zhou Li†, and Xing Li*
*Tsinghua University, changdl16@mails.tsinghua.edu.cn, xing@cernet.ed.cn
†University of California, Irvine, joann.chen@uci.edu, zhou.li@uci.edu

***Abstract*—Domain name system (DNS) is the address book of the Internet and domain names are queried before almost every network activity. Since the entities like recursive resolvers can monitor users' DNS queries, privacy concerns such as user tracking arise. Though a number of prior works have looked into this issue, they all focus on the closed-world setting, which means that victim users must be known to the adversary. We argue that it does not reflect the adversary's true capabilities. Moreover, there lacks an effective approach to defend against DNS-based user tracking. In this work, we revisit these issues by investigating the attack surface in both open-world and closed-world settings and studying how to protect users. First, we introduce a new tracking mechanism DSCORR which incorporates domain-based word embedding to capture the fine-grained distance between domain names, and automatic threshold generation for fine-tuning the attack outcome. The evaluation result on a real-world DNS dataset shows DSCORR is able to outperform the existing works by a large margin especially in the open-world setting. On the defense side, we develop a system called LDPRESOLVE, which incorporates a recently proposed differential privacy notion ULDP (Utility-optimized Local Differential Privacy) and a new technique named parallel domain resolving, to provide privacy guarantees without damaging the utility of legitimate applications. The evaluation result on the same dataset shows the DNS-based user tracking can be effectively curbed, e.g., tracking accuracy degraded from 93% to 10.1%.**

***Index Terms*—Domain name system, User tracking, Differential Privacy**

1. Introduction

Domain name system (DNS) translates human-readable domain names to machine-readable IP addresses. It is an essential component of the Internet infrastructure as DNS queries underpin almost every user's Internet activity. Nowadays, *trillions* of users' requests on a single day are processed by DNS [20].

Under normal configuration, a user's DNS queries will be sent to a recursive resolver, which acts as an agent to obtain the authoritative answers from authoritative nameservers. The plaintext information of every query is visible to recursive resolvers, even when DNS encryption like DNS-over-HTTPS [47] or DNS-over-TLS [45] is used (the requests are only encrypted between users and recursive resolvers). Hence, a recursive resolver holding a large amount of DNS logs also poses privacy threats to users. One prominent threat is *user tracking* [80], through which a user's activities across different networks and devices can be correlated, for purposes like personalized advertisement, surveillance, etc. This threat is more acute nowadays as public resolvers handle the lion's share of users' DNS requests and are well motivated to launch user tracking [11], [21], [43], [46].

Understanding DNS-based User Tracking. We found some works have studied DNS-based tracking [40]–[42], [52], [53], [55], [85], but they all considered the *closed-world* setting, where all possible victim users must be known to the adversary. How tracking performs in the more practical *open-world* setting (i.e., the user sending DNS request may be unknown) is unclear yet.

To advance the understanding of DNS-based user tracking, we design a new tracking mechanism tailored to the open-world setting (called DSCORR). The problem of DNS-based tracking can be formulated as assigning a DNS session (or a sequence of DNS queries in a short period) to a user, which exploits the similarity of DNS behaviors from the same user. It is important to correctly compute the similarity or the distance between two different DNS sessions, but we found the previous works all used the coarse-grained distance between domains, i.e., the distance is either 1 (different domains) or 0 (identical domain), which neglects the contextual correlation between domains. This leads to sub-optimal performance, especially in the open-world setting. Our key insight is that a DNS session resembles a text paragraph to some extent (both are sequences built up from tokens with semantic meanings). Thus, techniques well established in the NLP community, in particular *word embedding* [60], [74], can be applied to capture the fine-grained distance between DNS sessions. In particular, we use the domains queried before and after a targeted domain as the context to mine its semantics. We apply Word2vec [66] to automatically convert each domain into a numerical vector, such that the relation between domains can be quantified. The computed distance is then compared to a threshold to determine if the two sessions belong to the same user. Existing works use the identical threshold for every user, which leads to high error rate on certain users. We develop a new technique named *auto-threshold* to learn a user-centric threshold to tackle this issue.

Our result shows DSCORR can identify a user from DNS sessions with high accuracy. In the open-world set-

§. Both are first author. Part of Deliang Chang's work was done when visiting University of California, Irvine.

ting, DSCORR outperforms the existing approach [40] by a large margin (0.87 AUC comparing to 0.657 AUC when 20 sessions per user are used for training). In the closed-world setting, DSCORR is able to cluster an unlabeled session with 52.6% accuracy when training set contains only 1 session per user. This accuracy rises to 87.4% when 10 sessions are labeled.

Mitigation with differentially private domain resolving. To prevent user tracking, a straightforward solution is to add random dummy queries to the original users' queries. Spreading users' queries across resolvers is another solution, which has been exercised in K-resolver [43]. However, we argue that the threat is not adequately addressed by these solutions, as they may introduce significant overhead at the client-side to achieve a certain level of privacy. For instance, DNS queries have to be spread out to a large number of resolvers to defend against tracking, as discussed in Section 5.6. A greater concern is that these solutions impair the utility of DNS data irreversibly, and such DNS data is critical to legitimate applications like malicious domain detection [96].

To address the tension between utility and privacy, we make the *first* attempt to integrate differential privacy [29], a method that controls utility loss under privacy guarantee, to the process of domain resolution. We term our defense LDPRESOLVE, which changes the behavior of a stub resolver under *Local Differential Privacy (LDP)* so recursive resolver does not need to be trusted. Yet applying LDP to our setting has to address two prominent challenges. First, take frequency estimation (the primary usage scenario of LDP) as an example, only Direct Encoding (DE) of LDP [89] can be chosen to avoid revamping DNS protocol, but it will incur very high utility loss. Second, under the default randomized response protocol, the user has to issue false DNS queries, but doing so will give the user the wrong DNS response and break every Internet application.

To tackle these challenges, we propose a novel *$(\mathcal{X}_\mathcal{S}, \mathcal{Y}_\mathcal{P}, \epsilon_1, \epsilon_2)$-ULDP (Utility-optimized LDP)* protocol, which is adjusted from ULDP [67], as the base of our defense. Our key insight is that though both user tracking and legitimate applications inspect domain names, different domain names have different levels of importance to them (e.g., popular domain names are important to user tracking but less so for malicious domain detection). $(\mathcal{X}_\mathcal{S}, \mathcal{Y}_\mathcal{P}, \epsilon_1, \epsilon_2)$-ULDP allows us to assign different privacy budgets for DNS queries. Under $(\mathcal{X}_\mathcal{S}, \mathcal{Y}_\mathcal{P}, \epsilon_1, \epsilon_2)$-ULDP protocol, we adapt *parallel domain resolving* to address the issue of false DNS queries. Hence, users always obtain accurate responses.

Based on the evaluation, we found LDPRESOLVE can achieve the desired outcome: the tracking accuracy can be significantly reduced while the utility loss is controlled to a certain level. To highlight, the tracking accuracy of DSCORR can be degraded from 93% to 10.1% while the utility loss measured by the standard deviation of unpopular domains is less than 10.

Contributions. Our contributions are summarized below:

- We propose a new tracking method DSCORR that works under both open-world and closed-world settings. With the help of domain embedding, the tracking accuracy is significantly improved.
- To DNS-based user tracking, we make the first attempt to integrate LDP into DNS. Based on a novel LDP notion, $(\mathcal{X}_\mathcal{S}, \mathcal{Y}_\mathcal{P}, \epsilon_1, \epsilon_2)$-ULDP, and parallel domain resolving, we show it is feasible to solve the dilemma of DNS data utility and privacy.
- We evaluate DSCORR and LDPRESOLVE on a real-world DNS dataset and report our discoveries. A client-side prototype of LDPRESOLVE is also developed.
- The code of DSCORR and LDPRESOLVE is publicly released[1].

2. Background

2.1. DNS Communications and Dataset

DNS (Domain Name System) queries are issued before most network activities to map a user-friendly domain name (e.g., `www.google.com`) to an IP address (e.g., `216.58.193.196`). In particular, a user-end software named *stub resolver* receives the user's requests produced by other applications and forwards them to a *recursive resolver* if the responses are not cached. The recursive resolver can be an Internet Service Provider (ISP) resolver serving users within the same network or a public resolver (e.g., Google Public DNS [2]) serving users all over the Internet. It further forwards the requests to *authoritative nameservers*, which is organized in a hierarchical structure and provides authoritative answers to the iterative queries.

Users' DNS traffic between their stub resolvers and authoritative nameservers results in a wealth of information valuable to applications like malicious domain detection [19], [26], [63], [93] and Internet traffic estimation [36]. Several organizations are gathering those telemetry data and sharing it with other parties under the concept of Passive DNS, mainly through two approaches. The first is to place a sensor array between recursive resolvers and authoritative nameservers, such that only the DNS lookups resulting in cache miss on recursive resolver are captured, and the client IP addresses are not seen. The Security Information Exchange (SIE) of FarSight [12] is operated under this model. As clients' requests are aggregated by each recursive resolver, a prior study suggests users' privacy is not violated when the sensors are configured properly [84]. The second approach is to fetch raw DNS logs *directly from recursive resolvers* and share the logs with client IP anonymized [61]. DNS Pai Project maintained by Qihoo 360 is operated under this mode [10]. While it enables more powerful applications, like finding abnormal domain associations [61], it could also raise privacy issues like user tracking. In this work, we thoroughly study such risks and propose a new approach to protect the end-users.

2.2. DNS-based User Tracking

We define user tracking as linking users' network activities across different networks without their consent, for purposes like personalizing advertisements or surveillance. When the network activities are DNS communications, user tracking can be done by linking raw DNS logs

1. https://github.com/dl-chang/ldpresolve

collected and shared by the recursive resolvers. Though such tracking is trivial when a user uses a static IP address, many ISPs assign dynamic IP addresses that change periodically to their customers, so the adversary needs to re-identify the user after his/her IP address is changed. Also, the user could move between different ISP networks. Yet, a large number of users can be impacted due to the consolidation of DNS resolvers and the increasing dependency on the public resolvers [77], which can be queried wherever the users are. Here we highlight a few scenarios that this attack is relevant to: 1) the users use a public resolver configured by their ISP (e.g., campus IT without DNS infrastructure chooses google public DNS); 2) the users' browsers set a default public resolver so all DNS traffic about the users is collected (e.g., DNS-over-HTTPS resolver in Google Chrome [3]); 3) an ISP exchanges DNS data from its resolver with another ISP; 4) a company retrieves DNS logs from multiple resolvers, e.g., DNS Pai mentioned in Section 2.1.

Some research has been done on DNS-based user tracking. Herrmann et al. utilized classification methods such as Bayesian classifier and k-nearest neighbor to identify the re-occurrence of users based on their DNS queries [40]. In [42], [55], the authors used a modified k-means algorithm to cluster DNS logs of the same user. DNSMiner [52], [53] re-identified the user by extracting unique and repetitive fingerprints from DNS traces. Sun et al., proposed a method called constrained Dirichlet multinomial mixture for clustering DNS sessions without knowing the number of users in advance [85].

In our paper, we choose the methods from [40] as the baseline to compare against. The other methods are not chosen because they cannot be easily adjusted in our setting, e.g., open-world setting as described in Section 3.1, due to their assumptions on the DNS data. 1) the methods in [52], [53] required a DNS session (to be explained in Section 3.1) is fixed to 24 hours. 2) the methods in [42], [55], [85] need to know the number of users or the maximum number of DNS sessions for a single user.

Other types of tracking. Most of the other works in tracking looked into *web-based (or browser-based)* tracking. In its basic form, web code like JavaScript from a third-party content provider attempts to link user's visits based on tagging [14], [18], [56] or fingerprinting [30], [31], [57]. Different from web-based tracking, *network-based tracking* looks into the characteristics of network flows and maps them to the user. For example, Kumpost et al. [58] built user profile based on HTTP/HTTPS/SSH traffic. Verde et al. [87] leveraged characteristics of Netflow to track users behind the NAT. Previous works showed that machine-learning approaches using packet sizes and intervals can correlate flows of the same user, even when the network flows are encrypted and mixed under Tor [68]. A related attack is website fingerprinting, which identifies the website visited by a targeted user from the encrypted traffic with network features similar to flow correlation. Protocols like HTTPS [58], DNS-over-TLS [44] and DNS-over-HTTPS [83] are found vulnerable under this attack.

A few other works also use DNS to mine users' characteristics under very different settings. DNS cache-based tracking [56] uses JavaScript code to query domains and exploits the client-side DNS cache to tag a user, while we passively analyze the DNS dataset. DefecTor [35] exploits DNS packets for website fingerprinting, while ours focuses on re-identifying users. Our work also differs from [48], which reveals sensitive queries by an institution using the logged DNS queries between recursive resolvers and authoritative name servers, while we use DNS queries between stub resolvers and recursive resolvers.

2.3. Differential Privacy

In this work, we leverage mechanisms under differential privacy (DP) to protect users from DNS-based user tracking. Given a query from a data consumer [29], the original idea of DP assumes there is a trusted data curator adding noises to the result under a DP notion (also called Central DP). Different from the central setting, local DP (LDP) assumes there is no trusted data curator, and the noises are added by the data providers (e.g., Internet users) before the data are collected by the curator. The user enjoys better privacy as the data curator needs not be trusted, but the data utility is often worse than central DP under the same privacy budget.

Definition 1 (ϵ-Local Differential Privacy [89]). *An algorithm $\mathcal{A}$ satisfies ϵ-local differential privacy (ϵ-LDP), where $\epsilon > 0$, if and only if for any pair of input x_1 and x_2, we have*

$$\forall y \in \mathrm{Range}(\mathcal{A}) : \frac{\Pr[\mathcal{A}(x_1) = y]}{\Pr[\mathcal{A}(x_2) = y]} \leq e^{\epsilon} \qquad (1)$$

where $\mathrm{Range}(\mathcal{A})$ denotes the set of all possible output results of an algorithm $\mathcal{A}$.

LDP has seen strong adoption from the industry. Companies like Google [32], Apple [8] and Samsung [69] have developed their own LDP implementation to compute aggregated user statistics in a privacy-preserving way. The data collection protocol under LDP consists of three steps [89]: **Encode** (users report their answers in a specific format), **Perturb** (the answers are randomized), and **Aggregate** (the answers are merged and decoded to obtain statistics, e.g., item frequency). In our work, we focus on the LDP protocols that support frequency estimation (e.g., Google's RAPPOR [32]).

3. DNS Session Correlation with Domain Embedding

In this section, we first present the threat model. Then, we propose a tracking method (named DSCorr) tailored to DNS session correlation based on domain embedding. Last, we evaluate the effectiveness of DSCorr.

3.1. Threat Model

Section 2.2 has described the attacker's goal. Here we focus on the attack constraints. We assume though the attacker has access to raw DNS logs, he/she cannot access DHCP logs, so a user's dynamic IP addresses cannot be easily linked to the user's ID. Following the setting in existing works about DNS-based tracking [40], [42], [52], [53], [55], [85], we do not try to track users behind a NAT. Under NAT, one source IP address (or its anonymized

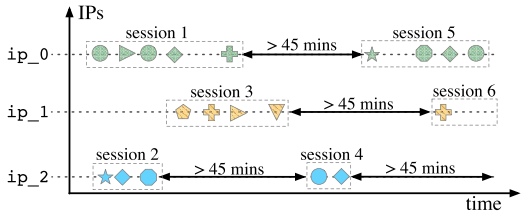


Figure 1: An example of sessions constructed from DNS queries. Different colors represent different source IP addresses, and different shapes represent different queried domain names.

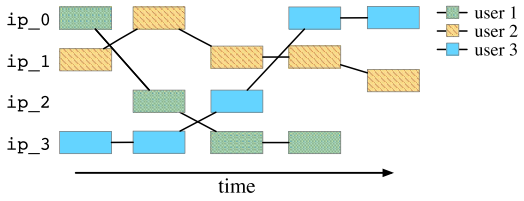


Figure 2: The objective of the adversary, who aims to correctly link a DNS session to its requesting user, regardless of IP churns.

version) can be associated with multiple users, which makes correct user profiling, linking, and labeling (for the evaluation purpose) very challenging.

To defeat DNS-based tracking, a user can use privacy-preserving DNS resolution techniques like Adaptive DNS [54] or Oblivious DNS [81]. However, these techniques are still at the early adoption stage and we assume they are not used by the victims in our setting. Notice that DNS encryption mechanisms like DNS-over-TLS and DNS-over-HTTPS do not prevent our adversary, as DNS packets are decrypted by the resolver. Ironically, those mechanisms might strengthen the adversary's capability in user tracking since most of the resolvers supporting these protocols are centralized [11], [21], [43], [46].

DNS session. Like previous works [40], [42], [52], [53], [55], [85], we assume the attacker constructs DNS sessions from the raw DNS logs and performs data linking at the session level. Here we give the definition of a DNS session and explain how it is constructed in our setting.

Without considering multiple users sharing the same IP address at the same time (e.g., under NAT), we define a DNS session as a sequence of DNS queries associated with a source IP address in a period issued by the same user. A new session is created for a DNS query and its following queries if the previous DNS query is observed at least 45 minutes ago. According to our empirical analysis, a device usually keeps sending DNS requests as long as it stays online. A different device may use the IP address only if the address has not been used for a period of time (usually longer than 45 minutes). Therefore, we use the aforementioned rule to split the DNS queries. Figure 1 illustrates how sessions are created from a stream of queries. Noticeably, our session does not have to contain all DNS queries from a source IP, and its covered time period is variable. Two consecutive sessions could belong to the same user. This setting is more flexible than the previous works setting a session to be 24 hours [52], [53], and it avoids the errors caused by IP churns across users. After session construction, user tracking is performed and Figure 2 gives an illustrative explanation.

Open-world and closed-world settings. Though the high-level goal of this work and previous works is the same, the previous works all assume *closed-world setting* [23], [40], [42], [52], [53], [55], [85], while ours assumes *open-world setting*. The two settings are well defined by the literature of website fingerprinting, and we use the definition proposed by Wang et al. [88]: in the closed-world setting, the users are assumed to only visit a set of web pages known to the attacker, and the attacker has to detect which ones are visited; in the open-world setting, the user is free to visit any pages, and the attacker also needs to detect the pages not included in the set. Here, we adjust the two settings to DNS-based tracking: in the closed-world setting, a DNS session comes from users in a set known to the attacker, and the attacker has to detect the user that issues the session; in the open-world setting, a DNS session can be from any user, and the attacker needs to tell even when the user is not in the set. Clearly, the open-world setting is more realistic yet challenging. As pointed out by Wang et al. [88], classifying pages of *low base rate* is error-prone. Users of low base rate (i.e., the users who only issue a few sessions) also exist here.

3.2. Ethics

We obtain a DNS dataset from the IT department of a campus, which manages the campus DNS resolvers. The source IPs are *hashed before* the dataset is given to us. As the campus also runs a DHCP server, the user ID behind each source IP is known to the IT department. To help us build the ground truth, the IT department also hashes the user IDs and provides us the mapping between them to source IPs. Noticeably, the hashed user IDs are only used to verify the effectiveness of DSCORR, and they are not used as the input to DSCORR.

Usage of the dataset for research purposes is authorized by the administration office of the campus. The raw data and all intermediate data are processed and stored in a server installed on the same campus. Only the authors of this paper have access to this data. Similar DNS datasets have been used by other works [36], [61], [73], and the treatment of data is similar.

Therefore, we are publishing the code of DSCORR and LDPRESOLVE, but we are not able to share the data as it ties to individual identities.

3.3. Domain Embedding

To determine whether a pair of sessions are close enough, i.e., belong to the same user, previous works treat each session as a vector of domains and compute the distance between them. The distance between a pair of domains is either 1 (different) or 0 (identical), but fine-grained distance is necessary and derivable, by considering what content they deliver and how they are connected. For example, domains of the same website category, like `cnn.com` and `cbsnews.com` under news, should have high affinity; `cnn.com` always loads advertisements from `turner.com`, so they are closely related. While the fine-grained domain distance can be manually characterized,

such an approach is not scalable. To this end, we develop a new technique under *domain embedding* to automatically infer the domain distance from the DNS data.

Our domain embedding is adapted from *word embedding*, a technique extensively used by NLP community [60], [74]. It can construct a vector for a word based on the surrounding words occurring in the same text, preserving its semantic similarity. Similar to the distribution of words, users' visits to domains are subject to the power law, according to previous works [22], [51], making embedding appropriate for domains. When we regard a domain as a word and consecutive queries in a DNS session as its context, the distance between domains can be captured similarly. Though there are a few works applying word embedding on domain names, the goals and approaches are different. Jiang et al. [50] embeds each character in URL/Domain while ours embeds an entire domain. Lopez et al. [64] finds similar domain names while ours links DNS sessions.

To generate domain embeddings, we follow the approach of Word2vec [66]. Word2vec builds a neural network with one hidden layer and uses the weights of the hidden layer as the embedding after training on a corpus of text, e.g., Wikipedia. For our case, we generate embeddings of domains on a large number of DNS queries, which are represented under the skip-gram model [66]. Those queries do not need to be labeled. After the embeddings are generated, a domain can be mapped to a vector, based on its preceding and subsequent domains queried from the same IP address. Our Word2vec-based method has achieved satisfactory results, but we will also test the newer methods like Glove [74] and Bert [28] in the future.

3.4. Design of DSCorr

Terms. We first define the terms used in DSCorr. We assume a DNS session is s and a domain is d. Like previous works [41], [42], we extract the sequence of domains requested by s, so s can be represented as $< d_1, ..., d_j, ..., d_m >$, where d_j is the d of the jth request and m is the number of requests in s. The user u behind a session s has either been labeled by the adversary before the tracking attack or is unknown. We term the sessions already labeled as **SL**, and the sessions unlabeled as **SU**. In the closed-world setting, the users behind **SL** and **SU** are identical. In the open-world setting, they might not be identical.

Workflow of DSCorr. The design of DSCorr is based on semi-supervised learning to handle the open-world setting. At a high level, DSCorr leverages **SL** as the training dataset and creates *session clusters* grouped by the labeled user IDs. Then, an s in **SU** will be assigned to a session cluster in **SL**, or classified as "unknown". Four steps are carried out.

1) We apply domain embedding described in Section 3.3 to convert each d to a numerical vector.
2) We group sessions in **SL** to create labeled session clusters and build profiles for them. A cluster consists of sessions of the same user.
3) Given an s in **SU**, the k nearest session clusters are identified through a data-sketching process.
4) Fine-grained distances between s to all neighboring session clusters are computed. s will be clustered to a session cluster in **SL** with the minimum distance in the closed-world setting. For the open-world setting, s might be classified as "unknown" if it is far from any session clusters.

Session clustering. Note that Step 1 directly applies Word2Vec, thus we focus on Step 2-4. First, we introduce the method to measure the distance between two DNS sessions, s_m and s_n. Since a session has a similar structure as a word vector after embedding, we are motivated to adopt the similarity metrics in the NLP domain. In particular, we utilize Text Semantic Similarity (TSS) [65] to compute the distance because it can model the context of words.

$$\begin{aligned} sim_{ss}(s_m, s_n) =& 1 - \frac{1}{2}\Big(\frac{\sum_{d_i \in s_m} \mathrm{tfidf}(d_i, s_m) \cdot \mathrm{sim_{ds}}(\mathrm{d_i}, \mathrm{s_n})}{\sum_{d_i \in s_m} \mathrm{tfidf}(d_i, s_m)} \\ &+ \frac{\sum_{d_j \in s_n} \mathrm{tfidf}(d_j, s_n) \cdot \mathrm{sim_{ds}}(d_j, s_m)}{\sum_{d_j \in s_n} \mathrm{tfidf}(d_j, s_n)}\Big) \\ dist_{ss}(s_m, s_n) =& 1 - sim_{ss}(s_m, s_n) \end{aligned} \tag{2}$$

where d_i and d_j are domains after the embedding of s_m and s_n respectively.

Note that we replace the idf (inverse document frequency) function used by TSS [65] with tf-idf (term frequency–inverse document frequency)[2]. This is because tf-idf works better when the similarity is computed on word embeddings [92]. In our setting, $\mathrm{tfidf}(d_i, s_m)$ reflects the importance of a domain d_i to the sequence enclosing it.

The similarity between a domain and a session, $\mathrm{sim_{ds}}(d_i, s_n)$, is defined as the maximum similarity:

$$\mathrm{sim_{ds}}(d_i, s_n) = max\{\mathrm{sim_{dd}}(d_i, d_j) | d_j \in s_n\} \tag{3}$$

The similarity between two domains $\mathrm{sim_{dd}}(d_i, d_j)$ is defined by the Cosine similarity[3] over their embeddings (each embedding has 200 dimensions). We choose distance based on Cosine similarity over other distances, such as Word Mover's Distance, because of its low computation overhead. With the distance between sessions defined, we now describe the details of our tracking method.

In Step 2, we generate a profile for each cluster in **SL** to enable fast comparison. Specifically, we merge all sessions of a certain cluster S to get a summary of sessions s_0. In Step 3, for an s in **SU**, we measure its distance to every S in **SU** by $dist_{ss}(s, s_0)$. The k clusters with the smallest distances are chosen, and the sessions within each cluster are considered as candidates.

In Step 4, we compute the fine-grained distance between s and candidate sessions from k neighboring clusters. For the closed-world setting, from all candidate sessions, the one with the smallest $dist_{ss}$ is chosen and its associated user identifier is used to label s.

2. $\mathrm{tfidf}(d_i, s_m) = c_{i,m} \cdot \log(|\mathbf{SL}|/|\{d_i \in s : s \in \mathbf{SL}\}|)$, where $c_{i,m}$ is the count of d_i in the session s_m. $dist_{ss}$ will remain the same whether we choose domain count or domain frequency here. $|\{d_i \in s : s \in \mathbf{SL}\}|$ is the number of sessions which contain domain d_i in S_L.

3. Cosine similarity of two vectors $d_i = (x_1, x_2, ...x_n)$ and $d_j = (y_1, y_2, ...y_n)$ is given by $\mathrm{sim_{dd}}(d_i, d_j) = \sum_{i=1}^{n} x_i y_i / \sqrt{\sum_{i=1}^{n} {x_i}^2 \sum_{j=1}^{n} {y_j}^2}$

Adjustment for the open-world setting. A session $s_{new} \in \mathbf{SU}$ might come from a user outside of **SL**. In Step 4, we change how the session-cluster distance ($dist_{sc}(s_{new}, S)$) is computed: instead of the smallest $dist_{ss}$, we use the pth percentile of the ordered $\{dist_{ss}(s_{new}, s_i)|s_i \in S\}$ (p is set to 25 empirically). We use p to model a user's varying patterns.

After the nearest S (or S_{near}) is discovered based on $dist_{sc}(s_{new}, S)$, we compare the nearest distance to a threshold t. If it is greater than t, the unlabeled session is classified as "unknown", meaning it does not belong to any user in **SL**. Otherwise, s is clustered to S_{near}. Note that algorithm in the close-world setting is a special case when $p = 0$ and $t = 1$.

As for the threshold, a fixed t can be set for every user, but false positives are introduced when users exhibit different browsing behaviors. To address this issue, we propose an *auto-threshold* technique to generate threshold t_i for every session cluster to be inspected. The key insight is that the threshold can be *learnt* from sampling the sessions of a user. For each session $s_i \in S$, we define $\{s_i\}^{\complement} = S \setminus \{s_i\}$. Then we construct inner-distance set of a cluster with $I = \{dist_{sc}(s_i, \{s_i\}^{\complement})|s_i \in S\}$. Given another parameter θ, the threshold t of a session cluster could be decided by sampling this inner-distance set:

$$t = \begin{cases} quantile(I, \theta) & 0 \leq \theta \leq 1 \\ \theta \cdot max(I) + (1-\theta) \cdot min(I) & otherwise \end{cases} \tag{4}$$

3.5. Evaluation of DSCORR

Dataset. We extract anonymized DNS query logs from a campus resolver as our evaluation dataset, as described in Section 3.2, which contain information such as hashed IPs, timestamps, domain names, qtypes, etc. We split the data into two parts, one for domain embedding, and another for tracking evaluation. For domain embedding, we use the DNS queries from 61,870 distinct IP addresses of 351 different class C subnets which belong to the campus network. For a queried domain name, we use the domain names of its neighboring queries from the same IP as the context to generate an embedding. In total, there are 163,762 domain embeddings generated. For tracking evaluation, the sessions of different users are differentiated using the anonymized DHCP logs (contains hashed MAC addresses, hashed IPs, timestamps, etc.). Consecutive sessions of the same user are merged. We argue that our labeling and data processing have limitations yet are reasonable. For example, session segmentation may be inaccurate, user DNS queries may be partially missing from the shared data, and users may change MAC addresses. However, the impact of these issues is expected to be limited. We use 30,716 DNS sessions collected from 1,000 different users in a two-week period to evaluate DSCORR in the open-world and the closed-world settings. The average, min, and max DNS sessions per user are 30.7, 12, 90 respectively. For a DNS session, the longest one covers 96.3 hours, while the shortest one only issued 1 query. The median duration is 2.74 hours. The campus wireless network assigns an IP for each device (though it is periodically changed based on DHCP), so NAT/VPN egress endpoint is not expected.

Experiment settings. As we use Word2vec to generate domain embeddings, its parameters are configured according to our DNS data: the sub-sampling rate is set to $1e-5$, the number of negative samples is 5, and at most 10 domains before and after the domain to be embedded are considered as the context.

For the open-world setting, we build **SL** with sessions from 500 users and **SU** with sessions from 1,000 users. We range the number of labeled sessions for each user (termed S_{SL}) from 2 to 20. The remaining sessions are considered as unlabeled sessions whose user IDs are to be inferred. In each run, we ensure the numbers of the test sessions belonging to known and unknown users are the same. We also compare DSCORR with the existing works that target the closed-world setting with 1,000 users in both **SL** and **SU**, and run the tracking experiment with different S_{SL}.

We implemented 3 models developed by the previous work [40] as the baseline, including 1NN-Jaccard, 1NN-Cosine, and Bayesian classifier. 1NN-Cosine and 1NN-Jaccard utilize Cosine Distance and Jaccard Distance respectively to calculate session-to-session distance, and then choose the nearest neighbor as the session owner. Bayesian classifier generates conditional probability distribution based on labeled sessions for each user, and then uses the maximum posterior probability estimate to infer user ID. Both the uni-gram version, named **jac**, **cos**, **bay** (1NN-Jaccard, 1NN-Cosine and Bayesian Classifier) and the bi-gram version (a pair of domains is considered as the basic unit), named **ja-bi**, **co-bi**, **ba-bi**, are tested.

Evaluation metrics. We term a session that belongs to a known user in **SL** as "known session", and otherwise as "unknown session". In our setting, a correctly identified known session is treated as a true positive (TP); a correctly identified unknown session is treated as a true negative (TN); a known session that is identified as not belonging to any known users is treated as false negative (FN). For false positives, there are two types: a known session identified as a wrong owner (FP_1), and an unknown session identified as belonging to a known user (FP_2). With this definition, four metrics are used in our experiment: true-positive rate (TPR, or recall), false-positive rate (FPR), precision (PPV, or positive predictive value), and accuracy (ACC). They are defined as follows.

$$\begin{aligned} TPR &= \frac{TP}{TP + FN} \\ FPR &= \frac{FP_1 + FP_2}{FP_1 + FP_2 + TN} \\ PPV &= \frac{TP}{TP + FP_1 + FP_2} \\ ACC &= \frac{TP + TN}{TP + TN + FP_1 + FP_2 + FN} \end{aligned} \tag{5}$$

Open-world setting. In this setting, the users in **SU** compose the superset of users in **SL**. We vary the labeled sessions for each user in **SL** (S_{SL}) from 2 to 20 (2, 5, 10, 20). According to our experiment in the closed-world setting (described later), ba-bi achieves the second-best result (the best result is achieved by DSCORR) when S_{SL} is set to 10, so we use it as the baseline to compare against. To accommodate the open-world setting, when the likelihood computed by ba-bi is lowered than a threshold

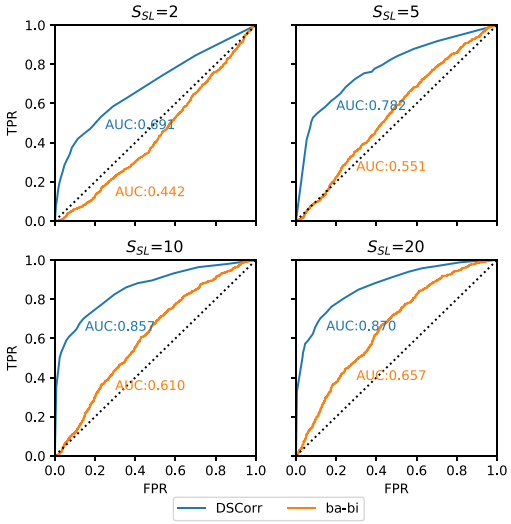


Figure 3: ROC of different methods in the open-world setting. S_{SL} is number of labeled sessions for each user in **SL**. AUC is Area under the curve of ROC.

t, it is considered as "unknown". Figure 3 shows the ROC curve of DSCORR and ba-bi with different θ and t under different size of S_{SL}. As expected, when S_{SL} increases, the attacker has more data to build a profile for a user, and the effectiveness of both tracking methods improves. Even though false positive rate is low when size of **SL** is small, DSCORR outperforms the baseline approach *at every* S_{SL} *by a big margin*, especially when S_{SL} is small (e.g., $S_{SL} = 2$): the difference between AUCs is consistently larger than 0.2 (0.691 vs. 0.442 for ba-bi[4] when $S_{SL} = 2$ and 0.87 vs. 0.657 when $S_{SL} = 20$). When a proper θ is chosen, DSCORR achieves the best overall accuracy, with 74.0% known sessions being classified correctly and 84.8% unknown sessions being labeled as unknown. The result shows the two key techniques of DSCORR, domain embedding, and auto-threshold, make a prominent contribution to the effectiveness of tracking.

Tracking effectiveness per user. Through our empirical analysis, we find that not all users are equally vulnerable under DNS-based tracking. Some users' sessions are easier to be identified (high recall) than others, or they are harder to be confused with others (high precision).

Here we change the measurement from all users to individual users and re-run the experiment of DSCORR. The heatmap of the detection precision and recall is shown in Figure 4.

The result suggests high recall rate is seen in the majority of users. Actually, about 64% users have a recall rate greater than 0.9 and 41% users have a recall rate greater than 0.99. There are 5.6% users in our dataset having precision and recall rates both greater than 0.99, meaning their internet activities are clearly distinguishable. After manually checking the DNS logs, we speculate that most of them are from Android devices while the rest are from PC/iOS devices or IoT devices. The domain names requested by them reflect the combination of applications

4. ba-bi performs worse than 0.5 because the goal is not just to find out whether a session is "known" or "unknown", but also its associated user, which is not a two-class classification problem.

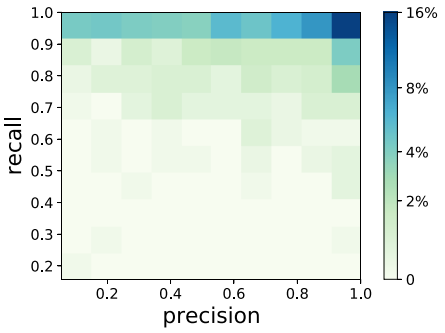


Figure 4: Heatmap of the precision/recall per user. The bar on the right shows the ratio of users.

#	jac	cos	bay	ja-bi	co-bi	ba-bi	DSCORR
1	42.2	40.7	37.4	45.4	40.1	36.5	**52.6**
2	56.0	52.8	54.8	59.2	52.8	54.3	**67.5**
3	67.2	60.3	65.7	67.2	60.3	65.8	**74.4**
5	74.8	69.3	76.3	74.8	69.3	76.8	**80.5**
10	78.8	78.0	86.2	82.7	77.6	87.3	**87.4**

TABLE 1: Comparison of tracking accuracy under the closed-world setting. # is the number of sessions in **SL** for each user.

running on their ends, which are not common among users.

Closed-world setting. Since the existing works focus on the close-world setting, we also run DSCORR in this setting for comparison. Here, the users in **SU** and **SL** are the same. Table 1 shows the comparison of tracking accuracy in both uni-gram (jac, cos and bay) and bi-gram (ja-bi, co-bi and ba-bi) versions. We vary S_{SL}, the number of sessions in **SL** for each user, from 1 to 10. We do not test $S_{SL} = 20$ because only a small subset of users has associated sessions of more than 20. The result shows that DSCORR works best in all settings. When S_{SL} is large, DSCORR and ba-bi have similar performances, while DSCORR performs much better when there are only a few labeled sessions (from 1-5) per user. For example, when there is only 1 labeled session, the accuracy of DSCORR is 52.6%, while the runner-up achieves 45.4% (ja-bi). When there are 10 labeled sessions, DSCORR achieves 87.4% accuracy.

Domain importance. For DSCORR and other methods, the similarity between sessions largely depends on the proportion of domains shared among them. We are interested in the properties of such domains, in particular, whether they are popular among users. To this end, for each session s_i, we compare it to the next session s_{i+1} of the same user, and categorize the domains appearing in both sessions as *shared* and the domains in s_i only as *unique*. Figure 5 shows the distribution of shared and unique domains in terms of session-wise popularity, defined by the number of sessions in our dataset the domain appears in. It turns out shared domains are more likely to be popular: 62.7% shared domains are popular, while the number is 35.4% for unique domains. Domains ranked after 10k have a 97.6% chance of being unique.

The result indicates *popular domain names have higher importance for tracking*, which may be counter-

intuitive. We speculate the main reason is that the combination of popular domains is more powerful in linking sessions of the same user. Our observation is also echoed by Kim et al. [52] that a large number of users cannot be fingerprinted by unique domains. Noticeably, DSCORR did not track users with user-specific domains (e.g., "ephemeral" or "disposable" domains) [15], [25], and we leave the integration of this method as future work.

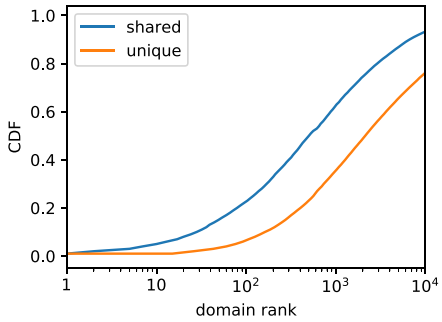


Figure 5: CDF of "shared" and "unique" domain for top 10k domains. Domains are ranked by session-wise popularity.

4. Domain Resolution under LDP

In this section, we propose a new privacy-preserving domain resolution method to protect the user against DSCORR and other tracking methods, while avoiding drastic changes to the client-side and server-side DNS infrastructure. We first overview the design of our proposed defense, LDPRESOLVE, and its motivation. Then, we elaborate how DNS resolution can be adapted to LDP so that the privacy leakage is bounded. In the next section, we evaluate the effectiveness of LDPRESOLVE.

4.1. Overview of LDPRESOLVE

At the high level, like other works against traffic fingerprinting [16], LDPRESOLVE adds noise to DNS queries from users' end. We expect users are willing to do so because they would like the DNS resolvers to serve them, and at the same time protect their privacy as much as possible. But in addition to privacy, we also consider the utility of DNS data. The main reason for this consideration is that DNS data is fundamental for legitimate third-party applications, therefore a defense ignoring the data utility is unlikely to be supported by the DNS service providers. This is a similar situation as crowdsourcing statistics [32], where Internet companies need to collect data from users at large while abiding by privacy laws. Legislation like Do Not Track Act [5] is directly related to DNS-based user tracking, thus we envision our proposal could also help companies avoid legal issues while using DNS data. We found Local Differential Privacy (LDP) has the potential to maintain both data utility and user privacy at the same time. Therefore, we develop LDPRESOLVE around LDP.

For the legitimate applications to be modeled under LDPRESOLVE, we focus on malicious domain detection, which extensively leverages DNS data [96]. After literature survey, we found the frequency of domain visits is an important detection feature used by many research works [19], [26], [27], [34], [59], [63], [78], [93], including the work published 2 years ago [86]. Hence, we select the LDP protocols used for frequency estimation [89]. We acknowledge that frequency is not the only feature useful for malicious domain detection. In Section 7, we also describe other features related to this topic.

Challenges. Applying LDP to our setting has to overcome two prominent challenges. First, though $\mathbf{Encode}$ of LDP (see Section 2.3) have different options (i.e., 5 options are listed in [89]), *Direct Encoding (DE)* is the only practical option for encoding DNS queries, as all other methods have to change the format of a request[5], which are unlikely to be supported by DNS stakeholders. On the other hand, DE often results in much higher utility loss than other methods when the cardinality of the answer set is large [89]. For example, the variance of the estimation under DE can be *two orders of magnitude* larger than other methods, when the privacy budget is tight and the answer set cardinality is high (see Table 2 of [89]). In our setting, users can resolve any domain among **billions** of the registered domains, resulting in an unbearable error margin potentially.

Second, randomized response [90] is the default communication protocol under DE, in which a user gives a false answer to the data curator at a certain probability. In our setting, randomized response means replacing the domain to be queried (e.g., `google.com`) with another domain drawn from a list (e.g., `yahoo.com`). Though it is expected to reduce tracking accuracy, because the user's DNS behaviors are altered, all Internet applications relying on DNS are likely to be broken.

Solutions. For the first challenge, it can be addressed through extending a recently proposed LDP concept, *Utility-optimized LDP (ULDP)* [67]. The key insight behind ULDP is that not all data are equally sensitive (e.g., the answer "Yes" is more sensitive to the question "Have you ever cheated in an exam?" than the answer "No" [67]), and the non-sensitive data output can be protected in a lesser way (i.e., adding less noise). Overall, ULDP provides much better utility when non-sensitive data are dominant. We extend ULDP to enable two-level privacy protection. For applications like malicious domain detection, popular domains like `google.com` are usually not scrutinized because they are quite unlikely to be malicious [19], [34], [71]. On the other hand, they are "anchors" to user tracking, connecting sessions of the same user, as examined in Section 3.5 ("Domain Importance"). As such, we treat the popular domains as sensitive data, adding higher-degree noise to them under ULDP. The remaining unpopular domains are treated as non-sensitive data and be processed with lower-degree noise to keep good data utility.

The second challenge can be addressed by *parallel domain resolving*: given a domain name d to query, a user can send d to an alternative resolver (termed AltRR) when the risk of user tracking is high, and send a dummy query d' to the primary resolver. The idea of dispersing DNS

5. For example, assuming v is the answer provided by a user, when Unary Encoding (UE) [89] is used, $\mathbf{Encode}(v) = [0, ..., 0, 1, 0, ..., 0]$, a binary vector where only v-th position is 1. When DE is used, $\mathbf{Encode}(v) = v$.

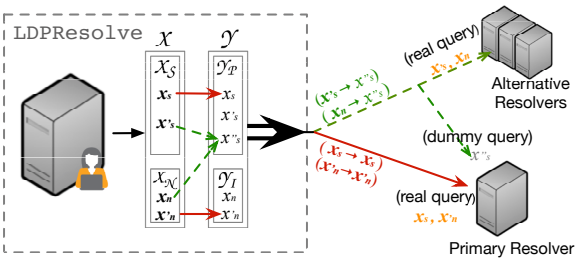


Figure 6: Workflow of LDPRESOLVE. The symbols are defined in Section 4.2. A DNS query might be perturbed and sent to AltRR based on $(\mathcal{X}_\mathcal{S}, \mathcal{Y}_\mathcal{P}, \epsilon_1, \epsilon_2)$-ULDP.

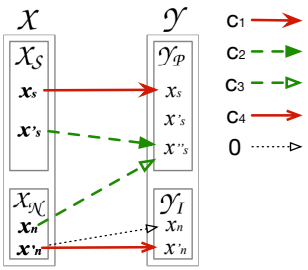


Figure 7: An illustration of how data are perturbed under $(\mathcal{X}_\mathcal{S}, \epsilon_1, \epsilon_2)$-URR.

queries for better privacy was described in [24], [43]. Compared to previous works, our mechanism is fine-tuned with LDP. As such, users' privacy can be guaranteed under a privacy budget ϵ. In addition, LDPRESOLVE ensures the DNS data is usable by legitimate applications that rely on aggregated statistics.

Workflow. Figure 6 illustrates the design and workflow of LDPRESOLVE, following the concept of randomized response. When the stub resolver is about to send a DNS query for domain d to a resolver configured by the user (i.e., primary resolver), she asks an oracle integrated by the stub resolver whether d should be perturbed, depending on a generated probability and whether d is listed in a *sensitive set* (set of popular domains). The sensitive set is generated by a third-party, e.g., Alexa [9] or authoritative nameservers, and it is periodically delivered to the stub resolver. When it is determined to be perturbed, it sends a different query with domain name d' to the primary resolver and the original d to an AltRR to obtain the authentic answer.

AltRR. AltRR can be another resolver that is not colluding with the primary resolver. It can be a local recursive resolver (RR), which directly talks to authoritative nameservers. Instead of altering the entire DNS resolution [81], we use a combination structure of AltRR and Primary RR, applying a DP-based approach to rationally distribute queries among RR. In doing so, unlike previous work that considered only privacy, we can achieve a balance between privacy, performance, and data utility. We expect the usage of AltRR will not significantly increase the latency. Firstly, Hoang et al. showed that DNS resolution is slightly longer with AltRR [43]. Secondly, only domains in the sensitive set go through AltRR and we can use the prefetch strategy since the sensitive set is known beforehand. In Section 5.7, we evaluate one AltRR implementation based on a local resolver.

As described in Section 2.3, **Encode**, **Perturb**, and **Aggregate** are the three key steps for an LDP protocol. Since we choose DE for **Encode**, **Aggregate** becomes a trivial process as no extra decoding is needed [89]. **Perturb** needs to be designed in light of ULDP and we elaborate it next.

4.2. Perturb for LDPRESOLVE

Let $\mathcal{X}$ be a set of DNS queries and $\mathcal{Y}$ be the perturbed queries. We use a randomized mechanism $\mathcal{A}$ to map $x \in \mathcal{X}$ to $y \in \mathcal{Y}$ with probability $\mathbf{P}(y|x)$. We divide DNS queries into sensitive queries about popular domains (termed $\mathcal{X}_\mathcal{S} \subseteq \mathcal{X}$) and non-sensitive queries about unpopular domains (termed $\mathcal{X}_\mathcal{N} \subseteq \mathcal{X}$). After perturbation, $\mathcal{Y}_\mathcal{P} \subseteq \mathcal{Y}$ and $\mathcal{Y}_\mathcal{I} \subseteq \mathcal{Y}$ are generated, which are associated with popular domains and unpopular domains respectively. We design $\mathcal{A}$ to satisfy a new DP notation $(\mathcal{X}_\mathcal{S}, \mathcal{Y}_\mathcal{P}, \epsilon_1, \epsilon_2)$-ULDP, as defined in Appendix A, and provide a concrete construction $(\mathcal{X}_\mathcal{S}, \epsilon_1, \epsilon_2)$-URR (or $(\mathcal{X}_\mathcal{S}, \epsilon_1, \epsilon_2)$-Utility-optimized Randomized Response) under it, as described below.

Definition 2 (**$(\mathcal{X}_\mathcal{S}, \epsilon_1, \epsilon_2)$-URR**). *Given $\mathcal{X}_\mathcal{S} \subseteq \mathcal{X}$ and $\epsilon_1, \epsilon_2 \in \mathbb{R}_{\geq 0}$, let $c_1 = \frac{e^{\epsilon_2}}{e^{\epsilon_2}+|\mathcal{X}_\mathcal{S}|-1}$, $c_2 = \frac{1}{e^{\epsilon_2}+|\mathcal{X}_\mathcal{S}|-1}$, $c_3 = \frac{1}{e^{\epsilon_1}+|\mathcal{X}_\mathcal{S}|-1}$, $c_4 = \frac{e^{\epsilon_1}-1}{e^{\epsilon_1}+|\mathcal{X}_\mathcal{S}|-1}$. Then the $(\mathcal{X}_\mathcal{S}, \epsilon_1, \epsilon_2)$-URR can be defined as:*

$$\mathbf{P}_{uRR}(y|x) = \begin{cases} c_1 & \text{if } x_i \in \mathcal{X}_\mathcal{S}, y = x \\ c_2 & \text{if } x_i \in \mathcal{X}_\mathcal{S}, y \in \mathcal{X}_\mathcal{S} \setminus \{x\} \\ c_3 & \text{if } x_i \in \mathcal{X}_\mathcal{N}, y \in \mathcal{X}_\mathcal{S} \\ c_4 & \text{if } x_i \in \mathcal{X}_\mathcal{N}, y = x \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where $c_4 \geq c_1 \geq c_2 \geq c_3$, $c_1 + (|\mathcal{X}_\mathcal{S}| - 1)c_2 = 1$, $c_4 + |\mathcal{X}_\mathcal{S}|c_3 = 1$. Figure 7 illustrates this protocol.

With these notations, we are able to prove via two different approaches that $(\mathcal{X}_\mathcal{S}, \epsilon_1, \epsilon_2)$-URR satisfies $(\mathcal{X}_\mathcal{S}, \mathcal{Y}_\mathcal{P}, \epsilon_1, \epsilon_2)$-ULDP. The proof is shown in Appendix B.

Our $(\mathcal{X}_\mathcal{S}, \mathcal{Y}_\mathcal{P}, \epsilon_1, \epsilon_2)$-ULDP notion is adapted from the $(\mathcal{X}_\mathcal{S}, \mathcal{Y}_\mathcal{P}, \epsilon)$-ULDP notion [67] by introducing an additional ϵ in order to provide stronger protection over the sensitive data entries (i.e., popular domains) while maintaining as much utility as possible for the non-sensitive data (e.g., unpopular domains). $(\mathcal{X}_\mathcal{S}, \mathcal{Y}_\mathcal{P}, \epsilon_1, \epsilon_2)$-ULDP provides ϵ_1-DP and ϵ_2-DP for different input, and it inherits other basic properties of ULDP [67], like *sequential composition* and *post-processing*.

Sensitive set. As described in Section 4.1, we want to build the sensitive set and associate it with a different privacy budget. The sensitive set consists of popular domains with a high volume of visits. As implied by the results we found in section 3.5, the repetitive queries of a user to popular domain names form his/her "identifier". Higher privacy protection for popular domain names is thus a reasonable deduction. In our experiment, the list is generated based on DNS sessions from another dataset of 9k users. When LDPRESOLVE is installed by the users, the sensitive set can be fetched periodically from a web server, like Adblock fetching EasyList [13] The sensitive set can also be augmented with domains chosen by the user, and we discuss this option in Section 6.

We assume the sensitive set is not a secret, so the adversary can obtain the set and actively prune the enlisted domains before user tracking. In Section 5.5, we discuss the impact of this strategy.

5. Evaluation of LDPResolve

We first describe the experiment settings for LDPResolve and three evaluation metrics. In Section 5.1, we investigate to what extent LDPResolve can curb DNS-based user tracking. In Section 5.2, the three parameters of LDPResolve are assessed. In Section 5.3 and Section 5.4, we discuss alternative settings of LDPResolve, In Section 5.5, we discuss the impact of the adaptive attack strategies. In Section 5.6, we compare against another relevant work that disperses DNS requests. In Section 5.7, we implement a prototype of LDPResolve and evaluate its overhead.

Experiment settings. The dataset used to evaluate LDPResolve is the same as described in Section 3.5, but we adjust how the unlabeled dataset **SU** and the labeled dataset **SL** are generated. Instead of using a fixed number of sessions of every user in **SL**, we choose 80% sessions of a user to fill **SL** and leave the remaining 20% for **SU**, which enhances the capability of the attacker. We test both the open-world and the closed-world settings, but focus on the closed-world setting, as it favors the adversary more.

To generate the sensitive set, we collect another DNS dataset with 272,078 sessions from 9,000 users. It has no overlap with **SL** and **SU**. We rank domains based on their frequencies in sessions and take the top N_S domains as the sensitive set.

We simulate DNS queries under LDPResolve by perturbing their enclosed domains. We vary different parameters, including ϵ_1, ϵ_2 and N_S to assess their impact. In the default setting, we only perturb sessions in **SU**, which represents the situation that the attacker has acquired a "clean" **SL** and tries to correlate it with a noisy **SU**.

Evaluation metrics. We consider three metrics to evaluate the data utility. The first is **tracking accuracy** (or `TrkAcc`), which is the same as the accuracy used to evaluate DSCorr (see Section 3.5). The goal of LDPResolve is to reduce it as much as possible.

The second is **standard deviation** (or `std`) of the domain frequency (used by prior works in malicious domain detection [19], [26], [63], [93]), measured by the session count. In addition to `std` across all domains, we also measure it on the ones in sensitive set and non-sensitive domains, and use `std_s` and `std_n` to represent their `std` respectively.

The third metric measures the utility at the session level. We measure the **change ratio** (or `ChgRatio`) of domain names and domain pairs after perturbation. If O is original set of domain names (or domain pairs) and P is perturbed set, `ChgRatio` is computed as $|O \cap P|/|O|$. For `ChgRatio` on a single domain name, sensitive domain (s) and non-sensitive (n) domain are calculated separately. For `ChgRatio` on domain pairs, pair of two sensitive domains (s,s), one sensitive domain and one non-sensitive domain (s, n), and two non-sensitive domains (n, n) are measured. We choose `ChgRatio` because it impacts the mapping of a domain name to the source IP address or domain names to domain names, which is also utilized a lot for malicious domain detection [27], [34], [59], [78].

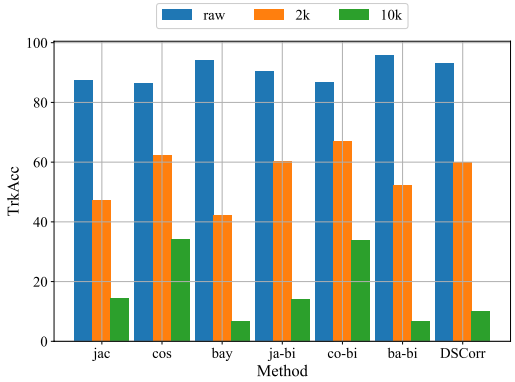


Figure 8: Comparison of `TrkAcc` before ("raw") and after deploying LDPResolve ("2k" and "10k"). "2k" and "10k" are values set to N_s. All numbers are percentage.

5.1. Impact on User Tracking

We first measure the impact of LDPResolve on all tracking methods (jac, cos, bay, ja-bi, co-bi, ba-bi, DSCorr) in the closed-world setting as described in Section 3.5. We set $\epsilon_1 = 10$ and $\epsilon_2 = 2$ to represent high & low privacy budgets. We set N_s to 2k and 10k out of more than 2 million domain names from the 9k-user set, to assess the impact of the sensitive set. When $N_s = 2000$, 72.6% domain names per session are sensitive, but 1.7% domains in **SU** are sensitive, showing a long-tail distribution. When $N_s = 10000$, the numbers are changed to 92.4% and 8.3% respectively.

Figure 8 shows `TrkAcc` before and after LDPResolve is applied[6]. It turns out DSCorr is influenced most: `TrkAcc` is dropped to 60.0% when N_s is 2k, and **10.1%** when N_s is 10k, from 93.0%. On the other hand, the impact to 1NN-Cosine (cos) is the the smallest among all tracking methods: `TrkAcc` is dropped to 62.2% and 34.1% from 86.6%, when N_s is 2k and 10k respectively. The result indicates LDPResolve is effective in protecting users' privacy, and it has stronger influence on tracking methods with higher `TrkAcc`.

We also test LDPResolve in the open-world setting, by setting N_s to 10k. The original `TrkAcc` for 1NN-Cosine and DSCorr are 70.9% and 82.3% respectively. Under LDPResolve, `TrkAcc` of them are dropped to 51.9% and 50.8%. In the open-world setting, one can achieve an accuracy of at least 50% by labeling all sessions as unknown. Thus, the `TrkAcc` drop here is sufficient enough to show the effectiveness of LDPResolve.

For the follow-up experiments, we choose 1NN-Cosine (uni-gram) as the tracking method since it is more robust to noise, and focus on the closed-world setting.

6. Noticeably, bi-ba has higher `TrkAcc` comparing to DSCorr (95.7% vs 93.0%). This is because we use 80% and 20% data for training and testing here, varying the number of labeled sessions per user. DSCorr performs better with less data.

ϵ_1	TrkAcc	std	std_s	std_n
15	38.7	332.30	1279.53	3.48
10	34.1	343.66	1279.63	5.71
9	28.4	352.52	1279.94	6.85
8	19.5	360.62	1280.39	8.54
7	10.2	365.38	1279.76	10.66
6	3.7	367.61	1279.92	10.75
5	1.4	368.45	1279.01	11.20
2	0.2	369.12	1280.31	10.84

TABLE 2: Impact of ϵ_1 on TrkAcc (shown in percentage) and overall std. std_s and std_n are std for sensitive and non-sensitive domains.

5.2. Impact of Parameters

We discuss the impact of different parameters ($\epsilon_1, \epsilon_2, N_s$) in LDPRESOLVE on the three evaluation metrics here. We set a large ϵ_1 (10) and a small ϵ_2 (2) initially because we intend to preserve more privacy for sensitive domains (more noise added) while maintaining better utility for other domains (less noise added). Theoretically, e^{ϵ_1} needs to be at least the same order as the size of the sensitive set N_s to avoid significant changes to non-sensitive domains. To evaluate the impact of each parameter, we fix other parameters and vary the tested parameter and obtain privacy and utility results.

We examine ϵ_1 from 2 to 15 and ϵ_2 from 0.5 to 10 with different size of the sensitive set N_s. We evaluate the impact of these parameters based on their TrkAcc, std and ChgRatio. Overall, a large ϵ_1 and N_s with small ϵ_2 is preferred. More specifically, by setting $\epsilon_1 = 10$, $\epsilon_2 = 2$ and $N_s = 20,000$, LDPRESOLVE is able to decrease the tracking accuracy to 23.3% for 1NN-Cosine, which is proved to be the method most robust to noise. Regarding the utility measured by std, our result indicates they can be preserved (especially for non-sensitive domains). For instance, when $\epsilon_2 = 2$ and $\epsilon_1 = 10$, std_n is only 5.71.

Impact of ϵ_1. ϵ_1 is the privacy budget for the whole domain set and the smaller ϵ_1 will introduce greater noise to all the domains. We tested 5 different ϵ_1 ranging from 2 to 15 while setting ϵ_2 and N_S to 2 and 10000 respectively. The result of TrkAcc and std are shown in Table 2. We see that as ϵ_1 drops, TrkAcc drops drastically due to the higher-level noise added. Besides, std of the whole domain set increases slowly with std of sensitive domains remaining almost the same and std for non-sensitive domains grows, because we use ϵ_2 to control the changes on the sensitive domains.

Figure 9a supports this claim as well with the result on ChgRatio. The co-occurrence of any sets of domains involving non-sensitive domains is dropping sharply from a very high level as ϵ_1 gets smaller. Meanwhile, 99.8% of non-sensitive domain names and 99.5% non-sensitive domain pairs are unchanged when ϵ_1 is set to 15. Only less than 16.8% of non-sensitive domains and 1.3% non-sensitive domain pairs remain the same after the ϵ_1 is decreased to 7. Since non-sensitive domains play a big role in security research, ϵ_1 should be set to a relatively high value in order to guarantee reasonable utility.

Impact of ϵ_2. ϵ_2 is the privacy budget for sensitive domains only, and the smaller ϵ_2 introduces more noise. Similarly to the last setting, we fix ϵ_1 to 10, N_S to 10000, and examine the impact of different ϵ_2 from 0.5 to 10. In this setting, we have the non-sensitive domains not impacted so their utility to legitimate applications is preserved. But because tracking relies on sensitive domains, it is disrupted.

ϵ_2	TrkAcc	std	std_s	std_n
10	84.8	121.59	241.10	3.27
8	80.2	264.24	731.94	3.80
7	70.3	305.47	967.65	5.31
6	57.4	326.82	1127.27	5.52
5	43.6	336.81	1214.65	5.67
2	34.1	343.66	1279.63	5.71
0.5	33.9	343.95	1282.55	4.38

TABLE 3: Impact of ϵ_2 on TrkAcc, std, std_s and std_n.

N_S	TrkAcc	std	std_s	std_n
1000	68.0	363.13	2552.22	1.72
2000	62.2	388.23	2205.18	2.15
5000	48.8	376.73	1669.81	6.54
10000	34.1	343.66	1279.63	5.71
20000	23.3	304.17	949.84	7.13

TABLE 4: Impact of N_S on TrkAcc, std, std_s and std_n.

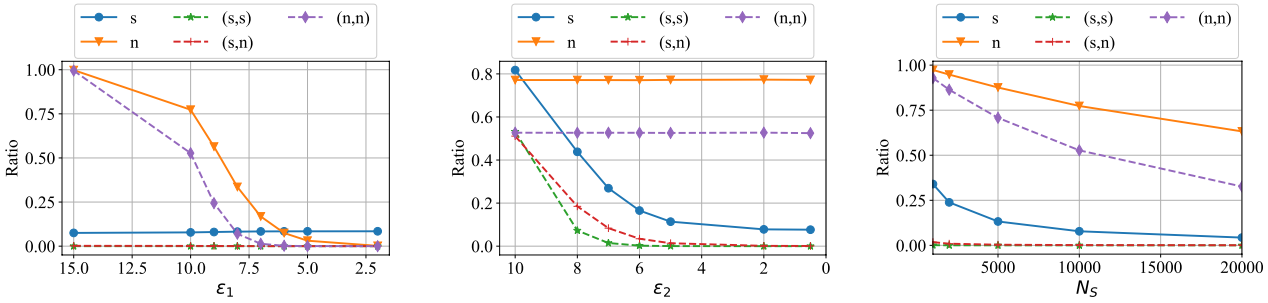
As shown in Table 3, std of non-sensitive domains is small (all less than 6) and stable across different ϵ_2. A small difference is observed because a different set of non-sensitive domains is perturbed every time we run the experiment. The same pattern can be found in Figure 9b where the ChgRatio of domains or domain pairs without involving sensitive domains remains almost unchanged under the fluctuation of ϵ_2.

In conclusion, higher ϵ_1 and lower ϵ_2 are preferred for a good balance between privacy and data utility.

Impact of N_S. We fix ϵ_1 to 10, ϵ_2 to 2 and change N_S from 1k to 20k. Table 4 shows that by increasing N_S, user tracking is severely interfered, with TrkAcc dropping from 68.0% to 23.3%. In the meantime, its impact on non-sensitive domains is controlled, with std of them ranging from 1.72 to 7.13. Figure 9c shows that with the increase of N_S, ChgRatio of sensitive domains, nonsensitive domains and pairs of nonsensitive domains are decreasing.

Here we explain this observation in depth. Followed by increase of N_s, 4 perturbation probabilities c_1, c_2, c_3, c_4 all decrease. Sensitive domains will have a greater chance to be changed as they are associated with c_1, c_2, c_3, while non-sensitive ones are less changed as it is only impacted by c_4. With N_s increased, sensitive set is expanded to contain more low-frequency domains, so std_n also increases. For pairs associated with sensitive domains, the perturbation breaks their relations, so the ChgRatio of them remain low with a trend of decreasing.

From the result and explanation above, it is clear that N_s is also an essential parameter for LDPRESOLVE. Larger N_s is recommended when the legitimate applications highly rely on the non-sensitive domains. Though due to the power-law distribution of domains, 1k and 20k have a big difference of influence on user sessions, only a small portion of the whole domain set is impacted, as shown in Section 5.1.



(a) ChgRatio vs. ϵ_1 (b) ChgRatio vs. ϵ_2 (c) ChgRatio vs. N_S

Figure 9: ChgRatio vs. ϵ_1, ϵ_2 and N_S. s, n, (s,s), (s,n) and (n,n) are explained in "Evaluation metrics".

5.3. Noisy SL

Our default setting assumes **SL** is clean to the adversary. We further explore the scenario when **SL** has noises injected by LDPResolve. The parameters are the same as our default setting: $\epsilon_1 = 10, \epsilon_2 = 2$, $N_s = 10000$.

Firstly, we assume **SL** is noisy while **SU** is clean. The tracking accuracy turns out to be 23.4%, which is even worse than 34.1% when we assume a clean **SL** and a noisy **SU**. Therefore, such "data poisoning attack" is even more effective against user tracking. Secondly, we allow one clean session for each user to be included in **SL** (**SU** is still clean). Tracking accuracy will be increased to 41.9%. The result suggests even one clean session in **SL** can give adversary great lift in countering LDPResolve. Finally, if both **SL** and **SU** are noisy, the accuracy will drop back to 32.5%.

5.4. Sensitive Set with SLDs

So far we fill the sensitive set with FQDNs (Full Qualified Domain Names). It can also be constructed by extracting the SLDs (Second-Level Domains) part from the popular FQDNs, by leveraging a public suffix list [4]. Then, if the SLD of a domain name matches the sensitive set, it will be considered as in $\mathcal{X}_S$. By doing so, the subdomains under sensitive domains are also protected.

By building the new sensitive set based on only 100 most popular SLDs from the same 9k-user dataset, tracking accuracy is dropped to 14.16%. This result shows that by adding strong noises to a small set of SLDs, tracking will be significantly disturbed.

When applying this change, certain domains need to be excluded, i.e., not adding their SLDs to sensitive set. One example is domains requested under PTR Record. Because *all* PTR records are under the same SLD in-addr.arpa or ip6.arpa, if including those domains in sensitive set, security research based on PTR records [70] will be significantly impaired.

5.5. Adaptive Tracking against LDPResolve

To counter LDPResolve, an adaptive attacker can try to eliminate the noises introduced by different means. As ULDP ensures domains in the non-sensitive set are invertible, one feasible option is to remove the observed domains that appear in the sensitive set, therefore reduce the effect of change of domains. In this way, the queries left in the records contain only authentic domains. Another option to eliminate the effect of LDPResolve is to estimate the domain frequency by *reversing* $(\mathcal{X}_S, \epsilon_1, \epsilon_2)$-URR.

Removing sensitive domains. By removing the sensitive domains in the DNS records, most of the noises would be removed along with them. On the other hand, the tracking effectiveness should not be restored to the level without LDPResolve, since there are less domains to be used to connect sessions of the same user.

It turns out that tracking accuracy rises from 34.06% to 53.08% after this adaptive strategy. For the vanilla setting (attacker has access to the clean data in both **SL** and **SU** and no domains are removed), the accuracy is 86.6%. As such, we argue that even this strategy is applied, the tracking accuracy is far from optimum for the adversary.

Reversing $(\mathcal{X}_S, \epsilon_1, \epsilon_2)$-URR. LDPResolve uses $(\mathcal{X}_S, \epsilon_1, \epsilon_2)$-URR to perturb a request. The process depends on a few parameters (c_1, c_2, c_3, c_4). When they are known to the adversary, she might attempt to reverse the perturbation process to estimate the real distribution of a domain based on the observed distribution. We propose an implementation for this strategy and evaluate its impact on LDPResolve. Mathematical details are listed in Appendix C. It turns out this strategy does not work well when the sensitive set is large, which introduces large randomness to the perturbation process. When N_s is 10k, TrkAcc is slightly increased from 34.06% to 34.54%. As an alternative solution, the adversary could choose to reverse the non-sensitive set only, which are derived by excluding sensitive set from the entire domain set. This strategy increases TrkAcc to 53.50%, but still far from the vanilla setting when LDPResolve is not deployed (86.6%).

5.6. Comparison with K-resolver

As described in Section 4.1, K-resolver [43] is expected to deter tracking by dispersing DNS queries across resolvers. We test K-resolver by splitting DNS requests into k slices, and launch 1NN-Cosine tracking against it. The detailed evaluation results are shown below.

We vary k from 18 to 90 (with step size 8) and TrkAcc is reduced from 62.89% to 30.7%, as shown in Figure 10. It turns out only when k is very large (over 74), K-resolver can outperform LDPResolve (TrkAcc

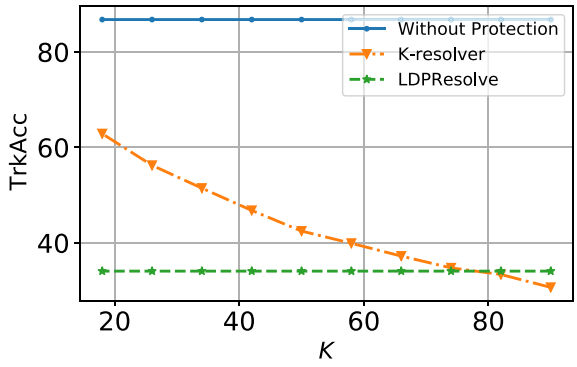


Figure 10: Comparison on TrkAcc of LDPRESOLVE and K-resolver by k.

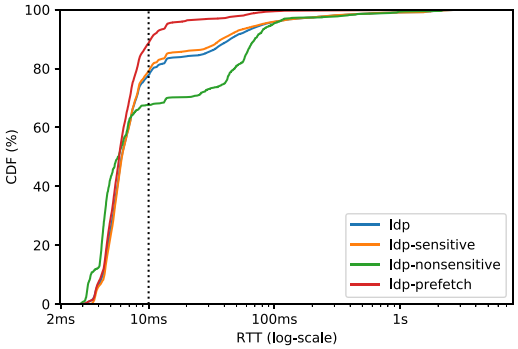


Figure 11: Comparison of RTT between different settings of LDPRESOLVE. "ldp" are all queries. "ldp-sensitive" and "ldp-nonsensitive" are queries to domains in and not in the sensitive list. "ldp-prefetch" is RTT of all queries when prefetch of sensitive domains is enabled.

at 34.06%) in theory. However, finding such a big pool of resolvers is quite difficult. In fact, Hoang et al. investigated 53 DNS-over-HTTPs resolvers and found only 26 of them can provide reliable services. LDPRESOLVE offers sufficient protection by involving much fewer resolvers (only 2). In fact, as a deterministic hash function is used by K-resolver, a group of domains will always be sent to the same resolver, regardless of how much information they leak. The randomized protocol of LDPRESOLVE addresses this limitation.

5.7. Prototype

We implement a prototype of LDPRESOLVE to evaluate its overhead. Our prototype is built on top of dnsdist [1], an open-source DNS load-balancer. We write Lua, Python, and Shell scripts to customize dnsdist to support LDPRESOLVE. The code we wrote is released on GitHub.

For domains that are unperturbed, we query them through a primary resolver, which is a public resolver (223.5.5.5) in our experiment. For domains that are perturbed, two kinds of queries are issues: 1) queries sent to the alternative resolver (AltRR), which is set to be a trusted local recursive resolver running PowerDNS [6] and directly talking to authoritative nameservers, and 2) dummy queries (i.e., noise) sent to the primary resolver. The two types of queries are issued in separate processes, so the impact on the normal DNS resolution is confined. It is also worth mentioning that we create a local cache with a fixed size similar to the sensitive set.

We randomly choose a group of sessions of one user in our dataset, replay the first 10k DNS queries (qname/qtype combination) and evaluate the Round-trip Time (RTT) and traffic volume.

RTT. Firstly, we measure the distribution of the query RTTs under LDPRESOLVE. Figure 11 shows the CDF plot of the 10K queries. Because of the local cache, RTT for most of the queries is less than 10ms (querying the primary resolver takes about 20ms). Responses to the non-sensitive domain names are slower than the sensitive ones, because they are less likely to be requested multiple times (so cached). We also examined the idea of prefetching the domain names in the sensitive set, and the RTT turns out to be even smaller generally. Overall, our result shows LDPRESOLVE is efficient under parallel domain resolving.

Traffic volume. We found the 10k queries generate about 2.76MB of DNS traffic to the public resolver. When using LDPRESOLVE, 3.28MB DNS traffic is generated, where 2.36MB goes to the primary resolver, and 0.92MB goes to AltRR, which is 18.8% higher than querying DNS normally. The traffic sent to the primary resolver is actually reduced because of the local cache.

6. Discussion

Limitations of DSCORR. 1) While fields in a DNS request other than the domain name, like query timestamp, could potentially be exploited for user tracking, they are not used by DSCORR. Through empirical analysis, considering those fields could significantly increase the computation complexity and introduce noises. 2) The domains not encountered when training the embedding model will be ignored during session clustering. As an alternative approach, the domain embedding can be updated on the fly when new domains emerge. The NLP community also faces this "out-of-vocabulary" problem, and methods in this area [75], [76] can be leveraged to tackle this issue. 3) Though DSCORR can distinguish unknown sessions from known, it can not automatically build profiles for unseen users.

Limitations of LDPRESOLVE. 1) The utility could be worse for popular domains due to ULDP, which can hurt the applications like domain popularity ranking at recursive resolvers. From the level of authoritative nameservers, the impact is expected to be smaller, as queries from RRs over the world are aggregated. 2) We consider the passive attacker who only does traffic analysis and do not consider the active attacker who is able to change the state of the user's device, like changing DNS cache [56]. It would be an interesting study about whether or how an active attacker would be affected under LDPRESOLVE. 3) According to our implemented prototype of LDPRESOLVE, the overhead of traffic volume is moderate and the latency can even be reduced when AltRR uses local resolver, local cache, or prefetch. These options might not always be available to the users, so the performance result should be perceived with a grain of salt.

Sensitive set. As the sensitive set is generated based on the historical domain popularity, an attacker can manipulate this set by crafting domain visits. There exist two strategies: 1) injecting a malicious domain into the sensitive set so LDPRESOLVE will perturb it at higher probabilities, but it also makes a user less likely to visit it; 2) injecting many irrelevant domains to "kick out" the real popular domains, but to override the query volume from users requires significant network bandwidth. As such, there is no strong motivation for an attacker to manipulate the sensitive set.

LDPRESOLVE issues the same sensitive set to all users. An alternative approach is to let a user customize it for better performance at her end. We can allow a user to manually change or automatically generate it from her visiting history.

Limitations of dataset. For evaluation, we use only one DNS dataset. We acknowledge that using more datasets can address the potential bias caused by the user population, but getting access to one, especially with session labels, is very challenging. Nevertheless, we want to emphasize that our DNS dataset contains 10,000 users launching over 300,000 DNS sessions, targeting 2 million domain names, which is sufficient for a large-scale study.

Impact of NAT. To the best of our knowledge, our dataset should not have data from users behind NAT. Though we did not evaluate the impact of NAT on DSCORR and LDPRESOLVE, we expect the impact is small when the number of users behind a NAT address is small, as each DNS session is more likely to be associated with only one user. However, the impact would be amplified when more users are behind a NAT address.

7. Related Works

In this section, we discuss the related works in differential privacy and DNS security and privacy. The related works about user tracking have been described in Section 2.2.

Differential privacy. Our defense mechanism LDPRESOLVE leverages LDP to protect users against user tracking while retaining sufficient data utility for legitimate applications. The utility function modeled by us belongs to frequency estimation, which is considered as a major use case for LDP with different protocols developed [32], [49], [79], [89]. To enable better data utility with LDP, we adopt ULDP [38], [67], a variation of LDP, and adjust it for our setting.

We apply LDP to network data generated by users, in particular DNS data. Prior works applied LDP mostly in the data from the browser, like monitoring aggregated browsing activities [33], learning the frequency of website visits [32], enabling local search [17], supporting web search with differentially-private query logs [95]. Different from the browser setting in which the data curator asks for the reports from users, in our setting the local resolver adds noise to a portion of queries actively.

DNS security and privacy. Cyber-attackers tend to leverage DNS infrastructure to hide the IP address of a malicious server, e.g., C&C server. Therefore, a long line of works studied how to leverage the DNS logs recorded by DNS servers for detection. The detection features mainly come from the DNS packet, the auxiliary information about domain name and IP, and statistical features about DNS traffic. The lexical features of domain names such as the distribution of characters [91] or their complexity [72] are used to identify DGA domains. Fields in DNS packets are also used as features, such as TTL and IP in DNS response [19]. Information like ASN [27], zone files [39] or WHOIS [94] can augment the DNS information. A few works applied graph-based algorithms on domain-IP graph [59], [71], [78] to identify malicious domains. Domain frequency is also an important feature and is primarily evaluated by LDPRESOLVE.

Though we introduce noises into DNS queries, the utility loss to domain detection is contained. First, by treating sensitive and non-sensitive domains with different ϵ, the feature values about suspicious domains are largely preserved. Second, the features about individual domains are unchanged, like their lexical features.

Regarding the privacy of DNS, in addition to user tracking, eavesdropping and manipulating DNS are also important issues. It has been found that parties like NSA and ISP are monitoring and hijacking DNS traffic [7], [37], [62], which leads to significant efforts from the Internet community in recent years to develop and deploy encrypted DNS, including DNS-over-TLS, DNS-over-HTTPS and etc, but users' traffic is not immune to the attacks based on statistical analysis on traffic [44], [82], [83], or when the resolver is the adversary.

8. Conclusion

In this paper, we study the issue of user tracking on DNS data. On the side of attack, we introduced a new mechanism named DSCORR, to track users based on their DNS queries, which turns out to be effective under both open-world and closed-world settings. Based on our observation on the attack side, we then present a new mechanism named LDPRESOLVE to make DNS sessions indistinguishable, using a generalized version of ULDP and new constructions satisfying its requirements. Finally, we evaluate the effectiveness of LDPRESOLVE in different settings to prove its capability to protect users' privacy from tracking while preserving the utility for legitimate applications based on DNS data. Our study suggests the threats coming from the DNS-based user tracking should be mitigated and it is feasible to protect users' privacy without damaging the utility of legitimate applications.

Acknowledgements

The authors are thankful to the anonymous reviewers for their supportive reviews. The authors from University of California, Irvine were supported by NSF 2047476 and gift from Cisco and Microsoft.

References

[1] dnsdist overview. https://dnsdist.org/.

[2] Google Public DNS. https://developers.google.com/speed/public-dns/.

[3] How to enable dns over https in google chrome. https://www.howtogeek.com/660088/how-to-enable-dns-over-https-in-google-chrome/.

[4] Public suffix list. https://publicsuffix.org/.

[5] S.1578 - do not track act. https://www.congress.gov/bill/116th-congress/senate-bill/1578/text.

[6] Welcome to powerdns. https://www.powerdns.com/documentation.html.

[7] The NSA and GCHQ's QUANTUMTHEORY Hacking Tactics. https://theintercept.com/document/2014/03/12/nsa-gchqs-quantumtheory-hacking-tactics/, 2014.

[8] Learning with privacy at scale. https://machinelearning.apple.com/2017/12/06/learning-with-privacy-at-scale.html, 2017.

[9] Alexa top sites. https://aws.amazon.com/alexa-top-sites/, 2019.

[10] Dns pai. http://www.dnspai.com/, 2019.

[11] Doh: (anti-)competitive and network neutrality aspects. https://blog.powerdns.com/2019/12/03/doh-anti-competitive-and-network-neutrality-aspects/, 2019.

[12] Security information exchange (sie) protects from cybercrime. https://www.farsightsecurity.com/solutions/security-information-exchange/, 2019.

[13] Easylist overview. https://easylist.to/, 2021.

[14] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 674–689, 2014.

[15] Mario Almeida, Alessandro Finamore, Diego Perino, Narseo Vallina-Rodriguez, and Matteo Varvello. Dissecting dns stakeholders in mobile networks. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 28–34, 2017.

[16] Noah Apthorpe, Danny Yuxing Huang, Dillon Reisman, Arvind Narayanan, and Nick Feamster. Keeping the smart home private with smart (er) iot traffic shaping. *Proceedings on Privacy Enhancing Technologies*, 2019(3):128–148, 2019.

[17] Brendan Avent, Aleksandra Korolova, David Zeber, Torgeir Hovden, and Benjamin Livshits. BLENDER: Enabling local search with a hybrid differential privacy model. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 747–764, 2017.

[18] Mika D Ayenson, Dietrich James Wambach, Ashkan Soltani, Nathan Good, and Chris Jay Hoofnagle. Flash cookies and privacy ii: Now with html5 and etag respawning. *Available at SSRN 1898390*, 2011.

[19] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. Exposure: Finding malicious domains using passive dns analysis. In *Ndss*, pages 1–17, 2011.

[20] Jim Black. Learn How Trillions of DNS Requests Help Improve Security. https://blogs.akamai.com/2018/05/learn-how-trillions-of-dns-requests-help-improve-security.html.

[21] Kevin Borgolte, Tithi Chattopadhyay, Nick Feamster, Mihir Kshirsagar, Jordan Holland, Austin Hounsel, and Paul Schmitt. How dns over https is reshaping privacy, performance, and policy in the internet ecosystem. *Performance, and Policy in the Internet Ecosystem (July 27, 2019)*, 2019.

[22] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: Evidence and implications. In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, volume 1, pages 126–134. IEEE, 1999.

[23] Jonas Bushart and Christian Rossow. Padding ain't enough: Assessing the privacy guarantees of encrypted dns. *arXiv preprint arXiv:1907.01317*, 2019.

[24] Sergio Castillo-Perez and Joaquin Garcia-Alfaro. Evaluation of two privacy-preserving protocols for the dns. In *2009 Sixth International Conference on Information Technology: New Generations*, pages 411–416. IEEE, 2009.

[25] Yizheng Chen, Manos Antonakakis, Roberto Perdisci, Yacin Nadji, David Dagon, and Wenke Lee. Dns noise: Measuring the pervasiveness of disposable domains in modern dns traffic. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 598–609. IEEE, 2014.

[26] Hyunsang Choi, Hanwoo Lee, Heejo Lee, and Hyogon Kim. Botnet detection by monitoring group activities in dns traffic. In *7th IEEE International Conference on Computer and Information Technology (CIT 2007)*, pages 715–720. IEEE, 2007.

[27] Hyunsang Choi and Heejo Lee. Identifying botnets by capturing group activities in dns traffic. *Computer Networks*, 56(1):20–33, 2012.

[28] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[29] Cynthia Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer, 2008.

[30] Peter Eckersley. How unique is your web browser? In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 1–18. Springer, 2010.

[31] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1388–1401, 2016.

[32] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067, 2014.

[33] Liyue Fan, Luca Bonomi, Li Xiong, and Vaidy Sunderam. Monitoring web browsing behavior with differential privacy. In *Proceedings of the 23rd international conference on World wide web*, pages 177–188, 2014.

[34] Hongyu Gao, Vinod Yegneswaran, Yan Chen, Phillip Porras, Shalini Ghosh, Jian Jiang, and Haixin Duan. An empirical reexamination of global dns behavior. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 267–278, 2013.

[35] Benjamin Greschbach, Tobias Pulls, Laura M Roberts, Philipp Winter, and Nick Feamster. The effect of dns on tor's anonymity. *arXiv preprint arXiv:1609.08187*, 2016.

[36] Chris Grier, Lucas Ballard, Juan Caballero, Neha Chachra, Christian J Dietrich, Kirill Levchenko, Panayiotis Mavrommatis, Damon McCoy, Antonio Nappa, Andreas Pitsillidis, Niels Provos, M. Zubair Rafique, Moheeb Abu Rajab, Christian Rossow, Kurt Thomas, Vern Paxson, Stefan Savage, and Geoffrey M. Voelker. Manufacturing compromise: the emergence of exploit-as-a-service. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 821–832, 2012.

[37] Christian Grothoff, Matthias Wachs, Monika Ermert, and Jacob Appelbaum. NSA's MORECOWBELL: Knell for DNS. https://leaksource.files.wordpress.com/2015/02/nsas-morecowbell-knell-for-dns.pdf, 2015.

[38] Xiaolan Gu, Ming Li, Li Xiong, and Yang Cao. Providing input-discriminative protection for local differential privacy. *arXiv preprint arXiv:1911.01402*, 2019.

[39] Shuang Hao, Alex Kantchelian, Brad Miller, Vern Paxson, and Nick Feamster. Predator: proactive recognition and elimination of domain abuse at time-of-registration. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1568–1579, 2016.

[40] Dominik Herrmann, Christian Banse, and Hannes Federrath. Behavior-based tracking: Exploiting characteristic patterns in dns traffic. *Computers & Security*, 39:17–33, 2013.

[41] Dominik Herrmann, Christoph Gerber, Christian Banse, and Hannes Federrath. Analyzing characteristic host access patterns for re-identification of web user sessions. In *Nordic Conference on Secure IT Systems*, pages 136–154. Springer, 2010.

[42] Dominik Herrmann, Matthias Kirchler, Jens Lindemann, and Marius Kloft. Behavior-based tracking of internet users with semi-supervised learning. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, pages 596–599. IEEE, 2016.

[43] Nguyen Phong Hoang, Ivan Lin, Seyedhamed Ghavamnia, and Michalis Polychronakis. K-resolver: Towards decentralizing encrypted dns resolution. *arXiv preprint arXiv:2001.08901*, 2020.

[44] Rebekah Houser, Zhou Li, Chase Cotton, and Haining Wang. An investigation on information leakage of dns over tls. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 123–137, 2019.

[45] Z Hu, Liang Zhu, John Heidemann, Allison Mankin, Duane Wessels, and Paul Hoffman. Specification for DNS over transport layer security (TLS). Technical report, 2016.

[46] Bert Hubert. Centralised doh is bad for privacy, in 2019 and beyond. https://labs.ripe.net/Members/bert_hubert/centralised-doh-is-bad-for-privacy-in-2019-and-beyond, 2019.

[47] P Huffman and P McManus. Dns queries over https (doh). Technical report, 2018.

[48] Basileal Imana, Aleksandra Korolova, and John Heidemann. Institutional privacy risks in sharing dns data. In *Proceedings of the Applied Networking Research Workshop*, ANRW '21, page 69–75, New York, NY, USA, 2021. Association for Computing Machinery.

[49] Jinyuan Jia and Neil Zhenqiang Gong. Calibrate: Frequency estimation and heavy hitter identification with local differential privacy via incorporating prior knowledge. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 2008–2016. IEEE, 2019.

[50] Jianguo Jiang, Jiuming Chen, Kim-Kwang Raymond Choo, Chao Liu, Kunying Liu, Min Yu, and Yongjian Wang. A deep learning based online malicious url and dns detection scheme. In *International Conference on Security and Privacy in Communication Systems*, pages 438–448. Springer, 2017.

[51] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. Dns performance and the effectiveness of caching. *IEEE/ACM Transactions on networking*, 10(5):589–603, 2002.

[52] Dae Wook Kim and Junjie Zhang. You are how you query: Deriving behavioral fingerprints from dns traffic. In *International Conference on Security and Privacy in Communication Systems*, pages 348–366. Springer, 2015.

[53] Dae Wook Kim and Junjie Zhang. Deriving and measuring dns-based fingerprints. *Journal of Information Security and Applications*, 36:32–42, 2017.

[54] Eric Kinnear, Tommy Pauly, and Chris Wood. Adaptive dns: Improving privacy of name resolution. Technical report, Technical report, 2019.

[55] Matthias Kirchler, Dominik Herrmann, Jens Lindemann, and Marius Kloft. Tracked without a trace: linking sessions of users by unsupervised learning of patterns in their dns traffic. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, pages 23–34, 2016.

[56] Amit Klein and Benny Pinkas. Dns cache-based user tracking. In *Network and Distributed System Security Symposium (NDSS'19). San Diego, CA, USA (Feb 2019)*, 2019.

[57] Tadayoshi Kohno, Andre Broido, and Kimberly C Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, 2005.

[58] Marek Kumpošt and Vašek Matyáš. User profiling and re-identification: case of university-wide network analysis. In *International Conference on Trust, Privacy and Security in Digital Business*, pages 1–10. Springer, 2009.

[59] Jehyun Lee, Jonghun Kwon, Hyo-Jeong Shin, and Heejo Lee. Tracking multiple c&c botnets by analyzing dns traffic. In *2010 6th IEEE Workshop on Secure Network Protocols*, pages 67–72. IEEE, 2010.

[60] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014.

[61] Baojun Liu, Zhou Li, Peiyuan Zong, Chaoyi Lu, Haixin Duan, Ying Liu, Sumayah Alrwais, Xiaofeng Wang, Shuang Hao, Yaoqi Jia, Yiming Zhang, Kai Chen, and Zaifeng Zhang. Traffickstop: Detecting and measuring illicit traffic monetization through large-scale dns analysis. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 560–575. IEEE, 2019.

[62] Baojun Liu, Chaoyi Lu, Haixin Duan, Ying Liu, Zhou Li, Shuang Hao, and Min Yang. Who is answering my queries: understanding and characterizing interception of the dns resolution path. In *USENIX Security Symposium*, pages 1113–1128, 2018.

[63] Daiping Liu, Zhou Li, Kun Du, Haining Wang, Baojun Liu, and Haixin Duan. Don't let one rotten apple spoil the whole barrel: Towards automated detection of shadowed domains. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 537–552, 2017.

[64] Waldemar Lopez, Jorge Merlino, and Pablo Rodriguez-Bocca. Vector representation of internet domain names using a word embedding technique. In *2017 XLIII Latin American Computer Conference (CLEI)*, pages 1–8. IEEE, 2017.

[65] Rada Mihalcea, Courtney Corley, Carlo Strapparava, et al. Corpus-based and knowledge-based measures of text semantic similarity. In *Aaai*, volume 6, pages 775–780, 2006.

[66] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[67] Takao Murakami and Yusuke Kawamoto. Utility-optimized local differential privacy mechanisms for distribution estimation. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 1877–1894, 2019.

[68] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. Deepcorr: Strong flow correlation attacks on tor using deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1962–1976, 2018.

[69] Thông T Nguyên, Xiaokui Xiao, Yin Yang, Siu Cheung Hui, Hyejin Shin, and Junbum Shin. Collecting and analyzing data from smart device users with local differential privacy. *arXiv preprint arXiv:1606.05053*, 2016.

[70] Jon Oberheide, Manish Karir, and Z Morley Mao. Characterizing dark dns behavior. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 140–156. Springer, 2007.

[71] Alina Oprea, Zhou Li, Ting-Fang Yen, Sang H Chin, and Sumayah Alrwais. Detection of early-stage enterprise infection by mining large-scale log data. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 45–56. IEEE, 2015.

[72] Hsing-Kuo Pao, Yan-Lin Chou, and Yuh-Jye Lee. Malicious url detection based on kolmogorov complexity estimation. In *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, volume 1, pages 380–387. IEEE, 2012.

[73] Vern Paxson, Mihai Christodorescu, Mobin Javed, Josyula Rao, Reiner Sailer, Douglas Lee Schales, Mark Stoecklin, Kurt Thomas, Wietse Venema, and Nicholas Weaver. Practical comprehensive bounds on surreptitious communication over {DNS}. In *22nd {USENIX} Security Symposium ({USENIX} Security 13)*, pages 17–32, 2013.

[74] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[75] Yuval Pinter, Robert Guthrie, and Jacob Eisenstein. Mimicking word embeddings using subword rnns. *arXiv preprint arXiv:1707.06961*, 2017.

[76] Victor Prokhorov, Mohammad Taher Pilehvar, Dimitri Kartsaklis, Pietro Lio, and Nigel Collier. Unseen word representation by aligning heterogeneous lexical semantic spaces. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6900–6907, 2019.

[77] Roxana Radu and Michael Hausding. Consolidation in the dns resolver market – how much, how fast, how dangerous? *Journal of Cyber Policy*, 5(1):46–64, 2020.

[78] Babak Rahbarinia, Roberto Perdisci, and Manos Antonakakis. Segugio: Efficient behavior-based tracking of malware-control domains in large isp networks. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 403–414. IEEE, 2015.

[79] Xuebin Ren, Chia-Mu Yu, Weiren Yu, Shusen Yang, Xinyu Yang, Julie A McCann, and S Yu Philip. Lopub: High-dimensional crowdsourced data publication with local differential privacy. *IEEE Transactions on Information Forensics and Security*, 13(9):2151–2166, 2018.

[80] Franziska Roesner, Tadayoshi Kohno, and David Wetherall. Detecting and defending against third-party tracking on the web. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 155–168, 2012.

[81] Paul Schmitt, Anne Edmundson, Allison Mankin, and Nick Feamster. Oblivious dns: Practical privacy for dns queries. *Proceedings on Privacy Enhancing Technologies*, 2019(2):228–244, 2019.

[82] Haya Shulman. Pretty bad privacy: Pitfalls of dns encryption. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 191–200, 2014.

[83] Sandra Siby, Marc Juarez, Claudia Diaz, Narseo Vallina-Rodriguez, and Carmela Troncoso. Encrypted dns–¿ privacy? a traffic analysis perspective. *arXiv preprint arXiv:1906.09682*, 2019.

[84] Jonathan M Spring and Carly L Huth. The impact of passive dns collection on end-user privacy. 2012.

[85] Mingxuan Sun, Guangyue Xu, Junjie Zhang, and Dae Wook Kim. Tracking you through dns traffic: Linking user sessions by clustering with dirichlet mixture model. In *Proceedings of the 20th ACM International Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems*, pages 303–310, 2017.

[86] Yixin Sun, Kangkook Jee, Suphannee Sivakorn, Zhichun Li, Cristian Lumezanu, Lauri Korts-Parn, Zhenyu Wu, Junghwan Rhee, Chung Hwan Kim, Mung Chiang, and Prateek Mittal. Detecting malware injection with program-dns behavior. In *2020 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 552–568, 2020.

[87] Nino Vincenzo Verde, Giuseppe Ateniese, Emanuele Gabrielli, Luigi Vincenzo Mancini, and Angelo Spognardi. No nat'd user left behind: Fingerprinting users behind nat from netflow records alone. In *2014 IEEE 34th International Conference on Distributed Computing Systems*, pages 218–227. IEEE, 2014.

[88] Tao Wang. High precision open-world website fingerprinting. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 152–167. IEEE, 2020.

[89] Tianhao Wang, Jeremiah Blocki, Ninghui Li, and Somesh Jha. Locally differentially private protocols for frequency estimation. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 729–745, 2017.

[90] Stanley L Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.

[91] Sandeep Yadav, Ashwath Kumar Krishna Reddy, AL Narasimha Reddy, and Supranamaya Ranjan. Detecting algorithmically generated malicious domain names. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 48–61, 2010.

[92] Xin Ye, Hui Shen, Xiao Ma, Razvan Bunescu, and Chang Liu. From word embeddings to document similarities for improved information retrieval in software engineering. In *Proceedings of the 38th international conference on software engineering*, pages 404–415, 2016.

[93] Ting-Fang Yen, Alina Oprea, Kaan Onarlioglu, Todd Leetham, William Robertson, Ari Juels, and Engin Kirda. Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. In *Proceedings of the 29th Annual Computer Security Applications Conference*, pages 199–208, 2013.

[94] Jialong Zhang, Sabyasachi Saha, Guofei Gu, Sung-Ju Lee, and Marco Mellia. Systematic mining of associated server herds for malware campaign discovery. In *2015 IEEE 35th International Conference on Distributed Computing Systems*, pages 630–641. IEEE, 2015.

[95] Sicong Zhang, Grace Hui Yang, Lisa Singh, and Li Xiong. Safelog: Supporting web search and mining by differentially-private query logs. In *2016 AAAI Fall Symposium Series*, 2016.

[96] Yury Zhauniarovich, Issa Khalil, Ting Yu, and Marc Dacier. A survey on malicious domains detection through dns data analysis. *ACM Computing Surveys (CSUR)*, 51(4):1–36, 2018.

Appendix A.
Definition of $(\mathcal{X}_\mathcal{S}, \mathcal{Y}_\mathcal{P}, \epsilon_1, \epsilon_2)$-ULDP

*Given $\mathcal{X}_\mathcal{S}, \mathcal{X}_\mathcal{N} \subseteq \mathcal{X}$, $\mathcal{Y}_\mathcal{P}, \mathcal{Y}_\mathcal{I} \subseteq \mathcal{Y}$ and $\epsilon_1, \epsilon_2 \in \mathbb{R}_{\geq 0}$, a randomize mechanism $\mathcal{A}$, where $\mathcal{A}(X) = Y$, provides $(\mathcal{X}_\mathcal{S}, \mathcal{Y}_\mathcal{P}, \epsilon_1, \epsilon_2)$-**ULDP** if it satisfies the following properties:*

1) For any $y \in \mathcal{Y}_\mathcal{I}$, $\exists\ x, x' \in \mathcal{X}_\mathcal{N}$

$$\mathbf{P}(y|x) > 0 \text{ and } \mathbf{P}(y|x') = 0, \forall x' \neq x. \tag{7}$$

2) For any $x, x' \in \mathcal{X}$ and any $y \in \mathcal{Y}_\mathcal{P}$,

$$\mathbf{P}(y|x) \leq e^{\epsilon_1}\mathbf{P}(y|x') \tag{8}$$

That is, ϵ_1-*differential privacy* is guaranteed for $\forall x \in \mathcal{X}$.

3) For any $x, x' \in \mathcal{X}_\mathcal{S}$ and any $y \in \mathcal{Y}_\mathcal{P}$,

$$\mathbf{P}(y|x) \leq e^{\epsilon_2}\mathbf{P}(y|x') \tag{9}$$

Where ϵ_2-*differential privacy* is guaranteed for $\forall x \in \mathcal{X}_\mathcal{S}$ and $\epsilon_1 \geq \epsilon_2$.

Appendix B.
Proof of $(\mathcal{X}_\mathcal{S}, \epsilon_1, \epsilon_2)$-URR

- *First Approach.* We can show how the properties in **Definition 3** hold by showing all possible scenarios of input and output combinations satisfy the definition:

1) For any $y \in \mathcal{Y}_\mathcal{I}$, there $\exists x, x' \in \mathcal{X}_\mathcal{N}$

$$\mathbf{P}_{uRR}(y|x) = c_4 > 0 \text{ and } \mathbf{P}_{uRR}(y|x') = 0, \forall x' \neq x. \tag{10}$$

2) For any $x, x' \in \mathcal{X}_\mathcal{N}$ and any $y \in \mathcal{Y}_\mathcal{P}$,

$$\frac{\mathbf{P}_{uRR}(y|x)}{\mathbf{P}_{uRR}(y|x')} \leq \frac{c_4}{c_3} \leq e^{\epsilon_1} \tag{11}$$

3) For any $x, x' \in \mathcal{X}_\mathcal{S}$ and any $y \in \mathcal{Y}_\mathcal{P}$,

$$\frac{\mathbf{P}_{uRR}(y|x)}{\mathbf{P}_{uRR}(y|x')} \leq \frac{c_1}{c_2} = e^{\epsilon_2} \leq e^{\epsilon_1} \tag{12}$$

4) For any $x \in \mathcal{X}_\mathcal{S}$, $x' \in \mathcal{X}_\mathcal{N}$ and any $y \in \mathcal{Y}_\mathcal{P}$,

$$\frac{\mathbf{P}_{uRR}(y|x)}{\mathbf{P}_{uRR}(y|x')} \leq \frac{c_1}{c_3} = \frac{e^{\epsilon_2}(e^{\epsilon_1} + |\mathcal{X}_\mathcal{S}| - 1)}{e^{\epsilon_2} + |\mathcal{X}_\mathcal{S}| - 1} \leq e^{\epsilon_1} \tag{13}$$

- *Second Approach.* We can demonstrate our protection mechanism as a two-layer model as well: all input data has ϵ_1-differential privacy guaranteed while for sensitive data, another layer of ϵ_2-differential privacy is provided.

– *Layer 1.* If we do not differentiate $\mathcal{X}_\mathcal{S}$ and $\mathcal{X}_\mathcal{N}$, then we have same conclusion as Equation 7 and 8.

1) For any $y \in \mathcal{Y}_\mathcal{I}$, there $\exists x, x' \in \mathcal{X}_\mathcal{N}$

$$\mathbf{P}_{uRR}(y|x) = c_4 \geq 0 \text{ and } \mathbf{P}_{uRR}(y|x') = 0, \forall x' \neq x. \tag{14}$$

2) For any $x, x' \in \mathcal{X}$ and any $y \in \mathcal{Y}_\mathcal{P}$,

$$\frac{\mathbf{P}_{uRR}(y|x)}{\mathbf{P}_{uRR}(y|x')} \leq \frac{c_1}{c_3} \leq e^{\epsilon_1} \tag{15}$$

Therefore, for all input data, ϵ_1-differential privacy is guaranteed.

– Layer 2. According to our definition, $\mathcal{Y}_{\mathcal{P}} \subseteq \mathcal{X}_{\mathcal{S}}$, let $\mathcal{Z}_{\mathcal{P}}$ be the protected output set which follows same definition of $\mathcal{Y}_{\mathcal{P}}$. Thus we have:

3) For any $y, y' \in \mathcal{Y}_{\mathcal{P}} \subseteq \mathcal{X}_{\mathcal{S}}$ and any $z \in \mathcal{Z}_{\mathcal{P}}$

$$\frac{\mathbf{P}_{uRR}(z|y)}{\mathbf{P}_{uRR}(z|y')} \leq \frac{c_1}{c_2} = e^{\epsilon_2} \tag{16}$$

Appendix C. Reversing $(\mathcal{X}_{\mathcal{S}}, \epsilon_1, \epsilon_2)$-URR

Suppose $\left|\overline{X}\right|$ is the total number of observed queries, $\left|\hat{X}\right|$ the estimate and $\left|\overline{X}^N\right|$ is the number of non-sensitive queries being observed. Note that $\left|\overline{X}\right| = \left|\hat{X}\right|$ under LD-PRESOLVE. $\left|\hat{X}^N\right|$ is the estimate of the real non-sensitive queries. According to the definition of $(\mathcal{X}_{\mathcal{S}}, \epsilon_1, \epsilon_2)$-URR, we have:

$$\left|\hat{X}^N\right| = \frac{1}{c_4}\left|\overline{X}^N\right| \tag{17}$$

Therefore, the estimate of real sensitive queries $\left|\hat{X^S}\right|$ would be:

$$\left|\hat{X}^S\right| = |X| - \left|\hat{X}^N\right| = |X| - \frac{1}{c_4}\left|\overline{X}^N\right| \tag{18}$$

For a specific observed sensitive query $\overline{x}_0^S$, the total number of $\overline{x}_0^S$ follows the equation below:

$$\begin{aligned}
\left|\overline{x}_0^S\right| &= \frac{c_1}{c_1+c_2+c_3}\left|\hat{x}_0^S\right| + \frac{c_2}{c_1+c_2+c_3}\sum_{k\neq 0}\left|\hat{x}_k^S\right|\frac{1}{|\mathcal{X}_{\mathcal{S}}|} \\
&+ \frac{c_3}{c_1+c_2+c_3}\left|\hat{X}^N\right|\frac{1}{|\mathcal{X}_{\mathcal{S}}|} \\
&= \frac{c_1}{c_1+c_2+c_3}\left|\hat{x}_0^S\right| + \frac{c_2}{|\mathcal{X}_{\mathcal{S}}|(c_1+c_2+c_3)}(\left|\hat{X}^S\right| - \left|\hat{x}_0^S\right|) \\
&+ \frac{c_3}{c_1+c_2+c_3}\left|\hat{X}^N\right|\frac{1}{|\mathcal{X}_{\mathcal{S}}|} \\
&= \frac{c_1}{c_1+c_2+c_3}\left|\hat{x}_0^S\right| + \frac{c_2}{|\mathcal{X}_{\mathcal{S}}|(c_1+c_2+c_3)}(\left|\overline{X}\right| \\
&- \frac{1}{c_4}\left|\overline{X}^N\right| - \left|\hat{x}_0^S\right|) + \frac{c_3}{c_4(c_1+c_2+c_3)|\mathcal{X}_{\mathcal{S}}|}\left|\overline{X}^N\right|
\end{aligned} \tag{19}$$

Therefore, we have the estimation of any observed domains as follows:

$$\left|\hat{x}_0^S\right| = \frac{\frac{c_2-c_3}{c_4} - c_2\left|\overline{X}\right| + \left|\overline{x}_0^S\right||\mathcal{X}_{\mathcal{S}}|(c_1+c_2+c_3)}{(c_1-c_2)|\mathcal{X}_{\mathcal{S}}|} \tag{20}$$

Attacker will then use $|\hat{x}_0^S|$ for user tracking.