Anomaly-based Filtering of Application-Layer DDoS Against DNS Authoritatives

Jonas Bushart
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
jonas.bushart@cispa.de

Christian Rossow
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
rossow@cispa.de

***Abstract*—Authoritative DNS infrastructures are at the core of the Internet ecosystem. But how resilient are typical authoritative DNS name servers against application-layer Denial-of-Service attacks? In this paper, with the help of a large country-code TLD operator, we assess the expected attack load and DoS countermeasures. We find that standard botnets or even single-homed attackers can overload the computational resources of authoritative name servers—even if redundancy such as *anycast* is in place. To prevent the resulting devastating DNS outages, we assess how effective upstream filters can be as a last resort. We propose an anomaly detection defense that allows both, well-behaving high-volume DNS resolvers as well as low-volume clients to continue name lookups—while blocking most of the attack traffic. Upstream ISPs or IXPs can deploy our scheme and drop attack traffic to reasonable query loads at or below 100k queries per second at a false positive rate of $1.2\,\%$ to $5.7\,\%$ (median $2.4\,\%$).**

1. Introduction

The Domain Name System (DNS) is at the core of Internet operations. Nearly all online services rely on DNS. Consequently, any DNS outage has broad consequences. For example, in July 2021, major online services, incl. Airbnb, FedEx, UPS, and large gaming networks (Steam, PlayStation Network), were unavailable for 30–60 minutes due to a DNS configuration issue at Akamai. In October 2016, a Distributed Denial-of-Service (DDoS) attack against DNS provider Dyn resulted in an hour-long downtime of Twitter, Amazon, Netflix, Spotify, and others. DNS outages become particularly critical higher up in the hierarchy. If top-level-domain (TLD) authoritatives are unresponsive, it interrupts the entire ecosystems underneath. The `.com` zone alone delegates more than 160 million domains to name servers lower down in the hierarchy. TLD outages have severe consequences for all services relying on this TLD. Any name lookups of these domains (or subdomains) strictly depend on the availability of the TLD name servers. While caching at DNS resolvers may delay the attack effect in some cases, caching is only effective for recently-resolved domains and clearly does not help against DNS outages in the long run.

Therefore, TLD operators aim to maintain high availability and responsiveness. They design the DNS infrastructure to be less susceptible to temporal traffic spikes. Redundancy and anycast infrastructures are typically at the core of these efforts. Most TLD zones are operated in one or more anycast "clouds", i.e., multiple geographically scattered name servers announce the same name server IP address. DNS resolvers will transparently contact the best (typically, "closest") anycast location for name resolution. Anycast is a significant pillar against DoS attacks. Single-sourced DoS attacks can only target a single anycast location, not the entire cloud at once. Likewise, DDoS attacks lose power, as they are automatically scattered among multiple anycast locations. Furthermore, when under attack, DNS operators can strategically update their BGP announcements to shift traffic from overloaded sites to those that still have resources. Rizvi et al. recently proposed so-called network playbooks that systematically automate this process [41]. Furthermore, DNS operators can use so-called *scrubbing services* to filter volumetric DDoS attacks that would otherwise overload the overall aggregated capacity of all anycast locations. For example, amplification attacks [42] allow adversaries to multiply their bandwidth by orders of magnitude and scatter the attack traffic across multiple sources—resulting in distributed target bandwidths that reach $\mathrm{Tbit/s}$. Having said this, amplification attack traffic does not follow normal usage patterns of requests-facing authoritative name servers [27], [28] and can be trivially filtered upstream.

But can existing defenses cope with targeted DDoS attacks against DNS infrastructures? In contrast to volumetric attacks, DNS application-level attacks blend into normal DNS traffic that follows valid semantics. Consequently, such attacks evade content-based filters. Moreover, as we will show, powerful application-level DoS attacks can easily overload DNS anycast infrastructures. For example, botnets may send massive volumes of benign-looking DNS queries that exceed the total (bandwidth or computational) capacity of DNS anycast setups. Even strategic traffic rerouting via BGP cannot mitigate this threat if the overall provisioning capacity falls short of the attack resources. The recent past has already documented application-level incidents in the form of random-subdomain attacks [10], [49], [54] (which force authoritatives to respond to non-existing query domains), and other DNS-specific attacks are possible [7], [27], [35]. When existing defenses such as content-based filters and anycast fail, what remains? If DNS operators lack resources to mitigate massive attacks on their own, upstream filtering remains the ultimate resort to prevent severe outages.

In this paper, we explore if upstream providers can leverage past resolver behavior to shield downstream DNS infrastructures. Given that attackers can evade content-based filters, we explore anomaly detection as *ultima*

ratio. Our goal is to enable upstreams to reliably filter malicious traffic *before* it reaches the authoritative name servers. To this end, we leverage past DNS lookup profiles derived from NetFlow statistics [4], [8], [9] to build behavioral profiles of DNS resolvers. TLD operators can learn such models before the attack and send them upstream—e.g., to scrubbing services or upstream providers using BGP Flowspec [32]—to filter attack traffic on demand. In principle, upstream filters can shield DNS infrastructures from abusive queries, mitigating application-level attacks that flood authoritatives with semantically-valid lookups.

When designing DDoS defenses for DNS, we have to overcome several obstacles. First, the set and behavior of DNS resolvers are dynamic due to network churn and temporal activities. Thus, anomaly detection cannot assume a static set of resolvers in an attempt to identify benign clients. Second, to retain net neutrality, we would like to avoid giving preference to large resolver operators (e.g., Quad9, Google DNS). That is, a defense should not favor "heavy hitters" (e.g., due to their sheer query volume), as not to discriminate against smaller decentralized DNS resolvers. And finally, if attackers can abuse resolvers that also serve benign users, DNS operators cannot clearly distinguish between "good" and "bad" resolvers.

In collaboration with a large European TLD operator, we propose a two-layered defense applied upstream to address these challenges. A low-pass filter allows DNS queries to pass if a resolver is below a particular query volume. We augment this low-pass filter with an allowlist of well-behaving, high-profile resolvers extracted from past behavioral profiles. We test our methodology on a real-world dataset of a large country-code TLD (ccTLD) operator that hosts one of the ten most popular TLDs worldwide[1]. To this end, we model two application-level DDoS attackers: (1) leveraging DDoS-capable botnets and (2) abusing open DNS resolvers. We then align attack data following these models with query behaviors recorded at the global anycast network of the TLD zone. Data fusion grants insights into the effectiveness and accuracy of the proposed countermeasure. We show that by training just one hour before the attacks, our last resort defense can reduce the attack traffic to reasonable levels at or below 100 kpps (packets per second) while dropping only 1.2 % to 5.7 % (median 2.4 %) of queries from benign clients.

To summarize, we present the following contributions:

- We provide insights into the daily operations of a large ccTLD provider and demonstrate the typical client population of their anycast infrastructure.
- We model two powerful DNS application-layer (D)DoS attacks and evaluate their impacts on the distributed TLD infrastructure.
- We introduce a low-pass filter based on anomaly detection that can be readily deployed upstream to mitigate application-layer DDoS attacks against Authoritative Name Servers (AuthNSes).

2. Background

This section outlines how modern authoritative DNS infrastructures are designed and how they achieve performance and resilience.

1. Source: Statista [46]; due to an NDA we cannot name the operator.

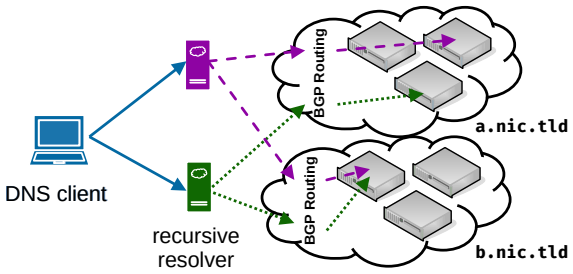


Figure 1: Example AuthNS setup with two `NS` records for two anycast clouds (a.nic.tld). Each cloud has multiple sites (blue servers). A resolver sends traffic to a specific cloud; BGP routing determines the site within a cloud.

2.1. Modern AuthNS Infrastructures

Redundancy is key to ensuring availability, resilience, and load balancing. Instead of only providing a single large DNS server, the service is distributed across multiple servers. This limits the impact of a broken server and allows for better performance by placing these close to the clients, thus reducing network latency.

Modern DNS infrastructures leverage two core mechanisms to obtain redundancy, both are visualized in Fig. 1. First, DNS operators announce multiple IP addresses for each zone, such that clients can choose a "good" server. A zone lists the AuthNSes in the `NS` records—at least two, but often more. Each `NS` record can point to one or multiple IP addresses via the `A` (IPv4) and `AAAA` (IPv6) resource records. A DNS client sends queries to any such IP address. Usually, the client would pick a low-latency name server and leverage the fallback options in case the preferred server is unresponsive.

Second, DNS operators use anycast and load balancers to place multiple servers behind the same IP address. BGP-based anycast allows announcing the same IP address space from multiple physical locations. Depending on the client's network origin, one of these anycast locations will receive and respond to the DNS request. The set of clients each AuthNS serves is its *catchment*. The catchment depends on the BGP announcement, the upstream provider connections, and the total number of locations announcing the same prefix. BGP allows operators to withdraw route announcements for faulty locations or add new locations to distribute the load more evenly. The convergence time is lower than the Time-to-Live (TTL) of `NS` records, providing faster reactions. By using anycast, DNS infrastructures can place servers in the physical proximity of their clients, reducing latency, especially if it avoids transcontinental traffic.

Not surprisingly, most popularly-used DNS infrastructures, therefore, use such anycast setups, as described by Sommese at al. [44]. They measured DNS infrastructures between 2017 and 2021. They find that 76.2 % of the TLDs consistently use anycast, and only 3.4 % are solely using unicast—the rest has a mixed setup. Furthermore, even the majority of *second*-level domains (SLDs) are now using anycast, with an ever higher rate among new SLDs. The adoption of anycast for SLDs is largely driven by big operators such as GoDaddy or Cloudflare. Given that these operators host millions of domains, their choice of anycast has a large impact.

2.2. DDoS Mitigation at the AuthNS

Regardless of the exact setup, DNS deployments face two main performance limitations: (i) compute power and (ii) network bandwidth [24]. Compute power is especially important for large zones with millions of registered domains, such as multiple European ones [3], [14], [39], [43], [55]. Large zones require more memory to store, and lookups become more expensive due to cache misses and necessary comparisons. Upload bandwidth can become scarce for DNSSEC-signed zones [53]. Cryptographic keys can be multiple kilobytes in size, and signatures take up hundreds of bytes. In the worst case, responses are up to $4\,\mathrm{KiB}$ large, where fewer than $500\,\mathrm{kqps}$ can exhaust a $10\,\mathrm{Gbit/s}$ connection. An AuthNS can respond to DNS queries in the range of $100\,\mathrm{kqps}$ to $10\,000\,\mathrm{kqps}$ [16], [24], [25], depending, among other things, on the zone file, software, configuration, and network connection.

To cope with traffic spikes, DNS operators leverage *over-provisioning* and provide more bandwidth and compute power than necessary for daily operations. Many DNS operators over-provision about $10\times$ to $100\times$ more than the daily peak [11], [16], [24], [48]. This is enough to mitigate many small DoS attacks but is powerless against targeted DDoS attacks. Furthermore, over-provisioning is expensive, as it entails buying services and keeping them unused for most of the time. Being economically more attractive, on-demand scaling allows adding new resources to meet rising demands. It can cope with demands far larger than solvable with over-provisioning without permanently investing in unused infrastructure. Having said this, scaling needs to be set up and tested before an attack. Further, operational and policy constraints may apply and making this mitigation difficult of unfeasible to deploy.

Scrubbing services turn the concept of over-provisioning into a service. Here, the defense capability is shared and amortized among many customers. A scrubbing service announces the IP address space of the attack target to "steal" its traffic, then filters the traffic, and finally, forwards the cleaned traffic via a tunnel to the customer. Scrubbing services are effective against volumetric attacks, such as amplification attacks, as they carry characteristic patterns. In such attacks, an AuthNS should never receive DNS responses but only DNS queries. However, they fail on application-layer attacks if the semantically valid attack traffic does not *per se* allow separating benign from malicious intent. This is further hampered if the scrubbing service is enabled only on demand since that prevents the service from learning the typical traffic profile and identifying anomalies.

Finally, there are mitigations if attacks only overload a single anycast location but not the entire anycast network capacity. Operators can then use BGP to steer traffic from overloaded anycasted servers to less-utilized ones [41]. This shifts the traffic such that no single location is overwhelmed. The efficiency of this mitigation depends on the BGP capabilities at each location. Generally available techniques, such as BGP path prepending, are imprecise and only remove traffic from the prepended locations. Either way, BGP rerouting only works for smaller attacks, but as we will show, it cannot protect against large DDoS incidents that overload the entire anycast infrastructure.

3. Problem Statement

Attacks against the DNS infrastructure have devastating effects [12], [23], [36], [45]. In this work, we focus on powerful application-layer attacks in particular, as they—surprisingly—represent an attack class for which DNS operators currently lack solid defenses. This section clarifies our threat model (Section 3.1) and then shows that existing defenses do not cope well with application-layer attacks (Section 3.2).

3.1. Threat Model

In this work, we consider a powerful application-layer attacker that aims to overwhelm AuthNSes with valid DNS queries. We focus on this attack category, as such attacks are harder to defend at the network layer (e.g., by scrubbing services). Unlike semantics-ignoring floods, defenses against application-layer attacks require application context. That is, we assume attacks that flood AuthNSes with syntactically and semantically valid DNS queries, either directly or via intermediaries (recursives). Requests can be for random non-existing names, similar to random subdomain attacks [10], [49], or to the millions of existing domains [3], [14], [39], [43], [55]. For the remainder of this work, the actual query content is irrelevant. In fact, we assume that a defense cannot filter for benign requests based on their content.

We envision attackers that use real systems to attack directly (e.g., by a botnet) or indirectly (by using closed and open resolvers as intermediaries). Using intermediaries allows attackers to hide the true originating source IP address and abuses resolvers that benign users might also use. This makes detection a bit trickier, as we may falsely filter traffic of benign users ("false positives").

We assume that attack traffic does *not* use IP address spoofing. Spoofed traffic can be filtered upstream, e.g., by enforcing that clients complete a handshake. This can be achieved with TCP, which is an implementation requirement [15] for any DNS software, or by using DNS Cookies [1], [47]. Both the TCP handshake and DNS Cookies require the client to echo back a value sent by the server, thus enforcing validity of the source IP address.

3.2. DNS-Specific Non-solutions

Although various DNS features may mitigate DDoS attacks, they are no match for them in our threat model. One significant limitation is that many either require support inside the resolvers or run fully resolver-side. As such, an AuthNS cannot expect these techniques to function since (i) many resolvers lack support and/or (ii) direct attacks do not need to adhere to these features. For completeness, we iterate these defenses in the following.

Negative caching: Aggressive negative caching [18] enables a resolver to answer new queries without any additional requests. This only works for DNSSEC-signed zones and only if the `NSEC3` opt-out mode is not used. Dynamic signing [19], [57] can also prohibit the usage and DNSSEC deployment is still quite low, such that not many zones can benefit. Many TLDs are signed, but often using the opt-out mode since it requires fewer signed resource records, such that these TLDs cannot benefit.

Using DNSSEC induces extra load on the AuthNSes since more records need to be stored, and upload bandwidth can become a bottleneck due to the large size of public keys and signatures. Finally, negative caching does not apply to *existing* records—as TLD zones that usually have tens (or hundreds) of millions of records, attackers can abuse those despite caching.

Response Rate Limiting (RRL): RRL [20], [56] limits the number of responses a DNS server sends per client. RRL thereby effectively limits the impact of name servers during amplification attacks. However, filtering is done only *after* generating a response. RRL thus only reduces upload bandwidth but does not save inbound bandwidth or processing time. In fact, processing requirements may even increase due to additional bookkeeping. To be effective in our scenario, RRL would have to be applied at the DNS client side. Indeed, some recursives even implement limits on outgoing queries to an AuthNS [21], [22], reducing overall traffic once the AuthNS is overloaded. Having said this, attackers are not bound to adhere to this pattern and can avoid such resolvers or even implement their own iterative lookup logic.

Stale Records: Lastly, recursives may mitigate the impact of an AuthNS's outage and serve records past their TTL time, so-called *stale records* [31]. This is effective in reducing queries while the AuthNS is unresponsive, but requires resolver support and deprives the AuthNS of its power of updating the records' data. Stale records can thus only be seen as a fallback "solution" to a situation (outage) that should really be prevented more proactively.

4. Methodology

We introduce an anomaly filtering methodology to mitigate non-spoofed application-layer attacks against AuthNSes. Before doing so, we explain the design goals of our envisioned defense in Section 4.1, then present our proposed defense methodology in Section 4.2, and discuss its deployment in Section 4.3.

4.1. Design Goals

In consultation with a partnering TLD operator (cf. Section 5.1), we derive eight design goals for a DNS application-layer DDoS defense:

G1: Offer DDoS Protection, even for Large TLDs. First and foremost, the defense should mitigate DDoS attacks such that only a manageable traffic level reaches the AuthNS infrastructure. It is an explicit non-goal to eliminate *all* attack traffic; it is sufficient if any remaining malicious queries do not overload any of the AuthNS's resources. Typical AuthNS can easily cope with up to $500\,\mathrm{kpps}$, but struggle with loads higher than $10\,\mathrm{Mpps}$ [16], [24], [25]. Any defense should keep the attack volume below the former query rate, even if the AuthNSes hosts a reasonably large zone (e.g., a TLD).

G2: Independence from Client Support. The DNS ecosystem consists of various DNS resolver implementations, versions, and configurations [6], [26], [40], [50], [52]. Thus, any DDoS defense must not rely on AuthNS clients implementing certain optional DNS features that theoretically could mitigate an attack.

G3: Upstream Filtering and Learning. If attack traffic arrives at the AuthNSes, it is "too late". Therefore, we study defenses that can be applied upstream by providers or scrubbing services. The filter needs to integrate into the current environment of servers and anti-DoS solutions. They are the only locations with the capacity to remove attack traffic before it overloads the servers or data centers' network connection. The simplest filter with wide support is an allowlist/denylist of IP prefixes (in our setting, networks of recursives). The lists are easy to share, well understood, and have support for upstreaming, e.g., using BGP FlowSpec [32], [33]. An upstream provider should itself even be able to derive the filtering policies.

G4: Content-Agnostic Filtering. Defenses should not rely on DNS request content. First, we want to minimize privacy leaks and do not want to require an upstream to constantly inspect privacy-sensitive DNS communication content to filter attacks. Second, content filters are unreliable—attackers can easily adapt their tactics by querying arbitrary (semantically valid) DNS records.

G5: High Evasion Resilience. Acknowledging that any defense can be evaded by attackers with an unlimited budget, we aim to resilience against evasion to render evasion attempts intractable in practice.

G6: Low False Positive Rate. The ultimate dilemma any attack prevention mechanism faces is that it has to distinguish between benign and malicious intent. In our setting, we believe that a defense does not need to filter *all* malicious or anomalous traffic. The ccTLD operator has sufficient over-provisioning for everyday events, and together with caching in DNS resolvers, this is a solid baseline defense. In fact, overzealous blocking causes collateral damage. Filtered benign requests harm clients such as critical infrastructures, given DNS's central role in many networks. Such false positives even risk amplifying the attack due to the retry behavior of clients.

G7: Ease of Use and Low Costs. We believe DDoS defenses have to be easy to use, especially in pressing and hectic times such as during a DDoS attack. Defenses should incur minimal costs before deployment and instead rely on *existing* network infrastructures and features. For example, the proactive collection of traffic statistics for IP prefixes can be done inexpensively on an AuthNS's router (or its upstream) in the form of NetFlow [4], [8], [9] data or a sampled stream of IP packets.

G8: Resource Neutrality. Defenses should not rely on adding resources at the DNS operator. While more resources may indeed provide a mid-term solution to higher loads, the defense should be able to handle temporal DDoS attacks that last 1–2 days max.

4.2. Anomaly Filtering

With these design goals in mind, we propose a defense based on anomaly detection specifically tailored for DNS operators. The key idea is to learn an expected traffic level for each source and use this to identify sources that send unexpectedly high amounts of traffic. This idea follows the observation that a DNS operator's traffic distribution matches a long-tail distribution, in which only a few sources are responsible for the vast amount of traffic. Consequently, we aim to filter traffic of DDoS attacks

that abuse previously unobserved high-volume clients, independent of the content of queries (G4).

We approach this by classifying resolvers into three different traffic behaviors. (1) *Low-volume clients* have a relatively low traffic volume and never show traffic spikes. Low-volume clients can be active throughout an entire measurement period or only for a short time. (2) In contrast, *steady high-volume clients* send a continuous, large stream of DNS queries. (3) Finally, and most challenging, we observe *sporadic high-volume clients* which emit high but sporadic query spikes, e.g., outlier spikes that only occur temporarily in a longer measurement interval.

To judge which category a client falls in, we monitor content-agnostic (G4) traffic statistics at the AuthNSes. Such data is available to both the DNS operator itself and its upstream providers (G3). We split a training dataset into time intervals and traverse this dataset using a sliding window of length W_{train} over time intervals of one hour. Per window, we count how many queries each source IP network (/24 for IPv4 and /48 for IPv6) sends. We regard a network as a steady high-volume client if it is active for at least *STEADY* time intervals and shows an average query rate of at least *HEAVY* packets per hour. All other networks either contain low-volume clients or sporadic high-volume clients only. Figure 2 shows one example run of this algorithm, where the first twelve windows are used for training, and the next six show the attack. The color during the attacks shows if traffic is unrestricted (green) or if some traffic is filtered (orange–red). The "active" and "max T_i" columns show if the value is large enough for allowlist inclusion.

Allowlist Creation: We now leverage this classification to build a defense that allows continuous operation of the benign clients while blocking abnormal query profiles. To this end, we build an allowlist that includes steady high-volume clients. For each client network i, we record its maximum past traffic level T_i, measured in packets per hour. We use this past traffic level to detect if an IP network sends unexpected amounts of traffic during the attack phase. To cope with situations in which a benign client increases its query volume, we introduce a tolerance factor *TOL*. During the filtering period, for an allowlisted client i, we only block all i's queries that exceed $T_i \times TOL$. This combination provides flexibility in how strict the filtering is as it compensates for traffic variability.

The allowlist does not capture low-volume and sporadic high-volume clients. We exclude low-volume clients to keep the allowlist reasonably small and manageable (G3). Ignoring sporadic high-volume clients makes it harder for an attacker to poison the training phase (G5). Yet, combined, these two query profiles likely still contribute a significant amount of the overall traffic. Just blocking all low-volume clients would create too many false positives, violating G6. Therefore, we permit a certain low traffic volume (*LPF*) for any traffic source not on the allowlist. Each non-allowlisted origin network can send as many queries as this low pass filter allows. Traffic exceeding this limit will be blocked. This policy allows low-volume sources to always reach the AuthNS—irrespective of whether the low-volume client sends only sporadic or continuous traffic. The *LPF* also allows queries of new low-volume clients, i.e., those not part of the training phase. The *LPF* only (but unavoidably) penalizes high sporadic traffic spikes of benign clients.

Site-specific Allowlists: Modern DNS infrastructures operate multiple logical AuthNS, separated by clouds (i.e., anycast IP addresses) and topological locations. We call each such logical AuthNS a *site*. Each combination of physical location and anycast IP address has a different catchment, meaning that IP networks visible on one anycast IP address for a location may not appear for the other anycast IPs. As client query behaviors may differ substantially between sites, and to minimize allowlists to ease deployment (G7), we train site-specific allowlists instead of a global one.

Filtering Phase: The filtering phase immediately succeeds the training phase and is assumed to contain attack traffic. During this phase, any source network may send up to *LPF* queries and allowlisted networks even more according to their past query volumes and the tolerance factor. Traffic that exceeds these limits will be blocked. We do not assume a certain length on the attack; we show in Section 5.4.4 the impact of attack lengths on detection accuracy. Our current prototype does not leverage retraining when under attack. That is, the allowlist is fixed as long as filtering is active, such that it ages and will slowly drift away from the current resolver population—an effect that we evaluate in Section 5.5. Having said this, we envision continuous retraining until a noticeable attack starts. This way, defenders have a recently trained detection model at hand when they most urgently need it.

4.3. Upstream Filtering

After having described *how* we filter traffic with our anomaly detection, we now discuss *where* filtering occurs. Filtering attacks at the AuthNS infrastructure comes too late. The downstream connection to the server or data center might already experience packet loss, or the server cannot handle filtering fast enough. As such, it is beneficial to offload filtering to upstream parties with sufficient bandwidth and processing power (G3). Commercial services such as *scrubbing services* or upstream ISPs/IXPs are perfect candidates. Such upstream services can filter and train the allowlist—all under the assumption that they have constant access to the network traffic to learn its characteristics. Indeed, the curated aggregated traffic statistics required for training can be captured/provided by NetFlow-like summaries that retain user privacy.

In some situations, upstream training is not possible or desired, though. If scrubbing services are enabled "on demand"—only to mitigate an ongoing attack—they lack past behavior to build accurate defense models. In fact, AuthNS operators may deliberately choose to enable scrubbing services only when really necessary (i) to save costs, (ii) to preserve user privacy, and (iii) to improve latency. In addition, upstream providers may lack the incentive to train behavioral models for their customers. In these cases, the DNS operator can train the allowlist themselves and push its filter decisions upstream, e.g., using standards such as BGP FlowSpec [32], [33] or proprietary APIs provided by upstream providers. The allowlist enforcement can then be implemented in software, such as using eBPF/XDP, or even closer to the hardware in P4-programmable switches.

window:	w1	w2	w3	w4	w5	w6	w7	w8	w9	w10	w11	w12	w13	w14	w15	w16	w17	w18	Active	Max T_i	Parameter	Value
Resolver A (allowlist)	19	28	21	18	15	18	29	29	30	35	33	31	26	22	20	17	22	18	12	35	HEAVY	10
Resolver B (allowlist)	11	14	4	7	19	15	5	4	8	19	9	10	88	74	99	86	70	77	12	19	LPF	20
Resolver C (LPF)	0	0	2	0	0	0	0	0	3	0	0	0	0	0	5	1	0	0	2	3	STEADY	5
Resolver D (LPF)	0	0	2	0	4	0	0	0	5	12	0	0	5	30	10	23	0	2	4	12	TOL	2

Figure 2: Application of our algorithm on an attack in windows w13 to w18 with four resolvers. The "active" and "max T_i" columns show if the value is large enough for allowlist inclusion. Resolvers A and B were added to the allowlist, and resolvers C and D were too low volume during the training phase. Resolver A behaves normally during training and attack, such that all its traffic passes the filter. An attacker abuses resolver B to launch an attack, such that we filter B's traffic (orange/red color), causing FPs in the benign fraction of B's traffic. Resolver C maintains a low query volume ($\leq$ *LPF*) and does not experience FPs. Finally, resolver D's temporal volume spikes cause a few FPs.

Costs: Finally, we analyze the costs (G7) for the envisioned deployment. We discuss the allowlist creation and filter enforcement separately. The allowlist maintenance does not incur extra costs. TLD operators can easily (or already do) record statistical traffic information such as NetFlow data. For moderate training intervals of a few days, storage costs are negligible; so are the computational costs for creating the allowlists, which mostly requires inexpensive statistical analyzes. The costs for filtering, i.e., the actual enforcement, are a bit harder to estimate. But assuming that larger TLD operators have subscriptions with scrubbing services anyway, they need agreements for allowlists paired with rate limiting—standard features of most scrubbing services. In the worst case, filtering requires a new appliance or P4-programmable switch, creating one-time costs in the range of $2000 per physical AuthNS location plus hosting fees. Overall, these worst-case costs are significantly lower than the economic damage of DNS outages at a TLD.

5. Evaluation

We now evaluate how effective an anomaly detection defense, as described in the previous section, can be against large-scale DDoS attacks. To this end, we acquire an attack-free DNS query dataset of a large TLD provider (Section 5.1). We then study the typical query behavior of benign clients in this dataset (Section 5.2). To model attacks, we obtain populations from two popular botnets and open resolvers (Section 5.3). We then train a detection model based on the benign dataset, optimize parameters, and evaluate the optimized model against the modeled attacks (Section 5.4). Finally, we assess to what extent concept drift (Section 5.5) and evasion attempts (Section 5.6) impede the proposed idea.

5.1. Dataset Description

We cooperate with one of the largest European (and global) ccTLD operators[2] to obtain a realistic evaluation dataset. The operator is authoritative for over 17 million domains, all of which are configured with a Time-to-Live (TTL) (caching) time of 24 hours. The anycast setup of our ccTLD consists of five anycast clouds, each having two to eight locations. A visualization is included in Section A. These locations are distributed over multiple countries. Overall, five to seven anycast locations are in the ccTLD country itself, Europe, and the rest of the world, respectively. Some physical locations are part of multiple clouds; we call a unique *(cloud, location)* tuple a *site* to resolve ambiguities.

2. We redact the concrete TLD for anonymity.

TABLE 1: Capture statistics of both anycast clouds in the NetFlow dataset. The prefixes and packets are split evenly.

Cloud	Locations	Prefixes v4	Prefixes v6	Packets
C1	8*	968 915	105 565	4 563 713 484
C2	7*	928 893	101 790	4 683 252 517

Our TLD partner gave us access to four weeks of traffic statistics recorded at sites in the two largest anycast clouds. The collection period was 2022-05-13 06:00 UTC to 2022-06-09 06:00 UTC. To make data size manageable for the operator, the recorded data is randomly sampled per IP packet at a rate of one-to-ten. Due to processing errors, one location is missing 18 hours, and another one was not monitored at all; having said this, the dataset still represents a vast majority of the DNS query behavior observed at these two zones. The traffic and observable IP networks are pretty even between the two anycast clouds, see Table 1. Overall, our dataset spans 9 246 966 001 inbound packets from 1 185 657 different IP networks. The traffic per physical locations ranges from 94 950 430 packets (31 544 networks) to 2 015 510 497 packets (from 511 063 networks). Due to the 1:10 sampling rate, the real number of packets (and hence, DNS requests) will be larger by a factor of ten.

Data Collection Ethics: Our partnering TLD provider minimized data collection to preserve user privacy to the best extent possible. First, the dataset only contains NetFlow data, such that no DNS query names or responses are recorded. Instead, we infer the number of queries from the number of packets per flow and direction, with the approximation that an IP packet corresponds to exactly one query (modulo sampling). Second, the collected data is aggregated based on Border Gateway Protocol (BGP) routing granularity of /24 for IPv4 and /48 for IPv6 to hide individual users. Third, the datasets were protected by strict access control.

Sampling Bias: Due to sampling, we miss nine out of ten DNS requests. Random sampling statistically guarantees that we only miss low-volume senders, all of which will pass any filter due to the LPF. This implies that any false positives rate reported in the following is an upper bound, and would likely be a few percent lower on unsampled data. To estimate *how many* IP networks we are missing due to the sampling, we inspect a secondary dataset that consists of 24 hours of unsampled traffic

recorded on 2021-01-14 from 14:57 UTC.

To this end, for each IP network in the unsampled dataset, we compute the probability that this IP network would not appear in a sampled dataset. This probability is $P = (1 - 0.1)^n$ for n queries received by the IP network. Summing the individual probabilities tells us how many IP networks we are likely missing. Indeed, our main dataset does not observe 30.5 %–42.9 % of IP networks per physical location, with a single exception where only 24.8 % of IP networks would be uncaptured. However, over 99.5 % of these IP networks send fewer than 32 queries during the whole day, and 99.99 % send fewer than 64 queries during the day. Even though we miss a significant portion of IP networks, these networks only contribute 3.7 %/5.4 % of the overall traffic.

5.2. Benign Client Behavior Analysis

Before turning to the evaluation of the defense itself, we analyze the DNS query behavior of client networks in our dataset. The behavior of the AuthNS's clients is quite diverse. Most IP networks send few overall packets and are ephemeral, only active for a couple of hours during the whole time. Most IP networks send fewer than ten packets per hour yet contribute less than 7.5 % of the recorded traffic. The networks with the highest query rates are active over 90 % of the time and are likely resolvers of ISPs, cloud operators, and public DNS offerings, since these have always active and large userbases. There are some high-volume clients with overall short temporal activity, which might be problematic for anomaly detection. A more detailed description can be found in Section D.

The overall very moderate query rates are not unexpected, as the resource records in the ccTLD zone can be cached for up to 24 hours. Overall, this seems to confirm that most client networks fit the proposed low-pass filter well. Having said this, the analysis also shows that there is a need for exceptions using an allowlist. Larger query rates, as observed in the heatmap, can occur for multiple reasons. First, clients can look up millions of different domains. The more popular a resolver, the more domains it will query. Second, some recursives do not adhere to the caching period and fetch data more frequently, which can cause hundreds of millions of queries over a day due to the larger zone file size. Third, some networks have a large concentration of cache-independent resolvers, all of which are merged in our analysis.

5.3. Attack Modeling

We now model attacks that target the DNS infrastructure as outlined in our threat model (Section 3.1). We thereby distinguish between the type of traffic sources an attacker chooses. As a single attack source will easily be blocked by the proposed filter without causing much collateral damage, we assume distributed attacks. We envision two possible attacks. First, as we explain in Section 5.3.1, attackers can relay attack traffic via DNS resolvers. Second, as we discuss in Section 5.3.1, a large network of compromised hosts ("botnet") may attack the AuthNSes directly. After discussing these attacker models in more detail, we use them in evaluating the proposed methodology. Figure 3 visualizes both attack vectors.

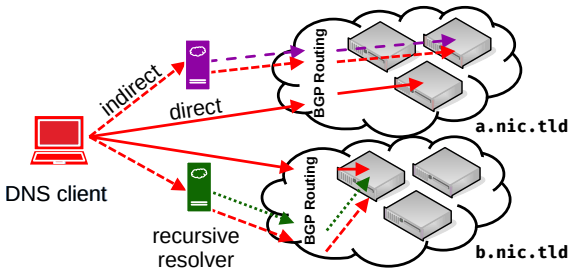


Figure 3: The attacker can attack indirectly by abusing benign resolvers (dashed lines) or query anycast clouds directly (solid lines). Both vectors lead to different sites (servers in the cloud). Indirect attacks increase the risk of false positives if regular clients use the same resolver for issuing benign requests (dotted lines).

5.3.1. Indirect Attack via Open Resolvers. Open resolvers represent an obvious choice for distributing an application-layer attack against DNS. The intermediary recursives hide the attacker's identity and enable distributed attacks even for single-homed attackers. Furthermore, resolvers are topologically distributed, and thereby also distribute the attack among all anycast locations.

The attacker can launch an indirect attack by abusing pre-configured resolvers of ISPs, and anycast or unicast open resolvers. The resolvers all act conceptually similar, but differences between anycast and unicast are important. Anycast resolvers, to which most public DNS offerings belong, have a wide range of locations sharing the same IP addresses. Thus it is likely that a location is "close" to the attacker and will be in the same catchment as the attacker. Alternatively, attackers can abuse unicast recursives. They can be selected such that they have diverse network views and can reach all locations of the anycasted AuthNS.

To model an attacker that abuses open resolvers, we scanned the IPv4 landscape of open resolvers via Zmap [17] and a custom DNS module. To this end, we follow the recommended scanning guidelines by Zmap and allow networks to opt out of the scans. The scans were performed on 2022-06-09 at reduced speed and lasted for 33 hours. Our scan generally identifies two types of resolvers: forwarders and recursive resolvers. That is, we find (1) the *client-facing* "scannable" recursives that receive and respond to the query, which may (optionally) forward the query to (2) *server-facing* recursives that perform the actual iterative resolution at the AuthNS. Furthermore, by including our own AuthNS infrastructure in the DNS resolution chain, we can map the client-facing scanned IP address (by encoding it into the query name) to the server-facing IP address of the recursives that perform the authoritative lookups.

Our IPv4 scan covered all geographic regions except Russia.[3] We find 919 174 scannable IP addresses in 374 464 networks of size /24 that contact our AuthNSes. The recursives forward the DNS queries to 48 263 recursives in 24 942 networks.

5.3.2. Direct Attack via Botnet. Instead of abusing resolvers, the attacker can also abuse a set of geographically

3. We excluded Russian target IP addresses from our scans as requested by our network operator due to the political situation.

diverse bots to steer traffic directly to the victim AuthNS. The diverse geographical locations of botnets allow the attacker to reach all locations in the anycast cloud(s) to attack all locations simultaneously.

To model botnet attackers, we analyze the population of two large in-the-wild botnets. (1) *Sality* is a peer-to-peer botnet targeting Microsoft Windows systems. On 2022-05-28 we discovered 53 824 IPv4 bots by crawling the Sality botnet. (2) *Mirai-like* bots are all hosts showing the typical SSH and Telnet protocol scanning behavior. We collect all SSH and Telnet connection attempts to a /20 large IPv4 darknet, i.e., a network without any hosts. We collected 152 486 IP addresses in 100 366 Mirai-infected IP networks between 2022-05-16 and 2022-06-22.

We then mapped these bots to their available attack bandwidth. To this end, we leverage the Measurement Lab [34] Network Diagnostic Tool (NDT) dataset, using the `measurement-lab.ndt.*` subset. This dataset was created via a dedicated network speed measurement website, where users can measure their Internet connection. IP addresses without a measurement get attributed the performance of the next smallest supernet, i.e., we take the data associated with the /24, /22, /20, or /18. If that is insufficient, we use the average bandwidth of all bots at a specific anycast location.

Using this dataset, we find that the Sality bandwidth ranges between 201 Gbit/s to 2827 Gbit/s (average 564 Gbit/s) while Mirai has between 1149 Gbit/s to 8947 Gbit/s (average 2675 Gbit/s). We use the maximum bandwidth for the later evaluations for a worst-case analysis. The average bandwidths translate to 705 Mpps and 3343 Mpps for Sality and Mirai, respectively. This analysis reveals that attacks would have a devastating effect on the anycast setup of our partnering TLD operator.

5.4. Learning Parameters / Accuracy

We now evaluate the effects of these attackers against the DNS infrastructures of our partnering ccTLD operator. To this end, we first define the metrics we will use throughout the evaluation (Section 5.4.1). We then describe how we select (Section 5.4.2) and optimize (Section 5.4.3) the input parameters of the anomaly detection. Finally, we evaluate how well the defense can mitigate the attacks (Section 5.4.4).

5.4.1. Evaluation Metrics. We follow standard terminology in our evaluation. We define malicious traffic as *positive* and correctly filtered DNS requests as *true positive* (TP). Conversely, benign traffic is *negative* and benign DNS requests that bypass the filter are *true negatives* (TNs). A *false positive* (FP) is erroneously blocked benign traffic, which can occur if a traffic source exceeds the allowed traffic level T_i or *LPF*, respectively. A *false negative* (FN) refers to undetected malicious traffic that bypasses the filter, e.g., all attack packets per source that do not exceed *LPF* packets. A FP negatively interferes with DNS lookups of benign clients and has negative consequences. In the best case, FPs just cause delays in DNS resolutions, potentially opening resolvers up for further attacks; in the worst case, they render entire services (e.g., websites) unavailable. In contrast, a FN just adds to the load and does not have any immediate negative consequences if there are sufficient infrastructural resources. In the following, we argue to minimize FPs while maintaining an acceptable level of FNs.

TABLE 2: Overview of the parameters we use in the methodology and evaluation.

Parameter	Description
HEAVY	Minimum traffic level (in packets per hour) before an IP network can be added to the allowlist.
LPF	Traffic level (in packets per hour), which is allowed without an allowlist entry for the IP network.
STEADY	Minimum number of active hours before an IP network can be added to the allowlist.
TOL	Factor to use on T_i to calculate the allowed traffic during the filtering phase, past which queries from the IP network will be blocked.
W_{test}	Windows size during the test phase, i.e., the attack length.
W_{train}	Windows size during the training phase.

Based on these classifications, we use standard evaluation measures, such as false positive rate $FPR = \frac{FP}{FP+TN}$ and false negative rate $FNR = \frac{FN}{FN+TP}$, and their inverses $TNR = 1 - FPR$ and $TPR = 1 - FNR$. To combine both FPs and FNs, we report on the F-score $F_\beta = \frac{(1+\beta^2)*TP}{(1+\beta^2)*TP+\beta^2*FN+FP}$ and balanced accuracy $BA = (TPR + TNR)/2$. Having said this, in the following, we optimize for and report on the FPR, as doing so still reaches an acceptable number of FNs.

5.4.2. Parameter Values. We now perform a grid search to find reasonable values for the *training parameters* we declared in Table 2. To this end, we first specify concrete value ranges for each parameter. We vary the training time (W_{train}) between one hour and 73 hours. The minimum traffic level for allowlist inclusion (*HEAVY*) ranges between 64 pph to 256 pph (packets per hour). The minimal active hours (*STEADY*) range from one to twelve hours, with the constraint that they are always smaller than W_{train}. The low-level traffic threshold (*LPF*) varies between 128 pph to 2048 pph. We also allow traffic sources to exceed the traffic levels (*TOL*) recorded in the allowlist by the factor one (no exceeding), two, or four.

The attack is determined by another set of *test parameters*. The attack starts immediately following the training phase (we will evaluate more outdated training models in Section 5.5) and lasts between an hour and three days (W_{test}). For the simulation, we model different attacker bandwidths from 1 Gbit/s to terabit speed, depending on the attacker model, and simulate different attack sources, such as open resolver and different botnets.

5.4.3. Parameter Optimization. We use a grid search on the aforementioned training parameters to find the best parameter choices. We consider those parameters ideal that minimizes the False Positive Rate (FPR). To this end, we leverage *sliding window cross-validation*, a standard cross-validation technique for time series that avoids mixing training and test data. It allows us to keep the chronological data order and still have multiple folds, as the test length is much smaller than our overall time series. In our evaluation, we test all parameter combinations and report on the one that worked best for a given site across all folds. Figure 4 illustrates the general idea.

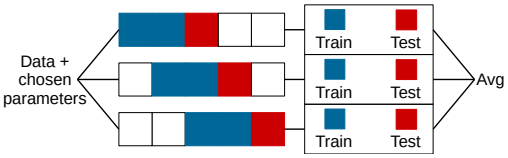
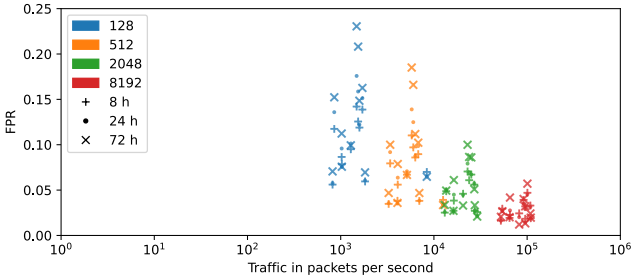
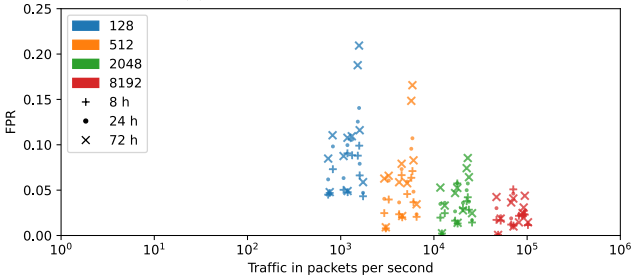


Figure 4: Symbolic representation of the grid search approach to find the best parameter combination. Given a fixed set of data and parameters, we pick distinct test intervals and compute the average of the performance.



(a) Attacker with Mirai-like



(b) Open Resolver Attacker

Figure 5: Effect of the *LPF* and W_{test} parameters on the FPR and FNs. Each point represents a site's performance depending on *LPF* and W_{test}. The graphs depict attackers using a Mirai-like botnet and open resolvers.

As an alternative cross-validation concept, we also tried an alternative cross-validation technique that optimizes Parameters per sliding window instead of using a fixed parameter configuration for all folds. This technique yields similar results yet requires constant parameter search. Consequently, in the following, we do not differentiate between the two methods and only report on results obtained from the global parameter search.

5.4.4. Results. Figure 5 shows results for an attack using the Mirai-like botnet (top), and open resolver (bottom). The same graph for Sality is in Section B. More detailed statistics about the Mirai-like botnet are presented in Table 3. The botnet uses the calculated maximum bandwidth, and the open resolver graph uses a theoretical 100 Tibit/s attacker to show the worst-case behavior in both cases. It shows the malicious traffic which needs processing after filtering (x-axis) in comparison to the FPR the filtering achieves (y-axis). Each point in the graph represents the evaluation result for a given site. Color shows different values for W_{test}, while the symbol shape represents *LPF*. As expected, the *LPF* has a significant impact on the false negatives (x-axis) while the respective site and W_{test} influences false positives (y-axis). In fact, the four chosen *LPF* thresholds result in four clearly separate clusters. The lower the attack duration, the lower the FPR. Some sites have a larger FPR than others, for all attack lengths.

The graph shows how we can trade better FPR scores by allowing larger amounts of malicious traffic. Larger *LPF* values mean that benign sources, which do not occur on the allowlist, are not as likely to be blocked and thus decrease the false positives by them. As a downside, more attack traffic will simultaneously be allowed. For example, consider *LPF* = 2048 pph, the second-highest *LPF* threshold in our evaluation. Here, the malicious traffic is generally quite low, with less than 13 000 pps (left) or 26 000 (right) at any single location. This volume is well in the realm of traffic an AuthNS can handle, and thus prioritizing further reductions are not relevant. For 8-hour long attacks and *LPF* = 2048 pph, the FPR is 4.2 % on median, ranging from 1.7 % to 6.9 % for the Sality botnet, while ranging from 0.2 % to 5.7 % with a median of 2.5 % for the open resolver evaluation.

W_{test} has a negative influence on the FPR due to population churn. As the allowlist ages, it no longer accurately mirrors the current state of benign resolvers.

TABLE 3: Detailed evaluation results for the Mirai-like attacker from Fig. 5a, since it has the worst FPR values. *Normal* counts each physical location identically, while in the *weighted* columns each location is weighted by the benign traffic levels. "Mdn" denotes the median.

		Normal			Weighted		
W_{test}	*LPF*	Min	Avg	Mdn	Avg	Mdn	Max
8	128	0.06	0.10	0.10	0.10	0.10	0.14
	512	0.03	0.07	0.07	0.07	0.07	0.11
	2048	0.03	0.04	0.05	0.04	0.05	0.07
	8192	0.02	0.03	0.02	0.03	0.02	0.05
24	128	0.06	0.11	0.10	0.11	0.10	0.18
	512	0.03	0.07	0.07	0.07	0.07	0.14
	2048	0.02	0.05	0.04	0.05	0.04	0.08
	8192	0.01	0.03	0.03	0.03	0.03	0.05
72	128	0.06	0.13	0.11	0.13	0.11	0.23
	512	0.03	0.09	0.08	0.09	0.08	0.19
	2048	0.02	0.05	0.05	0.05	0.05	0.10
	8192	0.01	0.03	0.02	0.03	0.02	0.06

TABLE 4: Attack traffic statistics for the evaluation results of Fig. 5. Only results for $W_{test} = 24$ are shown, as the test length only has minimal impact on the results. The rows contain the data for **S**ality, **M**irai-like, and open **R**esolver. *Normal* counts each physical location identically, while in the *weighted* columns each location is weighted by the benign traffic levels. "Mdn" denotes the median.

		Normal			Weighted		
	LPF	Min	Avg	Mdn	Avg	Mdn	Max
S	128	345	593	570	593	570	809
	512	1380	2350	2282	2350	2282	3225
	2048	5522	9378	9128	9378	9128	12 883
	8192	22 087	37 495	36 511	37 495	36 511	51 519
M	128	818	1958	1482	1958	1482	8394
	512	3270	5848	5778	5848	5778	12 535
	2048	13 082	21 401	22 964	21 401	22 964	29 100
	8192	52 323	83 603	91 704	83 603	91 704	109 446
R	128	733	1229	1187	1229	1187	1736
	512	2932	4717	4565	4717	4565	6518
	2048	11 728	18 645	17 965	18 645	17 965	25 643
	8192	46 913	74 285	71 362	74 285	71 362	102 133

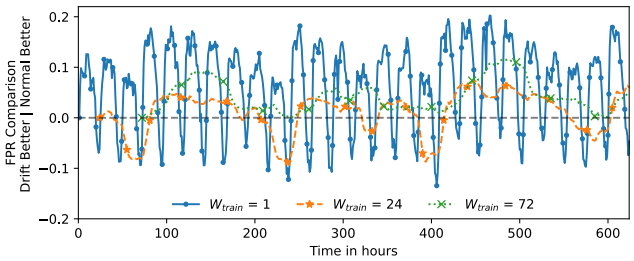


Figure 6: Impact of static (non-retrained) allowlists on the FPR based on three training lengths (one hour, one day, three days). Higher means worse.

Resolver churn leads to "dead" entries on the allowlist, which no longer match a resolver. Likewise, new resolvers appear, which may get blocked if they exceed the *LPF* and thereby increase the FPR.

5.5. Concept Drift

In the previous section, we observed a negative impact of long time spans between training and testing. The main reason for this decline is *concept drift*, which occurs because the population of active resolvers changes by hour and day. If the allowlist is not adapted accordingly, previously unseen resolvers may cause false positives. We evaluate this churn by artificially creating longer time gaps between the allowlist creation and the attack. For this, we train a static allowlist per site on the first W_{train} intervals in our dataset and test it on all possible future W_{test} intervals. This experiment indicates how quickly the allowlist becomes outdated and how often retraining is necessary. As the amount of unfiltered attack traffic is generally lower with the fixed allowlist, we omit an analysis of traffic changes and focus solely on the FPR.

Figure 6 shows the performance impact of using a static allowlist—compared to the standard procedure of constantly retraining the allowlist (as described in Section 5.4.2). The graph shows the evaluation performed for a single exemplary large AuthNS location for three training lengths (1 hour, 1 day, 3 days) and a fixed W_{test} = 24 hours interval. Positive values indicate that the fixed allowlist had a higher FPR. The shorter the training interval, the higher the maximum negative impact on the FPR. In fact, using short W_{train} results in a daily repeating pattern, where the static allowlist at times performs *better*. This is a result of diurnal patterns in the resolver population. Every 24 hours, the fixed allowlist matches exactly from the time of day it was created to the test interval, while the normal allowlist is shifted by at least an hour. However, this advantage is eaten up by large FPR increases at other times. Still, it shows a potential improvement in the allowlist creation to compensate for diurnal and weekly patterns. Instead of only training on the preceding x hours, we could additionally train on the previous day and week.

Longer training times result in patterns that fluctuate less. But we also see larger dips for W_{train} = 24, resulting from weekday-weekend shifts. The first training window starts on a Friday but captures part of the weekend too. This has positive effects on those weekend parts. Only the 3-day training interval captures the behavior of a full workday *and* the weekend, thus compensating for the resolver population cycle. But this long training period cannot account for longer shifts over weeks or months.

Independent of the training length, a static allowlist always causes significantly more FPs than periodically trained allowlists. Based on these results, we conclude that periodic retraining is highly advisable.

5.6. Evasion

Any learning mechanism has to content with smart attackers trying to evade the defense mechanisms. In our case, we see two evasion possibilities.

First, attackers could try to have their attack sources be allowlisted. For example, consider an attacker that adds attacker-controlled IP networks to the allowlist. This corresponds to a persistent attacker capable of placing their nodes undetected into the seemingly benign resolver population. For argument's sake, we simulate that each of the attacker networks is assigned a historic traffic level T_i equal to the highest benign value in the allowlist. Figure 7 shows the result of the evaluation with the attacker being able to transmit up to $100\,\mathrm{Tibit/s}$. The graph plots the number of IP networks that successfully evaded the allowlist (x-axis) and the amount of attack traffic the AuthNSes must process (y-axis). Each line represents a different site. We showcase a single configuration for our algorithm, the best-performing combination from Fig. 5 ($LPF = 2048$, $W_{test} = 24$). Overall, we see the trend that such evasion indeed leads to higher attack traffic. This is expected, as the bandwidth granted by the allowlist is significantly higher than the selected *LPF* value. The increase per location heavily depends on the source of the allowlisted attack hosts. In general, though, one can argue that thousands of poisoned allowlist entries do not hurt. Even $30\,000$ allowlisted attack sources seem uncritical; in the worst case, this causes a few $100\,000$s of attack requests per location. Having said this, this analysis indicates that care must be taken not to allowlist too many attack nodes. Indeed, the attack traffic increases roughly linearly with the number of poisoned allowlist entries.

Second, attackers may try to influence the *LPF* and *TOL* parameters, such that IPs not on the allowlist can send much traffic. Having said this, we observed that variations to the *LPF* and *TOL* parameters lead to similar performance. Thus, retraining these parameters is usually unnecessary, such that attackers cannot influence them. Furthermore, attackers cannot significantly influence the parameters without the AuthNS operator noticing, as the changes stand out compared to previous models.

5.7. Insider Threats

Insider threats against our defense are possible. We define three levels of access, each of which results in different threats: 1) knowledge about the algorithm parameters, 2) knowledge of the allowlists per site, and 3) modifications to the parameters or allowlist.

Knowledge of the algorithm parameters: Knowing the exact parameters enables an attacker to inject IP addresses into the allowlist during training more efficiently than without such knowledge. For each host to inject, a traffic profile using minimal theoretical bandwidth is selected. For example, each host only needs to be active

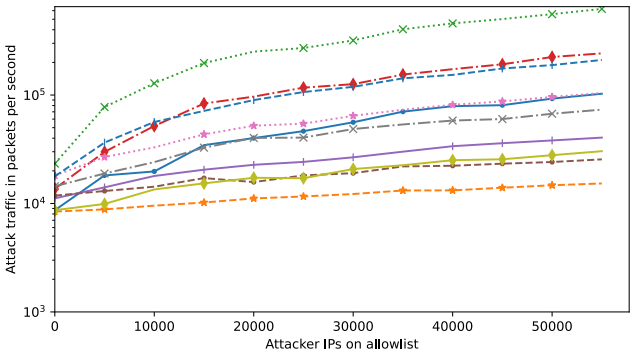


Figure 7: Impact of randomly selected attacker nodes added to the allowlist (x-axis) on the traffic in pps on a log scale (y-axis). Each line represents a different site.

for *STEADY* intervals per training window. An insider with parameter knowledge can thus save bandwidth by not always using the full capabilities of an attack source.

Knowledge of the allowlist: An attacker informed about the allowlist could aim to DoS a listed resolver by sending traffic from the resolver's IP range(s). This exhausts the resolver's traffic budget with mostly malicious traffic, drowning out the resolver's benign queries. It requires the attacker to have IP addresses in the same range as the victim[4], which is possible for resolvers hosted by cloud and shared hosting provides, but not for organizations with their own IP space. The feasibility depends on the aggregation level of the IP space, with stronger aggregation (e.g., /16 for IPv4) making it easier.

Direct allowlist or parameter modifications: Finally, we consider an unlikely yet strong attacker that finds a way to directly modify the allowlist of algorithm parameters. Such a strong insider attack can render the defense useless by picking rogue parameter values, enhancing the allowlists, or DoS resolvers by emptying it. Having said this, using standard access control mechanisms, only trusted parties—the AuthNS operator or its upstream providers—have such privileges. In addition, if the insider can only manipulate data at a single anycast site, resolvers could switch to other (non-compromised) sites.

5.8. Prototype and Performance

We implemented a prototype of our algorithm to test its feasibility. The prototype is written in Python and uses nfdump [38] for parsing the NetFlow data. The prototype is available with our other source code, see Section 8.

We ran the prototype on different NetFlow data and measured its runtime performance. Based on three sites, picked from small to large, we show how the performance scales with data size and the number of input files. Each NetFlow file contains the data for one hour. Table 5 shows the runtime for different training windows (W_{train}) and data sizes. The runtime scales well with the data size and the number of input files as long as there are enough CPU cores available. A problem arises when the individual NetFlow files grow too large, as they cannot be processed in parallel anymore. This is the main reason why the large site runs $7.7\times$ to $9.4\times$ slower than the small site.

4. Recall that our threat model assumes that AuthNS operators can enforce TCP to render spoofing attempts useless.

TABLE 5: Performance of our prototype for three sites, ranging from small to large data volume.

Value	Site Size	$W_{train} = 1$	3	6	12	24
Data in MB	small	4.4	13.3	25.0	41.9	89.1
	medium	9.5	26.6	50.4	115.5	273.2
	large	31.4	90.4	170.6	379.2	811.9
Mean Time in s	small	6.6	7.0	7.4	8.3	10.8
	medium	15.8	16.4	17.9	24.2	41.3
	large	51.5	53.9	58.1	76.4	102.1

TABLE 6: Comparison of different anti-DDoS solutions. A ✓indicates a good value for the criteria, and parentheses show that it might not apply in all situations.

	Our method	Reverse Proxy	Managed DNS
App.-Layer Flood	✓	(✗)	(✓)
Vol. Attack	✗	(✓)	(✓)
Privacy	✓	(✗)	(✗)
Latency	✓	↕	↕

5.9. Comparison with anti-DDoS Vendors

Our proposed system allows the AuthNS operator to defend against the attack itself (possibly with the help of upstreams). Commercial anti-DDoS services also aim to protect AuthNSes against DDoS attacks. Their general mode of operations falls into two categories: *reverse proxy* and *managed DNS*. We compare these third-party services to our method in the following, regarding their effectiveness and privacy—as summarized in Table 6.

Reverse Proxy: The anti-DDoS vendor can act as a *reverse proxy*, accepting traffic and forwarding valid DNS queries to the AuthNS operator. Technically, caching reverse proxies sit between the client (resolver) and AuthNS such that clients no longer query the AuthNS themselves. In principle, reverse proxies thereby significantly reduce the AuthNSes's load in that they only forward uncached queries. Assuming that the proxies have better network connectivity than the AuthNS operators, they also provide decent protection against volumetric attacks [29], [30], [42]—only valid DNS requests are forwarded to the AuthNS operator. Having said this, reverse proxies only provide limited defense against application-layer attacks that bust the cache. Given tens of millions of DNS records to choose from, plus non-cacheable queries to non-existing domains, attackers can evade the caches and flood the AuthNS infrastructure. There is a clear privacy downside, as a third party learns about the DNS queries. Finally, reverse proxies may not be able to retain the latency of the original AuthNS setup, especially if they are further away from the clients or still have to forward queries to the AuthNS. While there are opposite cases in which clients get cached responses faster, TLD operators that want to retain SLAs have to choose reverse proxy locations wisely.

Managed DNS: In the *managed DNS* mode, the anti-DDoS vendor directly answers any query with their copy of the AuthNS's zone file. The anti-DDoS vendor thus operates like an AuthNS operator. The big difference is that an anti-DDoS vendor operates a shared infrastructure for many customers. This results in a different financial model and likely larger infrastructure. If so, the managed DNS provider may even be able to protect against volumetric attacks. For application-layer attacks, the vendor will ul-

timately be similarly vulnerable as the AuthNS operator is for larger attacks that exceed its over-provisioning—unless protected by our defense. The privacy impact is similar to the reverse proxy, as a third party learns all DNS queries; even worse, it learns the zone file that may be deemed sensitive. The latency may change and depends on the anycast setup of the managed DNS operator. Again, to retain latency-based SLA guarantees, the anycast setup would have to be similar (or superior) to the original one.

6. Discussion

In this section, we first check whether our proposed defense aligns with the design goals. We then discuss the relevance of our work in light of other defenses. Finally, we iterate threats of validity in our evaluation.

6.1. Reflection on Design Goals

We first discuss how our proposed defense meets the design goals as outlined in Section 4.1. By design, our proposed content-agnostic (G4) anomaly detection can be applied by upstream providers (G3) and does not require client support (G2). The low-pass filter retains compatibility with low-volume resolvers (G8). Our proposed anomaly detection defense is easy to use (G7); it just requires NetFlow data (or similar statistics) to learn statistical profiles of past resolver behavior. There are no extra costs (G7); all standard routing/switching hardware nowadays offers flow monitoring, and storage costs for up to 72 hours of NetFlow data are negligible. Despite this simplicity, the method filters sufficient attack traffic to avoid packet loss at AuthNSes (G1). To further underline this, the *LPF* threshold can even be lowered to reduce the load in case of overloads. This even holds if attackers can poison the allowlist with thousands of entries, showing a high degree of resilience against the major evasion angle (G5). At the same time, there are few false positives (G6), especially when compared to the alternatives defenses (e.g., null routing) or no defense at all (high packet loss $\rightarrow$ high FPR). To conclude, the proposed methodology fully fulfills the design goals and thereby represents a vital last-resort defense for TLD operators.

6.2. Limitations

Our proposed methodology still has a few limitations, as discussed in the following.

Catchment Changes: Some mitigations, like changes in the BGP announcements, will influence the catchment area of the sites. This affects how well the computed allowlist matches and might lead to higher false positives. Computing a single shared allowlist across all sites alleviates that, as it guarantees that each resolver is allowlisted everywhere, but it is less specific.

Millions of attack sources: Very large botnets are problematic, as they can overwhelm the AuthNS with their sheer volume, as each network is allowed to send *LPF* amount of traffic. This can add up to significant amounts of traffic. It can be partially counteracted by decreasing the value for *LPF*, but this increases the number of false positives. The reduction of *LPF* will affect the benign resolvers equally, thus changes have to be made carefully.

Spoofing-capable attackers: IP address spoofing allows the attacker to mimic arbitrary sources. This not only allows mimicking millions of attack sources. It also offers the chance to DoS a specific set of victim resolvers. When our defense is enabled, attackers can spoof victim IP addresses to consume their traffic allowance. Detecting spoofing is simple when many IP addresses are affected but can be challenging when only a few are. In either case, the AuthNS operator can enable spoofing countermeasures (see Section 3.1), which effectively prevents these attacks.

Coarse-grained filters: Network aggregation may merge traffic from both benign and malicious sources into a single mixed entity. The attribution of network traffic is still correct. But both sources now compete for the same traffic allowance. If attackers send orders of magnitude more traffic, this prevents the benign resolver from accessing any AuthNS. This requires the attacker to have an IP address close to the victim, which is possible in the cloud or shared hosting environments, but impossible if the resolver has a dedicated IP address. The amount of aggregation is another contributing factor and can be reduced, i.e., longer prefixes, when this problem occurs. Longer prefixes result in more detailed attribution but increase the size of the allowlist and computation cost.

6.3. Relevance of Our Work

So far, we assumed that we have to use upstream filters to avoid overloaded AuthNSes (i.e., packet loss) at all costs. We see our proposed defense as the last level of defense if all other DDoS countermeasures fail. Again, for volumetric DDoS attacks, upstream content filters work well. Also, small application-layer attacks that overload a single anycast location can only be mitigated by traffic rerouting, as suggested by Rizvi et al. [41].

However, when **DDoS attacks overload the entire anycast infrastructure**, the countermeasures mentioned before are of no help. Indeed, this is a situation that will likely occur in our modeled attacks (Section 5.3). If a standard botnet attacks a large TLD, the total attack bandwidth (or query volume) exceeds the resources of the entire anycast infrastructure by far. This underlines that we indeed need additional countermeasures like ours that are agnostic to the content and filter attack traffic upstream.

We acknowledge that, for ethical reasons, we have not tested the consequences of successful DoS attacks against TLDs. We consider an attack successful if the overall traffic exceeds typical query capacities of AuthNSes. Especially short-term attacks may not have as severe consequences, as caching and even stale records (Section 3.2) provide short-term mitigation. In general, it is hard to model the clients' tolerance to resolution failures. Given many layers of caching and uncertainty, determining the final impact of DDoS attacks is difficult. Having said this, past attacks have illustrated that there *are* negative attack implications. First, Moura et al. [36] report on two DDoS attacks against the DNS root servers between 2015-11-30 and 2015-12-01. The DNS root servers are separated into multiple anycast clouds, containing a couple to hundreds of servers each. Multiple anycast clouds failed under a load of up to $5\,\mathrm{Mpps}$. End-user errors were reported, even though multiple anycast clouds were still

operational. Second, Sommese et al. [45] report on further DDoS activity against various DNS servers between 2020-11 and 2022-03. They achieve greater visibility by combining DoS activity from a large darknet with active DNS measurements of the OpenINTEL platform, giving them information about resolution times and failures. Most successful attacks are against smaller DNS providers, often without anycast deployments. Yet, larger providers with anycast infrastructures, such as Hetzner, Apple, or GoDaddy, are also affected. From these reports, we learn that successful DDoS attacks against DNS infrastructure happen and represent an open research problem. We hope that our work can help to mitigate future attacks.

6.4. Threats to Validity

In our evaluation, we tried to get as close as possible to a realistic attack by using real-world background traffic of a TLD and monitoring potential attack sources. While we believe these results are representative, there are a few factors that may influence the preciseness of the results. We iterate potential threats to validity in the following.

Requests per lookup: We modeled that each client lookup towards an open resolver creates a single query towards the AuthNS. In practice, resolvers may send multiple queries even during normal operations. For example, resolvers often issue the same query to multiple AuthNSes, e.g., to both an IPv4 and IPv6 server. During a successful DDoS attack, resolvers may even *amplify* the attack, as they do not receive responses to their queries. The default resolver behavior then is to issue the same query again [37]. Such repeated queries are an extra load the attacked server needs to handle. After multiple unsuccessful queries, the resolver might deem the AuthNS unresponsive and switch to another IP address or AuthNS. This, in turn, alleviates the load on the attacked server but may cause cascading failures at other AuthNSes. All these effects—the number of re-transmissions and how resolvers switch between name servers—are impossible to determine without performing an actual attack, as it depends on the software, configuration, and the specific connection to all AuthNSes. Therefore, we excluded this effect from our analysis. However, this underlines once more that we have to shield AuthNS infrastructures early on, not to run into these amplification issues.

Anycast cloud sizes: In this paper, we only looked at medium-sized anycast clouds. Other anycast deployments range from two locations to hundreds of locations for a single cloud, which influences DDoS resilience. Smaller clouds are more vulnerable, as they have less redundancy and fewer overall resources for traffic processing. They also have fewer catchment areas, allowing an attacker to more efficiently target a specific location or deploy attacking nodes in all catchment areas. Clouds with hundreds of locations are more capable of swallowing an attack and attract more benign traffic.

Another observation is that not all locations within an anycast setup receive the same traffic share. Resolvers prefer locations with low latency. Anycast locations with beneficial routing (e.g., close to exchange points or backbone networks) will attract traffic from many networks. Other locations will be located further from the Tier-1 providers, and thus their route announcements will have limited reach. Given the larger catchment, this imbalance makes some locations easier to attack. In contrast, other locations are "protected" since the attacker will struggle to assemble enough attack bandwidth in their catchment areas. Furthermore, such highly-imbalanced anycast setups have the potential for a cascading failure. If a high-volume location is overwhelmed and traffic switches to a relatively smaller location, that location will be overloaded.

Having said this, the mid-sized anycast clouds we studied provide a good balance and are representative of anycast setups of several other large ccTLD operators. Furthermore, the defenses we studied work independently of the concrete anycast setup.

Attack sources: Our evaluation consistently models the attacker using *all* available traffic sources. Given our filtering algorithm, this is the most beneficial choice for the attacker, as any IP network can contribute attack traffic up to the *LPF*. In reality, the attacker might not use all available traffic sources, e.g., as nodes are in the wrong catchment area or nodes provide less bandwidth than other sources. This is not a concern for our proposed filtering, as any deviation benefits the defenders, and thus we report worst-case results in the paper.

Alternatively, to using subsets of attack sources, attackers may find *larger* attacking sets than the ones we studied. We monitored reasonably large botnets that contain up to $\approx$ 100k attack networks. Larger botnets existed in the past (e.g., Conficker), and would demand stricter low-pass filters to protect DNS infrastructures.

Preconfigured resolvers: ISPs operate DNS resolvers which are only accessible to their customers. A botnet could leverage these resolvers if they have compromised hosts in the ISP's network. These preconfigured resolvers are shared with many benign users, more so maybe than many open resolvers. From the AuthNS perspective, misuse of the preconfigured resolvers is harder to detect and mitigate since benign and attack traffic is blended together. Consequently, we risk blocking legitimate traffic. Having said this, ISPs can rate limit their clients to prevent massive abuse and mitigate the resulting damage to their resolvers' reputations. We cannot measure the population of ISP-provided resolvers due to a lack of access to these networks. Therefore, we do not report on the effects of abusing such resolvers in attacks.

7. Related Work

Work about DNS-based DDoS attacks is split mainly into two broad categories: 1) classifiers and strategies to detect or evade attacks and 2) describing novel attacks.

7.1. DDoS Defenses

Moura et al. [37] investigate how resolvers and clients behave during attacks on AuthNSes. They set up an AuthNS infrastructure and drop packets to simulate an attack. They measure the success rate, latency, and more of clients querying the "attacked" zone. The paper explains how an AuthNS DoS will affect the larger ecosystem and highlights the importance of defenses for AuthNSes. We thus iterate over DNS-specific defenses in the following.

Davis et al. [13] propose a new mitigation strategy based on reverse DNS entries. An IP address space owner

can mark what kind of DNS traffic is expected to originate from the address space, e.g., specifying DNS Cookie support or noting that no DNS traffic is expected. AuthNSes can use this information to filter out traffic originating from a spoofed source and not respond. A high adoption rate in both the reverse DNS space and among all DNS servers could reduce reflective DNS attacks. This comes with a costly downside for AuthNSes since now they must perform DNS queries, something usually only performed by clients and resolvers. This adds network, processing, and memory overheads for little protection of the AuthNS itself, but rather for the rest of the internet.

Alonso et al. [5] suggest a more passive methodology to detect simple DNS floods. They leverage the "social structure" of resolvers, i.e., the relationship between resolvers and the queried domains. Their basic idea is to search for query patterns that increase or change the set of domains being queried, which happens, e.g., for random subdomain attacks. Unfortunately, their work is thus not content-agnostic, leaving leeway for trivial evasion techniques for attacks against TLDs. For example, in our setting, attackers can choose from querying millions of *existing* domains and could even mimic website popularity rankings to model query likelihoods for each domain. Moreover, their methodology requires a costly query analysis and subgraph detection (approximating the NP-complete problem of bipartite subgraph search).

Rizvi et al. [41] propose an active routing-based defense for attacks that overload just single AuthNS in a larger anycast setup. To mitigate these local attacks, they use BGP to steer traffic between multiple anycast servers. Anycast catchment changes as a result of altered BGP announcements are often unpredictable. As such, the paper describes the creation of a playbook of applicable BGP alterations and the likely outcome. During an attack, the alteration which best matches the current situation can be picked, bringing all servers back to capacity. This defense is elegant and free of false positives, yet can only be applied if the overall anycast network has sufficient capacities to defend against an attack. Our work applies to the exact opposite situation, i.e., when massive attacks overload the entire anycast network. In these situations, shifting traffic from one location to another would only trigger cascading collateral damage.

Finally, Trejo et al. [51] create DNS-ADVP, a nearest-neighbor-based system to detect the presence of amplification attacks against DNS infrastructures. First, the authors create visual classifiers to inspect and alert on abnormal traffic conditions. The visual classifiers are based on the traffic fraction of the largest source compared to the overall volume and the correlation of requested domain names between different resolvers. They then represent the DNS traffic as feature vectors, each vector capturing the overall traffic statistics of a second time window. The resulting system can, with $78\,\%$ accuracy, detect the presence of attacks against DNS. Having said this, their work has several limitations. First, the proposed defense is limited to detecting *that* attacks occur but cannot pinpoint the attack sources. It can complement our work in that it can be used to *activate* our defense. Second, the features are not content-agnostic, risking that targeted attacks bypass the methodology by adapting their query patterns.

7.2. DNS Attacks

Besides the ongoing efforts on exploring attack countermeasures, other related work documents new DNS attack strategies. While not directly related to our paper, the strategies listed below describe attacks against an AuthNS, for which our anomaly detection applies.

The tsuNAME attack disclosed by Moura et al. [35] describes a software bug, where a pair of dependent records can lead to an infinite loop. Two zones in different TLDs are set up to host their AuthNSes in the other zone. This causes a cyclic dependency where, looking up the AuthNS of the first zone, the second zone needs to be resolved, but that requires finding the AuthNS in the first zone, leading to a situation where neither zone can be resolved successfully. A bug in the software implementation Google used did not detect this situation and instead kept sending requests to both domains. The requests were not rate limited, causing flooding of the TLD AuthNSes. While this specific attack is caused by a combination of bad zone files, i.e., cyclic dependencies, and a software bug, it shows that even benign services can sometimes misbehave. Any defense strategy should be careful in generally allowlisting specific sources and, as we propose, better apply limits even to allowlisted sources.

The NXNSAttack discovered by Afek et al. [2] used application-layer amplification enabled by resolvers to flood AuthNSes with over $500\times$ the original bandwidth. The attack is enabled by resolvers, which are too generous in following DNS delegations, resulting in multiple identical queries. A DNS delegation is performed with a `NS` record, which points to a domain where the `A` and `AAAA` records for the AuthNS IP addresses are stored. Each delegation can have multiple `NS` records, and each domain has multiple `A`/`AAAA` records. All `A`/`AAAA` records can point to the same server. Resolvers were using all records simultaneously and sending hundreds of queries. Limiting the number of simultaneously followed delegations fixes the attack. This attack shows again how benign resolvers can suddenly turn bad. Our defense can cope with this attack; we would penalize "unpatched" resolvers by dropping their (partially benign) lookups. As soon as they mitigate their bogus behavior, they fall below the *LPF* or their prior allowlist threshold.

8. Conclusion

For many years, the DNS operators community has tacitly assumed that our core DNS infrastructure is sufficiently resilient against large-scale DDoS attacks. We showed that this might be a false assumption in the presence of strong adversaries. Motivated by this observation, we proposed a two-layer anomaly detection defense allowing upstream providers to effectively mitigate application-layer attacks against DNS. The proposed defense finds a sweet spot between collateral damage (false positives) and the remaining load on the critical DNS infrastructures (measured in requests per second). This provides a big step forward for DNS operators, especially if traffic engineering is no longer effective. Should an application-layer attack arise for which content-based filters fail, DNS operators now have a better alternative to extreme(ly bad) measures such as null routing.

Data Availability

Given the sensitivity of the DNS name lookups, the ccTLD asked for an NDA prior to our cooperation. We nevertheless thought about the possibility of releasing the dataset used throughout this work. In particular, we operate on traffic statistics and aggregated networks only Section 5.1—a seemingly curated dataset. However, according to national data protection law, sharing this data is not possible, as it would violate the well-defined reasons under which this data may be collected. In particular, the (sole) reason why the cooperating ccTLD operator may record the data in the first place is the possibility of protecting the corresponding DNS infrastructures. This reason no longer applies when data is shared for other purposes.

To ease reproducibility, we have released the code that we used to (i) ingest NetFlow data into a database, (ii) build the respective allowlists, and (iii) evaluate the methodology based on given datasets. We have a prototype implementation for the allowlist creation, which converts a NetFlow file into an allowlist. The data sets used to model the Mirai, Sality and Open Resolver populates are included.

The code and data are available online at https://github.com/cispa/DNS-Applayer-DDoS-Protection/.

Acknowledgments

We sincerely thank our partnering ccTLD operator, without whom this research would not have been possible. Our sincere thanks further belongs to the anonymous reviewers for their valuable feedback and suggestions which helped to improve the paper. Furthermore, we thank the Saarbrücken Graduate School of Computer Science for their funding and support.

References

[1] Donald E. Eastlake 3rd and Mark P. Andrews. Domain Name System (DNS) Cookies. RFC 7873, May 2016.

[2] Yehuda Afek, Anat Bremler-Barr, and Lior Shafir. NXNSAttack: recursive DNS inefficiencies and vulnerabilities. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 631–648. USENIX Association, August 2020.

[3] afnic. Statistiques. https://www.afnic.fr/observatoire-ressources/statistiques/, January 2022.

[4] Paul Aitken, Benoît Claise, and Brian Trammell. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011, September 2013.

[5] Roberto Alonso, Raúl Monroy, and Luis A. Trejo. Mining IP to domain name interactions to detect DNS flood attacks on recursive DNS servers. *Sensors*, 2016.

[6] BIND open source DNS server. https://www.isc.org/bind/.

[7] Jonas Bushart and Christian Rossow. DNS unchained: Amplified application-layer DoS attacks against DNS authoritatives. In *Research in Attacks, Intrusions, and Defenses - 21st International Symposium*, 2018.

[8] Benoît Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954, October 2004.

[9] Benoît Claise and Brian Trammell. Information Model for IP Flow Information Export (IPFIX). RFC 7012, September 2013.

[10] Secure64 Software Corporation. Water torture: A slow drip DNS DDoS attack, February 2014. https://secure64.com/water-torture-slow-drip-dns-ddos-attack/.

[11] Traffic dashboard. https://stats.nic.cz/dashboard/en/Traffic.html, February 2023.

[12] Amir Dahan. Business as usual for Azure customers despite 2.4 Tbps DDoS attack. https://azure.microsoft.com/en-us/blog/business-as-usual-for-azure-customers-despite-24-tbps-ddos-attack/, October 2021.

[13] Jacob Davis and Casey T. Deccio. Advertising DNS protocol use to mitigate DDoS attacks. In *29th IEEE International Conference on Network Protocols*, 2021.

[14] denic. Domain statistics of .de. https://www.denic.de/en/know-how/statistics/monthly-statistics-of-de/, January 2022.

[15] John Dickinson, Sara Dickinson, Ray Bellis, Allison Mankin, and Duane Wessels. DNS Transport over TCP - Implementation Requirements. RFC 7766, March 2016.

[16] DNS Privacy Project. Follow-up performance measurements (q4 2108). https://dnsprivacy.org/performance_measurements/follow-up_performance_measurements_q4_2108/, 10 2018.

[17] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. ZMap: fast internet-wide scanning and its security applications. In *Proceedings of the 22th USENIX Security Symposium*, 2013.

[18] Kazunori Fujiwara, Akira Kato, and Warren "Ace" Kumari. Aggressive Use of DNSSEC-Validated Cache. RFC 8198, July 2017.

[19] R. (Miek) Gieben and Matthijs Mekking. Authenticated Denial of Existence in the DNS. RFC 7129, February 2014.

[20] Suzanne Goldlust. Using the response rate limiting feature. https://kb.isc.org/docs/aa-00994, September 2018.

[21] Suzanne Goldlust and Cathy Almond. Recursive client rate limiting. https://kb.isc.org/docs/aa-01304, October 2021.

[22] Suzanne Goldlust and Cathy Almond. Recursive client rate limiting – FAQs. https://kb.isc.org/docs/aa-01316, October 2021.

[23] Scott Hilton. Dyn analysis summary of friday october 21 attack. https://web.archive.org/web/20180224030354/https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/, October 2016.

[24] Knot DNS. Benchmark. https://www.knot-dns.cz/benchmark/, January 2022.

[25] Knot DNS. Old benchmark. https://www.knot-dns.cz/benchmark-old/, January 2022.

[26] Knot Resolver. https://www.knot-resolver.cz/.

[27] Lukas Krämer, Johannes Krupp, Daisuke Makita, Tomomi Nishizoe, Takashi Koide, Katsunari Yoshioka, and Christian Rossow. AmpPot: monitoring and defending against amplification DDoS attacks. In *Research in Attacks, Intrusions, and Defenses - 18th International Symposium*, 2015.

[28] Christian Kreibich, Andrew Warfield, Jon Crowcroft, Steven Hand, and Ian Pratt. Using packet symmetry to curtail malicious traffic. *ACM Hotnets-IV*, 200, 2005.

[29] Johannes Krupp, Michael Backes, and Christian Rossow. Identifying the scanners and attack infrastructure behind amplification DDoS attacks. In *Proceedings of the 2016 ACM Conference on Computer and Communications Security*. ACM, 2016.

[30] Marc Kührer, Thomas Hupperich, Christian Rossow, and Thorsten Holz. Exit from hell? reducing the impact of amplification DDoS attacks. In *Proceedings of the 23rd USENIX Security Symposium*, 2014.

[31] David C Lawrence, Warren "Ace" Kumari, and Puneet Sood. Serving Stale Data to Improve DNS Resiliency. RFC 8767, March 2020.

[32] Christoph Loibl, Susan Hares, Robert Raszuk, Danny R. McPherson, and Martin Bacher. Dissemination of Flow Specification Rules. RFC 8955, December 2020.

[33] Christoph Loibl, Robert Raszuk, and Susan Hares. Dissemination of Flow Specification Rules for IPv6. RFC 8956, December 2020.

[34] Measurement Lab. The M-Lab NDT data set. https://measurementlab.net/tests/ndt, (2021-01-01 – 2021-01-31).

[35] Giovane C. M. Moura, Sebastian Castro, John Heidemann, and Wes Hardaker. tsuNAME: public disclosure and security advisory. Technical report, SIDN Labs, InternetNZ and USC/ISI, May 2021.

[36] Giovane C. M. Moura, Ricardo de Oliveira Schmidt, John S. Heidemann, Wouter B. de Vries, Moritz Müller, Lan Wei, and Cristian Hesselman. Anycast vs. DDoS: Evaluating the november 2015 root DNS event. In *Proceedings of the 2016 ACM on Internet Measurement Conference*, 2016.

[37] Giovane C. M. Moura, John S. Heidemann, Moritz Müller, Ricardo de Oliveira Schmidt, and Marco Davids. When the dike breaks: Dissecting DNS defenses during DDoS. In *Proceedings of the Internet Measurement Conference 2018*, 2018.

[38] nfdump. https://github.com/phaag/nfdump/, February 2023.

[39] Nominet. .uk register statistics – 2020. https://www.nominet.uk/news/reports-statistics/uk-register-statistics-2020/, January 2022.

[40] PowerDNS Recursor. https://www.powerdns.com/recursor.html.

[41] A. S. M. Rizvi, Leandro M. Bertholdo, João M. Ceron, and John S. Heidemann. Anycast agility: Network playbooks to fight DDoS. In *31st USENIX Security Symposium*, 2022.

[42] Christian Rossow. Amplification hell: Revisiting network protocols for DDoS abuse. In *21st Annual Network and Distributed System Security Symposium*, 2014.

[43] SIDN LABS. .nl statistieken. https://stats.sidnlabs.nl/nl/registration.html, January 2022.

[44] Raffaele Sommese, Gautam Akiwate, Mattijs Jonker, Giovane C. M. Moura, Marco Davids, Roland van Rijswijk-Deij, Geoffrey M. Voelker, Stefan Savage, Kimberly C. Claffy, and Anna Sperotto. Characterization of anycast adoption in the DNS authoritative infrastructure. In *5th Network Traffic Measurement and Analysis Conference*, 2021.

[45] Raffaele Sommese, kc claffy, Roland van Rijswijk-Deij, Arnab Chattopadhyay, Alberto Dainotti, Anna Sperotto, and Mattijs Jonker. Investigating the impact of DDoS attacks on DNS infrastructure. In *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022.

[46] Most popular top-level domains worldwide as of june 2022. https://www.statista.com/statistics/265677/number-of-internet-top-level-domains-worldwide/, June 2022.

[47] Ondřej Surý, Willem Toorop, Donald E. Eastlake 3rd, and Mark P. Andrews. Interoperable Domain Name System (DNS) Server Cookies. RFC 9018, April 2021.

[48] DNS statistics. https://www.nic.ch/statistics/dns/, February 2023.

[49] Yuya Takeuchi, Takuro Yoshida, Ryotaro Kobayashi, Masahiko Kato, and Hiroyuki Kishimoto. Detection of the DNS water torture attack by analyzing features of the subdomain name. *J. Inf. Process.*, 2016.

[50] DNS Implementations. https://dnsinstitute.com/implementations/.

[51] Luis A. Trejo, Victor Ferman, Miguel Angel Medina-Pérez, Fernando Miguel Arredondo Giacinti, Raúl Monroy, and Jose Emmanuel Ramirez-Marquez. DNS-ADVP: A machine learning anomaly detection and visual platform to protect top-level domain name servers against DDoS attacks. *IEEE Access*, 2019.

[52] Unbound. https://nlnetlabs.nl/projects/unbound/about/.

[53] Olivier van der Toorn, Johannes Krupp, Mattijs Jonker, Roland van Rijswijk-Deij, Christian Rossow, and Anna Sperotto. ANYway: Measuring the amplification DDoS potential of domains. In *17th International Conference on Network and Service Management*, 2021.

[54] Bruce Van Nice. Drilling down into DNS DDoS, February 2015. https://www.nanog.org/sites/default/files/nanog63-dnstrack-vannice-ddos.pdf.

[55] Verisign. Zone files for top-level domains (TLDs). https://www.verisign.com/en_US/channel-resources/domain-registry-products/zone-file/index.xhtml, January 2022.

[56] Paul Vixie and Vernon Schryver. DNS response rate limiting (DNS RRL). https://ftp.isc.org/isc/pubs/tn/isc-tn-2012-1.txt, April 2012.

[57] Samuel Weiler and Johan Ihren. Minimally Covering NSEC Records and DNSSEC On-line Signing. RFC 4470, April 2006.

A. Anycast Setup

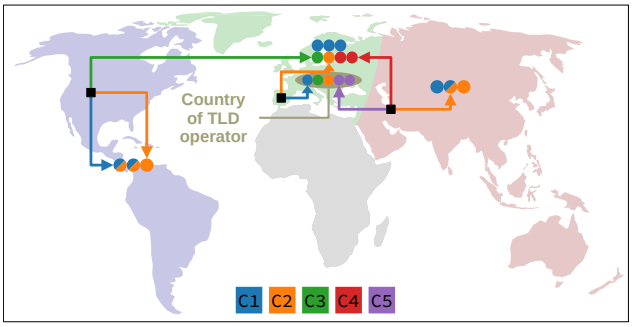


Figure 8: Approximate locations of AuthNS instances (circles colored by cloud), their anycast cloud assignment, and hypothetical DNS clients (■) using different anycast clouds (`C1`–`C5`). The country of the ccTLD's origin, shown as the oval in Europe, contains five locations, remaining Europe (pale green) further seven locations, while South/North America (pale blue) and Asia/Pacific (pale red) have three locations each. The exemplified clients (■) have arrows in the colors of the anycast clouds, indicating how BGP could route them.

B. Sality Results

The Sality results are identical to the presented results about Mirai-like botnets and open resolver. Figure 9 has an identical setup to Fig. 5 but contains the visualization for the Sality botnet.

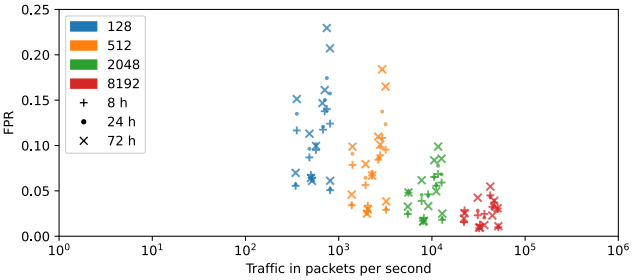


Figure 9: Effect of the *LPF* and W_{test} parameters on the FPR and FNs. Each point represents a site's performance depending on *LPF* and W_{test}. The graphs depict attackers using Sality, a Mirai-like botnet, and open resolvers.

C. Prototype Performance

This is an extended graph for the data in Table 5. It shows the runtime of our prototype based on the training window (W_{train}) and the NetFlow data size.

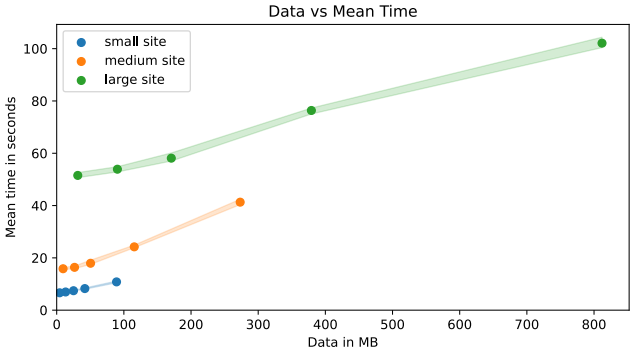


Figure 10: Performance of our prototype for three sites, ranging from small to large data volume. The training window (W_{train}) ranges from one (left) to 24 hours (right). The graph shows the mean runtime in seconds (y-axis) with the observed min/max values and data size (x-axis).

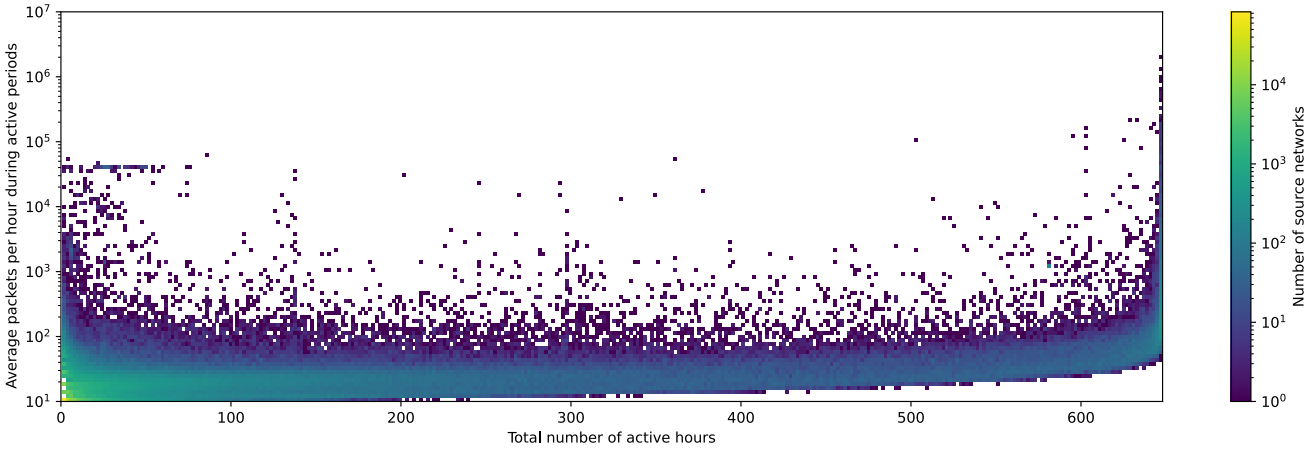


Figure 11: The histogram depicts the number of networks with packet rates and activity times. The x-axis shows in how many hours the IP network was observed, while the y-axis represents the average packet rate during the active period. The color shows how common each combination is, from non-existing white, over rare purple, to a common yellow.

D. DNS Client Query Behavior

The heatmap in Fig. 11 shows the most common combinations of average packet rate in packets per hour (y-axis in log scale) and the number of hours a network was active (x-axis; max. 682 hours). The color encodes how common these combinations are, ranging from purple (least number of networks) to yellow (high density of networks).

The graph shows the highest concentration towards the bottom-left corner. This shows that most networks in our dataset are only sporadically active and send packets at low rates. In fact, most IP networks send fewer than ten packets per hour, yet they contribute less than $7.5\,\%$ of the recorded traffic. The IP networks with the highest traffic rates are located on the right in the graph. This means that there is a stable set of IP networks, which are active during the whole observation period, sending one order of magnitude more traffic than other networks. These are likely resolver infrastructures of ISPs, cloud operators, and public DNS offerings since these are always active and attract large user bases. There are just a few sporadic high-volume clients, as indicated by the upper left part of the histogram. These source networks might become problematic for anomaly detection since they appear short-term but with significant traffic compared to other sources.