You Are How You Query: Deriving Behavioral Fingerprints from DNS Traffic

Dae Wook Kim(✉) and Junjie Zhang

Wright State University, Dayton, USA
{kim.107,junjie.zhang}@wright.edu

Abstract. As the Domain Name System (DNS) plays an indispensable role in a large number of network applications including those used for malicious purposes, collecting and sharing DNS traffic from real networks are highly desired for a variety of purposes such as measurements and system evaluation. However, information leakage through the collected network traffic raises significant privacy concerns and DNS traffic is not an exception. In this paper, we study a new privacy risk introduced by passively collected DNS traffic. We intend to derive *behavioral fingerprints* from DNS traces, where each behavioral fingerprint targets at uniquely identifying its corresponding user and being immune to the change of time. We have proposed a set of new patterns, which collectively form behavioral fingerprints by characterizing a user's DNS activities through three different perspectives including the domain name, the inter-domain relationship, and domains' temporal behavior. We have also built a distributed system, namely *DNSMiner*, to automatically derive DNS-based behavioral fingerprints from a massive amount of DNS traces. We have performed extensive evaluation based on a large volume of DNS queries collected from a large campus network across two weeks. The evaluation results have demonstrated that a significant percentage of network users with persistent DNS activities are likely to have DNS behavioral fingerprints.

Keywords: Domain Name System · Behavioral fingerprints · Privacy

1 Introduction

The Domain Name System (DNS) plays an indispensable role in the Internet by providing fundamental two-way mapping between domains and Internet Protocol (IP) addresses. Its practical usage has gone far beyond the domain-IP mapping service: it supports many critical network services such as traffic balancing [1] and content delivering [2]; it is also leveraged by attackers to build agile and robust malicious cyber infrastructures, where salient examples include fast-flux [3], random domain generator [4], and covert channels [5]. The importance and prevalence of DNS signifies the demand of its traces collected from real networks, which are essential for many DNS-relevant designs by serving as benchmark data or ground truth. For instance, DNS traces have been collected

© Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2015
B. Thuraisingham et al. (Eds.): SecureComm 2015, LNICST 164, pp. 348–366, 2015.
DOI: 10.1007/978-3-319-28865-9_19

to evaluate DNS cache algorithms [6] and to train statistical models for malicious domain detection [7,8]. Although the specific type and granularity of information extracted from DNS traces may vary for different applications, the demand for DNS traces is generally increasing.

Despite their practical values, DNS traces may introduce significant privacy concerns. For example, DNS queries that are triggered by the prefetching mechanisms of popular browsers can leak users' search engine queries [9]; DNS queries can also reveal the types of operating systems [10]. In this project, we study a new privacy risk introduced by passively collected DNS traffic: to which extent network users can be uniquely identified merely based on the way they issue DNS queries? In other words, we intend to derive *behavioral fingerprints* from DNS traces, where each behavioral fingerprint targets at uniquely identifying its corresponding user and being immune to the change of time. Such DNS-based behavioral fingerprints, once successfully derived, have strong privacy implications. For example, they can be used to de-anonymize the DNS traces with anonymized sources. To be more specific, when DNS traces are shared, the source (e.g., the IP address) that issues the DNS query is usually anonymized (e.g., by obscuring the IP address using hash functions). However, one can learn behavioral fingerprints from un-anonymized DNS traces and use the acquired fingerprints to reveal the presence of specific users in (other) anonymized traces. In addition, if one can get access to DNS traces collected from multiple access networks (e.g., through open DNS services or collecting traces from multiple networks), he/she can track users' locations across different networks by using behavioral fingerprints to reveal users in DNS traces.

This paper aims at investigating the extent to which behavioral fingerprints can be derived and measuring their accuracy on identifying the presence of corresponding network users. As a means towards this end, we have proposed a set of new patterns, which collectively form behavioral fingerprints. We also built a distributed, scalable system, namely *DNSMiner*, to automatically derive DNS-based behavioral fingerprints from a massive amount of DNS traces. Specifically, we make the following contributions in this paper.

- We have designed five new patterns including *domain set*, *domain sequence*, *window-aware domain sequence*, *period behavior*, and *hourly behavior*, which collectively form behavioral fingerprints. These patterns systematically characterize DNS behaviors from three aspects including the domain name, the inter-domain relationship, and the temporal behavior. Although more patterns might be discovered to enhance behavioral fingerprints, our proposed patterns serve as a lower bound of the capabilities to use DNS behaviors to fingerprint network users.
- We have built a system, namely *DNSMiner*, to automatically mine behavioral fingerprints from a massive amount of DNS traces. The design of the system leverages the MapReduce distributed infrastructure to scale up the system performance. After being deployed in a 15-nodes Hadoop platform, *DNSMiner* can process more than 467 million DNS queries using approximately 4 hours.

– We have performed extensive evaluation based on a large volume of DNS queries collected from a large campus network across two weeks.
  The experimental results demonstrated that the behavioral fingerprints derived from a historical DNS stream can effectively identify users in a new DNS stream. To be more specific, 69.63% of users, who have behavioral fingerprints in the historical DNS stream and experience persistent DNS activities in the new DNS stream, can be identified using their behavioral fingerprints. Among these identifiable users, our system accomplishes a high accuracy of 98.74% and a low false positive rate of 1.26%.

The rest of this paper proceeds as follows. Section 2 elaborates the related work. Section 3 shows the system design and Section 4 presents the evaluation results. We discuss the possible limitations and potential solutions in Section 5, and Section 6 concludes.

2 Related Work

Information leakage through collected network data has been recognized as a significant privacy concern, thereby attracting a lot of research efforts. A rich body of literature [11–16] have been proposed to infer application-level users' activities from (encrypted) network traffic. Chen et al. [13] have leveraged communication patterns of HTTP connections to infer the activities taken by browser users. In [14,15], Wright et al. have built statistical models to reveal languages and even spoken phases from encrypted VoIP traffic. Zhang et al. [16] designed a hierarchical classification system to identify users' online activities (i.e., a user's running applications) based on network-level traffic patterns. Sun et al. [11] also created traffic signatures to reveal webpages visited by users in encrypted network traffic. Different from these works that focus on inferring users' activities, our work targets at inferring users' identities.

Pang [17] et al. generated user fingerprints based on encrypted wireless traffic patterns. However, compared to deriving user fingerprints from wireless traffic, fingerprinting users based on DNS traffic is faced with unique challenges since DNS traffic has less semantics. Particularly, although encrypted, the wireless traffic can expose the set of SSIDs, packet sizes, and MAC protocol fields used by a user. Comparatively, DNS queries only make visiable the domain name and the timestamp if the source IP is anonymized. Therefore, how to design effective patterns based on semantic-limited DNS queries becomes the key of our solution. The work closest to ours is [18], where Herrmann et al designed a learning-based approach to attribute sessions of DNS queries to their corresponding users. However, our work significantly differs from the method proposed in [18] from two perspectives. First, a single feature, the visiting frequency of popular domains for each host, was adopted in [18] to characterize users' behaviors while we designed multifaceted features (i.e., total 5 features) to systematically characterize users' behaviors from three different perspectives. Second, the method [18] needs to separate a DNS stream into sessions according to the timestamp of DNS queries,

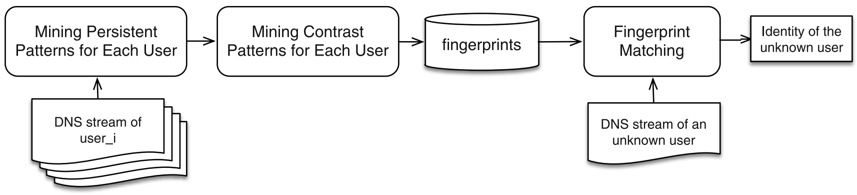
Mining Persistent Patterns for Each User
Mining Contrast Patterns for Each User
fingerprints
Fingerprint Matching
Identity of the unknown user
DNS stream of user_i
DNS stream of an unknown user

Fig. 1. *DNSMiner* architecture

which implies the necessity for fine-grained timing information for DNS queries. Despite the fact that our current implementation also used timestamp for DNS queries, the first pattern (i.e., the domain set pattern) is time-independent; the second and third patterns (i.e., the domain sequence and window-aware domain sequence patterns) only concern the order in which DNS queries are issued in each day. This implies that our mehtod can be used in DNS streams with coarse-grained timing information. In fact, the domain sequence and window-aware domain sequence patterns collaboratively accomplished a high detection rate of 90.72% in our experiment. A few projects [9,10] investigated information leakage from the same type of network traffic used by our work - the passively collected DNS packets. However, their objectives are different from ours. To be specific, Krishnan et al. [9] aimed at recovering search engine queries by investigating correlated domain names and Matsunaka et al. [10] intended to fingerprint operating systems rather than network users.

Several methods [19–21] have been proposed to de-anonymize network data. Specifically, Coull et al. [19] has proposed techniques to de-anonymize network flows by comparing the objects from the unanonymized and anonymized network data directly. Narayanan et al. [20] and Wondracek et al. [21] have leveraged the topology of an unanonymized social network to effectively identify users in an anonymized social network. Despite the fact that our method leverages different data sources, we do not need auxiliary information (e.g., the context of the anonymized data and additional topologies of unanonymized social networks). Nevertheless, DNS behavioral fingerprints extracted by our method complement existing methods [17,19–21].

3 System

The architectural overview of *DNSMiner* is presented in Fig. 1. *DNSMiner* takes as input a set of DNS-query streams, which is denoted as $\mathcal{S} = \{S_1, S_2, \ldots, S_N\}$. Each stream (e.g., S_i) contains DNS queries issued by a user (e.g., u_i) over a certain time period (e.g., several days). A stream is a series of tuples, where each tuple is denoted as $< u, domain, timestamp >$. u, $domain$, and $timestamp$ refer to the user identity, domain name, and the querying time, respectively. In a network where an IP address can be associated with a user, we can use IP addresses to represent users' identities. *DNSMiner* aims at generating a DNS-based behavioral fingerprint, namely $\mathcal{F}_i$, for a user u_i, where $\mathcal{F}_i$ is defined as a finite set of patterns (i.e., $\mathcal{F}_i = \{F_i^1, F_i^2 \dots F_i^K\}$). Each pattern in the fingerprint is named as a *fingerprint pattern*. Ideally, fingerprint patterns should be i) *unique* to their corresponding user (i.e., persistent to their corresponding users) and ii) *immune* to the change of time.

To illustrate the detailed design of *DNSMiner*, we first formulate the mining process of fingerprints (see Section 3.1). Next, we will discuss specific patterns used by *DNSMiner* and the motivations behind their design (see Section 3.2). Finally, we briefly describe the implementation of *DNSMiner* that takes advantage of MapReduce [22] to achieve high scalability (see Section 3.3).

3.1 Problem Formulation

Pattern Mining. *DNSMiner* aims at mining fingerprint patterns that exhibit both significant persistence and uniqueness to a user. Towards this end, we start from defining *persistence* and *uniqueness* of a fingerprint pattern. *DNSMiner* aggregates the DNS stream from a user (e.g., u_i) into a set of transactions (denoted as $\mathcal{T}_i = \{T_i^1, T_i^2, \dots, T_i^M\}$), where each transaction T_i^k is a set of tuples issued by u_i within the same epoch. Since Internet activities usually exhibit strong diurnal patterns [23], we currently use one day to represent an epoch. We denote "T_i^k satisfies F" if the pattern F is observed in T_i^k. The specific meaning of "satisfy" varies for different patterns and we will illustrate it along with the introduction of the patterns. For instance, if F is a set of domains, then T_i^k *satisfies* F when all domains in F are contained in the set of domains that are extracted from all tuples in T_i^k. We introduce a function $mt(F, \mathcal{T}_i)$ that returns all transactions in $\mathcal{T}_i$ that satisfy F. Specifically, $mt(F, \mathcal{T}_i)$ is defined as

$$mt(F, \mathcal{T}_i) = \{T_i^k \in \mathcal{T}_i \mid T_i^k \; satisfies \; F\} \tag{1}$$

We subsequently define a function $supp(F, \mathcal{T}_i)$ to quantify the persistence of a pattern (i.e., F) across the transactions generated by a user u_i. Its formal definition is presented as

$$supp(F, \mathcal{T}_i) = \frac{|mt(F, \mathcal{T}_i)|}{|\mathcal{T}_i|} \tag{2}$$

The $supp(F, \mathcal{T}_i)$ characterizes two trends. If a pattern F is persistent to u_i, $supp(F, \mathcal{T}_i)$ tends to be large. In contrast, a transient pattern is inclined to yield small $supp()$ value. We use a pre-defined threshold, namely α, to discriminate between persistent patterns and transient ones. To be more specific, F is considered to be persistent to u_i if $supp(F, \mathcal{T}_i) \geq \alpha$. We denote the set of persistent patterns for a user u_i as $P(\mathcal{T}_i)$, where $P(\mathcal{T}_i) = \{F | supp(F, \mathcal{T}_i) \geq \alpha\}$.

However, the high persistence of a pattern does not guarantee its uniqueness since a persistent pattern for u_i could also be a persistent pattern for an another user. We therefore define another metric, namely *contrast confidence*, to quantify uniqueness of a persistent pattern (e.g., F) for a user u_i (i.e., how well it F can differentiate u_i from other users).

$$conf(F, \mathcal{T}_i) = \frac{supp(F, \mathcal{T}_i)}{\sum_{F \in P(\mathcal{T}_j)} supp(F, \mathcal{T}_j)}, where\ F \in P(\mathcal{T}_i) \tag{3}$$

$conf(F,\mathcal{T}_i)$ characterizes the following trends: if a pattern is persistent to many users, then its contrast confidence tends to be low; otherwise, its contrast confidence tends to be high. Again, a threshold β is introduced in our current design to differentiate these two trends. A persistent pattern F will be considered as a fingerprint pattern for u_i if $conf(F,\mathcal{T}_i) \geq \beta$.

Pattern Matching. Given an unknown user u_u and his/her associated DNS stream, the pattern matching phase of *DNSMiner* aims at identifying whether this DNS stream can be attributed to any known user. To this end, *DNSMiner* will first follow the same method discussed in Section 3.1 to obtain persistent patterns for u_u. Specifically, we will derive a set of DNS transactions (denoted as $\mathcal{T}_u$) for the unknown user u_u and subsequently identify persistent patterns $P(\mathcal{T}_u)$. It is worth noting that the same criteria for epoch representation (e.g., 24 hours) and the same value of α will be applied. Next, we will evaluate the similarity between an unknown user u_u and a known user u_i, whose fingerprint is denoted as $\mathcal{F}_i$. A distance function, denoted as $dist(u_u, u_i)$, is consequently defined as

$$\begin{aligned} dist(u_u, u_i) \quad &= \quad 1 - \frac{\sum conf(F_i^k, \mathcal{T}_i)}{\sum conf(F_i^j, \mathcal{T}_i)}, \\ &\quad where\ F_i^k \in P(\mathcal{T}_u) \cap \mathcal{F}_i\ and\ F_i^j \in \mathcal{F}_i \end{aligned} \tag{4}$$

$\sum conf(F_i^k, \mathcal{T}_i)$ is the accumulated confidence for all patterns that belong to the intersection of u_i's fingerprint patterns and u_u's persistent patterns; $\sum conf(F_i^j, \mathcal{T}_i)$ is the accumulated confidence for all patterns in u_i's fingerprint. If $P(\mathcal{T}_u) \cap \mathcal{F}_i$ accounts for a large percentage of patterns in $\mathcal{F}_i$, which implies that two users tend to be similar, the distance tends to be small. If multiple users who have fingerprints have non-zero distance wtih u_u, we assign u_u to the user who has the smallest distance.

It is worth noting that a user with transient DNS behaviors may introduce a large volume of noises when discovering persistent patterns. For example, if a user is only active for one epoch (i.e., there is only one transaction for this user), then all of patterns for this user would be persistent since they are active for that transaction, resulting 100% for the *supp*() function. A large number of "persistent" patterns generated by transient users may significantly affect the effectiveness for both pattern generation and matching. In the pattern generation phase, these patterns may drastically decrease the contrast confidence of persistent patterns for persistent IPs. In the pattern matching phase, a transient user is likely to have a large overlap with a known user with respect to their patterns, which implies a false positive. Therefore, in our current design, we only consider those users (or IP addresses) that are sufficiently persistent by themselves. Specifically, if a set of users (or IP addresses) subject to analysis have up to M transactions, our implementation only considers those IP addresses that are active for at least $\frac{M}{2}$ transactions. For example, if a set of IP addresses have up to 7 transactions, we will only analyze their users that are active for at least 4 transactions.

3.2 Patterns

The querying behaviors of DNS are closely related to networking activities of individual users. For example, visiting a website or starting a network application (e.g., an instant messenger) usually triggers the resolution of associated domain(s). The routine and personal networking activities of a user may lead to persistent DNS patterns that are unique to him/her. Based on this intuition, we have designed five types of DNS patterns that characterize a user's DNS querying behaviors from three perspectives, including the domain name (i.e., Pattern 1), the inter-domain relationship (i.e., Pattern 2 and 3), and temporal behavior (Pattern 4 and 5). In this section, we will present the definitions of these patterns and the motivation behind their design.

Pattern 1 - Domain Set: A user may have steady interest for certain websites and use some applications routinely. These activities are likely to result in a set of domains that are repeatedly queried by this user across multiple epochs. Since the interest and application usage patterns are highly personal, the repeatedly queried domains may vary drastically across different network users. We therefore introduce the *domain set* pattern (denoted as F_{domain}), which is simply a set of domains that meets the requirements of persistence and uniqueness. Particularly, a transaction T satisfies the domain name pattern F_{domain} if all domains in F_{domain} are observed in transaction T.

In order to identify F_{domain} ideally, we can enumerate all possible domain set based on all domains derived from each transaction of a user, where the smallest domain set contains a single domain from this transaction and the largest domain set contains all domains in this transaction. We can then evaluate the persistence and uniqueness of these domain sets. Unfortunately, when the number of domains involved in a transaction is large, the sheer volume of domain sets will become overwhelming. In order to solve this problem, we generate domain sets that contain up to N unique domains, where $N = 2$ for our current implementation.

Table 1 presents an illustrative example: two users, u_1 and u_2, are active across five consecutive epochs, resulting in five transactions, respectively. All domains queried by u_1 and u_2 for each epoch are listed in the second and third columns in Table 1. If we configure $\alpha = \frac{3}{5}$, the u_1 has persistent F_{domain} patterns including {a}, {b} and {a,b} since $supp(\{a\}, \mathcal{T}_1) = \frac{|mt(\{a\},\mathcal{T}_1)|}{|\mathcal{T}_1|} = \frac{3}{5} \geq \frac{3}{5}$, $supp(\{b\}, \mathcal{T}_1) = \frac{|mt(\{b\},\mathcal{T}_1)|}{|\mathcal{T}_1|} = \frac{3}{5} \geq \frac{3}{5}$, and $supp(\{a, b\}, \mathcal{T}_1) = \frac{|mt(\{a,b\},\mathcal{T}_1)|}{|\mathcal{T}_1|} = \frac{3}{5} \geq \frac{3}{5}$. Similarly, u_2 will have two persistent patterns including {a} and {b}, where $supp(\{a\}, \mathcal{T}_2) = \frac{|mt(\{a\},\mathcal{T}_2)|}{|\mathcal{T}_2|} = \frac{3}{5} \geq \frac{3}{5}$ and $supp(\{b\}, \mathcal{T}_2) = \frac{|mt(\{b\},\mathcal{T}_2)|}{|\mathcal{T}_2|} =$

Table 1. Transactions and their associated domains for two users across 5 epochs, where {a,b} becomes the *domain set* fingerprint pattern for u_1.

Transaction	Domains		Epoch
	u_1	u_2	
T_1	a, b, c, d	a, c	1
T_2	a, b	a, e	2
T_3	b, a, f, k	a, b, c	3
T_4	e, f	b, k	4
T_5	c, d	b	5

$\frac{3}{5} \geq \frac{3}{5}$. Considering only these two users, it is easy to reach a conclusion that $conf(\{a\}, \mathcal{T}_1) = \frac{1}{2}$, $conf(\{b\}, \mathcal{T}_1) = \frac{1}{2}$, $conf(\{a,b\}, \mathcal{T}_1) = 1$, $conf(\{a\}, \mathcal{T}_2) = \frac{1}{2}$, and $conf(\{b\}, \mathcal{T}_2) = \frac{1}{2}$. If we set $\beta = 60\%$, {a,b} becomes the fingerprint pattern for u_1.

Pattern 2 - Domain Sequence: A network user's routine networking activities could involve his/her individualized preferences and the order in which network activities are carried might be able to reflect such preferences. We consequently define a *domain sequence* pattern denoted as F_{seq}, where F_{seq} is a finite sequence of domains. Given two domains in F_{seq} (i.e., $d_i \in F_{seq}$ and $d_j \in F_{seq}$), $d_i \preceq d_j$ means that d_i is issued before d_j.

Similar to the domain set pattern, the ideal implementation to derive domain sequence patterns should consider domain sequences with all possible lengths derived from a transaction. Unfortunately, the ideal solution could result in a prohibitively huge volume of domain sequence patterns when the number of domains contained in a transaction becomes large. Therefore, we only generate domain sequence patterns composed of two domains. To be more specific, $F_{seq} = (d_i, d_j)$ where $d_i \preceq d_j$ in the transaction.

Compared to domain set patterns, domain sequence patterns offer an additional dimension to differentiate two users. For example, if two users visit `facebook` and `twitter` routinely, they will have two identical F_{domain} patterns (i.e., "www.facebook.com" and "www.twitter.com"). However, if the first user always visits `facebook` before `twitter` while the second user follows the reverse order, *DNSMiner* will generate two disparate persistent domain sequence patterns (i.e., $(www.facebook.com, www.twitter.com)$ and $(www.twitter.com, www.facebook.com)$) for these two users, respectively.

Pattern 3 - Window-Aware Domain Sequence: *DNSMiner* further expands the domain sequence patterns by incorporating the first and last time when a domain is visited. Specifically, rather than considering every possible pairwise sequence for d_i and d_j from all tuples within a transaction, *DNS-Miner* considers the tuples in which d_i and d_j are first and last observed. To this end, we extract a 3-tuple for each domain (e.g., d_i) in a transaction denoted as $< d_i, s_i, e_i >$, where s_i and e_i refer to the first and last time d_i is observed in the transaction, respectively. In order to illustrate the design of this

Table 2. Window-Aware Patterns

Window-Aware Patterns	p_*'s Value
$< d_i, d_j, ss, p_1 >$	if($s_i < s_j$) $p_1 = 0$; else $p_1 = 1$;
$< d_i, d_j, se, p_2 >$	if($s_i < e_j$) $p_2 = 0$; else $p_2 = 1$;
$< d_i, d_j, es, p_3 >$	if($e_i < s_j$) $p_3 = 0$; else $p_3 = 1$;
$< d_i, d_j, ee, p_4 >$	if($e_i < e_j$) $p_4 = 0$; else $p_4 = 1$;

Table 3. A sequence of DNS queries

Timestamp	t_0	t_1	t_2	t_3	t_4	t_5
Domain	a	b	a	b	b	a

pattern, we consider two domains, d_i and d_j, whose 3-tuples are $< d_i, s_i, e_i >$ and $< d_j, s_j, e_j >$, respectively. Without loss of generality, we assume that d_i alphabetically precedes d_j. The comparison of both starting and ending times of these two domains will result in four 4-tuples as illustrated in Table 2. The third element in a 4-tuple indicates how two domains are compared. For example, "*ss*" indicates that d_i's starting time is compared to d_j's starting time and "*se*" indicates the comparison between d_i's starting time and d_j's ending time. The second column in Table 2 shows rules we have used to assign values for the fourth variable. It is worth noting that these four window-aware sequence patterns might not be independent. For example, if p_3 in $< d_i, d_j, es, p_3 >$ is 0, which means that the last time we observe d_i precedes the first time we observe d_j, then all p_* variables in other 4-tuples for d_i and d_j will always be 0. We exploit such dependency in our implementation to reduce the number of patterns yielded for each pair of domains.

Table 3 illustrates a series of domains queried by a user together with their timestamps, where all these domains belong to one transaction and $t_0 < t_1 \ldots t_4 < t_5$. For this user, two 3-tuples in the form of $< d_i, s_i, e_i >$ will be derived, including $< a, t_0, t_5 >$ and $< b, t_1, t_4 >$. For example, $< a, t_0, t_5 >$ indicates that the domain a is first and last queried in this transaction at t_0 and t_5, respectively. We follow the definition of window-aware patterns as indicated in Table 2 to derive four window-aware patterns for this example, which includes $< a, b, ss, 0 >$, $< a, b, se, 0 >$, $< a, b, es, 1 >$, and $< a, b, ee, 1 >$. As indicated in this example, some patterns may imply others, making it possible to simplify the generation of window-aware patterns. For example, if we know $< a, b, ss, 0 >$, we can directly conclude that $< a, b, se, 0 >$ without generating it from data.

Pattern 4 - Period Behavior: Network users' networking activities often exhibit strong temporal patterns. For example, a user could visit a news website every morning while an another user surfs it over every afternoon. Consequently, each domain together with its temporal information may well represent a user. We therefore introduce the *Period Behavior* pattern (denoted as F_{period}), which is defined as a domain-period combination. The "period" refers to a tag indicating "morning", "afternoon", and "evening". In order to derive such

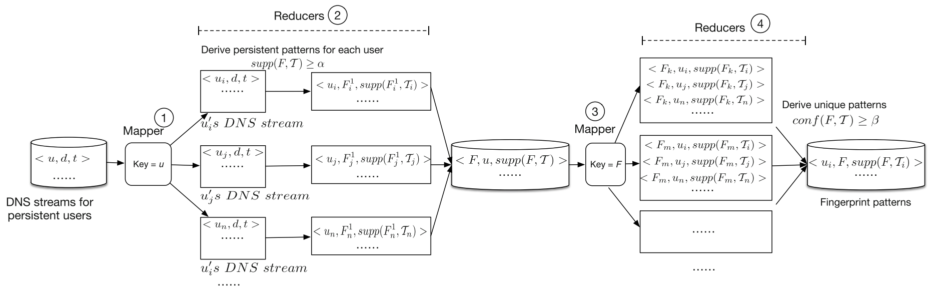
Reducers ②
Derive persistent patterns for each user
$supp(F, \mathcal{T}) \geq \alpha$
$< u, d, t >$
DNS streams for persistent users
① Mapper
Key = u
$< u_i, d, t >$
$u_i's\ DNS\ stream$
$< u_j, d, t >$
$u_j's\ DNS\ stream$
$< u_n, d, t >$
$u_i's\ DNS\ stream$
$< u_i, F_i^1, supp(F_i^1, \mathcal{T}_i) >$
$< u_j, F_j^1, supp(F_j^1, \mathcal{T}_j) >$
$< u_n, F_n^1, supp(F_n^1, \mathcal{T}_n) >$
$< F, u, supp(F, \mathcal{T}) >$
③ Mapper
Key = F
Reducers ④
$< F_k, u_i, supp(F_k, \mathcal{T}_i) >$
$< F_k, u_j, supp(F_k, \mathcal{T}_j) >$
$< F_k, u_n, supp(F_k, \mathcal{T}_n) >$
$< F_m, u_i, supp(F_m, \mathcal{T}_i) >$
$< F_m, u_j, supp(F_m, \mathcal{T}_j) >$
$< F_m, u_n, supp(F_m, \mathcal{T}_n) >$
Derive unique patterns
$conf(F, \mathcal{T}) \geq \beta$
$< u_i, F, supp(F, \mathcal{T}_i) >$
Fingerprint patterns

Fig. 2. *DNSMiner* implementation of identifying persistent patterns

pattern, we first map the timestamp of each tuple into one of three period tags, where "morning", "afternoon", and "evening" stand for [5:00AM, 11:00AM), [11:00AM, 5:00PM), and [5:00PM, 5:00AM), respectively. Next, for each tuple, we integrate its domain and its corresponding period tag into a domain-period combination. For example, *(www.facebook.com, 2013-09-17 08:30:23)*, a tuple in a DNS stream, will generate *(www.facebook.com, morning)* as its Period Behavior pattern.

Pattern 5 - Hourly Behavior: We further introduce the *Hourly Behavior* pattern to characterize a user's networking activities at a finer granularity. Rather than mapping a timestamp into a period tag, *DNSMiner* maps a timestamp to its corresponding hour, thereby leading to a domain-hour combination denoted as F_{hourly}. For instance, the tuple *(www.facebook.com, 2013-09-17 08:30:23)* will be mapped into *(www.facebook.com, 08)*.

3.3 System Implementation

A network user may generate a large number of DNS queries. As the number of network users increases, the scalability of *DNSMiner* becomes a concern. To address the challenge, we have implemented *DNSMiner* using the Hadoop MapReduce platform. The two phases of Map and Reduce workflows in the implementation are presented in Fig. 2. *DNSMiner* first identifies persistent patterns for each user. Since the identification of persistent patterns for each user is independent to that for other users, we can easily parallelize the computation by partitioning/mapping tuples (i.e., $< uid, domain, timestamp >$) into reducers based on their *uid*s (i.e., the step ① in Fig. 2). Each reducer will then enumerate all patterns for each transaction of u_i; for each derived pattern F_i^j, its *supp*() value in the context of u_i will be subsequently calculated; we consequently apply the predefined threshold α and preserve all persistent patterns (i.e., patterns whose *supp*() values are greater than α). These three actions together are performed in reducers for the step ② in Fig. 2. Next, we partition patterns together with their associated *uid*s and *supp*() values into reducers, where the

Table 4. The # of IPs in D_1 and D_2, # of persistent IPs ($|P_1|$ and $|P_2|$), # of IPs with persistent patterns in P_1 and P_2, and # of IPs with fingerprint patterns in D_1 (i.e., $|FP_1|$)

Week	# of IPs	# of Persistent IPs	# of IPs with Persistent Patterns	# of IPs with Fingerprint Patterns
Week 1 (D_1)	55,459	16,003	12,900	11,921
Week 2 (D_2)	54,751	9,120	7,119	-

pattern serves as the key (the step ③ in Fig. 2). Finally, each reducer will calculate the contrast confidence for each pattern with respect to each user and yield those unique ones in the step ④ (e.g., $conf(F, \mathcal{T}) \geq \beta$).

4 Experiments

We have evaluated *DNSMiner* using DNS queries collected from a large campus network. Our evaluation aims at answering three questions: "Can DNS-based fingerprints effectively identify their corresponding network users?", "How do parameter values impact *DNSMiner*'s effectiveness?", and "How effective is each category of patterns?".

4.1 Data and Experiment Setup

We obtained DNS queries collected from a large campus network of Xi'an Jiaotong University, China, where the DNS queries are collected below the major recursive DNS servers used by the campus network. Aiming at facilitating the network management, the campus network assigns *static* IP addresses to the vast majority of its users after they register at the network management center. Only a few buildings use dynamic IP addresses and we have excluded DNS queries issued from their corresponding subnets. Sensors were deployed to collect DNS queries that are issued by all hosts in campus network. For each DNS query, three pieces of information were extracted, including the domain name, the timestamp, and the IP address that issues this query. We collected two sets of DNS queries from two consecutive weeks at September 2013, which are denoted as D_1 and D_2, respectively. As illustrated in the second column of Table 4, D_1 and D_2 contain 55,459 and 54,751 unique IP addresses, respectively. Both D_1 and D_2 contain a large number of DNS queries (i.e., 467,388,490 queries in D_1 and 238,993,575 in D_2).

As Internet activities typically show diurnal patterns [23,24], we considers one day as one epoch. Specifically, an epoch starts from 5:00AM and lasts for 24 hours. Both transaction-sets for fingerprint extraction and matching contain 7 epochs (i.e., for 7 consecutive days). We configure $\alpha = \frac{5}{7}$, which means that a fingerprint pattern has to be persistent for at least $\frac{5}{7}$ out of the active days for its corresponding IP address. We also set $\beta = 60\%$.

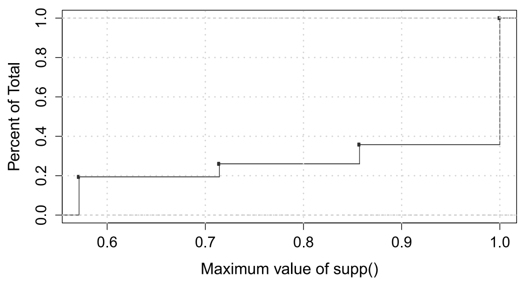

Fig. 3. The CDF distribution of the maximum *supp*() for all persistent IP addresses in D_1 (i.e., all IPs in P_1). A significant percentage (64.18%) of IPs in P_1 have patterns that are persistent across 7 epochs.

We use the queries of the first 7 days (i.e., D_1) to derive DNS fingerprints and those of the remaining week (i.e., D_2) to evaluate the extent to which the fingerprints can effectively de-anonymize users in a new DNS stream. As we have discussed in Section 3, IP addresses with transient DNS behaviors are likely to introduce noises. Therefore, we only consider those IP addresses that experience sufficient persistence by themselves. Specifically, since D_1 and D_2 contain up to 7 transactions, we preserve those IP addresses that are active for at least half of the 7 transactions (i.e., for at least 4 transactions). We use P_1 to represent a set of persistent IPs in D_1 and P_2 in D_2. As illustrated in Table 4, P_1 and P_2 contain $16,003$ and $9,120$ IP addresses, respectively.

4.2 Fingerprint Extraction

The first step of *DNSMiner* is to assess the persistence of patterns for each IP address in P_1. Specifically, for each IP address in P_1, we extract all of its patterns, investigate their *supp*() values, and preserve those whose *supp*() values are greater than the predefined threshold α. We identify the maximum *supp*() value for each IP address and plot the distribution of maximum *supp*() value for all IPs in P_1 in Fig. 3. As illustrated in the distribution, a significantly large percentage of persistent IPs (i.e., IPs in P_1) indeed have persistent patterns. Particularly, 64.18% of IPs in P_1 have the maximum *supp*() value of 1, indicating that each of these IPs has repeatedly shown at least one pattern across entire 7 epochs. In addition, a large percentage of 80.60% of IPs in P_1 have at least one persistent pattern whose *supp*() value is greater than $\alpha = \frac{5}{7}$. This results in 12,900 IPs with persistent patterns in P_1, which account for totally 313,248,287 persistent patterns.

The second step of *DNSMiner* is to investigate the uniqueness of persistent patterns based on their contrast confidence (i.e., $conf(F_i^j, \mathcal{T}_i)$). Again, $conf(F_i^j, \mathcal{T}_i)$ quantifies the uniqueness of a pattern F_i^j to its corresponding user u_i. In order to visualize the experiment results, for each IP with persistent patterns, we derive the highest contrast confidence for all its persistent patterns; we then present the distribution of the highest contrast confidence values for

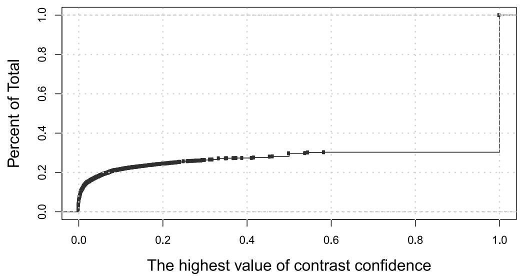

Fig. 4. The CDF distribution of the highest contrast confidence for each IP address that has at least one persistent pattern. Approximately 70% have *unique* persistent patterns (i.e., with contrast confidence of 1).

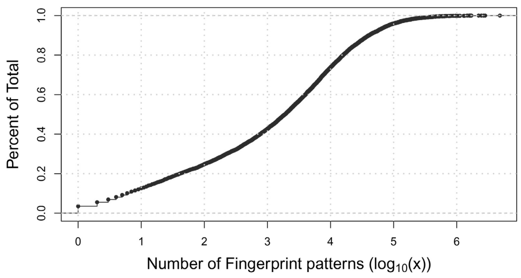

Fig. 5. The CDF distribution of the number of fingerprint patterns for each IP address. IP addresses with fingerprint patterns tend to have a large number of fingerprint patterns.

these IPs in Fig. 4. As illustrated in Fig. 4, about 70% percentage of IPs with persistent patterns have patterns whose contrast confidence is 1, which indicates that these patterns are unique for their corresponding users. In *DNSMiner*, we use the predefined threshold $\beta = 60\%$ to further identify those persistent that also experience significant uniqueness (i.e., fingerprint patterns). Totally, *DNS-Miner* has identified 11,921 IP addresses that have fingerprint patterns, where these IP addresses form a set namely FP_1 and $FP_1 \subseteq P_1$. *DNSMiner* totally generated 222,508,026 fingerprint patterns, among which the domain set pattern, the domain sequence pattern, the window-aware domain sequence pattern, the period pattern, and hourly behavior pattern account for 16.43%, 11%, 72.51%, 0.02%, and 0.04%, respectively. We count the total number of fingerprint patterns for each IP address and plot their distribution in Fig. 5. The distribution indicates that these IPs tend to have a large number of DNS fingerprint patterns, implying strongly discriminative DNS behaviors. Particularly, more than 78% of IP addresses in FP_1 have at least 100 fingerprint patterns.

Table 5. The accuracy of identifying users in a new DNS stream D_2 using fingerprint patterns extracted from a historical DNS stream D_1. Among 69.63% IPs that are identified by fingerprint patterns, 98.74% are correctly revealed.

Week	$\|P_2\|$	$\|FP_1 \cap P_2\|$	$\|K\|$	$\|KC\|$	$\|KI\|$	II(%)	DR(%)	FP(%)
Week 2 (D_2)	9,120	4,894	3,408	3,365	43	69.63	98.74	1.26

4.3 Fingerprint Matching

As introduced in Section 4.2, FP_1 represents a set of IPs in D_1 whose DNS behavioral fingerprints have been derived by *DNSMiner*. We also use P_1 and P_2 to represent sets of persistent IPs for D_1 and D_2, respectively. For fingerprint matching, *our objective is to use fingerprint patterns for IPs in FP_1 to reveal their presence in P_2*. Specifically, we perform the pattern matching as discussed in Section 3 to identify all IPs in P_2 whose distance (i.e., $dist(u_u, u_i)$) is smaller than 1 compared to any IP in FP_1, where these IPs together form a set named as K. K can be further divided into two sets, namely KC and KI, which represent the IPs that are correctly and incorrectly identified, respectively (i.e., $K = KC \cup KI$). Subsequently, we define the following three metrics to quantify the effectiveness of fingerprint patterns.

- The percentage of identified IP addresses (**II**): $\frac{|K|}{|FP_1 \cap P_2|}$. We expect *DNSMiner* to identify all IPs in $FP_1 \cap P_2$ since IPs in $FP_1 \cap P_2$ indeed have fingerprint patterns in the first week and are persistent in the second week. $\frac{|K|}{|FP_1 \cap P_2|}$ represents the overall effectiveness on identifying IPs in a new DNS stream.
- The detection rate: $\frac{|KC|}{|K|}$ (**DR**). This ration shows the ratio of the number of correctly identified IPs over the number of all identified IPs.
- The false positive rate: $\frac{|KI|}{|K|}$ (**FP**). This ration shows the ratio of the number of incorectly identified IPs over the number of all identified IPs.

We have performed the evaluation of fingerprint matching using the DNS stream of D_2, where the evaluation results are presented in Table 5. Specifically, 4,894 IPs in P_2 (i.e., persisent IPs in the second week) have fingerprint patterns in the first week (i.e., $|FP_1 \cap P_2| = 4{,}894$). In other words, the ideal objective is to identify all these 4,894 IPs in the DNS stream of the second week (i.e., D_2) using their fingerprint patterns extracted from the first week (i.e., D_1). The matching results show that totally 3,408 IPs have been identified, resulting in the percentage of identified IPs of 69.63%. Among these 3,408 IP addresses, 3,365 IPs are correctly attributed to those IPs in FP_1, resulting a high detection rate of 98.74% and a low false positive rate of 1.26%.

We have deployed *DNSMiner* on a Hadoop platform with 15 nodes. The entire process for both extracting and matching fingerprint patterns consumes approximately 4 hours.

Table 6. The detection performance under different α and β values. "PI" indicates the percentage of IPs in P_1 that have fingerprint patterns; "II" is denoted as the percentage of IPs in $FP_1 \cap P_2$ that are detected by fingerprint patterns; "DR" and "FP" refer to the detection rate and false positive rate, respectively.

Parameter	$\alpha = 4/7$				$\alpha = 5/7$			
	PI(%)	II(%)	DR(%)	FP(%)	PI(%)	II(%)	DR(%)	FP(%)
$\beta = 30\%$	74.04	68.91	92.42	7.58	72.48	68.16	94.17	5.83
40%	73.28	69.45	93.33	6.67	71.29	68.70	95.81	4.19
50%	72.66	70.02	96.09	3.91	70.95	69.26	98.56	1.44
60%	72.35	70.39	96.25	3.75	70.82	69.63	98.74	1.26
70%	71.98	70.43	95.79	4.21	70.65	69.69	97.45	2.55
80%	71.86	70.49	94.78	5.22	70.36	69.77	96.16	3.84
	$\alpha = 6/7$				$\alpha = 7/7$			
	PI(%)	II(%)	DR(%)	FP(%)	PI(%)	II(%)	DR(%)	FP(%)
$\beta = 30\%$	68.98	62.00	90.19	9.81	48.02	43.80	87.80	12.20
40%	67.66	62.50	91.74	8.26	47.53	44.15	88.85	11.15
50%	66.56	63.01	93.93	6.07	47.46	44.51	90.74	9.26
60%	66.29	63.34	94.04	5.96	47.34	44.74	90.82	9.18
70%	65.80	63.42	93.83	6.17	47.27	44.83	89.37	10.63
80%	65.01	63.68	93.07	6.93	47.15	44.99	89.14	10.86

4.4 Evaluating the Impact of Parameter Values

DNSMiner needs two parameters including α and β to be configured. While the evaluation result based on the current configuration ($\alpha = \frac{5}{7}$ and $\beta = 60\%$) yields a high detection rate, we further investigate how parameter values affect the system effectiveness. Specifically, we assign a wide range of values to α (i.e., $\alpha = \frac{4}{7}, \frac{5}{7}, \frac{6}{7}, \frac{7}{7}$) and β (i.e., $\beta = 30\%, 40\%, 50\%, 60\%, 70\%, 80\%$) and then perform the fingerprint extraction and matching for each combination of α' and β' values. The experimental results are summarized in Table 6, where each cell in the table contains i) the percentage of IPs in P_1 that have fingerprint patterns (i.e., $\frac{|FP_1|}{|P_1|}$), ii) the percentage of identified IPs (i.e., $\frac{|K|}{|FP_1 \cap P_2|}$), iii) the detection rate (i.e., $\frac{|KC|}{|K|}$), and iv) the false positive rate (i.e., $\frac{|KI|}{|K|}$). Fig. 6 visualizes the trend of detection rates when α increases from $\frac{4}{7}$ to $\frac{7}{7}$ for a fixed value of β.

As indicated by the experimental results, when both α and β increase, the percentage of IPs in P_1 that have fingerprint patterns drops. For example, 74.04% of IPs in P_1 have fingerprint patterns given $\alpha = \frac{4}{7}$ and $\beta = 30\%$ while the percentage is 47.15% given $\alpha = \frac{7}{7}$ and $\beta = 80\%$. The changes of α and β affect K and FP_1 simultaneously, thereby impacting the percentage of identified IP addresses (i.e., $\frac{|K|}{|FP_1 \cap P_2|}$). This measure stays very stable (i.e., close to 70%) when $\alpha = \frac{4}{7}, \frac{5}{7}$ and all β values under investigation. When $\alpha \geq \frac{6}{7}$, this measure drops significantly (i.e., around 63% for $\alpha = \frac{6}{7}$ and 44% for $\alpha = \frac{7}{7}$). Despite the fluctuation of the percentage of persistent IPs with fingerprint patterns and the percentage of identified IP addresses along with the changes of α and β, *DNSMiner*

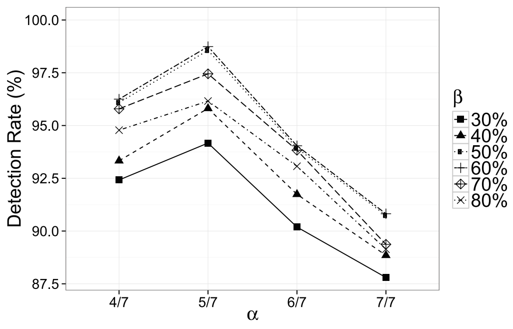

Fig. 6. The trend of detection rates when α increases given a fixed value for β, where *DNSMiner* achieves the best accuracy of 98.74% when $\alpha = 5/7$ and $\beta = 60\%$)

Table 7. The detection performance of *DNSMiner* for each category of patterns: "Domain name" refers to the domain set pattern; "Inter-domain relationship" includes the domain sequence pattern and window-aware domain sequence pattern; "Temporal behavior" contains period and hourly behavior pattern.

Pattern Category	II(%)	DR(%)	FP(%)
Domain name	29.65	85.83	14.17
Inter-domain relationship	43.21	90.72	9.28
Temporal behavior	32.50	87.90	12.10

accomplishes high detection performance. Specifically, for all combinations of α and β values in our experiments, the detection rates are above 87.80%. Particularly, when we configure $\frac{4}{7} \leq \alpha \leq \frac{6}{7}$, all β values lead to detection rates higher than 90%. Such experiment results imply that our method accomplishes the high detection accuracy over a wide range of parameter values. Nevertheless, considering the percentage of users with fingerprint patterns (i.e., "PI") and the percentage of identified IPs (i.e., "II"), $\alpha \in [\frac{4}{7}, \frac{5}{7}]$ and $\beta \in [40\%, 70\%]$ yield the best detection performance with approximately (i.e., approximately 70% for both "PI" and "II", and detection rates higher than 95%).

We have also investigated the detection performance of *DNSMiner* when only a category of patterns are used and the experiment results are presented in Table 7, where $\alpha = \frac{5}{7}$ and $\beta = 60\%$. As indicated in Table 7, patterns belonging to the category of the inter-domain relationship resulted in the best detection rates (i.e., a detection rate of 90.72% and a false positive rate of 9.28%) compared to patterns in the other two categories. Nevertheless, all these patterns collectively accomplish the best detection performance as indicated in Table 6, indicating that all patterns complement each other in *DNSMiner*.

5 Discussion

DNSMiner currently concentrates on network users whose DNS activities are persistent. For example, network users who were active for at least 4 days out of 7 days were considered in our experiments. Despite the fact that such design mitigates the noises caused by network users with transit DNS activities, it may actually result in limitations for the practical usage of *DNSMiner*. First, *DNSMiner* by design cannot generate fingerprint patterns for those network users with transit DNS activities. Second, *DNSMiner* requires that DNS queries can be attributed to their corresponding users over a relative long period (e.g., across the epochs for fingerprint generation). Specifically, when we use an IP address to represent a user, the IP address should not change across the epochs for pattern generation and matching. For networks using static IP addresses, this limitation can be easily overcome, which is actually the case for our evaluation. However, when the IP address associated with a user changes frequently (e.g., in networks that use dynamic IPs with small lease time), it becomes a challenging problem to directly attribute IP addresses to their corresponding users across a series of epochs.

We acknowledge such limitations in the current design and our future work will focus on systematically addressing them. Specifically, a few potential improvements can be explored. First, we plan to design an algorithm that can adaptively define epochs for each IP address and aggregate them into transaction set according to the DNS activities of this IP address. Particularly, the transaction set will be discovered in a way that it is very unlikely for the host to change its IP address across the epochs belonging to this transaction set. Second, rather than manually defining fingerprint patterns, we intend to propose methods that can automatically generate patterns and perform pattern selection. Particularly, we expect that the patterns will give more weight on characterizing the short-term DNS activities of a user.

6 Conclusion

This paper presents a novel system, *DNSMiner*, to automatically derive behavioral fingerprints from DNS queries, where behavioral fingerprints are expected to reveal the presence of their corresponding users in new DNS streams whose identities are unknown (e.g., anonymized). A behavioral fingerprint is composed of a collection of patterns that systematically characterize each user's DNS activities from three different perspectives including the domain name, the inter-domain relationship, and the temporal behavior. The extensive evaluation based on DNS queries collected from a large campus network has demonstrated that these patterns can accomplish a high detection accuracy of 98.74% and a low false positive rate of 1.26%. Despite its high detection accuracy, more patterns could be discovered and incorporated into *DNSMiner*. Nevertheless, *DNSMiner* demonstrates the lower bound of the effectiveness of using DNS-based patterns to reveal users' presence in network traffic.

References

1. Shaikh, A., Tewari, R., Agrawal, M.: On the effectiveness of dns-based server selection. In: INFOCOM (2001)
2. Vakali, A., Pallis, G.: Content delivery networks: Status and trends. IEEE Internet Computing **7**(6), 68–74 (2003)
3. Holz, T., Gorecki, C., Rieck, K., Freiling, F.C.: Measuring and detecting fast-flux service networks. In: NDSS (2008)
4. Antonakakis, M., Perdisci, R., Nadji, Y., Vasiloglou II, N., Abu-Nimeh, S., Lee, W., Dagon, D.: From throw-away traffic to bots: detecting the rise of dga-based malware. In: USENIX Security Symposium (2012)
5. Paxson, V., Christodorescu, M., Javed, M., Rao, J.R., Sailer, R., Schales, D.L., Stoecklin, M.P., Thomas, K., Venema, W., Weaver, N.: Practical comprehensive bounds on surreptitious communication over dns. In: USENIX Security (2013)
6. Jung, J., Sit, E., Balakrishnan, H., Morris, R.: Dns performance and the effectiveness of caching. IEEE/ACM Transactions on Networking **10**(5), 589–603 (2002)
7. Bilge, L., Kirda, E., Kruegel, C., Balduzzi, M.: Exposure: finding malicious domains using passive dns analysis. In: NDSS (2011)
8. Antonakakis, M., Perdisci, R., Lee, W., Vasiloglou II, N., Dagon, D.: Detecting malware domains at the upper dns hierarchy. In: USENIX Security Symposium (2011)
9. Krishnan, S., Monrose, F.: Dns prefetching and its privacy implications: when good things go bad. In: Proceedings of the 3rd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More. USENIX Association (2010)
10. Matsunaka, T., Yamada, A., Kubota, A.: Passive os fingerprinting by dns traffic analysis. In: 2013 IEEE 27th International Conference on AINA (2013)
11. Sun, Q., Simon, D.R., Wang, Y.-M., Russell, W., Padmanabhan, V.N., Qiu, L.: Statistical identification of encrypted web browsing traffic. In: Proceedings 2002 IEEE Symposium on Security and Privacy, pp. 19–30. IEEE (2002)
12. Liberatore, M., Levine, B.N.: Inferring the source of encrypted http connections. In: Proceedings of the 13th ACM Conference on Computer and Communications Security (2006)
13. Chen, S., Wang, R., Wang, X., Zhang, K.: Side-channel leaks in web applications: a reality today, a challenge tomorrow. In: 2010 IEEE Symposium on Security and Privacy (SP), pp. 191–206. IEEE (2010)
14. Wright, C.V., Ballard, L., Monrose, F., Masson, G.M.: Language identification of encrypted voip traffic: Alejandra y roberto or alice and bob. In: Proceedings of USENIX Security Symposium (2007)
15. Wright, C.V., Ballard, L., Coull, S.E., Monrose, F., Masson, G.M.: Spot me if you can: uncovering spoken phrases in encrypted voip conversations. In: IEEE Symposium on Security and Privacy, SP 2008. IEEE (2008)
16. Zhang, F., He, W., Liu, X., Bridges, P.G.: Inferring users' online activities through traffic analysis. In: Proceedings of WiSec (2011)
17. Pang, J., Greenstein, B., Gummadi, R., Seshan, S., Wetherall, D.: 802.11 user fingerprinting. In: MobiCom (2007)
18. Herrmann, D., Banse, C., Federrath, H.: Behavior-based tracking: Exploiting characteristic patterns in dns traffic. Computers & Security **39**, 17–33 (2013)

19. Coull, S.E., Wright, C.V., Keromytis, A.D., Monrose, F., Reiter, M.K.: Taming the devil: techniques for evaluating anonymized network data. In: Proceedings Network and Distributed System Security Symposium 2008, February, 10–13, San Diego, California, pp. 125–135. Internet Society 2008 (2008)
20. Wondracek, G., Holz, T., Kirda, E., Kruegel, C.: A practical attack to de-anonymize social network users. In: 2010 IEEE Symposium on Security and Privacy (SP) (2010)
21. Narayanan, A., Shmatikov, V.: De-anonymizing social networks. In: 2009 30th IEEE Symposium on Security and Privacy, pp. 173–187, May 2009
22. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Communications of the ACM **51**(1), 107–113 (2008)
23. Shafiq, M.Z., Ji, L., Liu, A.X., Wang, J.: Characterizing and modeling internet traffic dynamics of cellular devices. In: ACM SIGMETRICS (2011)
24. Dagon, D., Zou, C., Lee, W.: Modeling botnet propagation using time zones. In: NDSS (2006)