# SFMap: Inferring Services over Encrypted Web Flows Using Dynamical Domain Name Graphs

Tatsuya Mori[1](✉), Takeru Inoue[2], Akihiro Shimoda[3], Kazumichi Sato[3], Keisuke Ishibashi[3], and Shigeki Goto[1]

[1] Department of Computer Science and Communications Engineering, Waseda University, Tokyo, Japan
mori@nsl.cs.waseda.ac.jp

[2] NTT Network Innovation Laboratories, NTT Corporation, Tokyo, Japan

[3] NTT Network Technology Laboratories, NTT Corporation,Tokyo, Japan

**Abstract.** Most modern Internet services are carried over the web. A significant amount of web transactions is now encrypted and the transition to encryption has made it difficult for network operators to understand traffic mix. The goal of this study is to enable network operators to infer hostnames within HTTPS traffic because hostname information is useful to understand the breakdown of encrypted web traffic. The proposed approach correlates HTTPS flows and DNS queries/responses. Although this approach may appear trivial, recent deployment and implementation of DNS ecosystems have made it a challenging research problem; i.e., canonical name tricks used by CDNs, the dynamic and diverse nature of DNS TTL settings, and incomplete measurements due to the existence of various caching mechanisms. To tackle these challenges, we introduce domain name graph (DNG), which is a formal expression that characterizes the highly dynamic and diverse nature of DNS mechanisms. Furthermore, we have developed a framework called Service-Flow map (SFMap) that works on top of the DNG. SFMap statistically estimates the hostname of an HTTPS server, given a pair of client and server IP addresses. We evaluate the performance of SFMap through extensive analysis using real packet traces collected from two locations with different scales. We demonstrate that SFMap establishes good estimation accuracies and outperforms a state-of-the-art approach.

## 1 Introduction

**Background**:
Monitoring and understanding traffic mix is crucial for network operators. Port number conventions and deep packet inspection (DPI) are widely used to understand the breakdown of traffic mix. However, these techniques have become less effective for the following reasons. First, the majority of modern services, such as social networking service, video, and messaging services, are all performed over web traffic [9], and port number information is too coarse-grained to distinguish such services from each other. Second, the encryption of communication channels has disabled inspection of HTTP headers, which include useful information such as uniform resource identifiers (URIs). Modern protocols for accelerating the web such

© IFIP International Federation for Information Processing 2015
M. Steiner et al. (Eds.): TMA 2015, LNCS 9053, pp. 126–139, 2015.
DOI: 10.1007/978-3-319-17172-2_9

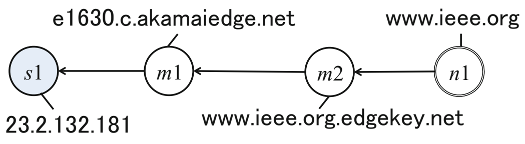

**Fig. 1.** Example of a CNAME chain

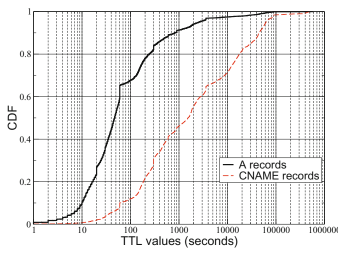

**Fig. 2.** CDFs of TTL values

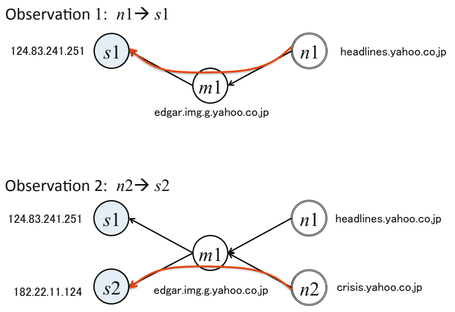

**Fig. 3.** An example of CNAME ambiguity

as SPDY and Websocket employ mandatory encryption of HTTP with SSL/TLS (secure socket layer/transport layer security), i.e., HTTPS. Naylor et al. [7] recently reported that fraction of HTTPS traffic volume measured at a large-scale ISP has significantly increased over these 2+ years (from April 2012 to July 2014). They also found that their meausrement study suggests that cost of deploying HTTPS is decreasing. Hence, the increasing adoption of HTTPS brings new research challenges to traffic classification problems [2,5][1].

**Goal and Challenges**:
Based on the aforementioned information, this work aims to enable network operators to infer the hostnames of HTTPS traffic. Hostname information is useful for network operators to understand what types of services are carried over HTTPS flows. Although the IP address property of an HTTPS server may reveal that the server is used by a particular company such as Google, this information often fails to provide us with information about the services that are used over the flow, such as web searches, blogs, and videos. Such services are associated with distinct hostnames such as www.google.com, www.blogspot.com, and www.youtube.com. Bermudez et al. [2] revealed that simple reverse DNS lookup does not return accurate domain

[1] We note that server name indication (SNI) extention of TLS can be used to obtain hostname of HTTPS server. However, there are many client/server implementations that do not adopt SNI. In fact, in our dataset, roughly half of HTTPS clients did not use the SNI extention.

information used by HTTPS servers. Thus, to understand the traffic mix of HTTPS flows, we need to infer server hostnames.

The main idea of our approach is to correlate HTTPS flows and DNS queries/responses. The basic assumption is that prior to requesting an HTTPS flow, a web application should resolve the IP address of the HTTPS server by querying a DNS query. Therefore, by monitoring prior DNS queries/responses, we can estimate the hostname that is associated with IP address of the HTTPS server. Although this approach might look trivial, there are three practical challenges.

(**Challenge 1**) *Canonical name (CNAME) tricks used by CDNs*

First, modern CDN providers leverage CNAME tricks to accelerate the efficiency of content delivery [10]. Figure 1 shows an example of a CNAME chain used by a CDN provider. Here, assume that we know that the IP address of an observed HTTPS server is $s1 = 23.2.132.181$. Now, our task is to associate $s1$ with the original hostname, $n1 =$ www.ieee.org. However, as is shown in Fig. 1, $n1$ is not directly resolved to $s1$ due to the existence of the CNAME chain. Using this chain structure, a CDN provider can provide the optimal server IP address $s1$ to serve the content of $n1$ to client $c1$. Thus, to associate $s1$ and $n1$, we need to keep track of the CNAME chain, which exhibits dynamic and complex behavior as we shall see soon.

(**Challenge 2**) *Incomplete measurements*

A DNS record can be cached by several mechanisms such as local DNS resolvers, DNS caching within operating systems, and DNS caching within applications such as web browsers. The implementations of these caching mechanisms are diverse. Some recent implementations used in web browsers store DNS records aggressively to improve response time, thereby ignoring DNS TTL settings [4]. Even though such implementations violate the rule of DNS TTL, they can work because even if a selected server IP address is no longer an optimal one, the server IP address generally continues to be valid. Thus, due to the standard and illicit caching mechanisms, a DNS query, which should have appeared prior to an HTTP request, is often invisible. The absence of DNS queries suggests that we require estimation techniques to recover incomplete measurements.

(**Challenge 3**) *Dynamicity, diversity, and ambiguity*

Every hostname used in DNS is assigned a time-to-live (TTL), which defines the lifetime of the hostname within a stub DNS resolver. If the hostname is not queried again before the TTL has expired, the DNS record of the hostname will be removed from a stub DNS resolver. In general, the hostnames in a CNAME chain have different TTL values. Figure 2 presents an example of cumulative distributive function (CDF) of TTL values for hostnames that are resolved to IP addresses (A record) and hostnames that are resolved to CNAMEs (CNAME record). Note that the data was taken from a mid-sized production network, and the characteristics of CDF were the same for other dataset. The graph clearly shows that A record hostnames have shorter TTLs than CNAME hostnames. For example, more than 50% of A record hostnames have TTL values that are less than 60 seconds. This indicates that the association between hostnames and IP addresses is highly dynamic. These hostnames have shorter TTLs because CDN providers tend to control traffic at a fine granularity [4].

The diversity of TTL values and DNS caching mechanisms leads to ambiguity of CNAME association behavior. We illustrate an actual sample in Fig. 3, which presents DNS resolutions for a client, $c1$. The first observation generates the relationship between $s1$ and $n1$ for client $c1$. The second observation generates the relationship between $s2$ and $n2$ for client $c1$. Now, assume an estimation problem. If we observe the pair $(c1, s1)$, which hostname should it be associated with? If we simply keep the relationships shown above, the answer is $n1$. However, due to the existence of intermediate CNAME node $m1$, the actual answer is $n2$ because $m1$ is now associated with $s2$ by a query of $n2$, and $n1$ is associated with $m1$ due to a caching mechanism. Note that this behavior depends on the implementation of the stub DNS resolver used by the client $c1$. If the implementation ignores intermediate CNAME nodes, the answer could be $n1$. Thus, there is an intrinsic ambiguity in CNAME associations.

**Contributions**:
In this work, we present a novel methodology that aims to infer the hostnames of HTTPS flows, given the three research challenges shown above. The key contributions of this work are summarized as follows.

- We present domain name graph (DNG), which is a formal expression that can keep track of CNAME chains (Challenge 1) and characterize the dynamic and diverse nature of DNS mechanisms and deployments (Challenge 3).
- We develop a framework called Service-Flow map (SFMap) that works on top of the DNG. SFMap estimates the hostname of an HTTPS server given a pair of client and server IP addresses. It can statistically estimate the hostname even when associating DNS queries are unobserved due to caching mechanisms, etc. (Challenge 2).
- Through extensive analysis using real packet traces, we validate the performance of SFMap in terms of accuracy and resource consumption.

The remainder of this paper is organized as follows. Section 2 summarizes the related work. Section 3 describes the proposed SFMap framework in detail. We evaluate the performance of SFMap in Section 4. Section 5 discusses the limitations of SFMap and future research directions. We conclude our work in Section 6.

## 2 Related Work

Many studies have examined the Internet traffic classification problem. Ref. [3] lists 68 studies on the topic. Here, we focus our attention on the studies that make use of DNS information to the traffic classification problem [2,6,8]. Mori et al. [6] proposed a method to identify traffic originating from large-scale video-sharing services such as YouTube. The key idea was to extract the rules of IP address numbering and naming conventions of fully qualified domain names (FQDNs) used for the services. Although their approach may work for a limited scope, it cannot be used to solve more generic web traffic classification problems. Plonka et al. [8] presented a traffic classification method that uses DNS traffic. They developed a method that stores per client DNS *rendezvous* state information in a tree-like data structure. Although their results demonstrated that the DNS rendezvous-based method performs well, even for encrypted traffic, their goal was different from ours because they assumed that DNS traffic implies the ground truth. In contrast, our goal is to estimate the hostnames of HTTPS traffic from the observations of DNS traffic. Bermudez et al.[2] developed a framework called DN-Hunter, which aims to classify traffic flows using DNS traffic. DN-Hunter uses a FIFO (first-in first-out) circular list to store the relationships among FQDN information and client-server pairs. Since the scope of DN-Hunter is mostly similar to ours, this work compares the performance of SFMap with DN-Hunter.

## 3 SFMap Framework

This section describes SFMap in detail. Section 3.1 presents the overview of the SFMap framework. Section 3.2 describes DNG, which is a key component of the SFMap framework. Section 3.3 details how SFMap estimates hostnames. Lastly, Section 3.4 explains how SFMap updates DNG and statistics that are used for the estimation.

### 3.1 Overview

The goal of SFMap is to infer a hostname $n$ of an HTTPS flow by associating preceding DNS responses with a flow key, which is defined with a pair of server IP address $s$ and client IP address $c$. To this end, SFMap needs to address the research challenges discussed in Section 1. To tackle the research challenges, the SFMap framework works on top of DNG, which will be detailed in the next subsection. A DNG keeps track of the structure of DNS records; thus, it can deal with CNAME chains (Challenge 1). Next, by relaxing the constraints of the DNG, the SFMap framework can handle cases wherein there are no preceding DNS responses that are associated with the client-server pair (challenge 2). The details of the hostname estimation will be described in Section 3.3. Finally, by adequately maintaining the DNG and using the observed TTL values, the SFMap framework can deal with the dynamic nature of DNS mechanisms (Challenge 3). The updating mechanism for the DNG will be discussed in Section 3.4.

Figure 4 summarizes the components of the SFMap framework. SFMap has three main functions, i.e., Learner, Estimator, and Updater. Learner consists of two components: the DNG and the Frequency counter. Learner component reads DNS queries/responses and builds and keeps the DNG and Frequency counter. Estimator performs host estimation; i.e., given a pair of client-server IP addresses $(c, s)$ for an HTTPS flow, estimator returns the most plausible hostname(s) using the information collected from DNG and Frequency counters. Updater reads DNS queries/responses and updates the status of the DNG and the Frequency counter.

Given these primitives, our problem can be formulated as maximum likelihood estimation (MLE) under the constraints of a DNG. Given $c$ and $s$ in an HTTPS flow,

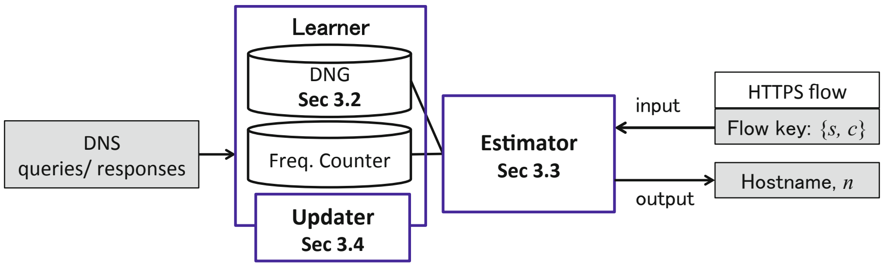

**Fig. 4.** Components of SFMap

the MLE is formulated as follows.

$$\hat{n}(c, s) = \operatorname*{argmax}_{n \in N} \Pr(n, c, s) \tag{1}$$

$$\text{s.t.} \quad N = \{n \in V_c : n \underset{G_c}{\rightarrow} s\}, \tag{2}$$

where $G_c = (V_c, E_c)$ denotes a DNG built for $c$, and binary operator $x \underset{G}{\rightarrow} y$ represents whether vertex $x$ can reach to vertex $y$ on graph $G$. In the following, we describe how we build and update $G_c$, how we extract $N$, how we compute the likelihood probability $\Pr(n, c, s)$, and how we get the final estimation $\hat{n}$.

## 3.2 DNG

A DNG, $G_c$, is a directed graph used to keep A and CNAME records observed in DNS responses queried by client $c$. DNGs can be built separately for each client $c$. A vertex, $v \in V_c$, is a server IP address or a hostname, while an edge, $e \in E_c$, represents an A or CNAME record that links a vertex to another vertex. Each edge is grafted by a corresponding A or CNAME record observed in a DNS response, and is associated with its expire time determined by observed TTL. If an edge, $e \in E_c$, is expired, it will be removed from $G_c$.

Here, we examine how the DNG expression naturally represents the behavior of DNS resolution. Assume that clients obtain a server address via DNS responses only and that we have never missed any DNS response for the clients; i.e., DNG $G_c$ represents all name resolutions requested by a client $c$. When a client $c$ sends an HTTP request to a server $n$, the server $n$'s IP address $s$ should have been resolved by DNS. This association of $n$ and $s$ obtained through the DNS mechanism can be expressed as a path from n to $s$ on the DNG $G_c$. Note that there are cases where we cannot find such a path due to the caching mechanisms. In such cases, we need to employ several techniques that will be described soon.

## 3.3 Estimator

In the estimation phase, we must first select candidate hostnames that are likely the original hostname for a given client-server pair $(c, s)$. We extract a set of candidate hostnames $N$ from DNG $G_c$, using Eq. 2. If $|N| \geq 1$, we estimate the hostname with the MLE shown in Eq. 1. A method to calculate the likelihood probability $\Pr(n, c, s)$ will be shown later.

As we mentioned in Section 1, $N$ can be an empty set due to the standard and illicit DNS caching mechanisms. In such cases, we cannot directly associate an HTTPS flow with preceding DNS responses. To deal with these cases, SFMap extends the candidate hostnames by relaxing the constraint of edge expiration. This relaxation enables us to select hostnames that are missed due to the existence of DNS clients that ignore DNS TTL for improving the user experience. Now, $N$ is obtained as

$$N = \{n \in V_c : n \underset{\tilde{G}_c}{\rightarrow} s\}, \tag{3}$$

where $\tilde{G}_c = (V_c, \tilde{E}_c)$ and $\tilde{E}_c$ include both valid and expired edges.

Finally, if we do not have any candidate hostnames at this stage, we use the union of all clients' DNGs (union DNG). In other words, we use the observations of other clients as a hint to estimate the most plausible hostname. Let $C$ denote a set of all clients. The union DNG is defined as $G = (V = \bigcup_{c \in C} V_c, E = \bigcup_{c \in C} E_c)$. Using the union DNG $G$, the candidate hostnames can be selected as

$$N = \{n \in V : n \underset{G}{\rightarrow} s\}. \tag{4}$$

It then estimates the hostname with the following MLE formulation:

$$\hat{n} = \underset{n \in N}{\operatorname{argmax}} \Pr(n, s). \tag{5}$$

Like Eq. 3, we can further relax the constraint of expiration for the union DNG $G$; i.e.,

$$N = \{n \in V : n \underset{\tilde{G}}{\rightarrow} s\}, \tag{6}$$

where $\tilde{G} = (V, \tilde{E})$ and $\tilde{E}$ include both valid and expired edges.

To recap, the Estimator runs the combinations below from top to bottom in a step-by-step manner until a plausible hostname is found. For future reference, we give names to these steps, where LE and UE refer to Local and Union Estimators, and NTE refers to "No TTL Expiration". For instance, the estimator LE-NTE (Local Estimator with No TTL Expiration) starts with the first step and continues to the second step until at least one candidate hostname is found, but will not proceed to the third and fourth steps. We will examine the accuracies of these estimators to study the factors that contribute to improve the estimation accuracies.

| Step | MLE | constraint | Name |
|---|---|---|---|
| 1st | Eq. (1) | Eq. (2) | LE |
| 2nd | Eq. (1) | Eq. (3) | LE-NTE |
| 3rd | Eq. (5) | Eq. (4) | UE |
| 4th | Eq. (5) | Eq. (6) | UE-NTE |

Finally, we note the time complexity of the Union Estimators. In the Union DNG, a single-source path search from $s$ with reverse edges requires $O(|E|)$ on a directed acyclic graph with topological sort, and frequency lookups are executed for $n \in N \subseteq V$. Therefore, the time complexity of Union Estimators is $O(|V| + |E|)$. However, we empirically revealed that the actual mean time complexity is much smaller than this worst-case upper bound, and is close to $O(|V_c| + |E_c|)$ because majority of hostnames can be estimated with LE and LE-NTE as we shall show in Section 4. The details are omitted due to the space limitation.

**Calculation of the Likelihood Probabilities.** To calculate the likelihood probabilities, we make use of empirical data. Let $F_c(n, s)$ denote the frequency of DNS messages queried by client $c$ for hostname $n$ with resolved address $s$. Using $F_c(n, s)$, Eq. 1 can be calculated as

$$\operatorname*{argmax}_{n \in N} \Pr(n, c, s) = \operatorname*{argmax}_{n \in N} F_c(n, s).$$

Similarly, Eq. 5 can be calculated as

$$\operatorname*{argmax}_{n \in N} \Pr(n, s) = \operatorname*{argmax}_{n \in N} F(n, s),$$

where $F(n, s) = \sum_{c \in C} F_c(n, s)$. The method to update the frequency will be shown in the next subsection.

### 3.4 Updater

The Updater updates DNG $G_c$ and frequency $F_c$ when it receives a DNS response. A DNS response is associated with client $c$ and queried hostname $n^\star$. The response also includes a set of A records and another set of CNAME records. Let these sets be $A$ and $M$, respectively. An A record associates hostname $n$ and server address $s$, while a CNAME record associates two hostnames $n'$ and $n$. Let these records be $(n, s) \in A$ and $(n', n) \in M$, respectively.

Due to the existence of short TTL value set for an A record, a client often resolves an intermediate hostname (i.e., CNAME) instead of the original one. In such a case, the frequency of an original hostname is undervalued. To cope with such a case, SFMap increments the frequencies of all original hostnames that can reach to the queried hostname. Let a set of edges be $E_c = \{(n'', n), (n', n), (n, s)\}$, where $n$ is a CNAME of $n''$ or $n'$. If $n^\star = n$ is queried, the Updater increments $F_c(n'', s)$ and $F_c(n', s)$, instead of $F_c(n, s)$. Note that we assume that original hostnames should be leaf vertices on a DNG (a leaf is a vertex without incoming edge). In fact, more than 99.7% of requested hostnames are leaf vertices in our observations.

Algorithm 1 presents an algorithm that updates $G_c$ and $F_c$ upon receiving a DNS response, $(c, n^\star, A, M)$. We discount the incremental value by the number of $(n', s)$ pairs at Line 7, because the algorithm increments $F_c$ for all $n' \in V$ reachable to $n^\star$ and for all $s$ in $A$. At Line 3, we update the expiration time of edge $(u, v)$. In addition to Algorithm 1, the Updater periodically checks the TTL expiration for

**Algorithm 1.** Updater

```
  Input: $c$, $n^\star$, $A$, $M$                                    // DNS response
1 for $(u, v) \in A \cup M$ do
2     $E_c = E_c \cup \{(u, v)\}$                                     // to add edge
3     update expire time of edge $(u, v)$
4 $N' = \{n' \in V_c : (*, n') \notin E_c,\ n' \underset{G_c}{\rightarrow} n^\star\}$   // leaf vertices reachable to $n^\star$
5 for $n' \in N'$ do
6     for $(*, s) \in A$ do
7         $F_c(n', s) = F_c(n', s) + \frac{1}{|N'| \cdot |A|}$          // to increment frequency
8 return $G_c, F_c$
```

all edges. If the DNS TTL expires for an edge $(u, v)$, the edge will be removed. The time complexity of maintenance is $O(|V_c|)$ for the loop at Line 5, assuming $O(|A|) = O(|M|) = O(1)$.

## 4 Evaluation

Here, we first describe the datasets used and present some basic statistics derived from the data. We then evaluate the estimation accuracy of SFMap. For reference, we compare the performance of SFMap with DN-Hunter [2]. Finally, we examine the resource consumption of SFMap, which was implemented with Python.

### 4.1 Datasets and Statistics

To investigate the effectiveness of SFMap, we used the two datasets, LAB and PROD, which are the packet traces collected from a gateway router of local area network used by a research group and a gateway router of middle-scale production network, respectively. The basic statistics of the datasets are summarized in Table 1. As is shown in Table 1, the datasets cover two different scales, small and middle. Both datasets have same time length, twelve hours. Of the twelve hours, the last two hours are used to examine the accuracy; i.e., the first 10 hours are used for *warm-up* phase. We adopted the length of warm-up from the observation of TTL distribution shown in Fig. 2; i.e., majority of the DNS resource records had TTL values less than 10 hours.

Here, we present the characteristics of DNGs derived from our datasets. Table 2 presents the statistics of the DNGs. For brevity, we omit DNGs with TTL expiration because these DNGs should be smaller than those without TTL expiration. As is shown in the table, Union DNGs have fewer nodes and edges. For instance, since the number of clients for the LAB dataset is 10 (see Table 1), the total number of nodes in the Local DNGs should be $10 \times 460 = 4600$. Thus, the number of total nodes in the Union DNG (=2849) is less than the number of total nodes in the Local DNGs. This observation implies that (1) each client-server pair in the Local DNGs

**Table 1.** Basic statistics of the datasets

| | learning time | # of clients | # of DNS responses | estimating time | # of servers | # of HTTP requests | # of hostnames |
|---|---|---|---|---|---|---|---|
| LAB | 0 ∼ 12 h | 10 | 5,226 | 10 ∼ 12 h | 1,705 | 542 | 1,135 |
| PROD | 0 ∼ 12 h | 4,250 | 86,854 | 10 ∼ 12 h | 10,785 | 55,091 | 10,534 |

**Table 2.** Statistics of the DNGs at the end of measurement period

| | Local DNG w/o TTL expiration | | Union DNG w/o TTL expiration | |
|---|---|---|---|---|
| | mean # of nodes | mean # of edges | total # of nodes | total # of edges |
| LAB | 460 | 755 | 2,849 | 5,979 |
| PROD | 56 | 80 | 25,403 | 172,974 |

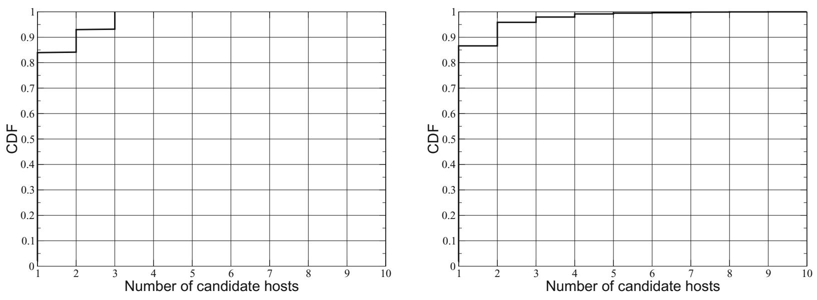

**Fig. 5.** CDFs of the number of candidate hostnames for each HTTP request: LAB (left) and PROD (right)

has duplicate nodes and edges, and (2) the Union DNGs can be maintained with less memory. Figure 5 shows the CDF of the number of candidate hostnames for each HTTP request. The results suggest that roughly 15% of the HTTP requests have multiple candidates; i.e., we must statistically estimate the original hostname from these candidates.

## 4.2 Estimation Accuracy

Our methodology was evaluated using the two datasets. We make use of HTTP as a means to evaluate the accuracy of our methodologies. The ground truth was obtained from HTTP request headers, which contain hostname information. We note that although the distributions of hostnames could be different between HTTP and HTTPS, the fundamental mechanism of resolving hostname before starting HTTP/HTTPS communication should be identical. From the packet traces, we

**Table 3.** Accuracies of the estimators (exact matching)

| | LE | LE-NTE | UE | UE-NTE | DN-Hunter |
|---|---|---|---|---|---|
| LAB | 54.98% | 68.08% | 71.59% | 92.25% | 67.90% |
| PROD | 79.90% | 88.29% | 90.88% | 90.88% | 85.40% |

**Table 4.** Accuracies of the estimators (public suffix matching)

| | LE | LE-NTE | UE | UE-NTE | DN-Hunter |
|---|---|---|---|---|---|
| LAB | 57.20% | 70.30% | 73.80% | 94.46% | 73.43% |
| PROD | 83.20% | 92.12% | 94.52% | 94.98% | 89.98% |

**Table 5.** Accuracies of the Top-3 estimations (UE-NTE)

| | Exact matching | | | Public suffix | | |
|---|---|---|---|---|---|---|
| | Hit in 1 | Hit in 2 | Hit in 3 | Hit in 1 | Hit in 2 | Hit in 3 |
| LAB | 92.25 | 97.23 | 98.16 | 94.46 | 98.16 | 98.16 |
| PROD | 90.88 | 95.77 | 96.71 | 94.98 | 97.01 | 97.43 |

read DNS packets to build and update the DNGs. For each HTTP request pair $(c, s)$, we estimate the hostname and compare it against the ground truth. For comparison purposes, we implemented DN-Hunter [2]. DN-Hunter has a single parameter that determines the size of memory, which keeps track of tuples of $(c, s, N)$, where $N$ is a hostname. To obtain the highest performance of DN-Hunter, we set infinite amount of memory size. We note that this configuration did not overflow physical memory we used in our experiments.

Table 3 and Table 4 summarize the results, where we use the notations introduced in Section 3.3. Table 3 shows the estimation accuracies in the context of exact matching, and Table 4 relaxes matching using a public suffix [1]; i.e., we can see that aaa.example.com and bbb.example.com are matched in the context of the public suffix. Using the public suffix matching allows us to distinguish hostnames with different domains, e.g., youtube.com and google.com.

First, the accuracies were improved for estimators with no TTL expiration (NTE). This observation suggests that there are a non-negligible number of DNS implementations that ignore TTL settings, which agrees with a previous report [4]. Second, the Union DNG also contributed to improve the accuracy. This observation suggests that using other clients' information is useful in improving the accuracy when no other hint is available. Third, if we allow public suffix matching, accuracies are further improved for all the estimators. The UE-NTE achieved roughly 95% of accuracy for both datasets. Finally, the UE-NTE outperformed DN-Hunter. For the exact matching experiments, while the estimation error rates of DN-Hunter were 15-32%, the estimation error rates of UE-NTE were 8–9%. Thus, UE-NTE successfully reduced the error rates by 50-70%.

DN-hunter returns a single hostname given a client-server pair; however, if there are multiple candidate hostnames, SFMap can return several hostnames with the highest likelihood probabilities. Table 5 shows the results where we accept the top

**Table 6.** Memory usage of RAM and processing time for UE-NTE

| | memory (MB) | time (s) |
|---|---|---|
| LAB | 35.1 | 0.8 |
| PROD | 686.2 | 20.6 |

three hostnames as estimation. Notably, accuracies exceed 96-98% for exact matching if we pick up the top three hostnames. We note that in most cases, the hostnames ranked in the top three look similar. For instance, the top three hostnames are: pagead2.googlesyndication.com, pubads.g.doubleclick.net, and googleads.g.doubleclick.net, which are all attributed to Ad Network services. Thus, by extending the candidate hostnames, we can establish better estimations that work in practice. This extension is acceptable for our original motivation; i.e., understanding the mix of HTTPS traffic.

### 4.3 Resource Consumption

We study the resource consumption of SFMap, using its implementation with Python. We note that the implementation has a much room for improvement in terms of optimizing resource management. Table 6 shows the amount of memory consumed and the amount of time to process the entire data, including data for warm-up. The results demonstrate that our implementation of SFMap works within a reasonable amount of memory, i.e., less than 40 MB for LAB and less than 700 MB for PROD. Also, processing time is much shorter than the actual measurement length, 12 hours. Thus, SFMap should work in a real-time fashion. We will further discuss the scalability of SFMap in the next section.

## 5 Discussion

Here, we discuss the limitations of the proposed SFMap framework. We also outline several future research directions that can help extend our framework.

### 5.1 Sources of Misclassification

By carefully examining the estimation results, we found several intrinsic sources of misclassification. There are several factors that are associated with the incomplete measurements. As we mentioned before, the first factor is the existence of aggressive DNS caching mechanisms that ignore DNS TTL setting. The second factor we found through this study was mobility of terminals; i.e., an IP address had already been resolved in other network before the terminal arrived to the vantage point. The third factor we found was the use of an IP address in the URI. We found a non-negligible number of HTTP requests had such URIs. We manually inspected the cases and found that there are several applications that likely hard-coded an IP address; thus, they never send DNS queries. Although these are not the controlling factors today, we may need to address them if such deployments become popular in future.

### 5.2 Scalability

As shown in Section 4.3, our SFMap implementation processed traffic collected at middle-scale production network within a reasonable amount of memory; i.e., less than 700 MB. Then, we may want to ask whether SFMap works for large-scale networks. First, because SFMap does not require per-packet processing, we believe that the processing time does not matter in practice. It just processes DNS response packets and the first packets of HTTPS flows, ignoring remaining packets. Furthermore, as we discussed in Section 3.3, empirical studies revealed that time complexity of estimation is close to $O(|V_c| + |E_c|)$, which is fairly small as shown in Table 2. We also note that estimation processes can be parallelized if we need it. Second, it is clear that the size of DNGs increases as the number of observed client increases. If the size of DNG becomes large enough to press the capacity of memory, we need to eliminate old records. Instead of keeping all the records for a certain amount of time, e.g., 12 hours, we may want to quickly delete old records that are less-likely to be reused in future. More sophisticated way to manage the elements in DNGs is left for the future study. Another possible solution would be to build a new algorithm that can maintain and update DNGs in a more compact data structure. The topic is also left for the future study.

## 6 Summary

The SFMap hostname estimation framework was presented. SFMap enables network operators to estimate the hostnames of HTTPS traffic by observing DNS queries/responses. To tackle the challenges that arise from the recent dynamic deployment and diverse implementations of DNS ecosystems, the proposed SFMap framework runs on top of a single key component; i.e., a DNG, which is a formal expression that characterizes the highly dynamic and diverse nature of DNS mechanisms. From extensive analyses using real packet traces collected from two distinct locations with different network scales, we have demonstrated that SFMap has good estimation accuracy and can outperform DN-Hunter, which is a state-of-the-art estimation technique. Our experiments using middle-scale network traffic with thousands of clients demonstrated that SFMap can be run on a standard commodity PC, using less than 700 MB of memory space. In future, we plan to enhance the scalability of SFMap.

**Acknowledgments.** This work was supported by JSPS KAKENHI Grant Number 25880020.

## References

1. Public suffix list. https://publicsuffix.org/
2. Bermudez, I.N., Mellia, M., Munafo, M.M., Keralapura, R., Nucci, A.: DNS to the rescue: discerning content and services in a tangled web. In: Proc. of IMC, pp. 413–426 (2012)

3. CAIDA. Internet traffic classification. http://www.caida.org/research/traffic-analysis/classification-overview/
4. Callahan, T., Allman, M., Rabinovich, M.: On Modern DNS Behavior and Properties. SIGCOMM Comput. Commun. Rev. **43**(3), 7–15 (2013)
5. Korczynski, M., Duda, A.: Markov chain fingerprinting to classify encrypted traffic. In: Proc. of INFOCOM, pp. 781–789 (2014)
6. Mori, T., Kawahara, R., Hasegawa, H., Shimogawa, S.: Characterizing traffic flows originating from large-scale video sharing services. In: Ricciato, F., Mellia, M., Biersack, E. (eds.) TMA 2010. LNCS, vol. 6003, pp. 17–31. Springer, Heidelberg (2010)
7. Naylor, D., Finamore, A., Leontiadis, I., Grunenberger, Y., Mellia, M., Munafo, M., Papagiannaki, K., Steenkiste, P.: The cost of the "S" in HTTP. In: Proc. of CoNext (2014)
8. Plonka, D., Barford, P.: Flexible traffic and host profiling via DNS rendezvous. In: Proc. of SATIN (2011)
9. Sandvine. Global internet phenomena report: 1H 2014. http://bit.ly/1jHpsW5
10. Su, A.-J., Choffnes, D.R., Kuzmanovic, A., Bustamante, F.E.: Drafting behind akamai (travelocity-based detouring). In: Proc. of SIGCOMM, pp. 435–446 (2006)