Investigating the Agility Bias in DNS Graph Mining

Jukka Ruohonen
Department of Future Technologies
University of Turku, Finland
juanruo@utu.fi

Ville Leppänen
Department of Future Technologies
University of Turku, Finland
ville.leppanen@utu.fi

***Abstract*—The concept of agile domain name system (DNS) refers to dynamic and rapidly changing mappings between domain names and their Internet protocol (IP) addresses. This empirical paper evaluates the bias from this kind of agility for DNS-based graph theoretical data mining applications. By building on two conventional metrics for observing malicious DNS agility, the agility bias is observed by comparing bipartite DNS graphs to different subgraphs from which vertices and edges are removed according to two criteria. According to an empirical experiment with two longitudinal DNS datasets, irrespective of the criterion, the agility bias is observed to be severe particularly regarding the effect of outlying domains hosted and delivered via content delivery networks and cloud computing services. With these observations, the paper contributes to the research domains of cyber security and DNS mining. In a larger context of applied graph mining, the paper further elaborates the practical concerns related to the learning of large and dynamic bipartite graphs.**

***Index Terms*—content delivery network, fast flux, fluxiness, bipartite graph, dynamic network, botnet, DNS, CDN, IPv6**

I. Introduction

The terms DNS agility and agile DNS refer to highly dynamic mappings between fully qualified domain names (FQDNs) and Internet protocol (IP) addresses [1]. This empirical paper examines the biases from this kind of agility for data mining that operates with DNS graphs comprised of addresses, domains, and links (edges) between addresses and domains.

Content delivery networks (CDNs), as pioneered particularly by Akamai Technologies, Inc., have had a significant impact upon this type of DNS agility. In order to improve reliability and efficiency, these networks essentially distribute the payload both across multiple servers and geographically across multiple locations. In a sense, therefore, CDNs can be viewed as a visible historical milestone in the emergence of cloud computing, the critical juncture having been located in the late 1990s and early 2000s during which Akamai introduced its core commercial technologies. Although the company's infrastructural empire has grown ever since—currently covering over two hundred thousand servers around the globe [2], the core DNS solution is still based on the same underlying rationale. In a nutshell: a client's DNS query is used to route the client to a topologically nearest and generally optimal caching edge server, provided that the queried domain has joined to a CDN by pointing its name servers to those provided by the CDN orchestrator [3], [4], [5]. While CDNs have been extensively studied in the fields of communications and computer networks, a different kind but still empirically highly similar DNS agility has been at the center of empirical DNS mining applications motivated by cyber security questions.

The malicious use of DNS became widely known by the late 2000s detection of a botnet that generated domain names dynamically. While the botnet used a traditional worm-like propagation to spread, it had a centralized command and control unit to which the bots connected with their daily routines for seeking out the pseudo-random domain names [6]. Around the same time, also the so-called fast flux service networks became widely known. While botnets have adopted fast flux algorithms, the phenomenon itself generalizes to CDN-like distributed proxy networks built on top of compromised hosts, which via DNS redirect the traffic to central hosts that serve the actual malicious web or other content [4]. In this paper, the study of agile DNS is motivated by revisiting two well-known [4], [7] metrics for observing the behavior of fast flux networks and their hypothesized empirical differences to CDNs and other legitimate DNS-based networking solutions.

The agility bias is operationalized with two criteria that emphasize the statistical effects upon the learning and underlying structure of DNS graphs. In essence, the first question examined in the empirical experiment is simple: (a) *how many DNS resolving rounds are roughly required to attain a representative DNS graph?* If many rounds are required, it follows that there is a "learning bias" for graphs learned with only one or few rounds. In other words, there would be many false negative vertices (absence of vertices that should be present) and false negative edges (missing of relations that should be present) [8]. The second question examines a different kind of a "sampling bias": (b) *what is the effect of extremely agile domains upon the statistical characteristics of already learned DNS graphs*? While the first question relates to obtaining a statistically representative sample, the sampling bias is best understood as a question about the impact of outliers for the structure of a DNS graph. Here, an outlier is understood broadly as a graph object that is rare and differs considerably from the majority of graph objects sampled [9]. Fast flux networks are a good example of such outliers seen in empirical DNS graph mining applications.

To answer to the two questions, the remainder of this paper first briefly revisits the scholarly background in Section II, proceeding to discuss operationalization in Section III as a preparation for the empirical experiment in Section IV. Conclusions and discussion follow in Section V.

II. Background and Related Work

The scholarly background behind the concept of DNS agility has been strongly motivated by network security [1]. By and large, the same applies to empirical DNS graph mining approaches. Therefore, the following discussion incrementally proceeds from the fast flux context to the measurement aspects.

A. Flux Types

The so-called fast flux networks refer to a bundle of largely unknown networking algorithms used to rapidly shuffle domain-address mappings in order to evade detection, improve reliability, and to control a botnet or other malicious network.

1) Domain and Address Fluxing: It is possible to separate domain name and IP address agility from each other [10]. The former (a.k.a. domain fluxing) refers to rapid generation of domain names algorithmically. The notorious Conficker worm is likely the earliest known historical example of a network using this technique—the underlying botnet that was constructed via worm-like infections eventually generated even up to fifty thousand domain names each day [6], [10]. Thereafter, these domain generation algorithms (DGAs) have been successfully used in a number of high-profile botnets, including such technically sophisticated but morbid cases as Rustock [11] and Gameover ZeuS [12], as well as in related malware implementations such as the CryptoLocker ransomware [13]. Although no DGAs are supposedly present in the empirical experiment, it goes without saying that the potential presence of DGAs may induce a severe bias for DNS graph mining due to the dynamic addition of vertices, including those vertices that are labeled to represent domain names.

Clearly, DGAs are generally only applicable for characterizing malicious agility; no legitimate DNS solution should generate thousands of domain names, and then pick one of these for a brief communication session with a command-and-control unit by activating and subsequently deactivating the associated DNS records [14]. In this paper, however, the interest is to observe the latter type of DNS agility (a.k.a. IP address fluxing); that is, the dynamic mapping of multiple IP addresses to a single fully qualified domain name. This type of fluxing is common for both legitimate and malicious agility.

2) Fluxing by Record Types: The fast flux phenomenon was historically related to (address) fluxing of IPv4 addresses (a.k.a. A records). Soon after the discovery of this type of fluxing, malicious networks were observed to also shuffle their name server (NS) records, adding the concept of "double fluxing" to the scholarly literature [7], [15], [16]. A network architecture for double fluxing can be complex. For instance, sometimes the compromised hosts may carry their own malicious DNS servers, which enables a highly customizable network for communication, control, and reliability.

Although single and double fluxing with A and NS records have supposedly remained the mainstream of malicious DNS agility, the versatility of DNS has continued to offer many further possibilities for protocol misuse. To provide further obfuscation guards against detection, DNS tunneling is possible via the text (TXT) records [17], [18]. These tunneling techniques provide further covert channels particularly for the command and control aspects. Also AAAA records may be used for address fluxing [19], which also motivates the inclusion of IPv6 in the empirical experiment.

B. Quantities

The seminal fast flux work of Holz and associates [4] considered two "unaggregated" DNS quantities alongside one "aggregated" one. The unaggregated and aggregated quantities are, respectively: the number of unique A records (IPv4 addresses), the number of unique NS records, and the number of unique autonomous system numbers for all of the A records. To elaborate the metrics derived from the unaggregated quantities, a couple of clarifications should be made beforehand.

1) Aggregation: In this paper the focus is restricted to the "unaggregated" level, but only in the sense that no attempts are made to analyze DNS agility at the level of larger aggregates, including subnets, IP address blocks [1], and even Internet registries. The same applies to FQDNs. While "aggregation" to the second highest level (cf. `xboxlive.com`) is used for one sample in the experiment, which is a common practice in DNS mining [20], [21], the analysis uses also a sample containing "unaggregated" domains. For instance, one observed FQDN is `download.gfwl.xboxlive.com`, which, through numerous canonical name records (CNAMEs), is aliased to a content delivery network.

2) Time: It is also important to understand that the study time refers to $i = 1, \ldots, q$ iteratively made DNS queries from a local resolver, using live DNS for the resolving. In other words, the study time operates with a client-side perspective to DNS, using simple polling to obtain the record sets [22]. Unlike with comparable query engines [15], the polling procedure ignores caching and bypasses questions related to time-to-live (TTL) values for the record sets obtained. This said, with small alterations, an equivalent analysis could be carried out also with flow data from DNS servers.

Depending on a bookkeeping solution, each resolving round can be assumed to contain also one or more timestamps in the calendar time [22]. If t_i is fixed to the integer-valued timestamp at which the i:th query completed, the magnitudes in this strictly monotonic time sequence,

$$(t_1, \ldots, t_i, \ldots, t_q),\ t_i \in \mathbb{N}, t_i < t_{i+1} \text{ for all } 1 \leq i < q, \quad (1)$$

depend on local infrastructural factors, including bandwidth and the software solution used for resolving. Consequently, the calendar time is generally irregular. Akin to the initial fast flux work [4], this paper does not explicitly observe (1), however, and also the t-indices are therefore omitted for convenience.

C. Fluxiness and Cumulative Counts

To measure the dynamics, the *fluxiness* of a domain has been operationalized as the number of all unique A records in a fixed learning period divided by the A records returned by a single DNS lookup of a given domain name [4]. In theory the same operationalization applies to all conventional DNS records returned for a given resource record type, but in

this paper the focus is on the conventional A records (IPv4 addresses) and AAAA records (IPv6 addresses). To fix the basic idea as well as the notation, let R_i denote a set of unique DNS records of specific type returned for the i:th query:

$$R_i \in \{A_i, AAAA_i\}. \tag{2}$$

To use one of the private address space networks from RFC 1918 as an example, a couple of queries could return

$$\begin{aligned} A_i &= \{\texttt{10.0.0.1}, \texttt{10.0.0.254}\} \quad \text{and} \\ A_{i+1} &= \{\texttt{10.0.0.1}, \texttt{10.0.0.2}, \texttt{10.0.0.254}\}, \end{aligned} \tag{3}$$

such that $|A_i| = 2$ and $|A_{i+1}| = 3$. This small private example network would be agile in the sense that the fluxiness score $3/2 = 1.5$ for the i:th query did not equal unity. If A_{i+2} would subsequently equal $\emptyset$, such that $|A_{i+2}| = 0$, the $(i+2)$:th query could have resulted a so-called NXDOMAIN case, that is, the domain would not have resolved to any IPv4 address. As said, timeouts and other errors may be equally likely in practice.

With this notation, the fluxiness of a domain at the i:th query round can be defined with a function

$$\varphi_{R_i} = f(R_i, R) = \begin{cases} 0 & \text{if } R_i = \emptyset, \\ |R| \ / \ |R_i| & \text{otherwise}, \end{cases} \tag{4}$$

where

$$R = R_1 \cup R_2 \cup \cdots \cup R_q \tag{5}$$

refers to all records obtained through q queries. For legitimate agile DNS, the later values in a vector

$$\boldsymbol{\varphi}_R = (\varphi_{R_1}, \ldots, \varphi_{R_i}, \varphi_{R_{i+1}}, \ldots, \varphi_{R_q}) \tag{6}$$

can be hypothesized to converge relatively fast to a fixed scalar, while slower convergence should be expected for malicious fast flux networks [4]. In other words, it should be generally easier to learn the record sets for legitimate agile DNS compared to malicious agility. Alternatively, the record cumulation

$$\boldsymbol{\phi}_R = (|R_1|, |R_1 \cup R_2|, \ldots, |R_1 \cup \cdots \cup R_q|) \tag{7}$$

should display a slowly decelerating but diverging growth curves for legitimate and malicious fluxing [4]. Both $\boldsymbol{\varphi}_R$ and $\boldsymbol{\phi}_R$ can be also used for probing agile domains in general.

III. Measurement Framework

The paper investigates the effect of particularly (7) upon the learning and sampling of DNS graphs. Before proceeding to elaborate the agility bias, a brief discussion is required for demonstrating the use of the two fast flux metrics in practice, and for introducing the construction of bipartite DNS graphs.

A. Degrees of Agility

There exists no single criterion according to which domains could be labeled as agile and non-agile or static and dynamic. When $q \to \infty$, some changes are bound to happen due to infrastructural changes—if only when t_q in (1) denotes a sufficiently large integer with respect to t_1, such that $t_q - t_1$ amounts to a few years, say. Nevertheless, at minimum, an agile domain should satisfy the following exclusion condition:

$$\mathrm{c}(\tau) : \# \text{ of unique values in } \boldsymbol{\phi}_R \geq \tau, \ 0 < \tau \in \mathbb{N}, \tag{8}$$

for a threshold value $\tau = 2$. In other words, at minimum the size of R must be larger than one for a change to be possible to begin with. This same $\mathrm{c}(2)$ condition is also equivalent to saying that the standard deviation of $\boldsymbol{\phi}_R$ should be non-zero. To demonstrate the criterion and the unaggregated fast flux metrics in practice, consider the four plots in Fig. 1, which are based on a dataset described later in Section IV-A. The solid and dashed lines in the figure refer to (7) and (6) for IPv4 addresses, or $\boldsymbol{\phi}_A$ and $\boldsymbol{\varphi}_A$, respectively.

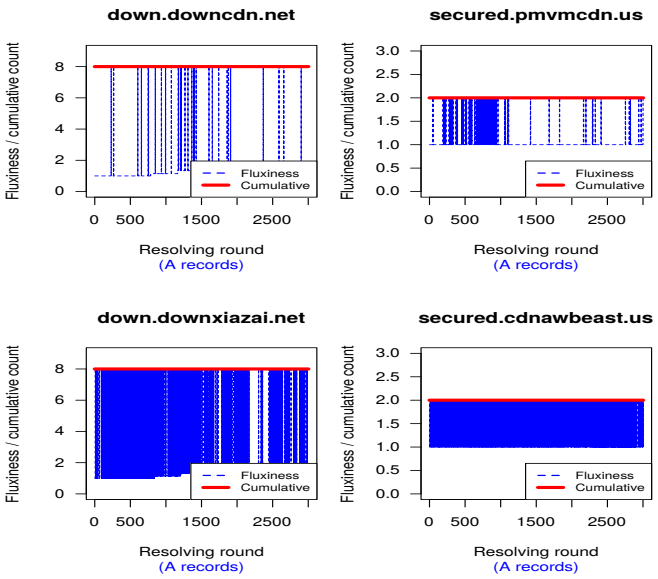

Fig. 1. A 4-Tuple of Agile Domains (sampled from [23])

The four visualized domains are clearly agile; there were changes throughout the $q = 3000$ resolving rounds represented on the x-axes. But while all four domains are agile in one sense or another, the degree of agility varies wildly between the domains. The two plots in the top row exhibit rather irregular, stochastic-looking fluxing. The domain `secured.cdnawbeast.us`, as visualized in the bottom-right plot, exhibits rather extreme agility of different kind by constantly, almost round by round, resolving to either one or two addresses. Despite of these rapid dynamics, however, the cumulative vector $\boldsymbol{\phi}_A$ is easy to learn for these domains. In fact, the three thousand resolving rounds were clearly wasteful for learning the domain-address mappings: for all four domains in Fig. 1, the cumulative IPv4 address counts remained constant throughout the resolving rounds.

Both $\boldsymbol{\varphi}_R$ and $\boldsymbol{\phi}_R$ may remain constant around a single unique value. If all records were learned already during the first round, such that $|R|$ in (5) equals $|R_1|$, the cumulative counts in the vector $\boldsymbol{\phi}_R$ in (7) would all equal the single unique integer $|R_1|$. Likewise, $\boldsymbol{\varphi}_R$ may remain constant. For

instance, consider a domain that resolves to two new IPv4s in each $i = 1, 2, 3$ resolving rounds. For such a domain, the function (4) results $6/2 = 3$ for each round yet still $\phi_A = (2, 4, 6)$. Constant fluxiness scores are also easily demonstrated empirically. An analogous empirical case is thus illustrated in the top-left plot of Fig. 2 by using the same dataset. This CDN domain `cdn3.opencandy.com` resolved very slowly to six IPv4 addresses—yet the fluxiness scores remained constant throughout the resolving rounds.[1]

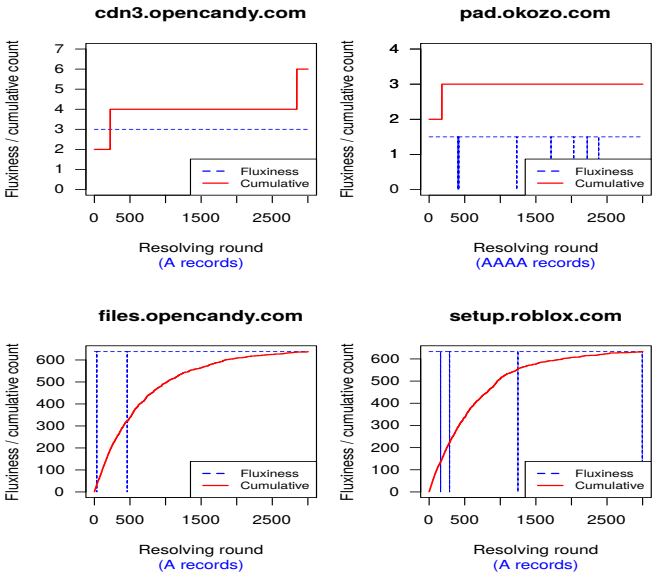

Fig. 2. Further Agile Domains (sampled from [23])

When moving to the top-right plot, it becomes evident that also IPv6 addresses may be agile. Moreover, the two bottom row plots both satisfy $c(630)$. In other words, more than 21 percent (out of $q = 3000$) of the resolving rounds added a new unique IPv4 address to the cumulative A record sets of `files.opencandy.com` and `setup.roblox.com`. Although not that visible from the plots, ϕ_A still follows a step function also for these two domains. The lengths between the steps—the number of resolving rounds that resulted the same unique cumulative count—only widen toward the end of the series, however. It is also worth noting that `cdn3.opencandy.com` is an alias to `cdn.opencandy.edgesuite.net`, which is an alias to a domain underneath Akamai's `akamai.net`. Furthermore, both `files.opencandy.com` and `setup.roblox.com` are aliased to Amazon Web Services. Particularly these two cloud-powered domains spotlight important implications for DNS graph mining—clearly, already the shape of the growth curves indicate the presence of a serious bias for "static" graphs learned with $q = 1$. In fact, even the $q = 3000$ choice may not have been enough for learning the addresses dynamically mapped to the bottom row domains in Fig. 2.

[1] According to Wikipedia, "the OpenCandy ecosystem" related to adware.

B. DNS Graphs in Brief

A typical—but by no means unique [24]—DNS graph is undirected and bipartite, connecting fully qualified domain names to their IP addresses [1], [16], [25]. The basic idea remains similar for server-side applications, although these allow further separating the "inside" (origins of queries) and "outside" (targets of queries) traffic passing through DNS servers, routers, or related machinery [26], [27], [28], [29]. Given the client-side perspective adopted for this paper, the underlying graph structure can be denoted with

$$G_R = (V_D \cup V_R, E), \tag{9}$$

where G_R denotes an R-type DNS graph given (2), V_D and V_R are ordered vertex sets of the observed FQDNs and R-type DNS records, respectively, and E is a set of edges that connect elements in V_D to elements in V_R. That is, the edge set E means that there is a relation $E \subseteq \{(v, u) \mid v \in V_D, u \in V_R\}$ between all R-type DNS records $u \in V_R$ to which a domain $v \in V_D$ resolved at i. In addition, a bookkeeping set, say B_R, is kept for recording the resolving round at which an R-type DNS record was added to the vertex set V_R. It follows that $|V_R| = |B_R|$ and $b \leq q$ for any positive integer $b \in B_R$. This bookkeeping conveys the dynamic construction of a G_R.

Vertices and edges are added dynamically to G_R according to the resolving rounds. That is, for each i, new vertices and edges are added to the graph in case these were not already previously added to the graph. To illustrate: for the small IPv4 example (i.e., $R = A$) in (3), at the i:th resolving round there would be a single domain name vertex $v \in V_D$, which would be connected to two IPv4 address vertices, such that $V_A = (\texttt{10.0.0.1}, \texttt{10.0.0.254})$, $(v, \texttt{10.0.0.1}) \in E$, and $(v, \texttt{10.0.0.254}) \in E$. The subsequent round with A_{i+1} would result a slightly larger graph, such that $V_A = (\texttt{10.0.0.1}, \texttt{10.0.0.254}, \texttt{10.0.0.2})$. For this example, $B_A = (i, i, i{+}1)$ because the IPv4 addresses `10.0.0.1` and `10.0.0.254` were added to V_A during the i:th round, while the next round added also `10.0.0.2`.

This dynamic addition of vertices and edges catalyzes to the learning process in DNS graph mining [16]. In a typical data mining application, the vertices and edges are added iteratively either during resolving or afterwards with an order-preserving database collection. When all $i = 1, \ldots, q$ resolving rounds have been processed, the resulting graph can be called a "learned graph". Once such a learned graph is available, an observable relational representation is also ready for empirically evaluating the contextual questions of interest.

Given the ever so slow adoption rate of the IPv6 protocol, an AAAA-based bipartite DNS graph offers a good small case for further illustrating the bookkeeping structure. Consider thus Fig. 3, which shows a learned graph for all IPv6 addresses in the previously noted dataset. Each rectangle denotes a FQDN in V_D, while IPv6 addresses in V_{AAAA} are represented by darker colored circles. Instead of using the labels for the AAAA records, each of the circles is labeled with the given $b \in B_{AAAA}$, that is, these labels represent the resolving round at which a given IPv6 address vertex was added to the graph.

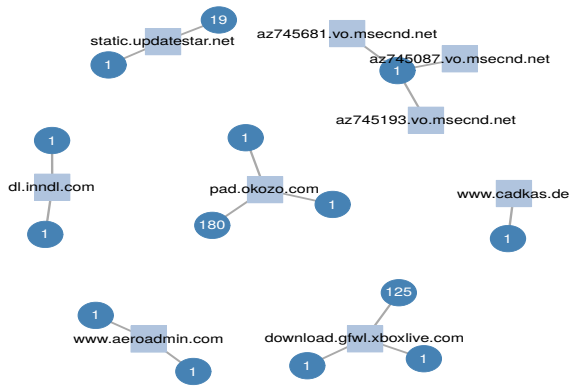

Fig. 3. A DNS Graph (learned G_{AAAA}; sampled from [23])

As can be seen, the graph is small due to the large amount of domains without active IPv6 addresses. Interestingly, though, many of the shown domains satisfy $\mathrm{c}(2)$, meaning that the domains eventually resolved to more than one IPv6 address. As an interesting further detail, one AAAA record was also mapped to three domains. More importantly, however, the resolving round integers visualized within the circles indicate that particularly two IPv6 addresses were learned relatively late. From these two domains, the one at the center of the graph is the same `pad.okozo.com` that was already visualized in the top-right plot in Fig. 2. Both figures also convey the same message in terms of the agility bias: if the resolving would have been stopped at $q = 100$, say, the learned G_{AAAA} would have missed two vertices in V_{AAAA} and two edges in E, although V_D would have been unaffected. Thus, the bias would have been rather small. Another way to look at the agility bias relates to the sampling characteristics. If a sample contains extreme cases like the two bottom row domains in Fig. 2, the bias is likely a result from outliers rather than overall agility.

C. The Agility Bias

Agility bias is understood to refer to the statistical effect of the dynamic domain-address mappings upon the construction and underlying quantifiable characteristics of a DNS graph. To study the effects, two different biases are considered next.

1) Learning Bias: The first bias is used to examine the effect of DNS agility on the longitudinal learning of a DNS graph. To carry out the empirical evaluation, an already learned G_R is pruned by iteratively removing vertices and edges in two steps. For each $j = 1, \ldots, q$, (a) a subset of R-type vertices

$$S_{R_j} = \{v \mid v \in V_R, b \in B_R, b > j\}, \quad S_{R_j} \subseteq V_{R_j}, \qquad (10)$$

is removed from G_{R_j}. In terms of the G_{AAAA} graph in Fig. 3, the first iteration at $j = 1$ removes $|S_{AAAA_1}| = 3$ vertices, for instance. To account for the potential disconnected domain name vertices, (b) the graph is pruned afterwards at each step by removing all unconnected vertices with a degree of zero. Thus, the learning bias answers to a "what if" question; how much information would have been lost if the learning would have been stopped at some $k < q$? To answer to the question, different graph metrics can be applied at each j to a potentially reduced G_{R_j} that has been processed via the twofold routine.

2) Sampling Bias: The sampling bias relates to a different viewpoint on the biases that agility causes for DNS graph mining. The question asked is: how a learned graph changes when extremely agile domains are excluded from the graph? The question and the term sampling both convey a statistical logic: for instance, if the intention would be to study a sample of non-agile domains resolving to their single unique addresses, even a single case misselection, such as the inclusion of a CDN-powered domain, can cause a severe bias—insofar as the sample should represent a population of non-agile domains.

To measure this kind of a sampling bias, for each domain, the exclusion criteria in (8) is iterated according to a sequence of threshold scalars $\tau_j \in (1, 2, \ldots, \tau_m)$, $j \leq m$, (a) removing the corresponding FQDN vertices in V_D for which a given $\mathrm{c}(\tau_j)$ is not satisfied. It is worth noting that this reduction is exactly the same as removing vertices iteratively according to an ordered vector of degrees for all $v \in V_D$. Analogously to the learning bias, (b) disconnected vertices are afterwards pruned from the processed G_{R_j}. Finally, to fix the iteration, τ_m is set to the maximum number of unique values in ϕ_R.

When considering the agile `files.opencandy.com` and `setup.roblox.com` cases in Fig. 2, an important question is whether there even is a saturation point for the visualized cumulative counts. While the plots allow to question whether the saturation was reached with $q = 3000$, the answer should still be negative; the counts should not grow without bound [7]. That is, the elaborated sampling bias should not be expected to tend to infinity. While speculating about the actual upper bounds is beyond the scope of this paper, it can be noted that the address pools may not be overly large in current CDNs [30]. Fast flux networks are a different beast.

IV. Experimental Results

The empirical experiment is carried out by examining the elaborated agility biases for DNS graphs constructed for two separate domain name collections. Before turning to the results, these two FQDN samples should be briefly elaborated.

A. Data and Sampling

The first, large sample refers to a single snapshot obtained from the list maintained by OpenDNS (Cisco) for weekly tracking of the most popular domains queried through the open DNS resolvers of OpenDNS [31]. The list is updated weekly, each weekly snapshot containing ten thousand domains. The second sample comes from the commonly used (e.g., [28]) malc0de domain name collection [23], which is maintained for keeping track of malware files downloaded from the Internet.

Although the "benign-versus-malicious" split should be always done with care [32], the two samples can be still characterized to reflect "popular" and "suspicious" domains. No profiling or other attempts were done to evaluate whether the latter sample would actually contain malicious fast flux

networks. As already elaborated, the sample contains content delivery networks and cloud-based domains, nevertheless.

While only one weekly snapshot is used, the OpenDNS sample enlarges to a very large dataset because each of the ten thousand domains were queried for nine hundred times with a custom client-side resolver [22]. As can be seen from the summary in Table I, also the resolving took a rather long time due to infrastructural and related reasons noted in Section II-B2. This, said, the calendar time in (1) is also affected by the ten minute waiting time between resolving rounds in the OpenDNS sample. In contrast, the malc0de sample was resolved with only one minute waiting time.

TABLE I
EMPIRICAL SAMPLES

	Sample	
	1. OpenDNS [31]	2. malc0de [23]
Context	Popularity	Malware
Resolver	OpenDNS	Local ISP
Domains[a]	10,000	283
Resolving rounds	900	3000
Time delay[b]	10	1
Start of resolving	29-5-2016 9:35	29-6-2016 8:19
End of resolving	21-6-2016 10:59	5-7-2016 18:45
Aggregation	To the 2nd highest level	No (raw FQDNs)

[a] From these, only domains that resolved to one or more A or AAAA records are added to the observed G_A and G_{AAAA} graphs; [b] the delay refers to a waiting time (in minutes) between each consecutive resolving round via the reported resolvers.

The OpenDNS and malc0de samples were resolved through live DNS by using the resolvers provided by OpenDNS and a local Internet service provider (ISP). Although a large number of records were obtained, the results are presented by focusing on the "pure" DNS graphs comprised of IP addresses and FQDNs. While a domain can obviously have both IPv4 and IPv6 addresses, the analysis is carried out separately for A and AAAA records in order to tentatively evaluate whether the agility biases may generally vary across the two IP versions.

B. Results

All learned graphs are very sparse, although, interestingly, the learned IPv6 less so (see Table II). The share of domains to IPv4 addresses, $|V_D|/|V_R| \times 100$, is about 39 % and 16 % for the OpenDNS and malc0de samples, respectively, meaning that both samples are generally comprised of agile domains. Moreover, both G_{AAAA} graphs are rather small, which implies that care should be used when interpreting the IPv6 results.

The learning biases are shown in Fig. 4 and Fig. 5. In both plots, the first rows refers to IPv4 graphs, while results for the AAAA graphs are shown in the bottom rows. The left-hand side plots visualize the number of vertices and edges, whereas the right-hand shows bipartite graph density, $|E| \ / \ |V_D| \times |V_R|$. In all plots, the y-axes represent these graph quantities, whereas the x-axes denote the resolving rounds.

A brief visual inspection reveals the first observation: while the IPv6 graphs are small for making definite conclusions, the trends are roughly similar between the two IP versions. The learning bias in the malc0de sample follows a highly similar

TABLE II
DESCRIPTIVE STATISTICS (LEARNED GRAPHS)

	Sample	
	1. OpenDNS [31]	2. malc0de [23]
Vertices	29946	2063
Edges	64472	1902
Density	0.0004	0.0038
Vertices	2911	23
Edges	5859	16
Density	0.0048	0.1270

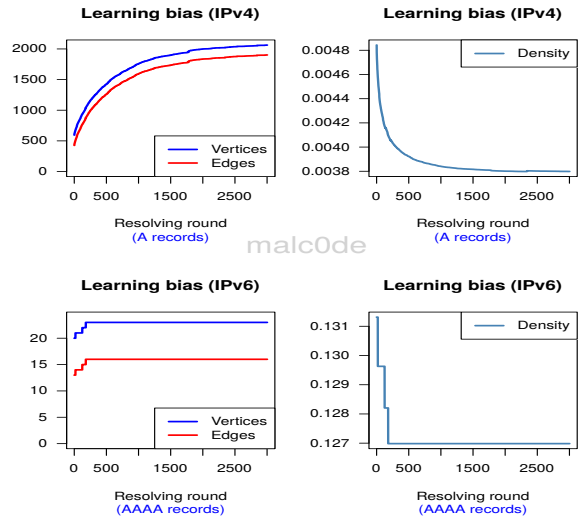

Fig. 4. Learning Bias for the malc0de Sample

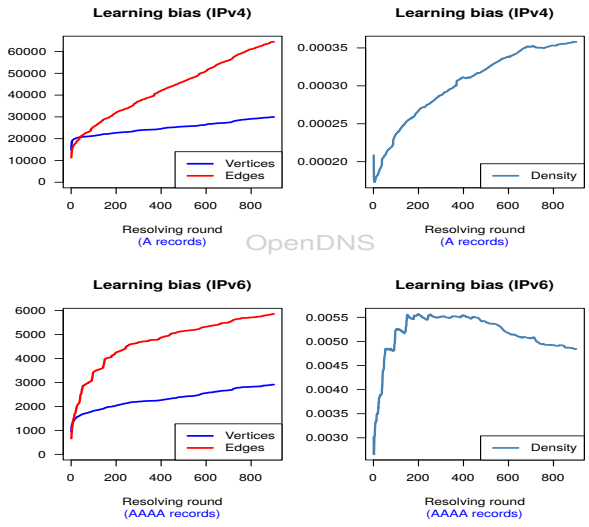

Fig. 5. Learning Bias for the OpenDNS Sample

curve than the one shown earlier for the two highly agile bottom-row domains in Fig. 2. Due to the general sparsity of DNS graphs, the density for the reduced FQDN-IPv4 graphs declines as learning increases in the malc0de graph, which

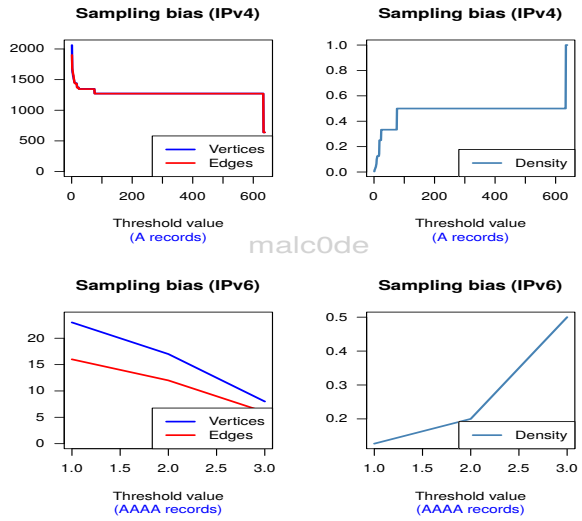

Fig. 6. Sampling Bias for the malc0de Sample

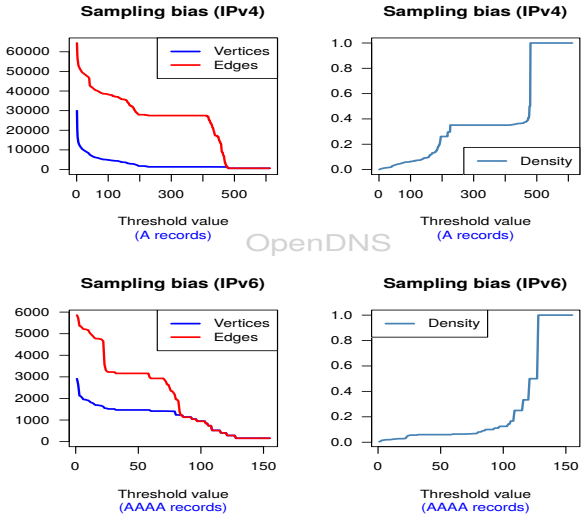

Fig. 7. Sampling Bias for the OpenDNS Sample

implies that domains and addresses are constantly added to the graph but in a way that does not connect domain vertices through address vertices. In other words, not many of the domains resolve to IPv4s to which also other domains resolve.

The contrary applies to the OpenDNS learning bias in Fig. 5. In particular, the top-left plot indicates that learning mostly involves adding new edges in this sample (cf. the red line). Although also new IPv4 addresses are constantly added, the growth rate of the vertex sets is still modest (cf. the blue line). This observation is largely explained by the nature of the sample. Because the dataset contains some of the busiest domains in the Internet, which are mostly served via CDNs and other cloud-based solutions, the IPv4 addresses are learned relatively quickly, and these addresses (or address pools) are used to serve multiple domains in the sample. To summarize, the learning bias can be severe for both samples. If the learning would have been stopped at $q = 100$, for instance, the OpenDNS sample would have missed a considerable amount of edges in particular, and the malc0de sample a considerable amount of both edges and IPv4 vertices.

Turing now to the sampling biases, the top-left plot in Fig. 6 shows that the bias in the malc0de sample is largely a result from the few highly agile, outlying domains. When domains are removed according to the number of their unique A records, the amount of vertices and edges drops sharply relatively early, remaining almost constant thereafter. Therefore, the increasing learning bias in the top-left plot in Fig. 4 is largely a result from learning the A records of the outliers, including `files.opencandy.com` and `setup.roblox.com` in Fig. 2. The sampling biases are less pronounced in the OpenDNS sample (see Fig. 7), although a similar pattern is present. All in all, also the sampling bias can be severe: a few extremely agile domains can easily dominate the structure of an otherwise relatively stable DNS graph.

V. Discussion

This empirical paper examined a so-called agility bias for DNS graph mining with two questions: how many resolving rounds are generally required for constructing domain-address mappings (learning bias), and what is the statistical effect of highly agile domains (sampling bias)? The answers to these two questions is largely in line with the existing recommendation of a two week learning period for typical DNS graph mining applications [1]. In terms of the pseudo-time resolving rounds, the $q = 3000$ choice used in the paper seems to provide a decent enough benchmark for client-side applications, amounting roughly to two weeks or less. In other words, a rather long period is required even for simple applied questions. To continue the work on mapping of CDN edge server addresses [5], a good question for further research would be to examine the potential for an optimal threshold.

When DNS graphs are constructed from server-side flow data in terms of source and destination addresses (see, e.g., [26], [33], [34]), it should be kept in mind that the destination addresses may be highly dynamic and volatile even though the clients' source addresses may be stable [20] or even entirely static. For such applications, the commonly used (see, e.g., [27], [28], [29]) alternative bipartite representations based on client (source) addresses and queried domains (instead of their addresses) may be more robust for measurement.

All in all, a word of warning can be reasonably reserved for DNS graph mining applications that utilize short learning periods. The severity of the agility bias varies from an application context to another. For instance, malware is often dropped to different file-sharing cloud services [32], which typically utilize either their own agility solutions or outsource the agility to CDNs. Consequently, the agility bias is presumably pronounced in DNS-based malware research, which likely translates to inaccuracies and other issues in machine learning applications for classifying benign and malicious domains.

ACKNOWLEDGMENTS

The authors gratefully acknowledge Tekes—the Finnish Funding Agency for Innovation, DIMECC Oy, and the Cyber Trust research program for their support.

REFERENCES

[1] A. Berger, A. D'Alconzo, W. N. Gansterer, and A. Pescapé, "Mining Agile DNS Traffic Using Graph Analysis for Cybercrime Detection," *Computer Networks*, vol. 100, pp. 28–44, 2016.

[2] Akamai Technologies, Inc., "Annual Report Pursuant to Section 13 or 15(d) of the Securities Exchange Act of 1934, Form 10-K," 2015, available online in May 2016: http://bit.ly/296kyDp.

[3] G. Haßlinger and F. Hartleb, "Content Delivery and Caching from a Network Providers Perspective," *Computer Networks*, vol. 55, no. 18, pp. 3991–4006, 2011.

[4] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling, "Measuring and Detecting Fast-Flux Service Networks," in *Proceedings of 15th Network and Distributed System Security Symposium (NDSS 2008)*, 2008, available online in February 2016: http://ei.rub.de/media/emma/veroeffentlichungen/2012/08/07/FastFlux-NDSS08.pdf.

[5] Y. Wang, Y. Shen, X. Jiao, T. Zhang, X. Si, A. Salem, and J. Liu, "Eploiting Content Delivery Networks for Covert Channel Communications," *Computer Communications*, vol. 99, pp. 84–92, 2017.

[6] H. Asghari, M. Ciere, and M. J. G. van Eeten, "Post-Mortem of a Zombie: Conficker Cleanup After Six Years," in *Proceedings of the 24th USENIX Security Symposium*. Washington: USENIX, 2015, pp. 1–16.

[7] S. Zhou, "A Survey on Fast-Flux Attacks," *Information Security Journal: A Global Perspective*, vol. 24, no. 4–6, pp. 79–97, 2015.

[8] D. J. Wang, X. Shi, D. A. McFarland, and J. Leskovec, "Measurement Error in Network Data: A Re-Classification," *Social Networks*, vol. 34, no. 4, pp. 396–409, 2012.

[9] L. Akoglu, H. Tong, and D. Koutra, "Graph Based Anomaly Detection and Description: A Survey," *Data Mining and Knowledge Discovery*, vol. 29, no. 3, pp. 626–688, 2015.

[10] N. M. Hands, B. Yang, and R. A. Hansen, "A Study of Botnets Utilizing DNS," in *Proceedings of the 4th Annual ACM Conference on Research in Information Technology (RIIT 2015)*. Chicago: ACM, 2015, pp. 23–28.

[11] J. Leyden, "RUSTOCK TAKEDOWN: How the World's Worst Botnet was KO'd: Redmond Posse Sends Bot-Herd Cowboys A-Runnin'," 2011, The Register, 23 March 2011. Available online in May 2016: http://www.theregister.co.uk/2011/03/23/rustock_takedown_analysis/.

[12] D. Andriesse, C. Rossow, B. Stones-Gross, D. Plohmann, and H. Bos, "Highly Resilient Peer-to-Peer Botnets Are Here: An Analysis of Gameover Zeus," in *Proceedings of the 8th International Conference on Malicious and Unwanted Software: "The Americas" (MALWARE 2013)*. Fajardo: IEEE, 2013, pp. 116–123.

[13] A. K. Sood and S. Zeadally, "A Taxonomy of Domain-Generation Algorithms," *Security & Privacy*, vol. 14, no. 4, pp. 46–53, 2016.

[14] K. Demertzis and L. Iliadis, "Evolving Smart URL Filter in a Zone-Based Policy Firewall for Detecting Algorithmically Generated Malicious Domains," in *Proceedings of the Third International Symposium on Statistical Learning and Data Sciences (SLDS 2015), Lecture Notes in Computer Science (Volume 9047)*, A. Gammerman, V. Vovk, and H. Papadopoulos, Eds. London: Springer, 2015, pp. 223–233.

[15] X. Hu, M. Knysz, and K. G. Shin, "Measurement and Analysis of Global IP-Usage Patterns of Fast-Flux Botnets," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM 2011)*. Shanghai: IEEE, 2011, pp. 2633–2641.

[16] J. Ruohonen, S. Šćepanović, S. Hyrynsalmi, I. Mishkovski, T. Aura, and V. Leppänen, "The Black Mark Beside My Name Server: Exploring the Importance of Name Server IP Addresses in Malware DNS Graphs," in *Proceedings of the IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW 2016)*. Vienna: IEEE, 2016, pp. 264–269.

[17] G. Farnham, "Detecting DNS Tunneling," 2013, SANS Institute InfoSec Reading Room. Available online in March 2016: http://www.sans.org/reading-room/whitepapers/dns/detecting-dns-tunneling-34152.

[18] Y. Jin, H. Ichise, and K. Iida, "Design of Detecting Botnet Communication by Monitoring Direct Outbound DNS Queries," in *Proceedings of the IEEE 2nd International Conference on Cyber Security and Cloud Computing (CSCloud 2015)*. New York: IEEE, 2015, pp. 37–41.

[19] J. Ullrich, "Command and Control Channels Using "AAAA" DNS Records," 2016, SANS Institute InfoSec Handlers Diary Blog. Available online in March 2016: https://isc.sans.edu/diary/Command+and+Control+Channels+Using+"AAAA"+DNS+Records/21301.

[20] P. K. Manadhata, S. Yadav, P. Rao, and W. Horne, "Detecting Malicious Domains via Graph Inference," in *Proceedings of the European Symposium on Research in Computer Security (ESORICS 2014), Lecture Notes in Computer Science (Volume 8712)*, M. Kutyłowski and J. Vaidya, Eds. Wroclaw: Springer, 2014, pp. 1–18.

[21] J. Ruohonen, S. Šćepanović, S. Hyrynsalmi, I. Mishkovski, T. Aura, and V. Leppänen, "Correlating File-Based Malware Graphs Against the Empirical Ground Truth of DNS Graphs," in *Proceedings of the 10th European Conference on Software Architecture Workshops (ECSAW 2016)*. Copenhagen: ACM, 2016, pp. 30:1 – 30:6.

[22] J. Ruohonen and V. Leppänen, "On the Design of a Simple Network Resolver for DNS Mining," in *Proceedings of the 17th International Conference on Computer Systems and Technologies (CompSysTech 2016)*. Palermo: ACM, 2016, pp. 105–112.

[23] Malc0de Database, 2016, Data feed available online in April 2016: http://malc0de.com/rss/.

[24] L. Deri, S. Mainardi, M. Martinelli, and E. Gregori, "Graph Theoretical Models of DNS Traffic," in *Proceedings of 9th International Wireless Communications and Mobile Computing Conference (IWCMC 2013)*. Sardinia: IEEE, 2013, pp. 1162–1167.

[25] M. Kührer, C. Rossow, and T. Holz, "Paint It Black: Evaluating the Effectiveness of Malware Blacklists," in *Proceedings of the 17th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2014), Lecture Notes in Computer Science (Volume 8688)*, A. Stavrou, H. Bos, and G. Portokalidis, Eds. Gothenburg: Springer, 2014, pp. 1–21.

[26] A. Jakalan, J. Gong, Q. Su, X. Hu, and A. M. Abdelgder, "Social Relationship Discovery of IP Addresses in the Managed IP Networks by Observing Traffic at Network Boundary," *Computer Networks*, vol. 100, pp. 12–27, 2016.

[27] D. Herrmann, C. Banse, and H. Federrath, "Behavior-Based Tracking: Exploiting Characteristic Patterns in DNS Traffic," *Computers & Security*, vol. 39, pp. 17–33, 2013.

[28] J. Lee and H. Lee, "GMAD: Graph-Based Malware Activity Detection by DNS Traffic Analysis," *Computer Communications*, vol. 49, pp. 33–47, 2014.

[29] X. Yuchi, X. Lee, J. Jin, and B. Yan, "Modeling DNS Activities Based on Probabilistic Latent Semantic Analysis," in *Proceedings of the 6th International Conference on Advanced Data Mining and Applications (ADMA 2010), Lecture Notes in Computer Science (Volume 6441)*, L. Cao, J. Zhong, and Y. Feng, Eds. Chongqing: Springer, 2010, pp. 290–301.

[30] H. Yin, B. Qiao, Y. Luo, C. Tian, and Y. R. Yang, "Demystifying Commercial Content Delivery Networks in China," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 13, pp. 3523–3538, 2016.

[31] OpenDNS, "OpenDNS Top Domains List," 2016, Data feed available online in April 2016 (weekly updates, hourly averages): https://github.com/opendns/public-domain-lists.

[32] J. Ruohonen, S. Šćepanović, S. Hyrynsalmi, I. Mishkovski, T. Aura, and V. Leppänen, "A Post-Mortem Empirical Investigation of the Popularity and Distribution of Malware Files in the Contemporary Web-Facing Internet," in *Proceedings of the European Intelligence and Security Informatics Conference (EISIC 2016)*. Uppsala: IEEE, 2016, pp. 144–147.

[33] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "Uncovering Relations Between Traffic Classifiers and Anomaly Detectors via Graph Theory," in *Proceedings of the International Workshop on Traffic Monitoring and Analysis (TMA 2010), Lecture Notes in Computer Science (Volume 6003)*, F. Ricciato, M. Mellia, and E. Biersack, Eds. Zurich: Springer, 2010, pp. 101–114.

[34] C. R. Harshaw, R. A. Bridges, M. D. Iannacone, J. W. Reed, and J. R. Goodall, "GraphPrints: Towards a Graph Analytic Method for Network Anomaly Detection," in *Proceedings of the 11th Annual Cyber and Information Security Research Conference (CISRC 2016)*. Oak Ridge: ACM, 2016, pp. 1–4.