DNSGreen: A Comprehensive Defense System Against Bounce-Style DNS DDoS Attacks With P4

Dan Tang, Xiaocai Wang, Pei Tan, Zheng Qin, Keqin Li, *Fellow, IEEE*, and Jiliang Zhang, *Senior Member, IEEE*

***Abstract*—DNS recursive resolver plays the role of intermediary agent in domain name query service, making it easy to observe various DNS flooding attack behaviors. The programmable data plane has promoted innovation in in-band attack detection, but previous work has not concentrated on DNS services, and there are concerns with mistakenly killing benign traffic and low classification accuracy. In this paper, we propose DNSGreen, a comprehensive defense system against bounce-style DNS DDoS attacks in the programmable data plane. DNSGreen discusses flow and packet characteristics under anomalous behavior, employing a "flow-detection and packet-filtering" pattern to ensure effective defense while allowing benign traffic to pass through as much as possible. Furthermore, DNSGreen designs a whitelist collection module, which eliminates the need to inspect trusted streams in subsequent filtering and thereby reducing the occurrence of false positives. Finally, DNSGreen improves the allocation method of the statistical structure to enhance the accuracy of system alarms. We deploy DNSGreen on BMv2 switches and conduct tests in three attack scenarios, demonstrating that DNSGreen safeguards normal users with excellent $F_{0.5}$ score while filtering attack traffic.**

***Index Terms*—Attack detection, denial-of-service, domain name system, P4.**

I. Introduction

DOMAIN Name System (DNS) is a crucial component in ensuring the normal activities of users on the Internet. Users initiate requests by sending queries to DNS resolvers, which then recursively or iteratively make queries to authoritative domain name servers. Throughout this process, the DNS recursive resolver operates with a global perspective, as it not only handles direct requests from users but also communicates with authoritative servers. Acting as an intermediary, the DNS recursive resolver facilitates the observation of various DNS attack behaviors.

As the query domain names in DNS requests can reflect malicious activities, many approaches [1], [2] employ deep learning methods such as LSTM for text analysis to detect DNS-related attacks. Since DNS operates on the UDP protocol, many attacks are carried out through IP deception techniques. One approach to discovering IP deception is by setting cookies to assess the source authenticity of the request messages [3]. However, these studies aim to improve the accuracy of recognition and classification, with little discussion on the practical deployment in high-speed networks. Firstly, some methods are offline and can only analyze the attack behavior from the historical traffic, by which time attackers may have already accomplished their intentions. Additionally, most online detection methods are deployed on DNS resolvers, and the computational overhead of learning models inevitably introduces traffic processing and forwarding delays, leading to a decline in server response quality. Some efforts employ GPU [4] and the Spark parallel framework [5] for acceleration, but the minute-level time overhead still seriously restricts performance, and attack flows also consume additional network bandwidth before reaching the resolver. In addition, research conducted within the SDN framework utilizes OpenFlow switches to report DNS packets to the controller, and the controller deployed with detection algorithms making decisions on whether to forward them. In this scenario, the channel between data and control planes becomes a bottleneck for forwarding [6], [7].

The current industry trend involves integrating attack detection directly into the forwarding plane, facilitated by the design of ASIC switching chips. These chips provide a framework of abstract forwarding logic for forwarding devices. With the programmability, switches can customize packet processing and forwarding behaviors, ensuring that analysis and forwarding execute at line speed with a throughput of up to Tbps. Attracted by the performance advantages of programmable switches, some scholars have also shifted security research from the network edge to the core.

The first category of attack detection methods in the data plane focuses on rate limit for malicious flows. POSEIDON [8] and Jaqen [9] designed a set of defense primitives for switches to block DDoS attacks. Sketch is a type of storage structure

Received 7 October 2024; revised 7 September 2025; accepted 19 October 2025. Date of publication 6 November 2025; date of current version 11 December 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62472153; and in part by Hunan Provincial Natural Science Foundation of China under Grant 2025JJ50350. Recommended for acceptance by Y. Chen. *(Corresponding author: Jiliang Zhang.)*

Dan Tang, Xiaocai Wang, Pei Tan, and Zheng Qin are with the College of Cyber Science and Technology, Hunan University (HNU), Changsha 410082, China (e-mail: Dtang@hnu.edu.cn; xiaocaiwang@hnu.edu.cn; tanpei@hnu.edu.cn; zqin@hnu.edu.cn).

Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561, USA (e-mail: lik@newpaltz.edu).

Jiliang Zhang is with the College of Integrated Circuit Science and Engineering, Nanjing University of Posts and Telecommunications, Nanjing 210023, China (e-mail: zhangjiliang@njupt.edu.cn).

Digital Object Identifier 10.1109/TC.2025.3625385

suitable for switches to store and track flow states [10], which can be used to estimate the frequency of IP addresses in each window and calculate the entropy of window sequences for DDoS attack detection [11]. Ripple [12] can detect dynamically changing types of attacks but require collaboration among multiple switches to achieve a global perspective. In this category, the punishment for attacks is flow-based, such as threshold-based filtering. When both attackers (with forged IPs) and legitimate users appear to be in the same flow, benign traffic may be erroneously blocked. Moreover, detecting different types of attacks necessitates deploying specific security primitives and adjusting switch configurations [8], [9], [12].

The second category of methods involves the field of machine learning-based packet classification. At present, there have been many precedents of deploying machine learning models to the data plane [13], [14], [15]. Switches can perform online inference based on packet header information to distinguish attack packets from benign packets. Because the decision tree model is simple and effective, it has been successfully deployed in many ways [16]. In addition, the binary neural network also performs packet-level classification of traffic entering the jurisdiction on the gateway switch [17]. However, due to hardware limitations, the computation of complex features is often constrained and forced to be approximated. Moreover, the limited capacity of switch pipeline stages and table entries also restricts the number of features for classification [16]. Such dual constraints hinder the effectiveness of machine learning models from reaching their full potential. Furthermore, achieving per-packet classification implies that each incoming packet must undergo steps involving extraction and computation of multiple features, thereby diminishing the performance of the switch.

To this end, we propose DNSGreen, an in-switch comprehensive defense system against bounce-style DNS DDoS attacks. While the ultimate victims of these attacks are usually external services or other nameservers, recursive resolvers are inevitably involved as stepping stones and bear additional workload from handling large volumes of bogus queries. Therefore, positioning the defense at switches that monitor resolver cluster traffic in real time allows us to implement effective protection with minimal overhead. By modeling the detection of different attacks as parallelizable query tasks, suspicious attack flows can be efficiently identified in the switch with minimal resource overhead. To reduce false positives on benign traffic, we implement two strategies: (i) a customized packet filtering engineering allowing for finer-grained selection, (ii) a whitelist collection mechanism ensuring that trusted traffic bypasses inspection. Our design enables DNSGreen to meet the requirement of line-rate execution on switches. The main contributions of this paper are summarized as follows:

(i) We propose DNSGreen, a defense system against bounce-style DNS DDoS attacks that uses a "flow-detection and packet-filtering" mechanism. In the P4 data plane, DNSGreen performs lightweight statistical analysis on flow information using an improved multi-query structure based on BeauCoup [18]. When anomalies are detected, it inspects each packet in the suspicious flow. Packet feature engineering ensures compatibility with the switch pipeline and facilitates deployment.
(ii) We propose an efficient whitelist collection mechanism in a flow-encoding matching manner. Packets from these whitelisted normal sessions are exempted from inspection, further reducing false positives.
(iii) We deploy DNSGreen on the BMv2 software switch to conduct a series of experiments on a wide range of datasets. We analyze computation and resource consumption using the open-source P4 Studio, and the results demonstrate that DNSGreen exhibits superior performance in both detection accuracy and computational efficiency.

The rest of the paper is organized as follows. Section II introduces the threat model and the background of P4 data plane. Sections III and IV provide detailed explanations of the design and deployment optimizations of DNSGreen. In Section V, we present a performance comparison between DNSGreen and baseline systems. We review relevant research work in Section VI. Section VII concludes the paper.

II. Background

In this section, we establish a threat model for bounce-style DNS DDoS attacks, highlight the disadvantages of current researches, and describe the new opportunities and challenges that P4 switches bring to the field.

A. Threat Model and Problem Statement

Domain name service is based on the tree hierarchy structure of "root domain - top level domain - authoritative domain". Generally, the requests from hosts to resolvers are recursive queries, and the subsequent queries by resolvers constitute iterative queries. In recent years, public DNS providers have incorporated various security measures, such as traffic scrubbing and redundancy mechanisms. However, due to misconfigurations or limitations in handling large-scale traffic, resolvers are still exploited as stepping stones to launch volumetric attacks, which in turn degrade their own service quality and threaten the security of other nameservers and web services. P4 switches that forward DNS traffic at line rate at the ingress of a provider's anycast site offer a natural vantage point for deploying such defense strategies. This paper discusses three types of bounce-style DNS DDoS attacks.

Distributed Reflection Denial-of-Service (DRDoS) attack, also known as an amplification attack, hinges on two key factors: IP spoofing and the benefits of amplification [19]. DNS is based on the UDP protocol and generates response packet payloads (1500+ Bytes) several times larger than the request packets (60+ Bytes) in the case of forced recursive queries. Additionally, its global accessibility unfortunately makes it an ideal target for attackers. The attack path is indicated by the red line in Fig. 1. Attackers first conduct port scanning to identify available open resolvers on the Internet, some of which may have recursive DNS services enabled. To maximize the response size, attackers typically include an "ANY" parameter in their queries. Without authenticity checks on the source IP address,

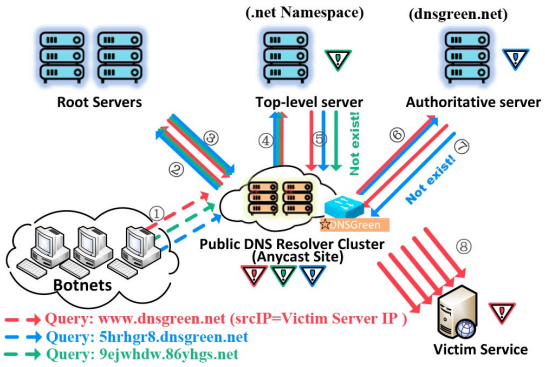


Fig. 1. Bounce-style DNS DDoS attack model.

request packets from attackers can trigger a substantial volume of response packets, which are then redirected to overwhelm third-party victim infrastructure, such as web servers, network gateways, or cloud services.

A Random Subdomain (RSD) attack is designed to target a specific domain name [20]. As shown by the blue line in Fig. 1, the attacker launches a flood of requests to the target domain name, but each query's subdomain consists of a meaningless sequence of characters generated by a domain name generation algorithm (DGA). This strategy overloads the authoritative server of the target domain with a vast number of invalid queries, ultimately leading to service paralysis. Legitimate users attempting to access the website will experience delays or fail to access it entirely. An NXDomain attack is similar to the RSD attack, except that the requested parent domain name is randomly generated [21]. For each request, the TLD nameserver must spend time performing a lookup. Since the queried domain does not exist, the nameserver returns an NXDomain reply and caches a large number of erroneous requests. This severely depletes its computational power and cache resources, leaving it unable to process legitimate queries. Additionally, RSD and NXDomain attacks can also disrupt the normal running of DNS recursive resolvers, causing them to become overwhelmed with handling large volumes of erroneous requests, which prevents clients from finding the servers they are searching for. The attack path is indicated by the green line in Fig. 1.

We categorize existing detection methods in Table I. The first category of methods is deployed on DNS resolvers, increasing the workload of the kernel. Detection and forwarding latency are gradually becoming unacceptable in today's high traffic environments such as data centers. The advent of programmable switches and traffic pressure have prompted an explosion of in-band detection schemes. Representative approaches like POSEIDON [8] and Jaqen [9] have developed a series of security primitives for P4 switches, where defense primitives control the rate based on collected flow information. The worst-case scenario occurs when normal users are grouped with attackers into the same flow, leading to the erroneous discard of a large number of legitimate packets. Learning models deployed on switches, such as Planter [13] and Taurus [15], can be used for traffic classification. However, this requires continuous extraction and computation of multiple packet features, posing challenges to switch performance.

TABLE I
COMPARISON WITH PRIOR WORK

Prior Works	Line-Speed	Distinguish in a Flow	Low-intensity computation
Methods deployed on servers [1], [2], [4], [5]	✗	–	✓
POSEIDON [8], Jaqen [9]	✓	✗	✓
Planter [13], Taurus [15]	✓	✓	✗
DNSGreen	✓	✓	✓

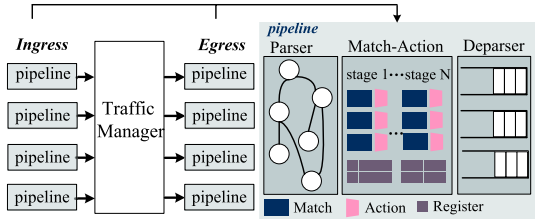


Fig. 2. The abstract forwarding logic of P4 switches.

B. Programmable Data Plane

Due to the original principle of "core simplicity, edge complexity" in the construction of computer networks, switches were initially responsible for basic routing and forwarding. Software-Defined Networking architecture abstracts the control plane and enables centralized management through southbound protocols such as OpenFlow. However, switches need to continuously convey flow table statistics to the controller [22], [23]. In current high-speed traffic forwarding requirements, this link is prone to becoming a bottleneck and a target for attacks, such as cross-path attacks and `Packet-In` flooding [24]. Building on this infrastructure, a programmable data plane has been developed, harnessing the power of ASIC chips and endowing switches with cognitive capabilities. As a crucial element in the network core, programmable forwarding devices support security functions such as deep packet inspection, congestion control, and fault detection [25]. And specialized hardware designs support their rapid operation, making them applicable in high-traffic environments.

As shown in Fig. 2, a programmable switch has multiple pipelines, where a packet is processed successively through the ingress pipeline and egress pipeline. There is a flow management opportunity between the two pipelines, such as queue control and packet replication. In each pipeline, the parser progressively extracts relevant packet header information, which serves as matching fields in the match-action (M-A) table. Upon successful matching, actions like modifying packets or updating registers are executed. Finally, the packet is repackaged through reverse parsing. Network operators can customize this abstract packet processing and forwarding logic through P4 language.

In fact, the efficiency of P4 switches is achieved within various constraints. Firstly, although switches provide multiple pipelines and each pipeline can contain numerous match-action units, data with dependencies must be placed in different stages, and the number of stages is limited [26]. Secondly, the switch provides stateful long-term storage across packets, such

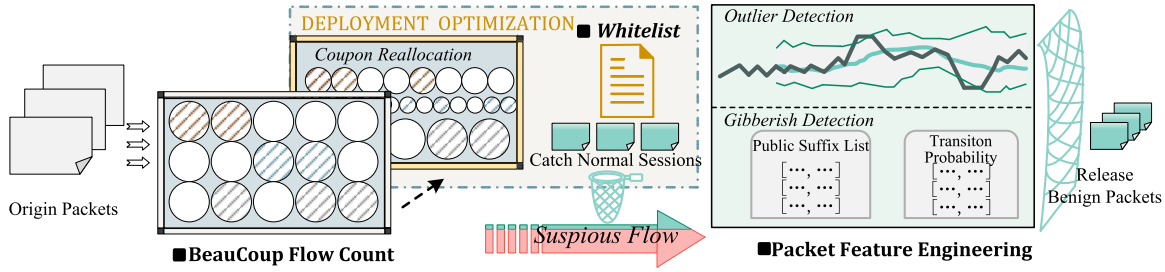


Fig. 3. The architecture of DNSGreen.

as registers and match-action tables, but the available SRAM (100+Mb per pipeline) and TCAM (< 10Mb per pipeline) resources are limited and highly valuable. Thirdly, the switch supports CRC Hash and meter, but only simple ALU and conditional logic operations are supported. In summary, P4 programs must be carefully designed to operate within these limits, making resource-aware design essential for in-band system deployment.

III. DNSGreen: Flow-Detection and Packet-Filtering

DNSGreen focuses solely on DNS traffic, expressing various attacks to be detected in the form of queries. It collects coupons in BeauCoup [18] to monitor flows. When the number of coupons collected by a flow exceeds the specified threshold, an alert is triggered. Subsequently, all packets in this flow undergo packet diagnosis. If one packet exhibits the characteristics of corresponding attack, it is an attack packet. Otherwise, it is a normal packet, which is called the "flow-detection and packet-filtering" mode. Additionally, in practical deployment, incorporating whitelist collection module and using an improved BeauCoup can achieve better results. The system architecture of DNSGreen is shown in Fig. 3.

A. Flow Statistic

BeauCoup [18] gets inspiration from the coupon collection problem and implements multiple queries simultaneously in the data plane. A query, denoted as q, is composed of its key, attributes, and a threshold T_q. It focuses on whether different attributes under the same key exceed the T_q. For each query, m_q coupons are set, and each coupon is drawn with a small probability of p_q. When n_q different coupons are drawn, it is considered that the threshold of the query has been reached. By reasonably setting (m_q, n_q, p_q), efforts are made to make the expected number of draws required to obtain n_q coupons, denoted as $E_q = \sum_{j=0}^{n-1} \frac{1}{p(m-j)}$, close to T_q, achieving good query performance. Taking DDoS attacks as an example, the key for this query is the destination IP, the attribute is the source IP, and the threshold is set at 1000. For instance, if 1000 different source IPs are found in the flow with the destination IP 10.0.0.1, it is considered that the host at 10.0.0.1 is a victim attacked by DDoS. In BeauCoup, this may be manifested as (32, 20, 1/1024), which means that 20 out of total 32 coupons need to be successfully drawn by the packets destined for 10.0.0.1, and the probability of draw for each coupon is 1/1024. The source IP field of the packet is hashed, mapping to the range [0, 65535], and only packets within [0, 2048] can successfully draw a coupon. With shared memory and extremely low-probability draw, BeauCoup can perform multiple queries simultaneously with minimal memory overhead and access times.

TABLE II
BeauCoup Queries Corresponding to the Attacks

Name	Key	Attribute	Threshold
DRDoS attack	*dstIP*	*timestamp* if srcPort=53, otherwise $\emptyset$	1000
RSD attack	*domain*	*subdomain*	800
NXDomain attack	*dstIP*	*timestamp* if NXDomain, otherwise $\emptyset$	600

Bounce-style DNS DDoS attacks still exhibit the characteristics of volumetric attacks, and the flood of attack streams in quantity enables them to be discovered by the set threshold. In order to comprehensively defend against various bounce-style DNS DDoS attacks and facilitate scalability, DNSGreen adopts BeauCoup for global flow monitoring. Once an anomaly is detected, it issues an alert. And the switch sends a clone packet to the controller which carries the identity information of the abnormal flow. After that, the controller will apply a table entry for the flow as a blacklist storing in the switch. This process is referred to as "flow-detection", and a flow is defined as a set of packets with the same key in this paper.

The quantitative characteristics of the three types of attacks are converted into the queries shown in Table II. These queries monitor the number of response packets sent to a forged source IP, the number of random subdomains for the same domain and the number of received NXDomain packets in a period of time respectively. Since the timestamp of entering the switch for each packet is different, we can treat timestamp as the attribute instead of packet count and hash it to draw coupons. Moreover, by leveraging the multi-query feature of the BeauCoup structure, DNSGreen exhibits strong scalability. Volumetric attacks that present distinctive quantitative patterns can be incorporated into its defense scope by allowing developers to configure customized query parameters.

B. Packet Feature Engineering

After BeauCoup issues an alert, discarding all the packets in the suspicious flow is a drastic measure. Because the concept of "flow" is relatively broad (a set of packets with same key),

packets from normal communication may be mixed in. Further screening of subsequent packets in suspicious flows can reduce the impact on normal hosts. In this way, a packet-level classification policy, called "packet-filtering", needs to be formulated for each attack.

1) Outlier Detection: DRDoS attackers disguise their identity by modifying the source IP to the victim IP and send requests to DNS servers. Consequently, a large number of amplified response packets from DNS servers are directed towards the victim host. Based on this phenomenon, flow statistics can be collected for monitoring. Additionally, DNS reflection attacks rely on oversized response packets to achieve amplification. Therefore, the suspicious flow is filled with numerous long packets, while a few relatively shorter ones in it are highly likely to be normal packets. If the packet length of packets in the suspicious flow is regarded as a time series, benign packets can be viewed as the abnormal points within it, and the work of screening benign packets is transformed into anomaly detection. Traffic anomaly detection methods include statistical approaches, machine learning, and tensor factorization. Due to limitations in the computational capabilities of domain-specific chips in the data plane, simple and efficient statistical method is the optimal choice.

The smoothed z-score algorithm [27] considers the overall mean μ to be a predict value of the next signal. Calculate the deviation between the signal X_i and the mean μ, and this deviation divided by the standard deviation δ gives the z-score of the signal X_i as Eq. (1) shows. When the z-score is greater than 3 or less than -3, X_i is considered significantly deviated from the overall pattern, indicating an outlier. In real-time detection, the reference signals $[smoothX_{j-l+1}, \ldots, smoothX_j]$ are established dynamically based on a sliding window of length l. If the current signal X_j is identified as an outlier, it is smoothed by borrowing the previous historical signal $smoothX_{j-1}$ and smoothing factor α to reduce its impact, as shown in Eq. (2). The $smoothX_j$ of normal signal is itself.

$$z\text{-}score = \frac{X_i - \mu}{\delta} \tag{1}$$

$$smoothX_j = \alpha \cdot X_j + (1-\alpha) \cdot smoothX_{j-1} \tag{2}$$

In traditional statistical methods, the calculation of the z-score generally necessitates the maintenance of a complete data window, specifically the most recent N packets, to accurately compute the mean and standard deviation. However, directly maintaining historical data windows is almost infeasible in resource-constrained programmable switches. To balance resource usage and accuracy, we adopt the exponentially weighted moving average (EWMA) streaming method, as shown in Eq. (3)-(5). This method incrementally updates the mean μ and the mean square $E(x^2)$ based on the previous values μ_{prev} and $E(x^2)_{prev}$, and then calculates the variance σ^2 from them. When an anomaly is detected, the z-score algorithm updates with the current packet length and the smoothed value of the previous packet length. In the streaming approach, dynamic adjustment of α (default 1/8) is used:$\alpha = \frac{1}{2}$ for rapid convergence to the true statistics during the first 16 packets and $\alpha = \frac{1}{16}$ for smoothing when an outlier is detected. For computational feasibility, shifting is used in place of division. Additionally, since attackers often exploit the ANY query type in recursive queries to amplify attack impact, we integrate an ANY query type check into the detection logic. The workflow of the streaming z-score is shown in Algorithm 1, and it can be completed within 10 stages of the switch pipeline.

Algorithm 1: Packet-filter for DRDoS_DNS attack

input : length $pLen$, query type $qType$ of a packet
output: $allow\_forward$ (0: discard, 1: forward)

allow_forward = 1;
if $qtype == ANY$ **then**
    $allow\_forward$= 0 ;
    exit;
$curr\_avg$ = avgR.read();
$curr\_sq\_avg$ = sq_avgR.read();
cnt= cntR.read();
if $cnt < 16$ **then**
    // Fast convergence during initial stage
    $new\_avg = ((\text{curr\_avg} \times 1) + pLen) \gg 1$;
    $new\_sq\_avg =$
    $(\!(curr\_sq\_avg \times 1) + (pLen \times pLen)\!) \gg 1$;
else
    // Streaming update of average
    $new\_avg = ((\text{curr\_avg} \times 7) + pLen) \gg 3$;
    $new\_sq\_avg =$
    $(\!(curr\_sq\_avg \times 7) + (pLen \times pLen)\!) \gg 3$;
    $new\_avg2 = ((\text{curr\_avg} \times 15) + pLen) \gg 4$;
    $new\_sq\_avg2 = (\!(curr\_sq\_avg \times 15) + (pLen \times pLen)\!) \gg 4$;
$var = new\_sq\_avg - (new\_avg \times new\_avg)$;
$z\_squared =$
$(pLen - new\_avg) \times (pLen - new\_avg)$;
if $z\_squared$ >$9 \times var$ **then**
    $allow\_forward$ = 0 ;
    // Apply smoothing if needed
    new_avg = new_avg2;
    new_sq_avg = new_sq_avg2;
else
    $allow\_forward$ = 1;
$avgR$.write(new_avg); // resubmit to update average register
sq_avgR.write(new_sq_avg);
$cntR$.write($cnt + 1$);

$$\mu = \alpha \cdot X_i + (1-\alpha) \cdot \mu_{\text{prev}} \tag{3}$$

$$E[X^2] = \alpha \cdot {X_i}^2 + (1-\alpha) \cdot E[X^2]_{\text{prev}} \tag{4}$$

$$\sigma^2 = E[X^2] - \mu^2 \tag{5}$$

2) Gibberish Detection: RSD attacks and NXDomain attacks involve the forgery of randomly generated subdomains and domains, respectively. Therefore, gibberish detection can be performed on subdomains and domains for packet filtering of these two attacks. However, the query name of a DNS packet is not a fixed-length field which consists of several labels of

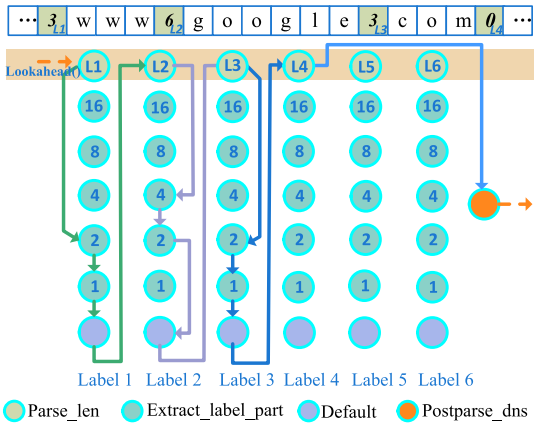


Fig. 4. State transition diagram of DNS domain name extraction in parser.

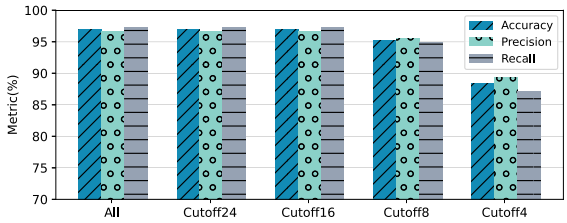


Fig. 5. Gibberish detection result in five cases of different truncation lengths.

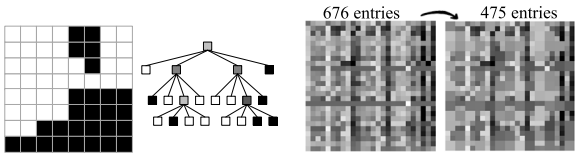
varying lengths. The encoding format of a label in the packet includes the label length followed by the label characters, with a terminating *"\00"* after the last label to indicate the end of the domain name. A feasible extraction method is to store each label in five parts and extract up to 16, 8, 4, 2, and 1 byte sequentially according to the length of the label. As shown in Fig. 4, the P4 parser supports a lookahead operation, allowing it to read the length of the possible upcoming label in advance. This length is then used as a trigger for state transitions, allowing the parser to transition from the state handling the longest possible label to the appropriate state. For example, the five parts of the label *"google"* are *null*+*null*+*"goog"*+*"le"*+*null*. When the current label can no longer trigger an extracting state, the parser transitions to the default state to process the next label. If a domain name is allowed to have a maximum of six labels parsed, this approach achieves a 97% successful extraction rate for DNS packets [28]. Expressing the Public Suffix List as entries stored in the switch and using these labels for matching can effectively divide a query name into domain and subdomain. Thus, we can get the domain *"google.com"* and the subdomain *"adhgc123"* from the query name *"adhgc123.google.com"*.

In general, for ease of memorization and use, legitimate domain names have high readability and may consist of common words. In contrast, randomly generated domain names by attackers lack this characteristic. Given a character set C, $|C| = n$, use a large training sample containing only normal words to count the occurrences times of $c_i c_j (0 \leq i < n, 0 \leq j < n)$ and take log values after normalization to form an $n \times n$ transition probability matrix T. A string can be considered as a Markov chain, with C being its state space. For a test set containing both normal words and random strings, calculate the average transition probability of the two categories according T and set a threshold for classification. For a string s of length l, Eq. (6) sums up the transition probabilities of each state and divide by state transition counts to get the overall $P(s)$. The use of addition instead of multiplication is due to the prior logarithmic processing of the transition probability matrix. If $P(s)$ is less than the threshold, s is considered infrequently occurring and a weakly readable random string.

$$P(s) = \frac{\sum_{i=0}^{l-2} T_{s[i]s[i+1]}}{l-1} \tag{6}$$

We use the English alphabet character set with $|C| = 26$. When considering implementing gibberish detection in the data plane, it is necessary to express the matrix T as a table with 26^2 entries. However, each table in the packet processing pipeline can only be accessed once. According to the extraction method by the exponent of 2 mentioned earlier, a label can be up to 31 characters long, so it is necessary to set such 30 tables to meet the requirements. This significantly increases the consumption of storage resources and the number of accesses. Generally, the composition of a string is relatively stable, that is, it rarely appears that the first half of the string is very random and the second half is highly readable. Therefore, taking a local segment of the string can reflect the overall randomness. For simplicity, we take only the leading portion of the string. In Fig. 5, the classification results are plotted, showing that gradually reducing the length of the extracted segment, even down to only the first 8 characters, does not significantly decrease the classification performance. Therefore, when performing gibberish detection, we can provide only 8 transition probability tables for querying.

P4 supports masked ternary matching, such as 010& 110, which means only the first two bits matter. The field value can be either 010 or 011 to match successfully. If the values of neighboring points in the transition probability matrix are close, they can be aggregated into a ternary entry, further reducing the table size. If we envision the transition probability matrix as an image, where each normalized probability value in the range [0, 255] corresponds to a grayscale intensity, the process of reducing table size is similar to image blurring. As illustrated in Fig. 6, the iterative division of an image into four blocks can be seen as constructing a quadtree. In this process, each node's value equals the average grayscale intensity. Subsequently, the quadtree is reconstructed from the leaf nodes upwards. When the color deviation among the four sibling nodes falls below the predetermined range, these nodes are merged, with the parent node adopting the average grayscale value of its four child nodes. Through the consolidation of nodes with similar colors, we successfully aggregate table entries, reducing the table from its original 676 (26× 26) entries to 475 entries, while maintaining an accuracy rate of 90.7%. Thus, by cutting the length of the label and compressing the transition matrix, we have efficiently minimized the consumption and access of



(a) Quadtree storage of an image. (b) Reduction of table entries.

Fig. 6. Transition probability matrix compression.

storage resources in terms of both the number and size of tables, with only a slight loss of classification accuracy.

IV. Deployment Optimization

In this section, we enhance DNSGreen from two aspects. Firstly, we introduce a whitelist collection module to provide more comprehensive protection for benign sessions. Secondly, we customize the access opportunity allocation for different queries in BeauCoup to improve counting accuracy.

A. Whitelist Collect Module

In practice, even with packet-filtering, there may still be cases of misclassifying benign packets. For example, some normal domain names may exhibit high randomness, like *"z-1.c10r.facebook.com"*. The classification of a packet can also be determined in a posteriori manner. In the case of RSD attacks, this is manifested as follows: for a querying subdomain, if its response packet has passed through the switch, it indicates that the subdomain can be resolved normally and is legitimate. This method requires recording the state of each flow and has a lag, making it unsuitable for universal application. However, when locally applied on the basis of the previous "flow-detection and packet-filtering", it can be an additional enhancement.

Upon receiving an alert, instead of immediately enabling packet filtering for packets in suspicious flows, a short waiting period is set. During this waiting period, the information (srcIP, dstIP, query domain) of connections that meet certain rules will be added to a whitelist. Packets in the whitelist are exempt from further inspection in subsequent packet-filtering and are directly passed through, avoiding misjudgments. We formulate that a trustworthy DNS session must have both request and response packets in the connection, and their difference of payload size should not be significant. This ensures the participation of both communicating parties and excludes reflection attacks.

1) Common Version: A common approach for whitelist collection is to use multiple sets of registers to record flow states including the identity of DNS packets (request or response), their lengths, and the timestamps when they entered the switch. By hashing the combination of source IP and destination IP to find the index, packets from both directions of a session can be addressed to the same location of registers. To determine whether the current packet forms a legitimate connection with a previous packet, three conditions must be simultaneously satisfied: (1) checking if it constitutes a request-response service relationship by conditional judgment, (2) evaluating whether the ratio of packet payload produces a reflective effect

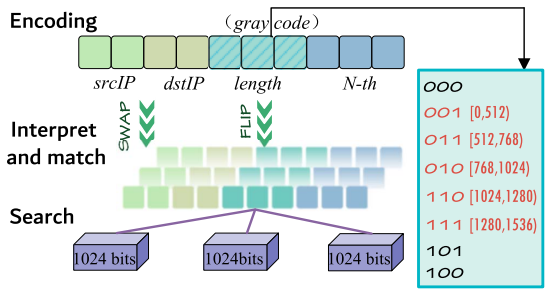


Fig. 7. The Bit-string version of whitelist collection.

by division, and (3) determining if the timestamp gap exceeds the timeout internal by subtraction. The disadvantage of this way is its consumption of massive register resources and the operations involved are complex. Especially considering that the fixed functions of ASIC chips in P4 switches do not include division operations, so it need to be implemented indirectly, often through preset table entries to obtain approximate results.

2) Bit-String Version: In P4, there is support for an arbitrary-length bit-string type. Base on this, we have reimplemented the above version to use bit operations as much as possible for accelerated processing. We aggregate several valid information of a packet into a binary encoding with the following rules:

- For the source IP and destination IP, instead of hashing, we use modulo 4 operation to obtain a two-bit IP code.
- For packet payload, we use 256 as the internal size and obtain the interval number by right-shifting the packet length field by 8 bits. It can be roughly believed that the packet length difference between two adjacent intervals is small and does not have the reflection effect, while the packet length difference between far apart intervals is large and may not meet the requirements of normal DNS sessions. Gray code is a reliable encoding, with only one binary bit different between adjacent codes. If we encode the interval number into Gray code, finding its neighboring interval only requires one bit-flipping operation. Following the method of dividing the interval range in Fig. 7, we first perform a bitwise OR operation with 100000000 (256D) on the packet length, then right-shift by 8 bits to get the interval number, and finally convert it to the corresponding Gray code representation. This partitioning is designed to cleverly achieve neighboring interval number by flipping the least significant bit or the second least significant bit (except for 010 and 110).
- For the original packet timestamp, we replace it with the order of the packet entering the switch and right-shift the order by 8 bits to get its three-bit encoding. Similar to dividing time windows, packets with the same order code are in the same time window, otherwise it is considered a timeout.

Concatenating the aforementioned encoding segments can produce a complete encoding X for a packet. Packets with three encodings $Y_{1,2,3}$ can form a legitimate connection with it. And they can be matched only by rapid bitwise operations, as shown in the following formulas. X' is the result of swapping the source IP and destination IP, equivalent to finding the other

TABLE III
COMPARISON OF WHITELIST COLLECTION OPERATIONS

Common Version	Bit-string Version	Bit Operation
IP Hashing for addressing	Modulus	$IP\&2^2-1$
Recording packet length	Gray code	$NO.=(length\|256)\gg 8$ $grayNO.=NO.\oplus(NO.\gg 1)$
Division for reflection multiple	Bit-flipping	$grayNO.\oplus\{000,001,010\}$
Ensuring two parties: IP_1=$min(srcIP,dstIP)$ IP_2=$max(srcIP,dstIP)$ register index=$Hash(IP_1+IP_2)$	IP swap	$((X\ll 4)\gg 4)\|(((X\ll 2)\&768)\|((X\gg 2)\&192))$
Subtracting timestamp	Order record	$order\gg 8$

party in the session. Then, XOR it with specific values to obtain the encodings Y_1, Y_2 and Y_3, which means finding packets with comparable lengths by bit flipping.

$$X^{'}=((X\ll 4)\gg 4)|(((X\ll 2)\&768)|((X\gg 2)\&192)) \tag{7}$$

$$Y_{1,2,3}=X^{'}\oplus\{0000001000B,0000010000B,0000000000B\} \tag{8}$$

As Fig. 7 shows, establish a set of registers with a length of 1024 (2^{10}), where each unit stores a single bit 0 or 1. And the index of registers corresponds to the encoding value of the packet. For an arriving request packet, update the corresponding unit to 1. For an arriving response packet, after obtaining its encoding, calculate the three possible matching encodings and check whether the values at the corresponding units are 1. If at least one of them is 1, the connection will be added to the whitelist. In practice, registers in P4 switches can only be accessed once per packet pipeline, so three sets of such registers are required. Since only flag bits are stored, the total overhead is extremely low. And the entire process involves only bitwise operations, this version speeds up the processing of packets in the pipeline. The comparison between the common version and the bit-string version, along with the required bitwise operations, is summarized in Table III.

B. Coupon Reallocation in BeauCoup

BeauCoup supports multiple queries simultaneously with a small memory access cost. By inputting the average activation number of coupon per packet γ_q and the alert threshold T_q of queries into the compiler, an optimal configuration (m_q, n_q, p_q) corresponding to each query can be generated to minimize errors. The alarm is triggered when n_q out of m_q coupons are drawn, and the probability of drawing each coupon is p_q. In the original design, each query was assumed to have a fair share of memory access opportunities, with γ_q evenly distributed based on the total per-packet activation limit Γ. However, the key and attribute involved in each query may belong to different types, such as some queries being only related to TCP and others only related to UDP. If only a few UDP packets are sent in a query traffic set and the majority are TCP packets, the memory allocated to UDP-related queries will be wasted partly. In these UDP-related queries, TCP packets are unconcerned. Thus, all these TCP packets could be assigned a fixed value while drawing, resulting in either non-hits or consistently drawing the same coupon. This situation offers no meaningful contribution to the query. Transferring a portion of these access budget to TCP-related queries could potentially enhance the overall accuracy of counting.

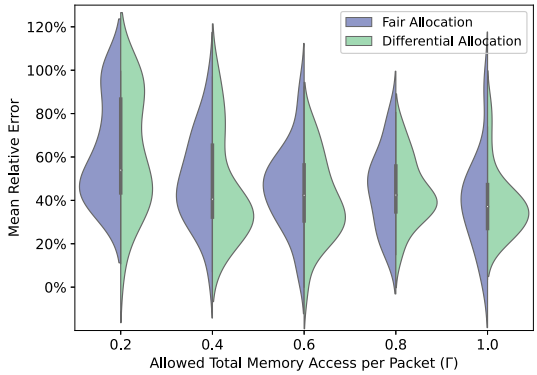


Fig. 8. The comparison of two allocation methods in queries for MAWI [29].

For example, when $m=32$ and $\gamma_q=0.05$, constrained by $m_q\cdot p_q\le\gamma_q$, the maximum value for p_q can be 1/1024. As γ_q increases to 0.1, p_q can be up to 1/512. If the threshold T_q of a TCP-related query is 1300, then the configuration (32, 30, 1/512) with $\gamma_q=0.1$ would require more hits and have a smaller error compared to the configuration (32, 23, 1/1024) with $\gamma_q=0.05$. For UDP-related queries, due to low proportion of UDP traffic, it can be understood as the case of a high alarm threshold which only requires very small p_q. Therefore, increasing their access budget does not provide significant assistance.

Using the 26 query examples provided in the original BeauCoup project (4 IP-related, 13 TCP-related, and 9 UDP-related queries), queries were conducted on the MAWI traffic dataset [29] (80% TCP, 20% UDP). Under two memory access allocation methods, we compare the average relative errors between T_q and the actual distinct attribute counts while alerting. One method is fair allocation, while the other allocates a portion of the UDP-related query's memory access budget to TCP-related and IP-related queries. As shown in the Fig. 8, differential allocation based on the proportion of traffic types results in a smaller average relative error, improving the query accuracy. In the scenario of this paper, because NXDomain packets have a minimal weight in DNS traffic, their γ_q can be appropriately reduced, giving the saved budget to DRDoS and RSD queries. This adjustment reduces the mean relative error from 31.65% to 28.47%.

V. EXPERIMENT AND EVALUATION

In this section, we conduct experiments on DNSGreen to validate its system performance, assess the effectiveness of deployment optimizations, and delve into more detailed issues.

TABLE IV
DETAILS OF MALICIOUS TRAFFIC DATASETS

Class	Dataset	Description	IP.*	PL.†
DRDoS	CIC-DDoS-2019 [31]	Simulated DRDoS attack on testbed.	69	1269
	Amp_UDP_DNSANY [1]	Real-world DDoS attack captures.	79	521
	Honeypot_DRDoS [2]	Honeypot for DRDoS attack tracking.	199	597
	DNS_Anomaly_Inject [3]	Anomaly-injected DNS attack simulator.	1190	932
RSD	Booter-RSD [32]	Booter traffic with modified queries.	7302	142
	DNSperf[4]	DNSperf simulator with Matsnu DGA.	1000	146
NX Domain	Booter-NXDomain [32]	Booter traffic with modified queries.	7323	127
	CIC-Bell-DNS [33]	Diverse malicious domain types.	500	147

∗: The number of distinct IP pairs. †: The average packet length.
[1]: https://github.com/StopDDoS/packet-captures
[2]: https://www.kaggle.com/datasets/ricardostoklosa/honeypot-drdos
[3]: https://github.com/niclabs/dns-anomaly-injection
[4]: https://github.com/DNS-OARC/dnsperf

A. Experimental Setup

1) Simulation Platform: We conduct experiments on a VMware virtual machine running Ubuntu 20.04 with 16GB RAM. The physical host is equipped with Intel Core i5-7500 @3.40GHz CPU. DNSGreen is deployed on the BMv2[1] software switch which is connected to the python-coded control plane by P4Runtime. After creating port connections for the switch in Mininet[2], we utilize the Tcpreplay[3] tool to replay traffic datasets.

2) Defense Tasks: We evaluate DNSGreen under the following tasks and summarize the attack datasets involved in Table IV. The benign traffic comes from a campus DNS network traffic with over 4000 active users [30]. We use the term "Confusing Normal Traffic Engagement Level" to represent the proportion of benign traffic mixed with attack traffic. Specifically, some benign packets in our test scenarios are blended into the attack flow by sharing the same IP address or domain name.

- DRDoS attack defense. We used four datasets. The CIC-2019DDoS and Amp_UDP_DNSANY datasets are packet captures of DRDoS attacks in testbed experimental environments and real-world scenarios, respectively. The Honrypot_drdos dataset uses honeypot technology to track DRDoS attacks. The Dns_anomaly_injection dataset generates attack simulations by injecting anomalies into benign traffic.
- RSD attack defense. We use five DGAs (corebot, monerodownloader, newsoz, reconyc, qadars) to generate random subdomains and modify the query name field of the Booter traffic dataset to simulate a random subdomain attack. Furthermore, to verify the detection robustness, we use the DNSperf simulator combined with Matsnu DGA to launch RSD attacks. The Matsnu DGA generates domain names by randomly concatenating existing words, making them highly readable and more difficult to detect.
- NXDomain attack defense. We also use the Booter traffic dataset to construct NXDomain attacks. The difference is that, in this case, the string generated by the DGA acts as the chaotic parent domain component. Additionally, the CIC-Bell-DNS dataset provides publicly available NXDomain attack traffic. The malicious domains span three categories of spam, phishing, and malware.

3) Baselines: We use the following four attack detection methods in P4-based programmable data planes as baselines.

- **FlowLens** [34]: FlowLens classifies traffic based on packet length distributions. It uses a flow marker aggregator in the data plane and optimizes register usage through quantization and truncation. The collector and classifier operate in the control plane.
- **Poseidon** [8]: Poseidon provides in-switch primitives for detecting various DDoS attacks. For DNS amplification attacks, we use Poseidon's customized defense primitives. For RSD and NXDomain attacks, we apply the `rlimit` primitive for mitigation.
- **IIsy** [16]: IIsy is a per-packet ML-based classification method using decision trees for easy deployment in the data plane. For DRDoS attacks, the features include packet length, query type, Answer Resource Record count, and Additional Resource Record count. For RSD attacks, the features include name length, the length of the leftmost label, and bigrams. NXDomain attack involves features such as Contains Digits (0 or 1), Has www prefix (0 or 1), Domain name length, and Alphabet cardinality [5].
- **Planter** [13]: Planter is a deployment framework for in-network machine learning models. We implement three models: KNN, SVM, and BNN, which correspond to Planter's encode-based, lookup-based, and direct-mapping solutions, respectively. The features used are the same as those in IIsy.

4) Metric: Our goal is to release benign traffic as much as possible and reduce the possibility of accidental killing, while achieving basic defense effects. Therefore, taking the attack packet as positive, we prioritize precision over recall. Consequently, we choose the $F_{0.5}$ score to evaluate the defense effectiveness and F_α score is calculated as Eq. (9). Additionally, TNR is included to assess the specificity of the attack detection system, as shown in Eq. (10). In the experiment, the switch forwards recognized normal and attack packets to different ports.

$$F_\alpha\ score = (1+\alpha^2) \times \frac{precision \times recall}{\alpha^2 \times precision + recall} \tag{9}$$

$$TNR = \frac{TN}{TN+FP} \tag{10}$$

B. Detection Evaluation

Table V presents an overall comparison of the $F_{0.5}$ score and TNR metrics for different methods across three attack scenarios. DNSGreen achieves improvements over the best baseline in all cases. With $F_{0.5}$ score ranging from 0.9146 to 0.9913 and TNR from 0.9878 to 0.9925, DNSGreen stands out as the most stable and effective method.

1) DRDoS Attack Defense: First, we evaluated the performance of DNSGreen and the baselines across four different DNS amplification attack datasets, with the results shown in Table VI. Even when compared to the best-performing baseline,

[1]https://github.com/p4lang/behavioral-model
[2]http://mininet.org
[3]https://tcpreplay.appneta.com/

TABLE V
OVERALL COMPARISON OF DETECTION METRICS

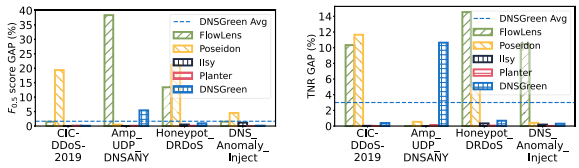
Method	Metric	DRDoS	RSD	NXDomain
FlowLens	$F_{0.5}$ score	0.8979▼9.42%	–	–
	TNR	0.9197▼6.9%	–	–
Poseidon	$F_{0.5}$ score	0.7852▼20.79%	0.9320▼5.30%	0.9250
	TNR	0.7110▼28.03%	0.9010▼8.79%	0.8801▼11.32%
IIsy	$F_{0.5}$ score	0.9048▼8.73%	0.9225▼6.27%	0.8865▼3.07%
	TNR	0.9520▼3.63%	0.9457▼4.26%	0.9343▼5.86%
Planter	$F_{0.5}$ score	0.9757▼1.57%	0.9378▼4.71%	0.8964▼1.99%
	TNR	0.9900	0.9312▼5.73%	0.9700▼2.27%
DNSGreen	$F_{0.5}$ score	0.9913▲1.60%	0.9842▲4.95%	0.9146
	TNR	0.9879	0.9878▲4.45%	0.9925▲2.32%

▼: Performance degradation compared to DNSGreen.
▲: Performance improvement compared to the best baseline.

DNSGreen outperforms it in most cases. The rule-based method Poseidon is simple to implement, requiring only basic counting operations, but it performs poorly across all datasets. We believe that such manually crafted heuristic rules can serve as a first line of defense against volumetric DDoS attacks, but additional traffic filtering tools are needed for further mitigation. Machine learning-based packet classification methods demonstrate better and more stable performance. However, in real-world applications, the effectiveness of supervised learning methods is highly dependent on the training dataset.

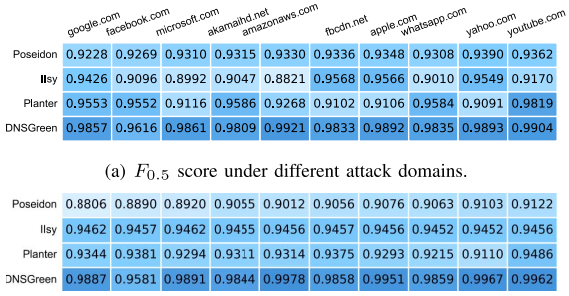
Second, we plot the gap between the best and worst performance metrics of each method in each case in Fig. 9. We observe that as the Confusing Normal Traffic Engagement Level increases, the performance of FlowLens and Poseidon significantly decreases. FlowLens performs anomaly detection based on the packet length distribution of flows, and its ability to make accurate judgments is affected when normal DNS packets and attack packets are mixed in a flow. As for Poseidon, it performs the worst in the DRDoS defense task. This is because the primitive Poseidon uses to defend against DNS amplification attacks is count(pkt.dport==53, [ip.src], duration), which allows traffic when there are corresponding query packets for DNS response packets within a certain time period (count([ip.dst]) $>$ 0), and drops otherwise. Therefore, when the victim communicates with the reflection server, the attack traffic is mistakenly allowed through. Moreover, we noticed that there are some IPs in the normal DNS background traffic that exhibit short-term one-way communication. This part of the traffic is mistakenly identified by Poseidon as attack reflection packets. However, DNSGreen sets a reasonable threshold for preliminary judgment and therefore does not classify them as suspicious flows.

2) RSD Attack Defense: First, random subdomain names were generated using a regular DGA algorithm to construct the attack dataset. The attack domains were selected from the 10 most frequently occurring domains in benign traffic. The results are shown in Fig. 10, where DNSGreen performs the best in this scenario, improving the $F_{0.5}$ score of baselines capable of detecting this attack by 4.64%—6.17% and increasing the TNR by 4.21%–8.67%. Moreover, we observed that each domain name has a different composition ratio in benign DNS traffic. However, DNSGreen remained stable and was not



(a) $F_{0.5}$ score gap. (b) TNR gap.

Fig. 9. Metrics gap of DNSGreen and baselines under DRDoS attack defense.



(a) $F_{0.5}$ score under different attack domains.

(b) TNR under different attack domain.

Fig. 10. Detection metrics of RSD attacks under regular DGAs.

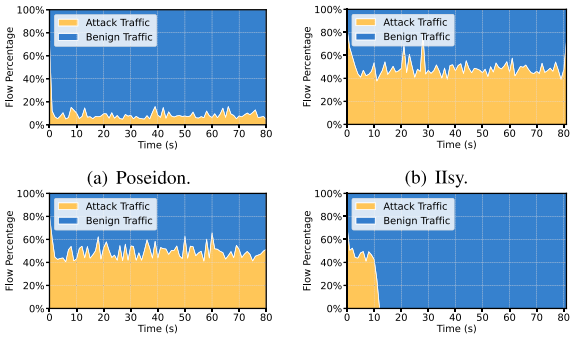
affected by these variations. One reason is that DNSGreen has strong gibberish detection capabilities in its data plane design, and another is that its whitelist collection mechanism effectively mitigates the impact of the increased Confusing Normal Traffic Engagement Level. In contrast, Poseidon applies rate limiting to DNS traffic of domains exceeding a threshold. As a result, when the attack domain has a high proportion in benign traffic, more benign DNS packets are mistakenly dropped, leading to a decrease in TNR. When the attack domain was set to *"google.com"*, the TNR dropped to as low as 88.06%. FlowLens fails to effectively identify attack traffic based on packet length distribution in this scenario.

Further, we used the DNSperf tool in combination with the Matsnu DGA to simulate an extreme RSD attack. The subdomains generated by Matsnu DGA are randomly composed of readable verbs and nouns, making them highly readable and difficult for random string detection algorithms to identify. In this scenario, machine learning-based methods such as IIsy and Planter lose their detection capability due to the failure of bigram features. Poseidon's rate-limiting defense strategy is independent of packet features and remains unaffected. To address this situation, DNSGreen can set up a simple counter to track the number of dropped packets. If the count value is too low, it indicates that the attacker has evaded the defense. At this point, DNSGreen stops using packet filtering for suspicious traffic. Instead, it only allows DNS packets that match the whitelist, ensuring a successful defense against this enhanced attack. Fig. 11 shows the traffic composition passing through the switch after deploying different methods. After a period of whitelist collection, DNSGreen can begin dropping attack packets. In contrast, IIsy and Planter can only filter a small portion of the attack traffic, failing to meet the bandwidth demands of normal traffic.

TABLE VI
PERFORMANCE COMPARISON OF DIFFERENT METHODS ON MULTIPLE DATASETS FOR DNS AMPLIFICATION ATTACKS

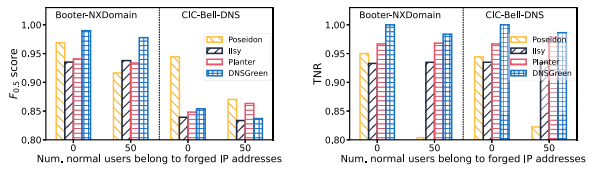
Method	Metric	CIC-2019-DDoS			Amp_UDP_DNSANY			Honeypot_DRDoS			DNS_Anomaly_Inject		
		Low[1]	Medium[2]	High[3]	Low	Medium	High	Low	Medium	High	Low	Medium	High
FlowLens	$F_{0.5}$ score	0.9890	0.9889	0.9742	0.9637	0.6763	0.5799	0.9687	0.8477	0.8345	0.9888	0.9892	0.9738
	TNR	0.9410	0.9386	0.8377	1.0000	1.0000	1.0000	0.9614	0.8241	0.8160	0.9404	0.9409	0.8364
Poseidon	$F_{0.5}$ score	0.8758[4]	0.7711	0.6820	0.8089	0.8136	0.8088	0.8039	0.7019	0.5918	0.8792	0.8518	0.8340
	TNR	0.8642	0.7637	0.7477	0.6695	0.6745	0.6691	0.7303	0.6821	0.6897	0.6794	0.6785	0.6827
IIsy	$F_{0.5}$ score	0.9518	0.9519	0.9515	0.7134	0.7134	0.7136	0.9907	0.9959	0.9960	0.9675	0.9557	0.9565
	TNR	0.9570	0.9570	0.9567	0.9725	0.9725	0.9725	0.9731	0.9764	0.9765	0.9044	0.9025	0.9029
Planter	$F_{0.5}$ score	0.9687	0.9671	0.9676	0.9716	0.9727	0.9719	0.9682	0.9672	0.9681	0.9953	0.9950	0.9954
	TNR	0.9966	0.9965	0.9964	0.9839	0.9853	0.9844	0.9923	0.9922	0.9917	0.9868	0.9868	0.9870
DNSGreen	$F_{0.5}$ score	0.9995	0.9997	0.9986	0.9967	0.9895	0.9423	0.9979	0.9885	0.9887	0.9987	0.9980	0.9973
	TNR	1.0000	0.9995	0.9961	1.0000	0.9824	0.8936	1.0000	0.9933	0.9932	1.0000	0.9995	0.9970

[1]: The victim is not communicating with the reflection server cluster.
[2]: The victim engages in a brief session with the reflection server cluster, with the communication volume not exceeding a total of 5%.
[3]: The victim is engaged in a prolonged communication, with the communication volume exceeding a total of 15%.
[4]:We highlight the best metric in ● and the worst metric in ●.



(a) Poseidon. (b) IIsy. (c) Planter. (d) DNSGreen.

Fig. 11. The proportion of attack traffic and benign traffic over time under the Matsnu DGA.



(a) $F_{0.5}$ score. (b) *TNR*.

Fig. 12. Detection results under two NXDomain attack datasets.

3) NXDomain Attack Defense: We conducted NXDomain attack experiments using two datasets: the Booter dataset modified with DGA and the CIC-Bell-DNS dataset. The results are shown in Fig. 12. When normal users are mixed with attacker-forged IP addresses, it indicates a higher Confusing Normal Traffic Engagement Level. Since the malicious domains in the CIC-Bell-DNS dataset originate from malware, spam, and phishing, their lower randomness increases the difficulty of detection. As a result, all feature-based analysis methods except Poseidon struggle with poor $F_{0.5}$ score. However, as the Confusing Normal Traffic Engagement Level increases, Poseidon still faces a significant decline in TNR due to rate-limiting constraints. In contrast, DNSGreen achieves the most effective defense while minimizing false positives for benign traffic.

C. Performance Results

1) Resource Consumption: Using the open-source Intel P4 Studio, we measured DNSGreen's memory usage, hash bits consumption, and the number of pipeline stages required for implementation on the TNA architecture. Detailed information is presented in Table VII.

TABLE VII
RESOURCE CONSUMPTION ANALYSIS FOR KEY COMPONENTS

Component	Flow Statistics	Z-Score	Gibberish Detection	Overall
TCAM (Kb)	42.78 (0.69%)	86.18 (1.39%)	279.62 (4.51%)	408.58 (6.59%)
SRAM (Mb)	10.87 (9.06%)	6.88 (5.73%)	4.00 (3.33%)	21.74 (18.12%)
Hash bits	140 (2.80%)	180 (3.60%)	150 (3.00%)	470 (9.41%)
Stages	8	10	5	10

2) Lantency: By using the egress and ingress timestamps, we measured the latency of DNSGreen and the in-network machine learning scheme Planter, as shown in Fig. 13. DNSGreen introduces lower latency compared to both implementations of Planter. When Planter deploys the binary neural network (BNN), the extensive XNOR and PopCount operations result in unacceptable latency, causing the switch queues to fill up. In fact, Planter claims that not all programmable network devices can realize the BNN model.

D. In-depth Analysis

1) Alarm Threshold Setting: Taking the DRDoS attack detection as an example, the focus is on the number of DNS response packets sent to a target IP. To address potential traffic pattern drift over time, we analyzed DNS traffic in different time segments and counted the number of DNS response packets for each target IP within each time window. The results show that the average value is 5.4, the 95th percentile is 14.0, the 99th percentile is 112.0, and the maximum value is 833. The DRDoS

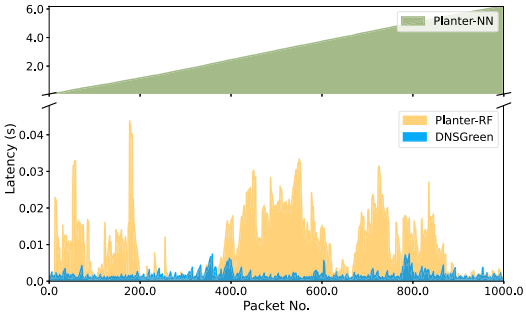


Fig. 13. Comparison of latency introduced by DNSGreen and planter in the BMv2 switch.

attack alarm threshold is set to start from 100 and increases in steps of 100. Recall, precision, and F1-score are calculated using the attack window as the positive sample and the benign window as the negative sample, and the trend of these metrics is plotted in Fig. 14(a). When the threshold reaches 400, the detection performance stabilizes.

Additionally, different thresholds correspond to different BeauCoup query parameters (m_q, n_q, p_q), resulting in variations in average memory access times. This is a significant limiting factor for the performance of network measurement tasks executed on P4 switches. Therefore, higher thresholds are preferable, as they generally lead to fewer memory accesses. However, overly high thresholds may impose stricter requirements on the uniformity of hash functions, requiring a trade-off. When switching application scenarios, such as in high-throughput and low-latency 5G core network environments, we recommend setting an appropriate threshold based on the above rules to better leverage the advantages of DNSGreen.

2) The Length of Registers: The flow states in DNSGreen are stored using registers, and the storage index is generated by hashing the flow identity (q, key_q), introducing a certain probability of hash collisions. As shown in Fig. 14(b), the average $F_{0.5}$ score and index size exhibit a positive correlation in three attack detection scenarios. When the number of registers is too small, the collision probability increases, causing BeauCoup to generate false alarms and blacklisting normal flows, resulting in a lower average $F_{0.5}$ score.

3) Whitelist Collection: We implement the whitelist collection module using three methods: common version 1, common version 2, and bit-string version. The key difference between common version 1 and 2 lies in calculating the reflection factor. Common version 1 uses dichotomization to find the position of the highest 1 and approximates division based on their distance. Common version 2 converts division into ln and exp operations $(a/b = exp(lna - lnb))$ and stores precomputed values in a table. As shown in Fig. 14(c), the blue line represents the baseline, which records packet timestamps at the egress pipeline during basic forwarding. With the addition of whitelist collection, the bit-string version introduces minimal processing delay, easing the burden on line-speed forwarding.

We also discuss the waiting period for whitelist collection. The packet-filtering stage is carried out only after the whitelist collection is complete, so attack traffic is unconditionally allowed during the collection period. As shown in Fig. 14(d), with the increase in collection time, the number of whitelisted entries increases, but at the same time, more malicious traffic will be released. In actual deployment, a trade-off between retrieving benign traffic and allowing attack traffic needs to be considered. For whitelist refreshing, DNSGreen reconstructs the whitelist upon detecting an attack and functions as a short-term on-demand mechanism. For a long-term whitelist in a running system, an approximate whitelist can be maintained using a lossy hash table with multiple layers of least recently used caches. Newly added entries gradually evict long-unused ones, ensuring adaptability to evolving network conditions. This dynamic maintenance mechanism helps mitigate security risks caused by aging static whitelists.

4) Security Analysis: The potential security risks that DNSGreen may face can be categorized into two types. Firstly, adversaries may attempt to flood the switch with a large number of flows to exhaust its resources and disrupt the detection ability of the system. However, BeauCoup only records a flow and allocate storage units for it when it hits the coupon, and the hitting probability is extremely low. In addition, to further enhance the system's resilience, the counting strategy can be adjusted, for example, by not recording on the first hit and starting recording from the second hit.

The second type of risk is attackers trying to escape detection. If the attacker disguises as normal mode during the whitelist collection phase, it might be marked as whitelist, and subsequent packets in this flow will not be inspected and filtered. This will face several obstacles. Firstly, attackers are not clear when the attack will be detected, and when the whitelist collection phase will be initiated which is very brief. And attackers attempting to forge packets sent in the resolver direction can be discovered by the switch's built-in port rules. Finally, even for flows added to the whitelist, if they exhibit suspicious behavior over time, BeauCoup will still issue an alert and remove them from the whitelist.

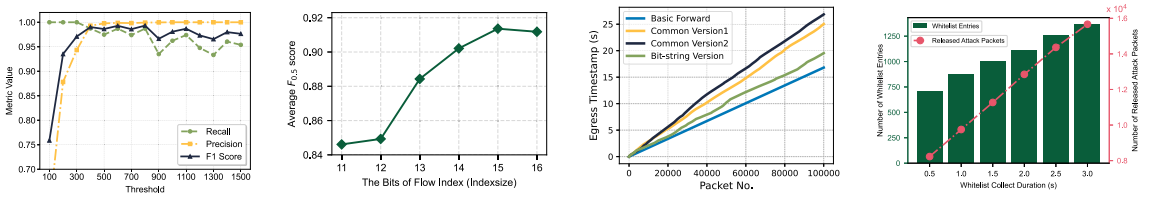
VI. Related Work

A. DNS Exploration in Programmable Switches

The critical information of DNS packets lies in the application layer, and handling variable-length domain names poses a challenge for the parser that can only extract fixed-length fields. Meta4 [35], P4DNS [36], and P4DDPI [37] have implemented domain name extraction using different methods. Currently, the most sophisticated approach [28] is to set multiple transition states for each label with exponential lengths of 2, allowing for complete domain name extraction without damaging packets or degrading switch performance. Building upon these works, WORD [38] divides parent domain and subdomain according to the Public Suffix List to detect random subdomain attacks. PINOT [39] encrypts IPv4 addresses of DNS packets into IPv6 addresses through the lightweight 2EM algorithm adapted to the switch architecture without altering the protocol, thus strengthening the privacy protection of DNS.

B. DNS DDoS Attack Defense

DNS amplification attack defense mainly relies on monitoring the legitimacy and rationality of request–response connections [40], often by managing states and cutting off illegal



(a) Threshold settings. (b) The impact of index size. (c) Bit-string version VS. Common version. (d) The impact of whitelist collection duration.

Fig. 14. In-depth exploration of DNSGreen.

connections with server-side cookies. Unlike traditional solutions, Poster [41] detects amplification traffic by parsing packet payloads, extracting frequency features, and setting thresholds on critical frequency bands. For RSD and NXDomain attacks, many studies [20], [42] focus on the lexical characteristics of DGA domain names based on anomalies in domain fields. To overcome the limitations of relying on local domain name features, graph inference techniques establish associations between domains, identifying strongly associated domains as malicious based on known samples [43]. For resource-constrained IoT networks, WaterPurifier [44] proposes a collaborative and hierarchical defense system that executes lightweight functions, asynchronous communication mechanisms, and real-time training processes at different layers. Despite extensive research, significant potential remains in defense coverage, response speed, and adaptability to diverse deployment environments.

VII. Conclusion

Moving DNS security policies to high-performance programmable switches can alleviate the burden on servers and mitigate the damage from attacks in advance. In this paper, we propose DNSGreen, a comprehensive in-band defense system against bounce-style DNS DDoS attacks with easy scalability. It can monitor multiple attacks simultaneously at the flow level and inspect packets in suspicious flows with fine granularity, allowing benign packets to pass through as far as possible. Along with the whitelist collection module and the improved counting structures, DNSGreen achieves detailed optimization in both packet filtering and flow statistics. Deployment and experimental verification demonstrate that DNSGreen is an excellent defense system that protects the rights of normal users well. In the future, we will strive to deploy it in practice and explore solutions to other types of DNS security issues in the programmable data plane.

References

[1] Ö. Kasim, "A robust DNS flood attack detection with a hybrid deeper learning model," *Comput. Electr. Eng.*, vol. 100, May 2022, Art. no. 107883.
[2] S. Chen, B. Lang, H. Liu, D. Li, and C. Gao, "DNS covert channel detection method using the LSTM model," *Comput. Secur.*, vol. 104, May 2021, Art. no. 102095.
[3] F. Guo, J. Chen, and T.-c. Chiueh, "Spoof detection for preventing dos attacks against DNS servers," in *Proc. 26th IEEE Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Piscataway, NJ, USA: IEEE Press, 2006, pp. 37–37.
[4] T. Zebin, S. Rezvy, and Y. Luo, "An explainable AI-based intrusion detection system for DNS over HTTPS (DoH) attacks," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 2339–2349, 2022.
[5] L. Chen, Y. Zhang, Q. Zhao, G. Geng, and Z. Yan, "Detection of DNS DDOS attacks with random forest algorithm on spark," *Procedia Comput. Sci.*, vol. 134, pp. 310–315, 2018.
[6] D. Tang, R. Dai, Y. Yan, K. Li, W. Liang, and Z. Qin, "When SDN meets low-rate threats: A survey of attacks and countermeasures in programmable networks," *ACM Comput. Surv.*, vol. 57, no. 4, pp. 1–32, 2024.
[7] D. Tang, R. Dai, C. Zuo, J. Chen, K. Li, and Z. Qin, "A low-rate DoS attack mitigation scheme based on port and traffic state in SDN," *IEEE Trans. Comput.*, vol. 74, no. 5, pp. 1758–1770, May 2025.
[8] M. Zhang et al., "Poseidon: Mitigating volumetric DDoS attacks with programmable switches," in *Proc. 27th Netw. Distrib. System Secur. Symp. (NDSS)*, 2020.
[9] Z. Liu et al., "Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric DDoS attacks with programmable switches," in *Proc. 30th USENIX Secur. Symp. (USENIX Secur.)*, 2021, pp. 3829–3846.
[10] L. Tang, Q. Huang, and P. P. Lee, "A fast and compact invertible sketch for network-wide heavy flow detection," *IEEE/ACM Trans. Netw.*, vol. 28, no. 5, pp. 2350–2363, Oct. 2020.
[11] Â. C. Lapolli, J. A. Marques, and L. P. Gaspary, "Offloading real-time DDoS attack detection to programmable data planes," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, Piscataway, NJ, USA: IEEE Press, 2019, pp. 19–27.
[12] J. Xing, W. Wu, and A. Chen, "Ripple: A programmable, decentralized link-flooding defense against adaptive adversaries," in *Proc. 30th USENIX Secur. Symp. (USENIX Secur.)*, 2021, pp. 3865–3881.
[13] C. Zheng et al., "Planter: Rapid prototyping of in-network machine learning inference," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 54, no. 1, pp. 2–21, 2024.
[14] T. Swamy, A. Zulfiqar, L. Nardi, M. Shahbaz, and K. Olukotun, "Homunculus: Auto-generating efficient data-plane ml pipelines for datacenter networks," in *Proc. 28th ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, vol. 3, 2023, pp. 329–342.
[15] T. Swamy, A. Rucker, M. Shahbaz, I. Gaur, and K. Olukotun, "Taurus: A data plane architecture for per-packet ML," in *Proc. 27th ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2022, pp. 1099–1114.
[16] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? Toward in-network classification," in *Proc. 18th ACM Workshop Hot Topics Netw.*, 2019, pp. 25–33.
[17] Q. Qin, K. Poularakis, K. K. Leung, and L. Tassiulas, "Line-speed and scalable intrusion detection at the network edge via federated learning," in *Proc. IFIP Netw. Conf. (Netw.)*, Piscataway, NJ, USA: IEEE Press, 2020, pp. 352–360.
[18] X. Chen, S. Landau-Feibish, M. Braverman, and J. Rexford, "Beaucoup: Answering many network traffic queries, one memory update at a time," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Archit., Protocols Comput. Commun.*, 2020, pp. 226–239.
[19] C. Fachkha, E. Bou-Harb, and M. Debbabi, "Fingerprinting internet DNS amplification DDoS activities," in *Proc. 6th Int. Conf. New Technol., Mobility Secur. (NTMS)*, Piscataway, NJ, USA: IEEE Press, 2014, pp. 1–5.
[20] X. Luo, L. Wang, Z. Xu, K. Chen, J. Yang, and T. Tian, "A large scale analysis of DNS water torture attack," in *Proc. 2nd Int. PoseidonConf. Comput. Sci. Artif. Intell.*, 2018, pp. 168–173.
[21] G. Liu et al., "Dial 'n' for NXDOMAIN: The scale, origin, and security implications of DNS queries to non-existent domains," in *Proc. ACM Internet Meas. Conf.*, 2023, pp. 198–212.
[22] D. Tang, Y. Yan, S. Zhang, J. Chen, and Z. Qin, "Performance and features: Mitigating the low-rate TCP-targeted DoS attack via SDN," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 1, pp. 428–444, Jan. 2022.
[23] D. Tang, Y. Yan, C. Gao, W. Liang, and W. Jin, "LtRFT: Mitigate the low-rate data plane DDoS attack with learning-to-rank enabled flow tables," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 3143–3157, 2023.

[24] S. Deng, X. Gao, Z. Lu, Z. Li, and X. Gao, "DoS vulnerabilities and mitigation strategies in software-defined networks," *J. Netw. Computer Appl.*, vol. 125, pp. 209–219, Jan. 2019.
[25] D. Tang, B. Liu, K. Li, S. Xiao, W. Liang, and J. Zhang, "PLUTO: A robust LDoS attack defense system executing at line speed," *IEEE Trans. Dependable Secure Comput.*, vol. 22, no. 3, pp. 2855–2872, May/Jun. 2025.
[26] F. Li et al., "Distributed program deployment for resource-aware programmable switches," *IEEE Trans. Comput.*, vol. 73, no. 5, pp. 1357–1370, May 2024.
[27] J. P. G. van Brakel, "Robust peak detection algorithm using z-scores," *Stack Overflow.* [Online]. Available: https://stackoverflow.com/questions/22583391/peak-signal-detection-in-realtimetimeseries-data/22640362#22640362 (version: 2020-11-08), 2014.
[28] A. Kaplan and S. L. Feibish, "Practical handling of DNS in the data plane," in *Proc. Symp. SDN Res.*, 2022, pp. 59–66.
[29] K. Cho, K. Mitsuya, and A. Kato, "Traffic data repository at the WIDE project," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2000, pp. 263–270.
[30] M. Singh, "10 days DNS network traffic from 2016," Mendeley Data 2, 2019.
[31] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy," in *Proc. Int. Carnahan Conf. Secur. Technol. (ICCST)*, Piscataway, NJ, USA: IEEE Press, 2019, pp. 1–8.
[32] J. J. Santanna, et al., "Booters–An analysis of DDoS-as-a-service attacks," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, Piscataway, NJ, USA: IEEE Press, 2015, pp. 243–251.
[33] S. Mahdavifar, N. Maleki, A. H. Lashkari, M. Broda, and A. H. Razavi, "Classifying malicious domains using DNS traffic analysis," in *Proc. IEEE Int. Conf Dependable, Autonomic Secure Comput., Int. Conf Pervasive Intell. Comput., Int. Conf Cloud Big Data Comput., Int. Conf. Cyber Sci. Technol. Congr. (DASC/PiCom/CBDCom/CyberSciTech)*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 60–67.
[34] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. Ramos, and A. Madeira, "FlowLens: Enabling efficient flow classification for ML-based network security applications," in *Proc. NDSS*, 2021.
[35] J. Kim, H. Kim, and J. Rexford, "Analyzing traffic by domain name in the data plane," in *Proc. SIGCOMM Symp. SDN Res.*, 2021, pp. 1–12.
[36] J. Woodruff, M. Ramanujam, and N. Zilberman, "P4DNS: In-network DNS," in *Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, Piscataway, NJ, USA: IEEE Press, 2019, pp. 1–6.
[37] A. AlSabeh, E. Kfoury, J. Crichigno, and E. Bou-Harb, "P4DDPI: Securing p4-programmable data plane networks via DNS deep packet inspection," in *Proc. NDSS Symp.*, 2022.
[38] A. Kaplan and S. L. Feibish, "DNS water torture detection in the data plane," in *Proc. SIGCOMM Poster Demo Sessions*, 2021, pp. 24–26.
[39] L. Wang, H. Kim, P. Mittal, and J. Rexford, "Programmable in-network obfuscation of DNS traffic," *NDSS: DNS Privacy Workshop*, 2021.
[40] Y. Dai, T. Huang, and S. Wang, "DAmpADF: A framework for DNS amplification attack defense based on Bloom filters and NAmpKeeper," *Comput. Secur.*, vol. 139, Apr. 2024, Art. no. 103718.
[41] S. M. H. Mirsadeghi, "Poster: In-switch defense against DNS amplification DDoS attacks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2024, pp. 4964–4966.
[42] T. Yoshida, K. Kawakami, R. Kobayashi, M. Kato, M. Okada, and H. Kishimoto, "Detection and filtering system for DNS water torture attacks relying only on domain name information," *J. Inf. Process.*, vol. 25, pp. 854–865, 2017.
[43] I. Khalil, T. Yu, and B. Guan, "Discovering malicious domains through passive DNS data graph analysis," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, 2016, pp. 663–674.
[44] L. Yin, M. Zhu, W. Liu, X. Luo, C. Wang, and Y. Li, "WaterPurifier: A scalable system to prevent the DNS water torture attack in 5G-enabled SIoT network," *Comput. Commun.*, vol. 199, pp. 186–195, Feb. 2023.

Dan Tang received the Ph.D. degree from Huazhong University of Science and Technology, in 2014. He is an Associate Professor with the College of Cyber Science and Technology, Hunan University (HNU), Changsha, China. His research interests include the areas of computer network security, computer information security, and architecture of future Internet.

Xiaocai Wang received the B.E. and M.S. degrees in computer science and technology from Hunan University, in 2022 and 2025, respectively. Her research directions include cyber-space security and programmable network.

Pei Tan received the B.E. degree in computer science and technology in 2024, from Hunan University, where she is currently working toward the Ph.D. degree with the College of Cyber Science and Technology. Her research directions include programmable network and network security.

Zheng Qin received the Ph.D. degree in computer software and theory from Chongqing University, China, in 2001. He is currently a Professor of computer science and technology with Hunan University, China. He has accumulated rich experience in product development and application services such as in the area of financial, medical, military. His research interests include computer network and information security, cloud computing, big data processing and software engineering. He is a member of the China Computer Federation (CCF) and ACM.

Keqin Li (Fellow, IEEE) is a SUNY Distinguished Professor of computer science with the State University of New York and also a National Distinguished Professor with Hunan University. His research interests include cloud computing, high-performance computing, computer networking, and machine learning. He is listed in Scilit Top Cited Scholars (2023–2025) and is among the top 0.02% out of over 20 million scholars worldwide based on top-cited publications in the last ten years. He is an AAAS Fellow and an AIIA Fellow. He is a member of the European Academy of Sciences and Arts. He is a member of Academia Europaea (Academician of the Academy of Europe).

Jiliang Zhang (Senior Member, IEEE) joined the Integrated Circuit Science and Engineering College of Nanjing University of Posts and Telecommunications, currently serving as the College's President, in 2025, and is also the Chair of CCF Fault-Tolerant Computing Professional Committee. He works on power/energy efficiency and security problems in the design of integrated circuits (IC), processors, and the Internet of Things. He has authored more than 100 technical papers in leading journals and conferences. He was the recipient of the CCF Integrated Circuit Early Career Award and the Winner of Excellent Youth Fund of the National Natural Science Foundation of China. He has been the Program Committee Member for a number of well-known conferences, such as DAC, ASP-DAC, GLSVLSI, and FPT. He served as the Associate Editor for some scientific journals, including IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I: REGULAR PAPERS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS PART II: EXPRESS BRIEFS, and *JEIT*.