Selective Aggressive Caching in DNS Resolvers

Hadar Cochavi, Gil Einziger
Computer Science Department, Ben Gurion University of the Negev,Be'er Sheba, Israel.
hadarco@post.bgu.ac.il, gilein@bgu.ac.il

***Abstract*—Volumetric attacks on the Domain Name System (DNS) are an escalating concern, particularly NXDomain attacks, in which attackers generate queries for nonexistent subdomains of a target domain. This tactic effectively bypasses resolver caches, degrading Internet performance for millions of users. Notably, RFC8198 proposes countermeasures to prevent attackers from circumventing resolver caching, significantly reducing the attack's effectiveness. However, RFC8198 requires resolvers to utilize DNSSEC-validated entries, which demand more computational resources than standard entries. As a result, implementing RFC8198 increases resolvers' resource consumption and reduces their effective throughput, both under normal conditions and during an attack. In essence, resolvers must allocate additional resources to shield authoritative name servers from attacks—a challenging proposition given that these servers are managed by separate entities with potentially conflicting interests. Our work reexamines the implementation of RFC8198 in resolvers and proposes self-optimizing strategies that enable resolvers to adopt the RFC while maximizing performance both during normal operation and under attack. Our solution increases throughput by approximately 40% during an attack and reduces average latency by a factor of two. Simultaneously, it maintains comparable peacetime performance and enhances protection for authoritative name servers against NXDomain attacks.**

***Index Terms*—DNS Resolver, Volumetric Attacks, Cache Policy, Heavy Hitters, Count-Min Sketch.**

I. Introduction

The *Domain Name System (DNS)* is a fundamental component of the Internet, facilitating the resolution of domain names into IP addresses and vice versa (whether IPv4 or IPv6). DNS consists of two primary types of users: *resolvers*, which query the system on behalf of clients, and *authoritative name servers*, which provide the necessary mappings. Each authoritative name server is responsible for a specific domain. The literature distinguishes between *top-level domains (TLDs)*, which manage domains such as `".com"` and `".net"`, and *second-level domains (SLDs)*, which oversee their respective subdomains (e.g., `"duckduckgo.com"` is a subdomain of `".com"`). In our work, we collectively refer to these entities as *authoritative name servers*. Various organizations, including ISPs, commercial enterprises, academic institutions, and others, own and operate different authoritative name servers and resolvers.

The performance of *Domain Name System (DNS)* directly impacts the entire Internet. To optimize efficiency, the community employs techniques such as caching query results at resolvers, reducing unnecessary access to authoritative name servers. DNS caches are known to achieve a high hit ratio, and one could argue that the service's feasibility relies on effective caching.

Attacks on DNS can disrupt Internet access for millions of users [1], making them particularly impactful. One such attack is *NXDomain attack*, where an attacker leverages a large botnet of malware-infected network devices connected to multiple resolvers. Bots send low-rate queries to evade detection, requesting domain names of the form `"<random string>.example.com"` that do not exist. Since these subdomains have never been requested before, they bypass the resolver's cache and reach the authoritative server. If the botnet is sufficiently large, it forces resolvers to generate an overwhelming load on the authoritative server of `"example.com"`.

Because the number of nonexistent subdomains is effectively infinite, attackers can sustain the attack indefinitely. The most well-known example of such an attack was *Dyn attack*, which disrupted Internet access for millions of users in the United States for an extended period [2]. RFC8198 proposes leveraging DNSSEC behavior to enhance caching efficiency. Specifically, when proving the non-existence of a domain name (e.g., `"cat.example.com"`), a DNSSEC-protected authoritative name server returns a validated record for `"albatross.example.com"`, which has `"elephant.example.com"` as the next existing subdomain. This record implicitly confirms that `"cat.example.com"` does not exist, as `"cat"` is lexicographically between `"albatross"` and `"elephant"`.

Notably, this proof also verifies the non-existence of `"dog.example.com"` and infinitely many other subdomains within the same range. RFC8198 allows resolvers to return synthesized NX responses to queries for such domains, even if the specific subdomain was not previously requested. To achieve this, resolvers aggressively cache these non-existence proof records.

ISBN 978-3-903176-XX-X

However, while aggressive caching of DNSSEC-validated records mitigates an NXDomain attack on "example.com", it does not offer the same protection for the resolver. This limitation arises from the cryptographic methods used by "example.com" to sign its responses and the validation process the resolver must perform. As a result, the resolver incurs higher resource consumption when handling queries, ultimately degrading its peacetime performance.

Our Contribution: Our work explores how resolvers can leverage aggressive caching to enhance performance. We propose an algorithm, *TopDomains(k)*, which requests DNSSEC-validated records only for domains that may be under attack—specifically, those generating a high volume of NX responses. By selectively applying DNSSEC validation, the algorithm preserves peacetime performance by limiting cryptographic processing to a small subset of domains. However, when an NXDomain attack is detected, the resolver identifies the surge in NX responses and enables DNSSEC validation for these domains. Since PowerDNS implements RFC8198, it benefits from improved caching efficiency without the overhead of indiscriminately validating responses from all secure authoritative name servers.

We implemented a *TopDomains(k)* prototype on PowerDNS and evaluated it using both real and synthetic DNS access traces. Our results demonstrate that *TopDomains(k)* maintains the resolver's peacetime performance while increasing throughput by up to 40% during an attack. In contrast, indiscriminate DNSSEC validation degrades performance, as the additional computational load outweighs the potential caching benefits.

Roadmap: The rest of this paper is organized as follows. In Section II, provides an overview of existing NXDomain attack mitigation methods, and the necessary background to understand, and motivate our approach. Our TopDomains(k) algorithm is presented in Section III; In that section, we start by surveying the model and its assumptions, explain the designs of popular DNS resolvers, and present our approach as a series of design decisions. Section IV evaluates the performance of the PowerDNS resolver with our improvements compared to its original version. Finally, Section V concludes and outlines future work plans.

II. Background and Related Work

This section provides a detailed background on the *Domain Name System (DNS)* and heavy-hitter algorithms, which we utilize to identify top domains. The DNS system comprises two main components: *authoritative servers* and *recursive resolvers*, which work together to resolve domain names into IP addresses [3].

A. The Authoritative Servers

The *authoritative servers* form a globally distributed database with a hierarchical structure, where each subdomain is managed by a separate entity. At the top of this hierarchy is the *root* (denoted as "."), followed by the *Top-Level Domains (TLDs)*, which are further categorized into *generic TLDs* and *country-code TLDs*. Below the TLDs are the *Second-Level Domains (SLDs)* and any additional lower levels. To streamline this structure, each component is separated by a dot ("."), including the boundary between the root and the TLDs, ensuring that the root maintains information about all TLDs. This is illustrated in Figure 1.

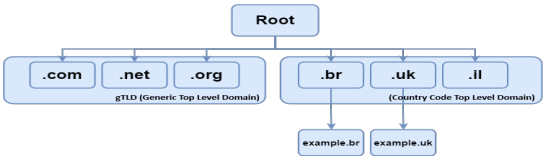


Fig. 1: The overall structure of DNS. The DNS hierarchy tree has a single domain at the top of the structure called the root domain, indicated by the ".". Below the root domain are the top-level domains that divide the DNS hierarchy into segments containing second-level domains, sub-domains, and hosts.

DNS Resolver *Resolver* act as the gateway to the DNS. Specifically, when a user accesses a certain domain (e.g., by clicking a link in their browser), their client issues a query to a DNS resolver. Thus, resolvers provide an interface to the DNS and return the corresponding IP address, allowing clients to access the requested domain.

More specifically, resolvers perform an *iterative DNS lookup*, gradually resolving the requested subdomain using information provided by authoritative name servers. This information is typically cached at the resolver to improve performance. For example, when a client requests "www.example.com", the resolver first checks its cache for an existing record. If a cache hit occurs, the resolver returns the stored information to the user. Otherwise, it begins the resolution process by first attempting to resolve the TLD server for ".com", which may be cached or retrieved from the root servers.

Next, the resolver searches its cache for "example.com". If the record is absent, it queries the ".com" TLD server to obtain the address of "example.com". Finally, the resolver contacts the SLD server of "example.com" to retrieve the IP address of "www.example.com" and returns it to the client.

DNS Cache As detailed above, resolvers may perform multiple cache accesses during a query. Since DNS workloads are highly cacheable, such optimizations significantly reduce the load on authoritative name servers while improving both user experience and throughput [4].

DNS caches store *Resource Records (RR)*, each labeled according to its type: an IPv4 address for a domain (A), an IPv6 address (AAAA), authoritative name servers for a domain (NS), or a record indicating a non-existent domain (NX).

We distinguish between caching *positive records*, which contain mappings between domains and IP addresses (IPv4 or IPv6), and *negative records*, which indicate that a domain does not exist. Positive records are stored using an exact match cache, which aligns with the fundamental functionality of DNS. Traditional DNS also employs an exact match approach for negative records, but this design opens the door to various DDoS attacks, such as the NXDomain attack. In contrast, DNSSEC enables an alternative approach known in the literature as *Aggressive Caching*, which mitigates these vulnerabilities.

DNS Security Extensions (DNSSEC) The *DNSSEC* protocol [5] is built on public key cryptography, certificates, and digital signatures, providing source authentication and integrity protection for DNS data. DNSSEC aims to prevent attacks such as cache poisoning and DNS spoofing [6], [7]. To achieve this, it introduces additional *Resource Records (RRs)*, including *Resource Record Signature (RRSIG)*, *DNS Public Key (DNSKEY)*, and *Delegation Signer (DS)*, which supplement the standard resource records mentioned earlier.

In practice, resolvers can work with DNS or DNSSEC, as most authoritative name servers would respond to both protocols. However, when DNSSEC validation is used, the resolver has to validate every record it receives, which means that it uses additional DNSSEC-related information, such as DNSSEC signatures and keys, and consumes more resources. Specifically, the resolver must perform the following steps:

(i) Retrieve the `DNSKEY` record for the zone in question, which involves an additional resolution to the parent zone to obtain the `DS` record, which contains a hash of the `DNSKEY` record. (ii) Verify the authenticity of the `DNSKEY` record using the `DS` record. (iii) Retrieve any necessary `RRSIG` records to validate the response. (iv) Validate the authenticity and integrity of the response using the DNSSEC signatures. Overall, the process of DNSSEC response validation introduces extra steps to verify digital signatures against the public keys of the signing zones, ensuring that the response is authentic and has not been tampered with.

Within the context of volumetric attacks, DNSSEC increases per-query resource consumption at the resolver, thereby reducing its throughput. Thus, while DNSSEC provides strong protection against various forms of attacks, this security comes at a computational cost. However, one of the promising aspects of DNSSEC is its efficient handling of queries to nonexistent domains, which is explained next.

1) Validation of NXDomain Response: To validate that a response received at the resolver side is not forged, DNSSEC employs a special record that ensures the response's authenticity.

Next Secure (NSEC/NSEC3): A pointer to the next valid name in a zone, used to verify the non-existence of a record name and type within a zone.

A queried server must return these `NSEC` or `NSEC3` records in response to a triggering query if the query contains the DNSSEC OK (DO) bit [8]. In a DNSSEC-signed zone, the answering DNS server provides the two domain names adjacent to the queried non-existent domain name. For instance, if a client queries for `"banana.example"`, the server responds with a pair of surrounding subdomains, such as `"apple.example"` and `"orange.example"`, thereby proving the non-existence of `"banana.example"`.

In principle, this capability can protect authoritative servers from NXDomain attacks [9]. However, validating DNSSEC entries introduces challenges such as increased performance overhead on resolvers and larger DNS response packets. Additionally, DNSSEC response validation consumes more computational resources at the resolver and, in some cases, generates additional messages to authoritative servers. This increased workload extends client latency, potentially causing timeouts and query retransmissions [10]. Attackers can exploit these inefficiencies [11] to amplify their attacks, further exacerbating DNSSEC's resource consumption.

Volumetric Attacks On DNS aim to overwhelm a DNS entity (authoritative server or resolver) by generating a high volume of requests, thereby degrading its performance and denying access to legitimate users. In such attacks, the attacker operates a botnet composed of numerous malware-infected devices to evade basic protection mechanisms such as rate limiting or blocking specific destinations. The botnet is controlled by a centralized *Command and Control program*.

Volumetric attacks remain an ongoing threat as their peak attack volume continues to grow each year. For example, in 2017, Google services faced an attack with a combined traffic volume of 2.54 Tbps [12], which was four times larger than the infamous Mirai attack of 2016—a record-breaking attack at the time.

Luo et al. [13] analyzed the prevalence and characteristics of NXDOMAIN attacks using a week-long real-world traffic dataset. They concluded that such attacks are widespread and proposed a mitigation technique based on statistical metrics, such as the ratio of distinct clients to queried domains. Similarly, Van et al. [14] conducted a detailed measurement study on a large dataset of DNSSEC-signed domains, covering approximately 70% (2.5 million) of all signed domains in operation at the time. Their analysis assessed the am-

plification potential of DNSSEC compared to standard DNS, confirming that DNSSEC significantly increases amplification potential, exceeding that of regular DNS by a factor of $6\times$ to $12\times$. This finding highlights that DNSSEC is more susceptible to volumetric attacks than traditional DNS. Furthermore, Sommese et al. [15] systematically tracked DNS-centric DDoS events over a 17-month period and showed that NXDOMAIN floods account for a large share of the disruption, reinforcing the need for lightweight in-resolver mitigation mechanisms. **Mitigation Techniques for Volumetric Attacks** We survey some of the methods for mitigating volumetric attacks on DNS. **Response Rate Limiting:** *Response Rate Limiting (RRL)* was introduced in 2012 [16] following a series of DNS amplification attacks that exploited DNSSEC-signed domains. The core concept of RRL is that authoritative name servers limit the rate of identical outgoing responses when sent to the same address or subnet (bots).

At first glance, RRL appears to be an effective mitigation strategy against DNS amplification attacks, as it throttles excessive queries from a resolver to authoritative servers under attack. However, upon closer examination, RRL itself can be exploited as a denial-of-service (DoS) weapon. An attacker can deliberately issue numerous queries to trigger the rate limit threshold, thereby causing legitimate requests to be blocked, effectively disrupting access to authoritative servers.

Aggressive Use of the DNSSEC-Validated Cache (RFC8198): Implementing the *Aggressive Caching* mechanism, as described in RFC8198 [9], enables resolvers to leverage `NSEC`/`NSEC3` resource records to synthesize negative responses based on cached information. Consequently, DNSSEC-validating resolvers can immediately return a negative response if the queried domain falls within a range covered by an `NSEC`/`NSEC3` record already stored in the cache, eliminating the need to initiate a new query to the authoritative name server. This technique mitigates NXDomain attacks on DNSSEC-protected zones by significantly reducing the number of queries sent to targeted authoritative servers. However, an adversary can exploit the caching behavior of resolvers by forcing them to perform an excessive number of DNSSEC validation operations. By generating numerous random subdomains across multiple DNSSEC-protected zones, an attacker can initiate a DDoS attack on the resolver itself, overwhelming its computational resources.

Identifying the Attack: The work of [17] proposes detecting attacked subdomains and filtering (dropping) queries for these subdomains. To achieve this, they monitor the number of distinct NX queries for each domain, i.e., the number of subdomains that returned an NX response. Their approach assumes that during an NXDomain attack, the targeted domains will exhibit a high number of distinct subdomains, allowing them to be identified.

Our method differs from theirs in two key aspects. First, Feibish et al. employ *distinct heavy hitters*, whereas we use *plain heavy hitters*. Second, their approach leverages heavy hitters to make the restrictive decision of blocking queries to attacked domains. In contrast, our method does not block clients; instead, we selectively validate responses from secure domains that might be under attack. As a result, our mitigation strategy presents a lower risk of disrupting legitimate user traffic.

Bushart et al. profile per-resolver query volumes from sampled NetFlow and deploy an upstream, content-agnostic low-pass filter: steady high-volume resolvers are allow-listed at their historical peak T_i, while all other (/24 for IPv4 and /48 for IPv6) prefixes may send up to a fixed LPF threshold; this caps AuthNS load below ˜100k qps during application-layer floods with 1.2–5.7% false positives [18]. In contrast, our scheme is implemented *inside* the recursive resolver, needs no external cooperation, and avoids blocking entirely: a lightweight CMS/heap detects the top-k NXDOMAIN heavy-hitter domains and enables RFC 8198 validation only for those, thus shielding the resolver and the targeted AuthNSes without modifying client traffic.

The Current Trade-offs Commonly, the target NX-Domain attacks is an authoritative name server. Indeed, aggressive caching (RFC8198) offers us a mitigation method that drastically reduces the attacker's capacity to bypass the resolver cache and reach the authoritative name server. PowerDNS implements RFC8198 but it does so in an all-or-nothing method. That is, PowerDNS either performs validation for all domains (DNSSEC = ON) or performs no validation at all (DNSSEC = OFF). Our evaluation shows that (i) performance-wise PowerDNS is better off using plain DNS, and (ii) during an NXDomain attack PowerDNS is still better off using plain DNS. This is explained by the high cost of validating DNS entries. Thus, the implementation of RFC8198 by PowerDNS makes it easier to attack the resolver. Using DNSSEC validation reduces PowerDNS throughput in peacetime, and even during an NXDomain attack compared to plain DNS (see Figure 4)

Our work seeks a middle road in implementing RFC8198. Namely, suppose we only perform DNS validation to the attacked subdomain and not the entire workload. In that case, the resolver's performance under attack can be considerably better, making the attack ineffective against the resolver while still ineffective against the authoritative servers.

III. TopDomains(k) Approach

Our design goal is to devise a scheme that implements RFC8198 [9] in a manner that optimally

benefits the resolver. Specifically, we aim to minimize performance overhead during peacetime while maximizing performance gains during an NXDomain attack. To achieve this, we focus on a real DNS resolver, *PowerDNS* that implements RFC8198.

In its current implementation, the resolver operates in one of two modes. When `DNSSEC=ON`, DNSSEC validation is enabled by default, meaning all responses undergo validation unless the authoritative server does not support DNSSEC. Conversely, when `DNSSEC=OFF`, the resolver defaults to standard DNS operation and performs DNSSEC validation when explicitly requested.

The core concept of *TopDomains(k)* is to restrict DNSSEC validation to the top k domains that generate the highest number of NX responses. For all other domains, standard DNS processing is used. This approach allows us to leverage the advantages of aggressive caching while minimizing the performance degradation associated with DNSSEC validation.

Our *TopDomains(k)* implementation modifies *PowerDNS* version 4.7.1, where k represents the number of domains that can undergo DNSSEC validation at any given time. In *TopDomains(32)*, for instance, the resolver does DNSSEC validation for only 32 domains.

During the recursive resolver's initialization phase, we initialize a data structure to store the heavy hitters, consisting of `<key, value>` pairs. The *key* is the domain name extracted from an incoming query (e.g., `"google.com"`, `"yahoo.com"`), and the *value* represents the number of corresponding queries that resulted in an `NXDOMAIN` response.

From an NXDomain attack perspective, the targeted domains are likely to be among the top k domains. This is based on the observation that the number of distinct subdomains in queries for these targeted domains increases significantly during attacks.

TopDomains(k) Algorithm

1) Algorithm Overview: The primary goal of our algorithm is to selectively enable DNSSEC validation for the top k heavy hitter domains. To achieve this, we analyze the incoming stream of DNS queries and their corresponding responses at the resolver. Our analysis is integrated into two key processes in the resolver: **Query Processing:** When a query arrives at the resolver: (1) Extract the domain name (key). (2) Check if the domain is among the top k most frequently queried domains. If it is, enable DNSSEC validation for this query; otherwise, disable DNSSEC validation. **Response Processing:** This phase focuses solely on `NXDOMAIN` responses, updating the relevant heavy hitter counters. A domain is considered a by the top k heavy hitter domains only if it is DNSSEC-protected.

Figure 3 illustrates the process applied to a query and its corresponding response upon arrival at the resolver, determining the appropriate actions based on the outlined criteria.

A. Determining the Top K Domains

We use a *count-min sketch* [19] to identify the top k heavy hitters. To determine whether a particular domain is among the k most frequent domains, we maintain a heap data structure containing the k largest domains. After updating a domain's counter, we insert it into the heap with its current frequency estimation (provided by the count-min sketch) and evict the smallest entry if necessary.

Figure 2 illustrates the approximate architecture of the heavy hitters mechanism. Consider a stream of DNS responses to queries received at the resolver, where d represents the domain extracted from each response. The update process for our heavy hitters data structure proceeds as follows: (1) Extract the domain d (key) from each response. (2) If d is DNSSEC-protected, invoke `Inc(d)` in the count-min sketch, followed by `Count(d)` to estimate its frequency. (3) If `Count(d)` $\geq \frac{n}{k}$, insert d into the heap using its frequency value, `Count(d)`. Otherwise, discard d. If d is already in the heap, remove its previous entry before reinserting it with the updated frequency [19]. Additionally, as n increases, if the stored value of a domain d in the heap falls below $\frac{n}{k}$ (a check that can be performed in $\mathcal{O}(1)$ time), the domain is removed from the heap.

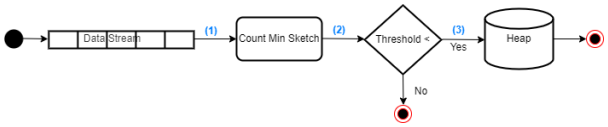
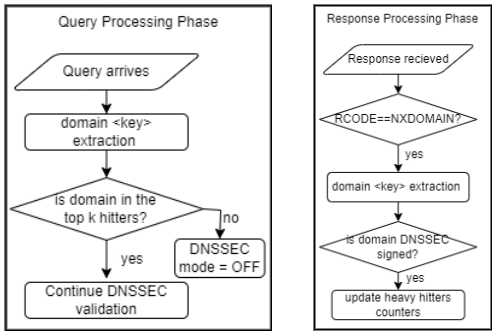


Fig. 2: The approximate heavy hitters' mechanism used in our solution. We update a count min sketch for every NX response in the stream and move requests above a certain threshold to a heap of potential heavy hitters.

Aging Mechanism We aim to prevent a scenario where the same domains consistently remain among the top k heavy hitters, resulting in DNSSEC validation and aggressive caching being applied exclusively to them. Such a situation may arise due to prolonged activity involving frequent `NXDOMAIN` responses or repeated attacks targeting these domains. Consequently, this can degrade responsiveness when NXDomain attacks shift to different domains, thereby undermining the mitigation benefits of aggressive caching. To address this issue, we implement an *aging scheme* for our sketch and heap counters. Inspired by [20], we periodically reduce all counters by a factor of 2. This approach ensures that past events are gradually forgotten, preventing stale heavy hitter data from dominating the selection process and keeping counter values relatively small over time.



(a) Query processing. (b) Response processing.

Fig. 3: TopDomain's behavior

IV. Evaluation

Throughout the experiments, we evaluate *TopDomains(32)* and compare it to the original implementation of the *PowerDNS* resolver with a configuration of DNSSEC validation enabled and disabled (i.e., `DNSSEC=ON` or `DNSSEC=OFF`).

Our evaluation setup comprises two physically separated workstations: a client and a recursive resolver. Both workstations are interconnected via a 1 Gigabit Ethernet (1GigE) NIC and include 16 CPU cores, 16 GB of RAM, running Ubuntu 22.04.1. In addition, the recursive resolver also runs *PowerDNS* 4.7.1 in recursion operation mode. Finally, the client is configured to send DNS requests directly to our recursive resolver.

Performance Metrics: Our measurements use the following performance metrics: maximum attainable throughput in queries per second (qps), latency distribution of successful DNS queries, and the number of packets/bytes generated in an experiment.

Amplification Factors The effectiveness of DDoS attacks is often measured by the *amplification factor*, which represents the ratio between the traffic generated by the attacker and the traffic processed by the resolver. Intuitively, a higher amplification factor makes the attack more efficient for the adversary. Below, we describe the amplification metrics measured in this work. **Cost:** The *packet cost* refers to the number of packets that the victim (resolver) processes in response to a single client request. This cost depends on several factors, including: - The `DO` flag, which determines whether DNSSEC validation is performed. - The `TC` bit (truncate bit), which indicates whether the UDP response size exceeded the configured limit ($TC = 1$) or remained within the allowable size ($TC = 0$).

If $TC = 1$, the request falls back to TCP, a common occurrence in DNSSEC response validation. The TCP exchange introduces additional overhead, involving ten packets: a DNS request, a DNS response, and eight TCP control packets (three for the handshake and five for session termination). In *PowerDNS*, the UDP response size limit is controlled by the `udp-truncation-threshold` parameter, which defaults to 1232 bytes.

The packet cost further includes each sent-received packet pair in the resolution process. Therefore, we define: - The resolver's packet cost as C_{pr}. - The attacker's packet cost as C_{pa}.

The *bytes cost* (C_b) represents the total number of payload bytes transmitted: - C_{br} denotes the number of payload bytes the resolver processes to answer a request. - C_{ba} represents the number of payload bytes in the attacker's request. **BAF:** The *bandwidth amplification factor (BAF)* is defined as the bandwidth multiplier: $BAF = \frac{C_{br}}{C_{ba}}$ **PAF:** The *packet amplification factor (PAF)* measures the ratio of packets processed by the resolver to answer a request compared to the number of packets sent by the attacker: $PAF = \frac{C_{pr}}{C_{pa}}$ **Maximum Throughput:** The resolver's maximum queries-per-second (`qps`) throughput is determined as the point at which the server begins dropping queries, causing the response rate to plateau. Indicating that the server has reached its capacity and is losing requests.

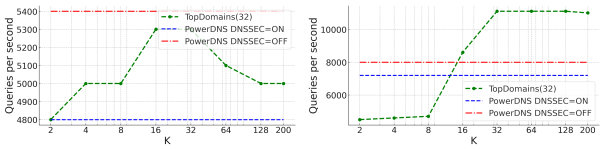
Collecting Statistics Seven datasets are used to study the *PowerDNS* resolver operation. To analyze the resolver's performance during normal operation, we rely on two datasets, while the remaining five are employed to evaluate its behaviour under `NXDOMAIN` attacks, where the resolver itself is the target. **Dataset A:** Lab-generated trace of the top million domains list [21]. While analyzing several recorded traces of live DNS traffic (each contains $\sim$ 1 million queries), we observed that they contain approximately 68,000 unique domains. To generate a similar trace to live DNS traffic, we generated DNS `'A'` requests (IPv4 resolution) for 67,997 domains in the trace with the same distribution as the live DNS traffic. The trace contains 1,360,000 queries, comprising 44,628 non-DNSSEC-protected domains and 23,369 DNSSEC-protected domains from the same list. **Dataset B:** Live DNS traffic trace obtained from the DNS-OARC GitHub repository [22]. Out of 1,000,000 total queries, we extracted 662,698 `'A'` queries, including 251,821 unique ones. The trace contains 13,962 secured domains. We include it to ensure our results hold on real-world, resolver traffic rather than only synthetic workloads. **Attack Dataset:** We created a dataset containing one million requests with bogus queries for distinct DNSSEC-protected domains to simulate an attack scenario. **Attack2 Datasets:** We generated three `NXDOMAIN` attack datasets consisting of 600,000 requests targeting the following domains: - `"Xfinity.com"` with 2,796 subdomains, - `"Paypal.com"` with 4,574 subdomains, and - `"Visma.com"` with 5,204 subdomains.

We scanned and used the dataset files as input to the `resperf` stress tool by Nominum [23], running

on the client machine, which acts as both a legitimate client and an attacker, depending on the experiment performed. Using this tool, the client sends a query stream of many unique DNS `'A'` requests to the resolver. The resolver's 1GB cache is empty at the beginning of each experiment. We record the traffic between the recursive resolver and the authoritative hierarchy and collect statistics from the *PowerDNS* recursive resolver.

Finally, we benchmarked the performance limitations of our own resolver that was erected for research purposes (rather than any commercial resolver). We used relatively short traces where each authoritative server receives a negligible number of requests (according to its normal operation) to evaluate the resolver's throughput and latency.

Selecting k Value We measure the maximum queries/sec (qps) throughput of the recursive resolver as a function of k to select the k value that yields the best performance. Figure 4 illustrates the performance of PowerDNS recursive resolver during normal operation and under NXDomain attack with the resolver as its target. The maximum qps of the original PowerDNS resolver, configured with DNSSEC-validation enabled and disabled, constitute baselines. Figure 4a illustrates the results during a normal operation where the performance for the values $k = 16$ and $k = 32$ is very similar to those of the original non-DNSSEC validating resolver and performs better than the original DNSSEC validating resolver. The reason is that in TopDomains(k), the DNSSEC validations occur only for the k domains' responses at a time and not for all responses arriving at the resolver (as in the original DNSSEC validating resolver). Therefore, when the resolver is configured to disable DNSSEC validation, it yields better performance as it does not suffer the overhead of DNSSEC validation processing for each DNSSEC signed zone encountered during the query resolution process. The following figure, 4b, illustrates the results during an attack on the resolver. Starting from the value $k = 32$, we get the finest performance compared to the two other PowerDNS versions due to the limitation of DNSSEC validation combined with the benefits of aggressive cache, which outweigh the overhead of our implementation. On the other hand, the DNSSEC-validating resolver exhibits inferior performance due to computational overheads. In contrast, the non-validating resolver demonstrates better performance, as it does not incur the overhead of response validation. However, it must still await responses from authoritative servers and is unable to synthesize responses due to the lack of aggressive caching capabilities. Notably, without DNSSEC validation, the resolver cannot employ aggressive caching as a mitigation strategy to shield authoritative name servers from NXDomain-based attacks. Consequently, we selected a value of $k = 32$, as it offers a favorable balance between performance and resilience, both under normal conditions and during an attack.



(a) Normal operation (b) Under NXDomain attack

Fig. 4: PowerDNS with TopDomains(k) performance measurement with different k_s

Cost and Amplification Analysis Table I depicts the amplification ratio and the factors mentioned in section IV. We tested the resolver with two queries with and without DNSSEC (DO=1: DNSSEC-enabled, otherwise DO=0). One legitimate query with a NOERROR response message (RCODE:0) and the other with NXDOMAIN (RCODE:3) [1]. As mentioned in II-A, DNSSEC validation enables an amplification factor of 3X to 18X (PAF) and 5X to 20X (BAF). Note that the responses contain multiple DNSSEC signatures, which exceed the maximum response size of the configured parameter *udp-truncation-threshold*. As a result, the TC bit is set to on, forcing the resolver to repeat the same query over TCP (which creates significant overhead on the resolver and the authoritative server to handle the TCP connections). The table highlights the potential of our approach, which limits DNSSEC validation to k domains only.

TABLE I: Amplification factor for queries when DNSSEC validation is enabled and disabled.

RCODE	0)	0	3	3
DO bit	1	0	1	0
TC bit	1	0	1	0
Attacker C_{pa}	2	2	12	2
Resolver C_{pr}	36	6	39	14
PAF	$18X$	$3X$	$3X$	$7X$
Attacker C_{ba}	406	110	1674	166
Resolver C_{br}	8017	957	7990	1302
BAF	$20X$	$9X$	$5X$	$7X$

TopDomains(32) evaluation under attack During an NXDomain attack, we expect TopDomains(32) to process fewer packets than the original DNSSEC-validating resolver since we limit the amount of DNSSEC traffic. We expect slightly more packets than the original non-validating resolver as we still perform DNSSEC-validation to protect authoritative name server from such an attack. Figure 6 shows the three described datasets' results. Observe that the behavior is as expected: TopDomains(32) is always between the two

[1]NOERROR: indicates that the DNS query was completed successfully, denoted by the Response Code 0 (RCODE:0). NXDOMAIN: indicates that the queried domain name does not exist, denoted by Response Code 3 (RCODE:3). https://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-6

TABLE II: PowerDNS under NXDomain attack

	D0=1	D0=0	TopDomains(32)
Max requests/sec	7200	8000	11200
Avg. latency (ms)	626	134	68
Median latency (ms)	94	59	0.5
Std latency (ms)	1380	600	282

extremes and often processes as few recursion packets as the non-validating resolver while providing an additional security feature of limiting traffic to attacked authoritative name servers.

Table II provides additional insight into the attack scenario, as it compares the original DNSSEC-validating and non-validating resolver to TopDomains(32). As observed, TopDomains(32) attains over $x1.5$ more throughput than the original DNSSEC-validating resolver and a massive reduction of circa $X10$ in average latency with an even more noticeable improvement in median latency. The reason is that the original validating resolver performs DNSSEC-validation processing for non-attacked domains in addition to the processing of the attacked ones, outweighing its benefit from aggressive caching. At the same time, TopDomains(32) brings almost all the benefits of an aggressive cache with a fraction of the overheads.

Figure 5 shows the cumulative distribution of the queries' latency values between 0 and 5 seconds. When the recursive resolver is under NXDomain attack, comparing the three PowerDNS resolver versions as above, we can see that both DNSSEC-validating and non-validating resolvers suffer greatly under attack regarding query latency. The query latency of the original validating resolver is extensive as it has to validate every response, while the non-validating resolver does not have to validate each response; it still has to wait for the authoritative servers to reply to bogus queries.

Due to its properties, TopDomains(32) protects the attacked domains and resolves queries quickly. To show it, we simulated an NXDomain attack through a single resolver that we controlled (Attack 2 trace). We run the attack trace slowly to not impact the authoritative server. In the simulation, the resolver starts with an empty cache and we measure the ratio of the attack requests forwarded to the authoritative server. We repeated the experiment for three subdomains each with thousands of subdomains. Our results show that with DNSSEC=ON and with TopDomains(32) less than 1% of the attack traffic reaches the authoritative server in all cases. Whereas with DNSSEC=OFF all the attack traffic reaches the authoritative server. This experiment shows that the use of TopDomains(32) protects the authoritative servers as much as the current implementation of aggressive caching in PowerDNS.

Normal Operation Our main goal is to ensure the TopDomains(32) does not degrade performance during normal server operation. As a result, we observe response delays and resolution failures (i.e., the number of SERVFAIL and NOERROR responses is not higher than that observed in the original PowerDNS).

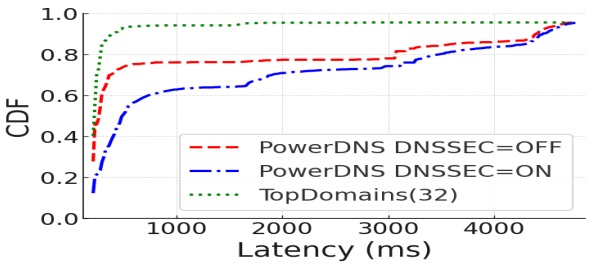


Fig. 5: CDF of latency of queries under attack: Comparison between original PowerDNS and TopDomains(32)

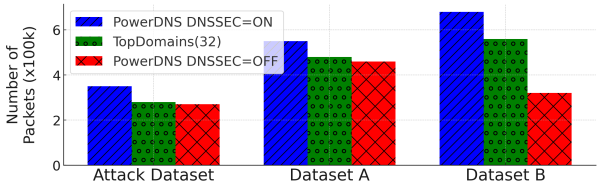


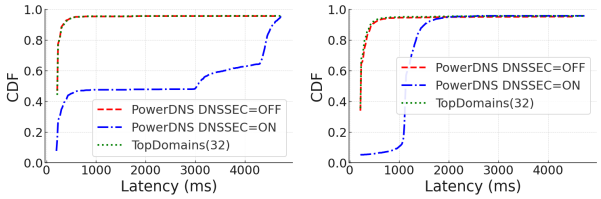
Fig. 6: Total packets exchanged by PowerDNS in the resolution of the three datasets' query streams

Table III shows that TopDomains(32) exhibits a slightly faster response time than the original DNSSEC-validating resolver and is very similar to the non-validating resolver in Dataset A (Alexa's top domains). However, in Dataset B (DNS-OARC trace), the original DNSSEC-validating resolver performance is far lesser than that of the TopDomains(32) and the non-validating resolver. That is because although Dataset B contains a relatively small number of queries of DNSSEC-signed domains, the number of DNSSEC-protected authoritative name servers participating in the resolution process (in the NS referral responses) is more significant. Meanwhile, the DNSSEC-validation process is performed only for the top k domains extracted from the queries received at the resolver (and not from the NS referral responses). Hence, TopDomains(k) reduces the overhead of the DNSSEC-validation for Dataset B.

Figure 7 shows the cumulative latency distribution for Dataset A and B. We can see that the original DNSSEC-validating resolver again suffers significantly due to the relatively long time the DNSSEC-validation processing takes, which increases its latency. However, during normal operation, the non-validating resolver does not suffer the validation cost; thus, it is very fast. The active defense on the selected domains had little impact on latency compared to non-validating resolvers. However, under attack it out perform both.

TABLE III: Comparing original PowerDNS when DNSSEC validation is enabled and disabled and TopDomains(32) during the resolution of query streams of Datasets A and B.

	Dataset A (Alexa's domains)			Dataset B (DNS-OARC)		
Resolver impl.	DNSSEC=ON	DNSSEC=OFF	TopDomains(32)	DNSSEC=ON	DNSSEC=OFF	TopDomains(32)
Total Req.	1,360,000	1,360,000	1,360,000	662,698	662,698	662,698
Secured domains	23,369	23,369	23,369	13,962	13,962	13,962
Unique Req.	465,292	465,292	465,292	251,821	251,821	251,821
Recursion pkts	567,694	450,405	453,435	672,478	371,230	528,140
%NOERROR	70.13%	71.66%	72.76%	69.37%	68.84%	69.5%
%SERVFAIL	2.61%	0.56%	0.51%	1.28%	1.29%	1.15%
%NXDOMAIN	27.26%	27.78%	26.74%	29.34%	29.87%	29.35%
Mean latency	49.14	41.43	42.52	1138.80	78.52	56.34
STD latency	239.64	181.01	185.57	818.57	287.76	211.4
Median latency	0.44	0.44	0.41	998.64	0.68	0.45



(a) 67k Alexa's domains (b) DNS-OARC trace

Fig. 7: TopDomains(32) and vanilla PowerDNS

V. Conclusion and Future Work

Our work demonstrates a fundamental dilemma in DNS resolver design. While it is desirable for resolvers to block `NXDOMAIN` attacks and protect authoritative name servers, achieving this requires performing DNSSEC validation on all traffic. However, this approach reduces the resolver's throughput by over 10% and increases the average latency. Under a volumetric attack, such an overhead is likely unsustainable. We propose a novel approach, *TopDomains(k)*, that enables the resolver to selectively employ DNSSEC validation solely to protect authoritative servers from `NXDOMAIN` attacks. Our approach demonstrates performance comparable to operating without DNSSEC (which is evidently faster than using DNSSEC) while still safeguarding authoritative servers against `NXDOMAIN` attacks. Our study suggests that resolvers should leverage DNSSEC validation, even when not using DNSSEC for general security purposes, to better utilize their caches and enhance the protection of authoritative servers.

References

[1] M. Anagnostopoulos, G. Kambourakis, P. Kopanos, G. Louloudakis, and S. Gritzalis, "Dns amplification attack revisited," *Computers & Security*, vol. 39, pp. 475–485, 2013.

[2] Y. Liu, Z. Wang, and N. Li, "Characterizing the impact of ddos attack on inter-domain routing system: A case study of the dyn cyberattack," in *2018 International Conference on Computer Science, Electronics and Communication Engineering (CSECE 2018)*. Atlantis Press, 2018.

[3] R. Radu and M. Hausding, "Consolidation in the dns resolver market–how much, how fast, how dangerous?" *Journal of Cyber Policy*, vol. 5, no. 1, pp. 46–64, 2020.

[4] X. Chen, H. Wang, S. Ren, and X. Zhang, "Maintaining strong cache consistency for the domain name system," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 8, pp. 1057–1071, 2007.

[5] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "Protocol modifications for the dns security extensions," Tech. Rep., 2005.

[6] M. Dooley and T. Rooney, "Dns vulnerabilities," 2017.

[7] S. Son and V. Shmatikov, "The hitchhiker's guide to dns cache poisoning," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2010, pp. 466–483.

[8] S. Bortzmeyer and S. Huque, "NXDOMAIN: There Really Is Nothing Underneath," RFC 8020, Nov. 2016. [Online]. Available: https://rfc-editor.org/rfc/rfc8020.txt

[9] K. Fujiwara, A. Kato, and W. Kumari, "Rfc 8198-aggressive use of dnssec-validated cache," 2017.

[10] D. Atkins and R. Austein, "Threat analysis of the domain name system (dns)," Tech. Rep., 2004.

[11] C. Rossow, "Amplification hell: Revisiting network protocols for ddos abuse." in *NDSS*, 2014, pp. 1–15.

[12] Cloudflare. (2020) World's largest 2.54 tbps ddos attack launched using several networks to spoof 167 mpps (millions of packets per second), september 2017. [Online]. Available: https://www.cloudflare.com/learning/ddos/famous-ddos-attacks/

[13] X. Luo, L. Wang, Z. Xu, K. Chen, J. Yang, and T. Tian, "A large scale analysis of dns water torture attack," in *Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence*, 2018, pp. 168–173.

[14] R. van Rijswijk-Deij, A. Sperotto, and A. Pras, "Dnssec and its potential for ddos attacks: a comprehensive measurement study," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, 2014, pp. 449–460.

[15] R. Sommese, K. Claffy, R. van Rijswijk-Deij, A. Chattopadhyay, A. Dainotti, A. Sperotto, and M. Jonker, "Investigating the impact of ddos attacks on dns infrastructure," in *ACM Internet Measurement Conference*, 2022, pp. 51–64.

[16] P. Vixie, "Rate-limiting state," *Communications of the ACM*, vol. 57, no. 4, pp. 40–43, 2014.

[17] S. L. Feibish, Y. Afek, A. Bremler-Barr, E. Cohen, and M. Shagam, "Mitigating dns random subdomain ddos attacks by distinct heavy hitters sketches," in *Proceedings of the fifth ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies*, 2017, pp. 1–6.

[18] J. Bushart and C. Rossow, "Anomaly-based filtering of application-layer ddos against dns authoritatives," in *2023 IEEE 8th European Symposium on Security and Privacy (EuroSP)*, 2023, pp. 558–575.

[19] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, Apr. 2005.

[20] G. Einziger, R. Friedman, and B. Manes, "Tinylfu: A highly efficient cache admission policy," *ACM Transactions on Storage (TOS)*, 2017.

[21] Majestic. (2022) Top million root domains list. [Online]. Available: https://majestic.com/reports/majestic-million/

[22] DNS-OARC. (2022) Sample query data files recorded dns traffic. [Online]. Available: https://github.com/DNS-OARC/sample-query-data

[23] Nominum. (2022) resperf performance tool manual. [Online]. Available: https://www.dns-oarc.net/files/dnsperf/2.0.0.0/resperf.pdf