DDD: A DNS-based DDoS Defense Scheme Using Puzzles

Hyeonmin Lee* Taehyun Kang Sukhun Yang Jinyong Jun Taekyoung Kwon†
*University of Virginia, Charlottesville, VA, United States
Seoul National University, Seoul, Republic of Korea
*frv9vh@virginia.edu {gangtaeng_parangvo, sukhuny, junjinyong, †tkkwon}@snu.ac.kr

***Abstract*—Distributed Denial-of-Service (DDoS) attacks have remained a significant threat to the Internet for years. One strategy for mitigating these attacks involves requiring clients to solve cryptographic puzzles to control the rate of incoming traffic to a target server. For such a puzzle-based DDoS defense mechanism to be effective, it necessitates robust methods for both distributing puzzles to clients and adjusting puzzle difficulty. In this paper, we introduce a puzzle-based DDoS defense mechanism, DDD, which utilizes the Domain Name System (DNS) for distributing puzzles to clients. The target server disseminates its puzzles by publishing them as a DNS record through its authoritative name server, distributing puzzles to clients via their DNS resolvers. Our design incorporates a monitoring server that continuously monitors incoming traffic to the target and dynamically adjusts puzzle difficulty based on the traffic originating from each Autonomous System (AS). This enables AS-specific puzzle difficulty customization, and consequently, traffic control. We have implemented our design into the Linux kernel and showcased its effectiveness in traffic control through prototype-based and controlled experiments.**

***Index Terms*—Distributed Denial-of-Service, Domain Name System, Client Puzzle**

I. Introduction

Distributed Denial-of-Service (DDoS) attacks have been a persistent and substantial threat to the security of Internet infrastructure over a couple of decades. Such attacks are orchestrated with the intent to render a victim (e.g., machine or network resources) inaccessible to its users, often driven by motives such as extortion, revenge, or political agendas [30], [7], [16], [36], [14]. Even DDoS-as-a-service has become available, enabling on-demand DDoS attacks [29]. Notably, the frequency and intensity of DDoS attacks are increasing, with 7.9 million attacks recorded in 2023, marking a 31% increase from the previous year [20].

To mitigate DDoS attacks, various defense mechanisms have been introduced, categorized by multiple criteria [40]. Let us first categorize the DDoS countermeasures based on their deployment locations. *Source-based* approaches [17], [18] try to perform countermeasures near the sources of an attack (e.g., at routers near bots) by monitoring traffic in a network. However, detecting attack traffic near its sources presents a challenge, as the volume typically remains insignificant until it accumulates along the paths toward the victim. *Destination-based* approaches [6], [2], [33], [15] try to mitigate the attack traffic near a victim. These countermeasures usually cannot be triggered before the attack traffic reaches a victim, resulting in a waste of resources along the paths to the victim.

As cloud computing becomes increasingly popular and cost-effective, *cloud-based* DDoS protection services [12] have gained significant momentum. Such service providers rely on globally deployed servers (e.g., edge servers in content delivery networks) to absorb and filter suspicious traffic, forwarding only legitimate (or "scrubbed") traffic to the target server. However, the details of how to mitigate and filter the attack traffic are not publicly disclosed. Considering that a cloud network intervenes between the sources and the destination, we could classify such proposals as a *network-based* approach. Other network-based approaches [5], [22], [23], [19] exist, but these methods often require significant storage and processing overheads at routers, rendering them impractical for deployment.

All of the above approaches typically operate in a centralized manner, in the sense that deciding the onset of attacks and triggering countermeasures are typically performed at a *single* point. Note that monitoring points may be distributed to a certain degree.

By contrast, *distributed* approaches to DDoS defenses have their components located at multiple points, which collaborate with one another. For instance, detection may occur at the destination-side (i.e., near victims), while the defense is executed at the source-side (i.e., near bots). This strategy offers the advantage of more collaborative and coordinated reactions, potentially enhancing the effectiveness of the DDoS mitigation mechanism. For this reason, various such approaches have been proposed [3], [38], [4], [39]. Such methods usually rely on packet filters and inspections, which are performed at network points, such as routers. Thus, they often impose significant memory/processing demands on the routers. Additionally, the cooperation and communication among distributed components increase the complexity and overhead.

One interesting approach working in a distributed manner is a *puzzle-based* DDoS defense mechanism, which employs *puzzles* to prevent clients from generating attack packets continuously. Such approaches necessitate clients to solve cryptographic puzzles, involving computationally intensive

*This work was conducted while the first author was a postdoctoral researcher at Seoul National University.
†Corresponding author.

operations, before being allowed to send a new eligible request to a server. Essentially, this approach mandates clients to demonstrate their eligibility by computing a solution for a given puzzle for each connection or request.

In this approach, a benign client may experience few disruptions since it is unlikely to send many requests continuously. On the other hand, a bot, upon a command from its control point, tries to send a large volume of requests to the target. To make a puzzle-based DDoS defense effective, efficient and resilient mechanisms to distribute puzzles to clients and to adjust puzzle difficulties are needed. Yet, most of the prior work has inadequately addressed these challenges, or they often concentrate on partial solutions (to be discussed in Section II-A).

In this paper, we propose a **D**NS-based **D**DoS **D**efense (**DDD**) scheme, leveraging the Domain Name System (DNS) to distribute puzzles to clients in a resilient manner against attacks. Our design primarily aims at transport-level or application-level flooding attacks, particularly those initiated with TCP SYN packets. These attacks are categorized as exhaustive DDoS, as they overwhelm a target server's resources. In our design, a target's monitoring server uploads a puzzle to its authoritative name server as a new DNS record, which is then subsequently distributed to DNS resolvers that are queried by the clients[1].

To send a request to be accepted by the target server, a client retrieves its puzzle from its DNS resolver and presents its solution to the target by sending a TCP SYN packet including the solution. Moreover, we adopt a DNS-based and "out-of-band" control approach, where a monitoring server (typically under the control of the same administrator of the target) continuously monitors the rate of TCP SYN packets towards the target. Depending on the rate of attack traffic to the target, its monitoring server adjusts the difficulty levels of puzzles. Then the authoritative name server of the target distributes the puzzles to clients (via local DNS resolvers).

Key contributions. Our approach makes the following contributions to address the limitations of the prior trials to mitigate DDoS attacks:

1. DDD relies on DNS, a critical component of the Internet infrastructure, to distribute puzzles to clients via their DNS resolvers. Leveraging DNS enhances the resilience to the attack traffic. That is, the DNS authoritative server could be available even if the resources of the target server are depleted due to the attack traffic. To successfully compromise our method, adversaries need to attack both the target and its DNS server (which can be hosted in a remote and distributed fashion) simultaneously, raising the bar for potential attacks.
2. We offer a practical method to adjust the difficulty of puzzles through a closed-loop control mechanism. Our design incorporates a monitoring server, which continuously measures the rate of incoming TCP SYN packets from each Autonomous System (AS). Upon detecting a significant increase in traffic, it dynamically adjusts the difficulty of puzzles depending on the current TCP SYN traffic from each AS.
3. DDD is integrated into the TCP stack of the Linux kernel, extending the *Options* field in the TCP header. We conduct prototype-based experiments to showcase the efficacy of DDD in regulating incoming traffic on a testbed.

Public release. To enable researchers to reproduce our work and build upon it, we publicly release our code.[2]

II. Related Work and Background

A. Puzzles

The concept of requiring clients to solve cryptographic puzzles was proposed over two decades ago [13]. Since then, many puzzle-based schemes have been studied, aiming at mitigating DDoS attacks.

Dean *et al.* [9] use a dedicated message exchange for puzzle dissemination and resolution, which increases the control traffic and latency. Moreover, it cannot adjust puzzle difficulty levels. Other approaches rely on third-party intermediaries [35] or direct transmissions of a code snippet (e.g., Java Applet [37]) to clients. However, both require placing additional trust in third-party entities. Wang *et al.* proposed Congestion Puzzles (CP) [34], which distributes puzzles via ICMP and adjusts difficulty at network routers, which increases the overhead of routers. Noureddine *et al.* [21] proposed a game-theoretical model for selecting puzzle difficulties. To send a puzzle and its solution, they exploited TCP SYN-ACK and ACK packets, respectively. However, the target server's direct handling of puzzle distribution may have an availability issue, particularly when its resources are depleted due to attacks. Unlike these approaches, DDD does not require trusts on third-parties or cooperations of network routers.

Portcullis [24] exhibits similarities to our design by utilizing DNS to distribute puzzles to clients. Portcullis however relies on a third-party entity, the seed generator, to generate puzzle seeds. These seeds are then disseminated to top-level domain (TLD) name servers. Clients fetch puzzles from the TLD servers and then send TCP packets containing the puzzle solutions. Routers en route verify the solutions using the puzzles retrieved from the TLD servers, ensuring the validity of the solution within the TCP packet from a client.

DDD is different from Portcullis [24] in multiple aspects. *First*, while Portcullis relies on a third party such as the seed generator, DDD eliminates the need for a third party. *Second*, DDD uses the target server's authoritative name server for puzzle seed distribution, which could be available even if the target is under a heavy attack. By contrast, Portcullis, relying on TLD servers for puzzle distribution, imposes substantial overhead on these servers. *Third*, while Portcullis cannot adjust the puzzle difficulty, DDD has a monitoring server that tracks traffic to the target and dynamically adjusts puzzle difficulty

[1]In this paper, clients are either benign clients or bots.

[2]https://github.com/Sagit25/DNS-based-DDoS-mitigation

to mitigate the attack traffic. DDD also supports difficulty adjustments for each AS, enabling fine-grained control based on real-time monitoring data. *Fourth*, in DDD, verification of puzzle solutions is performed by the target. Whereas, Portcullis requires verification to be conducted on routers, which requires changes to routers and burdens routers substantially. *Lastly*, DDD is implemented within the Linux TCP stack and tested in a lab environment, in contrast to Portcullis' simulation-based evaluation.

B. Counting Bloom Filter (CBF)

A *Counting Bloom Filter* (CBF) is a probabilistic data structure used for checking whether an element belongs to a set. If an element is added to the set, the CBF approximates its membership by incrementing the corresponding buckets (or counters) by one. It decrements the corresponding buckets by one when the element is removed. When searching for an element, it checks if all of the corresponding buckets are non-zero. In DDD, we employ a CBF to verify the authenticity of tokens presented by clients, ensuring they are legitimately issued by the monitoring server, and to provide protection against replay attacks. That is, the target keeps a CBF for each local DNS resolver, to be detailed later.

When employing a CBF, it is crucial to minimize the probabilities of false positives and false negatives in membership checking to reduce the risk of attacks on the validation of puzzle solutions. With the assumption of k hash functions, m buckets, and sufficiently large bucket sizes, when y elements are added, the probability that a given bucket is not incremented is $(1-\frac{1}{m})^{ky}$. Thus, the probability that any given bucket has been incremented and thus has a value greater than zero is $1-(1-\frac{1}{m})^{ky}$. Consequently, the false positive probability (α) is $(1-(1-\frac{1}{m})^{ky})^k$. Similarly, when x elements are removed, the expected number of counters that decrease by one or more is $1-(1-\frac{1}{m})^{kx}$. Thus, if w elements have been "accepted" by false positives (and hence removed from the set), the false negative probability (β) is $(1-(1-\frac{1}{m})^{kw})^k$. We provide further details on its mathematical properties in our design in Section III-B1a.

III. DDoS Traffic Control by DDD

A. Design Overview

*1) **Threat model:*** We consider transport-level and application-level flooding attacks, particularly those initiated with TCP SYN packets. The main objective of DDD is to control the rate of TCP SYN packets originated from each AS. Also, we assume that bots are globally distributed; however, the number of bots within individual ASes may vary significantly, depending on factors such as how their customers access malicious servers and how their network administrators manage their hosts.

*2) **Participating entities:*** On the target server-side, there are two more entities: the *monitoring server* and the *authoritative name server*. On the client-side, there is also a *local*

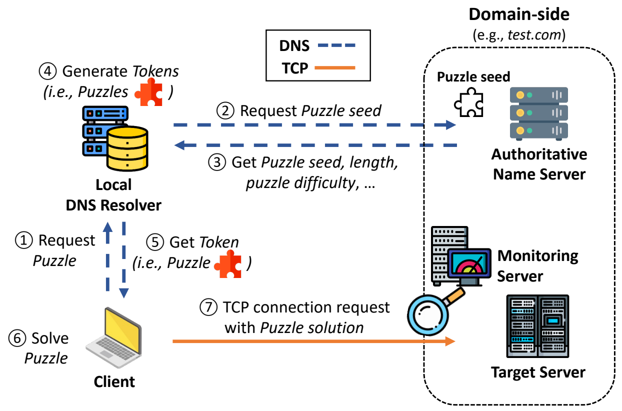

Fig. 1. An overview of the DDD workflow is shown.

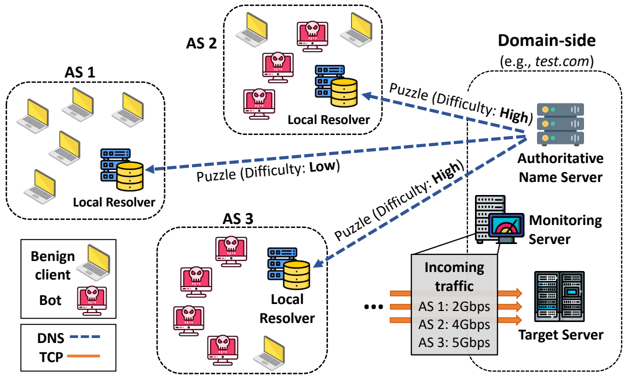

Fig. 2. DDD can adjust the difficulty level of puzzles for each AS based on the normal traffic from the AS.

DNS resolver. Here, we assume that each AS has a single local DNS resolver[3].

*3) **Operational flows:*** DDD's operations are illustrated in Figure 1. Initially, the monitoring server uploads a puzzle seed to its authoritative name server as a DNS record; we assume a new DNS record type (say, PUZZLE record). To establish a connection with the server, a client queries its local DNS resolver to retrieve the server's puzzle (step ① in Figure 1). The local DNS resolver, if no available puzzles, fetches the puzzle seed data from the target's authoritative name server and generates tokens (i.e., puzzles), which are distributed to its clients (steps ②∼⑤ in Figure 1). Note that each client will get a distinct puzzle. The puzzle distribution process will be detailed in Section III-B. The client then solves the received puzzle and submits the solution in its TCP SYN packet to the target (steps ⑥∼⑦ in Figure 1). If the solution is correct, the target establishes a TCP session with the client; otherwise, the TCP connection is aborted. Puzzle solving and solution verification will be covered in Section III-C.

The monitoring server keeps track of the incoming packets towards the target, classifying them by their source IP addresses (more precisely, their AS numbers). When flooding

[3]DDD can be easily extended to accommodate for an AS with multiple local DNS resolvers.

packets are arriving, it dynamically adjusts the difficulty of the puzzle for each AS comparing the flooding traffic and normal traffic. For instance, under normal operations, the monitoring server observes the number of TCP SYN packets per time unit from each AS, which would serve as a reference for DDoS attack detection and traffic control. If the rate of TCP SYN packets originating from certain ASes significantly exceeds the normal traffic rates (e.g., ASes 2 and 3 in Figure 2), the puzzle difficulty levels for those ASes are increased. This means that clients in an AS with few bots need to solve easy puzzles, resulting in minimal disruption. How to adjust puzzle difficulty for each AS is detailed in Section III-D.

The following sections will elaborate on the three main mechanisms of DDD: (1) puzzle generation by the monitoring server and distribution from the authoritative name server to clients via local DNS resolvers (Section III-B); (2) solving puzzles by clients and verifying solutions by the target (Section III-C); and (3) adjusting the puzzle difficulty based on current network traffic by the monitoring server (Section III-D).

B. Generating and Distributing Puzzles

A DNS resolver retrieves puzzle seed data from the authoritative name server of a target, and generates puzzles from the seed, which are then distributed to its clients as a DNS record (e.g., PUZZLE record). In DDD, an authoritative name server of the target has puzzle seed data for each AS (see Section III-B1). The seed data is then disseminated to the local DNS resolvers of the ASes. Upon receiving puzzle seed data, the DNS resolver generates a sequence of puzzles (e.g., using a hash chain). These puzzles are to be fetched by its clients (see Section III-B2).

1) *Puzzle seed distribution to local resolvers:*

a) ***Generating puzzle seed:*** A puzzle seed data for an AS consists of the puzzle seed, the number of puzzles, and the puzzle threshold (i.e., difficulty). Here, the number of puzzles is the length of the hash chain (to be generated from the seed). For each AS, a monitoring server tracks the incoming rate of TCP SYN packets.[4] Using this information, it generates a seed randomly and calculates the length of a hash chain for each AS. The length is the product of the "normal" TCP incoming rate and the seed update interval (say 5 minutes). The puzzle seed, the length of its hash chain, and the puzzle threshold (see III-D) for each AS are sent to the authoritative name server, which are to be disseminated to the local resolver of the AS.

A bot could potentially execute the following attacks: (i) reusing already solved puzzle tokens (i.e., replay attacks), (ii) fabricating spoofed puzzles, or (iii) selecting a nonce randomly without solving puzzles. To prevent the first two attacks, the target leverages a CBF (for each AS) generated by the monitoring server, to be discussed later.

Note that when all hash tokens are exhausted, they are to be replenished, and the CBF is also updated. All the analyses are conducted under steady-state conditions. The difficulty adjustment policy aims to sustain the rate of (previously measured) normal traffic from an AS by setting the difficulty level such that the current inter-arrival time between two successive TCP SYN packets from the AS equals the reciprocal of the rate.

Suppose the monitoring server monitors the TCP traffic from AS i. Let us denote the traffic by normal users in AS i as n_i and traffic from attacking users as a_i[5]. Then total traffic is $n_i + a_i$, and the puzzle difficulty for AS i is calculated so as to make the total incoming traffic reduced by $\frac{n_i}{n_i+a_i}$. By adjusting the difficulty level so as to reduce the total traffic by this ratio, DDD can achieve the traffic control in such a way that an AS with more bots (or more attack traffic) will get more penalties.

Configuring a CBF. Considering the probabilistic nature of a CBF, it must be designed to meet the upper bounds on false positive and false negative ratios, in accordance with the server's operational requirements. Suppose a CBF uses k hash functions and m buckets. Assuming the bucket is sufficiently large (i.e., no overflow) and y elements are added to the CBF, the false positive probability[6] (α) is given by $(1-(1-\frac{1}{m})^{ky})^k$, as detailed in Section II-B. Similarly, if x elements have been accepted by false positives (and hence removed from the set), the false negative probability[7] (β) is calculated as $(1-(1-\frac{1}{m})^{kx})^k$. In such cases, the client will re-try the TCP connection after solving a new puzzle.

In a DDoS attack with an incoming n_i normal TCP SYN packets and a_i[8] attack TCP SYN packets from AS i per unit time, the value $a_i \cdot \alpha$ indicates the additional attack load imposed on the server, and $a_i \cdot \beta$ indicates the number of tokens (or puzzles) that become wasted.

For sufficiently large m, we can approximate β as:

$$\beta \approx \left(1 - \left(1 - \frac{1}{m}\right)^{ka_i\alpha}\right)^k \approx \left(1 - e^{-\frac{ka_i\alpha}{m}}\right)^k$$

Although it is almost infeasible to eliminate both false positives and false negatives, by carefully adjusting m, y, and k, we can ensure that α and β remain within acceptable bounds (say 1%).

b) ***Disseminating puzzle seed:*** There are two cases in which the dissemination of puzzle data (from the authoritative name server) to the ASes (of the clients) becomes necessary.

Initial seed distribution. When a DNS resolver lacks puzzle data but receives a DNS query from a client requesting this data, it must then request the puzzle data from the authoritative name server. When the resolver receives the puzzle data, it will generate the hash chain, and construct the DNS records (e.g., PUZZLE records), each of which contains

[4]We assume the source address spoofing is mitigated by ingress and egress filtering.

[5]By keeping track of normal traffic, the monitoring server already predicts n_i for each time period (say 10 minutes).

[6]The probability that a spoofed or reused puzzle presented in a TCP-SYN packet is accepted by the server.

[7]The probability that a valid solution of the puzzle is not accepted by the server.

[8]The attack traffic will be TCP SYN packets (from the bots) with the valid or invalid solutions of valid or invalid (i.e., reused or spoofed) puzzles.

the puzzle token and puzzle difficulty threshold (for each potential client). Once a DDoS attack takes place, the target asks a client to request a puzzle from a resolver (to be detailed later). Then the client retrieves the puzzle from the resolver. Therefore, when the resolver exhausts all the hash values (or puzzles) in the hash chain, it must request a new seed and a new difficulty threshold to the authoritative name server.

Updating seeds. An authoritative name server may need to update the difficulty of solving puzzles. In such a scenario, TCP connection requests (with a solution of the previous puzzle difficulty) from clients would be rejected by a target, prompting clients to request a new puzzle with an up-to-date difficulty level from their DNS resolvers; we propose to use one of the reserved bits for future use in the DNS header to request the resolver to retrieve a new `PUZZLE` record for puzzle data (from the authoritative name server of the target). Then, even if a resolver has not used up all the generated hash values, it requests new puzzle data from the authoritative name server.

*2) **Puzzle distribution from resolvers to clients**:* After receiving puzzle seed data from the authoritative name server, a DNS resolver generates a hash chain of tokens (or puzzles). This is achieved by applying a hash function to the seed for a specified number of times, which is the chain length in the puzzle data. When a client requests a puzzle (e.g., `PUZZLE` record) from its resolver, the resolver provides a token from this chain, distributing it in reverse order - starting with the most recently generated one. This reverse order distribution prevents clients from predicting subsequent puzzles in the chain.

C. Solving Puzzles and Verifying Solutions

*1) **Puzzle Solution and Submission**:* Solving a puzzle involves finding a nonce value that satisfies the following condition. From an authoritative name server, a local DNS resolver of AS_i obtains its seed S_i, chain length L_i, and difficulty threshold T_i. For a given AS_i, it has a local DNS resolver whose address is R_i. A client j (in AS_i) who received a token h_j (from the local resolver) has an IP address $A_{i,j}$ and it should find a nonce that satisfies the following equation.

$$H\left(A_{i,j}, R_i, h_j, \text{nonce}\right) \le T_i$$

Here, the token h_j is an element of a hash chain. Including the source IP ($A_{i,j}$) and the local DNS resolver's IP (R_i) helps prevent sharing nonces among bots. Replay attacks, in which attackers try to reuse the solution of the same puzzle, can be mitigated using a CBF, as detailed in Section III-B1.

The equation below is the probability of successfully solving a puzzle, where l denotes the bit length of the hash output, and 2^l represents the total number of possible outcomes generated by the hash function.

$$\Pr(H\left(A_{i,j}, R_i, h_j, \text{nonce}\right) \le T_i) = \frac{T_i}{2^l}$$

For instance, SHA-256 can be used as the hash function, and only the least significant 32 bits of the result are extracted

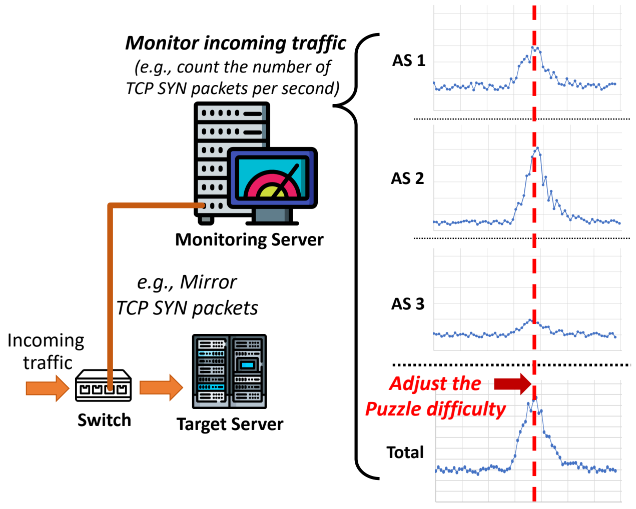
Monitor incoming traffic
(e.g., count the number of TCP SYN packets per second)
Monitoring Server
e.g., Mirror TCP SYN packets
Incoming traffic
Switch
Target Server
AS 1
AS 2
AS 3
Adjust the Puzzle difficulty
Total

Fig. 3. The operations of the monitoring server are illustrated. The monitoring server watches the incoming traffic from each AS to the target and dynamically adjusts the puzzle difficulty for each AS based on their traffic (e.g., the normal traffic and the attack traffic).

when $l = 32$. Here, the difficulty level is set to be an unsigned 32-bit integer. Also, the difficulty level is determined by adjusting the threshold, which is reduced as the difficulty level increases. For instance, when the threshold is $2^{32} - 1$, there is no need to solve the puzzle. When the threshold is T_i, an expected average number of hash operations is given by:

$$\frac{2^l}{T_i}$$

After finding a solution (i.e., a nonce value), a client j (in AS_i) sends the found nonce, token (h_j), and local DNS resolver's IP address (R_i) to the target by including them in its TCP SYN packet. How to include these data in the TCP header extension will be explained in Section IV-A.

*2) **Puzzle Solution Verification**:* After receiving a solution from the client, the target server verifies it as follows. First, the target verifies whether the received token was legitimately issued by checking the token against the CBF from the monitoring server. If valid, it then checks whether the solution satisfies the specified condition (say, less than or equal to the threshold).

If the solution and the puzzle are valid, the target will reply with a TCP SYN-ACK packet to complete the TCP handshake. If the puzzle or solution is invalid, the connection request is rejected; for instance, a TCP RST packet may be sent to the client. This may also happen if a TCP SYN packet contains a solution for a puzzle of old difficulty level, which typically occurs under a DDoS attack when the target server increases the puzzle difficulty. In this case, the target aborts the connection by sending a TCP RST packet (with a reserved bit set), which forces clients to retrieve new puzzles through their DNS resolvers.

D. Adjusting Puzzle Difficulty

The monitoring server, which adjusts the puzzle difficulty, monitors incoming traffic from each AS to the target server. It monitors the rate of TCP SYN packets from each AS.[9] Note that the monitoring server analyzes the Internet routing registry (IRR), resource public key infrastructure (RPKI), and BGP dataset so as to be able to map the source IP addresses to their AS numbers. Figure 3 illustrates how the monitoring server operates.

The monitoring server detects the onset of a DDoS attack if there is an abnormal increase of *total number* of incoming TCP SYN packets across the ASes. Here, c_i and n_i are the amounts of current traffic and normal traffic from AS i, respectively. The latter (n_i) is the average of the previously measured traffic for the given time interval (say, from noon to 10 minutes after noon). `DDD` is triggered when $T^c \geq \gamma \cdot T^n$, where $T^n = \sum_i n_i$, $T^c = \sum_i c_i$, and γ is a threshold for detecting DDoS attacks. Upon detecting a DDoS attack, the monitoring server determines the puzzle data for each AS considering the current traffic and the normal traffic measured previously.

Puzzle difficulty adjustment policy. The first principle governing puzzle difficulty adjustment is that the target server's resources are provisioned for regular traffic. The second principle is that an AS with a higher ratio of attack traffic to normal traffic will incur more penalties, requiring clients within the AS to solve more difficult puzzles.

Let us denote the current traffic rate from AS i as c_i. By setting the difficulty threshold of AS i, T_i, as follows, we can adjust the traffic rate to the desired[10] normal rate, n_i. Here, l denotes the length of the truncated bits from the hash output, and 2^l represents the total number of potential values for these truncated bits.

$$\frac{T_i}{2^l} = \frac{n_i}{c_i}$$

The time required to solve the puzzle is a critical factor impacting service availability. If we denote the hash rate of a given computer as r_h, its expected value of puzzle-solving time S can be expressed:

$$S = \frac{2^l}{r_h T_i}$$

As the monitoring server monitors incoming traffic from individual ASes, it periodically calculates the average amount of traffic coming from each AS (denoted by n_i from AS i, measured over, for instance, every 10 minutes). Considering that DDoS attacks frequently originate from a fluctuating number of bots spread unevenly across numerous ASes, it is crucial to adjust puzzle difficulty levels in a traffic volume-adaptive and AS-specific manner. Each AS will be penalized based on its rate of traffic increases. This strategy operates on the principle of selectively imposing higher difficulty on ASes that exhibit a significant influx of attack traffic. The aim is to mitigate the risk of reduced overall network availability caused by concentrated attacks from specific ASes.

[9]For instance, it can rely on the port mirroring function on the Ethernet switch connected to the target server to receive mirrored TCP SYN packets.

[10]We assume that the target server farm is provisioned for normal traffic.

IV. Evaluation

A. Implementation

We have integrated the puzzle mechanism into the TCP stack of the Linux kernel. Also, we develop a monitoring server, an authoritative name server, and a target server application to demonstrate the effectiveness of our design.

1) ***Puzzle mechanism (in the Linux kernel):*** As mentioned earlier, a client includes the solution of the target's puzzle within its TCP SYN packet. For this, we slightly modified the TCP handshake mechanism in Linux kernel version 6.1 (of Raspberry Pi OS); we assume that both clients (including bots) and the target operate on Linux OS. Specifically, we exploit the *Data Offset* and *Options* fields in the TCP header as follows. We use the *Options* field to include the DNS resolver's IP, nonce, puzzle token, and puzzle threshold. In addition, we implement functions that extract and validate the puzzle solution from the TCP header in the kernel. The hash function is the SHA-256 function from the OpenSSL library.

2) ***DNS and Target servers:*** We develop a C program that serves as a simple DNS resolver, which handles queries for DNS `A` and `PUZZLE` records[11], fetches them from the authoritative name server, and forwards the responses to clients. Additionally, it generates a chain of tokens (i.e., puzzles) from the seed in the `PUZZLE` record. Note that the authoritative name server, also implemented as a C program, distributes DNS `A` and `PUZZLE` records (of individual ASes), the latter of which is generated by the monitoring server. Note that all DNS communications use UDP. Moreover, we implement a C application (for a target server) to handle TCP connection setups, akin to an echo server, to evaluate our design's efficacy against DDoS attacks, especially TCP SYN flooding.

3) ***Monitoring server:*** Using the `libpcap` library, the monitoring server captures all incoming packets toward the target and counts TCP SYN packets. Upon receiving packets, the monitoring server categorizes their originating ASes by analyzing the source addresses of the packets. We also employ a circular buffer to establish a sliding window of one second, storing packet data received within the most recent second.

The monitoring server keeps track of TCP SYN packet counters for each AS, recording the number of packets from each AS in each time period (say 10 min). When a DDoS attack is detected (e.g., when the total size of the circular buffer reaches the threshold), the monitoring server calculates an AS-specific puzzle difficulty level based on the current counter for each AS and the previously measured normal traffic.

4) ***Reference scheme:*** Finally, to set a baseline, we implement a reference scheme where the target distributes puzzles directly to clients. Unlike `DDD`, in this scheme, the target distributes puzzles to clients directly after a TCP request reset message as follows. The target provides a puzzle via a separate

[11]We assume a new `PUZZLE` record is introduced in DNS.

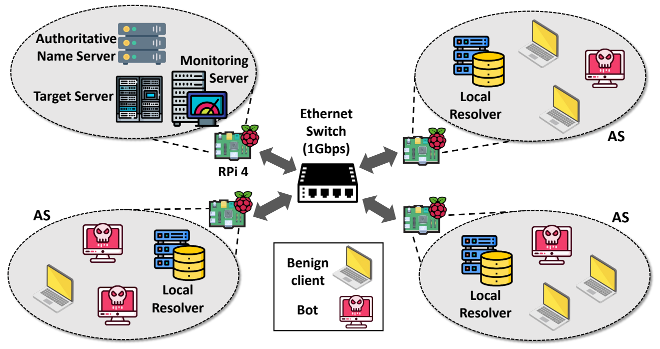

Fig. 4. Our testbed configuration is illustrated.

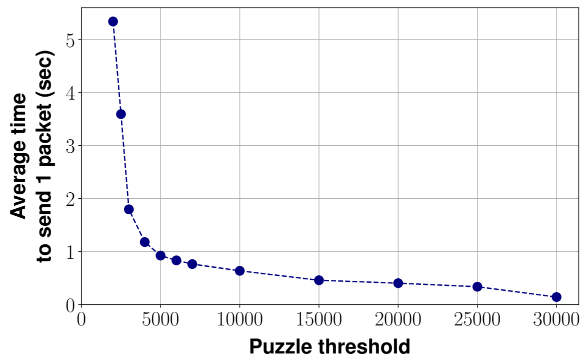

Fig. 5. The average time taken by a client to find a solution (and send a packet) to the target server, depending on the puzzle threshold, is plotted.

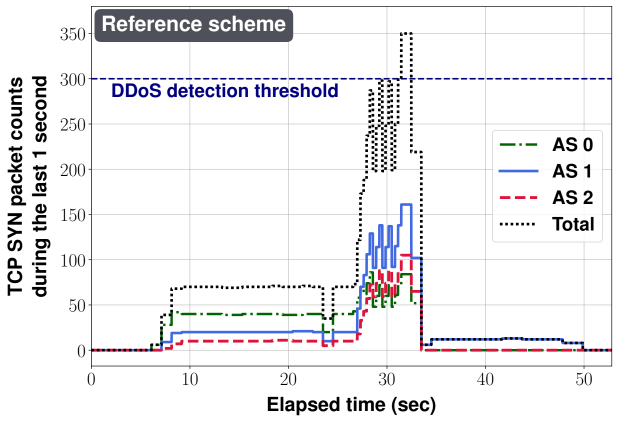

Fig. 6. This demonstrates the reference scheme's traffic control capability, by showing the counts of incoming TCP SYN packets before and during a DDoS attack.

UDP message (for ease of implementation). The client then solves it and submits the solution in a TCP SYN packet to the target. Such a direct distribution of puzzles has a drawback: *if the target becomes unavailable, its ability to control traffic through puzzles becomes ineffective* (which will be shown in Section IV-C2).

B. Experimental Configuration

Testbed setup. Our testbed comprises four Raspberry Pi (RPi) version 4 devices connected through a 1 Gbps Ethernet switch (in Figure 4). One RPi is assigned to host the following servers (i.e., co-location): a target server, its authoritative name server, and its monitoring server. The monitoring server captures all the traffic going to the target server.

The other three RPis are configured to represent three different ASes, respectively. Each RPi runs a local DNS resolver and three clients (including both benign and bot clients). Note that, given the RPi 4's four-core architecture, we can run up to four entities (e.g., one local resolver and three clients). To ensure our testing does not interfere with external networks, all experiments are conducted in a controlled, closed environment (e.g., the testbed is detached from the Internet).

Evaluation criteria. To assess the effectiveness of `DDD`, we conduct a series of experiments: Initially, (1) we investigate the rate-limiting capacity of puzzles by adjusting the puzzle difficulty (i.e., the puzzle threshold). This involves measuring the average time required for a client to find a solution (and send an eligible packet) as we vary the puzzle threshold. This measurement data is utilized to adjust puzzle difficulties in response to fluctuations in incoming traffic levels. As a comparison, (2) we explore the rate control capability of a reference scheme in which the target directly disseminates puzzles to clients. Lastly, (3) we examine whether `DDD` can effectively control incoming traffic rates and compare its result with that of the reference scheme.

C. Experiment Results

1) ***Rate limiting capability of puzzles:*** First, we evaluate the rate-limiting capability of puzzles by varying the puzzle difficulty. In the absence of the puzzle mechanism, a single client (e.g., a client application in an RPi) can transmit approximately 4,000 packets per second in our testing setup. Figure 5 shows the average time required for a single client to find a solution and send a packet with our puzzle mechanism, which is integrated within the kernel. Note that this experiment focuses on the client's process of acquiring puzzle data from a local DNS and solving the puzzle. As the puzzle threshold decreases (or the difficulty increases), the time required to solve a puzzle increases accordingly. This implies that the rate at which puzzles are solved (and consequently TCP SYN packets arrive at the target) is inversely proportional to the threshold value shown in Figure 5.

The monitoring server endeavors to maintain the flow of normal traffic from each AS by adjusting the puzzle difficulty level of each AS as incoming traffic surpasses the predefined DDoS threshold. This adjustment process continues until the volume of current traffic returns to normal levels.

2) ***DDoS traffic control by the reference scheme:*** Next, we conduct an experiment employing a reference scheme that distributes puzzles directly from the target, in contrast to `DDD`. In this attack scenario, all three ASes launch a SYN flooding attack against the target, with each AS contributing varying levels of traffic. The reference scheme identifies the attack when the total TCP SYN packet count per second exceeds 300. Figure 6 demonstrates the limitation of the reference

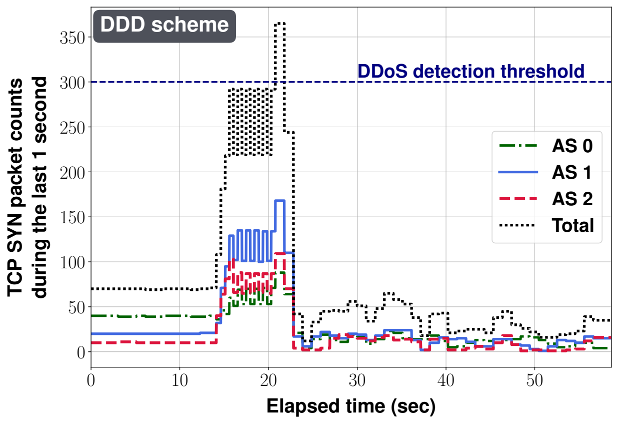

Fig. 7. This demonstrates DDD's traffic control capability, by showing the counts of incoming TCP SYN packets before and during a DDoS attack.

scheme in controlling the volume of incoming traffic under a DDoS attack scenario. The SYN flooding attack starts around 27 seconds, whereas the target, operating under the reference scheme, detects the attack near 31 seconds, as total TCP SYN packet counts surpass the threshold. Subsequently, the target attempts to regulate traffic by directly disseminating puzzles to clients. However, after 33 seconds, the target's TCP SYN count drops to zero for AS 0 and AS 2, which indicates the target's resources are depleted and hence it cannot reply with puzzles. As anticipated, should the target itself directly distribute puzzles, its ability to control traffic can be diminished when overwhelmed by DDoS traffic.

*3) **DDoS traffic control by DDD**:* Finally, we assess the efficacy of DDD in controlling the DDoS traffic, using the same setup as in the above experiment; all three ASes flood SYN packets, and the DDoS detection threshold is set at 300 TCP SYN packets per second. Figure 7 illustrates the capability of DDD to contain the DDoS traffic. At approximately 14 seconds, bots across ASes initiate SYN flooding, leading to an increase in TCP SYN packet counts per second. Specifically, these counts exceed 70 for AS 0, 130 for AS 1, and 80 for AS 2, with some fluctuations. When the total traffic from these ASes exceeds the DDoS detection threshold, DDD increases the puzzle difficulty to reduce the influx of TCP SYN packets to the target. Consequently, the incoming traffic begins to diminish and eventually stabilizes by around 22 seconds. In contrast to the results of the reference scheme (in Figure 6), DDD successfully adjusts the incoming traffic without any availability issues.

V. Discussion

*1) **Security of DNS**:* DDD does not require DNS security mechanisms (such as DNSSEC [26], [28], [27], DoT [11], and DoH [10]) for PUZZLE records. Deploying such mechanisms will enhance the trustworthiness of the puzzle data.

The DNS infrastructure itself is susceptible to DDoS attacks. Nonetheless, due to its critical role, defensive measures such as rate limiting, access control, anti-spoofing, and filtering have been developed. Legitimate queries to authoritative name servers typically originate from (local and public) DNS resolvers. Implementing these measures in such networking environments can be done seamlessly.

*2) **Puzzle solving overhead in non-attack situations**:* In DDD, clients solve puzzles to establish connections with a target server, regardless of whether flooding attacks are occurring. Under normal circumstances, the puzzle's difficulty level is effectively zero, as the threshold is set to the maximum value (e.g., $2^{32}-1$ for a 32-bit unsigned integer), enabling clients to find a nonce with a single hashing operation. In response to a sudden influx of traffic, the threshold will dynamically adapt, managing incoming traffic via puzzle-solving.

DDD can also be configured to enforce puzzle-solving only during DDoS attacks. During normal traffic, the target will not verify puzzle solutions, and the clients will not be required to solve puzzles. However, upon detecting excessive traffic, the target would counter incoming TCP SYN packets with TCP RST packets, prompting clients to engage in puzzle-solving.

*3) **Other types of DDoS attacks**:* DDD primarily aims at detecting and mitigating TCP SYN-based flooding attacks. However, it can be adapted to counter other TCP-based flooding attacks, such as TCP FIN flooding. Also, while UDP-based DDoS attacks are not the focus of DDD, it can be extended to address these threats. By monitoring UDP packets based on their port numbers (e.g., DNS, NTP, SSDP), we can implement port-level control for UDP flooding packets originating from each AS. Alternatively, we can regulate the overall rate of UDP traffic from each AS. Integrating a rate-limiting mechanism such as [25] into DDD can be easily implemented.

*4) **IP spoofing mitigation**:* In DDoS attacks, bots spoofing their source IP addresses challenge defense systems, including DDD, which depend on rate control tied to origin ASes identified through source IP addresses. Consequently, a key strategy against such attacks is identifying and blocking spoofed IP packets. Ingress filtering [31], employed on routers or firewalls, blocks suspicious incoming packets based on their IP addresses. Although the specific deployment of ingress filtering may vary to some extent, approximately 30% of ASes had implemented ingress filtering as of 2021 [8].

Relying on third-party IP blacklists [1], [32] can improve the effectiveness in filtering out malicious IP addresses. Additionally, for each target server, maintaining the whitelist of ASes where the majority of potential (benign) clients are located is straightforward. In cases of severe flooding attacks, it may be necessary to discard all incoming packets from the ASes other than the whitelist.

VI. Conclusion

In this paper, we propose a novel DNS-based DDoS defense mechanism, DDD, which tackles the challenges of puzzle distribution by leveraging DNS. Employing DNS for puzzle distribution as an 'out-of-band' loop for traffic control enhances the resilience and scalability of DDD against DDoS attacks. We also propose implementing AS-specific rate control to incentivize ASes to upkeep their networks with fewer bots. To do so, DDD has a mechanism for adjusting puzzle difficulty through the monitoring server, enabling AS-level traffic monitoring and control. Through prototype-based experiments, we evaluate DDD to demonstrate its performance in controlled network environments. Our findings underscore the effectiveness and robustness of DDD in mitigating DDoS traffic.

VII. Acknowledgments

This work was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2024-2021-0-02048) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation), and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (NRF-2022R1A2C2011221 and No. RS-2023-00220985).

References

[1] abuse.ch. Feodo Tracker. https://feodotracker.abuse.ch/, (accessed May 3, 2024).

[2] B. Al-Duwairi and M. Govindarasu. Novel hybrid schemes employing packet marking and logging for IP traceback. *IEEE Transactions on Parallel and Distributed Systems*, 17(5):403–418, 2006.

[3] T. Anderson, T. Roscoe, and D. Wetherall. Preventing Internet denial-of-service with capabilities. *ACM SIGCOMM Computer Communication Review*, 34(1):39–44, 2004.

[4] K. J. Argyraki and D. R. Cheriton. Active internet traffic filtering: Real-time response to denial-of-service attacks. In *USENIX annual technical conference, general track*, volume 38, 2005.

[5] K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. A. Olsson. Detecting disruptive routers: A distributed network monitoring approach. *IEEE network*, 12(5):50–60, 1998.

[6] R. Chen, J.-M. Park, and R. Marchany. Nisp1-05: Rim: Router interface marking for ip traceback. In *IEEE Globecom 2006*, pages 1–5. IEEE, 2006.

[7] CNET. DOJ, FBI, entertainment industry sites attacked after piracy arrests, Jan. 2012. https://www.cnet.com/news/privacy/doj-fbi-entertainment-industry-sites-attacked-after-piracy-arrests/, (accessed May 3, 2024).

[8] T. Dai and H. Shulman. Smap: Internet-wide scanning for spoofing. In *Annual Computer Security Applications Conference*, pages 1039–1050, 2021.

[9] D. Dean and A. Stubblefield. Using Client Puzzles to Protect TLS. In *10th USENIX Security Symposium (USENIX Security 01)*, 2001.

[10] P. E. Hoffman and P. McManus. DNS Queries over HTTPS (DoH). RFC 8484, Oct. 2018.

[11] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. E. Hoffman. Specification for DNS over Transport Layer Security (TLS). RFC 7858, May 2016.

[12] M. Jonker, A. Sperotto, R. van Rijswijk-Deij, R. Sadre, and A. Pras. Measuring the adoption of DDoS protection services. In *Proceedings of the 2016 Internet Measurement Conference*, pages 279–285, 2016.

[13] A. Jules and J. Brainard. Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks. In *Proceedings of the Network and Distributed Systems Security Symposium*, 1999.

[14] S. Khandelwal. 602 Gbps! This May Have Been the Largest DDoS Attack in History, Jan. 2016. https://thehackernews.com/2016/01/biggest-ddos-attack.html, (accessed May 3, 2024).

[15] Y. Kim, W. C. Lau, M. C. Chuah, and H. J. Chao. PacketScore: a statistics-based packet filtering scheme against distributed denial-of-service attacks. *IEEE transactions on dependable and secure computing*, 3(2):141–155, 2006.

[16] T. Kitten. DDoS: Lessons from Phase 2 Attacks, Jan. 2013. https://www.bankinfosecurity.com/ddos-attacks-lessons-from-phase-2-a-5420, (accessed May 3, 2024).

[17] J. Mirkovic, G. Prier, and P. Reiher. Attacking DDoS at the source. In *10th IEEE International Conference on Network Protocols, 2002. Proceedings.*, pages 312–321. IEEE, 2002.

[18] J. Mirkovic, G. Prier, and P. Reiher. Source-end DDoS defense. In *Second IEEE International Symposium on Network Computing and Applications, 2003. NCA 2003.*, pages 171–178. IEEE, 2003.

[19] A. T. Mizrak, S. Savage, and K. Marzullo. Detecting compromised routers via packet forwarding behavior. *IEEE network*, 22(2):34–39, 2008.

[20] Netscout. DDoS Threat Intelligence Report: Issue 11, Sept. 2023. https://www.netscout.com/threatreport, (accessed May 3, 2024).

[21] M. A. Noureddine, A. M. Fawaz, A. Hsu, C. Guldner, S. Vijay, T. Başar, and W. H. Sanders. Revisiting client puzzles for state exhaustion attacks resilience. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 617–629. IEEE, Aug. 2019.

[22] K. Park and H. Lee. On the effectiveness of probabilistic packet marking for IP traceback under denial of service attack. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, volume 1, pages 338–347. IEEE, 2001.

[23] K. Park and H. Lee. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets. *ACM SIGCOMM computer communication review*, 31(4):15–26, 2001.

[24] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y.-C. Hu. Portcullis: Protecting connection setup from denial-of-capability attacks. *ACM SIGCOMM Computer Communication Review*, 37(4):289–300, 2007.

[25] S. Rao and S. Rao. Denial of Service attacks and mitigation techniques: Real time implementation with detailed analysis. *This paper is from the SANS Institute Reading Room site*, 2011.

[26] S. Rose, M. Larson, D. Massey, R. Austein, and R. Arends. DNS Security Introduction and Requirements. RFC 4033, Mar. 2005.

[27] S. Rose, M. Larson, D. Massey, R. Austein, and R. Arends. Protocol Modifications for the DNS Security Extensions. RFC 4035, Mar. 2005.

[28] S. Rose, M. Larson, D. Massey, R. Austein, and R. Arends. Resource Records for the DNS Security Extensions. RFC 4034, Mar. 2005.

[29] J. J. Santanna, R. van Rijswijk-Deij, R. Hofstede, A. Sperotto, M. Wierbosch, L. Z. Granville, and A. Pras. Booters—an analysis of ddos-as-a-service attacks. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 243–251. IEEE, 2015.

[30] E. Schonfeld. Swedish parliament website hit by DDoS attack, May 2023. https://www.reuters.com/world/europe/swedens-parliament-hit-by-cyber-attack-2023-05-03/, (accessed May 3, 2024).

[31] D. Senie and P. Ferguson. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827, May 2000.

[32] Spamhaus. Spamhaus Botnet Controller List. https://www.spamhaus.org/bcl/, (accessed May 3, 2024).

[33] H. Wang, C. Jin, and K. G. Shin. Defense against spoofed IP traffic using hop-count filtering. *IEEE/ACM Transactions on networking*, 15(1):40–53, 2007.

[34] X. Wang and M. K. Reiter. Mitigating bandwidth-exhaustion attacks using congestion puzzles. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 257–267, 2004.

[35] B. Waters, A. Juels, J. A. Halderman, and E. W. Felten. New client puzzle outsourcing techniques for DoS resistance. In *Proceedings of the 11th ACM conference on Computer and Communications Security*, pages 246–256, Oct. 2004.

[36] N. Woolf. DDoS attack that disrupted internet was largest of its kind in history, experts say, Oct. 2016. https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet, (accessed May 3, 2024).

[37] Y. Wu, Z. Zhao, F. Bao, and R. H. Deng. Software puzzle: A countermeasure to resource-inflated denial-of-service attacks. *IEEE Transactions on Information Forensics and security*, 10(1):168–177, 2014.

[38] A. Yaar, A. Perrig, and D. Song. Siff: A stateless internet flow filter to mitigate ddos flooding attacks. In *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, pages 130–143. IEEE, 2004.

[39] X. Yang, D. Wetherall, and T. Anderson. TVA: A DoS-limiting network architecture. *IEEE/ACM Transactions on Networking*, 16(6):1267–1280, 2008.

[40] S. T. Zargar, J. Joshi, and D. Tipper. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE communications surveys & tutorials*, 15(4):2046–2069, 2013.