On DGA Detection and Classification using P4 Programmable Switches

Ali AlSabeh[a], Kurt Friday[b], Elie Kfoury[a], Jorge Crichigno[a], Elias Bou-Harb[b]

[a] *College of Engineering and Computing, University of South Carolina, Columbia, USA*
[b] *College of Engineering, Louisiana State University, USA*

Abstract

Domain Generation Algorithms (DGAs) are highly effective strategies employed by malware to establish connections with Command and Control (C2) servers. Mitigating DGAs in high-speed networks can be challenging, as it often requires resource-intensive tasks such as extracting high-dimensional features from domain names or collecting extensive network heuristics. In this paper, we propose an innovative framework leveraging the flexibility, per-packet granularity, and Terabits per second (Tbps) processing capabilities of P4 programmable data plane switches for the rapid and accurate detection and classification of DGA families. Specifically, we use P4 switches to extract a combination of unique network heuristics and domain name features through shallow and Deep Packet Inspection (DPI) with minimal impact on throughput. We employ a two-fold approach, comprising a line-rate compact Machine Learning (ML) classifier in the data plane for DGA detection and a more comprehensive classifier in the control plane for DGA detection and classification. To validate our approach, we collected malware samples totaling hundreds of Gigabytes (GBs), representing over 50 DGA families, and utilized campus traffic from normal benign users. Our results demonstrate that our proposed approach can swiftly and accurately detect DGAs with an accuracy of 97% and 99% in the data plane and the control plane, respectively. Furthermore, we present promising findings and preliminary results for detecting DGAs in encrypted Domain Name System (DNS) traffic. Our framework enables the immediate halting of malicious communications, empowering network operators to implement effective mitigation, incident management, and provisioning strategies.

Keywords: DGA detection and classification, P4 programmable switches, high-speed networks, machine learning, DPI, encrypted DNS traffic

1. INTRODUCTION

Cybersecurity Ventures predicts cybercrime costs will soar to $10.5 trillion by 2025 [1]. Attackers use C2 servers, or rendezvous points, for communication to steal data, control compromised systems, and conduct malicious activities [2]. Communication methods are broadly categorized as static or dynamic. Static methods rely on fixed IP addresses and domain names, making them detectable and less favored by adversaries [3]. In contrast, dynamic methods, exemplified by DGAs, continually shift IPs and domains, presenting formidable challenges [4]. DGAs, extensively used by cybercriminals, generate thousands of domains daily to elude next-generation firewalls and denylists [4]. Malware utilizing DGAs queries the DNS to resolve IPs, with only a small fraction being associated with the C2 server, while the remainder yields Non-Existent Domain (NXD) responses [5].

Ultimately, the advent of DGAs has put practitioners at a disadvantage, as such algorithms create a high asymmetry between attackers and defenses, due to the fact that an attacker only requires a single domain from which to control and send commands to infected machines, whereas defenses must control all of the domains in order to mitigate the attack. To put the magnitude of this disadvantage in perspective, the DGA of the `Conficker` botnet, whose first variant appeared in October 2008 and was followed by several other variants, generated tens of thousands of pseudo-random domains registered in more than 100 Top-Level Domains (TLDs). The operational response to `Conficker` required an ad hoc partnership between Internet security researchers, operating system companies, antivirus software vendors, law enforcement parties, ICANN, TLD registries, and registrars around the world to contain the threat [6].

Current research into DGAs primarily focuses on detection (binary decision to segregate DGAs from benign traffic) and classification (multiclass classification to assign the DGA to a family). Generally, such approaches either rely on contextual network traffic collected retrospectively (context-aware) or analyze domain name features without depending on other metadata (context-less) [7]. While context-less approaches can obtain high accuracy, they require a general-purpose Central Processing Unit (CPU) / Graphics Processing Unit (GPU) to extensively process and analyze the domain names, which could create a bottleneck and significantly impact throughput due to the ubiquitous use of DNS on the Internet. On the other hand, context-aware approaches can be slow since they typically analyze batches of traffic offline (e.g., using NetFlow [8]).

According to the 2023 global DNS traffic of one of the largest DNS authoritative DNS providers, NS1 saw a consistent one million DNS queries per second, 10% being NXD responses.

Email addresses: `aalsabeh@email.sc.edu` (Ali AlSabeh), `kfrida1@lsu.edu` (Kurt Friday), `ekfoury@email.sc.edu` (Elie Kfoury), `jcrichigno@cec.sc.edu` (Jorge Crichigno), `ebouharb@lsu.edu` (Elias Bou-Harb)

Additionally, some companies managed by NS1 reported over 60% NXD responses [9]. These statistics render even state-of-the-art approaches that exclusively analyze NXD responses infeasible, necessitating an efficient solution capable of detecting DGAs without disrupting high-speed networks.

While the challenges of promptly fingerprinting DGA artifacts are glaring, the introduction of the P4 programmable switches technology offers practitioners a unique opportunity to achieve this aim. More specifically, the P4 language [10] and the programmable devices it is compiled on, allow network operators to describe in the software (i.e., within the P4 program) how packets are processed. The flexibility offered by the P4 language allows network designers to overcome the restrictions of the Software-Defined Networking (SDN) paradigm coupled with OpenFlow protocol, which is restricted to a fixed set of header fields [11]. For instance, when new versions of OpenFlow emerge, each one should be first approved by the Open Networking Foundation (ONF) and then implemented by hardware manufacturers [12]. Since the emergence of the P4 language, several network applications were offloaded to the data plane, allowing operators to implement custom switch-based programs that execute at wire speed (reaching 25.6 Tbps [13]). A program that performs DPI on a general-purpose server simply cannot keep pace with today's busy networks, resulting in severe network bottlenecks or even crashes.

To address the aforementioned shortcomings, this paper proposes a novel framework for performing the detection and classification of DGA families via the P4 programmable data plane technology. The proposed framework leverages the customization that P4 switches offer to offload a unique hybrid feature extraction technique to the data plane. The selected features are a combination of context-aware and context-less attributes, extracted via shallow and deep inspection of the packet, respectively. Such a framework allows bypassing the latency associated with extracting context-aware analysis and the potential bottlenecks and privacy issues coupled with context-less techniques. To this end, the contributions of this paper are summarized as follows:

- Introducing a novel framework that leverages P4 switches employing a hybrid context-aware and context-less feature extraction technique entirely in the data plane to overcome the associated obstacles of past DGA detection and classification techniques.
- Developing an in-network mechanism on Intel's Tofino chipset [23] for the purpose of processing the entirety of the domain name. This mechanism extracts both structural and statistical features to detect DGAs within $\approx$ 1-2 microseconds (μs), orders of magnitude faster than state-of-the-art (by hundreds and thousands of microseconds). This preserves the privacy of users while reducing the overhead on the control plane associated with feature extraction and preprocessing.
- Evaluating the proposed approach on 57 notable DGA families collected by crawling hundreds of gigabytes of malware samples from multiple sources. The proposed approach demonstrated that it can classify DGAs with very high accuracy from only a small number of NXD responses. The implementation codes and the collected dataset are made publicly available to facilitate research developments in this area [24].
- Presenting novel findings and promising results on detecting DGAs in encrypted DNS over Transport Layer Security (TLS), referred to as DoT, using features that address the limitations of the programmable data plane.

The remainder of the paper is organized as follows. In Section 2, we highlight the related work and how they compare to our approach. Subsequently, Section 3 provides a background on DGAs and P4 switches. In Section 4, we describe the proposed approach, present the selected features, and detail the P4 implementation. We then present the experimental results of the proposed approach in Section 5, in addition to showcasing the methodology and experiments conducted to deal with DoT. Section 6 presents some discussions and limitations pertaining to our proposed system. Section 7 concludes the paper with discussions and future work in this area.

2. Related Work

2.1. DGA Binary and Multiclass Classification

The vast majority of DGA detection techniques rely on using either context-aware or context-less features to output a binary decision on whether a DGA exists in the network or not. In terms of context-aware approaches, Grill et al. [8] utilized NetFlow whereas Iuchi et al. [14] relied on an SDN controller to perform the feature collection for DGA detection. However, such approaches can cause additional delays. For example, the average latency in [14] was approximately 3 seconds, which is larger than the timeout limit of the "nslookup" tool used for resolving domain names (2 seconds). In a novel work, Ahmed et al. [15] developed an SDN monitoring system to monitor TCP/UDP flows corresponding to DGA-based malware-infected devices based on the OpenFlow switches. The proposed approach sends a copy of all incoming DNS responses and checks whether any domain name exists in DGArchive (referred to as suspicious network traffic). If so, a copy of all suspicious network traffic is analyzed by an ML-based packet processing engine. EXPOSURE [16] is a technique that utilizes context-less and context-aware features to detect DGAs. The proposed system reduced the volume of traffic being processed by sampling subsets of traffic to accommodate the server's resources. However, the sampling of data might result in missing some activities that might be malicious.

Alternatively, context-less approaches reduce some of the latency of context-aware methods, at the risk of causing throughput degradation and bottlenecks under high DNS traffic load. FANCI [2] and ANCS [17] used a combination of multiple structural, linguistic, and statistical features to detect malicious Algorithmically Generated Domains (mAGDs), i.e., domain names generated by DGAs. Their models required an average of 2.5 and 100 milliseconds (ms), respectively, to detect if a

Table 1: Comparison between DGA detection and classification approaches with our work.

Approach	Detect	Classify	Features		Traffic Monitored			Performance		
			Context-less	Context-aware	DNS Reqs	EXD Reps	NXD Reps	CPU/GPU	ASIC	Latency
[8]	✓			✓	✓	✓		●		*minutes*
[14]	✓			✓	✓	✓	✓	●		*seconds*
[15]	✓		✓	✓	✓			●		*ms*
EXPOSURE [16]	✓		✓	✓	✓	✓	✓	●		*minutes*
FANCI [2]	✓		✓				✓	●		*ms*
ANCS [17]	✓		✓				✓	●		*ms*
[18]	✓		✓		✓			●		*ms*
[19]	✓		✓			✓	✓	●		*ms*
Phoenix [20]	✓	✓	✓	✓				●		*ms*
Pleiades [21]	✓	✓	✓				✓	●		*ms*
EXPLAIN [7]		✓	✓				✓	●		100's μs
[22]	✓	✓	✓		✓			●		*ms*
Our Approach	✓	✓	✓	✓	✓		✓	⊙	✓	1-2 μs (0/1) 100's μs (1..57)

● Heavy usage ⊙ Light usage (used if a DGA is detected in the ASIC)

domain name is an mAGD. Highnam et al. [18] and Yu et al. [19] used Deep Learning (DL) to detect DGAs and their models approximately took 10 ms and 50-70 ms, respectively.

Phoenix [20] is an earlier approach that detected and clustered (i.e., classified DGAs into multiple clusters) DGAs using a combination of linguistic and IP-based features. Focusing on context-less features, Pleiades [21] clustered domains that are similar based on syntactic features and those that are queried by multiple machines. Additionally, Pleiades applied a classification algorithm to map the cluster to a DGA variant. EXPLAIN [7] extracted 76 features from a domain name within tens to hundreds of microseconds on a dedicated GPU for ML and achieved 81% accuracy in classifying DGAs. More recently, Tuan et al. [22] improved DGA family classification accuracy to 99% and 85.55% on two different datasets; however, the authors did not discuss key aspects pertaining to the deployment feasibility of their approach, such as the monitoring of DNS traffic, the time required to perform feature extraction and inference, etc. In our previous work [25], we assumed that a DGA exists in the network and we only performed family classification (without detection) on 50 DGA families. Thus, all the features were sent to the control plane to classify the DGA to a family type without any intelligence running in the programmable data plane. Additionally, the implementation in the data plane required recirculation (i.e., reiterating the packet multiple times) per each subdomain, which added substantial processing overhead on the switch. Also, encrypted traffic was also not addressed at all.

Table 1 compares DGA detection and classification approaches with our work, along with the latency required for each. Essentially, our work is novel as it combines context-aware and context-less feature extraction techniques entirely in the programmable Application-Specific Integrated Circuit (ASIC) switch; thus, accurately and swiftly *detecting* DGAs at line rate. In particular, the switch takes $1-2\ \mu s$ to perform *DPI*, *extract the features*, and apply *in-switch inference* on whether a DGA exists in the network or not (i.e., run a classifier in the data plane to detect DGAs). Beyond this DGA detection, the control plane is notified with all the extracted features to further make a more accurate decision and subsequently *classify* the DGA to a family (i.e., assigning a class between 1 and 57 representing the collected DGA families). The latency corresponding to the DGA family classification is in tens to hundreds of microseconds. Such a framework is orders of magnitude faster than CPU/GPU-based machines [7] and is scalable in high DNS rates networks. Our work also presents novel results pertaining to DGA detection within *encrypted traffic*.

2.2. DNS and DPI in P4

Meta4 [26] is a framework that monitors network traffic by parsing a limited number of labels and characters of the domain name in DNS replies. P4DDPI [27] is a previous work of ours that increases the number of parsed labels in domain names for the purpose of applying security functionalities. Kaplan et al. [28] handle a vast majority of DNS packets in the data plane without sacrificing the network bandwidth or tampering with the DNS packets. Despite the novelty and effectiveness of the aforementioned DNS inspection approaches, none of them parse the whole domain name regardless of its size.

Our approach herein is the first approach that leverages P4 programmable switches to analyze network traffic at line rate to detect and classify DGA-based malware infections. In particular, (1) the P4 programmable switch can perform real-time analysis and DPI on domain names; (2) flexibly collect relevant features that are not available in traditional routers; (3) perform in-switch inference to detect DGAs; (4) involve the control plane only when a DGA exists in the network for enhanced classification of the DGA family; (5) process Tbps of network traffic, which is infeasible in general-purpose servers.

3. Background

3.1. Domain Generation Algorithm (DGA)

DGAs generate domains by typically utilizing a seed, which can consist of numeric constants, the current date/time, or even Twitter trends [29]. The seed serves as a shared secret between attackers and infected machines to compute the shared rendezvous points. By constantly generating domains, such attackers do not have to hardcode the domains in their malware binaries, thus rendering C2 takedown efforts largely ineffective and negating static domain denylisting techniques. Additionally, such constant domain generation circumvents domain reputation services since the generated domains are short-lived.

According to Plohmann et al. [30], there were 43 DGAs known by 2016, which has more than doubled to 99 domains that are known today [31]. This increase is primarily attributed to the ability of DGAs to dramatically improve the resistance of many malicious activities to takedowns. The growing diversity of these algorithms can also be observed based on how they generate domains. For instance, the `Locky` ransomware generates domains with $5 - 18$ characters while using one of 14 TLDs (e.g., "hjaoxrrwoocdsrr.uk", "firrsn.yt"). On the other hand, `Emotet`, an information-stealing malware, generates domains with fixed lengths of 16 characters while exclusively using the ".eu" domain (e.g., "fureeqnicoyejedm.eu"). While these DGAs are considered among those that generate random domains, other variants referred to as dictionary DGAs (e.g., `Gozi` and `Matsnu`) use a combination of random dictionary words to generate domain names that closely mirror the appearance of legitimate ones (e.g., "company-depend.com", "saladdoctortrainer.com"); thus making it more challenging for humans and traditional defenses to detect them. Indeed, the evolution and proliferation of DGAs have necessitated state-of-the-art ML-based DGA techniques to adjust their features accordingly.

The majority of existing related works pertaining to DGA detection and classification solely rely on the features of the domain name (see Table 1). The network characteristics of DGAs (e.g., inter-arrival time between packets, protocol counts, etc.) are not considered for several reasons, such as avoiding the exhaustion of the proposed system under high traffic loads and preserving the privacy of the users. However, the aggregation of network heuristics was proven to be highly effective in malware and botnet detection [32]. To this end, such heuristics are incorporated into the proposed approach without significantly affecting throughput.

3.2. Programmable Switch Primer

The programmable data plane enables network operators to customize the behaviors of network devices, such as routers and switches, that used to be proprietary with fixed functionality [33]. Fig. 1 shows the Protocol Independent Switch Architecture (PISA), a data plane programming model that includes the following elements: programmable parser, programmable match-action pipeline consisting of multiple stages, and programmable deparser. The programmable parser is represented as a state machine that can define the headers that need to be parsed (e.g., Ethernet, IP, DNS, or even custom headers). The programmable match-action pipeline consists of multiple match-action units (control blocks) to match against packet header fields and apply actions with supplied action data. Each unit can include one or more Match-Action Tables (MATs) that are coupled with Static Random Access Memory (SRAM) or Ternary Content Addressable Memory (TCAM) for storing lookup keys and the action data. The arithmetic operations that are encoded in a program are implemented via Arithmetic Logic Units (ALUs). Additional action logic can be implemented using stateful objects, such as registers that are stored in SRAM. Lastly, the programmable deparser defines how packet headers are reassembled when they exit the switch [34]. The high-level language for programming PISA is P4. Unlike general-purpose programming languages, P4 is domain-specific and optimized to handle Tbps of network traffic.

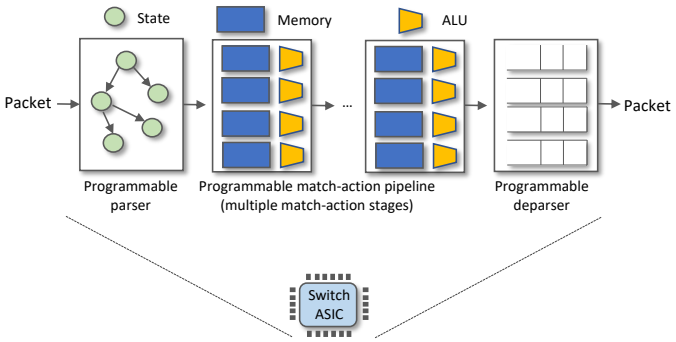

Figure 1: Components of a PISA-based data plane.

Ever since the P4 language was proposed, the networking community leveraged the flexibility and programmability offered by the language to develop in-network applications pertaining to In-band Network Telemetry (INT), load balancing, network performance, security, and so forth. For instance, Jaqen [35] designed a switch-native approach for Distributed Denial of Service (DDoS) defense within seconds while handling Tbps of traffic. Despite the exciting opportunities of pushing custom network algorithms to the data plane, the P4 switch promises Tbps throughput by limiting the complexity of pipeline stages and permitting only elementary actions (e.g., removing looping operations and allowing only simple arithmetic). Additionally, the switch has a limited number of stages. Essentially, any behaviors encoded within the same stage operate in parallel, and therefore there can be no dependencies between such behaviors (e.g., the results from one MAT are needed to perform actions of another). As a result, the amount of traditional sequential processing that can be done per a packet's pass through the program is bounded [36].

Several programmable architectures have been developed, some with extended functionalities to provide more flexibility. For instance, the Portable Switch Architecture (PSA), developed by the P4 Working Group (WG), is divided into ingress and egress pipelines, each consisting of three programmable parts: parser, multiple match-action control blocks, and de-

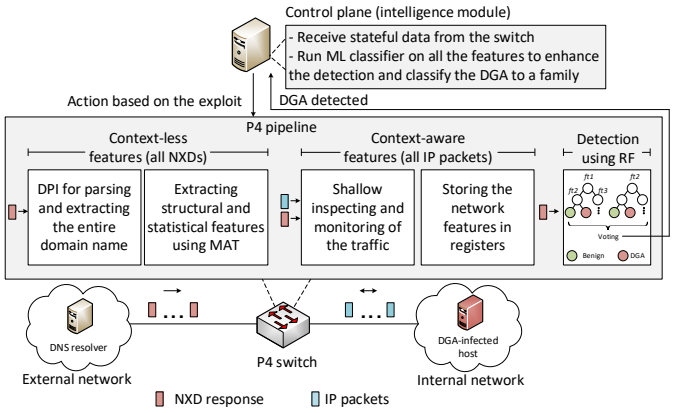

Figure 2: System overview.

parser. PSA specifies several packet processing primitives, such as packet recirculation that allows the packet to repeat ingress processing once it has already traversed the egress pipeline. Such a mechanism can be used to simulate loops in P4 and add more processing logic to the packet. The Tofino Native Architecture (TNA) is an extended version of the PSA developed by Intel [34], and the P4 program of the proposed approach is based on it.

4. Proposed System

4.1. System Overview

The primary goal of the proposed framework is to promptly detect and classify DGAs to facilitate immediate incident response and subsequent investigations. To this extent, we aspire to fingerprint the existence of a DGA on the network as soon as it begins attempting to resolve domain names (i.e., receiving NXD responses), and ultimately isolate the infected host from the network before contacting the C2 server. To achieve this aim, we offload the entirety of the feature extraction and preprocessing steps to the data plane, which take place as packets traverse the switch's pipeline. Subsequently, a subset of the context-less (associated with the received NXD) and context-aware features corresponding to the host receiving the NXD are fed to a compact in-switch inference model to detect any traces of a DGA. Consequently, detected DGAs in the data plane alert the control plane for further investigation and classification of the DGA. Indeed, the proposed approach offloads the bulk of data processing and decision-making to the data plane that can process Tbps and utilizes the control plane only when a serious threat exists.

The high-level architecture of the proposed approach is shown in Fig. 2, which can be summarized in the following steps. (1) The P4 switch monitors the communication of the internal network's hosts with the Internet and counts the number of DNS requests and unique IPs contacted by each host (context-aware features). (2) When an NXD response is received, the switch parses the whole domain name and extracts its context-less features (i.e., statistical and structural domain name features). Furthermore, the switch updates other necessary context-aware network features (e.g., inter-arrival times). (3) Once the domain name and the features are extracted, the switch takes a subset of these features and feeds them to the ML classifier running at line rate to detect if the host receiving this NXD is compromised with a DGA. (4) Next, if a DGA is detected in the data plane, the switch sends all aggregated features to the control plane via message digests. The latter passes these features to a more comprehensive ML classifier that gives more certainty on whether the host receiving the NXD response is harboring a DGA. (5) If such a DGA presence is detected, the control plane can promptly install a rule in the switch to block the communication of the offending host. Additionally, it runs a classification algorithm to assign a family to the detected DGA. To maximize the effectiveness of the proposed architecture, the programmable switch should be strategically placed to intercept DNS queries from hosts to the DNS resolver/cache. Note that such optimal placement varies from one network implementation to another; however, a P4 switch's ability to seamlessly process vast amounts of traffic allows for a broad array of deployment scenarios.

4.2. Feature Selection

This section details the features selected by the proposed approach to detect a DGA and classify it using the P4 switch. To realize the performance gains of P4-programmable switches, the software and hardware limitations (e.g., limited memory and arithmetic operations) must be adhered to; thus, special consideration was taken to arrive at informative features that can simultaneously be implemented within switch hardware in a practical manner.

4.2.1. Context-aware Features

Context-aware features make use of the network traffic behavior of DGAs while they are attempting to contact the C2 server. In addition to having the potential to promote DGA detection before the malware has made contact with the malicious actor, network features can also enhance DGA classification performance without relying heavily on complex domain name feature extraction. To this extent, the proposed framework leverages network-based features, labeled as context-aware, to enhance its capacity to fingerprint DGA artifacts in a timely manner.

Number of NXDs. Many state-of-the-art approaches focus on non-resolving DNS traffic (i.e., NXDs) since the activity of a DGA can be fingerprinted well before one of the mAGDs resolves to an IP address [37]. To empirically assess such intuition, we examined the number of NXDs of normal hosts (benign) within a campus network, in comparison to that from DGA-based malware samples collected. While we found that the number of NXDs mapped to DGAs is noticeably larger than that for normal hosts, there are some exceptions, such as DNS misconfigurations. For instance, benign hosts were repeatedly querying the same domain "*static.kpn.net.felk.cvut.cz*" and getting a large number of NXDs, which may be labeled as a DGA by current defense strategies solely based on the number of associated NXDs. Note that the behavior of repeatedly querying the same domain name successively is not common in DGAs, as

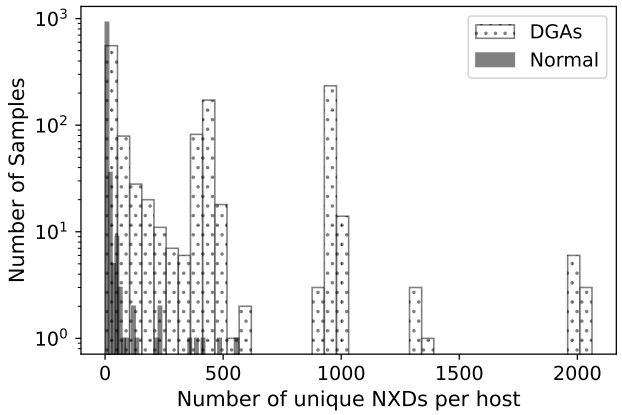

Figure 3: Histogram of the number of unique NXDs in normal users and DGA-based malware samples.

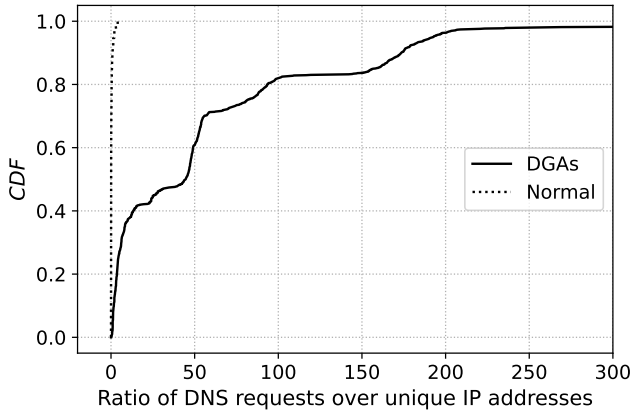

Figure 4: CDF of the ratio of DNS requests over unique IP addresses contacted in normal users and DGA-based malware samples.

they generate different domains to evade being detected. Thus, to reduce false positives, only unique NXDs should be counted for each host in the network. This technique will render DNS misconfigurations of a certain domain name to have a low number of NXDs, while DGAs would still have a high NXD count. Fig. 3 validates this claim by showing the number of unique NXDs corresponding to DGA-based malware samples and normal users in the campus network. Approximately, 90% of normal samples have less than 10 NXDs, compared to only 24% of DGA samples.

Number of DNS requests and unique IPs contacted. Grill et al. [8] used NetFlow to monitor the amount of DNS requests ($\sigma(a)$) of each host (a) in the local network and the number of unique IP addresses that it contacts ($\pi(a)$). The intuition behind these features is that users in the network need to make a DNS query before contacting any IP. Thus, normal users have a low ratio of $\sigma(a)$ over $\pi(a)$, whereas DGAs have higher ratios since, among all the DNS requests, only a few are contacted. We corroborate this claim by plotting the Cumulative Distribution Function (CDF) of the ratio of $\sigma(a)$ over $\pi(a)$ in both DGA-based malware samples and normal hosts in a campus network in Fig. 4. The samples taken in both datasets correspond to those that show at least one NXD response during the packet capture file. Additionally, since some samples could have cached DNS records (especially in the campus network), which leads to contacting IP addresses without resolving a domain, we omit the IP addresses that are seen before the first DNS request for each host.

Critiques of [8] stress that monitoring all DNS traffic is resource intensive and intrusive, thus, they focus on monitoring NXDs as it is orders of magnitude smaller than the full amount of DNS traffic [37]. Such a claim is valid on general-purpose CPUs (e.g., using NetFlow) which are not equipped to handle DNS traffic rates, however, with P4 switches, monitoring these statistics can be done at line rate, with per-packet granularity. Our work monitors the features in [8] while having the advantage of using P4 switches to solve its limitations. Another limitation of the work in [8] is that matching the requests with responses in NetFlow is not straightforward since different network probes have different error distributions of timestamps. Additionally, the authors chose five minutes as a time interval to collect the statistics in NetFlow for better performance. Such a duration is too long and could potentially allow the C2 to communicate with the botnets. In our approach, such limitations are resolved by using P4 registers to perform per-packet counting and aggregation of these features for each host in the local network. Additionally, since the inspection and aggregation are done in-network, the timestamps of the flows would not be erroneous.

Inter-arrival time between NXDomains. DGA-based botnets periodically repeat a certain cycle of DNS domain name resolution until they resolve the IP address of the C2 server [14]. Fig. 5a and Fig. 5b show the inter-arrival times of NXD responses belonging to various DGA families, as well as normal samples in a campus network (Czech Technical University (CTU)) over several days, respectively. For each sample (i.e., host) the average inter-arrival time of NXD responses is computed.

The inter-arrival time values between NXD responses in the majority of DGA families vary from milliseconds to tens of seconds. With a few DGA families (*Ctuwail*, *Ursniff*, and *Qsnatch*) having inter-arrival times ranging in 100-150 seconds, these families are still well below that of typical normal hosts in CTU traffic (hundreds and thousands of seconds). Smaller inter-arrival times of NXD responses in normal hosts (e.g., CTU-48 and CTU-52) are typically due to a misconfigured service that is repeatedly trying to resolve the same domain or a set of domains that is down. Nonetheless, repeated domains can be ignored in our system by observing only unique domain names in the data plane (using the hash of the domain name). This variation of inter-arrival times is an important metric used for both the detection and classification of DGAs. A full list of inter-arrival times of all DGA families and campus network benign samples can be found online in our repository [24].

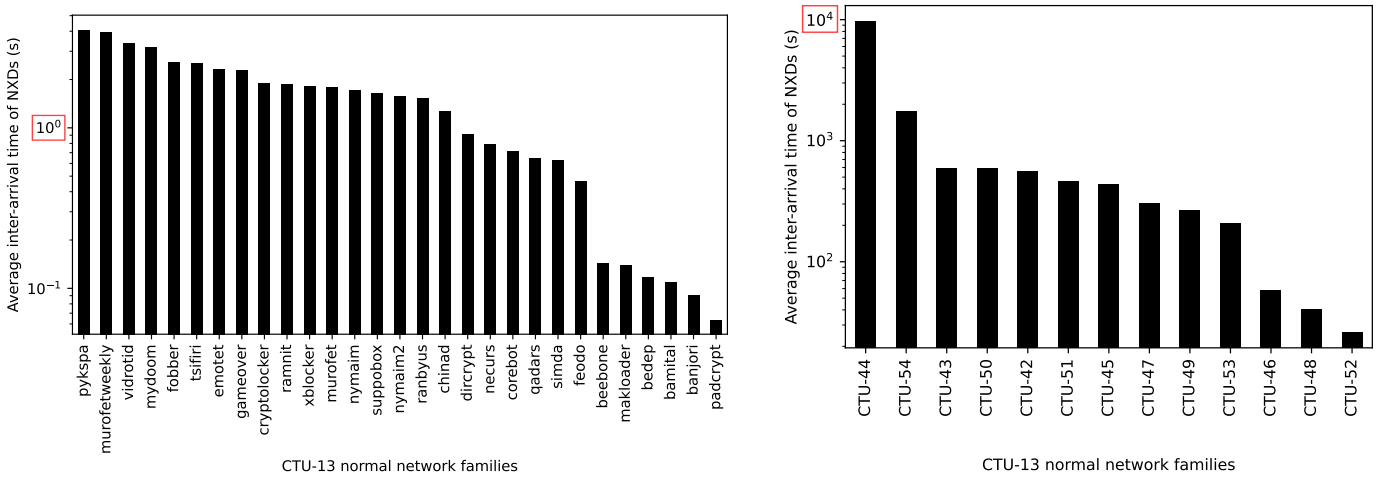

Figure 5: Inter-arrival time of (a) NXD responses in normal users and (b) DGA-based malware samples.

4.2.2. Context-less Features

Our work aims to identify the features that have been proven to be effective in the literature [38, 39] and can be implemented in P4. In particular, we utilize the randomness of a certain domain by implementing the n-gram character frequency as a statistical feature, in addition to a number of structural features of the domain name.

The 2-grams (bigrams) of a subdomain name d is a multi-set of all character sequences b, such that $b \in d$ and $|b| = 2$. For instance, the bigrams of the word "*google*" are: "*$g*", "*go*", "*oo*", "*og*", "*gl*", "*le*", and "*e$*" (the leading "*$g*" and trailing "*e$*" bigrams are sometimes omitted). We utilize the bigram frequency because the distribution of frequencies can vary between domain names belonging to DGAs and normal samples (detection), as well as they can vary among DGA families based on their algorithms (classification) [40]. Additionally, it is practical to store the distribution frequencies in the limited SRAM in the data plane. To arrive at the bigram frequency feature on the switch, the domain name D is first divided into multiple subdomains, where each subdomain d can be viewed as a separate word. Subsequently, the bigram frequency distribution is applied to each subdomain separately. Next, we calculate the randomness of a domain name D according to formula 1. The formula is adapted from [38], where the original one requires a division operation which we eliminate since it is not supported in the data plane. Despite this elimination, formula 1 can still perform well in separating normal from random domains.

$$score\ (D) = \sum_{\forall\ subdomain\ d \in D} \left(\sum_{\forall\ bigram\ b \in d} f_d^b \right) \qquad (1)$$

where f_d^b is the frequency of the bigram b in the subdomain d, read from the pre-computed match-action tables. The frequency of the bigrams utilized in this equation was obtained by processing the English dictionary and the one million most popular domains by Cisco Umbrella (Top 1M) [41] and counting the number of occurrences of each bigram within every word. Consequently, benign domains are more likely to have a high bigram frequency value than DGA-based domains since they are often meaningful [2].

Table 2: Notations used for P4-based domain name parsing.

Definition	Notation
Domain name	D
Length of the whole domain name	L
Subdomain of a domain name	d
Length of a subdomain	l
Maximum len. of a subdomain parsed per iteration	max_len_d
Maximum num. of subdomains parsed per iteration	max_nb_ds
Maximum num. of characters parsed per iteration	max_chars

The structural features of a domain name include the length of the domain name, the number of subdomains, the TLD, and the validity of the TLD. These features have been proven to be useful for detecting and distinguishing DGA families [31], and their implementation is feasible in the data plane [24].

4.3. P4 implementation

Parsing domain names in P4 switches has been discussed and implemented in [42, 26]. However, their implementations solely focus on fingerprinting a domain name in the data plane without discussing or implementing other functionalities that might not fit in the switch. In this section, we lay out the challenges associated with parsing domain names and extracting their context-less features, in particular the n-gram frequency value. Additionally, we detail our P4-based architecture that extracts the domain name and the features in a novel optimized technique. Our architecture can be easily adapted to scale with the resources of P4 programmable devices. We use the definitions in Table 2 for generalization when discussing the challenges and implementations in P4.

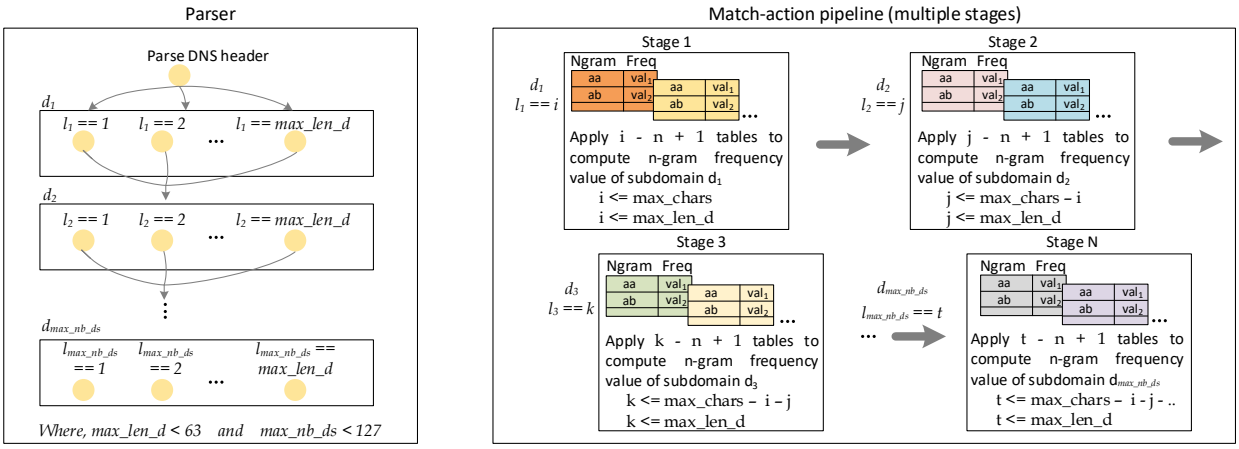

Figure 6: Challenges associated with domain name DPI and n-gram frequency computation in the parser and the match-action pipeline.

4.3.1. Challenge 1: Limited Parsing Capabilities

PISA-based switches are designed to parse fixed headers, thus, making it challenging to parse DNS domain names as they have variable-length subdomains. A P4-based solution for such a variable header length is to create a state for each possible subdomain length. However, this creates a large number of possible states and could be infeasible with complex P4 programs. According to RFC 1035 [43], the length of a subdomain is limited to 63 characters (octets). Additionally, the length of the entire domain name is limited to 255 characters (including a byte for the length of each subdomain, and a zero byte that terminates the whole domain). Each subdomain can have one character, which makes the maximum number of subdomains = $\left\lfloor \frac{255}{2} \right\rfloor = 127$ (each subdomain is separated by the dot "." character). Therefore, the total number of possibilities (states) is 63×127 (maximum number of characters in a subdomain multiplied by the maximum number of subdomains). The parser design in Fig. 6 shows the states that cover all possible subdomains and their lengths, where the maximum number of subdomains *max_nb_ds* and the maximum length of a subdomain *max_len_d* parsed can go up to 127 and 63, respectively, to cover all domain name combinations.

4.3.2. Challenge 2: Limited Number of Stages

Even if the parser allows for extracting a large number of characters [26], the limited number of stages in the match-action pipeline restricts the number of operations that can be performed on the parsed header. For instance, Kaplan et al. [42] can extract up to six subdomains ($max\_nb\_ds = 6$), each with length up to 31 characters ($max\_len\_d = 31$) in the parser. However, the authors utilize a parser counter to limit the maximum number of extracted characters to 60 ($max\_chars = 60$). Additionally, domain names with more than four subdomains cannot be fingerprinted, i.e., cannot be inferred in the data plane using MATs. Although the approach in [42] can handle a large number of domain names, it cannot be applied to our system since each extracted subdomain of length l has a total of $(l - n + 1)$ n-grams, and every n-gram requires a MAT to derive its frequency value. Moreover, each subdomain requires multiple branches (if conditions) to check for the validity of its headers before applying the MATs for computing the n-gram frequency value. Such features should be implemented all while taking into consideration other features (e.g., inter-arrival times, number of IPs and DNS requests, structural features of the domain name, etc.). The match-action pipeline of Fig. 6 shows an example of the MATs that need to be applied across multiple stages of the pipeline to calculate the n-gram frequency value. The length of each extracted subdomain d must be less than *max_chars* minus the lengths of all previously parsed subdomains. This condition should be satisfied to adhere to the total number of characters that can be extracted in the parser (*max_chars*). Dissecting the subdomains into multiple headers necessitates creating multiple branches and logic for each subdomain separately, resulting in increased resource utilization and limiting the number of subdomains that can be parsed.

In a previous work of ours [25], every pipeline pass parses a maximum of seven characters from a single subdomain, then the bigram of the extracted characters is computed. The limit on the number of characters (seven characters) and labels (one label) was chosen based on several attempts to fit the feature extraction algorithm in the switch while respecting the limited number of stages. However, extracting a single subdomain in every pipeline pass could induce an excessive number of recirculations for many domains. For instance, the domain "*a.bc.co*" would require three recirculations even though the length of the entire domain is seven (including the dot "." separators in the middle), which could have been parsed in one pipeline pass if it is treated as one subdomain (string). Therefore, a more optimized and efficient P4 implementation is needed to be devised.

4.3.3. Proposed P4 Architecture

To tackle the first challenge, we approach parsing the entirety of the domain name by utilizing packet recirculation. Subsequently, the domain name does not have to be fully parsed in one pipeline pass. This allows for limiting the number of parsed subdomains as well as the length of each subdomain.

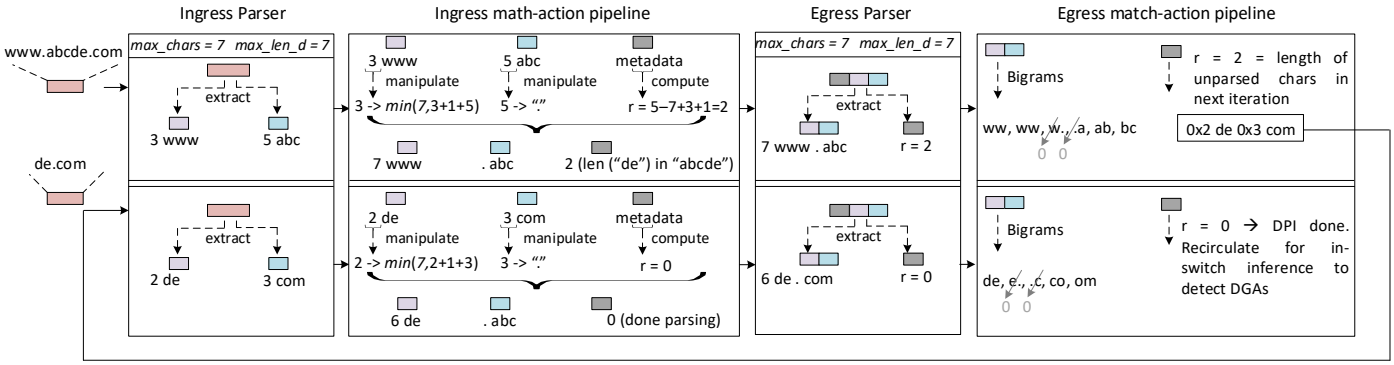

Figure 7: P4 implementation of the proposed approach showing the DPI mechanism and bigram frequency calculation of the domain name "www.abcde.com".

As for the second challenge, we aim to minimize the number of branches required for each subdomain by treating the parsed characters in all the subdomains as one string and, subsequently, computing the n-gram frequency of that string instead. For instance, if the parsed domain is "*www.abc*", we compute the frequency value of the bigrams "*ww*", "*ww*", "*w.*", "*.a*", "*ab*", "*bc*". Treating the parsed characters in all the subdomains as one string is not a trivial task in P4 since the structure of the DNS packet does not include the length of the whole domain name at once, but rather prepends each subdomain with its corresponding length. Thus, when parsing a domain name, we must first read the length of the subdomain, and then extract it. We deal with the restrictions imposed by the P4 language by utilizing the ingress and the egress parsers and match-action pipelines as follows.

Ingress parser defines a limited number of subdomains *max_nb_ds* and a limited number of characters *max_len_d* per subdomain to be parsed. Additionally, the parser utilizes a counter so that the total number of parsed characters across all subdomains does not exceed *max_chars*. Given a domain D, if the first extracted subdomain d_1 has a length l_1, such that ($l_1 >= max\_chars$), then the ingress parser will extract *max_chars* characters of d_1 and move to the ingress match-action pipeline, where d_1 is partially parsed. Otherwise ($l_1 < max\_chars$), it will extract the entirety of d_1 and proceed to the next subdomain d_2 with length l_2 and try to parse it. If ($l_1 + l_2 < max\_chars$), then it will extract d_2 and proceed to d_3 similarly. Otherwise ($l_1 + l_2 > max\_chars$), then it will only extract ($max\_chars - l_1$) characters from d_2, and thus, d_2 will be partially parsed. The extracted subdomains are stored in separate headers in the ingress parser (variable-length subdomains cannot be extracted in one header in P4), and hence, cannot be treated as one string in the ingress match-action pipeline. The maximum length of the subdomain *max_len_d* is correlated with the maximum number of characters parsed *max_chars*. For instance, the first subdomain can have ($max\_len\_d = max\_chars$). The second subdomain can have ($max\_len\_d = max\_chars - 1$) since there will be one byte representing the length of the second subdomain. The third subdomain can have ($max\_len\_d = max\_chars - 2$) and so forth.

Ingress match-action pipeline computes the number of characters extracted in the ingress parser across all the subdomains so that these characters can be extracted as one string in the egress parser. The length of the extracted characters can be characterized by formula 2:

$$N = min(max\_chars, l_1 + 1 + l_2 + 1 + ... + l_i) \quad (2)$$

where $l_1..l_i$ correspond to the length of the extracted subdomains $d_1..d_i$. Adding a value of one for each additional subdomain parsed corresponds to the length of the subdomain (the length of the subdomain is one byte, so it is represented as one character).

A subdomain d_j with length l_j can have a length larger than what the switch can pars This necessitates packet recirculation to be utilized so that the domain is treated over multiple pipeline passes. The subdomain d_j would be partially parsed in the current pipeline pass, since fully parsing it would exceed *max_chars*. As a result, the ingress match-action needs to compute the length of the remaining characters in a partially parsed subdomain so that the rest of the domain name is parsed in the next recirculation. The length of the remaining characters within this subdomain is computed based on formula 3:

$$r = l_j - max\_chars + l_1 + 1 + l_2 + 1 + ... + l_{j-1} + 1 \quad (3)$$

Where $l_1...l_{j-1}$ correspond to the lengths of the subdomains $d_1...d_{j-1}$ fully extracted before d_j. The length of the remaining unparsed characters r in d_j is equal to the length of the subdomain l_j subtracted to the number of characters that the switch can currently parse ($max\_chars - l_1 - 1 - l_2 - 1 - ... - l_{j-1}$).

Since multiple subdomains will be treated as one string in the egress, this will introduce outlier n-grams when computing the randomness (e.g., the bigrams "*w.*" and "*.a*" in "*www.abc*"). These outlier n-grams are the ones that overlap between subdomains and contain the dot separator, which is in fact one byte representing the length of the subdomain succeeding it. For these outlier n-grams to have a frequency value of zero, each byte representing the length of an extracted subdomain (after the first subdomain) is replaced with a special character that does not appear in an English word. The chosen special character is the dot character ".", represented in hexadecimal as "0x2E".

Egress parser receives the number of characters that are extracted in the ingress and uses it to extract the characters as one string.

Egress match-action pipeline computes the frequency value of the n-grams obtained from the parsed subdomain (string) using a series of match-action tables, where the n-gram is the table key and its value is computed offline and assigned using the action data in P4. The performed computations, along with the needed metadata are recirculated with the packet for additional DPI.

Fig. 7 shows an example of how the switch performs DPI on the domain "*www.abcde.com*" (represented as *(0x3) www (0x5) abcde (0x3) com*) and extracts its n-gram frequency value. The bigram is selected as it has been proven to be effective [2] and requires fewer resources in the switch. To conform to the hardware limitations, *max_chars* is set to seven bytes (characters) and *max_nb_ds* to three (subdomains). However, with future P4 switches that are expected to have more resources, our code is modular and could be easily adapted to parse more characters (i.e., by scaling these variables). The ingress parser fully extracts the first subdomain "*www*" and partially extracts the second subdomain (i.e., it only extracts "*abc*" from "*abcde*"), since it cannot extract the whole subdomain ($3+1+5 > 7$). The ingress match-action pipeline computes the number of characters extracted in the parser ("*www.abc*") to be seven so that the egress parser knows exactly how many characters to extract as one string. Additionally, it computes the number of characters remaining in the partially parsed subdomain (i.e., $r = 2$ representing "*de*" in "*abcde*"). This number is crucial when recirculating the packet since the domain will start from "*de*", and the parser needs to know the length of the subdomain before parsing it. Lastly, it swaps the length of the first subdomain with seven, and that of the second domain with the special character dot ".". The egress parser will extract seven characters per the information provided previously, then, the egress match-action pipeline will read the string "*www.abc*" as one string and computes the frequency of the bigrams "*ww*", "*ww*", "*w.*", "*.a*", "*ab*", and "*bc*". The frequency value of any bigram containing the special character dot "." is always zero.

The extracted characters will be removed while preserving the computed features (e.g., bigram frequency value and the number of characters extracted so far). Then, the packet gets recirculated, where the domain name becomes "*de.com*". Once the whole domain is extracted, the context-less and context-aware features are fed to a Random Forest (RF) classifier running in the ingress to check if the host receiving the NXD is infected with DGA-based malware. The implementation of the RF classifier follows a state-of-the-art optimized approach by Flowrest [44] so that it can fit in the switch while also handling DPI and feature extraction.

5. Evaluation

5.1. Dataset

Due to the increased interest in DGA detection, a plethora of mAGD datasets exist (i.e., domain names generated by DGAs)

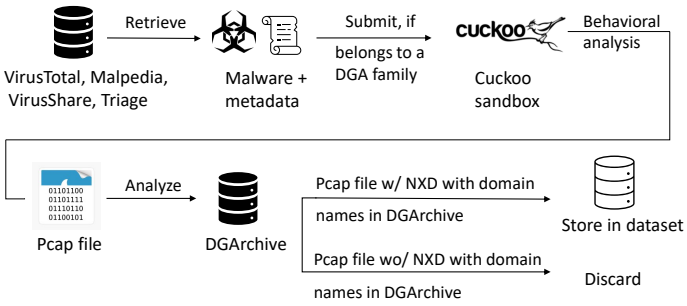

Figure 8: Data collection and curation process of DGA samples.

[31, 45]. These datasets only comprise domain names and therefore are not sufficient for techniques such as ours that leverage network-related features. Moreover, due to privacy concerns, finding a publicly available dataset of network traffic (comprising the required features) of normal hosts and those infiltrated by DGAs is not trivial. Thus, data collection, filtering, and curation were performed to construct a representative dataset of both DGA and normal samples.

DGA samples. Despite the variety of mAGD datasets that have been utilized in the literature, there is no public repository containing a large number of labeled DGA-based malware. Fig. 8 shows the data collection technique used to obtain the DGA samples. To this extent, we crawled and retrieved hundreds of gigabytes of malware samples from notable cyber security services and websites including VirusTotal (2017 until 2021) [46], VirusShare [47], Malpedia [48], and Triage [49]. The hashes of the obtained samples were submitted to VirusTotal to retrieve their metadata information and determine if they were previously reported to have DGA behavior. Subsequently, each sample was instrumented in an isolated environment (using Cuckoo sandbox [50]) to capture its network traffic behavior, i.e., the Packet Capture (Pcap), file for 10 to 30 minutes. The execution time is selected to scale for the number of samples in our dataset, while still being able to capture the behavior of the malware [51]. To further validate that the collected samples demonstrate DGA behavior, only Pcap files that have NXD responses registered in DGArchive [31] are considered. DGArchive contains a database of well-known DGAs and their generated samples. The resulting dataset includes about 1300 samples containing 57 DGA families. We kindly refer interested readers to our dataset which we make publicly available for additional details pertaining to the collected DGA families [24].

CTU-13 dataset. This dataset contains network traffic captured at CTU university [52] and also has a large capture of real botnet traffic mixed in with the normal and background traffic. Since we require only the normal traffic to model benign behaviors, all IPs corresponding to the botnets were removed from the Pcap files. For privacy reasons, the Pcap files containing network traffic of normal users were truncated to include up until the transport layer only. Unfortunately, such a dataset would not be useful in our setup, since we are parsing DNS packets; however, DNS log files are provided for each Pcap file, which contain all the DNS traffic (via Zeek [53]) along with the times-

Table 3: System Accuracy reported by various classifiers amid varying the number of NXD responses.

Approach		Model	Accuracy with respect to the number of NXD responses received												
			2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**20**	**30**	**40**	**50**
Detection	P4-DGAD	RF	0.903	0.908	0.918	0.927	0.933	0.933	0.935	0.942	0.944	0.960	0.971	0.973	0.977
	DGAD	RF	0.991	0.994	0.994	0.995	0.996	0.997	0.997	0.996	0.997	0.998	0.998	0.998	0.999
		SVM	0.968	0.963	0.961	0.963	0.972	0.964	0.966	0.960	0.969	0.964	0.972	0.976	0.979
		MLP	0.991	0.994	0.993	0.99	0.994	0.995	0.994	0.996	0.996	0.996	0.997	0.997	0.998
		GNB	0.826	0.896	0.94	0.943	0.955	0.960	0.957	0.955	0.955	0.955	0.960	0.957	0.957
		LR	0.956	0.967	0.969	0.968	0.967	0.972	0.967	0.972	0.970	0.977	0.977	0.971	0.973
Classification	DGAMC	RF	0.894	0.900	0.921	0.927	0.934	0.938	0.945	0.946	0.951	0.965	0.972	0.976	0.979
		SVM	0.836	0.866	0.863	0.875	0.874	0.890	0.881	0.896	0.892	0.880	0.901	0.906	0.915
		MLP	0.866	0.877	0.888	0.917	0.905	0.904	0.921	0.927	0.933	0.943	0.952	0.962	0.961
		GNB	0.769	0.716	0.696	0.611	0.666	0.596	0.630	0.640	0.641	0.672	0.709	0.722	0.722
		LR	0.799	0.806	0.818	0.818	0.828	0.818	0.840	0.834	0.836	0.800	0.822	0.841	0.849

tamps when the DNS queries were made in the Pcap file. In turn, we were able to associate the timestamps in the Pcap file with those in the log files and reconstruct the Pcap file containing the DNS headers.

While the two datasets (Pcap files of DGAs from the Cuckoo sandbox and Pcap files corresponding to normal hosts in CTU) come from different environments, this does not affect the conducted experiments and evaluations of the proposed approach. A normal host in a campus network that gets infected with a DGA-based malware is going to change its behavior to become similar to one of the DGAs observed in the Cuckoo sandbox (i.e., start querying domains with NXD responses). Subsequently, the proposed system will get activated with the returned NXD responses and flag the infected host in the network.

Dataset format. After the collection phase, the dataset is in Pcap file format and needs preparation for training and testing. Each host is considered a sample identified by its IP address. Our approach is based on studying the behavior of samples that receive NXD responses. Thus, each sample is characterized by a vector of vectors as follows:

$$\begin{aligned} x_i = [&[ft^1_{NXD_1}, \quad ft^2_{NXD_1}, ..., \quad ft^m_{NXD_1}] \\ &[ft^1_{NXD_2}, \quad ft^2_{NXD_2}, ..., \quad ft^m_{NXD_2}] \\ &... \\ &[ft^1_{NXD_n}, \quad ft^2_{NXD_n}, ..., \quad ft^m_{NXD_n}]] \end{aligned}$$

Where x_i is a sample in the dataset, the first subvector $[ft^1_{NXD_1}, ft^2_{NXD_1}, .., ft^m_{NXD_1}]$ contains all the features from 1 to m that we selected, and have been seen at the first NXD response (i.e., NXD_1). Each subsequent subvector contains the features collected at the current NXD response, until the last NXD response seen by the host (i.e., NXD_n). Such a format facilitates the training and inference of samples with respect to the number of NXD responses.

In total, 990 normal samples and 1,247 DGA-based samples exist in the dataset. The final dataset is stored in a Comma-Separated Value (CSV) file, and it is available online along with the scripts to retrieve and preprocess it [24].

5.2. Experimental Setup

The collected datasets comprising DGA samples and normal campus traffic users were used for offline training and testing on a general-purpose CPU. In particular, 80% of the dataset was used for training and 20% for testing. Since a large number of normal users would not necessarily receive an NXD response, only those getting at least two NXDs are included in the dataset. Furthermore, the number of DGA and normal user dataset records are balanced before they are fed to the classifier. Grid search was then applied to select the best hyperparameters for the model. Lastly, Cross Validation (CV) was used with five folds in order to avoid overfitting the model.

The trained models can be divided into three based on their placement in the proposed system. (1) A model for detecting DGAs that can fit in the P4 switch, referred to as P4 DGA Detection (P4-DGAD). (2) A model for detecting DGAs with more resources and enhanced accuracy that runs in the control plane, referred to as DGA Detection (DGAD). (3) A model for classifying the family of the detected DGA, referred to as DGA Multiclass Classification (DGAMC). The ML models in P4-DGAD, DGAD, and DGAMC only operate when an NXD is received, thus, the evaluation results are based on features collected at the i^{th} NXD received, where $2 < i < 50$. The results of the detection modules (P4-DGAD and DGAD) is a binary decision outputting one if the sample is found to be a DGA, and zero if benign. However, the output of the classification module (DGAMC) is a number between $[1, 57]$ representing the DGA family. In all experiments, the accuracy metric is considered the primary criterion to assess the classifier's performance. The accuracy measures the ratio of correctly predicted instances (true positives and true negatives) to the total number of instances in the dataset (true positives, true negatives, false positives, and false negatives).

5.3. System's Accuracy

Table 3 reports the accuracy of P4-DGAD, DGAD, and DGAMC with different ML classifiers. The choice for the P4-DGAD classifier depends on the software and hardware limitations of the P4 switch. Hence, we adopt an RF classifier in the data plane since their deployment and efficacy were proven in several prior studies [54, 32, 44]. The P4-DGAD model was

Feature		Importance
Context-aware	Number of unique IPs	0.365
	DNS requests	0.283
	NXD inter-arrival time	0.351
Context-less	Bigram score	0.373
	Domain name length	0.144
	Number of subdomains	0.150
	Has a valid TLD	0.044
	TLD	0.286

Table 4: Importance analysis of context-aware and context-less features.

trained on four features, namely, the number of unique IP addresses contacted, the number of DNS requests, the inter-arrival time between NXDs, and the bigram frequency value of the domain.

The DGAD and DGAMC models are implemented in the software, allowing for more flexibility with the choice of the classifier selected, as well as the features fed to the classifiers. The reported models in DGAD and DGAMC are RF, Support Vector Machine (SVM), Multi-layer Perceptron (MLP), Gaussian Naive Bayes (GNB), and Logistic Regression (LR) classifiers. Moreover, these models are trained on all the features discussed in Section 4.2.

The accuracy of P4-DGAD starts at approximately 90% with the second NXD received and steadily increases to 94% at the 10^{th} NXD until it reaches 97% at the 50^{th} NXD. This model runs in the hardware (P4 switch) at line speed, and reports to the software (control plane), only if a DGA is detected. This significantly reduces the overhead on the control plane from the true negatives (i.e., benign hosts receiving NXDs). Such accuracies are high enough to sense if there is any DGA-related behavior at an early stage of the malware. As for DGAD, RF outperforms all other reported models with a 99.1% at the second NXD, and increasing to 99.7% and 99.9% at the 10th and 50^{th} NXD responses, respectively. This model is used to increase the confidence on whether a DGA exists in the network or not. For instance, if P4-DGAD reports a false positive (i.e., a normal host labeled DGA) at 3^{nd} NXD with accuracy 90%, DGAD would receive the features and make a more informed decision with accuracy up to 99.4%.

Similarly, RF performs best in DGAMC with an accuracy of 89.4% at the second NXD and increases to 95.1% and 97.7% at the 10^{th} and 50^{th} NXD responses, respectively. The MLP classifier performs closely to RF, however, its training and testing times are significantly higher. The classification is performed on 57 DGA families, which is the most representative number of families among all state-of-the-art DGA classification approaches utilizing context-aware features. The list of DGA family names is reported in our GitHub repository, along with the reported ML models and the evaluation results.

5.4. Feature Importance Analysis

To study the importance of features, we differentiate between the context-aware features which are time-dependent, and the context-less features which are time-independent. The context-aware features are primarily network features (e.g., number of unique IPs contacted) that change with time. For instance, at the first NXD response received, the number of unique IPs contacted by a sample is less than or equal to that at the second NXD response. The context-aware feature importance values presented in Table 4 correspond to RF classifiers trained per NXD response, and the values are averaged across all NXD responses (from 2 to 50).

On the other hand, the context-less features depend on the characteristics of the domain names, rather than the time they are queried. Consequently, we study their importance separately using an RF classifier fed with all collected NXD responses regardless of the time they are queried. The bigram score of the domain name has the highest importance value among other context-less features.

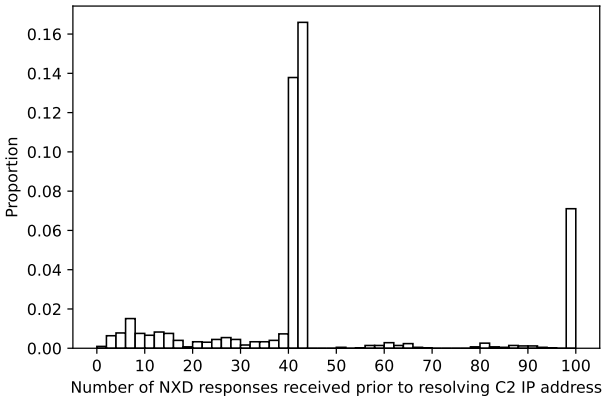

Figure 9: Histogram showing the percentages of C2 servers getting resolved before the i-th NXD.

5.5. Communication with the C2 Server

The proposed system aims to detect hosts infected with DGA-based malware and halt their operations prior to communicating with the C2 server. Consequently, we analyze the communication between the DGA and the C2 server with respect to the number of NXD responses received. The network traffic of each sample is analyzed against the Suricata [55] Intrusion Detection System (IDS) to infer if a communication occurs with the C2 server, and the time it occurs (i.e., the time the IP address of the C2 server was resolved by a DNS query in the Pcap file). Subsequently, we check how many NXD responses were received prior to resolving the C2 server IP address and plot the histogram in Fig. 9. Approximately, only 7.5% of the seen C2 servers were resolved before 10 NXDs, and our classifier was able to detect all of them as DGAs in the data plane before the 10^{th} NXD. Additionally, a significant percentage of DGA-based malware resolves the IP address of the C2 server after the 40^{th} NXD, which is a sufficient time to detect the DGA based on the accuracy results in Table 3.

Approach		Accuracy	Training time
NetFlow [8]		93%	5 mins
SDN [15]	iForest	51%	6 secs
	K-means	<50%	
FANCI [2]	RF	98.8%	Hours (grid search)
EXPLAIN [7]	RF RFE	90.9%	307 secs*
	OVR RFE	91.6%	2533 secs*
	OVR union	92.1%	7,036 secs*
	OVR Union Spearman	92.2%	3,036 secs*

* Training time is based on the reported metrics in the paper.

Table 5: Accuracy and training time of the state-of-the-art models.

5.6. Comparison with State-of-the-Art

In the context of DGA detection and classification, the public datasets (e.g., DGArchive [31]) mainly include domain names only, which is not sufficient to evaluate approaches that use context-aware feature (i.e., network traffic features). The private datasets are not made publicly available online due to privacy concerns. Thus, we use our dataset containing context-less and context-aware features to evaluate state-of-the-art work on DGA detection and classification. We choose two related work [8, 15] that use context-aware features, and two related work [2, 7] that use context-less features. All implementations are available online and were tested on a machine with 64 GB Random Access Memory (RAM) and a 2.9 Gigahertz (GHz) processor with 8 cores [24]. Python is the main programming language used.

Grill et al. [8] detect DGAs using NetFlow, a Cisco technology that provides statistics on packets flowing through the router. In particular, they monitor the ratio of DNS requests over unique IPs contacted and assume that this ratio follows a normal distribution in normal traffic. Thus, anomaly values are obtained using the following fuzzy function in formula 4:

$$f(x) = \begin{cases} 0 & \text{if } x \leq \mu + t_1 \cdot \sigma \\ \frac{x-(\mu+t_1\cdot\sigma)}{t_2-t_1} \cdot \sigma & \text{if } \mu + t_1 \cdot \sigma < x < \mu + t_2 \cdot \sigma \\ 1 & \text{if } x \geq \mu + t_2 \cdot \sigma \end{cases} \quad (4)$$

Where x is the value of the ratio for a given sample, μ and σ are the mean and standard deviation values of the distribution, and t_1 and t_2 are thresholds. We reimplement the approach [8] and obtain the μ and σ values of our normal dataset. We use the threshold values $t_1 = 2$ and $t_2 = 4$ and monitor traffic for 5 minutes, as suggested by the authors. Then, we test the fuzzy function on DGA samples and report the accuracy in Table 5. Even though the fuzzy function scores 93% when data is collected for 5 minutes, it is also worth mentioning that the score is poor (accuracy = 56%) when traffic is collected after one minute, which deems it unsuitable for fast detection approach. Additionally, the approach requires constant monitoring and evaluation of all hosts in the network, in contrast to our approach which only evaluates hosts receiving NXD responses.

Ahmed et al. [15] monitor the following attributes per flow to detect DGAs in SDN/OpenFlow environment. (1) Flow volume in bytes; (2) flow duration; (3) number of packets; (4) average packet size. These attributes are computed for incoming and outgoing directions of each flow. The authors rely on a real-time database server that receives all DNS responses and checks if the domain name exists in DGArchive [31]. However, such an assumption requires a prior knowledge of all DGA domains and a powerful server to process all DNS responses in real-time. We do not take this assumption into consideration since our dataset includes normal hosts that received NXD responses not found in DGAarchive. Subsequently, we evaluate the effectiveness of these attributes in identifying normal and DGA behavior. We reimplement and train the Isolation Forest (iForest) [56] and K-means algorithms [57] on our dataset with the suggested parameter values by the authors and report the results in Table 5 (SDN/OpenFlow). Both classifiers perform very poorly, indicating that the proposed attributes cannot be used solely to detect DGAs in our dataset.

As for context-less approaches, we evaluate FANCI [2] and EXPLAIN [7] that perform detection and classification of DGAs, respectively. The source code of FANCI is publicly available online, thus, we use it to train its RF classifier on our dataset and detect DGAs. FANCI performs grid search to find the best parameters of a model, a process that takes hours on our machine. Similarly, EXPLAIN's source code, along with the trained models are available online, and we utilize them to test the models on our dataset for DGA detection and classification (57 DGA families). In particular, EXPLAIN provides four trained model based on chosen hyperparameters and feature set.

FANCI and EXPLAIN achieve high accuracy detection and classification results since they use tens of features related to the domain name. However, a drawback of such an approach is that a general-purpose CPU/GPU is required to extract the features. Also, if the approach is to be deployed in real-time, the servers must be able to withstand DNS traffic loads. To this end, we measure the time it takes to process an NXD in our switch (P4 Switch), which includes DPI, feature extraction, and the in-switch inference. Fig. 10 shows the CDF of the feature extraction time in microseconds. Our proposed approach incurs feature extraction latency that is orders of magnitude lower than FANCI and EXPLAIN. Additionally, the optimization that was performed in our proposed approach over the previous one (Section 4.3) allowed us to go deeper into the packet (per recirculation), fit an RF classifier for in-switch inference, and ultimately enhance the detection speed.

We note that even though the authors of EXPLAIN measure its source code on a dedicated GPU and receive an average feature extraction time of 276 μs on their best-performing model, it is still orders of magnitude lower than that received by our switch.

5.7. Preliminary Results on Encrypted Traffic

By nature, DNS leaks almost all user behavior to any party that is eavesdropping on the network. Solutions like DoT and DNS over HTTPS (DoH) provide confidentiality to the transported DNS traffic and are being increasingly supported by re-

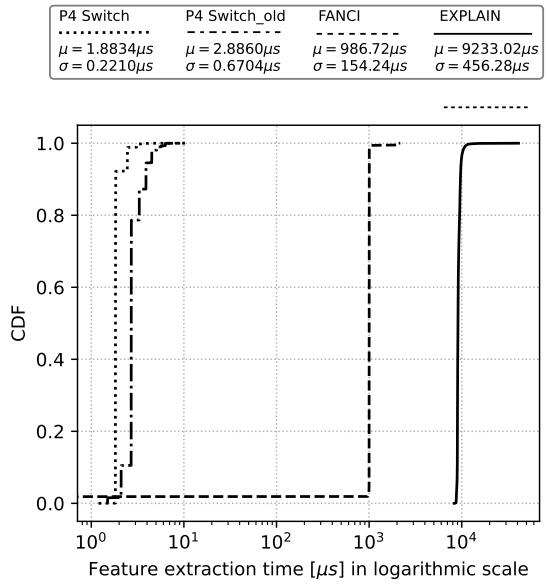

Figure 10: CDF of the latency incurred while extracting the features in P4-DGAD and state-of-the-art approaches.

solvers, browsers, and operating systems [58]. In DoH, DNS queries are sent via the HTTP or HTTP/2 protocols instead of being sent directly over UDP; thus, DoH traffic looks like other HTTPS traffic (all traffic is sent via port 443). On the contrary, DoT adds TLS encryption on top of UDP and uses the dedicated port 853 for sending DNS queries; thus, anyone sniffing on the network can infer that this is DNS traffic. Subsequently, the focus of our current evaluation is on DoT since it is practical to infer the DNS traffic using the P4 switch, where DoH is left for future work.

5.7.1. Dataset collection

Two datasets were constructed to study the characteristics of domain names belonging to DGAs and normal samples. The first dataset contains domain names requested by real DGA-based malware where their response is non-existent. The second dataset contains domain names from the one million most popular domains by Cisco Umbrella (Top 1M) [41]. DoT traffic can be padded to reduce the information that can be extracted from the query length. For instance, RFC 8467 recommends padding all requests to a multiple of 128 bytes, while all responses are padded to a multiple of 468 bytes. In the collected dataset, the domain names are queried using DoT via two utilities: (1) DNS python module `dnspython` [59] for unpadded traffic, and (2) `kdig` [60] for padded traffic. The DNS resolver used in both utilities is Google (8.8.8.8), and the network traffic generated from querying and resolving each domain is saved in a Pcap file. Each Pcap file contains the TCP handshake, the TLS handshake (for exchanging the keys), and the DNS request and response of the queried domain (in encrypted format). Even though the DNS request and response packets are encrypted in a TLS packet, they can be differentiated by observing the direction of the flow (i.e., requests have destination port 853 and responses have source port 853). Indeed, when discussing encrypted DNS traffic, the requests and responses include all DNS types (type A for resolving IPv4 address, type AAAA for resolving IPv6, type MX for mail services, etc.).

5.7.2. DoT-tailored Feature Selection

In the conducted experiments, DoT traffic is assumed to be resolved by a third-party DNS resolver, thus, it cannot be decrypted. Consequently, traffic features that can unveil useful information for detecting DGA-based malware without the need to inspect the payload are considered. Such features are represented by (1) DNS request packet size, (2) DNS response packet size, (3) inter-arrival time between a DNS request and its response, and (4) inter-arrival time between DNS requests.

In unpadded DoT traffic (Fig. 11.a), the size of the DNS request packet changes uniformly with the size of the domain name, regardless of the TLD queried. As for the DNS replies, the size of the packet varies if the response is NXD or Existent domain (EXD). In NXD responses, the size of the packet changes based on the size of the queried name as well as the TLD queried since NXD replies might return additional information about the TLD. In EXD responses, the size of the packet varies based on the size of the queried name as well as the answer returned (i.e., the resolved IP(s) addresses returned). The size of DNS packets in unpadded traffic could be relevant for some DGA families that generate domain names with specific length ranges and TLDs. For instance, *Bedep* DGA family generates domain names of lengths between 12 and 18 and ".com" TLD, thus, the size of the DNS requests will always range between 124 and 130 bytes, and their corresponding NXD replies will always range between 741 and 747 bytes. In padded traffic, the size of the DNS packet cannot be used as a strong indicator for the domain name size (Fig. 11.b) as the packet is always padded by zeros.

As for the inter-arrival time between a DNS request and its corresponding reply, NXD replies take a longer time to resolve in both padded and unpadded DoT traffic compared to EXDs (Figs. 11.c and 11.d). This could be due to caching frequent domains in local DNS servers. For example, a domain belonging to the Top 1M most used domains is likely to be cached by the DNS resolver of the ISP, thus, the latter will immediately return a response and bypass several steps in the DNS query process [61]. Conversely, NXD replies take longer to be resolved since once the request arrives at the DNS resolver, its whole database should be searched, and when no reply is found in the local DNS resolver, then recursive resolution occurs to query authoritative DNS servers; thus, leading to increased delay.

The inter-arrival time between DNS requests is a significant indicator and it is dependent on the sample behavior rather than the encryption used (padded or unpadded). Fig. 12 shows the average inter-arrival times of DNS requests in normal users within CTU campus over several days, as well as that of various DGA families. DNS requests are more spaced out over tens and hundreds of seconds between benign samples, whereas it is more compact in DGA-based malware samples.

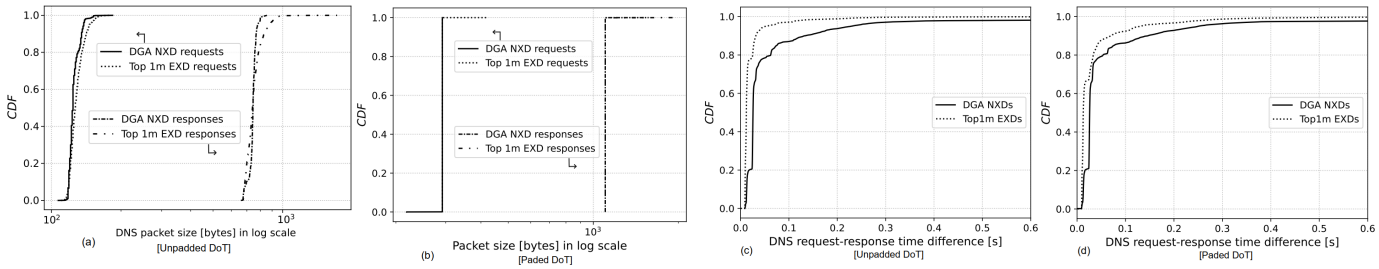

Figure 11: CDF graphs of the DNS request and response packet sizes and their inter-arrival time.

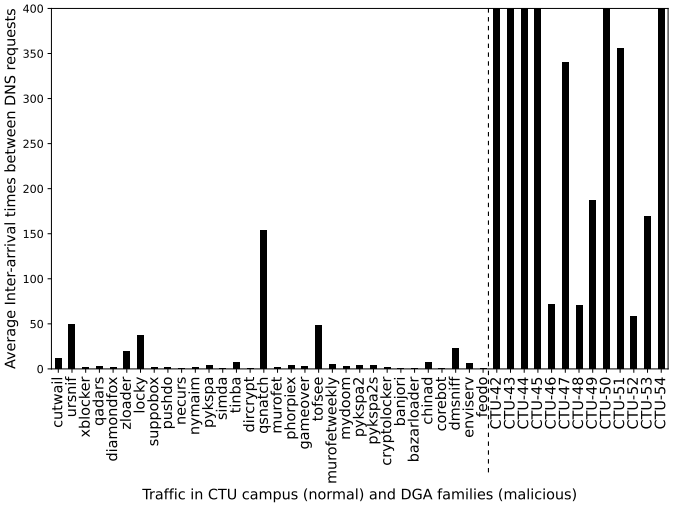

Figure 12: Average inter-arrival times between DNS request in CTU and DGA samples.

5.7.3. Accuracy of Detecting DGAs using DoT

To evaluate the effectiveness of the DoT-tailored features, the existing dataset described in Section 5.1 containing normal and DGA traffic is modified to simulate that the hosts are using DoT instead of raw DNS. Thus, every DNS request and response packet is replaced with its equivalent counterpart using `dnspython` (for unpadded traffic) or `kdig` (for padded traffic). Multiple RF classifiers are trained with the average and the variance of the DoT-tailored features at different intervals. The intervals are based on the number of DNS requests received thus far for each sample and can be differentiated based on the TCP port number. Table 6 reports the accuracy of the RF classifiers starting from the second DNS request till the 40^{th} one. Unpadded traffic starts with 94% accuracy which is slightly higher than padded traffic (92%) since it benefits from the sizing information of the packets. The accuracy reaches 95% and 96% with unpadded and padded DoT traffic, respectively, however, the increase in the accuracy is not always uniform (e.g., 93% then 92% at the 5^{th} and the 10^{th} DNS requests, respectively). This is due to some benign samples having a burst of DNS queries within a short time window, or DGA samples being stealthy by spacing out the DNS queries they generate.

Table 6: Reported accuracy of the RF classifier when evaluated on CTU (benign) and DGA (malicious) samples amid varying the number of DNS request received.

DoT Traffic	**Accuracy with respect to the number of DNS request**						
	2	**5**	**10**	**15**	**20**	**30**	**40**
Unpadded	0.94	0.94	0.94	0.95	0.95	0.96	0.96
Padded	0.92	0.93	0.92	0.92	0.93	0.94	0.95

6. Discussion

The proposed approach aims to fingerprint the presence of DGAs, classify them based on the malware family they belong to, and subsequently block them from reaching the C2 server. This is achieved using heuristics of network features coupled with statistical and structural features of the domain name, which are all collected in the P4 switch. The benefit of implementing such a system becomes apparent in congested networks, where the processing capacity of CPUs may be insufficient to examine each DNS packet thoroughly [2]. Additionally, the delay incurred by waiting to gather network statistics through conventional network monitoring tools may be unacceptable [8]. Moreover, the switch-based feature extraction and inference take a few microseconds, without the need for sharing domain names (i.e., only the extracted features of the domain name need to be shared). As a result, the privacy of individuals within an organization is preserved [62].

Despite these benefits, there are a few limitations tied to the proposed scheme and P4 switches in general. Due to the limited resources in the switch, the number of users (per IP address) that can be monitored in our program is approximately 110,000.

Furthermore, the proposed approach saves the context-less and context-aware features of the users in the internal network in stateful registers, which are limited in memory. Hence, it is necessary to remove stale entries that point out to inactive users (e.g., a user received an NXD reply and remained inactive for a few hours). However, an attacker who has knowledge of such implemented features can craft the malware in a way that bypasses the proposed detection approach. For instance, an attacker can increase the inter-arrival time among the queried DNS requests until the resources of the switch are reset.

Additionally, given that the accuracy of the model increases with the number of NXDs, an attacker could choose to register a domain (i.e., this domain will have the IP address of

the C2 server) that is queried at the beginning (within the first few NXDs) to reduce the probability of being detected. As for encrypted DNS traffic, the features experimented with require statistical computations and memory (average and variance), which are not feasible in the P4 switch and require external processing.

7. Conclusion and Future Work

The proliferation of attack vectors that DGA currently supports, coupled with the magnitude and scale of attacks they facilitate, necessitates that their presence coincides with immediate incident response and further investigation. To this extent, this paper presents a novel network framework for detecting and classifying DGA artifacts on an Intel Tofino hardware switch and promptly blocking them before the compromised hosts are ordered to take part in malicious actions. The proposed scheme leverages programmable switches for privacy-preserved microsecond-scale feature extraction and in-switch inference of DGAs. In particular, it extracts network heuristics for each host on the internal network, as well as statistical and structural features of entire NXD domain names obtained via DPI. Consequently, an ML module running on the switch is utilized to detect the presence of DGAs at line rate. Once a DGA is detected, the control plane is activated, where it receives the pertinent features of the DGA-infected host to perform a more enhanced and thorough analysis. The proposed approach is able to detect DGAs on real network traces with an accuracy up to 97% accuracy in the data plane, and 99% in the control plane. Additionally, the detected DGA can be further classified into its corresponding malware family with an accuracy of up to 97%. Novel experiments on DoT with features tailored to encrypted traffic output high accuracy values as well (94%-96%). Future work in this area includes investigating other network and domain name linguistic features to achieve higher accuracy in a short period of time. Additionally, a concrete framework for DoT, DoH, and other encrypted DNS traffic can be devised in a way that utilizes SmartNICs via Data Plane Development Kit (DPDK) and general-purpose CPUs via Extended Berkeley Packet Filter (eBPF). This would allow for more flexibility in implementing a security solution while maintaining the requirements of high-speed networks.

Acknowledgment

This material is based upon work supported by the National Science Foundation (NSF) under grant number 2118311, and the Office of Naval Research (ONR) award number N00014-23-1-2245.

References

[1] Steve Morgan, Cybercrime To Cost The World $10.5 Trillion Annually By 2025, [Online]. Available: `https://tinyurl.com/3fd99zew`. Accessed: 11-15-2023.

[2] S. Schüppen, D. Teubert, P. Herrmann, U. Meyer, FANCI: Feature-based Automated NXDomain Classification and Intelligence, in: 27th USENIX Security Symposium (USENIX Security 18), 2018, pp. 1165–1181.

[3] Y. Li, K. Xiong, T. Chin, C. Hu, A Machine Learning Framework for Domain Generation Algorithm-based Malware Detection, IEEE Access 7 (2019) 32765–32782.

[4] MITRE ATT&CK, Dynamic Resolution: Domain Generation Algorithms, [Online]. Available: `https://tinyurl.com/44hz9hpm`. Accessed: 11-15-2023.

[5] S. Yadav, A. K. K. Reddy, A. N. Reddy, S. Ranjan, Detecting Algorithmically Generated Domain-Flux Attacks With DNS Traffic Analysis, IEEE/Acm Transactions on Networking 20 (5) (2012) 1663–1677.

[6] Dave Piscitello, ICANN Senior Security Technologist, Conficker Summary and Review, [Online]. Available: `https://tinyurl.com/4d5mee8k`. Accessed: 11-15-2023.

[7] A. Drichel, N. Faerber, U. Meyer, First Step Towards EXPLAINable DGA Multiclass Classification, in: The 16th International Conference on Availability, Reliability and Security, 2021, pp. 1–13.

[8] M. Grill, I. Nikolaev, V. Valeros, M. Rehak, Detecting DGA malware using NetFlow, in: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), IEEE, 2015, pp. 1304–1309.

[9] NS1, Global DNS Traffic Report, [Online]. Available: `https://tinyurl.com/5y6j6n9y`. Accessed: 11-15-2023.

[10] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al., P4: Programming Protocol-Independent Packet Processors, ACM SIGCOMM Computer Communication Review 44 (3) (2014) 87–95.

[11] A. AlSabeh, J. Khoury, E. Kfoury, J. Crichigno, E. Bou-Harb, A survey on security applications of P4 programmable switches and a STRIDE-based vulnerability assessment, Computer Networks 207 (2022) 108800.

[12] J. Tourrilhes, P. Sharma, S. Banerjee, J. Pettit, The Evolution of SDN and OpenFlow: A Standards Perspective, IEEE Computer Society 47 (11) (2014) 22–29.

[13] Intel Tofino 3 Intelligent Fabric Processors, [Online]. Available: `https://tinyurl.com/mryubmxj`. Accessed: 11-15-2023.

[14] Y. Iuchi, Y. Jin, H. Ichise, K. Iida, Y. Takai, Detection and Blocking of DGA-based Bot Infected Computers by Monitoring NXDOMAIN Responses, in: 2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), IEEE, 2020, pp. 82–87.

[15] J. Ahmed, H. H. Gharakheili, C. Russell, V. Sivaraman, Automatic Detection of DGA-Enabled Malware Using SDN and Traffic Behavioral Modeling, IEEE Transactions on Network Science and Engineering (2022).

[16] L. Bilge, S. Sen, D. Balzarotti, E. Kirda, C. Kruegel, EXPOSURE: A Passive DNS Analysis Service to Detect and Report Malicious Domains, ACM Transactions on Information and System Security (TISSEC) 16 (4) (2014) 1–28.

[17] L. Fang, X. Yun, C. Yin, W. Ding, L. Zhou, Z. Liu, C. Su, ANCS: Automatic NXDomain Classification System Based on Incremental Fuzzy Rough Sets Machine Learning, IEEE Transactions on Fuzzy Systems 29 (4) (2020) 742–756.

[18] K. Highnam, D. Puzio, S. Luo, N. R. Jennings, Real-Time Detection of Dictionary DGA Network Traffic Using Deep Learning, SN Computer Science 2 (2) (2021) 1–17.

[19] B. Yu, D. L. Gray, J. Pan, M. De Cock, A. C. Nascimento, Inline DGA Detection with Deep Networks, in: 2017 IEEE International Conference on Data Mining Workshops (ICDMW), IEEE, 2017, pp. 683–692.

[20] S. Schiavoni, F. Maggi, L. Cavallaro, S. Zanero, Phoenix: DGA-Based Botnet Tracking and Intelligence, in: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, 2014, pp. 192–211.

[21] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, D. Dagon, From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware, in: 21st USENIX Security Symposium (USENIX Security 12), 2012, pp. 491–506.

[22] T. A. Tuan, H. V. Long, D. Taniar, On Detecting and Classifying DGA Botnets and their Families, Computers & Security 113 (2022) 102549.

[23] Intel, Intel Tofino P4-programmable Ethernet switch ASIC, [Online]. Available: `https://tinyurl.com/2p97j3pe`. Accessed: 11-15-2023.

[24] Aalsabeh Github, P4-DGA, [Online]. Available: `https://tinyurl.com/bddfhsyd`. Accessed: 11-15-2023.

[25] A. AlSabeh, K. Friday, J. Crichigno, E. Bou-Harb, Effective DGA Family Classification using a Hybrid Shallow and Deep Packet Inspection Tech-

nique on P4 Programmable Switches, in: ICC 2023-2023 IEEE International Conference on Communications (ICC), IEEE, 2023, pp. 1–6.
[26] J. Kim, H. Kim, J. Rexford, Analyzing Traffic by Domain Name in the Data Plane, in: Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR), 2021, pp. 1–12.
[27] A. AlSabeh, E. Kfoury, J. Crichigno, E. Bou-Harb, P4DDPI: Securing P4-Programmable Data Plane Networks via DNS Deep Packet Inspection, in: Proceedings of the 2022 Network and Distributed System Security (NDSS) Symposium, 2022, pp. 1–7.
[28] A. Kaplan, S. L. Feibish, DNS Water Torture Detection in the Data Plane, in: Proceedings of the SIGCOMM'21 Poster and Demo Sessions, 2021, pp. 24–26.
[29] Cybereason, What is Domain Generation Algorithm: 8 Real World DGA Variants, [Online]. Available: `https://tinyurl.com/2p9c2mrc`. Accessed: 11-15-2023.
[30] D. Plohmann, K. Yakdan, M. Klatt, J. Bader, E. Gerhards-Padilla, A Comprehensive Measurement Study of Domain Generating Malware, in: 25th USENIX Security Symposium (USENIX Security 16), 2016, pp. 263–278.
[31] D. P LOHMANN, DGArchive, [Online]. Available: `https://tinyurl.com/yc6whwrc`. Accessed: 11-15-2023.
[32] K. Friday, E. Kfoury, E. Bou-Harb, J. Crichigno, INC: In-Network Classification of Botnet Propagation at Line Rate, in: European Symposium on Research in Computer Security, Springer, 2022, pp. 551–569.
[33] E. F. Kfoury, J. Crichigno, E. Bou-Harb, An Exhaustive Survey on P4 Programmable Data Plane Switches: Taxonomy, Applications, Challenges, and Future Trends, IEEE Access (2021).
[34] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, M. Menth, A survey on data plane programming with p4: Fundamentals, advances, and applied research, Journal of Network and Computer Applications 212 (2023) 103561.
[35] Z. Liu, H. Namkung, G. Nikolaidis, J. Lee, C. Kim, X. Jin, V. Braverman, M. Yu, V. Sekar, Jaqen: A High-Performance Switch-Native Approach for Detecting and Mitigating Volumetric DDoS Attacks with Programmable Switches, in: 30th USENIX Security Symposium (USENIX Security 21), 2021.
[36] R. B. Basat, X. Chen, G. Einziger, O. Rottenstreich, Designing Heavy-Hitter Detection Algorithms for Programmable Switches, IEEE/ACM Transactions on Networking 28 (3) (2020) 1172–1185.
[37] A. Drichel, U. Meyer, S. Schüppen, D. Teubert, Analyzing the Real-World Applicability of DGA Classifiers, in: Proceedings of the 15th International Conference on Availability, Reliability and Security, 2020, pp. 1–11.
[38] C. Qi, X. Chen, C. Xu, J. Shi, P. Liu, A Bigram Based Real Time DNS Tunnel Detection Approach, Procedia Computer Science 17 (2013) 852–860.
[39] V. Tong, G. Nguyen, A Method for Detecting DGA Botnet Based on Semantic and Cluster Analysis, in: Proceedings of the seventh symposium on information and communication technology, 2016, pp. 272–277.
[40] H. Mac, D. Tran, V. Tong, L. G. Nguyen, H. A. Tran, Dga botnet detection using supervised learning methods, in: Proceedings of the Eighth International Symposium on Information and Communication Technology, 2017, pp. 211–218.
[41] Cisco, Cisco Umbrella, [Online]. Available: `https://tinyurl.com/mph6tvf3`. Accessed: 11-15-2023.
[42] A. Kaplan, S. L. Feibish, Practical Handling of DNS in the Data Plane, in: Proceedings of the Symposium on SDN Research, 2022, pp. 59–66.
[43] P. V. Mockapetris, Rfc1035: Domain names-implementation and specification (1987).
[44] A. T.-J. Akem, M. Gucciardo, M. Fiore, et al., Flowrest: Practical Flow-Level Inference in Programmable Switches with Random Forests, in: IEEE International Conference on Computer Communications, 2023.
[45] Netlab360, DGA Families, [Online]. Available: `https://tinyurl.com/22bfxzz4`. Accessed: 11-15-2023.
[46] VirusTotal, [Online]. Available: `https://tinyurl.com/2p9cscna`. Accessed: 11-15-2023.
[47] VirusShare, [Online]. Available: `https://virusshare.com/`. Accessed: 11-15-2023.
[48] Malpedia, [Online]. Available: `https://tinyurl.com/28k2snbk`. Accessed: 11-15-2023.
[49] Triage, [Online]. Available: `https://tria.ge/`. Accessed: 11-15-2023.
[50] Cuckoo Sandbox, What is Cuckoo?, [Online]. Available: `https://cuckoosandbox.org/`. Accessed: 11-15-2023.
[51] A. Küchler, A. Mantovani, Y. Han, L. Bilge, D. Balzarotti, Does Every Second Count? Time-based Evolution of Malware Behavior in Sandboxes, in: NDSS, 2021.
[52] S. Garcia, M. Grill, J. Stiborek, A. Zunino, An empirical comparison of botnet detection methods, computers & security 45 (2014) 100–123.
[53] Zeek, [Online]. Available: `https://zeek.org`. Accessed: 11-15-2023.
[54] J.-H. Lee, K. Singh, SwitchTree: in-network computing and traffic analyses with Random Forests, Neural Computing and Applications (2020) 1–12.
[55] Suricata, [Online]. Available: `https://suricata.io`. Accessed: 11-15-2023.
[56] S. learn, Sklearn Ensemble Isolation Forest, [Online]. Available: `https://tinyurl.com/ykmc9w48`. Accessed: 13-05-2024.
[57] S. learn, Sklearn Cluster KMeans, [Online]. Available: `https://tinyurl.com/yh2vebye`. Accessed: 13-05-2024.
[58] J. Bushart, C. Rossow, Padding Ain't Enough: Assessing the Privacy Guarantees of Encrypted DNS, in: 10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20), 2020.
[59] Halley, Bob, dnspython 2.4.2, [Online]. Available: `https://tinyurl.com/jmpem66s`. Accessed: 11-15-2023.
[60] Knot DNS, kdig – Advanced DNS lookup utility, [Online]. Available: `https://tinyurl.com/5n7e8u7z`. Accessed: 11-15-2023.
[61] Cloudflare, What is DNS? — How DNS works, [Online]. Available: `https://tinyurl.com/2s4b3cef`. Accessed: 11-15-2023.
[62] A. Drichel, B. Holmes, J. von Brandt, U. Meyer, The More, the Better: A Study on Collaborative Machine Learning for DGA Detection, in: Proceedings of the 3rd Workshop on Cyber-Security Arms Race, 2021, pp. 1–12.