# Akamai DNS: Providing Authoritative Answers to the World's Queries

Kyle Schomp[†], Onkar Bhardwaj[†], Eymen Kurdoglu[†], Mashooq Muhaimen[†], Ramesh K. Sitaraman[†‡]

[†]Akamai Technologies
kschomp,obhardwa,ekurdogl,mmuhaime,ramesh@akamai.com
[‡]University of Massachusetts Amherst
ramesh@cs.umass.edu

## ABSTRACT

We present Akamai DNS, one of the largest authoritative DNS infrastructures in the world, that supports the Akamai content delivery network (CDN) as well as authoritative DNS hosting and DNS-based load balancing services for many enterprises. As the starting point for a significant fraction of the world's Internet interactions, Akamai DNS serves millions of queries each second and must be *resilient* to avoid disrupting myriad online services, *scalable* to meet the ever increasing volume of DNS queries, *performant* to prevent user-perceivable performance degradation, and *reconfigurable* to react quickly to shifts in network conditions and attacks. We outline the design principles and architecture used to achieve Akamai DNS's goals, relating the design choices to the system workload and quantifying the effectiveness of those designs. Further, we convey insights from operating the production system that are of value to the broader research community.

## CCS CONCEPTS

• **Networks → Application layer protocols**; **Naming and addressing**;

## KEYWORDS

DNS, Distributed Systems

**ACM Reference Format:**
Kyle Schomp, Onkar Bhardwaj, Eymen Kurdoglu, Mashooq Muhaimen, and Ramesh K. Sitaraman. 2020. Akamai DNS: Providing Authoritative Answers to the World's Queries. In *Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM '20), August 10–14, 2020, Virtual Event, NY, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3387514.3405881


*SIGCOMM '20, August 10–14, 2020, Virtual Event, NY, USA*

ACM ISBN 978-1-4503-7955-7/20/08…$15.00
https://doi.org/10.1145/3387514.3405881

## 1 INTRODUCTION

Naming is a central service of the Internet and is primarily addressed by the Domain Name System (DNS). Originally described in 1983 [31], DNS enables the mapping of human-legible hierarchical names to arbitrary records, most notably IP addresses. Thus, we refer to websites as "example.com" instead of "12.23.34.45". From its original design, DNS has expanded and grown in complexity [37, 46, 50] and continues to be an area of innovation today [13, 19, 21].

DNS consists of two types of systems that coordinate to provide domain name translations for end-users. The *client-side system* primarily consists of *recursive resolvers* that are charged with resolving queries from end-users. A request from an end-user for a domain name translation is first sent to its assigned resolver. If a valid translation is not found in the resolver's cache, the resolver obtains the answer by querying a system of *authoritative nameservers* for the requested name. The authoritative system stores the associations of domain names to records and provides definitive answers to queries.

The authoritative system is organized hierarchically in accordance with the name hierarchy. At the top, "root" nameservers are responsible for the empty label "." while one level down the "toplevel domain" nameservers are responsible for the labels under the root (e.g., "com"). Below that, organizations operate authoritative nameservers for their respective domains, e.g., "google.com" is served by Google's nameservers. To obtain an answer to a query, recursive resolvers iteratively search starting at the root and following delegations down the naming hierarchy, until reaching a nameserver that is responsible for the domain of the query and returns an answer. Nameservers include a Time-To-Live (TTL) field in answers, allowing the resolver to cache the answer for a prescribed amount of time, a feature that greatly improves performance and decreases DNS traffic.

We present Akamai DNS, one of the largest authoritative DNS infrastructures in the world, providing insights into its architecture, algorithms, design principles, and operation. We start by describing the services that it supports.

**Authoritative DNS Services:** Akamai DNS supports three authoritative DNS services. The first is an authoritative DNS hosting service (ADHS) that allows enterprises to host their DNS domains on Akamai. The second service is global traffic management (GTM) that allows DNS-based load-balancing among server deployments owned by an enterprise. Third, Akamai DNS is a component of Akamai's CDN service, serving 15-20% of all web traffic [36], and

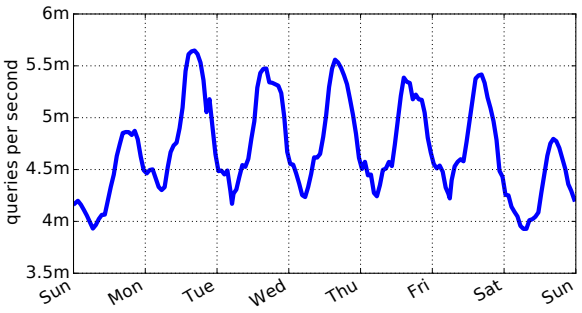


**Figure 1: Queries per second served by Akamai DNS.**

allows enterprises to outsource their entire content and application delivery infrastructure to Akamai. While these services impose differing requirements on the design of Akamai DNS, they can also be used together by a single enterprise, e.g., DNS hosting for their domains, GTM for their datacenters, and CDN services for edge delivery of their content fetched from those datacenters.

**Design Requirements:** Akamai DNS is the starting point for a significant fraction of the world's interactions with the Internet, whether it be end-users downloading web pages, watching videos, shopping online, downloading software, or accessing social networks. Given its critical role in the Internet ecosystem, the first and foremost requirement is 24/7 availability of the services that it supports. Because DNS translations preface the majority of Internet connections [45], even a minor disruption in Akamai DNS can cause a worldwide disruption in online services, severely impacting the conduct of commerce, business, and government around the globe. Yet, server and network failures are common in distributed systems. Also, due to the central role of DNS and high visibility when it fails, DNS has become a popular target of distributed denial of service (DDoS) attacks. Thus, Akamai DNS is architected to be *resilient* to both failures and attacks.

Since querying Akamai DNS forms the first step in an end-user's interaction with many online services, the answers must be provided quickly, so as not to increase the response times experienced by end-users. The system must also serve millions of queries per second (see Figure 1), with query volumes increasing (an 18% increase in the past year) in proportion to global Internet usage. Thus, Akamai DNS is architected for both *scalability* and *performance*.

Finally, the authoritative answers provided by Akamai DNS must adapt rapidly to changes in enterprise configurations, server liveness and load, and Internet conditions. For instance, to provide GTM and CDN services, Akamai DNS must always resolve an end-user's query to a *proximal* server that can deliver the content with low latency to the end-user [36]. When server or network conditions degrade, new DNS records are computed by Akamai's mapping system [11] and propagated to resolvers through Akamai DNS within seconds, so as to reroute end-user requests and prevent performance degradation. Unlike traditional authoritative DNS whose translations remain relatively static, Akamai DNS is architected for rapid *reconfigurability*.

**Our Contributions:** Our work is the first in-depth view of the architecture and capabilities of one of the world's largest authoritative DNS infrastructures that is a key part of the global Internet ecosystem. Specific contributions follow.

(1) We characterize how domain names are queried by resolvers around the world from the unique vantage point of Akamai DNS. We show that 3% of resolvers generate 80% of the DNS queries and that those same resolvers consistently send high volumes of DNS queries for periods of weeks to months.
(2) We outline the system architecture of Akamai DNS, including key features such as its wide-area deployment, its use of anycast to distribute DNS queries among locations, its software and server architecture within each location to provide resiliency, and its two-tier delegation system to provide rapid answers with low TTLs.
(3) We describe our anycast failover mechanism for resilience. We measure how long failover from one location to another takes when advertising or withdrawing routes via BGP. We show that in most scenarios failover is rapid – less than 1 sec in 76% of measurements.
(4) We present the system design elements that provide resiliency to network, hardware, and software failures and malicious DDoS attacks. We present a taxonomy of attack scenarios and the mitigations designed to thwart them.
(5) We show how Akamai DNS provides high performance by anycast traffic engineering and two-tier delegation. We measure the performance of two-tier delegation and show that it reduces DNS times for 87-98% of resolutions over a single-tier.

**Roadmap:** The rest of the paper is laid out as follows. In §2, we characterize the workload that Akamai DNS supports. Then in §3, we present the system architecture. Next, §4 and §5 describe the architectural features and algorithms that provide failure resilience, attack resilience, and performance. Finally, we list related work (§6) and conclude (§7). This work does not raise any ethical issues.

## 2 CHARACTERIZING QUERY TRAFFIC

We analyze the DNS queries served by Akamai DNS to understand its basic properties and to justify the design decisions we made in architecting Akamai DNS as described in this paper. Further, since Akamai DNS serves a wide cross-section of the Internet ecosystem, its query traffic is representative of how end-users across the world access DNS as a prelude to accessing content and applications.

We analyze traffic served by Akamai DNS over a typical week in December 2019. In this period, Akamai DNS served ~360B DNS queries per day originating from over 5.4M source IP addresses. As shown in Figure 1, the rate of queries received varies diurnally from 3.9M to 5.6M queries per second (qps), with weekend-weekday variations. Using the EdgeScape geolocation service [3], we geolocate the source IP addresses of DNS queries. While we observe DNS queries from all around the globe, 92% of queries arrive from source IP addresses in North America, Europe, and Asia.

We now examine how the DNS queries are distributed among source IP addresses of resolvers. Figure 2 in line "IPs" shows a CDF of what percent of resolver IP addresses account for what percent of the total DNS traffic. The 3% of resolver IP addresses that drive the most DNS queries account for 80% of all DNS queries, similar to

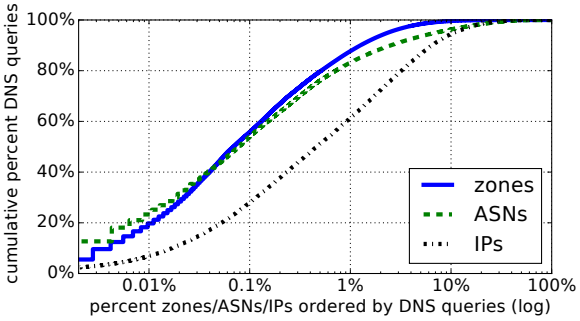


**Figure 2: Percent of queries for/from percent of zones, ASNs, and source IP addresses.**

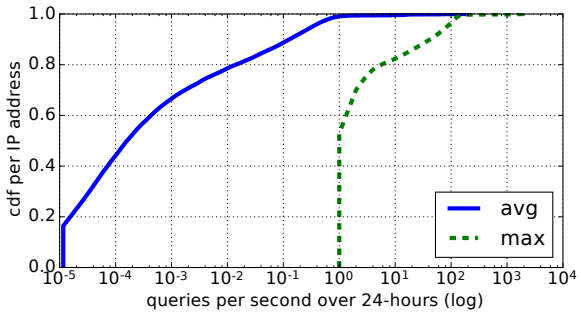


**Figure 3: The avg/max queries per second per resolver.**

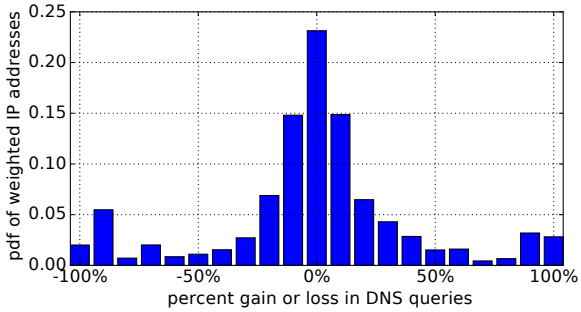


**Figure 4: Change in query rate of resolvers in a week.**

observations in [17]. The resolvers that drive the most DNS queries to Akamai DNS are also highly consistent over time. Using a list of the top 3% of resolvers by DNS queries constructed weekly over 69 weeks, we find that week-to-week the lists contain 85-98% (mean 92%) of the same resolvers and month-to-month 79-98% (mean 88%). The "ASNs" line shows that 1% of ASNs account for 83% of DNS queries. The top 6 ASNs include 3 public DNS services, 2 major ISPs, and Akamai itself. Both highly-skewed distributions demonstrate that a small and relatively stable set of resolvers drive the majority of DNS queries. The relatively stable access patterns observed here allow us to detect and filter anomalous traffic as described in §4.3.4.

We breakdown the queries by domain requested in our domain hosting service (ADHS)[1]. Figure 2 shows that the top 1% of the zones account for 88% of all DNS queries, with one zone receiving 5.5% of all DNS queries and many infrequently-accessed zones.

Next, we examine the workload on individual authoritative nameservers. Figure 3 shows the queries received by one specific, modestly-loaded nameserver from 60K resolvers. The distribution is highly skewed with most resolvers sending very few queries – less than 1% sent greater than 1 qps on average. Further, we observe that the workload exhibits bursty behavior with the highest average being only 173 qps while the maximum qps observed is 2,352. These observations inform the design of filters that use historically-observed query rates of resolvers to detect and flag anomalous requests, e.g., the rate limiting filter described in §4.3.4.

We also observe that the resolvers sending the most DNS queries to an individual nameserver are consistent over time. Taking two one-hour samples of DNS queries exactly one week apart, we compute per resolver the percent difference in DNS queries sent during the two samples. Figure 4 shows the PDF of the differences, weighted by DNS queries sent. We observed that 53% of the weighted resolvers differed by less than ±10%, indicating the resolvers that send the most DNS queries predominantly continued to do so a week later.

# 3 SYSTEM ARCHITECTURE

Akamai DNS consists of authoritative DNS nameservers that answer DNS queries and supporting components that handle tasks such as metadata processing and transmission, monitoring and analysis, and complex control and business logic. Figure 5 shows the high-level architecture whose components we describe below.

## 3.1 Authoritative Nameservers

To provide quick responses to DNS queries received from resolvers all around the world, Akamai's authoritative nameservers number in the tens of thousands and are distributed among hundreds of points of presence (PoPs) in 157 countries. Like many other large DNS platforms [12, 18, 39], Akamai relies heavily on IP anycast to distribute load among the PoPs and to reduce the round-trip-time (RTT) between resolvers and the authoritative nameservers. We use a total of 24 distinct IPv4-IPv6 anycast prefix pairs for the authoritative service. Each prefix pair forms an "anycast cloud" of PoPs, from which they are advertised. To provide resiliency to PoP failures, each of the 24 clouds are distributed among the PoPs, with no PoP advertising more than two clouds.

**PoP Architecture:** Each PoP (Figure 6) consists of a router in front of one or more purpose-built machines running our specialized nameserver software. Besides the nameserver, each machine also runs a BGP-speaker that establishes a session with the PoP router and advertises the clouds assigned to the PoP over that session. The machines also run a local monitoring agent which continuously tests the nameserver's health. If an problem is detected, the BGP-speaker [40] withdraws the advertisement of the anycast clouds, as further discussed in §4.2. When the router receives a BGP advertisement of a cloud from at least one machine within the

[1]CDN and GTM use specific zones owned by Akamai and traffic patterns are likely unique to Akamai. ADHS, on the other hand, hosts generic third-party zones that enterprises may create for any purpose.

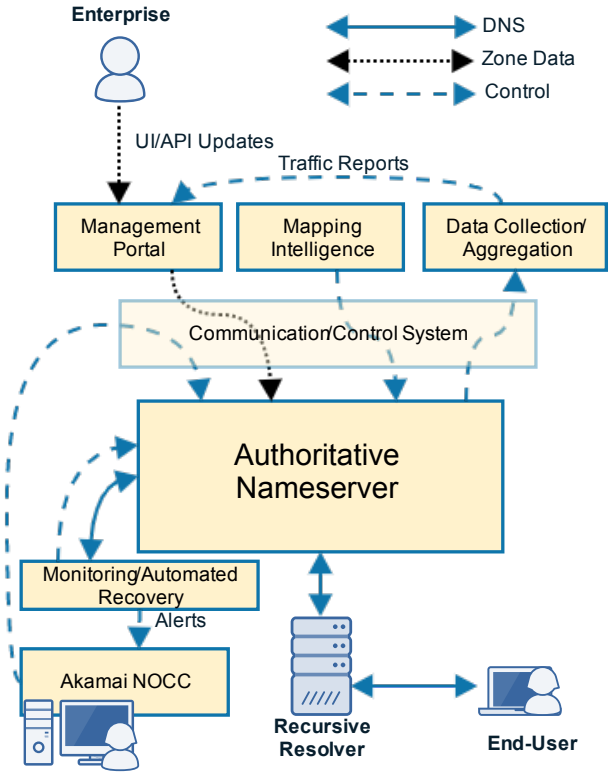


**Figure 5: Akamai DNS high-level architecture.**

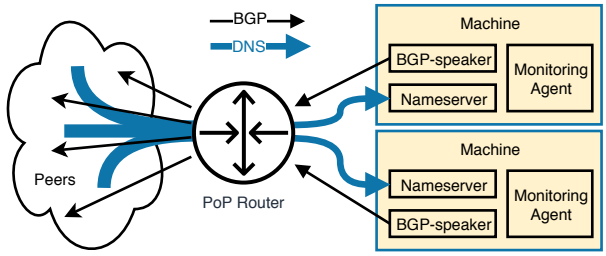


**Figure 6: Architecture of a point of presence (PoP).**

PoP, it advertises the cloud to the PoP's BGP neighbors, or peers. The number of peers per PoP varies from PoPs within eyeball networks peering with only that network to PoPs in Internet exchange points (IXPs) having hundreds of peers. Features of BGP advertisements, e.g., AS Path and BGP Communities [10], are controlled on a per-peer basis.

Packets arriving at the router destined for one of the anycast prefixes are forwarded to only one of the machines within the PoP that advertises the prefix to the router using Equal-Cost-MultiPath (ECMP) [20] by creating a hash from the tuple of (source IP address/port, destination IP address/port). Because most resolvers use a random ephemeral source port per DNS query [47], each DNS query from the resolver may be routed to any of the machines in the PoP advertising the prefix. DNS traffic spreads approximately uniformly across the machines at sufficiently large volumes. However, resolvers that do not use a random ephemeral source port will always be forwarded to the same machine.

**Authoritative DNS Services:** The authoritative nameservers support the *Authoritative DNS Hosting Service (ADHS)*. Enterprises who wish to host their own DNS zones (e.g., "ex.com") on Akamai's infrastructure are assigned a unique set of 6 different clouds called a delegation set from the total 24 clouds, enabling the architecture to support up to $\binom{24}{6}$ enterprises before adding additional clouds. Enterprises add NS records, each corresponding to a cloud in the delegation set, to every zone they own, along with the respective parent zone in the DNS hierarchy. Adding the NS records to the parent zone ensures that resolvers are directed to Akamai DNS, and will query one of the 6 clouds to obtain an answer to DNS queries for the enterprise's zones. We discuss the design decision to use unique delegation sets in §4.3.1.

The nameservers also host domains for the *Content Delivery Service (CDN)*. Enterprises using the CDN redirect a hostname in their zone to Akamai DNS, e.g., "www.ex.com" ⇒ "ex.edgesuite.net", The domain "edgesuite.net" is an entry point to the Akamai CDN and is delegated to 13 anycast clouds[2] because of its cross-enterprise role. These human-readable hostnames are themselves redirected to hostnames used by the CDN – e.g., "ex.edgesuite.net" ⇒ "a1.w10.akamai.net" – to add an additional layer of indirection and control. Hostnames like "a1.w10.akamai.net" resolve to the CDN edge servers that serve content. Domains like "w10.akamai.net" take advantage of nameservers co-located with the wide CDN footprint – which is deployed within 1,600 networks worldwide [54] – to accelerate resolution of hostnames, as discussed in §5.2. Integration with the GTM service is similar to CDN.

## 3.2 Supporting Components

We describe other components in Figure 5 that either publish metadata to authoritative nameservers or monitor them.

**Mapping Intelligence:** The Akamai mapping system [11, 36] determines to which edge servers end-users are directed for content delivery. Towards this end, Akamai DNS changes the IP address returned for a hostname, in response to the query's source IP address or EDNS-Client-Subnet option [13]. While the mapping intelligence determines what IP addresses should be returned, the nameservers are charged with delivering that answer. In practice, this means the mapping system publishes frequent metadata updates in reaction to changing conditions, to which the nameservers subscribe.

**Management Portal:** Enterprises make modifications to their DNS zones, GTM configurations, and CDN properties through the Management Portal via the website or API, while DNS zones can also be updated through zone transfers [29]. The Management Portal validates the metadata and publishes it for consumption by the nameservers.

**Communication/Control System:** This system provides generic metadata delivery services using a publish/subscribe model. The Mapping Intelligence and Management Portal publish metadata to these systems and the nameservers request subscription from these systems. Enterprise DNS zone files and configuration are

[2]We chose 13 delegations to match the model used by the root and many critical toplevel domains.

delivered via Akamai's CDN using a proprietary protocol built upon HTTP. Mapping intelligence requires near real-time delivery for rapid reaction to changing network conditions and so uses Akamai's overlay multicast network [4, 25].

**Monitoring/Automated Recovery:** This system aggregates health data across nameservers, tracks trends, and alerts human operators in the Network Operations & Control Center (NOCC) when anomalies occur. But, the speed of this process is bounded by human operations, and our goal is to mitigate impact as quickly as possible. Thus, a monitoring agent is deployed with each nameserver to continually detect and mitigate a variety of issues (§4.2).

**Data Collection/Aggregation:** Finally, metrics published by nameservers are also compiled into reports displayed to enterprises through the Management Portal.

## 4 RESILIENCY

Akamai DNS is a crucial component of the global Internet ecosystem. As such, resiliency is factored into every aspect of its design. We consider two types of resiliency: *failure resiliency* which is the ability of the systems to tolerate failures either of the systems themselves or the underlying network (§4.2), and *attack resiliency* which is the systems ability to protect itself from malicious attack (§4.3).

### 4.1 Anycast Failover Mechanism

Anycast failover is a key mitigation mechanism for events such as a PoP failure. By withdrawing a prefix from one PoP, it allows traffic to be rerouted to another PoP within the same cloud. The time for such rerouting to occur is called *failover time*. We show that failover time is small enough to justify its use in our system.

**Experimental Methodology:** We conduct experiments to measure failover time for two cases: advertising a prefix and withdrawing a prefix in a 2-PoP anycast cloud (Figure 7). We select 267 CDN edge servers – selected to roughly cover our geographic footprint – to use as vantage points and instrument them to send DNS queries to an IP address within a test prefix every 100 msec. When a nameserver receives one of these DNS queries, it responds uniquely identifying its PoP. The vantage points log the time that the DNS queries are sent and the response that was received (or timeout if no response received).

Figure 7(a) shows our setup for measuring the impact of a new advertisement. A nameserver within PoP $Y$ is already advertising the prefix and all vantage points are routed to $Y$. Next, a nameserver in PoP $X$ is instructed to advertise the prefix and the BGP-speaker resident with the nameserver advertises the prefix to $X$'s router shortly there after, triggering the router to update it's routing table and propagate the advertisement to its peers. Within 100 msec of $X$'s router updating its routing table, the local vantage point within $X$ will issue a DNS query, receive a response identifying $X$, and log the time the query was sent, $t_L$. As the BGP update propagates through the Internet, remote vantage points will also receive DNS responses identifying $X$ and log the time $t_X$. We estimate failover time as the time from the BGP advertisement to when the application is routed to $X$ as $t_X - t_L$. This calculation uses two different clocks. All vantage points sync with the same set of NTP servers and we estimate that the clock discrepancy is 7.4 msec average and 46ms in the worst case across all pairs of vantage points. Combined with the

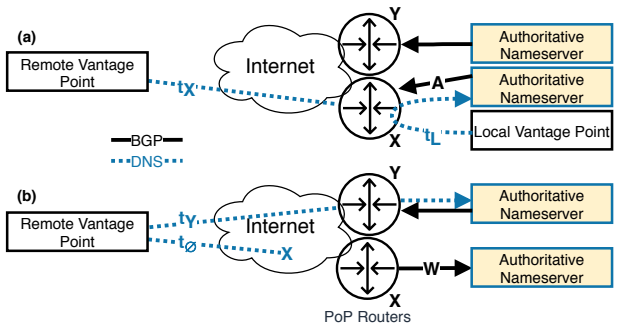


**Figure 7: Experimental setup for evaluating failover times for prefix (a) advertisement and (b) withdrawal.**

100 msec measurement frequency, our measurements are accurate to within $[-50, 250]$ msec and overestimate failover time by $100 \pm 7.4$ msec on average.

Figure 7(b) shows our setup for prefix withdrawal. The nameserver in PoP $X$ withdraws the advertisement while PoP $Y$ continues to advertise. Unlike with the advertisement experiment above where it took some time for vantage points to be routed away from $Y$, with withdrawals the vantage points stop receiving DNS responses from $X$ *immediately*. This is because at some point along the path between the vantage point and $X$, the packet traverses a router that has already updated its routing table. At that point, one of two things can happen: ($i$) the packet will be re-routed eventually reaching $Y$, or ($ii$) the packet will bounce between routers with divergent routing tables and ultimately be discarded when IP TTL = 0. The former case results in instantaneous failover, while the latter results in timeouts until the BGP routing tables converge. We measure the failover time in the latter case as the time $t_\phi$ when the vantage point sends the first DNS query that results in a timeout to the time $t_Y$ when the vantage point sends the first DNS query that gets an answer from $Y$. This calculation depends upon a single clock, making clock sync irrelevant.

For both the new advertisement and withdrawal experiments above, we cycled through a random permutation of the 267 PoPs, advertising and withdrawing the test prefix from each PoP $X$, using the previous PoP in the permutation as $Y$, and measuring failover time using the remaining PoPs as the vantage points. In each experiment, we waited 5 minutes for the vantage points to fail over, before continuing to the next PoP. Finally, to understand failover for larger anycast clouds, we reran our experiments again cycling through all 267 PoPs, and randomly selecting 20 other PoPs to act as $Y$, rather than using a single PoP as in the first experiment.

**Experimental Results:** Figure 8 shows the failover time for a new advertisement in the line "advertise 2 PoPs". In 76% of the measurements, failover time is under 1 sec. Further, some vantage points experienced timeouts, i.e., were not routed to either $Y$ or $X$, but this occurred in only 3% of measurements. We also see that the failover time for withdrawals is similar to that of a new advertisement in line "withdraw 2 PoPs"[3]. However, the failover time has a significant tail with 5.8% of the measurements taking 10

[3]The withdraw line has steps at our measurement granularity unlike the advertise line which is smoothed due to clock jitter.

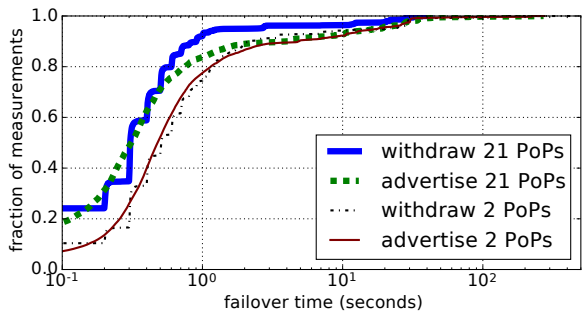


**Figure 8: Failover time for clouds with 2 and 21 PoPs.**

seconds or more. The tail includes measurements using 19% of PoPs and all vantage points, so we conclude that it is likely not driven by localized network issues at the time of our measurements.

Figure 8 also shows the results for 21-PoP experiments. The median failover time for both advertising and withdrawing decreases by 200 msec in comparison with the 2-PoP case. The reason is that the set of vantage points in the catchment of a PoP and the topological distance a BGP update must travel from a PoP to a vantage point are both smaller when the number of PoPs is larger. Thus, 2-PoP failover likely captures the worst-case times for anycast failover.

Finally, because we wait 5 minutes for vantage points to failover, it is possible that we do not observe failovers that take longer than 5 minutes. We note, however, that in the 21 PoP withdraw experiment we observed 0 vantage points that timed out for ≥5 minutes, indicating that very long failover times are extremely unlikely. In conclusion, these results suggest that most resolvers would failover within a second. Thus, anycast failover is a suitable mechanism for making Akamai DNS failure resilient.

**Relation to Prior Work:** BGP update propagation through the Internet has been studied before. In 2000, [27] observes that BGP convergence for route advertisements typically takes 1-2 minutes and route withdrawals greater than 2 minutes, with the time required varying among 5 different ISPs. More recently in 2011, [5] measured propagation of a route advertisement from the Amsterdam Internet Exchange (AMS-IX) to 90 vantage points around the globe and observed an advertisement propagating to all in 38 seconds and a withdrawal in 3 minutes. We complement these existing studies by (*i*) updating findings to the state of BGP propagation as of 2020, and (*ii*) covering the case of anycast advertisements where the same prefix is advertised from multiple PoPs. Importantly, our experiments are also the first to measure application-layer failover for DNS resolutions rather than BGP convergence. Previous studies demonstrate that BGP convergence can take minutes, whereas we demonstrate that failover between the PoPs at the application layer is much faster. This is because failover does not require *full* propagation of the BGP updates to the entire Internet.

## 4.2 Failure Resiliency

Akamai DNS must be resilient to all sources of failure, including the software, hardware, and network. While software releases are vetted via a thorough QA process and extensive effort is made to validate inputs, some problems may only present at the nameservers themselves. Thus, Akamai DNS is built to tolerate failures and continue to operate – even if in a degraded state – until fully recovered. Here, we cover a few specific failures and how the design mitigates them, allowing Akamai DNS to continue answering DNS queries.

*4.2.1 Machine-Level Failures.* In large distributed networks like Akamai DNS, it is not unusual for a small number of machines to experience software or hardware failures at any given time. Therefore, Akamai DNS is built to identify failures and shift DNS query traffic to healthy machines.

The most common failure mode we observe is disk failure, but any hardware subsystem (e.g. memory, network card) can fail. Hardware failures often manifest in the nameserver software not responding to DNS requests, or responding slowly, or responding with incorrect answers (e.g. answering based on stale data). Also, despite our rigorous QA process, some bugs are only observable in production due to a confluence of unpredictable events. These bugs can manifest themselves in ways similar to hardware failures.

We deploy a common mitigation strategy to handle localized failures. Every nameserver is monitored by an on-machine monitoring agent (Figure 6) that continually runs a suite of tests against the nameserver and detects incorrect or missing responses. The test suite includes DNS queries for each DNS zone and regression tests for known failure cases. If a failure is detected, that machine is *self-suspended*, the monitoring agent instructs the BGP-speaker to withdraw anycast advertisement, resulting in traffic shifting to other healthy machines. If all machines within a PoP are self-suspended, the anycast failover mechanism of §4.1 will route the DNS requests to other PoPs. But, there is a danger to self-suspension if the nameserver failure is widespread or the bug is in the monitoring agent itself. Either could lead to widespread self-suspension, significantly reducing capacity. The Monitoring/Automated Recovery system (Figure 5) prevents such scenarios by limiting concurrent nameserver suspensions using a distributed consensus algorithm, and preventing self-suspension on some nameservers (§4.2.3). In this way, Akamai DNS is designed to always return an answer, even if there are widespread failures.

*4.2.2 Stale State.* The metadata on which nameservers base their answers can change rapidly, particularly the Mapping Intelligence metadata (§3.2). The consequence of serving DNS answers based on stale metadata can be poor performance or an outage for end-users.

Typically, updates propagate in less than 1 second, however we observe a small fraction of nameservers with stale metadata at any time. Stale state can be caused by the scenarios described in §4.2.1, but it can also occur for reasons independent of machine level faults. One common cause of stale state is isolated connectivity issues. Similar to hardware failure, isolated connectivity failures are common in large networks with causes including hardware failures in switches/routers, cable cuts, and misconfigurations. Once connectivity is restored, the nameserver will have stale state for a brief period until catching up. During this time, DNS queries could be answered incorrectly, if not mitigated.

A particularly insidious case is a partial connectivity failure, causing the nameservers to be unable to receive metadata from the Akamai network, yet still able to receive DNS queries from some subset of the Internet. The most common such failure mode is when

the transit links – typically the links over which metadata arrive – for the PoP fail, but DNS traffic still reaches the nameservers via peering links.

To mitigate the issues described above, the nameservers check for staleness in critical state and, if determined to be stale, self-suspends as described in §4.2.1. The exact criteria for staleness varies among metadata. A common strategy is to declare state stale if a critical input's timestamp is older than a threshold.

*4.2.3 Input-induced Failure.* Since the nameservers consume a wide variety of metadata inputs from varied internal and enterprise-related sources, a great deal of care goes into validating each of these inputs to ensure the safety of the nameservers. However, despite this effort, there remains a highly unlikely but not impossible scenario where a new input exercises a bug in the nameservers leading to widespread crashes and potentially an outage. Even with very long odds, such a scenario must be mitigated in order to meet our resiliency mandate and protect the Internet ecosystem.

Akamai DNS protects against input-induced failures using *input-delayed nameservers*. For each of the 24 anycast clouds, one PoP is selected to house the input-delayed nameservers (in addition to regular nameservers) that differ from other nameservers in three ways. First, they receive all inputs with an artificially imposed 1-hour delay. Second, they do not self-suspend due to input staleness. Third, the BGP-speaker running along side the input-delayed nameserver advertises the anycast prefixes to the POP's router with a higher Multi-Exit Discriminator (MED) value than other nameservers. The router prefers the advertisements with lowest MED. So, in the common case where the regular nameservers are also advertising the anycast prefixes to the router, the input-delayed nameservers receive no DNS traffic.

The input-delayed nameservers will receive DNS traffic, however, when all other nameservers within the PoP withdraw their advertisements, as would occur if an input caused them all to crash. Similarly, if all other PoPs advertising the same anycast prefix also withdraw their advertisements due to crashes, then all traffic globally to the anycast prefix will failover to the input-delayed nameservers within seconds as shown in §4.1. Since the input-delayed nameservers have not yet received the input, they continue to answer DNS queries with intentionally stale data ensuring that Akamai DNS remains available, until Akamai DNS is fully restored. Also, the input-delayed nameservers stop receiving any new inputs upon use, giving the operations team ample time to identify and resolve the issue. Thus, the input-delayed system reduces an extremely rare but potentially devastating outage to a period of degraded service until mitigated.

*4.2.4 Query-of-Death.* Given that software crashes due to unexpected client traffic are a potential failure mode for all networked systems, it is important for any DNS infrastructure to be resilient against unexpected DNS queries, regardless of whether there is malicious intent behind them. We call a DNS query that causes the nameserver to crash a query-of-death (QoD). Although they are extremely rare, we observe that a QoD is seldom a malformed packet not conforming to the relevant DNS RFCs. More often, a QoD arises due to a corner-case in a complex query processing code path. No matter the cause, when a nameserver crashes during answering a query, the resolver will not receive an answer, eventually leading to timeout & retry. If crashes are frequent, QoDs can cause a partial or total service outage.

When a nameserver crashes, the on-machine monitoring agent (Figure 6) detects it and instructs the BGP-speaker to withdraw anycast advertisements, causing the router to forward traffic to other machines in the PoP. However, forwarding a QoD to other nameservers is problematic, as it could make them crash as well.

To mitigate QoDs, the nameservers detect unrecoverable faults in their query processing logic and write the DNS payload of the packet that it is currently processing to disk. A separate process on the machine constructs and inserts a firewall rule to drop similar DNS queries, preventing repeated crashes due to potential QoDs, while allowing the nameserver to continue answering dissimilar queries. However, the firewall rule may be too broad, dropping false positives. Therefore, the rule is expunged after a configurable time $T_{QoD}$, so that the nameserver will occasionally attempt to answer potential QoDs while limiting the crash rate to at most once per $T_{QoD}$. Further, this feature is only deployed on a subset of nameservers. Thus, queries similar to the QoD that *do not* themselves cause crashes experience a partial outage at worst while operations teams work to identify the precise cause of the crash.

## 4.3 Attack Resiliency

Distributed Denial of Service (DDoS) attacks against authoritative nameservers are frequent [6, 33] and sufficiently large attacks could bring down all services the DNS supports. It is crucial that Akamai DNS continues responding to valid DNS queries during attacks. A DDoS attack attempts to exhaust the compute and/or network resource of the DNS infrastructure. We describe architectural features for resiliency and then show how these features can be put into play in the context of both observed and hypothesized attack scenarios.

*4.3.1 Distributed Deployment.* The first line of defense against attacks is our highly distributed deployment. As mentioned in §3.1, each enterprise is assigned a *unique* set of 6 anycast clouds to use for their DNS zones and each anycast cloud is advertised from a large set of PoPs. These PoPs are distributed worldwide and connected to the Internet with thousands of peering links. Individually, PoPs are over-provisioned in both bandwidth and compute to handle spikes in traffic, allowing them to absorb a large distributed attack. No PoP supports more than two anycast clouds. Even if an attacker saturates a PoP that advertises one or two of the 6 clouds that support a zone, resolvers, upon receiving a timeout, will retry against the other 4-5 clouds assigned to that zone [34]. Since the resolver is routed to different PoPs for the other clouds, the resolver will, with high probability, obtain an answer to the query.

Further, in case the target of an attack is a specific enterprise deployed on Akamai DNS, rather than Akamai DNS itself, the uniqueness of the 6 delegations used by that enterprise limits the collateral damage to other enterprises not directly under attack. In the worst case scenario that the PoPs advertising the clouds assigned to enterprise $A$ are all saturated, any other enterprise $B$ will have at least one delegation not in common with $A$ and likely advertised from a different PoP. Resolvers thus will be able to obtain an answer for $B$'s DNS zones even in the worst case scenario. The design choice of using 6 delegations is arbitrary and serves only to

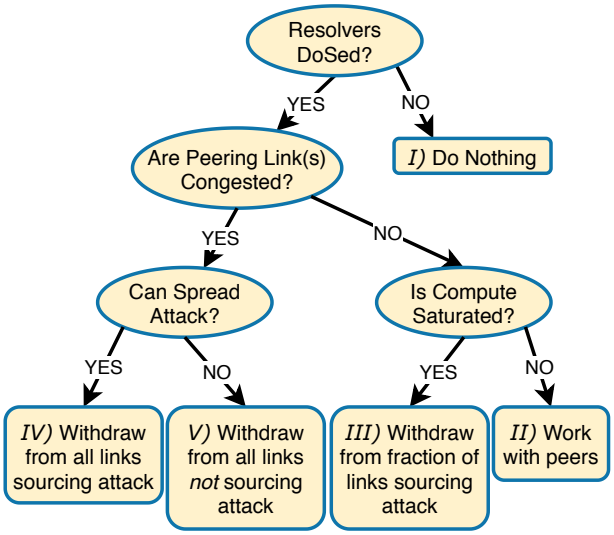


**Figure 9: Decision tree of anycast traffic engineering actions taken during an attack.**

balance between assigning each enterprise a unique set and limiting the total number of clouds needed.

*4.3.2 Anycast Traffic Engineering.* Another tool to combat DDoS attacks is traffic engineering via BGP advertisements. As noted in [33], PoPs within an anycast cloud may either absorb attacks or withdraw advertisements to shift the attack to other PoPs. Since anycast prefixes are advertised to each peer at each PoP individually, the decision to withdraw can be made per advertisement. A human operator chooses an action during an attack following the decision tree in Figure 9 as described below.

*I*) The preferred action is always *do nothing*. As described in §4.3.1, resolvers are only DoSed if multiple PoPs are saturated causing packet loss on all delegations for a zone. If that is not the case, then absorbing the attack at the few saturated PoPs effectively mitigates the attack. We also note that any active reaction leaks information which could be of use to the attacker to improve their attack. Further, shifting traffic among PoPs during an attack can reduce the effectiveness of some automated mitigation mechanisms described in §4.3.4. To know whether resolvers are DoSed we rely upon our external monitoring and information sharing with peers.

*II*) If resolvers are DoSed, determine what resource (bandwidth or compute) is saturated. Measuring saturation of compute on the nameservers is straightforward, while peering link congestion can be determined with external monitoring or information sharing with the peer. If neither is saturated, then there is likely upstream congestion and we *work with peers* to determine where and how to mitigate it.

*III*) If compute is saturated, *withdrawing from a fraction of peering links sourcing attack traffic* can disperse the attack among more PoPs while absorbing a manageable fraction of the attack traffic in each PoP.

*IV*) However, if one or more peering links are congested, *withdrawing from these attack-sourcing links* will shift the traffic elsewhere, possibly to larger peering links or spreading the attack across more peering links. Deducing exactly how anycast traffic will shift can be hard, but in many cases we can infer that the other PoPs with links to the same peer from which we withdraw will absorb the attack.

*V*) If spreading the attack is not possible, then *withdrawing from non-attack-sourcing links* minimizes the collateral damage by shifting as much legitimate traffic out of the saturated PoP as possible.

Finally, we note that while the above reactions are described in terms of withdrawing routes, there are alternatives including appending BGP communities[10] to implement remote triggered blackhole filtering [35] or path prepending to reduce preference for the route. Deciding which action to take is non-trivial and potentially requires discussion with our network peers. Together with the sensitivity of the issue and our preference to take no action unless needed, we opt to leave the traffic engineering decisions to human operators. Instead of automated systems for these tasks, we focus on rich controls and rapid delivery of configuration safely to PoPs that are under attack. Automated mechanisms to perform traffic engineering and share information between network peers are important areas for future work.

*4.3.3 Query Scoring and Prioritization.* To complement the distributed mitigations described earlier, we also built mitigation mechanisms that run on each machine as a part of the nameserver software. Each query received by the nameserver is first given a penalty score that represents the "legitimacy" of the query, where "suspicious" queries receive more penalty than "legitimate" ones. Then, when the queries are processed to generate a response, the legitimate queries with lower penalty scores receive more resources than the queries with higher penalty scores. This allows the nameserver to prevent malicious queries from exhausting resources that it could have used to serve legitimate ones. We describe this approach in more detail below.

**Query Scoring:** Each DNS query passes through a sequence of filters (described in §4.3.4), where each filter performs a set of checks on the query parameters and adds a penalty score to the query if needed. The total penalty score $S$ assigned by the filters is a measure of the legitimacy of the query. Next, the DNS query is placed into one of a configurable number of queues according to score. Each queue $i$ has a maximum score value, $M_i$ and the query is placed into the queue $i$ with the minimum $M_i$ such that $S \leq M_i$. Queries with a high score, $S \geq S_{max}$, are discarded outright as definitively malicious.

**Query Processing:** Queries are read from queues in the increasing order of penalty for processing. If a lower-penalty queue is empty, it reads from the next higher-penalty queue. In this way, more legitimate queries are processed ahead of suspicious queries. Our query processing is work-conserving, so if there are any enqueued queries, it will attempt to answer them, even if suspicious. Starvation is allowed in all queues except for the lowest-penalty queue. We note that starvation is only possible if the compute capacity of the nameserver is saturated answering lower-penalty DNS queries.

*4.3.4 Attack Scenarios and their Mitigations.* We present a taxonomy of DDoS DNS attacks and show which architectural features and mechanisms described above are most effective at mitigating each type of attack. We present the attacks in the order – from

our perspective – of the simplest to the most complex in both the attacking instrument and mitigation mechanisms. Note that this is not equivalent to ordering based upon impact or cost of the attacks, as each one of these attacks can have significant impact if not appropriately defended. Each attack is unique and all of Akamai DNS's mitigation mechanisms are reconfigurable so that they can be tuned to react to a specific attack.

**1) Volumetric:** The goal in this class of attack is to saturate the available bandwidth and cause DoS by dropping legitimate traffic in queues at routers along the path. The attack traffic used need not be DNS queries because the target is not the application but the underlying network. Attacks in this class may use sources of amplification including DNS reflection [23] or NTP reflection [14]. The attack traffic is typically easy to filter, e.g., simple firewall rules can drop anything not destined to port 53 or distinguish DNS reflection traffic from legitimate DNS queries using the QR-bit. In practice, we observe that the bottleneck for volumetric attacks is usually upstream from the nameservers as we have sufficient compute capacity to filter in the firewall at a higher rate than the bandwidth available in peering links. Thus, volumetric attacks are the only class of attacks listed here that typically fall into the category of bandwidth saturating rather than compute saturating. Mitigating them is a matter of having sufficient bandwidth to absorb the attack and filtering in the firewall so that the traffic never reaches applications. We respond to this class of attacks by overprovisioning peering links and reacting to saturated links as described in §4.3.2.

**2) Direct Query:** The simplest DNS-based DoS attack is to send DNS queries directly to authoritative nameservers from one or more attack machines. While this attack could saturate either bandwidth or compute, in practice we observe that compute tends to be the bottleneck for any class of attack that arrives at the application. To combat this attack, we use a *rate limiting filter* in the query scoring module that learns the "typical" query rate (in qps) of resolvers from historical data and assigns a rate limit on a per-resolver basis. A query received from a resolver that is over its rate limit is assigned a penalty score. As shown in Figure 3, DNS traffic is bursty, hence we use a leaky bucket rate limiting mechanism.

Rate limiting is most effective when the attack is from a small number of source IP addresses, but becomes less effective when the attack is from a large number of source IPs that each need to be rate limited, e.g., a Mirai botnet attack[24]. As the cumulative volume and source diversity of the attack increases, the query scoring module activates an *allowlist filter* that maintains an "allowlist" of resolvers that are historically-known to Akamai DNS. As noted in §2, the resolvers that drive the most DNS queries to Akamai DNS are consistent over time, and so the allowlist changes only gradually. Queries originating from sources not in the allowlist are assigned a penalty, de-prioritizing them further.

**3) Random Subdomain[52]:** This unique attack deserves special attention because of how common it is and its ability to "pass-through" resolvers. By randomizing the hostname in each query and sending the query to resolvers, an attacker can force extremely low cache hit rates in resolvers, causing the resolvers – including ones on the allowlist – to send a high volume of queries to Akamai DNS. Because the traffic originates from resolvers, the above described filters are ineffective as the rate limiting filter is equally likely to assign a penalty for a legitimate query as a random subdomain attack query from the same resolver.

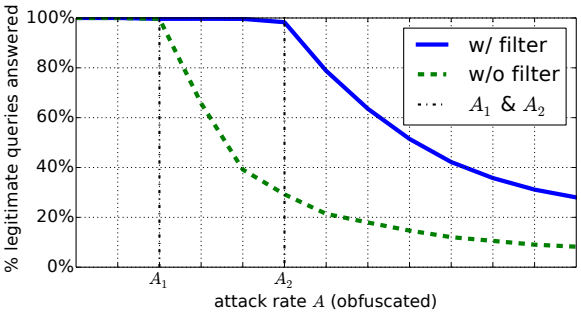


**Figure 10: Percent legitimate queries answered with & without NXDOMAIN filter.**

To combat this class of attacks, our query scoring module uses the *NXDOMAIN filter* that exploits the fact that the random hostnames[4] used in attack do not exist, resulting in an NXDOMAIN response. Thus, during a random subdomain attack, early identification of queries that will result in an NXDOMAIN response and filtering them can potentially mitigate the attack. Legitimate traffic is unlikely to be penalized by this filter as NXDOMAIN responses are rare in legitimate traffic, accounting for only ~0.5% of the DNS responses Akamai DNS typically returns.

The NXDOMAIN filter functions by tracking NXDOMAIN responses per zone and if the count exceeds a threshold, the filter builds a tree of all valid hostnames in the zones above the threshold. Queries for hostnames in the zones that are not present in the tree are assigned a penalty score. An alternate approach is to build a tree from *all* zones, rather than just those zones that exceed a threshold number of NXDOMAINs. However, this approach results in a tree that is much larger and updating such a tree results in greater contention due to locking.

We use a testbed comprised of two machines connected via a switch to demonstrate the effectiveness of query scoring and prioritization. We focus on the NXDOMAIN filter. The other filters described in this section behave similarly when applied to the attack traffic that they are designed to mitigate. One machine in the testbed acts as the source of DNS query traffic while the other is a nameserver. From the source, we drive both legitimate traffic sampled from observed production traffic and attack traffic where the hostnames are selected from a test domain prepended with a random string. The legitimate traffic is set at a fixed rate of $L$ queries/sec while the attack rate of $A$ queries/sec is ramped up over time. Figure 10 shows the percentage of the legitimate traffic answered versus the attack rate $A$ and has three regions of interest. In the first region where $A \leq A_1$, the cumulative query rate $A + L$ is smaller than the processing capacity of the nameserver, so all legitimate queries are answered with or without the filter. In the second region $A_1 < A \leq A_2$, the nameserver does not have sufficient processing capacity to answer all of the DNS queries received. Without the filter, the percentage of legitimate queries

[4]Often implemented by prepending a random string onto a valid zone, e.g. "a3n92nv9.akamai.com".

answered decreases as legitimate queries are equally likely to be dropped as attack queries. With the filter, the nameserver continues to answer nearly all of the legitimate queries as they are prioritized over the attack queries. In the third region when $A > A_2$, we reach the I/O capacity of the nameserver machine. The nameserver software is unable to read queries off of the network stack as fast as they arrive causing drops below the application layer of both legitimate and attack queries. These results demonstrate that the NXDOMAIN filter can effectively increase the cumulative rate that the nameserver can handle before dropping legitimate queries.

**4) Spoofed Source IP:** A modification of direct query attacks occurs when attackers spoof the source IP address, both hiding the origins of the attack and enabling the use of many more source IP addresses than physical machines. The rate limit filter quickly becomes ineffective due to the large set of source IPs an attacker is likely to use, while the allowlist filter remains effective. But, an attacker may intelligently spoof IP addresses to impersonate known resolvers (e.g. Google Public DNS[18]), including ones on the allowlist, causing allowlist filtering to also be ineffective.

To combat this class of attacks, we use the well-established technique of *hop-count filtering* [22]. The hopcount filter learns the IP TTL of DNS queries for resolvers on the allowlist using historical data. When the IP TTL of a DNS query diverges from the expected value, the query is assigned a penalty score. We observe in the DNS traffic arriving at our nameservers that the IP TTL is consistent per source IP address, with only 12% of source IP addresses showing any variation in IP TTL over one hour and 4.7% ever varying by more than ±1. On the other hand, when an attacker spoofs a resolver IP address from a different topological location than that resolver, it is likely that the spoofed query will arrive at the nameserver with a different IP TTL.

**5) Spoofed Source IP & IP TTL:** Further enhancing the previous attack, we hypothesize that an attacker can spoof *both* the source IP address and IP TTL of allowlisted resolvers. This implies that the attacker knows the number of hops from the allowlisted resolver to Akamai DNS. To combat this sophisticated attack, the query scoring module contains a *loyalty filter*. Each nameserver *independently* tracks the resolvers that historically send DNS queries to it. Recall the use of anycast for our nameservers and that each resolver is routed to a PoP via BGP. Thus, allowlisted resolvers only appear in the loyalty filter of nameservers to which the allowlisted resolver is routed. When a nameserver receives a query from a resolver that is not in the loyalty filter, the query is assigned a penalty score. Thus, the attacker must not only spoof the source IP address and IP TTL but also be routed to the same PoP as the allowlisted resolver in order for the attack traffic to not be filtered. Further, since the resolvers that drive the most DNS queries to nameservers are consistent over several days (Figure 4), they will with high probability be in the loyalty filter.

**Discussion.** Mitigating attacks by shifting the resolver traffic via traffic engineering actions such as those described in §4.3.2 can negate the efficacy of filters that rely on leveraging historical traffic patterns. In such a situation, the filters described here do not differentiate between legitimate and attack traffic in the worst-case, and our work-conserving query processing attempts to answer all queries (§4.3.3). This is one reason why the preferred action during an attack is to take no action.

While all of the mechanisms described above can together effectively mitigate a wide range of attacks, we recognize that there is still the possibility of an attack that cannot be distinguished from legitimate traffic. Such a "perfect" attack would have to mimic legitimate traffic so well that the likelihood of its occurrence is extremely low, yet extremely costly. Thus, Akamai DNS is designed for this event, by overprovisioning both bandwidth and compute, and by compartmentalizing the infrastructure as described in §4.3.1.

# 5 DNS PERFORMANCE

While resiliency of Akamai DNS is critical due to its role in the Internet ecosystem, its performance is also important. A significant fraction of requests for Internet content and services start with a query to Akamai DNS, so it is critical that Akamai DNS provides answers with low latency.

## 5.1 Anycast Performance Tuning

Because Akamai DNS uses anycast routing, BGP path selection plays an important part in performance. All 24 anycast clouds are advertised from PoPs spread around the globe, so that there is always a geographically nearby PoP for any resolver to provide low RTT DNS resolutions for all 24 clouds. However, ensuring that the route to the nearest PoP is selected by BGP is non-trivial and requires significant engineering as well as communication with our peers to align our routing policies. Common practice in anycast optimization is to ensure that the peering links at PoPs consist of the same major providers [55] and that the advertisements from those PoPs appear identical upstream. We use these common practices to select which PoPs should advertise which of our 24 anycast clouds and modify our BGP advertisements *per peer* to achieve similarity. Recent work on modeling anycast catchments [49], measuring performance [16], and automated configuration of advertisements [30] help. However, today anycast optimization remains a challenging and operationally time-consuming task. One that we view deserves further study.

## 5.2 Two-Tier Delegation System

Akamai DNS is the entry point for the Akamai CDN, as each content request to the CDN is prefaced by a DNS query to Akamai DNS. To accelerate DNS resolutions for the CDN, Akamai DNS uses the Two-Tier delegation system. Continuing the example from §3.1, the zone "akamai.net" is delegated to 13 anycast clouds, called *toplevels* in Two-Tier context. From the toplevels, the zone "w10.akamai.net" is delegated to a set of unicast *lowlevel* nameservers co-located with the wide CDN footprint. The Akamai mapping system [11, 36] tailors the set of lowlevel delegations to be near the resolver issuing the query. The CDN hostnames use very low TTLs – currently 20 seconds – to enable quick reaction to changing network conditions and edge server liveness. So, the resolvers' cache must be frequently refreshed. The lowlevels provide rapid responses to queries for CDN hostnames, minimizing the cost of refreshes. The delegation from toplevel to lowlevel has a large TTL – currently 4000 seconds – so that resolvers need to refresh the lowlevel delegation set infrequently, Thus, the majority of resolutions occur between the resolver and the lowlevels.

The Two-Tier system accrues two separate advantages over a single-tier of IP anycast toplevels. First, the Two-Tier system is able to utilize lowlevel nameservers deployed with the CDN's edge, including those in co-location sites where it is not possible to inject eBGP route advertisements, and hence not usable for IP anycast. Second, in the Two-Tier system, Akamai is able to route requests from resolvers to a proximal nameserver using its mapping system[11, 36], often achieving lower RTTs than anycast.

We now develop an analytical model of Two-Tier and use it to measure the performance impact of Two-Tier in isolation from other components of DNS performance. The performance achieved by Two-Tier depends upon the resolvers' cache state and the RTTs between the resolver and the lowlevels/toplevels. Consider the resolution of "a1.w10.akamai.net" and let $L$ be the RTT to the lowlevels and $T$ be the RTT to the toplevels[5]. If the resolver has the A/AAAA records for "a1.w10.akamai.net" in cache, there is no need to contact any authoritative nameservers and the resolution takes no time. There is no performance impact to using Two-Tier in this case. However, if "a1.w10.akamai.net" is not in cache but the NS records (and associated A/AAAA records) for "w10.akamai.net" are cached, then the resolver must only contact the lowlevels and the resolution time is $L$ msec. If the records for "w10.akamai.net" are not cached, then the resolver must contact the toplevels first, resolution time $L + T$ msec. We define $r_T$ as the fraction of DNS resolutions that require contacting the toplevels, the value of which depends upon many factors including (*i*) the TTLs of the NS/A/AAAA records involved and (*ii*) the frequency and inter-arrival times of DNS queries from end-users to the resolver for Akamai CDN hostnames. Thus, we can calculate the average resolution time using Two-Tier and find the speedup over answering from the single-tier of toplevels as:

$$S = \frac{T}{(1 - r_T) \cdot L + r_T \cdot (L + T)} \quad (1)$$

When $S > 1$ , Two-Tier reduces resolution time on average in comparison to answering directly from the single-tier of toplevels. Intuitively, Two-Tier is most beneficial when $r_T$ is small – the resolver has to consult the toplevels infrequently – and the difference between $T$ and $L$ is large – the resolver has a shorter RTT to lowlevels than to toplevels.

**Measuring $T$ & $L$:** We use RIPE Atlas [41] to measure $T$ and $L$, scheduling DNS measurements on 1,663 probes, selected with 1 probe per ASN/country combination. The DNS measurements instruct the probes to send a query directly from the probe to the toplevel delegations and lowlevel delegations. For the toplevels, we configure the measurement target as one of the toplevel anycast addresses. For the lowlevels, the measurement target should be the unicast address of a lowlevel tailored to be near the probe. We achieve this by setting the measurement target to the hostname of one of the unicast lowlevel delegations, and using the "Resolve on Probe" option [42], causing the probe to look up the hostname using the probe's resolver first. The experiment ran for one month with hourly measurements and we compute the median RTT against each toplevel and lowlevel delegation, and use the per delegation RTTs to compute $T$ and $L$ as follows. Research in [34, 44, 56] shows

[5]Note that both the toplevel and lowlevel delegation sets contain multiple IP addresses and thus multiple RTTs. In this formulation, we assume an aggregate RTT is used and discuss its computation below.

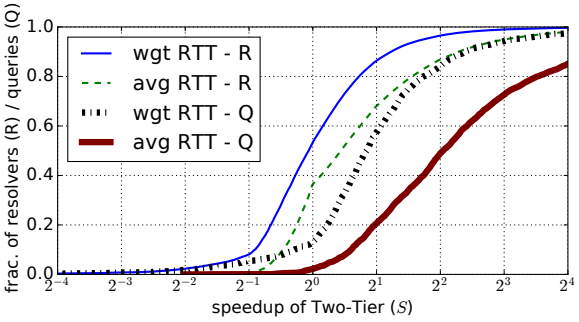


**Figure 11: Speedup in average resolution time using Two-Tier over a single-tier of toplevels.**

a range of behaviors among resolvers in sending DNS queries to delegations, from apparent uniformity to preferencing delegations with lower RTT. The former is a best case scenario for Two-Tier as toplevel delegation RTTs vary widely due to anycast routing, often not coinciding with lowest RTT. Similarly, the latter is a worst scenario for Two-Tier since the highest toplevel RTTs contribute less to the aggregate. Per RIPE Atlas probe, we simulate both behaviors to bound the expected RTT. For the former we calculate the *average RTT*, while for the latter we assume that a resolver's preference for a nameserver is inversely proportional to the delegation RTT and calculate the *weighted RTT*. The lowlevel RTT $L$ is less than the toplevel RTT $T$ for 98% of the probes using the average RTT and 87% of the probes using the weighted RTT. Thus, Akamai mapping reduces the RTT between the resolver and the authoritative nameserver over the RTT of anycast routing for the majority of probes.

**Measuring $r_T$:** Next, we investigate values of $r_T$ using resolvers in the wild. Collecting logs from toplevels and lowlevels over one day, we compute the number of queries received per resolver IP address by toplevels and lowlevels for the domain "w10.akamai.net". For each of the 575K resolver IP addresses in the dataset, the number of queries received by toplevels divided by the number of queries received by lowlevels provides an estimate of $r_T$. The mean value of $r_T$ is 0.48. However, as previously noted in §2, the distribution of DNS queries among resolvers is highly skewed. So, when weighted by the lowlevel DNS queries sent by the resolvers, the weighted mean $r_T$ is only 0.008.

**Results:** Combining the RTT dataset from RIPE Atlas with the traffic logs from resolvers in the wild, we calculate the value of $S$ (Eq. 1). As RIPE Atlas probes are not resolvers themselves, they do not appear in the traffic logs and there is no direct way to merge the datasets. Instead, we choose to combine all $(T, L)$ and $r_T$ values from both datasets to produce a collection of simulated resolvers based upon our real world measurements. These simulated resolvers cover a wide range of situations for resolvers, including situations encountered by real-world resolvers and situations not at present encountered by any real-world resolvers, while also missing some situations that real-world resolvers may encounter. Figure 11 in the lines "wgt RTT - R" and "avg RTT - R" shows CDFs of the speedup using the weighted and average RTT, respectively. Between 47% (448M) using the weighted RTT and 64% (609M) using the average

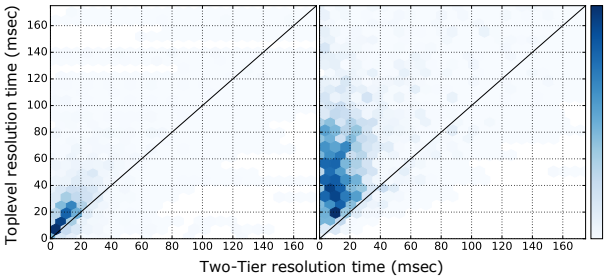


**Figure 12: Computed resolution time per query from simulated resolvers to toplevels ($Y$-axis) and Two-Tier ($X$-axis) using average (right) and weighted (left) RTTs. Tint represents a linear scaling on the number of simulated resolvers within a hexbin.**

RTT experience reduced average resolution time with Two-Tier, i.e., $S > 1$. Due to the skew in DNS queries among resolvers, those 47-64% resolvers account for 87-98% of all DNS queries, as shown in the lines "wgt RTT - Q" and "avg RTT - Q". Since $S$ is a ratio, we also plot the absolute resolution times in Figure 12 for the "wgt RTT - Q" (left) and "avg RTT - Q" (right). The $Y$-axis is the numerator in Eq. 1 while the $X$-axis is the denominator. Thus, Two-Tier reduces resolution time compared to toplevels for points above the diagonal. For both "wgt RTT - Q" and "avg RTT - Q", the average Two-Tier resolution time is roughly 16 msec. The average toplevel resolution time is 27 and 61 msec in "wgt RTT - Q" and "avg RTT - Q", respectively. Thus, we conclude that Two-Tier can reduce resolution time in most situations over Akamai's single-tier of toplevels.

**Improvements:** Our results show that there is a cost for some resolvers, however, and particularly those that weight delegation selection or have low DNS query volumes. Clearly, the cost is incurred each time the resolver must query both the toplevels and the lowlevels. If the DNS response from the toplevels could, in addition delegating to lowlevels, push an answer so that the resolver need not query the lowlevels in the same resolution, then Two-Tier would always be beneficial when the lowlevel RTT is less than the toplevel RTT, which is the case for 87-98% of the simulated resolvers. Pushing answers requires a modification to the DNS protocol. However, server push is a feature in recently standardized DNS-over-HTTPS [19].

## 6 RELATED WORK

Since DNS was conceived during the Internet's early stages [31], it has been extensively studied, resulting in numerous RFCs [1], as well as a vast array of academic work. DNS lies in the intersection of various fields such as security and privacy [21, 38, 47], BGP and anycast [7, 15, 30, 43], resiliency against malicious attacks [32, 51], and DNS-based traffic load balancing and CDNs [50]. In terms of systemic analysis and measurement studies, prior work has extensively explored the behaviors and interactions of end-users and their resolvers [2, 8, 17, 26, 48, 56]. In comparison, authoritative DNS infrastructures have not been studied in as much depth, with the exception of the root nameservers [9, 28, 53]. We focus on the design and operation of one of the largest authoritative DNS infrastructures in the world, Akamai DNS.

Several elements of Akamai DNS and how it is used by the Akamai CDN have been studied before. In [36], the authors present the Akamai CDN and how Akamai DNS answers DNS queries for the CDN, including a high level description of the Two-Tier delegation system (§5.2). In this paper, we demonstrate the effectiveness of Two-Tier. In [11], the authors demonstrate an extension of the Mapping Intelligence component and Akamai DNS to support end-user mapping using the edns-client-subnet (ECS) EDNS0 option. This work presents a use of Akamai DNS, while we present the Akamai DNS infrastructure in detail. Finally, the overlay multicast network that Akamai DNS uses for near real-time delivery of certain critical metadata is similar to that discussed in [4, 25]. Akamai DNS is a consumer of these delivery services, so we do not discuss it here.

## 7 CONCLUDING REMARKS

This paper presents design principles and experiential insights gleaned over two decades of architecting, deploying, and operating Akamai DNS, a critical component of the Internet infrastructure. We show how Akamai DNS is designed to provide resiliency, scalability, performance, and reconfigurability. We describe a taxonomy of failure modes and attack scenarios, and the mechanisms designed to mitigate them. As DNS query volumes increase rapidly and attacks on DNS become more sophisticated, the Akamai DNS architecture provides a flexible platform to build more capabilities to meet future challenges.

We now summarize the key design principles that underlie the architecture of Akamai DNS: (*i*) Avoid single points of failure (§4.3.1); (*ii*) Use general mitigation strategies for failure modes rather than specific point solutions, as such strategies potentially also cover unanticipated failure modes (§4.2), (*iii*) Under widespread failure, continue to operate in a degraded state as the alternative is not operating at all (§4.2.1), (*iv*) Build in contingencies for even extremely unlikely but high impact scenarios, so that Akamai DNS is always available (§4.2.3, §4.2.4), (*v*) Avoid actively reacting to an attack – instead rely upon automated mitigations – until action becomes absolutely necessary (§4.3.2).

We also highlight the following areas of future work for the research community. Mechanisms for automating anycast traffic engineering (§4.3.2) and the methods for information sharing between network peers to enable those mechanisms is an important area of work. Similarly, methods for predicting anycast routing or improving BGP route selection would greatly advance anycast performance (§5.1). Further, we believe there remain opportunities to improve the DNS protocol (§5.2), adding features to provide faster answers to the world's queries.

## ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers and our shepherd for their insightful comments that helped improve this paper. We would like to also thank Jean Roy, Larry Campbell, Brian Sniffen, and Joshua Matt for providing valuable feedback on early drafts of this paper. Finally, we thank the numerous engineers at Akamai who contributed to building Akamai DNS into the impressive system that it is today.

# REFERENCES


[1] 2020. DNS Camel Viewer. (2020). https://powerdns.org/dns-camel/

[2] Bernhard Ager, Wolfgang Mühlbauer, Georgios Smaragdakis, and Steve Uhlig. 2010. Comparing DNS resolvers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. 15–21.

[3] Akamai. 2019. EdgeScape. (2019). Retrieved December 2019 from https://developer.akamai.com/edgescape

[4] Konstantin Andreev, Bruce M Maggs, Adam Meyerson, and Ramesh K Sitaraman. 2003. Designing overlay multicast networks for streaming. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*. ACM, 149–158.

[5] Vasco Asturiano. 2011. The Shape of a BGP Update. (2011). Retrieved January 2020 from https://labs.ripe.net/Members/vastur/the-shape-of-a-bgp-update

[6] Chris Baker. 2016. Dyn, DDoS, and the DNS. (2016).

[7] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. 2015. Analyzing the Performance of an Anycast CDN. In *Proceedings of the 2015 Internet Measurement Conference*. 531–537.

[8] Thomas Callahan, Mark Allman, and Michael Rabinovich. 2013. On modern DNS behavior and properties. *ACM SIGCOMM Computer Communication Review* 43, 3 (2013), 7–15.

[9] Sebastian Castro, Duane Wessels, Marina Fomenkov, and Kimberly Claffy. 2008. A day at the root of the internet. *ACM SIGCOMM Computer Communication Review* 38, 5 (2008), 41–46.

[10] R. Chandra, P. Traina, and T. Li. 1996. *BGP Communities Attribute.* RFC 1997. https://tools.ietf.org/html/rfc1997

[11] Fangfei Chen, Ramesh K Sitaraman, and Marcelo Torres. 2015. End-user mapping: Next generation request routing for content delivery. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 167–181.

[12] Cloudflare. 2019. Cloudflare 1.1.1.1 Public Recursive Resolver. (2019). Retrieved June 2019 from https://1.1.1.1/

[13] C. Contavalli, W. van der Gaast, D. Lawrence, and W. Kumari. 2016. *Client Subnet in DNS Queries.* RFC 7871. https://tools.ietf.org/html/rfc7871

[14] Jakub Czyz, Michael Kallitsis, Manaf Gharaibeh, Christos Papadopoulos, Michael Bailey, and Manish Karir. 2014. Taming the 800 pound gorilla: The rise and decline of NTP DDoS attacks. In *Proceedings of the 2014 Internet Measurement Conference*. ACM, 435–448.

[15] Ricardo de Oliveira Schmidt, John Heidemann, and Jan Harm Kuipers. 2017. Anycast latency: How many sites are enough?. In *International Conference on Passive and Active Network Measurement*. Springer, 188–200.

[16] Wouter B De Vries, Ricardo de O Schmidt, Wes Hardaker, John Heidemann, Pieter-Tjerk de Boer, and Aiko Pras. 2017. Broad and Load-Aware Anycast Mapping with Verfploeter. In *ACM Internet Measurement Conference*.

[17] Hongyu Gao, Vinod Yegneswaran, Yan Chen, Phillip Porras, Shalini Ghosh, Jian Jiang, and Haixin Duan. 2013. An empirical reexamination of global DNS behavior. In *ACM SIGCOMM Computer Communication Review*, Vol. 43. ACM, 267–278.

[18] Google. 2019. Google Public DNS. (2019). Retrieved June 2019 from https://developers.google.com/speed/public-dns/

[19] P. Hoffman and P. McManus. 2018. *DNS Queries over HTTPS (DoH).* RFC 8484. https://tools.ietf.org/html/rfc8484

[20] C. Hopps. 2000. *Analysis of an Equal-Cost Multi-Path Algorithm.* RFC 2992. https://tools.ietf.org/html/rfc2992

[21] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman. 2016. *Specification for DNS over Transport Layer Security (TLS).* RFC 7858. https://tools.ietf.org/html/rfc7858

[22] Cheng Jin, Haining Wang, and Kang G Shin. 2003. Hop-count filtering: an effective defense against spoofed DDoS traffic. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*. ACM, 30–41.

[23] Georgios Kambourakis, Tassos Moschos, Dimitris Geneiatakis, and Stefanos Gritzalis. 2007. Detecting DNS amplification attacks. In *International Workshop on Critical Information Infrastructures Security*. Springer, 185–196.

[24] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. 2017. DDoS in the IoT: Mirai and other botnets. *Computer* 50, 7 (2017), 80–84.

[25] Leonidas Kontothanassis, Ramesh Sitaraman, Joel Wein, Duke Hong, Robert Kleinberg, Brian Mancuso, David Shaw, and Daniel Stodolsky. 2004. A transport layer for live streaming in a content delivery network. *Proc. IEEE* 92, 9 (2004), 1408–1419.

[26] Marc Kührer, Thomas Hupperich, Jonas Bushart, Christian Rossow, and Thorsten Holz. 2015. Going wild: Large-scale classification of open DNS resolvers. In *Proceedings of the 2015 Internet Measurement Conference*. 355–368.

[27] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. 2000. Delayed Internet routing convergence. *ACM SIGCOMM Computer Communication Review* 30, 4 (2000), 175–187.

[28] Bu-Sung Lee, Yu Shyang Tan, Yuji Sekiya, Atsushi Narishige, and Susumu Date. 2010. Availability and Effectiveness of Root DNS servers: A long term study. In *2010 IEEE Network Operations and Management Symposium-NOMS 2010*. IEEE, 862–865.

[29] E. Lewis and Ed. A. Hoenes. 2010. *DNS Zone Transfer Protocol (AXFR).* RFC 5936. https://tools.ietf.org/html/rfc5936

[30] Stephen McQuistin, Sree Priyanka Uppu, and Marcel Flores. 2019. Taming Anycast in the Wild Internet. In *Proceedings of the Internet Measurement Conference*. 165–178.

[31] P. Mockapetris. 1987. *Domain names - implementation and specification.* STD 13. https://tools.ietf.org/html/rfc1035

[32] Giovane Moura, John Heidemann, Moritz Müller, Ricardo de O Schmidt, and Marco Davids. 2018. When the Dike Breaks: Dissecting DNS Defenses During DDoS. In *Proceedings of the Internet Measurement Conference 2018*. ACM, 8–21.

[33] Giovane Moura, Ricardo de O Schmidt, John Heidemann, Wouter B de Vries, Moritz Muller, Lan Wei, and Cristian Hesselman. 2016. Anycast vs. DDoS: Evaluating the November 2015 root DNS event. In *Proceedings of the 2016 Internet Measurement Conference*. ACM, 255–270.

[34] Moritz Müller, Giovane Moura, Ricardo de O Schmidt, and John Heidemann. 2017. Recursives in the wild: engineering authoritative DNS servers. In *Proceedings of the 2017 Internet Measurement Conference*. ACM, 489–495.

[35] Marcin Nawrocki, Jeremias Blendin, Christoph Dietzel, Thomas C Schmidt, and Matthias Wählisch. 2019. Down the Black Hole: Dismantling Operational Practices of BGP Blackholing at IXPs. In *Proceedings of the Internet Measurement Conference*. ACM, 435–448.

[36] Erik Nygren, Ramesh K Sitaraman, and Jennifer Sun. 2010. The Akamai Network: A Platform for High-Performance Internet Applications. *ACM SIGOPS Operating Systems Review* 44, 3 (2010), 2–19.

[37] Jeffrey Pang, Aditya Akella, Anees Shaikh, Balachander Krishnamurthy, and Srinivasan Seshan. 2004. On The Responsiveness of DNS-Based Network Control. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*. 21–26.

[38] Jeman Park, Aminollah Khormali, Manar Mohaisen, and Aziz Mohaisen. 2019. Where Are You Taking Me? Behavioral Analysis of Open DNS Resolvers. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 493–504.

[39] Quad9. 2019. Quad9 DNS Service. (2019). Retrieved June 2019 from https://www.quad9.net/

[40] Yakov Rekhter, Susan Hares, and Tony Li. 2006. A Border Gateway Protocol 4 (BGP-4). RFC 4271. (Jan. 2006). https://doi.org/10.17487/RFC4271

[41] RIPE. 2019. Atlas. (2019). Retrieved January 2020 from https://atlas.ripe.net/

[42] RIPE. 2019. Atlas API v2 manual: Base Attributes. (2019). Retrieved June 2020 from https://atlas.ripe.net/docs/api/v2/manual/measurements/types/base_attributes.html

[43] Sandeep Sarat, Vasileios Pappas, and Andreas Terzis. 2006. On The Use of Anycast in DNS. In *Proceedings of 15th International Conference on Computer Communications and Networks*. IEEE, 71–78.

[44] Kyle Schomp. 2019. DNS Recursive Resolver Delegation Selection in the Wild. (2019). Retrieved May 2019 from https://indico.dns-oarc.net/event/31/contributions/676/

[45] Kyle Schomp, Mark Allman, and Michael Rabinovich. 2014. DNS resolvers considered harmful. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*. ACM, 16.

[46] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. 2013. On Measuring the Client-side DNS Infrastructure. In *Proceedings of the 2013 Conference on Internet Measurement (IMC '13)*. ACM, New York, NY, USA, 77–90.

[47] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. 2014. Assessing DNS Vulnerability to Record Injection. In *International Conference on Passive and Active Network Measurement*. Springer, 214–223.

[48] Kyle Schomp, Michael Rabinovich, and Mark Allman. 2016. Towards a model of DNS client behavior. In *International Conference on Passive and Active Network Measurement*. Springer, 263–275.

[49] Pavlos Sermpezis and Vasileios Kotronis. 2019. Inferring Catchment in Internet Routing. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3, 2 (2019), 30.

[50] Anees Shaikh, Renu Tewari, and Mukesh Agrawal. 2001. On The Effectiveness Of DNS-Based Server Selection. In *Proceedings of IEEE INFOCOM 2001*, Vol. 3. IEEE, 1801–1810.

[51] Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. 2014. DNSSEC and Its Potential for DDoS Attacks: A Comprehensive Measurement Study. In *Proceedings of the 2014 Conference on Internet Measurement (IMC '14)*. ACM, New York, NY, USA, 449–460. https://doi.org/10.1145/2663716.2663731

[52] Ralf Weber. 2014. Latest Internet Plague: Random Subdomain Attacks. (2014). Retrieved May 2019 from https://indico.uknof.org.uk/event/31/contributions/349/

[53] Duane Wessels. 2019. Long Term Analysis of Root Server System Performance Using RIPE Atlas Data. (2019). Retrieved Nov 2019 from https://indico.dns-oarc.net/event/32/contributions/713/

[54] Florian Wohlfart, Nikolaos Chatzis, Caglar Dabanoglu, Georg Carle, and Walter Willinger. 2018. Leveraging interconnections for performance: the serving infrastructure of a large CDN. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 206–220.

[55] Bill Woodcock. 2016. Best Practices in DNS Service-Provision Architecture. In *ICANN 55*. ICANN.
[56] Yingdi Yu, Duane Wessels, Matt Larson, and Lixia Zhang. 2012. Authority server selection in DNS caching resolvers. *ACM SIGCOMM Computer Communication Review* 42, 2 (2012), 80–86.