BOTMETER: A Nimble Navigator for DGA-Bot Landscape of Large Networks

***Abstract*—Recent years have witnessed a rampant use of domain generation algorithms (DGAs) in major botnet crimewares (e.g., Conficker, Bobax, Zeus), which significantly strengthens a botnet's capability to evade detection and/or takedown. Despite a plethora of existing studies on detecting DGA-generating domains in DNS traffic, remediating such threats still relies on vetting DNS behavior of each individual machine. However, in large networks featuring complicated DNS infrastructures, we often lack either the capability or the resource to exhaustively investigate every part of the networks to identify infected machines in a timely manner. It is of great interest to first navigate the threat landscape of the networks and to prioritize the remediation efforts. In this paper, we present BotMeter, a novel navigation tool that accurately assesses populations of DGA-embedded bots distributed over large-scale 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 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 analytical models and the efficacy of BotMeter.**

I. INTRODUCTION

A botnet is a large number of malware-infected machines (*bots*) that can be remotely controlled by their operator (*botmaster*) through *command-and-control* (C2) channels. Being 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 [27].

Over the recent years, a rampant use of *domain-fluxing* has been witnessed in major botnet crimewares (e.g., *Torpig*, *Conficker*, *BankPatch*, *Bobax*, *Gameover Zeus*) [3], [1]. With domain-fluxing, bots employ *domain generation algorithms* (DGAs) to dynamically compute a list of pseudo-random domains. This list is generated independently by each bot and is often re-generated periodically. The bot attempts to contact domains on the list in a certain order until one succeeds (i.e., the domain resolves to an IP address and the corresponding server provides a valid response according to the botnet's protocol) or aborts after a certain number of trials. If a domain is found blocked or invalid (e.g., it is suspended by an authoritative takedown operation), the bot simply rolls over to the next one on the list. The botmaster needs to register only a few domains on the list to serve as C2 servers. As an example, each *Conficker.C* worm [21] 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 a DGA rather than a list of hardcoded domains significantly strengthens botnets' capability to evade detection and takedown. Foremost, the enormous number of potential rendezvous points makes it extremely difficult to preemptively eliminate C2 channels (e.g., using blacklisting to restrict outbound communication or pre-registering domains that bots will contact in future). 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 automatically reject any commands not signed by their botmasters.

A plethora of techniques have since been proposed to detect DGA-generating domains[1] in DNS lookups, including analyzing algorithmic patterns of domains [36], reverse-engineering malware instances [25], [13], [30], [21], clustering non-existent domains (NXDs) in DNS lookups [12], [34], and even directly capturing C2 traffic [17], [22], [24]. Nevertheless, the studies on how to leverage detected DGA-domains to remediate practical botnet threats in large-scale networks are still fairly limited.

At a first glance, this may seem a rather simple problem with trivial solutions, including: a) capturing C2 channels and tracing back to infected machines [33] and b) detecting DGA-domains and vetting DNS behavior of each individual machine to identify positive matches. Unfortunately, directly applying either naïve solution faces major challenges in practice. First, as aforementioned, 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 may quickly roll over to relocated C2 domains. Both factors result in the extremely low coverage of solution a). Second, in large networks, hundreds of thousands of machines are often geographically distributed and probably managed by different entities. Applying solution b) exhaustively to 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 therefore of great interest to first quickly navigate the threat landscape of the networks and to prioritize the remediation efforts.

We concretize this objective as the problem of assessing populations of DGA-bots distributed over large networks. More

[1]Below we refer to "DGA-generating domain" as "DGA-domain" and "DGA-embedded bot" as "DGA-bot" for short when the context is clear.

specifically, we aim at lightweight, non-intrusive solutions only relying on (i) confirmed domains generated by the target DGA and (ii) aggregated DNS lookups observed at certain vantage points. We remark that this assumption is both realistic and minimal. Regarding (i), as aforementioned, a plethora of off-the-shelf techniques (e.g., [36], [25], [13], [30], [21], [12]) are available to effectively identify DGA-domains. Regarding (ii), as illustrated in Figure 1, the DNS infrastructures of large networks are typically hierarchical with *caching*-and-*forwarding* mechanisms enabled [10]. That is, DNS servers at upper-levels (e.g., *border* DNS servers) only observe aggregated DNS traffic (i.e., without client information) forwarded by *local* servers at lower-levels; meanwhile, local servers only forward lookups missed in their caches. Due to management and operational costs, DNS traffic is often visible and collectable only at certain high-level, vantage points (e.g., border servers).

Embracing the prevalent yet challenging scenario above, we present BotMeter, a novel navigation tool that accurately estimates DGA-bot population distributions by exploiting the temporal and semantic patterns inherent in the DNS dynamics of DGAs. To build BotMeter, we first establish a new taxonomy of DGAs based upon their characteristic domain fluxing behaviors. This allows us to develop a rich library of rigorous analytical models for BotMeter to capture the DNS dynamics of a wide range of DGAs. Compared with alternative solutions, BotMeter distinguishes itself with a set of desiderata for our target context: (i) it works on highly aggregated DNS traffic; (ii) it takes account of the impact of caching-and-forwarding mechanisms enabled in DNS infrastructures; (iii) it is backed by a rich library of estimation models designed for varied DGA families; and (iv) it is resilient against noisy and missing observations.

To our best knowledge, BotMeter is the first non-intrusive solution capable of accurately assessing DGA-bot population distributions while incurring minimal operational costs. Our contributions can be summarized as follows.

- We highlight and articulate the problem of estimating DGA-bot population distributions in large networks solely using aggregated DNS lookups observed at a few upper-level vantage points.
- We establish a new taxonomy of DGAs 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.
- Most importantly, we implement BotMeter, a novel navigation tool that realizes these ideas. We extensively evaluate its empirical performance using both synthetic data and real DNS traces collected from a large enterprise network over a one-year timespan. The promising results indicate that BotMeter is able to accurately assess 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.

The remainder of the paper proceeds as follows. § II introduces fundamental concepts and formalizes the problem of assessing DGA-bot population distributions; § III summarizes a set of representative DGA families and establishes a new taxonomy of DGAs based on inherent patterns of their domain-fluxing behaviors; § IV describes in detail the design and implementation of BotMeter, followed by its empirical evaluation in § V; § VI surveys relevant literature; the paper is concluded in § VII.

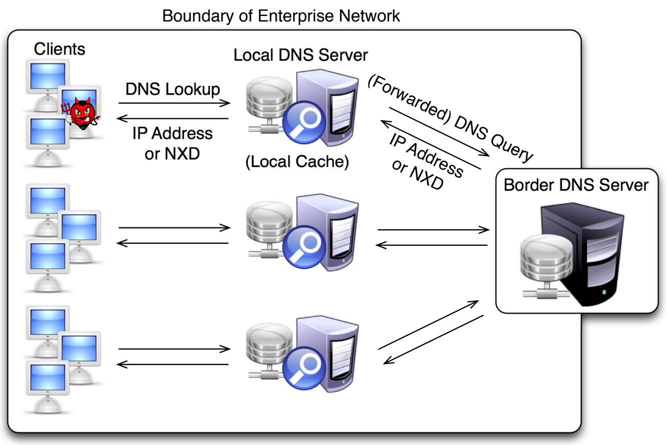

Figure 1: A schematic hierarchical DNS architecture in a large-scale network.

II. Background

In this section, we first introduce a set of fundamental concepts and assumptions used throughout the paper, then formalize the problem of estimating DGA-bot population distributions, and finally motivate the design of BotMeter.

A. Domain Generation Algorithm

From the earliest documented DGA-bot *Srizbi* [32] to the most recently discovered *Ranbyus* [6], an alarming number of botnet crimewares now employ DGAs as their primary coordination strategy for bots to receive updates and commands from their botmasters. Compared with alternatives (e.g., P2P C2 infrastructure), this strategy provides a tremendous level of evasiveness and agility because (i) the enormous number of potential rendezvous points in conjunction of their dynamic nature make it essentially infeasible to take any preemptive actions and (ii) DGA-bots can automatically roll over to relocated C2 servers via dynamically generated domains when current C2 domains or IPs are taken down. Well-known instances of DGA-bots include *Torpig* [33], *Conficker* [21], *Murofet* [30], *Kraken* [28], *Bobax* [5], *Necurs* [4], and *Gameover Zeus* [2].

B. Existing Taxonomies of DGA

While each DGA appears as a unique "snowflake", from an algorithmic perspective, DGAs are essentially a class of algorithms that take as input a *seed* and produce a sequence of pseudo-random domains. Therefore one can classify existing DGAs based on properties of their seeds [13].

- The first class is *time-independent* and *deterministic* DGA, which produce the same set of domains every time they are executed due to using static, hardcoded seeds (e.g., *Kraken*).
- The second class is *time-dependent* and *deterministic* DGA, wherein the seeds are updated on a regular basis. For example, *Conficker* uses the current date as the seed.

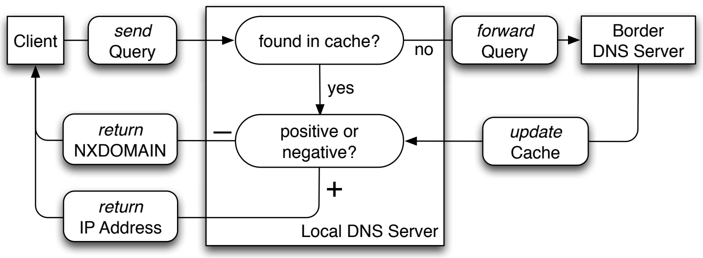

Figure 2: Life cycle of a DNS lookup query

Yet, it is possible to pre-compute the domain names because the seeds can be anticipated.

- The third class is *time-dependent* and *non-deterministic* DGA. Here the seeds cannot be anticipated; therefore pre-computation is impossible. For example, *Torpig* [33] and *Necurs* [4] use popular trending topics of *Twitter* and foreign exchange rates as seeds, respectively.
- The last class is *time-independent* and *non-deterministic* DGA. Botnets employing this class of DGAs have not been spotted in the wild yet.

Because of its much higher agility and evasiveness, time-dependent DGAs (TD-DGAs) have become the de facto choice employed in major botnet crimewares [1]. Therefore we focus our following discussion on TD-DGAs.

One may further classify TD-DGAs based on other notable features, including: (i) whether it introduces a magic number (i.e., *salt*) to the algorithm for added complexity or campaign-specific control, (ii) whether it uses a dictionary of words to generate domain names (e.g., *Rovnix* uses words from the *US Declaration of Independence* as input to its DGA [7]), and (iii) whether it relies on DGA as the primary coordination strategy or as a fallback mechanism.

C. Life Cycle of A DNS Lookup Query

As one backbone component of Internet, the *Domain Name System* (DNS) is a naming system that resolves domain names to their corresponding IP addresses. For example, when a client looks up "www.example.com", the DNS server recursively or iteratively discovers and responds with "www.example.com $\rightarrow$ 192.0.43.10" (i.e., an *A-type* resource record). For non-existent, invalid domain names, the server will return "NXDomain" (or "NXD" for short).

As shown in Figure 1, the DNS infrastructure of a large-scale network is often organized hierarchically. A DNS lookup query issued by a client is first processed by its corresponding local DNS server. For efficiency purpose, a caching-and-forwarding mechanism is often enabled. Concretely, the local server stores previously queried domains and their results (i.e., either A-type resource records or NXDs), with each record valid for a certain pre-defined time period (TTL). Given an incoming DNS lookup query, the server will first attempt to answer it using its cached results. Only when a miss occurs, the query will be forwarded to an upper-level DNS server (e.g., border DNS servers which connect to ISP's DNS servers); once the result has been returned, the cache at the local server will be updated accordingly. The life cycle of a DNS lookup query is illustrated in Figure 2.

D. Visibility

We make the following assumptions regarding the observations we have regarding DNS traffic generated by DGA-bots.

Vantage Point - In large-scale networks, due to management and operational costs, it is not uncommon that DNS traffic is visible and collectable only at certain vantage points. In the following we assume that the vantage points are set at the border DNS servers, as shown in Figure 1, which collect DNS lookup queries forwarded by local DNS servers at lower levels. In particular, we assume that the forwarded lookups come in as a stream of tuples of the form:

$$\langle \text{timestamp } t, \text{ forwarding server } s, \text{ domain } d \rangle$$

i.e., each tuple represents a lookup for domain name d issued at time t and forwarded by server s. Via analyzing the DNS traffic forwarded by each local DNS server, we are interested in estimating the DGA-bot population in the network behind this local server.

Positive and Negative Caching - As aforementioned, if the cache of a local DNS server is enabled, a DNS lookup query 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 [19].

Detection Window - Most TD-DGAs use current dates as parts of their seeds and are executed on a daily basis. We may therefore use a general 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 (e.g., via reverse engineering the bot malware, particularly its DGA) is able to detect all domains in the pool. However, in reality, due to all sorts of constraints (e.g., the botnet malware may be updated fairly frequently or it may be obfuscated and only decrypted and executed by external triggers such as time), the D3 algorithm is able to detect only a part of the pool, which is referred to as its *detection window*. Further, a small number of domains in the pool may coincide with valid domains, which we refer to as *collision cases*.

E. Problem Formulation

We are now ready to formulate the problem of *assessing DGA-bot population distributions in large networks*.

Given the detection window for domains generated by a target DGA, via analyzing DNS lookups observed at a border DNS server, we intend to accurately estimate the population of bots that employ this particular DGA within the network behind each local DNS server.

F. BotMeter in A Nutshell

We present BotMeter, a novel navigation tool built for this task. Figure 3 illustrates its high-level architecture. Tapped at border DNS servers (①), BotMeter analyzes DNS lookups forwarded by each local DNS server. BotMeter allows analysts to specify algorithmic patterns (or plain lists) of domains

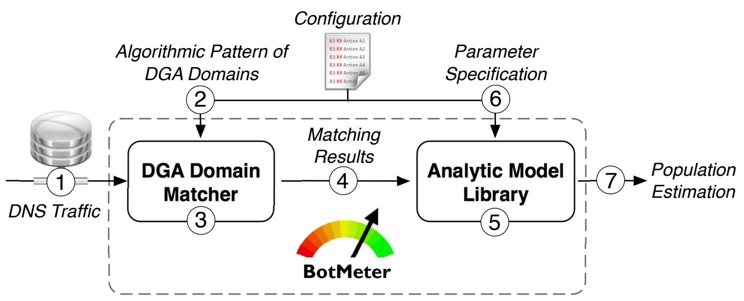

Figure 3: Anatomy of BotMeter

generated by targeted DGAs (②) and matches incoming DNS traffic against such patterns (③). The matched results (including timestamps and domains of matched DNS lookups) are fed as input to the next phase (④). At the heart of BotMeter is a rich library of analytical models designed for a broad range of DGA families (⑤). Through the configuration interface (⑥), analysts select the most appropriate analytical model as well as specify other key parameters (e.g., detection window size), which are then applied over the matched results to perform DGA-bot population estimation.

To build BotMeter, the existing taxonomies of DGAs (cf. § II-B) are insufficient. We need a new taxonomy that captures DGAs-characteristic DNS dynamics. In § III, we establish such a taxonomy. This allows us to develop the library of analytical models for BotMeter, which we elaborate in § IV.

III. A New DGA Taxonomy

In this section, we establish a new taxonomy to characterize domain fluxing behaviors of TD-DGAs. We start with a simple model of TD-DGA, branch from it, and instantiate a broad range of DGA families.

A. A Strawman Model

Everyday from a *query pool* comprising $(\theta_\emptyset + \theta_\exists)$ pseudo-random domains[2], the botmaster selects a subset of $\theta_\exists$ domains (typically only 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 (i.e., with Internet access) 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 it hits one valid domain or aborts otherwise.

Following this model, we can 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. Below we introduce a collection of query pool and barrel models, as well as representative DGA families that follow each model.

B. Query Pool Model

Drain-and-Replenish Pool - This is perhaps the simplest query pool model, wherein the entire pool of pseudo-random domains is replaced on a regular basis (e.g., daily). A majority of known TD-DGAs all follow this model (e.g., *Murofet*, *Srizbi*, *Conficker*, *Gameover Zeus*, *Necurs*).

Sliding-Window Pool - A class of recent TD-DGAs (e.g., *Ranbyus*, a banking Trojan family [6] and *PushDo*, a downloader malware [8]) adopt a sliding-window pool model. Under this 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.

As concrete examples, *Ranbyus* generates a fresh set of 40 domains everyday and also maintains the domains generated in the past 30 days, i.e., a query pool of 1,240 domains; *PushDo* maintains a window of -30 to +15 days of 30 domains per day, which gives it a query pool of 1,380 domains.

Multiple Mixture Pool - To further increase the resilience against takedown, a few recent TD-DGAs introduce the multiple mixture pool model. Under this 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*, a worm that spreads via *Skype* [23], employs two identical DGA instances, one generating a pool of 200 useful domains and the other generating a pool of 16K noisy domains.

C. Query Barrel Models

Uniform Barrel - This is the simplest as well as the most popular query barrel model, adopted by a majority of TD-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* (a variant of *Conficker* [21]), this class of DGA families 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 successful rate to establish bot-to-botmaster connections.

RandomCut Barrel - This class of TD-DGA families predefines a global sequential order over domains in the query pool. Then each active bot randomly picks a sequence number as the starting point and considers the next θ_q domains (in a modular arithmetic sense) as its query barrel. For example, *newGoZ*, a recent variant of *Gameover Zeus*, follows the randomcut barrel model. Each *newGoZ* bot constructs its query barrel by randomly selecting 500 consecutive domains from a sequence of 10K domains [2].

Interestingly, the randomcut model strikes a balance between the uniform and sampling models in the sense that it features more randomness than the former due to using random

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

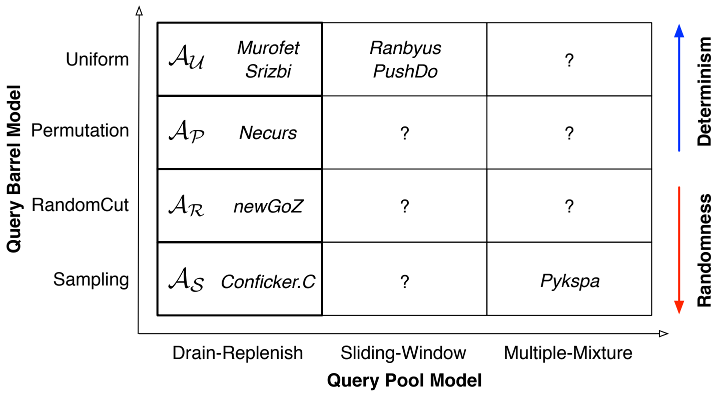

Figure 4: A taxonomy of DGAs and representative DGA families ("?" indicates that DGAs following the particular model have not been spotted in the wild).

starting points but more determinism than the latter due to enforcing a global sequential order.

Permutation Barrel - For TD-DGA families belonging to this class, the query pool is re-generated regularly, meanwhile each bot attempts to look up all 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*, a backdoor Trojan [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.

D. Universe of DGAs

Equipped with the query pool and barrel models above as reference, we are able to establish a new taxonomy of DGAs. As illustrated in Figure 4, 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 pool-barrel-model combination. It is clear that compared with existing DGA taxonomies (cf. § II-B), this new taxonomy well captures the characteristic DNS dynamics of DGAs. As we will show shortly (cf. § IV), 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. For ease of presentation, 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 detail bot population estimation models designed for DGAs in these categories.

IV. Library of Analytical Models

At the heart of BotMeter lies the analytical model library which comprises a collection of estimation models applicable for a broad range of DGA families. In this section we detail the design and implementation of these models. All the theoretical proofs in this section are deferred to the Appendix.

A. Preliminaries

Tapped at border DNS servers, BotMeter observes DNS lookups forwarded by each local DNS server, among which it identifies lookups that match confirmed domains generated by the target DGA. In particular, we focus on DGA-generating NXDs (DGA-NXDs), which constitute a dominant majority of DGA-domains. Without loss of generality, we assume that the matched results comprise a sequence of tuples $\langle$timestamp t, domain $d\rangle$, each representing a lookup for domain d issued at time t. Then, corresponding to the target DGA, BotMeter attempts to assess the population of active bots behind each local server. Below we elaborate the estimation models for DGAs in each category as defined in Figure 4.

To ease the presentation, we first introduce a set of fundamental concepts. Given a DGA-bot, each round of execution of its DGA is called an *activation*; the timespan of an activation is called its *duration*; the time gap between two consecutive DGA-triggering DNS lookups is referred to as a *query interval*; the waiting time between the bot's two consecutive activations is an *epoch*; finally, the timespan for which a DNS server caches a DNS response is determined by the value of *time to live* (TTL). Below 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 at the granularity of a day while the setting of $\delta_\mathtt{l}$ varies from minutes to hours for NXD [19]. For most DGAs, there is minimal interval (e.g., $\delta_\mathtt{i} = 500ms$) between two consecutive DGA-triggering DNS lookups. Thus we assume that $\delta_\mathtt{d}$ is negligible compared with $\delta_\mathtt{e}$ or $\delta_\mathtt{l}$ (i.e., $\delta_\mathtt{d} \ll \delta_\mathtt{e}, \delta_\mathtt{l}$). For simplicity of discussion, we also assume that (i) $\delta_\mathtt{e}$ = one day and (ii) $\delta_\mathtt{e}$ is a multiple of $\delta_\mathtt{l}$.

Following the DGA model in § III-A, the query pool comprises $(\theta_\exists + \theta_\emptyset)$ domains, among which $\theta_\exists$ are registered as C2 servers with the remaining $\theta_\emptyset$ as invalid 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 four estimation models, namely, *Timing* estimator ($\mathcal{M}_\mathcal{T}$), *Poisson* estimator ($\mathcal{M}_\mathcal{P}$), *Combinatorial* estimator ($\mathcal{M}_\mathcal{C}$), and *Bernoulli* estimator ($\mathcal{M}_\mathcal{B}$). The roadmap of their use for different DGA categories is summarized in Table I.

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

We start with the Timing estimator (denoted by $\mathcal{M}_\mathcal{T}$), the foremost simplest estimation model. The intuition of $\mathcal{M}_\mathcal{T}$ is to differentiate DNS lookups belonging to different bots based on their temporal and semantic traits.

More specifically, $\mathcal{M}_\mathcal{T}$ leverages three types of traits to achieve this goal. First, during a time window of one epoch, two lookups regarding a same NXD are likely to be produced by different bots. Second, two lookups with time separation longer than the maximum possible duration (i.e., $\theta_\mathtt{q} \cdot \delta_\mathtt{i}$) are attributed to different bots. Finally, it is noted that the query interval of a given DGA is typically fixed (e.g., $1sec$ for *newGoZ* and $500ms$ for *Ramnit*), i.e., each activation produces a train

Estimator \ Model	Uniform $\mathcal{A}_\mathcal{U}$	Sampling $\mathcal{A}_\mathcal{S}$	RandomCut $\mathcal{A}_\mathcal{R}$	Permutation $\mathcal{A}_\mathcal{P}$
Timing $\mathcal{M}_\mathcal{T}$	$\surd$	$\surd$	$\surd$	$\surd$
Poisson $\mathcal{M}_\mathcal{P}$	$\surd$			
Combinatorial $\mathcal{M}_\mathcal{C}$		$\surd$		$\surd$
Bernoulli $\mathcal{M}_\mathcal{B}$			$\surd$	

Table I. Roadmap of using bot population estimation models in BOTMETER.

of lookups with regular temporal intervals [16]. Therefore, 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 $\delta_\mathtt{i}$ = 500*ms*, two lookups with temporal separation of 750*ms* are likely to be issued by different bots.

Putting these heuristic rules together, Algorithm 1 sketches the main flow of $\mathcal{M}_\mathcal{T}$. It maintains a list $\mathcal{L}$, wherein each entry corresponds to one distinct bot (line 1). It applies the aforementioned three heuristic rules to each tuple in the sequence of DNS lookups (line 5-7). If this 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 (line 8). Otherwise, a new entry representing an unseen bot will be created (line 9). Eventually the number of entries in $\mathcal{L}$ indicates the number of bots $\mathcal{M}_\mathcal{T}$ has discovered (line 10).

It is worth pointing out that $\mathcal{M}_\mathcal{T}$ solely relies on the temporal dynamics of individual DNS lookups, thereby applicable to all the DGA models defined in Figure 4.

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 N
    // classification of lookups to distinct bots
 1  $\mathcal{L} \leftarrow \emptyset$;
 2  for each tuple $\langle$timestamp $t$, domain $d\rangle \in \mathcal{S}$ do
 3      if $\mathcal{L}$ is empty then add $(t, \{d\})$ to $\mathcal{L}$ and continue;
 4      for each entry (timestamp $t^*$, domains $\mathcal{D}$) $\in \mathcal{L}$ do
            // heuristic #1
 5          if $d$ appears in $\mathcal{D}$ then continue;
            // heuristic #2
 6          if $t^* + \delta_\mathtt{i} \cdot \theta_\mathtt{q} \leq t$ then continue;
            // heuristic #3
 7          if $(t - t^*) \mod \delta_\mathtt{i} \neq 0$ then continue;
 8          add $d$ to $\mathcal{D}$ and break;
 9      if $\langle t, d\rangle$ is not "absorbed" then add entry $(t, \{d\})$ to $\mathcal{L}$;
10  return number of entries in $\mathcal{L}$ as N;
```

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

While generally applicable to all DGA models, $\mathcal{M}_\mathcal{T}$ faces a major challenge in estimating $\mathcal{A}_\mathcal{U}$ bot populations for the reasons below. According to its definition (cf. §III-C), DGA-bots employing $\mathcal{A}_\mathcal{U}$ generate identical query barrels during each epoch; because of the caching effect at local DNS servers, if multiple bots are activated within a same time window of length TTL, only the first one will be visible, as illustrated in Figure 5. It is therefore infeasible to estimate bot population directly from observable DNS lookups. Clearly, the key challenge here is to infer the number of bots whose activations have been "masked" by the caching mechanism.

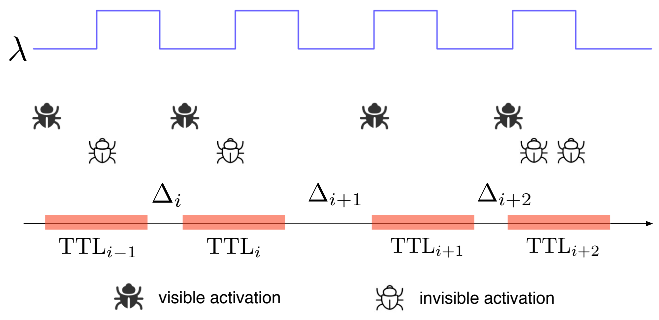

Figure 5: Illustration of (in)-visible activations of $\mathcal{A}_\mathcal{U}$, TTL of negative caching, and Poisson estimator $\mathcal{M}_\mathcal{P}$.

To address this challenge, we assume that the activations of DGA-bots follow a Poisson process, a stochastic process widely applied to model punctual phenomena wherein events occur independently from each other [26]. 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.

Concretely, denote by Δ_i the temporal gap between the end of $(i-1)$-th TTL and the start of i-th TTL, as illustrated in Figure 5. Base on the aforementioned assumption, we consider $\{\Delta_i\}$ as an indication of the activation rate λ. Intuitively, the larger population of bots leads to the shorter "waiting time" before the next bot to be activated. Considering n TTLs with temporal gaps $\{\Delta_i\}_{i=1}^{n}$[3], the expectation of average activation rate λ can be derived as:

$$\mathbb{E}(\lambda) = \frac{n}{\sum_{i=1}^{n} \Delta_i}$$

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

$$\begin{aligned} \mathbb{E}(\mathtt{N}) &= \mathbb{E}(\lambda) \sum_{i=1}^{n} (\Delta_i + \delta_\mathtt{I}) \\ &= n + \frac{n^2 \delta_\mathtt{I}}{\sum_{i=1}^{n} \Delta_i} \end{aligned} \quad (1)$$

We note that $\mathcal{M}_\mathcal{P}$ does not assume a *constant* activation rate; rather, λ captures the average activation rate during the observation window.

D. Combinatorial Estimator ($\mathcal{M}_\mathcal{C}$)

Both $\mathcal{M}_\mathcal{T}$ and $\mathcal{M}_\mathcal{P}$ rely on temporal traits of DNS lookups generated by DGA-bots to assess bot populations. In realistic

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

settings, due to all sorts of interference issues (e.g., network and server queuing delay), coarse-grained and erroneous timestamps are not uncommon, requiring designing estimators that depend on more reliable traits.

Below we introduce two estimators which leverage semantic patterns of DGA-bots' collective DNS behaviors, independent of temporal dynamics of individual lookups. Next we first present the Combinatorial estimator ($\mathcal{M}_{\mathcal{C}}$), which takes as input the total number of distinct NXDs queried from the query pool and estimates the population of responsible bots. $\mathcal{M}_{\mathcal{C}}$ is applicable to $\mathcal{A}_{\mathcal{S}}$ and $\mathcal{A}_{\mathcal{P}}$, for bots in either model generate distinct yet possibly overlapping query barrels.

We start with modeling the number q of NXDs an individual bot attempts to resolve during each activation. Following the sampling barrel model (cf. § III-C), we consider q a random variable obeying the distribution below.

$$\mathsf{Pr}(q = i) = \begin{cases} \frac{\theta_{\exists}}{i+1} \frac{\binom{\theta_{\emptyset}}{i}}{\binom{\theta_{\exists}+\theta_{\emptyset}}{i+1}} & 0 \le i < \theta_{\mathtt{q}} \quad \text{case (i)} \\ \frac{\binom{\theta_{\emptyset}}{\theta_{\mathtt{q}}}}{\binom{\theta_{\exists}+\theta_{\emptyset}}{\theta_{\mathtt{q}}}} & i = \theta_{\mathtt{q}} \quad \text{case (ii)} \\ 0 & \text{otherwise} \quad \text{case (iii)} \end{cases} \tag{2}$$

where $\binom{x}{y}$ represents the binomial coefficient of "x choose y". Here the first two cases correspond to that the bot hits a valid domain and that it aborts after trying $\theta_{\mathtt{q}}$ lookups, respectively; in both cases the first i queries are all chosen from NXDs of the query pool.

Recall that $\mathcal{A}_{\mathcal{S}}$ bots generate their query barrels by independently and randomly sampling subsets from the query pool (cf. § III-C). Considering DGA-NXDs as "coupons" and each bot as a "drawing", we can formulate this as a variant of the *coupon collector's* problem: from a pool of $\theta_{\emptyset}$ distinct coupons, during each trial, one randomly draws a subset (with replacement) of coupons. Given the number of distinct coupons one has collected, we attempt to infer the expected number of drawings needed. We note that this formulation differs from the classical coupon collector's problem in that (i) each trial is a batch drawing while (ii) the batch size follows a distribution as defined in Eqn (2). Also note that the DNS caching mechanism has no effect over the number of distinct coupons collected.

Assume that from $\theta_{\emptyset}$ NXDs of the query pool, a sequence of subsets $w_1, w_2, \dots$ of size $q_1, q_2, \dots$ are drawn (with replacement) and that k distinct NXDs are observed in the aggregated DNS lookups. Then, the bot population $\mathtt{N}$ corresponds to the number of subsets necessary to collect at least k distinct NXDs. We have the following proposition.

Lemma 1. *For given* $k \in \mathbb{Z}^+$, *we have:* $\mathsf{Pr}(\mathtt{N} = n) =$

$$\sum_{i=0}^{k-1} (-1)^{k+i} \binom{\theta_{\emptyset}}{i} \binom{\theta_{\emptyset} - i - 1}{\theta_{\emptyset} - k} \frac{\binom{i}{q_n} - \binom{\theta_{\emptyset}}{q_n}}{\binom{\theta_{\emptyset}}{q_n}} \prod_{j=1}^{n-1} \frac{\binom{i}{q_j}}{\binom{\theta_{\emptyset}}{q_j}} \tag{3}$$

To calculate the expectation of $\mathtt{N}$, we differentiate the cases of $\mathcal{A}_{\mathcal{S}}$ and $\mathcal{A}_{\mathcal{P}}$.

$\mathcal{M}_{\mathcal{C}}$ for $\mathcal{A}_{\mathcal{S}}$

In the case of $\mathcal{A}_{\mathcal{S}}$, to achieve sufficient detection-resilience, $\theta_{\emptyset}$ is often extremely large (e.g., $\theta_{\emptyset} \approx 50K$ in the case of *Conficker.C*), while $\theta_{\mathtt{q}}$ is in contrast much smaller (e.g., $\theta_{\mathtt{q}} = 500$ for *Conficker.C*), which makes case (ii) in Eqn (2) easily dominate the overall distribution. For example, with the setting of *Conficker.C* [21]: $\theta_{\mathtt{q}} = 500$, $\theta_{\exists} = 2$, and $\theta_{\emptyset} = 49998$, we have $\mathsf{Pr}(q = \theta_{\mathtt{q}}) = 0.98$. We can thus approximate the subset sizes q_1, q_2, ... all with a constant $\theta_{\mathtt{q}}$, i.e.,

$$\prod_{j=1}^{n} \frac{\binom{i}{q_j}}{\binom{\theta_{\emptyset}}{q_j}} \approx \left[\frac{\binom{i}{\theta_{\mathtt{q}}}}{\binom{\theta_{\emptyset}}{\theta_{\mathtt{q}}}} \right]^n$$

which leads to the following theorem.

Theorem 1. *The expectation of bot population* $\mathtt{N}$ *necessary to generate at least* k *distinct NXDs is given by:*

$$\mathbb{E}(\mathtt{N}) = \sum_{i=0}^{k-1} (-1)^{k+i} \binom{\theta_{\emptyset}}{i} \binom{\theta_{\emptyset} - i - 1}{\theta_{\emptyset} - k} \frac{\binom{\theta_{\emptyset}}{\theta_{\mathtt{q}}}}{\binom{i}{\theta_{\mathtt{q}}} - \binom{\theta_{\emptyset}}{\theta_{\mathtt{q}}}}$$

$\mathcal{M}_{\mathcal{C}}$ for $\mathcal{A}_{\mathcal{P}}$

In the case of $\mathcal{A}_{\mathcal{P}}$, bots share the same query pool but look up domains in the pool in random permutation orders. Thus a bot either aborts at a valid domain or queries all NXDs in the pool. From the perspective of $\mathcal{M}_{\mathcal{C}}$, we can consider $\mathcal{A}_{\mathcal{P}}$ as a special case of $\mathcal{A}_{\mathcal{S}}$ under the setting of $\theta_{\mathtt{q}} = \theta_{\emptyset}$.

However, unlike $\mathcal{A}_{\mathcal{S}}$, case (ii) in Eqn (2) no longer dominates the overall distribution due to $\theta_{\mathtt{q}} = \theta_{\emptyset}$. For example, with the setting of *Necurs* [4]: $\theta_{\exists} = 2$, $\theta_{\emptyset} = 2046$, and $\theta_{\mathtt{q}} = 2046$, we have $\mathsf{Pr}(q = \theta_{\mathtt{q}}) = 4.77 \times 10^{-7}$. No closed form of $\mathbb{E}(\mathtt{N})$ exists for $\mathcal{A}_{\mathcal{P}}$. We thus resort to Monte Carlo methods [15] to estimate $\mathbb{E}(\mathtt{N})$. In specific, as sketched in Algorithm 2, it first samples a sequence of subset sizes from the distribution of Eqn (2) (line 4); it then estimates an instance of $\mathsf{Pr}(\mathtt{N} = n)$ for each possible population n and computes the expectation $\mathbb{E}(\mathtt{N})$ using these samples (line 5-9); this procedure is repeated a number of times to derive the average estimation (line 10). Here Z is a normalizer to ensure the probabilities of all possible n's sum up to one.

Algorithm 2: Combinatorial Estimator $\mathcal{M}_{\mathcal{C}}$ (for $\mathcal{A}_{\mathcal{P}}$)

```
Input: $k$ - # of distinct NXDs observed; $\bar{n}$ - maximum # of
       bots; $\bar{i}$ - maximum # of iterations
Output: expectation of bot population
1  $i \leftarrow 0$, $\mathtt{N} \leftarrow 0$;
2  while $i$++ $< \bar{i}$ do
3      $\mathtt{N}_i \leftarrow 0$, $Z \leftarrow 0$;
       // Monte Carlo method
4      sample subset sizes $q_1, q_2, \dots, q_{\bar{n}}$ according to Eqn (2);
       // loop over possible bot populations
5      for $n = 1, 2, \dots, \bar{n}$ do
6          compute $\mathsf{Pr}(\mathtt{N} = n)$ according to Eqn (3);
7          $\mathtt{N}_i \leftarrow \mathtt{N}_i + n \cdot \mathsf{Pr}(\mathtt{N} = n)$;
8          $Z \leftarrow Z + \mathsf{Pr}(\mathtt{N} = n)$;
9      $\mathtt{N} \leftarrow \mathtt{N} + \mathtt{N}_i / Z$;
10 return $\mathtt{N} / \bar{i}$;
```

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

$\mathcal{A}_{\mathcal{R}}$ differs from $\mathcal{A}_{\mathcal{S}}$ and $\mathcal{A}_{\mathcal{P}}$ in that it dictates a global sequential order over domains in the query pool; meanwhile, it differs from $\mathcal{A}_{\mathcal{U}}$ in that each bot randomly selects a sequence number as the starting point to query (cf. § III-C). These unique characteristics demand an estimator tailored to $\mathcal{A}_{\mathcal{R}}$.

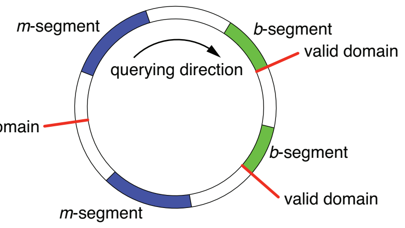

Figure 6: Illustration of DNS dynamics of $\mathcal{A}_{\mathcal{R}}$.

We introduce the following model to describe $\mathcal{A}_{\mathcal{R}}$ (which is illustrated in Figure 6): the DGA-domains form a circle with the clockwise direction indicating their 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.

Conceivably, all distinct NXDs queried by bots during an epoch form a set of *segments*, each comprising consecutive NXDs. Given the set of arcs and the set of segments associated with them, we intend to infer the number $\mathtt{N}$ of bots responsible for generating NXDs to "cover" all such segments. We differentiate two types of segments, namely, those that end in the middle of an arc (m-segment) and those that end at an arc boundary (b-segment). We have the following key observations. (a) An m-segment must be covered by a set of bots in case (ii) of Eqn (2). (b) A b-segment of length less than $\theta_{\mathtt{q}}$ must be covered by bots in case (i). (c) A b-segment of length no less than $\theta_{\mathtt{q}}$ must be covered by a mixture of bots in case (i) and (ii).

Based on these observations, we have the following theorem that establishes the expectation of bot population required to cover a given m-/b-segment .

Theorem 2. *Let* $f(\tilde{l}, n, m) = \frac{m!}{\tilde{l}^n}\binom{\tilde{l}}{m}\left(\left\{{n \atop m}\right\} - \tilde{l}\left\{{n-1 \atop m}\right\}\right)$ [4] *and* $g(\tilde{l}, m) = \sum_{k=0}^{\left\lfloor\frac{\tilde{l}-2}{\theta_{\mathtt{q}}}\right\rfloor}(-1)^k\binom{m-1}{k}\binom{\tilde{l}-k\theta_{\mathtt{q}}-2}{m-2}/\binom{\tilde{l}-2}{m-2}$. *We define:*

$$h(\tilde{l}, n) = \sum_{m=1}^{\tilde{l}} f(\tilde{l}, n, m)g(\tilde{l}, m) \qquad (4)$$

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:*

$$\mathbb{E}(\mathtt{N}_{\mathtt{L}}) = \sum_{n=1}^{+\infty} n \sum_{\tilde{l}=l_l}^{l_u} h(\tilde{l}, n)$$

Theorem 2 naturally leads to an estimation model, which we refer to as the Bernoulli estimator ($\mathcal{M}_{\mathcal{B}}$). As sketched in Algorithm 3, $\mathcal{M}_{\mathcal{B}}$ directly applies Theorem 2 to estimate the expectation of bot population required to cover each segment (line 3-10) and then aggregates the results as the overall bot population (line 11). Here Z is a normalizer, which ensures for each segment the probabilities of all possible bot populations as well as effective lengths sum up to one (details are referred to the Appendix).

[4]Here $\{\}$ represent Stirling numbers of the second kind.

Algorithm 3: Bernoulli Estimator $\mathcal{M}_{\mathcal{B}}$

```
    Input: $\mathcal{S}$ - observed segments, $\bar{n}$ - maximum # of bots
    Output: expectation of bot population
 1  $\mathtt{N} \leftarrow 0$;
 2  for each segment $\mathtt{L} \in \mathcal{S}$ (assuming $|\mathtt{L}| = l$) do
 3      $\mathtt{N}_{\mathtt{L}} \leftarrow 0$, $Z \leftarrow 0$, $l_l \leftarrow l - \theta_{\mathtt{q}} + 1$;
 4      if $\mathtt{L}$ is m-segment then $l_u \leftarrow l_l$;
        // b-segement
 5      else $l_u \leftarrow l$;
        // loop over possible bot populations
 6      for $n = 1, 2, \ldots, \bar{n}$ do
            // loop over possible effective lengths
 7          for $\tilde{l} = l_l, l_l + 1, \ldots, l_u$ do
 8              compute $h(\tilde{l}, n)$ according to Eqn (4);
 9              $\mathtt{N}_{\mathtt{L}} \leftarrow \mathtt{N}_{\mathtt{L}} + n \cdot h(\tilde{l}, n)$;
10              $Z \leftarrow Z + h(\tilde{l}, n)$;
11      $\mathtt{N} \leftarrow \mathtt{N} + \mathtt{N}_{\mathtt{L}}/Z$;
12  return $\mathtt{N}$;
```

V. Empirical Evaluation

This section presents our empirical studies on 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. We start with describing the synthetic data.

A. Evaluation over Synthetic Data

DGA Simulator

We first implemented a set of simulators generating realistic DNS traffic according to different DGA models. More specifically, we fix the drain-and-replenish model (cf. § III-B) as the query pool model given its popularity and consider varied query barrel models $\mathcal{A}_{\mathcal{U}}$, $\mathcal{A}_{\mathcal{S}}$, $\mathcal{A}_{\mathcal{R}}$, and $\mathcal{A}_{\mathcal{P}}$ (cf. § III-C). The default setting of global parameters 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 II. Also for Algorithm 2 and 3, the maximum possible bot population $\bar{n} = 500$ and the maximum number of iterations $\bar{i} = 100$.

DGA Model	**Prototype**	$\boldsymbol{\theta_{\emptyset}}$	$\boldsymbol{\theta_{\exists}}$	$\boldsymbol{\theta_{\mathtt{q}}}$	$\boldsymbol{\delta_{\mathtt{i}}}$
$\mathcal{A}_{\mathcal{U}}$	*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 II. DGA-specific parameter setting.

Given a population of $\mathtt{N}$ bots in the network, we assume that their activations follow a Poisson process, which is a counting process widely applied to model networking events [26]. 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

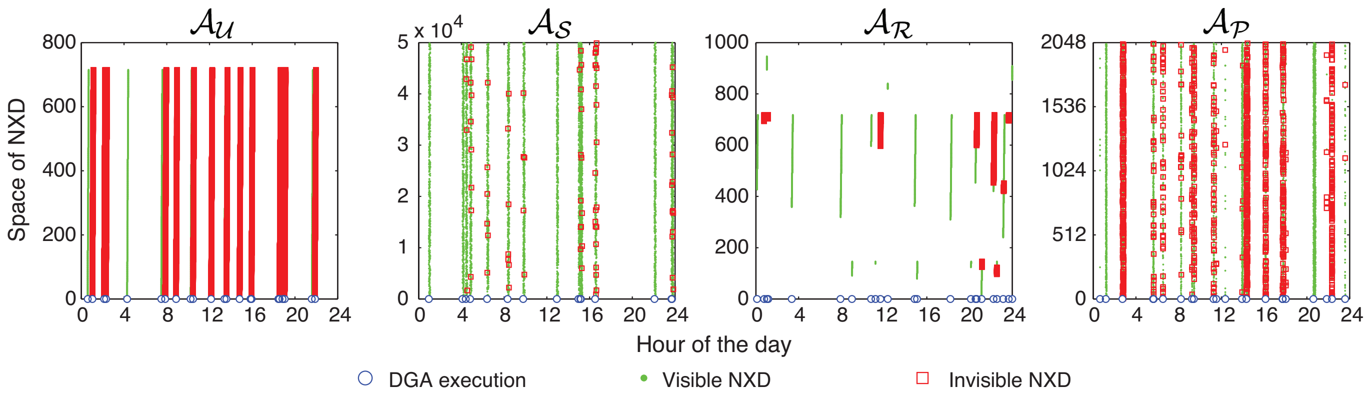

Figure 7: Domain generation behaviors of DGAs following different models and their interplay with caching and forwarding mechanisms (negative caching TTL = 2 hour, bot population N = 25, observation window = 1 day)

(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.

Given DGA $\mathcal{A}$, the simulator outputs a sequence of tuples of the form $\langle$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.

Figure 7 illustrates domain generation behaviors of the DGA families listed in Table II and the impact of caching and forwarding mechanisms over our observations at a border DNS server. We use a constant activation rate λ_0 corresponding to bot population $\mathtt{N} = 25$. It is observed that different DGAs show vastly diverse behaviors in terms of the total number of NXDs generated as well as the overlap between NXDs generated by different bots. Moreover, the caching and forwarding mechanisms significantly influence our observations. For example, in the case of $\mathcal{A}_\mathcal{U}$, since all bots generate the same set of NXDs, most lookups are "masked" out, representing a great challenge for accurate assessment of bot populations.

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, which is defined as:

$$\text{ARE} = \left| \frac{\text{estimated population} - \text{actual population}}{\text{actual population}} \right| \quad (5)$$

In conjunction of all the DGA models ($\mathcal{A}_\mathcal{U}$, $\mathcal{A}_\mathcal{S}$, $\mathcal{A}_\mathcal{R}$, and $\mathcal{A}_\mathcal{P}$), we consider five key parameters, including: (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. More concretely, in each set of experiments, with the remaining parameters fixed, we vary one particular parameter and observe its influence over the performance of BotMeter. Following the roadmap in Table I, we apply the Timing estimator ($\mathcal{M}_\mathcal{T}$) to all the DGA models, the Poisson estimator ($\mathcal{M}_\mathcal{P}$) to $\mathcal{A}_\mathcal{U}$, the Combinatorial estimator ($\mathcal{M}_\mathcal{C}$) to $\mathcal{A}_\mathcal{S}$ and $\mathcal{A}_\mathcal{P}$, and the Bernoulli estimator ($\mathcal{M}_\mathcal{B}$) to $\mathcal{A}_\mathcal{R}$. All the experiments are conducted on a workstation with two 2.53 GHz Intel Xeon quad core CPUs and 96 GB of RAM running CentOS 5.8.

Experiment Results

Population of DGA-bots - In the first set of experiments, we measure the estimation accuracy of different estimators as a function of bot population N existing in the network[5]. As illustrated in Figure 8 (a), when N varies from 16 to 256, we observe shrinking error bars (25*th*-75*th* percentile of errors) across all 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 (5)), amplifiying the difference of absolute errors when N is small. However, as 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 $\mathcal{M}_\mathcal{T}$. Interestingly, in the cases of $\mathcal{A}_\mathcal{S}$ and $\mathcal{A}_\mathcal{R}$, we observe continuous 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 4), it is more likely for $\mathcal{M}_\mathcal{T}$ to differentiate bots based on domains they have queried. It is also noticed that $\mathcal{M}_\mathcal{B}$, $\mathcal{M}_\mathcal{C}$, and $\mathcal{M}_\mathcal{P}$ all outperform $\mathcal{M}_\mathcal{T}$, because of their lesser dependence on temporal traits of individual lookups. The case of $\mathcal{A}_\mathcal{P}$ is rather interesting, wherein as query barrels can be of size similar to query pools (i.e., $\theta_\text{q} \approx \theta_\emptyset$), a large number of active bots may easily generate all possible NXDs, making both $\mathcal{M}_\mathcal{T}$ and $\mathcal{M}_\mathcal{C}$ less effective (cf. Figure 7).

[5]Due to the stochastic nature of activation process (cf. § III), the population of bots that are actually activated may differ from N.

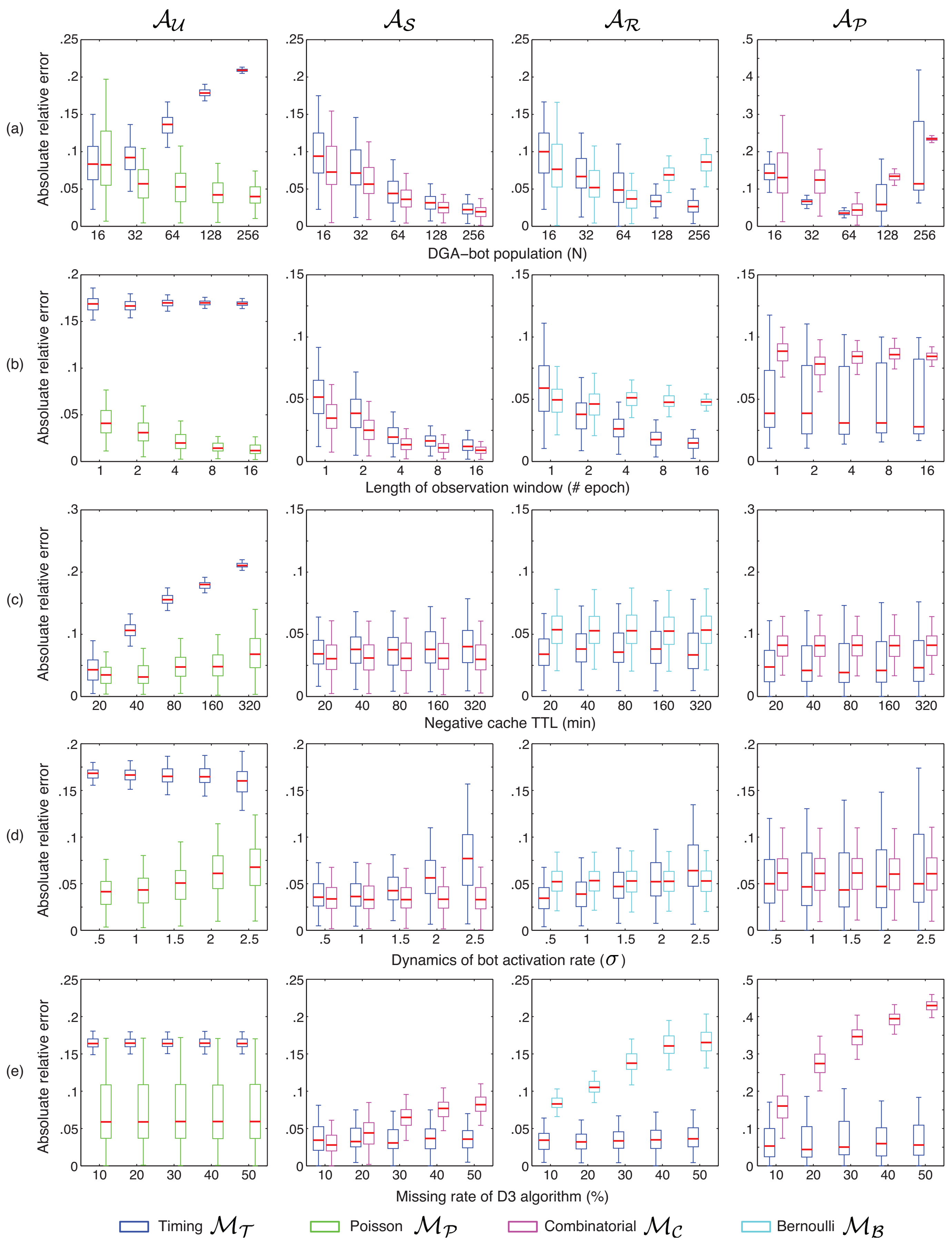

Figure 8: Estimation accuracy of BotMeter with respect to different DGA families and parameter settings.

Length of observation window - In this set of experiments, we gradually increase the length of observation window (1 epoch $\sim$ 16 epochs) and average the estimates over the number of epochs. The results are depicted in Figure 8 (b). As the window size grows, all the estimators are observed to enjoy improved accuracy in terms of reduced medium errors and/or diminished error bars. Intuitively, this is because 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 4): stronger randomness leads to higher estimate variance, but also more improvement margin when the observation window is enlarged.

TTL of negative cache - We intend to understand the impact of caching mechanism at local DNS servers over the performance of BotMeter. As demonstrated in Figure 8 (c), $\mathcal{M}_{\mathcal{T}}$ suffers extensive 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 not able to identify any bots that are completely masked by the caching. Meanwhile, because $\mathcal{M}_{\mathcal{B}}$ and $\mathcal{M}_{\mathcal{C}}$ only rely on collective statistics of the bot population (i.e., distinct NXDs that have been queried by DGA-bots), their 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. Here 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 8 (d). It is noted that like the negative cache TTL, the dynamics of bot activation rate only affects temporal traits of DNS lookups, which $\mathcal{M}_{\mathcal{C}}$ and $\mathcal{M}_{\mathcal{B}}$ are 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 increasing 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 (i.e., one day in our experiments).

Coverage of D3 algorithm - So far we have implicitly assumed an ideal D3 algorithm which correctly identifies all DGA-NXDs. In reality, however, this algorithm may detect only a subset of such DGA-NXDs for reasons specified in § II-D. It is crucial to understand how the detection window of D3 algorithm impacts the estimators. In this set of experiments, we assume that the D3 algorithm randomly misses x percent of DGA-NXDs, where x increases from 10 to 50. The results are shown in Figure 8 (e). As expected, $\mathcal{M}_{\mathcal{C}}$ and $\mathcal{M}_{\mathcal{B}}$ observe extensive accuracy losses as the detection window shrinks (e.g., the medium error of $\mathcal{M}_{\mathcal{C}}$ increases from .16 to .43 in the case of $\mathcal{A}_{\mathcal{P}}$), because they solely rely 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 $\mathcal{M}_{\mathcal{T}}$, because of their dependency on temporal traits of individual NXD lookups: concretely, 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*

To assess the efficacy of BotMeter in actual deployment, we evaluate its performance using real DNS traces collected from a large-scale enterprise network over a longitudinal period. We start with describing the data collection.

Experiment Setup

We collected DNS lookup 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 collected 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 $\langle$timestamp, client, domain$\rangle$ while the observable dataset consists of a series of tuples of $\langle$timestamp, domain$\rangle$. Note that the forwarding server is omitted here given there is only one local server. The granularity of timestamp is one second.

Furthermore, from *DGArchive*[6], a lookup service for domain generating malware, we extracted a number of DGAs, which are believed to be active during this time period. Using the APIs provided by *DGArchive*, we queried and collected all the domains generated by these DGAs during the given time period, which we refer to as the *pool* dataset. We then matched 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 identified the population of active bots corresponding to each DGA for each day, which serves as the ground truth in our experiments[7]. Meanwhile, we fed the correlation results of the observable and pool datasets as input to the analytical models of BotMeter to evaluate its performance.

Figure 9 depicts the total number of NXD lookups which have been generated by three notable DGAs (*newGoZ*, *Qakbot*, and *Ramnit*) and have been spotted in the raw dataset during the given time period. It is noticed that different DGA-bots exhibit highly varying activity levels. Specifically, the *newGoZ* bots show peak activities between 09/12/2014 and 09/19/2014,

[6]DGArchive: https://dgarchive.caad.fkie.fraunhofer.de

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

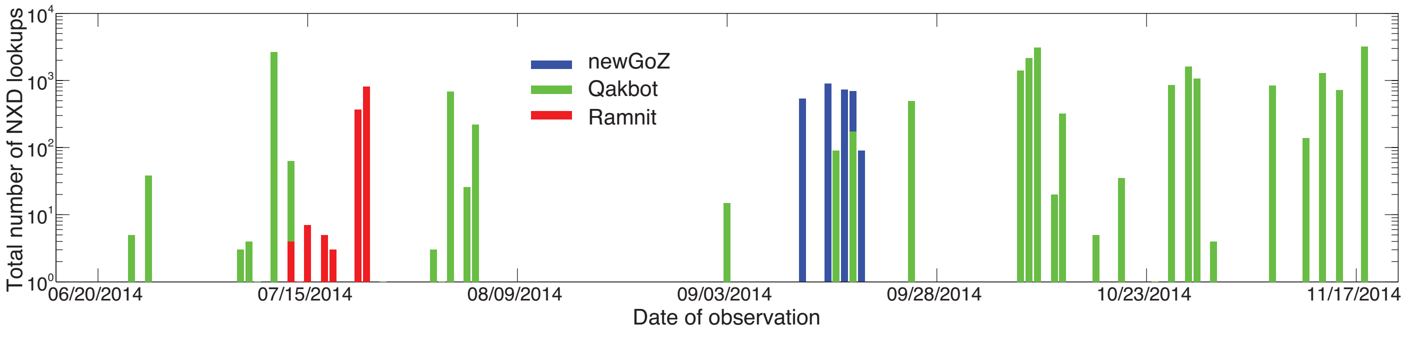

Figure 9: Total number of DGA-generating NXDs observed at a local DNS server within a large enterprise network over the time period of 05/01/2014 $\sim$ 04/30/2015.

an outbreak of the *Ramnit* bots is observed during 07/13/2014 $\sim$ 07/22/2014, while the *Qakbot* bots are spotted over the entire period from 06/24/2014 to 11/18/2014.

Next we fix the length of observation window as one day and apply BotMeter to estimating the daily population of active bots corresponding 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* (cf. Table I).

Experiment Results

We contrast the estimates made by $\mathcal{M}_\mathcal{T}$, $\mathcal{M}_\mathcal{B}$, and $\mathcal{M}_\mathcal{P}$ (when applicable) against the actual daily populations of active bots (i.e., the ground truth). Figure 10 illustrates in detail the DGA-bot populations and the estimates for each day, with the accuracy of different estimators summarized in Table III.

It is observed that $\mathcal{M}_\mathcal{P}$ and $\mathcal{M}_\mathcal{B}$ perform highly accurate estimation with respect to 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 III, 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 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, it is observed that $\mathcal{M}_\mathcal{B}$ performs slightly better than $\mathcal{M}_\mathcal{P}$. This may be explained by that (i) $\mathcal{M}_\mathcal{B}$ does not depend on any temporal traits and (ii) 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 also echo our findings in § V-A.

C. Lessons Learned

Bestowed with a rich library of estimators in BotMeter, yet, we face the critical question of selecting the most suitable estimator for given target DGA and DNS data.

As summarized in Table I, the target DGA narrows down this choice to one of the specialized estimators ($\mathcal{M}_\mathcal{B}$, $\mathcal{M}_\mathcal{C}$, and $\mathcal{M}_\mathcal{P}$) or $\mathcal{M}_\mathcal{T}$. Nevertheless, it is the given DNS data that determines the superiority of the specialized estimator or $\mathcal{M}_\mathcal{T}$. From the experiments using both synthetic and real traces, we have learned the following lessons. When the temporal traits of individual DNS lookups are fine-grained and accurate, $\mathcal{M}_\mathcal{T}$ is a more generally applicable estimator, as it is less sensitive to the detection window of D3 algorithm. In contrast, when the coverage of D3 algorithm is high, leading to statistics faithfully reflecting bots' collective behaviors, $\mathcal{M}_\mathcal{P}$ is more reliable than $\mathcal{M}_\mathcal{T}$ under modest dynamics of DGA-bot activations, while $\mathcal{M}_\mathcal{B}$ and $\mathcal{M}_\mathcal{C}$ are better options than $\mathcal{M}_\mathcal{T}$, as they are less subject to the DNS cache TTL, the dynamics of bot activation rate, and the timestamp granularity.

Finally, astute readers may notice that we have implicitly assumed a priori knowledge on the model of target DGA. In the case that one is completely agnostic about the DGA model, the thumb rule is to apply $\mathcal{M}_\mathcal{T}$ given its general applicability to all the DGA models.

VI. Related Work

To comprehend the operations of botnets, the security community has conducted intensive research on domain generation algorithms (DGAs). Most existing research has focused on detecting DGA-generated domains and identifying compromised machines to hinder C2 traffic. Yadav et al. [36] proposed to discover DGA-domains by leveraging their characteristic distributions of alphanumeric characters and bigrams. Schiavoni et al. [29] also leveraged linguistic features (e.g., meaningful characters ratio and domain pronounceability) to detect DGA-domains. Antonakakis et al. [12] presented Pleiades, a system that identifies botnet groups running the same DGA by clustering NXDs based on syntactic features. Thomas and Mohaisen [34] 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 [31] 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 backtracking to infected machines. Stone-Gross et al. [33]

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

Table III. Average estimation errors (standard deviation shown in subscripts) of varied estimators in assessing *newGoZ*, *Ramnit*, and *Qakbot* bot populations.

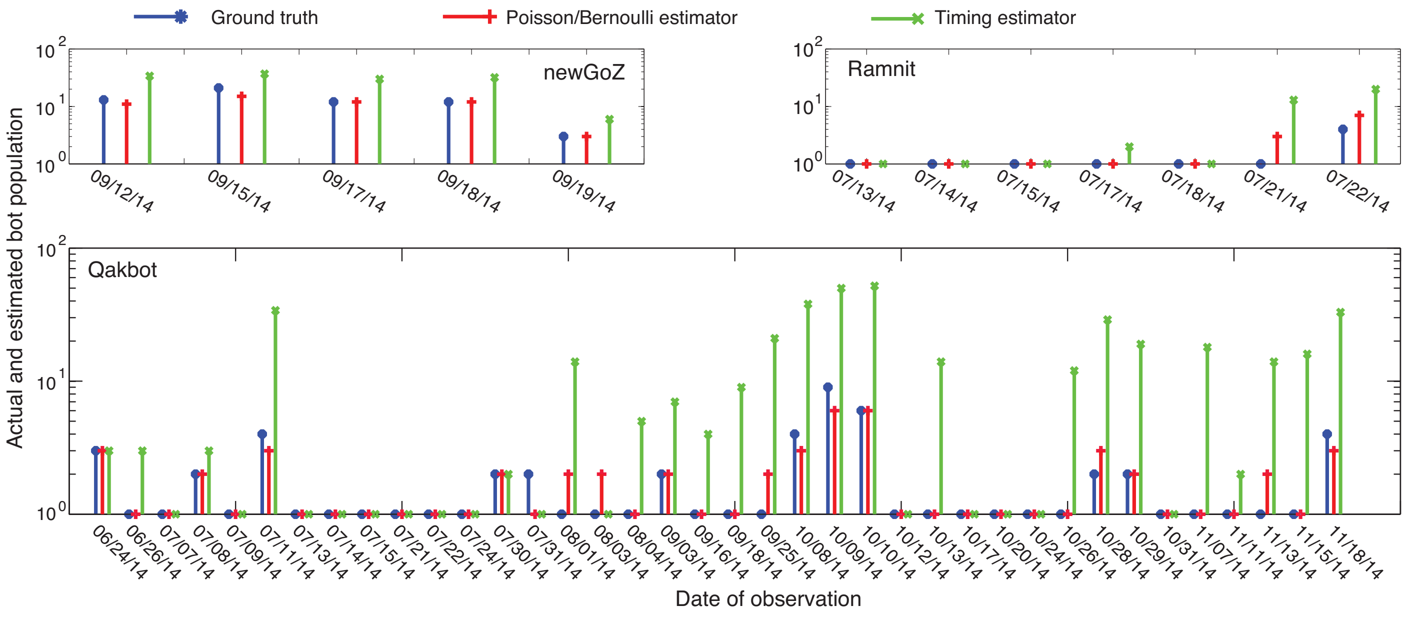

Figure 10: Daily populations of active *newGoZ*, *Ramnit*, and *Qakbot* bots and estimates made by $\mathcal{M}_{\mathcal{P}}$, $\mathcal{M}_{\mathcal{B}}$, and $\mathcal{M}_{\mathcal{T}}$ over the period of 05/01/2014 $\sim$ 04/30/2015.

monitored *Torpig* botnet operations for ten days by hijacking its C2 communications and measured the size of the botnet using unique identifiers of compromised hosts. Nelms et al. [24] 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 every individual machine and identifying positive matches with confirmed DGA-domains. This work, to our best knowledge, is the first non-intrusive solution capable of accurately assessing DGA-bot population distributions while incurring minimal operational costs.

In general, as DNS systems have become one primary target for Internet miscreants (e.g., fast-fluxing, domain-fluxing, DNS tunneling), extensive research has been conducted on misuse and abuse of DNS systems. Bilge et al. [14] proposed a passive DNS analysis method to detect domains involved in malicious activities based on time-, DNS response-, TTL-, and linguistic-features. Antonakakis et al. [11] presented a system that detects malware domains via analyzing global DNS query resolution patterns. Hao et al. [18] investigated the domain registration process of *com* TLD to derive time-of-registration features (e.g., bulk registration and expired domain reuse) to determine the likeliness of malicious use. Alrwais et al. [9] studied domains hosted by major domain parking services to understand their monetization mechanisms, including click fraud, traffic spam, and traffic stealing. Vissers et al. [35] confirmed that parked domains were used for distributing malware and abusing popular domains through typosquatting, and developed a client-side classifier to determine parked domains leveraging features such as third-party advertising and malicious redirections. We note that this work complements these studies by examining DNS abusing behaviors *inside* enterprise networks.

VII. Conclusion

In this paper, we present BotMeter, a novel tool designed 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 in BotMeter to capture the DNS dynamics of a wide range 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; 3) (from attacker's perspective) designing advanced DGA models that evade effective population estimation; and 4) systemizing existing DGAs via integrating taxonomies in literature and our new taxonomy.

References

[1] DGAs in the Hands of Cyber Criminals. http://www.damballa.com/downloads/r_pubs/WP_DGAs-in-the-Hands-of-Cyber-Criminals.pdf.

[2] GameOver Zeus Mutates, Launches Attacks. http://blog.malcovery.com/breaking-gameover-zeus-returns.

[3] Malware Authors Expand Use of Domain Generation Algorithms to Evade Detection. http://www.pcworld.com/article/250824/.

[4] Necurs: The Malware that Breaks Your Security. http://www.trendmicro.com/vinfo/us/threat-encyclopedia/web-attack/3133/necurs-the-malware-that-breaks-your-security.

[5] On the Kraken and Bobax Botnets. http://www.damballa.com/downloads/r_pubs/Kraken_Response.pdf.

[6] Ranbyus Banking Trojan, Cousin of Zbot. http://www.mysonicwall.com/sonicalert/searchresults.aspx?ev=article&id=693.

[7] Rovnix and the declaration generation algorithm. http://researchcenter.paloaltonetworks.com/2014/10/rovnix-declaration-generation-algorithm/.

[8] Unveiling The Latest Variant of Pushdo. http://www.damballa.com/downloads/r_pubs/Damballa_mv20_case_study.pdf.

[9] S. Alrwais, K. Yuan, E. Alowaisheq, Z. Li, and X. Wang. Understanding the Dark Side of Domain Parking. In *Proceedings of the 23rd USENIX Conference on Security*, 2014.

[10] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a Dynamic Reputation System for DNS. In *Proceedings of the 19th USENIX Conference on Security*, 2010.

[11] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou, II, and D. Dagon. Detecting malware domains at the upper dns hierarchy. In *Proceedings of the 20th USENIX Conference on Security*, 2011.

[12] 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 *Proceedings of the 21st USENIX conference on Security symposium*, 2012.

[13] T. Barabosch, A. Wichmann, F. Leder, and E. Gerhards-Padilla1. Automatic Extraction of Domain Name Generation Algorithms from Current Malware. In *Proceedings of the 2012 NATO Symposium on Information Assurance and Cyber Defense*, 2012.

[14] L. Bilge, S. Sen, D. Balzarotti, E. Kirda, and C. Kruegel. Exposure: A passive dns analysis service to detect and report malicious domains. *ACM Trans. Inf. Syst. Secur.*, 16(4):14:1–14:28, 2014.

[15] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006.

[16] F. Giroire, J. Chandrashekar, N. Taft, E. Schooler, and D. Papagiannaki. Exploiting Temporal Persistence to Detect Covert Botnet Channels. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, 2009.

[17] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium*, 2008.

[18] S. Hao, M. Thomas, V. Paxson, N. Feamster, C. Kreibich, C. Grier, and S. Hollenbeck. Understanding the domain registration behavior of spammers. In *Proceedings of the 2013 ACM SIGCOMM Conference on Internet Measurement*, 2013.

[19] IETF. Common DNS Operational and Configuration Errors. http://tools.ietf.org/html/rfc1912, 1996.

[20] N. L. Johnson and S. Kotz. *Urn Models and Their Applications: An Approach to Modern Discrete Probability Theory*. Wiley, New York, 1977.

[21] F. Leder and T. Werner. Know Your Enemy: Containing Conficker, To Tame a Malware. http://www.honeynet.org/files/KYE-Conficker.pdf, 2009.

[22] Y. Nadji, M. Antonakakis, R. Perdisci, D. Dagon, and W. Lee. Beheading Hydras: Performing Effective Botnet Takedowns. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, 2013.

[23] 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 *Proceedings of the 7th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2010.

[24] T. Nelms, R. Perdisci, and M. Ahamad. Execscent: Mining for new c&c domains in live networks with adaptive control protocol templates. In *Proceedings of the 22Nd USENIX Conference on Security*, 2013.

[25] P. Porras, H. Saïdi, and V. Yegneswaran. A Foray into Conficker's Logic and Rendezvous Points. In *Proceedings of the 2nd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*, 2009.

[26] S. M. Ross. *Stochastic Processes (Wiley Series in Probability and Statistics)*. Wiley, 2 edition, 1995.

[27] 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 *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, 2013.

[28] P. Royal. On the Kraken and Bobax Botnets. http://www.damballa.com/downloads/r_pubs/Kraken_Response.pdf, 2008.

[29] S. Schiavoni, F. Maggi, L. Cavallaro, and S. Zanero. Phoenix: DGA-Based Botnet Tracking and Intelligence. In *Proceedings of the 11th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2014.

[30] Sergei Shevochenko. Domain Name Generator for Murofet. http://blog.threatexpert.com/2010/10/domain-name-generator-for-murofet.html, 2010.

[31] R. Sharifnya and M. Abadi. A novel reputation system to detect DGA-based botnets. In *Proceedings of the 2013 International Conference on Computer and Knowledge Engineering*, 2013.

[32] S. Shevochenko. Srizbi's Domain Calculator. http://blog.threatexpert.com/2008/11/srizbis-domain-calculator.html, 2008.

[33] 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 *Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009.

[34] M. Thomas and A. Mohaisen. Kindred domains: detecting and clustering botnet domains using DNS traffic. In *Proceedings of the 2014 International Conference on World Wide Web Companion*, 2014.

[35] T. Vissers, W. Joosen, and N. Nikiforakis. Parking Sensors: Analyzing and Detecting Parked Domains. In *Proceedings of the 2015 Network and Distributed System Security Symposium*, 2015.

[36] S. Yadav, A. K. K. Reddy, A. N. Reddy, and S. Ranjan. Detecting Algorithmically Generated Malicious Domain Names. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010.

Appendix

Proof: (Lemma 1) Note that this setting is similar to the committee selection problem in [20]. Consider all $\theta_\emptyset$NXDs in the query pool. Denote by X_n the number of distinct NXDs included in at least one of n query barrels of size $q_1, q_2, \ldots, q_n$, respectively. Applying the result in [20], we first have:

$$\mathsf{Pr}(X_n = k) = \binom{\theta_\emptyset}{k} \sum_{i=0}^{k} (-1)^{k-i} \binom{k}{i} \prod_{j=1}^{n} \frac{\binom{i}{q_j}}{\binom{\theta_\emptyset}{q_j}}$$

We then have the following derivation:

$$\begin{aligned}
&\mathsf{Pr}(X_n \geq k)\\
= \quad & \sum_{\kappa=k}^{\theta_\emptyset} \binom{\theta_\emptyset}{\kappa} \sum_{i=0}^{\kappa} (-1)^{\kappa-i} \binom{\kappa}{i} \prod_{j=1}^{n} \frac{\binom{i}{q_j}}{\binom{\theta_\emptyset}{q_j}}\\
= \quad & \sum_{i=0}^{\theta_\emptyset} (-1)^i \prod_{j=1}^{n} \frac{\binom{i}{q_j}}{\binom{\theta_\emptyset}{q_j}} \sum_{\kappa=\max(i,k)}^{\theta_\emptyset} (-1)^{\kappa} \binom{\theta_\emptyset}{\kappa} \binom{\kappa}{i}\\
= \quad & \sum_{i=0}^{\theta_\emptyset} (-1)^i \prod_{j=1}^{n} \frac{\binom{i}{q_j}}{\binom{\theta_\emptyset}{q_j}} \sum_{\kappa=\max(i,k)}^{\theta_\emptyset} (-1)^{\kappa} \binom{\theta_\emptyset}{i} \binom{\theta_\emptyset - i}{\kappa - i}
\end{aligned}$$

$$= \sum_{i=0}^{k-1}(-1)^{k+i}\binom{\theta_\emptyset}{i}\binom{\theta_\emptyset - i - 1}{\theta_\emptyset - k}\prod_{j=1}^{n}\frac{\binom{i}{q_j}}{\binom{\theta_\emptyset}{q_j}} + 1$$

Now consider the number of bots $\mathtt{N}$ necessary to obtain at least k distinct NXDs. We have:

$$\begin{aligned} & \mathsf{Pr}(\mathtt{N} = n) \\ = \; & \mathsf{Pr}(X_n \geq k) - \mathsf{Pr}(X_{n-1} \geq k) \\ = \; & \sum_{i=0}^{k-1}(-1)^{k+i}\binom{\theta_\emptyset}{i}\binom{\theta_\emptyset - i - 1}{\theta_\emptyset - k}\frac{\binom{i}{q_n} - \binom{\theta_\emptyset}{q_n}}{\binom{\theta_\emptyset}{q_n}}\prod_{j=1}^{n-1}\frac{\binom{i}{q_j}}{\binom{\theta_\emptyset}{q_j}} \end{aligned}$$

which proves the result in Lemma 1. ■

Proof: (Theorem 1) Applying the approximation of $q_j = \theta_\mathtt{q}$ for all j, we obtain:

$$\prod_{j=1}^{n}\frac{\binom{i}{q_j}}{\binom{\theta_\emptyset}{q_j}} \approx \left[\frac{\binom{i}{\theta_\mathtt{q}}}{\binom{\theta_\emptyset}{\theta_\mathtt{q}}}\right]^n \triangleq z^n$$

Applying Lemma 1, we have the derivation below:

$$\begin{aligned} & \mathbb{E}(\mathtt{N}) \\ = \; & \sum_{n=1}^{+\infty} n\mathsf{Pr}(\mathtt{N} = n) \\ = \; & \sum_{n=1}^{+\infty} n \sum_{i=0}^{k-1}(-1)^{k+i}\binom{\theta_\emptyset}{i}\binom{\theta_\emptyset - i - 1}{\theta_\emptyset - k}\frac{\binom{i}{\theta_\mathtt{q}} - \binom{\theta_\emptyset}{\theta_\mathtt{q}}}{\binom{\theta_\emptyset}{\theta_\mathtt{q}}} z^{n-1} \\ = \; & \sum_{i=0}^{k-1}(-1)^{k+i}\binom{\theta_\emptyset}{i}\binom{\theta_\emptyset - i - 1}{\theta_\emptyset - k}\frac{\binom{i}{\theta_\mathtt{q}} - \binom{\theta_\emptyset}{\theta_\mathtt{q}}}{\binom{\theta_\emptyset}{\theta_\mathtt{q}}}\boxed{\sum_{n=1}^{+\infty} n z^{n-1}} \end{aligned}$$

It is noted that $0 < z < 1$. By means of the identity, we have $\sum_{n=1}^{+\infty} n z^{n-1} = \frac{1}{(1-z)^2}$. By substituting the boxed part in $\mathbb{E}(\mathtt{N})$ with this result, we prove Theorem 1. ■

Proof: (Theorem 2) Note that each bot selects its starting position at random, so we examine each segment independently. Let us start with an m-segment L of length l. Denote by $\tilde{L}$ the effective region of L, over which a bot's starting position can be placed. The length of $\tilde{L}$ is defined as: $\tilde{l} = l - \theta_\mathtt{q} + 1$. We introduce two sets of variables: X_n represents combined length covered by n bots over L such that (i) they all start within $\tilde{L}$ and (ii) there is no gap between these domains; Z_l represents the number of bots required to cover L.

First consider some trivial cases. If $l < \theta_\mathtt{q}$, $\mathsf{Pr}(X_n = l) = 0$ for $n \geq 1$; If $l = \theta_\mathtt{q}$, $\mathsf{Pr}(X_n = l) = \frac{1}{\tilde{l}^n}$. In the following we assume $l > \theta_\mathtt{q}$.

For given n, let $p(\tilde{l}, n, m)$ represent the probability that m distinct positions in $\tilde{L}$ are selected as stating points of these n bots. Using the inclusion-exclusion principle, we have:

$$\begin{aligned} p(\tilde{l}, n, m) &= \binom{\tilde{l}}{m}\frac{1}{\tilde{l}^n}\sum_{i=0}^{m}(-1)^i\binom{m}{i}(m-i)^n \\ &= \frac{m!}{\tilde{l}^n}\binom{\tilde{l}}{m}\left\{ n \atop m \right\} \end{aligned} \tag{6}$$

where $\left\{ n \atop m \right\}$ denotes Stirling numbers of the second kind.

We then introduce $g(\tilde{l}, m)$, which represents the probability that L is covered without a gap and there are m distinct starting positions. Without loss of generality, assume the domains on L are listed as $1, 2, \ldots, l$. Let $Y_1 < Y_2 < \ldots < Y_m$ be the m distinct starting positions. It is noted that $g(\tilde{l}, m)$ corresponds to the case that (i) $Y_m - Y_1 = \tilde{l} - 1$ and (ii) $Y_{i+1} - Y_i \leq \theta_\mathtt{q}$ for $i = 1, 2, \ldots, m-1$. Let $z_i = Y_{i+1} - Y_i$ for all i. The problem is re-formulated as below.

$$\begin{aligned} & z_1 + \ldots + z_{m-1} = \tilde{l} - 1 \\ \text{s.t.} \quad & 0 < z_i \leq \theta_\mathtt{q} \quad (i = 1, 2, \ldots, m-1) \end{aligned} \tag{7}$$

The number of integer solutions for $\sum_{i=1}^{m-1} z_i$ such that $\forall i, z_i > 0$ is given by the binomial coefficient $\binom{\tilde{l}-2}{m-2}$; Further, the number of integer solutions such that $\forall i, z_i > 0$ and there are k distinct terms of $z_1, z_2, \ldots, z_{m-1}$ greater than $\theta_\mathtt{q}$ is given by $\binom{m-1}{k}\binom{\tilde{l}-k\theta_\mathtt{q}-2}{m-2}$.

Applying the inclusion-exclusion principle, we obtain:

$$g(\tilde{l}, m) = \begin{cases} 1 & m = 1, l = \theta_\mathtt{q} \\ 0 & m = 1, l \neq \theta_\mathtt{q} \\ \sum_{k=0}^{\left\lfloor \frac{\tilde{l}-2}{\theta_\mathtt{q}} \right\rfloor}(-1)^k\binom{m-1}{k}\frac{\binom{\tilde{l}-k\theta_\mathtt{q}-2}{m-2}}{\binom{\tilde{l}-2}{m-2}} & m \neq 1 \end{cases}$$

It is clear that $\mathsf{Pr}(X_n = l) = \sum_{m=1}^{\tilde{l}} p(\tilde{l}, n, m) g(\tilde{l}, m)$. Further, the case that it requires n bots to cover L implies that $\mathsf{Pr}(Z_l = n) = \mathsf{Pr}(X_n = l) - \mathsf{Pr}(X_{n-1} = l)$. Now, integrating Eqn (6) and Eqn (7), the expectation of number of bots $\mathtt{N}_L$ to cover L is given by:

$$\begin{aligned} \mathbb{E}(\mathtt{N}_L) &= \sum_{n=1}^{+\infty} n\mathsf{Pr}(Z_l = n) \\ &= \sum_{n=1}^{+\infty} n \sum_{m=1}^{\tilde{l}} \boxed{\left[p(\tilde{l}, n, m) - p(\tilde{l}, n-1, m)\right]} g(\tilde{l}, m) \\ &\triangleq \sum_{n=1}^{+\infty} n \sum_{m=1}^{\tilde{l}} \boxed{f(\tilde{l}, n, m)} g(\tilde{l}, m) \end{aligned}$$

Next we consider a b-segment L of length l. The difference lies in that the effective length $\tilde{l}$ of L may vary from $l - \theta_\mathtt{q} + 1$ to l. However, for each effective length, we can apply the same derivation above. Therefore, the expectation of number of bots $\mathtt{N}_L$ to cover a b-segment L of length l is given as:

$$\begin{aligned} \mathbb{E}(\mathtt{N}_L) &= \sum_{n=1}^{+\infty} n\mathsf{Pr}(Z_l = n) \\ &= \sum_{n=1}^{+\infty} n \sum_{\tilde{l}=l-\theta_\mathtt{q}+1}^{l} \sum_{m=1}^{\tilde{l}} f(\tilde{l}, n, m) g(\tilde{l}, m) \end{aligned}$$

which completes the proof of Theorem 2.

■