2016 IEEE 36th International Conference on Distributed Computing Systems

BotMeter: Charting DGA-Botnet Landscapes in Large Networks

Ting Wang* Xin Hu* Jiyong Jang† Shouling Ji◇ Marc Stoecklin† Teryl Taylor‡
*Lehigh University *Pinterest †IBM Research
◇[1]Georgia Tech ◇[2]Zhejiang University ‡UNC Chapel Hill
*ting@cse.lehigh.edu *huxinsmail@gmail.com †{jjang, mpstoeck}@us.ibm.com ◇sji@gatech.edu ‡tptaylor@cs.unc.edu

***Abstract*—Recent years have witnessed a rampant use of domain generation algorithms (DGAs) in major botnet crimewares, which tremendously strengthens a botnet's capability to evade detection or takedown. Despite a plethora of existing studies on detecting DGA-generated domains in DNS traffic, remediating such threats still relies on vetting the DNS behavior of *each* individual device. Yet, in large networks featuring complicated DNS infrastructures, we often lack the capability or the resource to exhaustively investigate every part of the networks to identify infected devices in a timely manner. It is therefore of great interest to first assess the population distribution of DGA-bots inside the networks and to prioritize the remediation efforts. In this paper, we present BotMeter, a novel tool that accurately charts the DGA-bot population landscapes in large networks. Specifically, we embrace the prevalent yet challenging setting of hierarchical DNS infrastructures with caching and forwarding mechanisms enabled, whereas DNS traffic is observable only at certain upper-level vantage points. We establish a new taxonomy of DGAs that captures their characteristic DNS dynamics. This allows us to develop a rich library of rigorous analytical models to describe the complex relationships between bot populations and DNS lookups observed at vantage points. We provide results from extensive empirical studies using both synthetic data and real DNS traces to validate the efficacy of BotMeter.**

I. INTRODUCTION

A botnet is a large number of malware-infected devices (*bots*) that may be remotely controlled by their operators through *command-and-control* (C2) channels. As the workhorse of varied large-scale attacks (e.g., DDoS, email spamming, key logging, click fraud), botnets represent a major threat to Internet security. A fundamental aspect of any botnet is that of coordination, i.e., how the bots identify and communicate with their botmasters (C2 servers). Conventionally, bots locate C2 servers using their IP addresses, DNS names, or node IDs in peer-to-peer overlays [18].

Over recent years, a rampant use of *domain generation algorithms* (DGAs) has been witnessed in major botnet crimewares (e.g., *Torpig*, *Conficker*, *NewGoZ*) [1]. By using DGA, each bot dynamically and independently generates a list of pseudo-random domains. It then attempts to contact domains on the list sequentially until one succeeds (i.e., the domain resolves to an IP address and the corresponding server provides a valid response) or aborts after a certain number of trials. The botmaster needs to register only a few domains on the list to serve as C2 servers. For example, each *Conficker.C* worm [12] generates 50K random domains everyday, among which it attempts to contact up to 500 domains, giving itself 1% of chance of being updated if the botmaster registers only one domain per day.

The use of DGA significantly strengthens a botnet's capability to evade detection or takedown. Foremost, the sheer number of potential rendezvous points makes it extremely difficult to preemptively eliminate C2 channels (e.g., blacklisting or pre-registration). Moreover, even if the current C2 domains or IPs are captured and taken down, the bots will eventually identify the relocated C2 servers via looking up the next set of automatically generated domains. Further, due to the use of public-key encryption, it is infeasible to mimic communication from the botmasters for the bots will reject any commands not signed by their botmasters.

Varied techniques have since been proposed to detect DGA-domains in DNS traffic, including analyzing algorithmic patterns of domains [25], reverse-engineering malware instances [16, 8, 20, 12], clustering non-existent domains in DNS lookups [7, 23], and directly capturing C2 traffic [10, 13, 15]. However, our understanding of how to leverage detected DGA-domains to remediate practical botnet threats in large-scale networks is still fairly limited.

At a first glance, this may seem a simple problem with trivial solutions, e.g., capturing C2 channels and tracing back to infected machines [22], or detecting DGA-domains and vetting the DNS behavior of each individual device to identify positive matches. Unfortunately, these naïve solutions face major challenges in practice. First, due to the randomness nature of DGA, only a small portion of bots may successfully establish connections with C2 servers everyday; further, the captured C2 may be valid only for a short timespan, as the bots quickly roll over to relocated C2 domains. Second, in large networks, hundreds of thousands of machines are often geographically distributed and probably managed by different entities. Exhaustively investigating every part of the networks can be prohibitively expensive in terms of operational costs. In face of rapidly emerging botnet threats, yet with limited resources or accesses, this may considerably delay addressing the threats in a timely manner. It is thus of great interest to first quickly assess the populations of DGA distributed over the networks and to prioritize the remediation efforts.

1063-6927/16 $31.00 © 2016 IEEE
DOI 10.1109/ICDCS.2016.77

Concretely, we assume confirmed domains generated by target DGAs and aggregated DNS lookups observed at certain vantage points. We remark that this assumption is realistic. First, varied off-the-shelf techniques (e.g., [25, 16, 8, 20, 12, 7]) are available to effectively identify DGA-domains. Second, the DNS infrastructures of large networks are often hierarchical with *caching*-and-*forwarding* mechanisms enabled [6]. Upper-level DNS servers only observe aggregated DNS traffic forwarded by lower-level servers; meanwhile, local servers only forward lookups missed in their caches. Due to operational costs, DNS traffic is often visible and collectable only at certain upper-level DNS servers.

Embracing the challenging setting, we present BOTMETER, a novel tool that accurately assesses DGA-bot populations by exploiting the temporal and semantic patterns inherent in DNS dynamics of DGAs. To build BOTMETER, we first establish a new taxonomy of DGAs based on their characteristic domain fluxing behaviors. This allows us to develop a rich library of rigorous analytical models to capture the DNS dynamics of a broad range of DGAs. Compared with alternative solutions, BOTMETER departs in significant ways: (i) it works on highly aggregated DNS traffic; (ii) it takes account of the impact of caching-and-forwarding mechanisms in DNS infrastructures; (iii) it is backed by a rich library of estimation models designed for a range of DGA families; and (iv) it is resilient against noisy and missing observations. To the best of our knowledge, BOTMETER is the first non-intrusive solution capable of accurately assessing DGA-bot populations while incurring minimal operational costs. Our contributions can be summarized as follows.

- We articulate the problem of charting DGA-botnet landscapes in large networks using aggregated DNS lookups observed at a few upper-level vantage points.
- We establish a new DGA taxonomy based on their characteristic domain-fluxing behaviors. For a wide range of DGA families, we develop rigorous analytical models to capture the complex relationships between bot populations and DNS lookups observed at vantage points.
- We implement BOTMETER, a novel tool that realizes all these ideas. We extensively evaluate its empirical performance using both synthetic and real DNS traces. The promising results indicate that BOTMETER is able to accurately assess the severity of botnet infection, thereby helping analysts quickly navigate the threat landscapes of their networks and prioritize the remediation efforts.

II. PRELIMINARIES

A. Life Cycle of DNS Lookup Queries

As one of the backbone components of Internet, the *Domain Name System* (DNS) resolves domain names to corresponding IP addresses. For an invalid domain name, "NXDomain" (NXD) is returned. The DNS infrastructure of a large network is typically hierarchical. A DNS lookup query is first processed by the client's local DNS server. For efficiency purpose, a caching-and-forwarding mechanism is often enabled. Concretely, the local server stores previously queried domains and responses, each record valid for a certain period (TTL). Given a DNS lookup query, the server first attempts to answer it using its cached results. Only when a miss occurs, the query is forwarded to an upper-level DNS server; once the response is returned, the cache at the local server is updated accordingly.

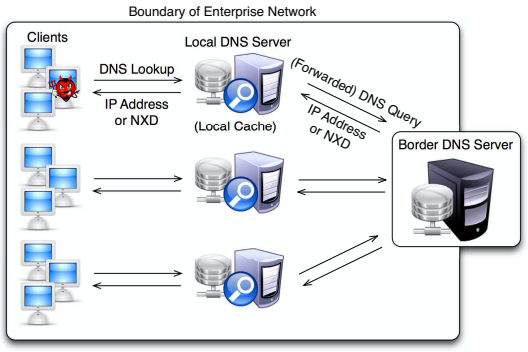

Figure 1: A hierarchical DNS architecture in a large network.

B. Visibility of DNS Traffic

Vantage Point - Because of operational costs, oftentimes, DNS traffic is visible and collectable only at certain vantage points. We assume that the vantage points are set at the border DNS servers, as shown in Figure 1, which collect DNS lookups forwarded by lower-level DNS servers. Also we assume that the forwarded lookups come in as a stream of tuples of the form: $\langle$timestamp t, forwarding server s, domain $d\rangle$, with each tuple representing a lookup for domain name d issued at time t and forwarded by server s.

Positive and Negative Caching - A DNS lookup will not be forwarded to the border DNS server (i.e., invisible) if it is found in the cached results, being it a valid domain (positive) or a NXD (negative). Thus, at border DNS servers we only observe cache-filtered DNS lookups. It is noted that the TTLs of valid domains and that of NXDs are often different. As IETF suggests, positive TTLs are typically one to several days while negative TTLs varies from minutes to hours [11].

Detection Window - A number of DGAs use current dates as their seeds of randomness and are executed on a daily basis. We therefore use a uniform model to describe their DNS behaviors. Everyday from a pool of pseudo-random domains, each bot selects a subset of them to query the DNS server. Ideally a perfect DGA-domain detection (D3) algorithm is able to detect all the domains in the pool. However, in reality, due to all sorts of constraints (e.g., the botnet malware may be updated frequently or it may

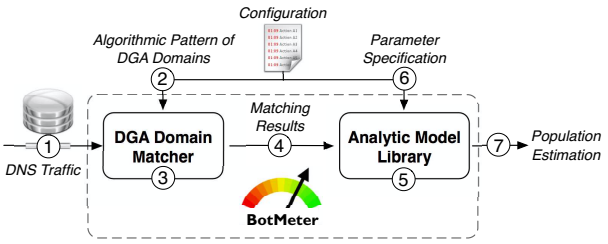

Figure 2: Architecture of BOTMETER

be obfuscated and only decrypted and executed by external triggers such as time), the D3 algorithm is usually often able to detect a part of the pool, which we refer to as its *detection window*. Furthermore, a small number of domains in the pool may coincide with valid domains, which we refer to as collision cases.

C. BotMeter in A Nutshell

We now formulate the problem of *DGA-bot population assessment*: given the detection window for domains generated by a target DGA, via analyzing DNS lookups observed at a border DNS server as forwarded by lower-level DNS servers, we intend to accurately estimate the population of bots that employ this particular DGA within the network behind each local DNS server.

We present BOTMETER, a novel tool built for this task. Figure 2 illustrates its high-level architecture. Tapped at a border DNS server (①), BOTMETER analyzes DNS lookups forwarded by lower-level DNS servers. BOTMETER allows analysts to specify algorithmic patterns (or plain lists) of domains generated by targeted DGAs (②) and matches incoming DNS traffic against such patterns (③). The matching results (including timestamps, domains of matched DNS lookups, and corresponding local DNS servers) are fed as input to the next phase (④). The workhorse of BOTMETER is a library of analytical models designed for a range of DGA families (⑤). Through the configuration interface (⑥), the analysts select the most appropriate analytical model, specify key parameters (e.g., detection window size), and perform bot population estimation using the matching results.

To build BOTMETER, we need a new taxonomy to capture DGA-specific DNS dynamics. In § III, we establish such a taxonomy, allowing us to develop the library of analytical models for BOTMETER in § IV.

III. A NEW DGA TAXONOMY

We start with a simple model of DGA, branch from it, and instantiate a broad range of DGA families.

Everyday from a *query pool* of $(\theta_\emptyset + \theta_\exists)$ pseudo-random domains[1], the botmaster selects a subset of $\theta_\exists$ domains (typically a few, i.e., $\theta_\exists \ll \theta_\emptyset$) and registers them as C2 servers, with the remaining $\theta_\emptyset$ being invalid NXDs; meanwhile, each active bot attempts to query its local DNS server with up to θ_q domains (which we refer to as the *query barrel*) chosen from this pool until hitting one valid domain or aborts otherwise.

We now differentiate DGA families based on (i) how the query pool is generated, (ii) how the query barrel is selected, and (iii) the concrete setting of $\theta_\exists$, $\theta_\emptyset$, and θ_q.

A. Query Pool Model

Drain-and-Replenish Pool - This is perhaps the simplest query pool model, wherein the entire query pool is replaced on a regular basis (e.g., daily). A majority of DGAs follow this model (e.g., *Murofet*, *Srizbi*, *Conficker*, *GameoverZeus*).

Sliding-Window Pool - Some recent DGAs (e.g., *Ranbyus* [5] and *PushDo* [2]) adopt a sliding-window pool model. On a regular basis, a set of new pseudo-random domains are generated to replace a set of "expired" domains in the query pool. This gives DGAs the benefit of fast changing domains in case that old domains are blocked or sinkholed, while at the same time enabling older domains to be re-used as long as they still work. For example, *Ranbyus* generates a fresh set of 40 domains everyday and also maintains the domains generated in the past 30 days, i.e., a pool of 1,240 domains; *PushDo* maintains a window of -30 to +15 days of 30 domains per day, giving it a query pool of 1,380 domains.

Multiple Mixture Pool - To further increase the resilience against takedown, a few recent DGAs introduce the multiple mixture pool model. Each bot employs mutliple identical DGA instances, but with different seeds and parameter settings, to generate multiple sets of interleaved domains. One DGA generates useful domains while the others generate noisy ones. For example, *Pykspa* [14], employs two identical DGA instances, one generating a pool of 200 useful domains and the other generating a pool of 16K noisy domains.

B. Query Barrel Models

Uniform Barrel - This is the simplest as well as the most popular query barrel model, adopted by a majority of DGA families (e.g., *Murofet*, *Srizbi*, *Torpig*). Under this model, a bot simply queries all the domains in the query pool following the order they have been generated.

Sampling Barrel - Represented by *Conficker.C* [12], this class features a query barrel as a randomly sampled subset from the query pool. For example, everyday a *Conficker.C* bot generates a pool of 50K domains, among which it randomly selects up to 500 of them to form its query barrel. In contrast of the uniform barrel model, this model provides bots with stronger resilience against detection (i.e., bots tend to query different subsets of domains), but at the cost of lower success rate to establish bot-to-botmaster communications.

[1]Note that because the botmaster and bots share the same DGA, this query pool is known to both of them.

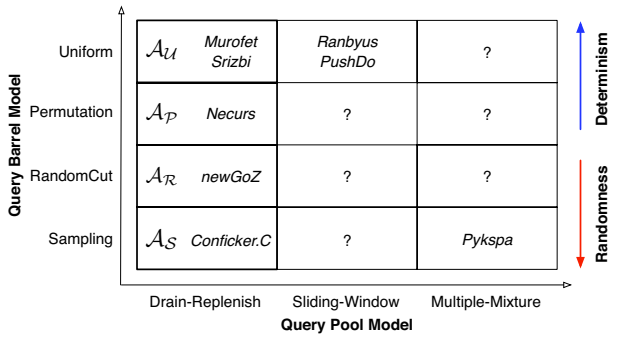

Figure 3: A taxonomy of DGAs and representative DGA families ("?" indicates that DGAs following the particular model have not been spotted in the wild).

RandomCut Barrel - This class defines a global sequential order over domains in the query pool. Each active bot randomly picks a sequence number as the starting point and considers the next $\theta_\mathtt{q}$ domains (modular arithmetically) as its query barrel. For example, each *newGoZ* bot constructs its query barrel by randomly selecting 500 consecutive domains from a sequence of 10K domains [3]. Interestingly, the model strikes a balance between the uniform and sampling models in that it features more randomness than the former due to using random starting points but more determinism than the latter due to enforcing a global sequential order.

Permutation Barrel - For DGAs belonging to this class, the query pool is re-generated regularly, meanwhile each bot attempts to look up all the domains in the pool in a randomly shuffled order. In other words, the query barrel is a random permutation of the query pool. For example, *Necurs* [4], adopts this barrel model. A *Necurs* bot changes its query pool every four days, which contains 2,048 domains; everyday the bot queries these domains in a random permutation order. Similar to the randomcut model, this one achieves a balance between detection-resilience and coordination-simplicity.

C. Summaries

With the query pool and barrel models as reference, we establish a new taxonomy of DGAs, as illustrated in Figure 3. Letting the horizontal and vertical axes represent query pool and barrel models respectively, we partition the "universe" of DGAs into twelve categories, each corresponding to one particular barrel-model combination. It is clear that compared with existing DGA taxonomies, this new taxonomy well captures the characteristic DNS dynamics of DGAs. As we will show shortly, this taxonomy facilitates to develop DGA-bot population estimation models applicable for a wide range of DGA families, for we only need to design one analytical model for all DGA families following the same pool-barrel-model combination.

The space limitations preclude the possibility of exploring all the combinations of query pool and barrel models. In the following, we focus on the most popular pool model (i.e., the drain-and-replenish pool) and consider all the barrel models. We use $\mathcal{A}_\mathcal{U}$, $\mathcal{A}_\mathcal{S}$, $\mathcal{A}_\mathcal{R}$, and $\mathcal{A}_\mathcal{P}$ to denote uniform-, sampled-, randomcut-, and permutation-barrel DGAs, respectively. Next we elaborate bot population estimation models designed for DGAs in these categories.

IV. Analytical Models

A. Preliminaries

We first introduce a set of fundamental concepts. Given a DGA-bot, each round execution of its DGA is called an *activation*; the timespan of an activation is called its *duration*; the temporal gap between two consecutive DGA-triggered DNS lookups is referred to as a *query interval*; the waiting time between two consecutive activations is an *epoch*; finally, the timespan for which a DNS server caches a DNS response is determined by the setting of *time to live* (TTL). We use $\delta_\mathtt{i}$, $\delta_\mathtt{e}$, $\delta_\mathtt{d}$, and $\delta_\mathtt{l}$ to denote query interval, epoch, duration, and TTL, respectively.

Typically, $\delta_\mathtt{e}$ is of the granularity of a day, while $\delta_\mathtt{l}$ varies from minutes to hours for NXD [11]. For most DGAs, there are minimal intervals (e.g., $\delta_\mathtt{i} = 500ms$) between consecutive DGA-triggered DNS lookups. Thus $\delta_\mathtt{d}$ is negligible compared with $\delta_\mathtt{e}$ or $\delta_\mathtt{l}$. For simplicity, we assume $\delta_\mathtt{e}$ = one day and $\delta_\mathtt{e}$ is a multiple of $\delta_\mathtt{l}$. Following the DGA model in § III, the query pool comprises ($\theta_\exists$ +$\theta_\emptyset$) domains, among which $\theta_\exists$ are registered as C2 servers with $\theta_\emptyset$ being NXDs; a bot attempts to resolve up to $\theta_\mathtt{q}$ domains until it either hits a valid domain or aborts otherwise.

The current implementation of BotMeter includes three analytical models: *Timing* estimator ($\mathcal{M}_\mathcal{T}$), *Poisson* estimator ($\mathcal{M}_\mathcal{P}$), and *Bernoulli* estimator ($\mathcal{M}_\mathcal{B}$).

B. Timing Estimator ($\mathcal{M}_\mathcal{T}$)

We start with the Timing estimator (denoted by $\mathcal{M}_\mathcal{T}$), which intuitively differentiates DNS lookups belonging to different bots based on their temporal traits.

Specifically, $\mathcal{M}_\mathcal{T}$ leverages three kinds of traits. First, during a one-epoch-long time window, two lookups regarding a same NXD are likely to be generated by different bots. Second, two lookups with a gap longer than the maximum possible duration (i.e., $\theta_\mathtt{q} \cdot \delta_\mathtt{i}$) are attributed to different bots. Finally, the query interval of a given DGA is typically fixed (e.g., $1sec$ for *newGoZ*); that is, each activation produces a train of lookups with regular temporal intervals [9]. Thus, if the timestamp granularity is fine enough and the timestamps faithfully reflect the actual issuing time of DNS lookups, $\mathcal{M}_\mathcal{T}$ is able to discern lookups of different bots if they are not separated by a multiple of query intervals. For example, given $\delta_\mathtt{i} = 500ms$, two lookups with a temporal gap of $750ms$ are likely to belong to different bots.

The workflow of $\mathcal{M}_\mathcal{T}$ is sketched in Algorithm 1. It maintains a list $\mathcal{L}$, each entry corresponding to one distinct

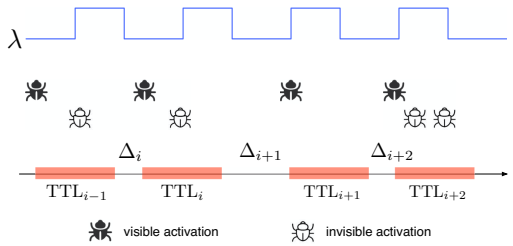

Figure 4: (In)-visible activations of $\mathcal{A}_\mathcal{U}$, TTL of negative caching, and Poisson estimator $\mathcal{M}_\mathcal{P}$.

bot. It applies the aforementioned three heuristic rules to each tuple in the sequence of DNS lookups. If the lookup is considered to belong to a bot existing in the list, it is "absorbed" and the domain list of the corresponding entry is updated. Otherwise, a new entry representing an unseen bot is created. Eventually the number of entries in $\mathcal{L}$ indicates the number of bots $\mathcal{M}_\mathcal{T}$ has discovered.

Algorithm 1: Timing Estimator $\mathcal{M}_\mathcal{T}$

Input: $\mathcal{S}$ - sequence of tuples $\{\langle t, d\rangle\}$; $\theta_\mathtt{q}$- maximum barrel size; $\delta_\mathtt{i}$- query interval
Output: estimation of bot population $\mathtt{N}$

```
// classification of lookups to distinct bots
L ← ∅;
for each tuple ⟨timestamp t, domain d⟩ ∈ S do
    if L is empty then add (t, {d}) to L and continue;
    for each entry (timestamp t*, domains D) ∈ L do
        // heuristic #1
        if d appears in D then continue;
        // heuristic #2
        if t* + δ_i · θ_q ≤ t then continue;
        // heuristic #3
        if (t − t*) mod δ_i ≠ 0 then continue;
        add d to D and break;
    if ⟨t, d⟩ is not "absorbed" then add entry (t, {d}) to L;
return number of entries in L as N;
```

C. Poisson Estimator ($\mathcal{M}_\mathcal{P}$)

Solely relying on the temporal traits of DNS lookups, $\mathcal{M}_\mathcal{T}$ is applicable to all the DGA models in Figure 3. However, it faces a major challenge in estimating $\mathcal{A}_\mathcal{U}$ bot populations: according to its definition (cf. § III-B), DGA-bots employing $\mathcal{A}_\mathcal{U}$ generate identical query barrels during each epoch; due to the caching effect at local DNS servers, if multiple bots are activated within a same time window of length TTL, only the first one is visible, as illustrated in Figure 4. It is thus infeasible to estimate bot population directly from observable DNS lookups. The key challenge here is to infer the number of bots whose activations have been "masked" by the caching mechanism.

To address this challenge, we model the activations of DGA-bots as a Poisson process, a stochastic process widely applied to model punctual phenomena wherein events occur independently of each other [17]. We introduce the Poisson estimator (denoted by $\mathcal{M}_\mathcal{P}$) with the intuitive idea of (i) first estimating the average activation rate λ of this Poisson process and (ii) then determining the total number of activations that occur during the given observation window.

Let Δ_i denote the gap between the end of $(i-1)$-th TTL and the start of i-th TTL, as shown in Figure 4. We consider $\{\Delta_i\}$ as an indication of the activation rate λ. Intuitively, a larger bot population leads to a shorter "waiting time" before the next bot to be activated. Considering n TTLs with temporal gaps $\{\Delta_i\}_{i=1}^n$[2], the expectation of average activation rate λ is derived as: $\mathbb{E}(\lambda) = n/\sum_{i=1}^n \Delta_i$.

The expectation of total number of active bots appearing in the given observation window is thus given by:

$$\mathbb{E}(\mathtt{N}) = \mathbb{E}(\lambda)\sum_{i=1}^{n}(\Delta_i + \delta_1) = n + \frac{n^2\delta_1}{\sum_{i=1}^n \Delta_i} \qquad (1)$$

D. Bernoulli Estimator ($\mathcal{M}_\mathcal{B}$)

$\mathcal{A}_\mathcal{R}$ differs from the rest DGA models in that it dictates a global sequential order over domains in the query pool and each bot randomly selects a sequence number as the starting point to query (cf. § III-B). Such unique characteristics demand a customized estimator. We introduce the following model to describe $\mathcal{A}_\mathcal{R}$ (as shown in Figure 5): the DGA-domains form a circle with the clockwise direction indicating the order to be queried; the set of $\theta_\exists$ valid domains partition the circle into $\theta_\exists$ arcs and serve as arc boundaries; each bot randomly picks one domain on the circle and selects (clockwise) the next $\theta_\mathtt{q}$ domains to query or stops on hitting an arc boundary.

All the distinct NXDs queried by bots during an epoch form a set of *segments*, each comprising consecutive NXDs. Given the set of arcs and their associated segments, we infer the number $\mathtt{N}$ of bots responsible for generating NXDs to "cover" all such segments. We differentiate two types of segments, those ended in the middle of an arc (m-segment) and those ended at an arc boundary (b-segment). Consider a random variable q obeying the following distribution:

$$\Pr(q=i) = \begin{cases} \frac{\theta_\exists}{i+1}\binom{\theta_\emptyset}{i}\Big/\binom{\theta_\exists+\theta_\emptyset}{i+1} & 0 \le i < \theta_\mathtt{q} \quad \text{(a)} \\ \binom{\theta_\emptyset}{\theta_\mathtt{q}}\Big/\binom{\theta_\exists+\theta_\emptyset}{\theta_\mathtt{q}} & i = \theta_\mathtt{q} \quad \text{(b)} \\ 0 & \text{otherwise} \quad \text{(c)} \end{cases} \qquad (2)$$

where $\binom{}{}$ is the binomial coefficient. (a) and (b) correspond to that the bot hits a valid domain and that it aborts after trying $\theta_\mathtt{q}$ lookups respectively. It is noticed that 1) an m-segment must be covered by bots in (b), 2) a b-segment of length less than $\theta_\mathtt{q}$ must be covered by bots in (a), and 3) a b-segment of length no less than $\theta_\mathtt{q}$ is covered by a mixture of bots in (a) and (b).

[2] Δ_1 corresponds to the elapse time between the start of observation window and the activation of the first bot.

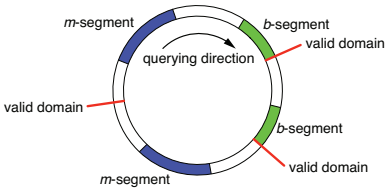

Figure 5: Illustration of DNS dynamics of $\mathcal{A}_{\mathcal{R}}$

Based on these observations, we derive the expectation of bot population required to cover a given m- or b-segment. Let $f(\tilde{l},n,m) = \frac{m!}{l^n}\binom{\tilde{l}}{m}\left(\left\{{n \atop m}\right\} - \tilde{l}\left\{{n-1 \atop m}\right\}\right)$ and $g(\tilde{l},m) = \sum_{k=0}^{\left\lfloor\frac{\tilde{l}-2}{\theta_{\mathtt{q}}}\right\rfloor}(-1)^k\binom{m-1}{k}\binom{\tilde{l}-k\theta_{\mathtt{q}}-2}{m-2}/\binom{\tilde{l}-2}{m-2}$, where $\left\{\right\}$ represents Stirling numbers of the second kind. We define:

$$h(\tilde{l},n) = \sum_{m=1}^{\tilde{l}} f(\tilde{l},n,m)g(\tilde{l},m) \tag{3}$$

Theorem 1. *Given a segment* $\mathtt{L}$ *of length* l*, let* $l_l = l-\theta_{\mathtt{q}}+1$ *and* $l_u = l_l$ *if* $\mathtt{L}$ *is an m-segment and* $l_u = l$ *otherwise. The expectation of bot population* $\mathtt{N}_{\mathtt{L}}$ *required to cover* $\mathtt{L}$ *is given by:* $\mathbb{E}(\mathtt{N}_{\mathtt{L}}) = \sum_{n=1}^{+\infty} n \sum_{\tilde{l}=l_l}^{l_u} h(\tilde{l},n)$.

Proof: Referred to our technical report [24]. ■

Theorem 1 naturally leads to an estimation model, the Bernoulli estimator ($\mathcal{M}_{\mathcal{B}}$), with details referred [24].

V. EMPIRICAL EVALUATION

In this section, we empirically evaluate the performance of BOTMETER using both synthetic data and real DNS traces collected from a large enterprise network over a one-year timespan. Specifically, with synthetic data, we intend to measure (i) the *accuracy* of estimation with respect to different DGA models and parameter settings and (ii) the *robustness* of estimation against noisy and missing observations as well as network dynamics; with real DNS traces, we intend to evaluate the efficacy of BOTMETER in actual deployment.

A. Evaluation over Synthetic Data

We first implement a set of simulators generating realistic DNS traffic according to different DGA models. Specifically, we fix the drain-and-replenish model (cf. § III-A) as the pool model and consider varied barrel models $\mathcal{A}_{\mathcal{U}}$, $\mathcal{A}_{\mathcal{S}}$, $\mathcal{A}_{\mathcal{R}}$, and $\mathcal{A}_{\mathcal{P}}$ (cf. § III-B). The default parameter setting is as follows: epoch = 1 day, length of observation window = 1 day, negative cache TTL = 2 hour, positive cache TTL = 1 day, and timestamp granularity = 100*ms*. The setting of DGA-specific parameters is listed in Table I.

Given a population of $\mathtt{N}$ bots, we model their activations as a Poisson process, a counting process widely applied to model networking events [17]. To capture varying network dynamics, we consider two variants of processes, one with a constant activation rate $\lambda_0 = \mathtt{N}/\delta_{\mathtt{e}}$, and the other with this rate changing dynamically for every bot. Specifically, for the i-th bot to be activated, its activation rate (since the activation of its predecessor) is given by $\lambda_i = \lambda_0 e^{\kappa_i}$, where κ_i is randomly sampled from the normal distribution $\mathcal{N}(0,\sigma^2)$. We control the dynamics of bot activation rate through the variance parameter σ; intuitively, larger σ implies more dynamically varying activation rate.

DGA Model	Prototype	$\theta_{\emptyset}$	$\theta_{\exists}$	$\theta_{\mathtt{q}}$	$\delta_{\mathtt{i}}$
$\mathcal{A}_{\mathcal{U}}$	*Murofet*	798	2	798	500*ms*
$\mathcal{A}_{\mathcal{S}}$	*Conficker.C*	49995	5	500	1*sec*
$\mathcal{A}_{\mathcal{R}}$	*newGoZ*	9995	5	500	1*sec*
$\mathcal{A}_{\mathcal{P}}$	*Necurs*	2046	2	2046	500*ms*

Table I. DGA-specific parameter setting.

For a given DGA $\mathcal{A}$, the simulator outputs a sequence of tuples $\langle$timestamp t, client c, domain $d\rangle$, each representing a DNS lookup regarding domain d (generated by $\mathcal{A}$) issued by an infected client c at time t. We then apply the caching and forwarding mechanism (i.e., acting as a local DNS server) over this sequence and generate a sub-sequence of tuples of the form $\langle$timestamp t, domain $d\rangle$, each representing a DNS lookup forwarded by the local server to the boarder DNS server (i.e., observable by BOTMETER). Note that the client information is invisible in the forwarded DNS lookups.

Experiment Setup

We measure the accuracy of the estimators in BOTMETER under varied parameter settings. We use the metric of *absolute relative error* (ARE) to quantify estimation accuracy:

$$\mathsf{ARE} = \left|\frac{\text{estimated population} - \text{actual population}}{\text{actual population}}\right| \tag{4}$$

We consider five key parameters: (i) actual bot population, (ii) size of observation window, (iii) negative cache TTL, (iv) dynamics of bot activation rate, and (v) detection window of D3 algorithm. In each set of experiments, with the remaining parameters fixed, we vary one particular parameter and observe its influence on different estimators. Specifically, we apply the Timing estimator ($\mathcal{M}_{\mathcal{T}}$) to all the DGA models, the Poisson estimator ($\mathcal{M}_{\mathcal{P}}$) to $\mathcal{A}_{\mathcal{U}}$, and the Bernoulli estimator ($\mathcal{M}_{\mathcal{B}}$) to $\mathcal{A}_{\mathcal{R}}$.

Experiment Results

Population of DGA-bots - We measure the accuracy of different estimators as a function of bot population $\mathtt{N}$ in the network. As shown in Figure 6 (a), when $\mathtt{N}$ varies from 16 to 256, we observe shrinking error bars (25*th*-75*th* percentile of errors) across all the estimators in the cases of $\mathcal{A}_{\mathcal{U}}$, $\mathcal{A}_{\mathcal{S}}$, and $\mathcal{A}_{\mathcal{R}}$. This is because ARE is essentially a relative metric (cf. Eqn (4)), amplifying the difference of absolute errors when $\mathtt{N}$ is small. However, as $\mathtt{N}$ increases, $\mathcal{M}_{\mathcal{T}}$ suffers significant accuracy loss in the case of $\mathcal{A}_{\mathcal{U}}$. This is expected, because more active bots result in higher possibility of "collision" between their DNS behaviors (i.e., more lookups of distinct bots overlap and become invisible due to DNS caching), significantly impairing the temporal "discernibility" of bots from the perspective of $\mathcal{M}_{\mathcal{T}}$. In opposite, in the cases of

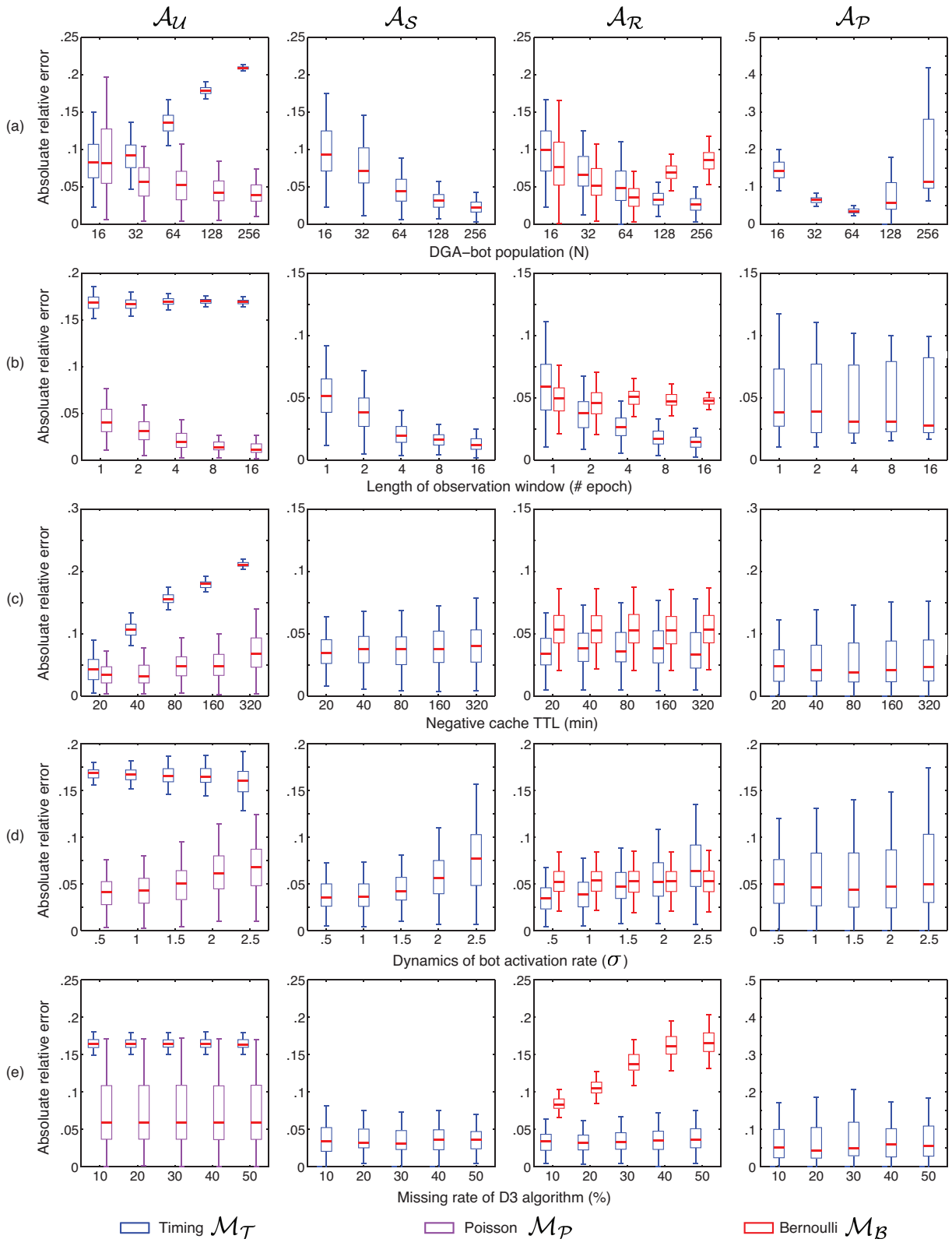

Figure 6: Estimation accuracy of BOTMETER with respect to different DGA families and parameter settings.

$\mathcal{A}_\mathcal{S}$ and $\mathcal{A}_\mathcal{R}$, we observe monotonic improvement in the accuracy of $\mathcal{M}_\mathcal{T}$. This may be explained by that $\mathcal{M}_\mathcal{T}$ leverages more than temporal traits to distinguish lookups of different bots (cf. § IV). Given the stronger randomness of $\mathcal{A}_\mathcal{S}$ and $\mathcal{A}_\mathcal{R}$ (cf. Figure 3), it is more likely for $\mathcal{M}_\mathcal{T}$ to differentiate bots based on domains they have queried. It is also noticed that $\mathcal{M}_\mathcal{B}$ and $\mathcal{M}_\mathcal{P}$ both outperform $\mathcal{M}_\mathcal{T}$, because of their lesser dependence on the temporal traits of individual lookups.

Length of observation window - In Figure 6 (b), we vary the length of observation window ($1 \sim 16$ epochs) and average the estimates over the number of epochs. As the window size grows, all the estimators are observed to enjoy improved accuracy. Intuitively, this may be explained by that the estimate variances of different estimators within individual epochs "cancel out" in an enlarged observation window. The improvement is especially evident for $\mathcal{A}_\mathcal{S}$ and $\mathcal{A}_\mathcal{R}$ (e.g., the maximum error of $\mathcal{M}_\mathcal{T}$ decreases from around .12 to below .03 in the case of $\mathcal{A}_\mathcal{R}$). One can attribute this to the stronger randomness of $\mathcal{A}_\mathcal{S}$ and $\mathcal{A}_\mathcal{R}$ in comparison with $\mathcal{A}_\mathcal{U}$ and $\mathcal{A}_\mathcal{P}$ (cf. Figure 3). The stronger randomness leads to the higher estimate variance, but also more improvement margin when the observation window is enlarged.

TTL of negative cache - Next we study the impact of caching mechanism of local DNS servers over the performance of BOTMETER. As shown in Figure 6 (c), $\mathcal{M}_\mathcal{T}$ suffers evident accuracy loss (e.g., with medium error jumping from about .05 to over .2 in the case of $\mathcal{A}_\mathcal{U}$) as the negative cache TTL is increased. Intuitively, longer cache TTLs imply more DNS lookups that are "masked" out, thereby significantly reducing the visibility of temporal dynamics of individual lookups, which $\mathcal{M}_\mathcal{T}$ and $\mathcal{M}_\mathcal{P}$ depend on. Moreover, between these two estimators (in the case of $\mathcal{A}_\mathcal{U}$), $\mathcal{M}_\mathcal{P}$ appears less sensitive to the TTL setting. This is because $\mathcal{M}_\mathcal{P}$ exploits temporal traits of observable lookups to assess the average bot activation rate and then estimates the number of activations hidden by the caching; in contrast, $\mathcal{M}_\mathcal{T}$ is unable to identify any bots that are completely masked by the caching. Meanwhile, because $\mathcal{M}_\mathcal{B}$ only relies on collective statistics of the bot population (i.e., distinct NXDs that have been queried by DGA-bots), its performance is not influenced by the caching mechanism.

Dynamics of bot activation rate - Recall that in the DGA simulator configured with varying activation rate, we control the dynamics of bot activation rate via the parameter σ. We let σ increase from 0.5 to 2.5 and observe its influence over the performance of different estimators. The results are shown in Figure 6 (d). Note that like the negative cache TTL, the dynamics of bot activation rate only affects temporal traits of DNS lookups, which $\mathcal{M}_\mathcal{B}$ is largely immune to. The comparison between $\mathcal{M}_\mathcal{T}$ and $\mathcal{M}_\mathcal{P}$ in the case of $\mathcal{A}_\mathcal{U}$ is rather interesting. While $\mathcal{M}_\mathcal{P}$ outperforms $\mathcal{M}_\mathcal{T}$ over the entire range of σ, the accuracy of $\mathcal{M}_\mathcal{P}$ decreases more significantly (e.g., with maximum error increased from around .07 to about .12) as σ grows. This is because $\mathcal{M}_\mathcal{P}$ is premised on the assumption of approximately stable bot activation rate, while the extremely fluctuating rate can cause over- or under-estimation, especially when the observation window is limited (e.g., one day).

Coverage of D3 algorithm - In reality, the D3 algorithm may detect only a subset of DGA-NXDs (cf. § II-B). It is thus critical to understand the impact of the detection window of D3 algorithm over the estimators. Here, we assume that the D3 algorithm randomly misses x percent of DGA-NXDs, where x increases from 10 to 50. As shown in Figure 6 (e), $\mathcal{M}_\mathcal{B}$ observes considerable accuracy losses as the detection window shrinks, given that it solely relies on the statistics of NXDs queried by the bot population. Meanwhile, the effectiveness of D3 algorithm has limited influence over $\mathcal{M}_\mathcal{P}$ and $\mathcal{M}_\mathcal{T}$, due to their dependency on temporal traits of individual NXD lookups; that is, the timestamps of a subset of DGA-domains may be sufficient for $\mathcal{M}_\mathcal{P}$ to estimate the average bot activation rate and for $\mathcal{M}_\mathcal{T}$ to distinguish lookups belonging to different DGA-bots.

B. Evaluation over Real Data

Experiment Setup

To generate real DNS traces for our studies, we collect DNS queries and responses from a local DNS server in a large enterprise network over a one-year timespan (from 05/01/2014 to 04/30/2015), which we refer to as the *raw* dataset. This local DNS server is responsible for resolving DNS queries for a sub-network comprising more than 22.5K IP addresses. During this time period, on average, there are 15,027 active IP addresses that issued DNS lookup queries to the server per day. We also collect the DNS lookups forwarded by this local server to the border DNS server, which we refer to as the *observable* dataset (i.e., in the sense that it is visible to BOTMETER). We assume a simplified data format similar to that of the synthetic data, wherein the raw dataset consists of a sequence of tuples of ⟨timestamp, client, domain⟩ while the observable dataset consists of a series of tuples of ⟨timestamp, domain⟩. Note that the forwarding server is omitted here given that there is only one local server. The granularity of timestamp is one second.

Further, from *DGArchive*[3], a lookup service for DGA malware, we extract a number of DGAs, which are believed to be active during this time period. Using the APIs provided by *DGArchive*, we query and collect all the domains generated by these DGAs during the given time period, which we refer to as the *pool* dataset. We match both the raw and observable datasets against the pool dataset. From the correlation results of the raw and pool datasets, via counting the number of distinct client IP addresses, we identify the population of

[3]https://dgarchive.caad.fkie.fraunhofer.de

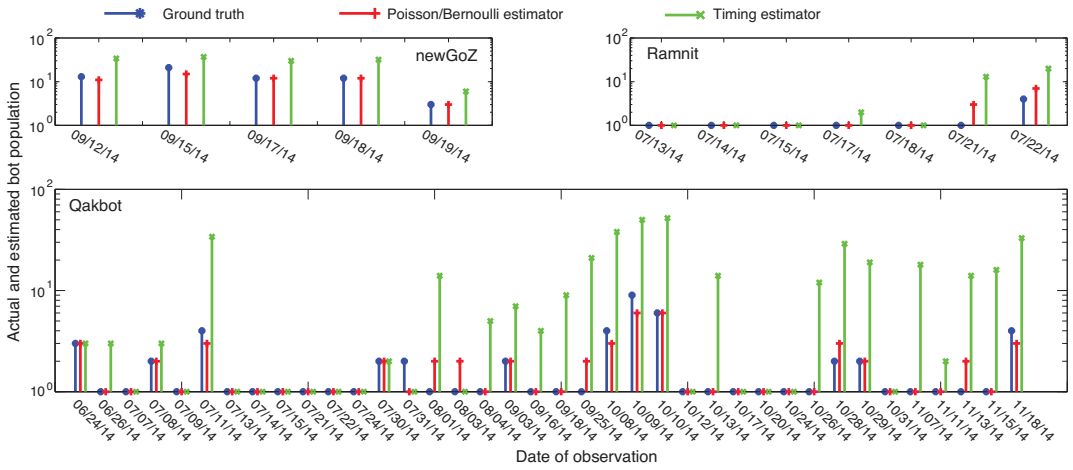

Figure 7: Daily populations of active bots and estimates made by $\mathcal{M}_{\mathcal{P}}$, $\mathcal{M}_{\mathcal{B}}$, and $\mathcal{M}_{\mathcal{T}}$ over 05/01/2014 $\sim$ 04/30/2015.

DGA	δ_i	$\mathcal{M}_{\mathcal{B}}$	$\mathcal{M}_{\mathcal{P}}$	$\mathcal{M}_{\mathcal{T}}$
newGoZ	1sec	$.116_{\pm .177}$		$1.545_{\pm .393}$
Ramnit	none		$.157_{\pm .276}$	$.884_{\pm 1.297}$
Qakbot	none		$.127_{\pm .237}$	$4.294_{\pm 5.118}$

Table II. Average estimation errors of varied estimators in assessing *newGoZ*, *Ramnit*, and *Qakbot* bot populations.

active bots corresponding to each DGA for each day, which serves as the ground truth in our experiments[4]. Meanwhile, we feed the correlation results of the observable and pool datasets as input to the analytical models of BOTMETER to evaluate its performance.

We fix the observation window length as one day and apply BOTMETER to estimating the daily population of active bots with respect to each of the three DGAs. Note that *newGoZ* belongs to $\mathcal{A}_{\mathcal{R}}$ while *Ramnit* and *Qakbot* are instances of $\mathcal{A}_{\mathcal{U}}$. Therefore, besides $\mathcal{M}_{\mathcal{T}}$, we apply $\mathcal{M}_{\mathcal{B}}$ to *newGoZ* and $\mathcal{M}_{\mathcal{P}}$ to *Ramnit* and *Qakbot*.

Experiment Results

We contrast the estimates of $\mathcal{M}_{\mathcal{T}}$, $\mathcal{M}_{\mathcal{B}}$, and $\mathcal{M}_{\mathcal{P}}$ (when applicable) against the actual daily populations of active bots (i.e., the ground truth). Figure 7 illustrates in detail the DGA-bot populations and the estimates for each day, with the accuracy of different estimators summarized in Table II.

It is observed that $\mathcal{M}_{\mathcal{P}}$ and $\mathcal{M}_{\mathcal{B}}$ perform highly accurate estimation across all three DGA families. For example, $\mathcal{M}_{\mathcal{P}}$ achieves average error below .12 in estimating *newGoZ* populations; in comparison, the accuracy of $\mathcal{M}_{\mathcal{T}}$ could be arbitrarily bad, with average error exceeding 4.2 in estimating *Ramnit* populations. This contrast is attributed to the reasons as below. First, as listed in Table II, all three DGA families have query intervals equal to or finer than the timestamp granularity in the data (i.e., one second), which substantially blurs $\mathcal{M}_{\mathcal{T}}$'s view to differentiate different bots based on their temporal periodicity. Second, $\mathcal{M}_{\mathcal{P}}$ and $\mathcal{M}_{\mathcal{B}}$ rely on the statistics of bots' collective behaviors (e.g., waiting time between two TTLs for $\mathcal{M}_{\mathcal{P}}$ and segment lengths for $\mathcal{M}_{\mathcal{B}}$), which is less subject to the timestamp granularity of individual lookups.

Also interestingly, $\mathcal{M}_{\mathcal{B}}$ performs slightly better than $\mathcal{M}_{\mathcal{P}}$. This may be explained by that $\mathcal{M}_{\mathcal{B}}$ does not depend on any temporal traits and the high coverage of D3 algorithm (based on reverse engineering of DGAs) leads to reliable statistics of DGA-generating NXDs, which is crucial for $\mathcal{M}_{\mathcal{B}}$ to make effective estimation. These observations are consistent with our findings in § V-A.

[4]Wireless devices are assigned IP addresses dynamically via DHCP, but their IP-MAC bindings are valid during a one-day time window.

VI. RELATED WORK

As DNS systems become one primary target for Internet miscreants (e.g., fast-fluxing, domain-fluxing, DNS tunneling), the security community has conducted intensive research on the misuse and abuse of DNS systems.

Most existing research on DGAs has focused on detecting DGA-domains and identifying compromised machines to hinder C2 traffic. Yadav et al. [25] proposed to discover DGA-domains by leveraging their characteristic distributions of alphanumeric characters and bigrams. Schiavoni et al. [19] leveraged linguistic features to detect DGA-domains. Antonakakis et al. [7] presented a system that identifies botnet groups running the same DGA by clustering NXDs based on syntactic features. Thomas and Mohaisen [23] analyzed NXD queries at *com*, *net*, *tv*, and *cc* top-level domain (TLD) authoritative name servers to identify communities of malicious domains sharing similar lookup patterns. Sharifnya and Abadi [21] proposed a reputation-based DGA detection method by calculating reputation scores of hosts based on suspicious DGA-related domain queries and abnormal amounts of failed queries. This work complements these studies by leveraging identified DGA-domains as input to accurately assess the threat landscape.

In contrast of the bulk of work on detecting DGA-domains in DNS traffic, the studies on assessing DGA-bot populations are fairly limited. The first approach is capturing C2 channels and backtracing to infected machines. Stone-Gross et al. [22] monitored *Torpig* botnet operations by hijacking its C2 communications and measured the botnet size using unique identifiers of compromised hosts. Nelms et al. [15] proposed to learn control protocol templates from confirmed C2 communications and adapt such templates to discover previously unknown C2 channels. The second approach is vetting DNS behavior of individual machines and identifying positive matches with confirmed DGA-domains. This work is among the first non-intrusive solutions capable of accurately assessing DGA-bot populations while incurring minimal operational costs.

VII. CONCLUSION

In this paper, we present BOTMETER, a novel tool to assess populations of DGA-embedded bots distributed over large networks solely using DNS traffic observable at upper-level DNS servers. We established a new taxonomy of DGAs based on their inherent DNS querying patterns. This allows us to develop a rich library of rigorous analytical models to capture the DNS dynamics of an array of DGA-embedded botnets. Through extensive empirical studies using both synthetic and real DNS traces collected from a large enterprise network over a one-year timespan, we demonstrated that BOTMETER is able to accurately estimate the severity of botnet infection in a non-intrusive and noise-tolerant manner, thereby helping analysts quickly navigate the threat landscape of their networks and prioritize the remediation efforts. This work also opens up several directions for future investigations: 1) combining temporal and semantic traits of DNS lookups to develop more effective bot population estimators; 2) complementing BOTMETER with visual analytical components to fully exploit its potential; and 3) (from attacker's perspective) designing advanced DGA models that evade effective population estimation.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their insightful comments and suggestions to improve the quality of this paper. They are also grateful for the fruitful discussion with Josyula R. Rao, Douglas Schales and Dhilung Kirat. This work was partly supported by the National Science and Technology Support Program of China under grant No. 2014BAH24F01.

REFERENCES

[1] Malware Authors Expand Use of Domain Generation Algorithms to Evade Detection. http://www.pcworld.com/, 2012.
[2] Unveiling The Latest Variant of Pushdo. http://www.damballa.com/, 2013.
[3] GameOver Zeus Mutates, Launches Attacks. http://blog.malcovery.com/, 2014.
[4] Necurs: The Malware that Breaks Your Security. http://www.trendmicro.com/, 2014.
[5] Ranbyus Banking Trojan, Cousin of Zbot. http://www.mysonicwall.com/, 2014.
[6] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a Dynamic Reputation System for DNS. In *SEC*, 2010.
[7] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon. From Throw-away Traffic to Bots: Detecting the Rise of DGA-based Malware. In *SEC*, 2012.
[8] T. Barabosch, A. Wichmann, F. Leder, and E. Gerhards-Padilla1. Automatic Extraction of Domain Name Generation Algorithms from Current Malware. In *NIAS*, 2012.
[9] F. Giroire, J. Chandrashekar, N. Taft, E. Schooler, and D. Papagiannaki. Exploiting Temporal Persistence to Detect Covert Botnet Channels. In *RAID*, 2009.
[10] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In *NDSS*, 2008.
[11] IETF. Common DNS Operational and Configuration Errors. http://tools.ietf.org/html/rfc1912, 1996.
[12] F. Leder and T. Werner. Know Your Enemy: Containing Conficker, To Tame a Malware. http://www.honeynet.org/, 2009.
[13] Y. Nadji, M. Antonakakis, R. Perdisci, D. Dagon, and W. Lee. Beheading Hydras: Performing Effective Botnet Takedowns. In *CCS*, 2013.
[14] A. Nappa, A. Fattori, M. Balduzzi, M. Dell'Amico, and L. Cavallaro. Take a Deep Breath: A Stealthy, Resilient and Cost-effective Botnet Using Skype. In *DIMVA*, 2010.
[15] T. Nelms, R. Perdisci, and M. Ahamad. Execscent: Mining for new c&c domains in live networks with adaptive control protocol templates. In *SEC*, 2013.
[16] P. Porras, H. Saïdi, and V. Yegneswaran. A Foray into Conficker's Logic and Rendezvous Points. In *LEET*, 2009.
[17] S. M. Ross. *Stochastic Processes (Wiley Series in Probability and Statistics)*. Wiley, 2 edition, 1995.
[18] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C. J. Dietrich, and H. Bos. Sok: P2pwned - modeling and evaluating the resilience of peer-to-peer botnets. In *S&P*, 2013.
[19] S. Schiavoni, F. Maggi, L. Cavallaro, and S. Zanero. Phoenix: DGA-Based Botnet Tracking and Intelligence. In *DIMVA*, 2014.
[20] Sergei Shevochenko. Domain Name Generator for Murofet. http://blog.threatexpert.com/, 2010.
[21] R. Sharifnya and M. Abadi. A novel reputation system to detect DGA-based botnets. In *ICCKE*, 2013.
[22] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your Botnet is My Botnet: Analysis of a Botnet Takeover. In *CCS*, 2009.
[23] M. Thomas and A. Mohaisen. Kindred domains: detecting and clustering botnet domains using DNS traffic. In *WWW*, 2014.
[24] T. Wang, X. Hu, J. Jang, S. Ji, M. Stoecklin, and T. Taylor. Technical Report: A Nimble Navigator for DGA-Botnet Landscape. http://x-machine.github.io/reports/BotMeterTR.pdf, 2015.
[25] S. Yadav, A. K. K. Reddy, A. N. Reddy, and S. Ranjan. Detecting Algorithmically Generated Malicious Domain Names. In *SIGCOMM*, 2010.