Measuring Update Performance and Consistency Anomalies in Managed DNS Services

Zhaoyu Gao, Arun Venkataramani
College of Information and Computer Sciences, University of Massachusetts Amherst
{gaozy, arun}@cs.umass.edu

***Abstract*—Managed DNS (MDNS) services today excel at providing a simple and cost-effective way to outsource domain management and ensure rapid lookup times for geo-distributed users. The intense focus on optimizing lookup performance coupled with DNS' inherent expectations of weak consistency has had unfortunate side effects: updates are inexplicably slow and MDNS providers pay scant attention to consistency correctness. We conduct an empirical measurement-driven study of 8 top-tier managed DNS providers and find that inter-nameserver update propagation delays commonly take tens of seconds with little improvement over the last several years. Client-perceived inconsistency is rampant with roughly a third of end-users being vulnerable to TTL abuse by local DNS resolvers. Furthermore, we find that 6 of the 8 MDNS providers violate monotonic read consistency under frequent updates and at least one large MDNS provider appears to violate even eventual consistency.**

I. Introduction

The Internet's Domain Name System (DNS) is widely presented as a textbook example of a distributed system with eventual consistency, a system that effectively trades off consistency to improve performance and reduce cost. DNS achieves this design goal through its liberal use of TTL-based caching, a design that also implicitly embeds the synergistic assumption of infrequent updates. Weakly consistent caching alongside hierarchical federation is key to DNS' scalability.

The expectation of weak consistency coupled with that of infrequent updates in DNS has had some unintended consequences. For one, managed DNS (MDNS) service providers have invested enormously in optimizing lookup performance, e.g., through highly geo-distributed anycast clouds, down to milliseconds but update performance has been relegated to second-class status, an artifact we confirmed through our measurement as well as by interviewing technical personnel from a number of top MDNS companies (refer §II). Furthermore, observed update delays don't always match advertised values, e.g., a leading MDNS provider claims to propagate an update within 5 seconds [1], but it takes nearly 30 seconds in practice (refer §III). There is reduced attention if at all paid to ensuring consistency *correctness* across replicated authoritative nameservers of MDNS providers.

Our position is that stronger consistency in DNS is important for existing and emerging application scenarios for several reasons. First, inconsistency implies service unavailability when service replicas are migrated or taken down as DNS update propagation can take hours or even several days [2], [3], [4], [5]. Second, inconsistency implies reduced agility of load balancing, especially but not only in the context of CDNs, as it causes reduced control over traffic redirection and reduced responsiveness of elastic provisioning to load spikes. Third, in today's mobile world, enabling any-to-any communication with seamless endpoint mobility (as opposed to communication only initiated by mobiles) requires a consistent global name service in order to ensure mobile reachability. We are hardly the first to make these and other arguments for stronger consistency amidst frequent updates in DNS [6], [7], [8], [9], [10], [11].

In this paper, we seek to empirically determine answers to the following high-level questions: (1) How does DNS update performance in MDNS providers compare to that of lookups? and (2) How strong or weak are the consistency semantics achieved by MDNS providers and, in particular, if they at least satisfy even eventual consistency? To this end, we conduct a large-scale distributed measurement study involving 8 top MDNS providers using 40 PlanetLab nodes and over 1,000 RIPE vantage points to measure update propagation delays and as-is or *existential consistency* [12] amidst frequent updates.

Our main findings are summarized as follows:

1) Inter-nameserver update propagation delays across authoritative nameservers of MDNS providers are inordinately high compared to network propagation delays, e.g., Google Cloud DNS and Route53 incur update propagation delays of over or close to 30 seconds on average and occasionally over a minute (§III-A).
2) End-to-end user-perceived update propagation delays are far worse because many local DNS resolvers do *not* respect TTL values, presumably to reduce DNS load on their networks, and roughly a third of end-users are impacted by such abusive resolvers (§III-B).
3) Update propagation can result in anomalous observations causing older values to be observed after a more recent value by an end user; in particular, 6 of the 8 MDNS providers violate *monotonic reads* consistency under frequent updates (§IV-B1).
4) At least one major MDNS provider, GoDaddy, likely violates even eventual consistency, a claim that cannot be verified with certainty without access to proprietary internal details (§IV-B2).

II. Managed DNS Background

Outsourcing to an MDNS provider is the predominant way for people or businesses to manage their domain names.

MDNS providers ease the task of managing domain names by offering their customers the promise of high performance and availability in a cost-effective manner by leveraging economies of scale. To ensure high performance and availability, MDNS providers typically maintain a geo-distributed anycast cloud of authoritative nameservers that store replicated name records on behalf of their customers. As with any software-as-a-service segment, MDNS vendors offer a variety of pricing plans that may limit the number of queries (e.g., to 500K/month in one plan offered by NS1); allow unlimited queries (e.g., CloudFlare or GoDaddy); adopt a pay-as-you-go model (e.g., Route53 or Google Cloud DNS); or offer package plans with different levels of geo-replication, SLAs, and query limits (like those offered by Verisign, Dyn, DNSMadeEasy and others). The cost to the customer could be rolled in for "free" with other hosting packages as is common for individuals or small businesses and range anywhere from a few dollars to hundreds of dollars or more per month for large customers.

A. MDNS state-of-the-art architecture

MDNS services are organized as a distributed system as shown in Fig. 1 and described next.

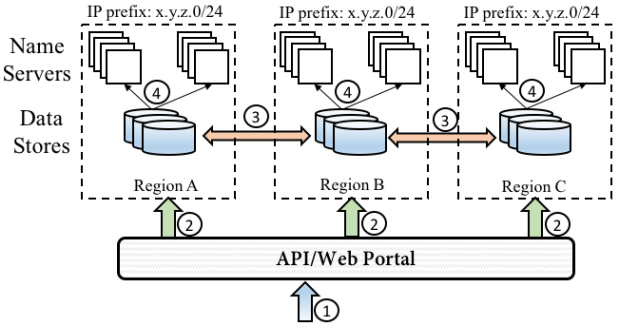


Fig. 1. Architecture of MDNS services

Web portal and API: MDNS customers use a web or API portal to manage their domains, e.g., create or update resource records for their domains. In our measurements, we use their APIs to issue updates as it allows us to interact with MDNS services programmatically. An alternative is to use the standards-based Dynamic DNS protocol [13] but that is not supported by all MDNS providers (e.g., GoDaddy). Some MDNS API servers are geo-distributed while others are not. We infer the location of these MDNS API servers by pinging their endpoint domain names (shown in Table I) using a global ping test service [14]; Table I shows these inferred locations. Although some MDNS providers, e.g., Verisign, disable echoing ICMP packets on their portal servers, but we can still infer the API portal location through GeoIP [15].

Replicated storage: Upon receiving an update request, the portal server forwards the update to an underlying persistent data store that is replicated across some or all of the MDNS' authoritative nameservers. The replicated server store is responsible for maintaining consistency, and must at least ensure eventual consistency as expected of DNS [16].

Name servers: Nameservers answer DNS queries from clients. Each domain must have at least two nameservers for fault-tolerance purpose [16] that maintain authoritative or primary copies of a name record. If an MDNS nameserver caches answers, the underlying replicated store must invalidate the cached answers upon updates. Some providers (like NS1) adopt a best practice of never caching answers on authoritative name servers, which also mitigates multi-level caching that is discouraged by DNS [6].

Anycast network: All of the MDNS providers we interviewed use an anycast network. They announce an anycast IP prefix from their different regional datacenters causing end-user traffic to be directed to the nearest (in network distance) datacenter, which helps improve user-perceived response times as well as mitigate DDoS attacks via route diversion [17]. As shown in Table I, Cloudflare has the most number (119) of IP anycast locations. Most MDNS providers have more than 10 IP anycast locations, and it is common to gradually add new IP anycast sites to their infrastructure in order to improve their overall lookup performance.

B. MDNS update delays and consistency

Given our high-level goal of a measurement-driven study of update performance and consistency in state-of-the-art MDNS services, we picked the 8 top MDNS providers (as ranked by Datanyze [18] in May 2018) as listed in Table I. Datanyze reports the cumulative market share of these companies at 80.1% and they host 4263 out of Alexa top-10K domains [18]. We started this study by interviewing their sales or technical team representatives as appropriate and asked them the same scripted set of questions seeking to understand their system architecture, identify their most important market differentiating features, advertised lookup and update performance, and consistency amidst frequent updates. The responses across the board indicated that they prioritized lookup performance, availability, and security (e.g., resilience against DDoS attacks but not necessarily DNSSEC that is not universally supported).

On the other hand, update performance or consistency were either secondary or non-goals. Some providers appeared unclear on their own update performance (quite unlike lookup performance that is advertised as a competitive advantage), e.g., CloudFlare claims that they can propagate an update within 5 seconds [1], but our measurements indicate much higher typical values for them as well as other providers. It was also clear from the responses that the providers had given little attention to consistency semantics, and it wasn't obvious whether they even guaranteed eventual consistency.

Our position is that existing and emerging application scenarios can significantly benefit from improved update performance and stronger consistency semantics. Dynamically scaled cloud applications [19] as well as the agile edge computing services [20], [21], [22] rely on frequent DNS updates. For example, a cloud application can reduce its cost by using low-cost resources (such as AWS spot instances [23] or Google preemptible VMs [24]), however such resources are also typically short-lived and must be dynamically provisioned,

MDNS	Number of anycast locations	Minimal TTL	Portal endpoint	API portal locations	Number of authoritative nameservers
Cloudflare	119	120	api.cloudflare.com	Global	2
AWS Route53	38	0	route53.amazonaws.com	US East	4
Dyn	20	1	api2.dynect.net	US West	4
Google Cloud DNS	15	0	www.googleapis.com	Global	4
GoDaddy	9	600	api.godaddy.com	Global	2
DNSMadeEasy	16	5	api.dnsmadeeasy.com	US East	6
NS1	25	0	api.nsone.net	Global	4
Verisign	17	0	mdns.verisign.com	US East	3

TABLE I
TOP MDNS PROVIDERS STUDIED IN THIS WORK

which necessitates frequent DNS updates. As another example, consider an edge computing platform [20] that requires frequent DNS updates to reposition edge services close to their mobile end-users or terminate some services due to resource constraints at edge servers. High user mobility is commonplace today (e.g., 20% of users change IP addresses over 10 times a day [9]), and slow update propagation or inconsistency can respectively render mobile endpoints temporarily or even permanently unreachable. These concerns motivate us to conduct a measurement-driven study of update performance and server- and client-centric consistency in today's MDNS providers.

III. MDNS UPDATE PROPAGATION DELAY

Expectation. Given the high-level architecture in Fig. 1, we expect the update delay to be no more than a few seconds. This estimate is derived by dividing the overall delay into the delay: 1) from an API client to the portal, 2) from the portal to replicated storage, 3) to propagate the update among replicated storage, 4) to refresh answers at name servers. The third one dominates the overall delay as the rest are all expected to happen within a single region. Any reasonable replica coordination protocol, including consensus-based protocols ensuring stronger consistency semantics [25], [26], should take no more than two expected wide-area round-trip times (but may be somewhat inflated by variance across a large number of geo-distributed replicas). Update propagation delays significantly exceeding this expectation likely imply unnecessary overheads imposed by the MDNS provider.

Measurement setup. To conduct this study, we purchased DNS service from all these providers for separate domain names. We use 40 PlanetLab nodes (across Europe, America, and Asia-Pacific) for active probing so that we have full control on when to start and to stop a measurement. An alternative would have been to use other platforms with more vantage nodes like DNSPerf [27], RIPE Atlas [28], and Catchpoint [29] to monitor DNS lookup performance, these platforms do not allow active probing, i.e., immediately start and stop probing contingent on specified conditions. Moreover, 40 PlanetLab nodes are enough to cover most of the IP anycast locations of these major MDNS providers (except Cloudflare).

Before delving into updates, we first measure the distribution of lookup latency of the 8 MDNS providers as shown in Fig. 2. For each service provider, we measure 500 lookup latencies from across 40 PlanetLab nodes (over 20K data points for each service provider). We ensure that lookups are served from the name servers maintained by its provider by directly querying the name servers via their external IP anycast addresses. We apply a 1-second trim filter to remove outliers, which removed 0.2% of the points. Fig. 2 shows that median lookup latency of all the providers is within 50ms. This result is consistent with the result of state-of-the-art DNS lookup performance measurement platforms such as DNSPerf [27] and Catchpoint [29], and shows that the measurements in this paper are representative of those with other probe platforms. It also shows that our probe nodes can resolve the domain with these MDNS providers very efficiently, introducing a negligible performance overhead to all of our measurements.

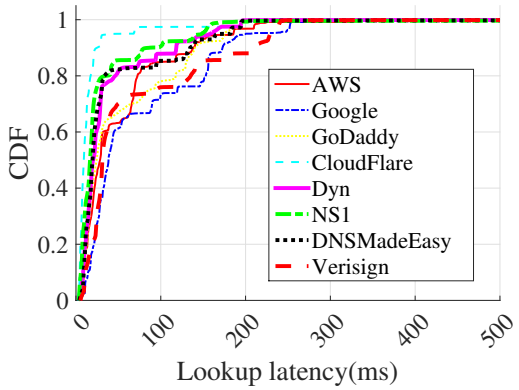


Fig. 2. Lookup latency distribution

A. Update propagation delay across MDNS authoritative nameservers

To measure the update propagation latency among authoritative name servers, we use the 40 PlanetLab nodes to send lookup queries for a name to its provider's nameservers directly. Each PlanetLab node sends back-to-back queries to all authoritative name servers and stops when all of them return the updated address. Immediately after starting the lookup process on all PlanetLab nodes, we issue an address update request to the DNS provider through its API. All update requests in the measurements of this work are sent from a host on the US east coast, and all update requests return

within a second for all these MDNS providers. We measure the propagation latency of 100 updates for each MDNS provider.

Fig. 3 shows the update propagation delay of all these MDNS providers. Three MDNS services, i.e., Dyn, NS1, and DNSMadeEasy, can quickly propagate an issued update within 5 seconds. The other MDNS providers take an order of magnitude longer than network round-trip time, e.g., Google Cloud DNS takes close to 50 seconds on average to propagate an update. Curiously, the median of update latency of Route53 increased to 26.7s from 7.9s as measured by CloudHarmony in 2012 [30]. One explanation is that both measurements just lower-bound the update latency, and it is possible that the 2012 experiment happened to not observe higher values. We also found that our measured update propagation latency for Cloudflare is *inconsistent* with their claim of 5 seconds [1].

By inspecting the time of the first PlanetLab node that gets the most recent update, the MDNS providers with large update delay appear to wait unnecessarily for a period of time before propagating the update, e.g., 28.5s for Route53, 36.1s for Google Cloud DNS, 16.5s for Cloudflare (on average). A plausible explanation for the delays could be message queuing overhead incurred in their systems or even a deliberate hold to leverage batching, but that seems unlikely given the infrequent nature of updates. Caching records for a fixed duration could be another explanation. Unlike these MDNS systems, the three with a small update delay all start propagating immediately (within a second) upon receiving the update.

To ascertain if there are specific laggard regions inflating the update propagation delay, a question of interest also to several MDNS network operators with whom we shared our measurement results, we inspect the region that first and last completes the update; the overall update propagation delay is in part determined by the intervening interval. We classify all the PlanetLab nodes we use into five continental regions, i.e., North America(NA), South America(SA), Oceania(OC), Europe(EU), and Asia(AS). Table II shows the number of rounds that a region first and last completes an issued update. Nodes in Asia are mostly the last to see the update. Almost all MDNS services start update propagation from America, which is expected as the update is issued from a US east coast host.

Can changing the location of the host issuing the update qualitatively change our observations? We believe that is unlikely. For MDNS providers like Route 53, Dyn, DNSMadeEasy, and Verisign, whose portals are not geo-distributed and are in N. America, changing the location of the east-coast host issuing the update is likely to modestly increase the overall update propagation delay because of the likely increase in delay from the client to the portal (step 1 in Fig. 1). For MDNS providers like Google, Cloudflare, NS1, and GoDaddy whose portals are geo-distributed, neither step 1 nor step 2 is likely to change significantly, implicitly assuming here that inter-nameserver propagation delays are not sensitive to which nameserver first receives the update. Thus, the choice of the host issuing the update is unlikely to explain away the tens of seconds of observed update propagation delays.

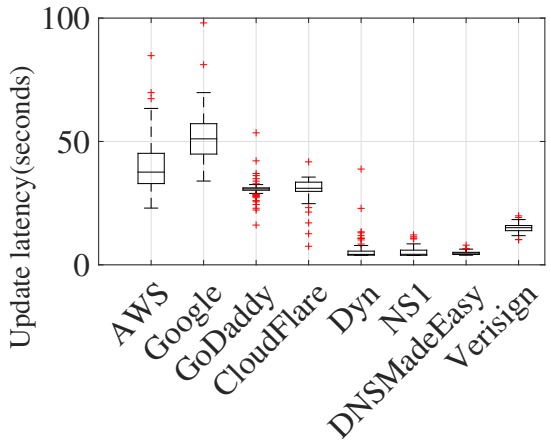


Fig. 3. Update propagation latency among authoritative name servers: the vertical lines of the whiskers and box from top to bottom mean 100th, 75th, 50th, 25th, 0th percentiles, respectively. Symbol '+' means outliers.

B. User-perceived update propagation delay

Some ISP local DNS resolvers and open DNS resolvers are anecdotally known to not respect DNS TTLs, caching a record for much longer [31], presumably to shed load or perhaps to mitigate DoS attacks [32], thereby causing end-users to see stale answers for unexpectedly long. We next measure the fraction of end-users affected by such illegitimate caching.

Experiment setup. We use 1,000 RIPE Atlas probe nodes [28] (aka "probes") to resolve a domain name we own. RIPE Atlas is an open measurement platform operated by RIPE NCC consisting of over 10K probe nodes [28]. The platform only permits launching 1,000 probes at a time. We got probing results from 937 out of the selected 1,000. These 937 probes belong to 925 ASNs and 937 network prefixes respectively. We force all these probes to perform a resolution by using their local DNS resolvers, i.e., a DNS query will always hit a local DNS resolver first and then go to an authoritative name server if the record is not cached. We set the probing interval to 1 minute and set the TTL of our record to 30 seconds. In this way, a cached answer on a local DNS resolver must have expired before the next query. We use NS1 to serve the target domain because it propagates updates relatievly quickly and has good lookup performance [27]. More importantly, we were able to confirm with NS1 that their authoritative name servers never cache query results, which helps narrow down the illegitimate caching problem to local DNS resolvers. We issue update requests every 1 minute, which gives enough time for the record to get propagated.

Fig. 4 exemplifies the probing process. The update process issues updates every 60s. Suppose the update sequence is a_1, a_2, a_3. Probe 1 truthfully resolves the name and evicts the cache after 30s, the TTL value. So the probing sequence observed by probe 1 is also a_1, a_2, a_3. However, probe 2 illegitimately caches an answer for 180s, so its probing sequence is a_1, a_1, a_1. Note that the update process and probes are not exactly synchronized and the probe's clock may also drift. But RIPE Atlas' Message Queue Cluster mitigates the clock drift

MDNS	# of rounds that the region first complete an update					# of rounds the region last complete an update				
	NA	SA	EU	OC	AS	NA	SA	EU	OC	AS
Cloudflare	58	8	5	13	16	4	0	0	7	89
AWS Route53	33	22	10	15	20	29	14	12	2	43
Dyn	97	2	1	0	0	20	0	0	7	73
Google Cloud DNS	44	40	4	3	9	5	3	19	2	71
GoDaddy	100	0	0	0	0	43	0	0	1	56
DNSMadeEasy	99	1	0	0	0	8	0	0	28	64
NS1	43	51	1	3	2	86	0	1	1	12
Verisign	99	0	0	0	1	10	3	1	1	85

TABLE II
NUMBER OF ROUNDS THAT A REGION FIRST AND LAST COMPLETES AN UPDATE.

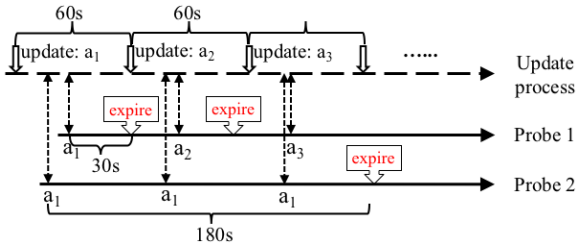


Fig. 4. Probing process with RIPE Atlas

problem among probes [33] and the parameters we choose ensure that probes detected as illegitimately caching an answer for over a minute are unlikely to be false positives.

We define `caching period` as the average number of intervals for each repeated answer. For example, if the probing sequence is $\{a_1, a_2, a_2, a_3, a_3, a_3, a_4, a_4\}$, wherein each a_i is an IP address, then its caching period is (1+2+3+2)/4 =2. Therefore, 1 means a local DNS resolver never abuses TTL, and a larger value implies a longer illegitimate caching period.

Fig. 5 shows the CDF across 937 probes for the caching period. The caching period of about two-third probes is 1, which means these probes always returns an up-to-date value. Over 5% probes have a caching period greater than 2, implying that 5% DNS resolvers may cache an expired result for over 2 minutes on average. The longest caching period observed in the experiment is close to 30 minutes, i.e., some DNS resolvers cache an answer illegitimately for nearly half an hour.

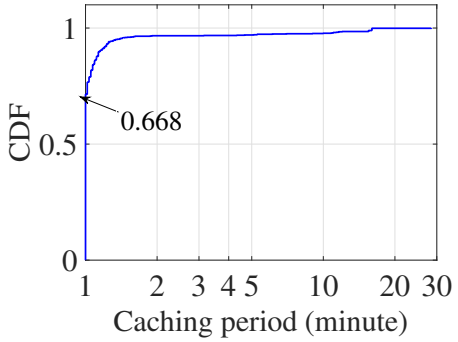


Fig. 5. CDF of caching period

IV. MDNS CONSISTENCY ANOMALIES

We next empirically measure the consistency semantics (or lack thereof) satisfied by our MDNS providers.

Expectation. DNS strictly speaking is expected to satisfy only *eventual consistency*, i.e., if no updates take place for a long time, all replicas will eventually reach the same state. As such, this server-centric consistency property does not necessarily constrain client-perceived consistency. For example, an end-client may reasonably expect to not see an older value after it has observed a more recently updated value, but DNS may violate this property, especially when multiple updates are issued concurrently or in close succession. We refer to as an *update anomaly* a user-observed event representing a violation of expected consistency semantics [12].

A. Consistency models

Eventual consistency as defined is not a refutable property in closed systems, i.e., verifying whether or not a system is eventually consistent via external observations alone is not possible. Any blackbox system can trivially claim to be eventually consistent despite the passage of an arbitrarily long but *finite* duration of no updates and inconsistent replica state. Nevertheless, a dispassionate observer would after a reasonable duration, say a day, conclude that such a system is not eventually consistent. So we introduce δ-consistency, a property refutable via external observations.

δ-consistency: A server system is δ-consistent if all replicas converge to the same state after a duration δ during which no updates or failures occur. (Note that this definition may be slightly different from others in the literature that restrict the identical state to be the one reached after the most recent update for a suitable definition of *most recent*.) With our definition, δ-consistency is equivalent to eventual consistency as δ approaches infinity.

There is little reason for a commercial system to relax δ to very large values and every reason to ensure that it is small. We consider two δ values to be of particular interest: (1) $\delta = 1$ minute, given that update propagation was rarely observed to take over a minute in our measurements; and (2) $\delta = 24$ hours. Our implicit wager is that if a replica remains inconsistent with other replicas of a record for as long as 24 hours despite no new updates during that period, it will never reach the correct state, a correctness violation.

Monotonic reads is a *client-centric* consistency property meaning that if a client reads the value of a record, any successive read operation on the same record by that same client will always return that same value or a more recently written value [34]. A violation to monotonic reads happens mainly due to reconciliation when an MDNS system needs to resolve a write-write conflict as during the reconciliation, the value of a DNS record may change back and forth potentially violating monotonic read consistency.

MDNS services using reconciliation to enforce data consistency tend to have a higher update delay when multiple updates are issued in parallel. It is also notoriously hard for those relying on DNS to debug their applications if monotonic reads is violated. As an example, consider a sequence $\{a_1, a_2, a_1, a_2\}$ observed by a client after querying the domain for a few times, where each a_i is the effect of an update to the domain. There is no way for a client to definitively conclude which update is the most recent stable value, potentially rendering the domain unreachable. Update anomalies violating monotonic reads do happen with a non-negligible fraction in our measurement with some MDNS providers.

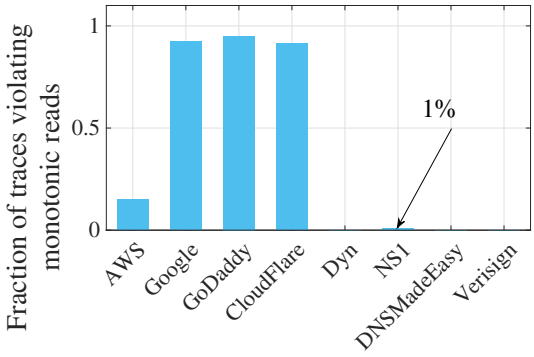


Fig. 6. Monotonic reads consistency with different MDNS providers

B. Measuring update anomalies

In this measurement, we measure both types of update anomalies, i.e., violation of monotonic reads and violation of δ-consistency, with the same set of PlanetLab nodes as §III. We first collect traces with PlanetLab nodes as follows in order to check for update anomalies.

Trace collection: In each round of the experiment, we issue exactly two updates near-simultaneously at the beginning. Then we start probing the record from all the PlanetLab nodes periodically. Each PlanetLab node sends DNS queries to all nameservers in parallel every second and stops probing after a minute. The number of nameservers in the NS record for each MDNS service is shown in Table I. We repeat the experiment for 100 rounds and collect the trace from all PlanetLab nodes. A trace is uniquely identified by a tuple of the round number, PlanetLab node hostname, and the target nameserver IP address, e.g., `(5, pl_node, 205.251.195.212)` denotes a trace in the 5th round collected on *pl_node* to a name server whose IP is `205.251.195.212`. Consequently, the total number of traces collected varies from 8,000 to 24,000 for different MDNS providers. All the results shown in this subsection are conducted with the traces collected in April 2018.

1) Violation of monotonic read: To measure violation of monotonic reads, we check each trace independently. Since we only issue updates at the beginning of each round, the only violation of monotonic reads that can happen is that an older value reappears in a trace, e.g., $\{(0, 2.2.2.2), (3, 1.1.1.1), (9, 2.2.2.2), \dots\}$ is a portion of a trace that violates monotonic reads where the first value in each tuple is the timestamp and the second is the IP address returned by a name server.

Fig. 6 shows the fraction of traces that violate monotonic read. The fraction is calculated as the number of traces that violate monotonic reads over the total number of traces. This fraction is 0 only for DNSMadeEasy and Dyn, i.e., we were unable to observe any violations of monotonic reads by them in our measurement. NS1 and Verisign both have about 1% traces that violate monotonic reads and the others have much higher fractions of violations. These MDNS providers may adopt a lazy reconciliation mechanism to resolve write-write conflicts, thereby violating monotonic reads.

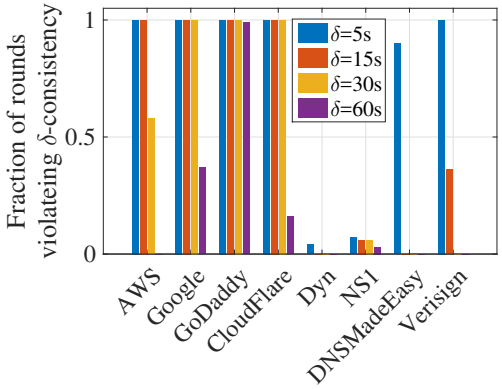
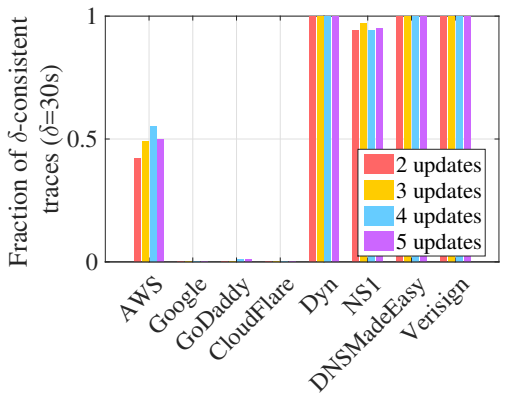


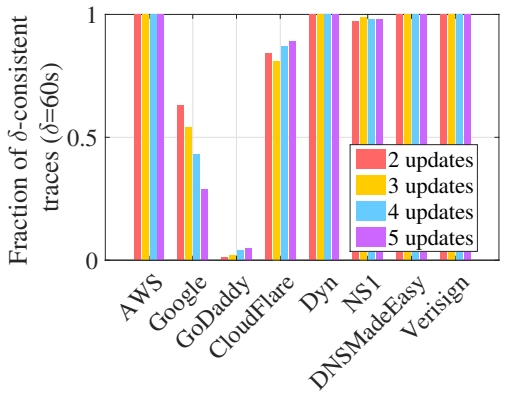
Fig. 7. δ-consistency violation with two concurrent updates

2) Violation of δ-consistency: We check δ-consistency with all the traces collected in each round as follows. First, the answer returned by a name server at time δ must be the same as the answer at the end of each trace, i.e., there is no ongoing update and the state has already converged to the final state. Second, the answers at time δ across all the replica traces must be the same. We also enforce that the final state only has a single IP address because that is how we update it.

Fig. 7 shows the fraction of rounds that violates δ-consistency when δ equals to different values. The fraction is calculated as the number of rounds that violate δ-consistency (given a certain δ) over the total number (100) of rounds. The fraction decreases gradually along with δ for all MDNS providers as expected. However, it also shows that there are still four MDNS providers including GoDaddy, Google Cloud DNS, Cloudflare, and NS1, whose fraction is still larger than 0



(a) δ-consistency with $\delta = 30s$



(b) δ-consistency with $\delta = 60s$

Fig. 8. Sensitivity analysis with different numbers of concurrent updates for δ-consistency measurement

when $\delta = 60s$, i.e., after a minute, they still haven't converged to a consistent state for many rounds of the experiment. That is, concurrently issued updates can markedly degrade the update propagation delay. For example, the 99th percentile update delay with NS1 is within 10 seconds as shown in Fig. 3. However after 30 seconds, there are still 9 rounds under reconciliation and 4 after 60 seconds. Interestingly, not every MDNS' update performance deteriorates with concurrent updates, for example, the performance of Google and Route53 is comparable to that with single updates in Fig. 3.

We also check the state of the updated record 24 hours after the end of the experiment. All MDNS providers correctly propagate our issued updates except GoDaddy, the only MDNS provider that violates the single-IP-address condition, i.e., it has more than 1 IP address for the target domain. The reason is that GoDaddy doesn't execute their customers' requests transactionally, so multiple write operations can be executed on a single record without isolation. The large number of domains hosted by GoDaddy, currently about 3 million domains (a number that has escalated dramatically since June 2017 [18]), are potentially vulnerable to this problem.

C. Sensitivity analysis

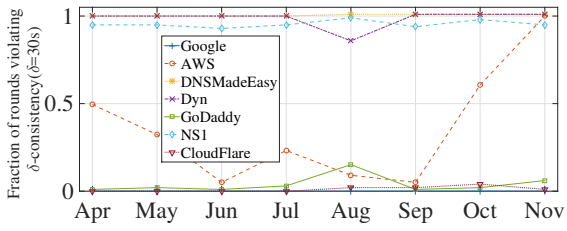
Our consistency anomaly measurement results may be potentially sensitive to different input workloads. For instance, it may take longer for an MDNS provider to reconcile write-write conflicts if more updates are issued concurrently. The performance of different MDNS providers may also fluctuate significantly due to the diverse workload they get over time, potentially affecting our result. To deal with these concerns, we conduct two experiments to measure: 1) the fraction of violations of δ-consistency with different numbers of concurrent updates 2) the fraction of violations of δ-consistency with the data collected at different times over a few months.

1) Result sensitivity to the number of concurrent updates: We did the following experiment to study the impact of different numbers of concurrent updates on our result. We rerun the experiment for 100 rounds with 3, 4, and 5 concurrent updates respectively. Some MDNS providers have an API update limitation on their customers, e.g., Dyn allows its customers to issue at most 5 API updates within a second [35], DNSMadeEasy allows its customers to issue 150 API requests within 5 minutes. To comply with the rules of these MDNS providers and compare them fairly, we issue at most 5 concurrent updates in this experiment. Note, our update rate is still quite negligible compared to the load of these MDNS providers received from their customers.

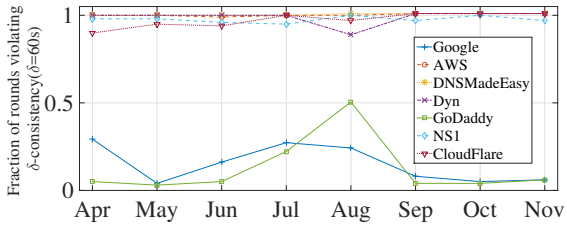
Fig. 8 (a) and (b) show the fraction of rounds that satisfy δ-consistency with different number of concurrent updates. It shows that different numbers of concurrent updates have little impact on the result with different δ values except for Google whose fraction decreases along with the number of concurrent updates as shown in Fig. 8 (b). For DNSMadeEasy and Dyn, since they do not rely on reconciliation to resolve write-write conflicts, therefore more concurrent updates do not affect their results. Verisign only executes 2 update requests no matter how many updates are issued simultaneously as we never observed more than 2 IP addresses shown up in its traces. Thus the result for Verisign does not get affected either. GoDaddy almost always violates δ-consistency within 60 seconds; thus with more concurrent updates, its behavior won't get affected as it can't be worse than 0. The other MDNS providers (Route 53, NS1, and Cloudflare) correctly execute all update requests, thus have a higher variation across the different numbers of updates as shown in Fig. 8. However, the maximum difference is still within 10%.

The reason that the fraction of δ-consistent rounds decreases for Google along with the number of concurrent updates is that their API does not support the `update` operation as of the date of our measurement [36]. Therefore, to update an existing record, we first have to delete it and then create a new one (or invoke the record update operation supported via the Google Cloud web console). Hence our updates to Google Cloud DNS are not atomic, which can potentially explain their small fraction of monotonic-read-consistent traces.

Overall, these results show that our δ-consistency measurements are insensitive to the number of concurrent updates



(a) $\delta = 30s$



(b) $\delta = 60s$

Fig. 9. Sensitivity analysis to data collection time with 5 concurrent updates.

for these MDNS providers except Google Cloud DNS. We conjecture that this claim would likely also be true for Google were they to support an atomic `update` API.

2) Result sensitivity to data collection time: Since all the traces used to show the consistency results above are collected in April 2018, in order to asses their sensitivity to the time of data collection, we continued to collect data on a monthly basis until November 2018. Note that the set of Planetlab nodes may vary from month to month, but as long as they are enough to cover most of the provider's name servers, the results are unlikely to be qualitatively different.

Fig. 9 shows the δ-consistency results for $\delta = 30s$ and $\delta = 60s$ over eight months. The values for AWS Route53 are not stable when $\delta = 30s$ due to the queueing delay in its system that is $\approx$ 30 seconds (refer §III). However, its value for $\delta = 60s$ is always close to 100%, i.e., it always converges to a consistent, correct state within 60 seconds. The results for other MDNS providers are very stable over the eight months when $\delta = 30s$ as shown in Fig. 9 (a). Fig. 9 (b) shows that the results for $\delta = 30s$ for Google and GoDaddy tend to fluctuate a lot due to the lack of transactional update support. We also checked our GoDaddy name record state well after 24 hours in these months. Our claim about GoDaddy likely violating eventual consistency still holds.

3) Result sensitivity to other factors: We believe that our findings are not sensitive to other factors such as the cost we incurred or the set of probe nodes. On the cost front, although the costs we incurred on different MDNS services are different, that is mainly because of their pricing models and/or the different total number of DNS queries we issued to them. More expensive plans with these MDNS providers provide a larger (instead of better performing) package plan with more DNS lookup queries and more domain names. On the latter front, our result is not sensitive to the set of probe nodes as long as we use enough geo-distributed nodes to cover the anycast network locations of these MDNS providers. The set of PlanetLab nodes we use in different months are already different as PlanetLab nodes tend to go up and down, but the results in the last section suggest that that does not affect the consistency measurements.

V. Related work

To our knowledge, this paper is the first to conduct an empirical study of consistency (or lack thereof) in popular managed DNS services, and the first to quantify the extent of TTL abuse by local resolvers that was known to exist only anecdotally if at all.

Previous works have investigated consistency issues in DNS in order to make a stronger case for consistency or propose alternatives to achieve stronger consistency. Jung et al. [37] suggest that reducing TTLs aggressively is unlikely to impact DNS-induced latency while improving freshness. DNScup by Chen et al. [7] proposes a cache update protocol using dynamic leases for strong(er) consistency. The specific lease-based scheme(s) proposed use lazy propagation upon updates, so they don't provably ensure strong consistency semantics like sequential or linearizable, but do help reduce staleness. ECO-DNS by Chen et al. [6] presents a model to set TTLs so as to optimize *aggregate consistency* or the fraction of lookup queries that see outdated values. Sharma et al. [10] present a "next-generation" global name service that ensures totally ordered writes based on consensus to coordinate updates. In contrast, although our work shares the overlapping motivation of the case for stronger consistency, it focuses on an empirical measurement of consistency in MDNS providers in the wild.

RFC 2136 [13] describes *Dynamic DNS*, a protocol extension to support DNS updates. In practice, "dynamic DNS" may be used also to refer to alternatives like HTTP APIs that essentially accomplish the same. RFC 2136 requires dynamic DNS updates and lookups to satisfy "atomicity", which is used to mean transactional semantics. One interpretation of RFC 2136's atomicity requirement is that a replicated nameserver system should ensure linearizability, however a literal reading of RFC 2136 is ambiguous on whether transactional semantics should be satisfied only at each nameserver independently or across the replicated nameserver system; our study indicates that the latter is definitely not obeyed by most MDNS providers (as linearizability would imply monotonic reads).

Some prior studies have observed the poor update performance exhibited by popular MDNS services, e.g., Sharma et al. in 2014 [10] and by CloudHarmony in 2012 [30]. We reaffirm the results for some MDNS providers but find others (e.g., Route53) to be out-dated. Different from the prior works, we also identify the impact of stale caching posed by some local DNS resolvers operated by ISPs. Most performance monitoring of MDNS services is outsourced to performance measurement platforms such as DNSPerf [27] and Catchpoint [29] that are suitable for monitoring the performance of lookups, but are unsuitable for updates.

Our work at a high-level is also related to the prior studies that measure observable consistency (or lack thereof)

in distributed systems in contexts other than DNS, e.g., measuring linearizability and sequential consistency for social networks [12] or key-value stores [38].

VI. Conclusion

The Internet has come a long way from DNS' early days when updates were expected to be rare and the design decision by the creators of DNS to favor weak consistency and extensive caching over more sophisticated replication systems [39] was understandable. In recent years, despite the widely noted need for agile updates for modern application scenarios, we find that DNS update performance continues to be poor. Update propagation delays commonly take tens of seconds and have hardly improved over the last several years. Worse, eventual consistency appears to be interpreted in practice as a license to not worry about consistency correctness at all. We find that roughly a third of end-user queries are affected by resolvers that disrespect TTL limits. Few MDNS providers ensure monotonic reads consistency and at least one provider likely violates even eventual consistency. Our study suggests that MDNS providers should consider critically re-examining update performance and consistency correctness in their production services. *All measurements in this paper can be independently validated using the data and scripts at [40].*

Acknowledgments. This work was in part supported by National Science Foundation awards #1636574, #1413963, #1719386, and #1823131. We thank all the representatives from various managed DNS companies for their assistance. The findings or opinions presented in this paper are not necessarily endorsed by anyone except the authors.

References

[1] "CloudFlare: Fast, Powerful, and Secure DNS," https://www.cloudflare.com/dns/.
[2] "Firefox invalidate dns cache," https://stackoverflow.com/questions/13063496.
[3] "How to force DNS refresh for a website?" https://stackoverflow.com/questions/21091321.
[4] "Subdomain caching issue," https://stackoverflow.com/questions/5334395.
[5] "Why is my domain still going to the wrong server?" https://serverfault.com/questions/172101.
[6] C. Chen, S. Matsumoto, and A. Perrig, "ECO-DNS: expected consistency optimization for DNS," in *ICDCS*. IEEE Computer Society, 2015, pp. 256–267.
[7] X. Chen, H. Wang, and S. Ren, "DNScup: Strong Cache Consistency Protocol for DNS," in *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, ser. ICDCS '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 40–. [Online]. Available: https://doi.org/10.1109/ICDCS.2006.31
[8] C. Dannewitz, M. DAmbrosio, and V. Vercellone, "Hierarchical dht-based name resolution for information-centric networks," *Computer Communications*, vol. 36, no. 7, pp. 736–749, 2013.
[9] Z. Gao, A. Venkataramani, J. F. Kurose, and S. Heimlicher, "Towards a quantitative comparison of location-independent network architectures," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 259–270.
[10] A. Sharma, X. Tie, H. Uppal, A. Venkataramani, D. Westbrook, and A. Yadav, "A global name service for a highly mobile internetwork," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 247–258, 2015.
[11] G. Liu, H. Shen, H. Chandler, and J. Li, "Measuring and evaluating live content consistency in a large-scale cdn," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 7, pp. 2074–2090, 2016.
[12] H. Lu, K. Veeraraghavan, P. Ajoux, J. Hunt, Y. J. Song, W. Tobagus, S. Kumar, and W. Lloyd, "Existential consistency: measuring and understanding consistency at facebook," in *Proceedings of the 25th Symposium on Operating Systems Principles*. ACM, 2015, pp. 295–310.
[13] P. Vixie, "Dynamic updates in the domain name system," *RFC 2136*, 1997.
[14] "Ping Test," https://tools.keycdn.com/ping.
[15] "MAXMIND," http://www.maxmind.com.
[16] P. Mockapetris, "Rfc 1034: Domain names-concepts and facilities, 1987," *URL: ftp://ftp.isi.edu/in-notes/rfc1034.txt*.
[17] M. Jonker, A. Sperotto, R. van Rijswijk-Deij, R. Sadre, and A. Pras, "Measuring the adoption of DDoS protection services," in *Proceedings of the 2016 Internet Measurement Conference*. ACM, 2016, pp. 279–285.
[18] "Datanyze DNS ranking list," https://www.datanyze.com/market-share/dns/.
[19] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically scaling applications in the cloud," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 45–52, 2011.
[20] F. Manco, C. Lupu, F. Schmidt, J. Mendes, S. Kuenzer, S. Sati, K. Yasukata, C. Raiciu, and F. Huici, "My VM is Lighter (and Safer) than your Container," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 218–233.
[21] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computinga key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.
[22] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, 2014.
[23] "Amazon EC2 Spot Instances," https://aws.amazon.com/ec2/spot/.
[24] "Google Preemptible Virtual Machines," https://cloud.google.com/preemptible-vms/.
[25] L. Lamport *et al.*, "Paxos made simple," *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, 2001.
[26] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "ZooKeeper: Wait-free Coordination for Internet-scale Systems." in *USENIX annual technical conference*, vol. 8, no. 9. Boston, MA, USA, 2010.
[27] "DNSPerf," https://www.dnsperf.com/.
[28] "RIPE Atlas," https://atlas.ripe.net.
[29] "Catchpoint," http://www.catchpoint.com/.
[30] J. Read, "Comparison and analysis of managed dns providers, aug 2012," *Cloud Harmony Inc*.
[31] M. Kührer, T. Hupperich, J. Bushart, C. Rossow, and T. Holz, "Going Wild: Large-Scale Classification of Open DNS Resolvers," in *Proceedings of the 2015 Internet Measurement Conference*, ser. IMC '15. New York, NY, USA: ACM, 2015, pp. 355–368. [Online]. Available: http://doi.acm.org/10.1145/2815675.2815683
[32] H. Ballani and P. Francis, "Mitigating dns dos attacks," in *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 2008, pp. 189–198.
[33] "RIPE Atlas Architecture," https://labs.ripe.net/Members/kistel/ripe-atlas-architecture.
[34] A. S. Tanenbaum and M. Van Steen, *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.
[35] "Managed DNS API Rate Limit," https://help.dyn.com/managed-dns-api-rate-limit/.
[36] "Google Cloud DNS Changes," https://cloud.google.com/dns/api/v1beta2/changes#resource.
[37] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, "DNS performance and the effectiveness of caching," *IEEE/ACM Transactions on networking*, vol. 10, no. 5, pp. 589–603, 2002.
[38] E. Anderson, X. Li, M. A. Shah, J. Tucek, and J. J. Wylie, "What consistency does your key-value store actually provide?" in *HotDep*, vol. 10, 2010, pp. 1–16.
[39] P. Mockapetris and K. J. Dunlap, "Development of the domain name system," *SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, pp. 123–133, Aug. 1988. [Online]. Available: http://doi.acm.org/10.1145/52325.52338
[40] "Data and scripts," https://people.cs.umass.edu/~gaozy/dns.html.