Implications of Netalyzr's DNS Measurements

Nicholas Weaver
ICSI

Christian Kreibich
ICSI

Boris Nechaev
HIIT & Aalto University

Vern Paxson
ICSI & UC Berkeley

***Abstract*—*Netalyzr* is a widely used network measurement and diagnosis tool. To date, it has collected 198,000 measurement sessions from 146,000 distinct IP addresses. One of the primary focus areas of *Netalyzr* is DNS behavior, including DNS resolver properties, common name lookups, NXDOMAIN wildcarding, lookup performance, and on-the-wire manipulations. Additional tests detect and categorize the behavior of any DNS proxies in the users' gateways or firewalls. In this paper we report on DNS-specific insights from *Netalyzr*'s growing dataset. We identify significant problems in the existing DNS infrastructure, including unreliability of IP-level fragmentation, several kinds of result wildcarding, surprisingly poor lookup performance, and deliberate in-path DNS message manipulations. As these observations affect implementers of the DNS protocol as well as developers using common DNS APIs, we offer recommendations on common pitfalls and highlight likely impediments to the deployment of upcoming DNS technologies.**

I. Introduction

The *ICSI Netalyzr* is a widely used network diagnosis and debugging tool, available at http://netalyzr.icsi.berkeley.edu. This publicly available service enables the user to obtain a detailed analysis of the operational envelope of their Internet connectivity, serving both as a source of information for the curious as well as an extensive troubleshooting diagnostic should users find anything amiss with their network experience. *Netalyzr* tests a wide array of properties of users' Internet access, from the network layer to applications. Its tests include IP address use and translation, IPv6 support, DNS resolver fidelity and security, TCP/UDP service reachability, proxying and firewalling, antivirus intervention, content-based download restrictions, content manipulation, HTTP caching prevalence and correctness, network and protocol-level latencies, and access-link buffering.

In this paper we report on DNS-specific findings from *Netalyzr*'s longitudinal dataset, including sessions recorded in the six months since we first published *Netalyzr*'s results [18] as well as results from tests we only implemented recently. We focus particularly on findings that affect implementers of DNS protocols, and on those that software authors using DNS should be aware of.

Our measurements lead to three main conclusions. First, we observe significant complications for the deployment of DNSSEC and other size-sensitive DNS features. These rely on message sizes large enough to require IP fragmentation (which works only suboptimally in practice) and, potentially, TCP failover (which thankfully works well). Second, we find recursive resolvers untrustworthy in practice, with frequent NXDOMAIN wildcarding, limited cases of Internet Service Providers (ISPs) using DNS to redirect web search traffic to their own proxies, and malicious resolvers that inject advertisements or block system updates. Third, we note that buggy or restrictive DNS proxy implementations are common in today's home-network gateways, imposed upon users as part of a gateway's DHCP service. We find such DNS proxies frequently refuse to process AAAA, EDNS0, TXT, and non-standard resource records (RRs).

We next briefly review *Netalyzr*'s architecture before discussing our findings and their implications in more detail.

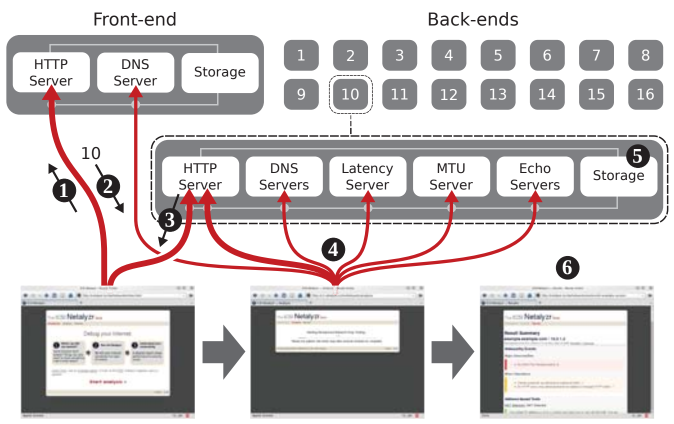

Fig. 1. *Netalyzr*'s conceptual architecture. ❶ The user visits the *Netalyzr* website. ❷ When starting the test, the front-end redirects the session to a randomly selected back-end node. ❸ The browser downloads and executes the applet. ❹ The applet conducts test connections to various *Netalyzr* servers on the back-end, as well as DNS requests that are eventually received by the main *Netalyzr* DNS server on the front-end. ❺ We store the test results and raw network traffic for later analysis. ❻ *Netalyzr* presents a summary of the test results to the user.

II. System Design & Implementation

When designing *Netalyzr* we had to strike a balance between a tool with sufficient flexibility to conduct a wide range of measurements and tests, and a simple interface that non-technical users would find usable. To this end, we decided to base our approach on a Java applet ($\approx$ 5,000 lines of code) to drive the bulk of the test communication with custom servers ($\approx$ 12,000 lines of code), since (*i*) Java applets run automatically within most major web browsers, (*ii*) applets can engage in raw TCP and UDP flows to arbitrary ports (though not with altered IP headers), (*iii*) if the user approves trusting the applet, it can contact hosts outside the same-origin policy, (*iv*) Java applets come with intrinsic security guarantees for users (e.g., no host-level file system access allowed by default runtime policies), (*v*) Java's fine-grained permissions model allows us to adapt gracefully if a user declines to fully trust our applet, and (*vi*) no alternative technology matches this level of functionality, security, and convenience.

Figure 1 shows the *Netalyzr* architecture. For more details, we refer the reader to our prior work [18], and here only reiterate the setup's conceptual separation into front- and back-ends. For our DNS-related tests, we operate DNS servers running identical code ($\approx$ 2,800 lines) on both. We developed a custom implementation because our server deliberately strays from the DNS standards, including replying to corrupted requests, manipulating glue, and keeping state across queries. This implementation also facilitates extensive logging. The Java applet uses a combination of the runtime's DNS API and direct UDP requests for which we craft the DNS PDUs ourselves, particularly for our recent DNS test additions.

The applet can run either in *trusted* or *untrusted* mode. Depending on the browser, the Java applet runtime presents a message to the user at applet startup that asks whether the user trusts the applet, showing the applet's signature as an indication as to the origin of the code. If the user confirms trust, the applet is allowed to conduct a more extensive set of tests. Details depend on the runtime's configuration. However, we can identify the extent to which we are allowed to engage in network I/O by catching Java's security exceptions.

The DNS tests are generally driven by client-side requests; these come in the form of queries containing test directives as part of the name that the applet looks up, such as *testname*.*nonce*.`netalyzr.icsi.berkeley.edu`. When mentioning test names, we will generally only show the relevant part of the queried name. For tests with Boolean outcomes, we return the IP address of the applet's origin server to encode "true," and another address for "false." For example, a name starting with `has_edns` triggers a scan of the request message for EDNS support, returning "true" or "false" as appropriate. Note that this Boolean approach works within the confines of Java's default same-origin policy, so we can employ it even when the applet runs untrusted. In this mode, Java only allows the applet to contact the IP address of the server that provided the applet, but the applet can still attempt to look up arbitrary names. The runtime enforces restrictions on the DNS responses: if the response contains the IP address of the applet's origin server, the response is delivered; otherwise, Java generates a security exception. On the client, receipt of the origin server's IP address thus signals "true," while a security exception indicates "false." In trusted mode, we can also perform tests that encode numeric values in the returned A records.

We note that our measurements have some skew regarding who decided to run *Netalyzr*, with a bias towards more technologically-aware users. In particular, we observe a large number of OpenDNS and Comcast users, mainly because a major technology news site featured *Netalyzr* in context of coverage of Comcast's DNS policy. Our data collection is generally prone to such "flash crowds," resulting from exposure the tool receives on technical blogs and news sites. Another skew comes from *Netalyzr*'s adoption as a debugging tool by the on-line game League of Legends, which has resulted in 14,700 sesions to date.

III. In-Gateway DNS Proxies

The user's NAT or gateway device plays a key role with respect to the correct functioning of the DNS path. Home gateways often provide DHCP leases that configure the gateway's own IP address as the LAN's DNS recursive resolver. Doing so allows the gateway to support a functioning LAN independently of the global DHCP lease (which also includes DNS configuration) that the gateway acquires from the ISP. It also enables the user to access the gateway's administrative interface via custom DNS (such as `www.routerlogin.net` for Netgear devices, or `gateway.2wire.net` for 2Wire systems).

DNS proxy detection. *Netalyzr* does not try to determine the gateway's IP address via intrusive scanning, so we cannot definitively check whether a system is configured to use a DNS proxy. However, we can detect the existence of a potential DNS proxy by sending requests to common gateway addresses directly and seeing whether they elicit a response. If *Netalyzr* detects that the client is behind a NAT, it probes a set of typical gateway IP addresses. Initially, we only tried the `.1` address within the local subnet (e.g., given local IP address `192.168.1.24`, *Netalyzr* probed `192.168.1.1`). More recently, it came to our attention that 2Wire and some other devices instead use `.254`, which we then added to the test. The 73% of sessions behind a NAT in which we identify an in-gateway DNS proxy thus reflect a lower bound. Since 88% of sessions are behind NATs, the behavior of these DNS proxies is crucial, as these are the DNS "resolvers" seen by a large fraction of Internet users.[1]

Proxy behavior. We recently broadened our DNS tests to determine how these gateways behave, including whether they can correctly process: (*i*) AAAA lookups (96% of sessions),[2] (*ii*) TXT records (92% of sessions), (*iii*) unknown RRs (RTYPE 1169) per RFC 3597 [13] (91% of sessions), and (*iv*) EDNS0 requests (91%).

External DNS functionality. To our surprise, a significant number of NATs have *externally* usable DNS proxies, as evidenced by 5.3% of the clients' global IP addresses. Not only do these proxies provide access to the ISP's DNS resolution path to unintended third parties, these proxies can be used to launch reflector attacks [22], where the attacker uses small spoofed query to generate large responses launched at the target. As a 100B DNS request can elicit a 4000B (enabled by EDNS0) reply, attackers can obtain 40-fold message size amplification for spoofed-source reflectors. Even without DNS, DNS replies of 512B can provide 5-fold amplification. Arbor Networks

[1]Unfortunately, we did not check whether the end user's system is configured to use the gateway's proxy, or if the gateway's DHCP server will instead return the IP address of the recursive DNS resolver it receives from its own DHCP lease [11].

[2]Silent failures in this manner can prove problematic for some Linux systems that perform A and AAAA lookups in parallel, and require that both lookups complete or time out before returning a record from `gethostbyname()`, even when the system lacks routable IPv6 connectivity.

reports that many of the largest DDoS attacks include DNS reflectors [3].

Other aberrations. We have also noticed specific aberrations with subtle effects. When debugging one of our home networks, we observed that a 2Wire gateway would prepend `gateway.2wire.net` to the DNS search path, but responded to `*.gateway.2wire.net` with SERVFAIL rather than NXDOMAIN errors. Our browser repeatedly retried these lookups before moving on to the next element of the search path, which stalled web searches conducted by typing a word into the location bar by up to 30 sec.

Finally, we have seen a small number of cases where NATs (likely D-Link products [4]) confuse AAAA and A records. When the user's system queries for both A and AAAA records for a name and the AAAA record exists but the A record does not (e.g., because the name reflects an IPv6-only site), the A record request instead returns the first 4 bytes of the AAAA record. This behavior can also manifest if the reply for the A record request is simply unduly delayed.

IV. Fragmentation, Path MTU, and Filtering

DNS primarily relies on UDP for transmission. With the EDNS0 [24] extension mechanism for DNS, replies may exceed the common 1500B Ethernet MTU, requiring the underlying IP fragmentation mechanism to work correctly in order for DNS PDUs to survive the transfer. Therefore, *Netalyzr* tests the client's MTU on the path to our server. The MTU test consists of two parts: fragmentation and path MTU discovery.

Fragmentation. We test fragmentation by attempting to send 2000B UDP datagrams from the client (for the client $\rightarrow$ server path) and the server (server $\rightarrow$ client path). Such datagrams are naturally fragmented, as their size exceeds the 1500B Ethernet MTU. If the reassembled original datagram does not arrive as expected, the system cannot handle fragmented IP traffic correctly. Unfortunately this is quite common: 8% of the sessions cannot send fragmented UDP, and, more importantly for DNS, 9% cannot receive fragmented UDP.

Fortunately, TCP remains an option for most of these clients. While 5% of the systems cannot contact a DNS server over TCP, only 0.16% of the sessions both fail to receive IP fragments and lack the ability to contact DNS servers using TCP. DNS's TCP failover, coupled with strategies for detecting and mitigating fragmentation failures, should therefore work fairly well in practice.

Path MTU. As expected, the path MTUs are commonly, but not exclusively, that of Ethernet. We discover the precise path MTU only in the server $\rightarrow$ client direction, as the required IP "Don't Fragment" (DF) bit is not accessible via the Java API. 78% of our sessions report the full Ethernet MTU of 1500B, while 16% show the PPP-over-Ethernet [20] MTU of 1492B. Only 2% indicate an MTU of less than 1450B.

Sessions with an MTU less than Ethernet's do not reliably trigger ICMP Destination Unreachable messages with code "Too Big:" only 60% of such sessions included these. Systems that employ path MTU discovery via UDP by default (such as recent Linux versions) can in such situations frustrate the developer, as the combination of unintended IP DF bits and the failure of the ICMP mechanism conspire to create MTU "black holes" or spurious application-level exceptions.[3]

Filtering. We have also encountered numerous firewalls and gateways that filter, block, or modify DNS over UDP. We detect these by performing both correctly and incorrectly formatted DNS queries to our server. Content-aware devices may block incorrectly formatted queries, while leaving proper queries unaffected.

99% of clients were able to access our DNS server over UDP with legitimate queries. Sometimes this ability is controlled by the gateway: 1.4% of sessions showed evidence of DNS traffic being proxied or modified by the network. Thus the request never actually was sent out over the network to the intended recipient server, but was instead redirected to a recursive resolver which processed the request.

10% of clients could access our DNS server directly with correct requests but not when using ill-formed requests, suggesting that a firewall or gateway enforced DNS semantics on requests. This includes cases where a mandatory DNS proxy blocked the invalid request, as well as those behind firewalls which only block ill-formed requests.

For the results reported in [18], *Netalyzr* included tests to detect whether direct EDNS requests to our server are allowed. We subsequently added tests to check for filtering of AAAA records, TXT records, and requests for unknown RRs. 98.3% of sessions succeeded at fetching AAAA records, 98.3% for TXT records, 98.5% for retrieving records using EDNS0, and 95.8% for 1400B records using EDNS0. To test for retrieval of unknown RRs, we arbitrarily used RTYPE 169, finding 97.2% of sessions could successfully retrieve these.

Recent DNS proposals suggest encoding data of various novel types into TXT records. For example, the Sender Policy Framework [25] recommends that SPF-enabled domains should provide both a record of RTYPE 99 and an equivalent TXT record textually encoding equivalent information. We find little benefit in this redundancy, since the capability to receive TXT records highly correlates with being able to receive arbitrary RTYPE RRs: only 1.2% of sessions could only retrieve one of the two types.

V. Recursive Resolver Properties

Netalyzr includes extensive tests of the recursive resolver. Because of limitations the Java runtime imposes on applets, we were unable to obtain the IP address of the recursive resolver configured on the client directly. We thus conduct our tests of the recursive resolver *indirectly*, by querying regular names

[3]If the ICMP "Too Big" message is not received, the application naturally has no immediate way of knowing that delivery failed; if it is received, related exceptions may be unexpected by the application developer because there was no reason to assume sensitivity to message sizes.

using `getbyname()` and related API calls that return A or AAAA records.[4]

EDNS MTUs. The path problems observed for clients also apply to many recursive resolvers. When we detect that the resolver uses EDNS0 (55% of sessions), *Netalyzr* captures the EDNS0 MTU using a name lookup that encodes the advertised MTU in the lower 16 bits of the response A record. *Netalyzr* then verifies that the resolver indeed supports this claimed MTU.

We initially used three names whose lookups triggered responses padded to $\approx$ 300B, 1350B, and 1800B, respectively, using unrelated CNAME records in the Additional field. Since these records are almost always stripped by the recursive resolver when returned to the client, these lookups serve to test whether the resolver indeed supports its advertised MTU.

Of those sessions in which the resolver claimed the ability to receive an 1800B response, only 87% could actually process the large reply, while 98.5% could process the medium sized reply. This finding suggests that a common failure mode for these queries arises from networks that cannot handle fragmentation.

Accordingly, we recommend that resolvers both detect this condition and fail over to using a smaller (1400B) EDNS0 MTU when it occurs. BIND [15] has supported a slightly different failover strategy since version 9.5: when a request times out, BIND will retry with an EDNS0 MTU of 512B and, if successful, use the smaller MTU for communication with the server. We believe that first a 1400B MTU failover should be used and, if several servers all fail with a larger MTU but succeed with a 1400B MTU, the resolver should assume that the problem lies in a local firewall rather than with the remote servers and adjust its MTU accordingly.

TCP failover. We recently added a TCP failover test to *Netalyzr*. The name `truncate` always returns a result with the DNS truncation bit (TC) set when using UDP, but a normal (different) result via TCP. Queries for this name can determine whether the resolver properly fails over to TCP. Only 2% of the sessions that included this test failed. 0.35% of the sessions exhibited resolvers that ignored the truncation bit, instead returning the value from the UDP datagram. 0.51% of sessions showed both an inability of the recursive resolver to handle large EDNS0 responses while advertising a large MTU, and a failure to respond to truncation requests. These results suggest that TCP failover can effectively compensate for UDP fragmentation problems.

Port randomization. We find DNS port randomization widespread but not universal. We released *Netalyzr* one year after cache poisoning attacks received widespread press coverage [6]. We observe that 4% of sessions still do not evince DNS source port randomization by the recursive resolver. Most of these sessions reflect home networks running a local resolver on the NAT or end-system; in 24% of these non-randommized sessions we find the global IP address for general traffic matches the address that issues DNS requests to our server, compared to 3% of the sessions where the identified recursive resolver does not match the end-system's global IP address. Thus, although lack of randomization remains an issue for some users, most ISPs and institutions have fixed this problem.

[4]We subsequently learned that a Sun Java extension exists that can obtain the IP address of the recursive resolver. The next release of Netalyzr will use this information to probe the recursive resolvers directly when the extended API is available.

Lookup latencies. We measure name lookup latency for uncached names for each *Netalyzr* execution, and cached-name latency whenever the recursive resolver accepts glue records. We find the generally poor performance seen by many clients striking. In 10% of sessions it took 300ms longer to look up a DNS record from our server than a round-trip of a UDP packet to our back-end server. We crudely estimate that up to 100ms are due to the different server locations: the front-end server, which hosts the DNS authority, is located at ICSI in Berkeley, while the back-end servers are located at Amazon's East coast EC2 location. 4.4% of sessions required an additional 600ms or more.

Perhaps more surprisingly, even cached lookups are often slow: 11% of sessions exhibit a delay of 200ms or more for looking up a cached record. This is strongly correlated with the usage of third-party resolvers: 15% of OpenDNS customers experienced cached lookups taking 200ms, while 9% of non-OpenDNS users experienced an equivalent delay. 19% of the non-OpenDNS users experience at least 100ms of delay to the cache.

Figure 2 compares the latencies across different service providers. For uncached latencies, distributions center just below 100ms with the exception of the slightly faster SBC and the slightly slower Google. DNS providers do not appear obviously faster. A second modality is frequently present around 25ms. As expected cached latencies are faster throughout, but with differing variances.

Miscellaneous tests. We employ a series of tests probing the recursive resolver to infer its glue-handling policy. More specifically, we test the ability to cache glue records that appear in either the Authoritative or Additional field. These tests use a different A record when included in the Additional field than returned by a direct lookup. We find caching of Additional records in 15% of sessions, but 44% of sessions cache glue included in Authoritative records.

We do not observe widespread implementation of 0x20 [8] (only 4.0% of sessions), but 94% of resolvers either propagate capitalization unmodified, or directly implement 0x20. Thus, this defense holds solid promise.

We find significant EDNS0 usage, with 55% of sessions exhibiting a recursive resolver that supports it. Most of the usage occurs in the context of requesting DNSSEC data (95% of the sessions). The most common EDNS0 MTU (92% of the sessions) is the 4096B BIND MTU, with other MTUs including 512B (2.2%), 1280B (0.5%), and 2048B (1.3%).

We recently added a test for whether resolvers support AAAA-only glue records. 5.1% were able to generate requests to IPv6-only DNS servers.

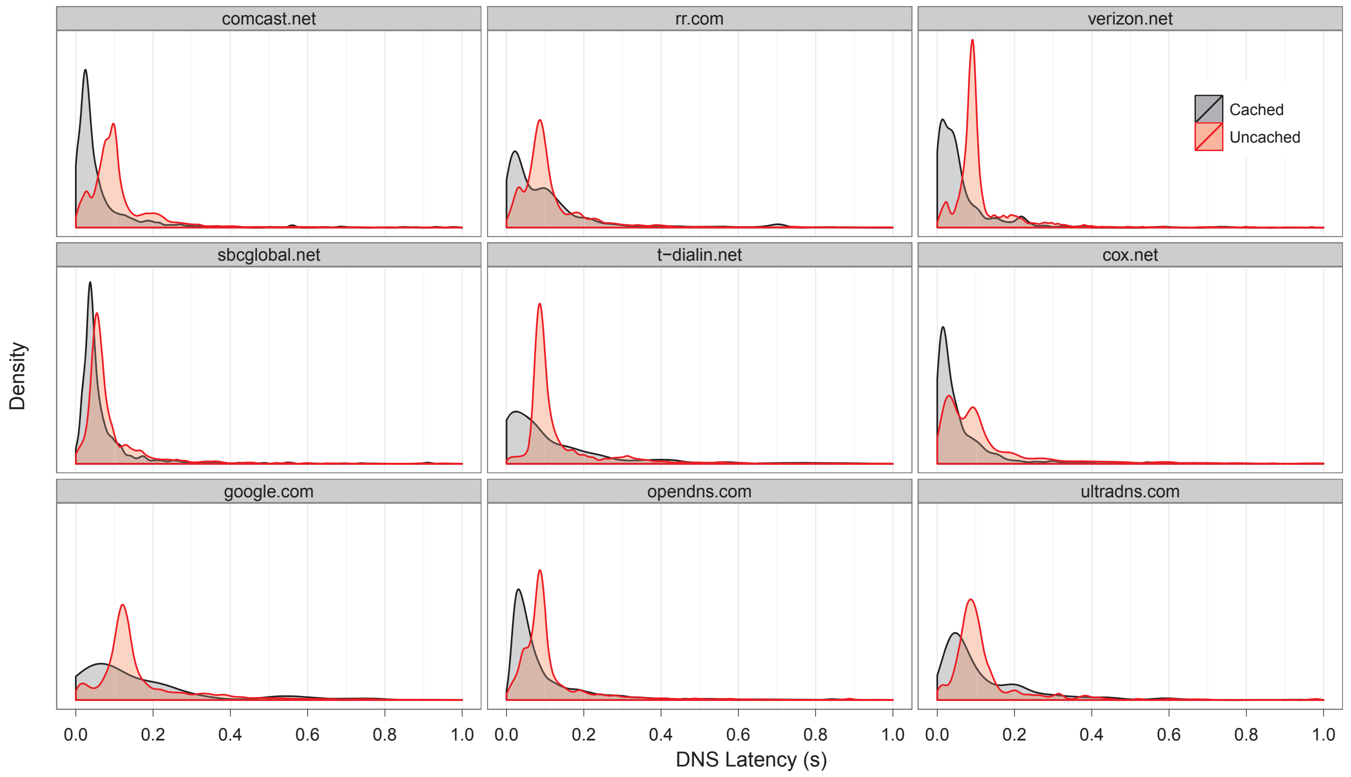

Fig. 2. Probability densities for uncached and cached DNS lookup latencies, for the largest ISPs (top two rows) and DNS providers (bottom row). Y-axis scaling is identical across all plots.

VI. Lookup Result Fidelity

A striking finding from the *Netalyzr* sessions regards widespread manipulation of results by recursive resolvers.

Result wildcarding. 27% of sessions show NXDOMAIN error wildcarding, where the recursive resolver masks query errors with artificial, valid responses that usually direct browsers to a third-party site showing advertising or search engine results potentially related to the original query. *Netalyzr*'s dataset has a bias towards Comcast (which now wildcards) and OpenDNS (which has always wildcarded). Even excluding these, 24% of the sessions show wildcarding. Many of these resolvers do not only wildcard names that begin with `www`, but instead act as though *all* name lookups returning errors stemmed from queries generated by web browsers. Excluding Comcast (which only wildcards `www`) and OpenDNS (which wildcards universally), 42% of such sessions wildcard all NXDOMAINs. Going further, 6% of sessions (1% excluding OpenDNS users) wildcard SERVFAIL in addition, an ill-advised practice because it transforms the semantics of transient error conditions.

While controversial, NXDOMAIN wildcarding remains a commercial reality. Like others [7], we argue that its implementation should strive to minimize collateral damage, viewed through the lens that only web surfing provides revenue opportunity for those implementing NXDOMAIN wildcarding in its current form.

We observe two other forms of wildcarding. In the first, the resolver masks a valid name's response with a different IP protocol family. We discovered this behavior inadvertently, while adding IPv6 functionality tests. In this form of wildcarding, a query for an IPv6-only name receives in response an IPv4 address generated by the wildcarding logic. This change undermines any IPv6-only names (e.g., `ipv6.google.com`). Of the sessions including this test, 5% (1.0% excluding OpenDNS) will wildcard to an IPv4 address where there only exists an IPv6 address. In the second form, the resolver treats responses that confirm the server as authoritative but that omit additional Answer RRs (e.g., because a query was performed for an A record but the given name only has an MX or AAAA record associated with it) as NXDOMAIN errors, and wildcards them similarly. We have observed this behavior in sessions using OpenDNS, and alerted them to the problem.

Target-dependent redirection. Other forms of result manipulation exist. As we reported previously, *Netalyzr* identified multiple ISPs that use DNS to redirect web searches for popular sites, such as `www.google.com`, `search.yahoo.com`, and `www.bing.com` [18]. Instead of visiting the intended search engines' IP addresses, the user winds up redirected to proxy servers. Some ISPs only manipulate Yahoo and Bing, while others manipulate all three.

The proxy servers appear to operate as an outsourced service, with each ISP redirecting Yahoo and Bing to an ISPunique address in one of two prefixes 8.15.228/24 or 69.25.212/24. The redirection occurs via the ISP's DNS resolver, rather than via interception of DNS requests within the network itself, since users who use third-party resolvers do not experience redirection.

Some ISPs that redirect Google use the same outsourced proxy for it too, while others redirect Google to an ISP-controlled proxy running apparently the same software. This software has a recognizable signature in terms of the particular headers it when queried for hosts it does not proxy, and in particular regarding the HTML page it returns in response to ill-formed HTTP requests. A request for an unproxied host returns a HTTP 302 redirect to http://255.255.255.255/ with a banner indicating the Squid software version. Ill-formed requests include a HTML page with a note that the page was generated by phishing-warning-site.com, a parked domain rather than an actual live site.

Offending ISPs include Cavalier, Charter, Cincinnati Bell, Cogent Communications, DirecPC, Frontier, Insight Broadband, Iowa Telecom, RCN, and Wide Open West. Cursory inspection of the search result pages shows no evidence of obvious content or header manipulation.

Malware. Finally, 141 sessions showed signs of malware tampering with the resolver configuration, as observed previously by Dagon et al. [9]. These changes direct DNS requests to malicious resolvers that can then control responses at will, such as by injecting advertisements [12] or deliberately disabling the resolution of windowsupdate.microsoft.com in order to prevent system updates.

In summary, our findings demonstrate that we cannot in general treat today's recursive resolvers as trustworthy systems, as they are prone to manipulation. As recursive resolvers also play a key role in DNSSEC validation, we argue that here they also pose a trust problem. Ideally, end systems would only trust DNSSEC validations they perform themselves, bypassing the recursive resolver to conduct queries. Clearly, such an approach would incur significant implementation complexity, as well as diminished caching efficacy due to lack of broader sharing of cached results.

VII. Summary of Recommendations

Netalyzr's ongoing operations for over 1.5 years has allowed us to gather a diverse set of measurements regarding the capabilities of today's end systems when interacting with DNS. Based on our analysis of this data, we can make several recommendations to DNS implementers and users of DNS APIs.

DNS client software such as web browsers should not rely on correct reporting of NXDOMAIN or SERVFAIL errors. Non-cryptographic protocols such as HTTP can test for error reliability by attempting a few known-to-fail and known-to-succeed queries, using the results to check for wildcarding. For cryptographic protocols, a failure to establish a connection should catch faulty wildcarding before the user is informed that a problem exists, in order to prevent false alarms that might trip up even savvy users [10].

If client software wishes to use novel RRs or TXT records to encode newly defined types, the client software may need to include its own DNS library in order to bypass the host's stub resolver, as the latter is often configured to use the gateway's not-fully-functional DNS proxy. The same applies to DNSSEC validation, though here the situation is exacerbated by the fact that current stub resolvers will not necessarily perform validations, and validations performed by the recursive resolver cannot be trusted. Thus, the Authenticated Data (AD) bit should be considered to hold little weight.

Stub resolvers and resolver libraries should accommodate failure modes such as that (*i*) the configured "recursive" resolver provided by the DHCP server may not support the full DNS specification, (*ii*) such resolvers cannot be fully trusted, and (*iii*) the network may impinge on UDP DNS traffic, including filtering or blocking of fragments. Implementations must support TCP failover and should include routines to detect and respond to filter-induced failures, including blocked EDNS, blocked RRs, blocked large DNS replies, and blocked fragments.

Performance-oriented stub libraries might also benefit from bypassing the recursive resolver if testing indicates that it is slow to return cached responses. Since 19% of sessions require at least 100ms to fetch a *cached* item, a significant number of clients benefit only marginally from caching.

DNS proxies in gateways should be tested for handling of unknown resource records, à la RFC 3597. The easiest mechanism for ensuring correct operation is for proxies to act as forwarders, relaying unknown messages unchanged (as done e.g. by DNSMask [17]). Likewise, such proxies should not respond to external requests. Unfortunately, correcting the existing installation base of improperly implemented NATs will at best likely take significant time, so we require software to work around these problems explicitly.

ISPs may find it advisable to manage DNS port 53 traffic not just outbound [21] but also *inbound*, perhaps similarly to how they handle outbound SMTP today, in order to detect open DNS proxies and prevent contributing to DNS reflector attacks.

Recursive resolvers commonly face transport issues with fragmented UDP, making it advisable to add an additional fallback mechanism to the normal EDNS failure chain, where the resolver retries requests with a 1400B EDNS0 MTU before reverting to 512B. Luckily, TCP fallback generally works well.

If the resolver performs wildcarding, it should not wildcard SERVFAIL or valid names that simply lack an A record, as such behavior can prove disruptive, and also does not help with driving traffic to the advertising portal.

Authorities might, in a similar vein, consider always using a 1400B EDNS0 MTU and attempting to keep replies below this limit. This recommendation may prove controversial, as it will cause more TCP failovers; but one could argue that consistent results are preferable to unpredictable timeouts.

Software developers under Linux need to consider the UDP

PMTU behavior. If they do not disable PMTU discovery on a given socket, they may encounter spurious exceptions if a path MTU bottleneck generates ICMP too-big messages, and path MTU “black holes” at bottlenecks if they do not send ICMP too-big messages. If a software developer does not wish to do TCP-like PMTU failure detection in software, they should disable PMTU discovery. In terms of resolver support, we see little advantage in encoding data in TXT records when compared to defining new resource records. We would thus encourage developers to define a new RR type when justified.

VIII. Related Work

There exists a significant body of related work dedicated to measuring the DNS infrastructure. In 2002, Jung et al. studied DNS traces recorded at MIT and KAIST [16] and found significant error rates, including 23% of requests remaining unanswered and 13% returning error codes. In the same year, Brownlee et al. reported on the traffic arriving at one of the DNS root servers [5], finding request caching sorely lacking and 14% of queries completely broken/bogus. Ager et al. are conducting an ongoing measurement study using DNS probing tools driven entirely by the client [2]. They see similar performance artifacts on lookup time, as well as significant performance degradation from using third-party resolvers such as OpenDNS. They have also observed load balancers that inhibit response caching completely. Dagon et al. performed a survey of open recursive resolver behavior, including analyzing why so many open such resolvers exist and their characteristics [9].

Lukie et al. have recently reported on path MTU behavior from the server’s vantage point [19]. They see comparable, if somewhat lower, failure rates for path MTU discovery where servers have a path MTU bottleneck.

Hatonen et al. [14] are actively cataloging and probing home gateways. Their checks include whether the gateway responds to DNS requests over TCP or UDP, whereas *Netalyzr* currently only tries UDP. Dietrich [11] measured commonly available gateways in Germany for support of unknown RR types, the ability to proxy requests with EDNS0 and DO bits set, and the ability to route requests directly to the network. He observed a set of common failures similar to ours, and also reports that most gateways will always return their proxy address rather than the ISP’s resolver for DHCP replies. DNS-OARC [1] has a DNS reply-size tester that checks for resolver transport problems. The Root DNS servers also reported a significant increase in TCP failover after the introduction of DURZ (Deliberately Unverifiable Root Zone) signatures [23].

IX. Conclusions

In 18 months of ongoing operation, the *Netalyzr* network diagnosis and debugging tool has collected a wide range of measurements of the Internet’s edge networks in general, and their DNS properties in particular. The picture emerging from 198,000 sessions from 146,000 distinct IP addresses is diverse. We find basic DNS functionality working well, but observe significant DNS implementation shortcomings in many gateway DNS proxies, in-network filtering of DNS traffic, widespread result wildcarding, ISP-driven manipulation of DNS results, and general difficulty in using IP fragmentation. Many of these problems affect future upgrades to the DNS protocol, since the existing edge network infrastructure is entrenched and problems will require significant time to repair. To this end we have provided a set of first recommendations to DNS implementers and developers using DNS APIs, for which we welcome the community’s feedback.

X. Acknowledgements

We are deeply grateful to the *Netalyzr* users for enabling this study and to our beta-testers for the insightful comments and feedback. We would particularly like to thank Mark Allman, Paul Barford, Scott Bradner, John Brzozowski, Randy Bush, Niels Bakker, Richard Clayton, Chris Cowart, Keith Dawson, Adrian Dimcev, Holger Dreger, Brandon Enright, Kevin Fall, Carrie Gates, Andrei Gurtov, Mark Handley, Theodore Hong, Kelly Kane, Simon Kelley, Matthew Kogan, Keith Medcalf, Thomas Narten, Michael Ross, Chris Switzer, Wooter Wijngaards, and Richard Woundy. We thank Amazon for supporting our EC2 deployment. This work was supported by the National Science Foundation under grants NSF CNS-0722035, NSF-0433702, and CNS-0905631, with additional support from Google and Comcast.

References

[1] “OARC’s DNS Reply Size Test Server,” https://www.dns−oarc.net/oarc/services/replysizetest.

[2] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig, “Comparing DNS Resolvers in the Wild,” in *Internet Measurement Conference (IMC)*, 2010.

[3] Arbor Networks, “Worldwide Infrastructure Security Report,” http://www.arbornetworks.com/report, 2010.

[4] ARIN IPv6 Wiki, “Customer problems that could occur,” http://getipv6.info/index.php/Customer_problems_that_could_occur#D−Link.

[5] N. Brownlee, K. Claffy, and E. Nemeth, “DNS measurements at a root server,” in *IEEE Global Telecommunications Conference (GLOBECOM)*, 2002, pp. 1672–1676.

[6] CERT, “Vulnerability Note VU#800113, Multiple DNS implementations vulnerable to cache poisoning.”

[7] T. Creighton, C. Griffiths, J. Livingood, and R. Weber, “DNS Redirect Use by Service Providers,” Internet draft, work in progress.

[8] D. Dagon, M. Antonakakis, P. Vixie, T. Jinmei, and W. Lee, “Increased DNS Forgery Resistance Through 0x20-bit Encoding,” in *Proceedings of Cryptography and Computer Security (CCS)*, 2008.

[9] D. Dagon, N. Provos, C. Lee, and W. Lee, “Corrupted DNS Resolution Paths: The Rise of a Malicious Resolution Authority,” in *Proceedings of the Network And Distributed Security Symposium (NDSS)*, 2008.

[10] DarkTangent, “DEFCON SSL Compromise? NO! – Defcon Forums,” https://forum.defcon.org/showthread.php?t=9805.

[11] T. Dietrich, “DNSSEC Support by Home Routers in Germany,” in *RIPE-60*, 2010.

[12] M. Fauenfelder, “How to get rid of Vimax ads,” http://boingboing.net/2009/01/16/how−to−get−rid−of−vi.html, January 2009.

[13] A. Gustafsson, “Handling of Unknown DNS Resource Record (RR) Types,” RFC 3597 (Proposed Standard), Tech. Rep. 3597, Sep. 2003, updated by RFCs 4033, 4034, 4035, 5395.

[14] S. Hatonen, A. Nyrhinen, L. Eggert, S. Strowes, P. Sarolahti, and M. Kojo, “An Experimental Study of Home Gateway Characteristics,” in *Internet Measurement Conference (IMC)*, 2010.

[15] Internet Systems Consortium, “BIND,” http://www.isc.org/software/bind.

[16] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, “DNS Performance and the Effectiveness of Caching,” *Networking, IEEE/ACM Transactions on*, vol. 10, no. 5, pp. 589–603, 2002.

[17] S. Kelley, “DNSMasq—A DNS Forwarder for NAT Firewalls,” http://www.thekelleys.org.uk/dnsmasq/doc.html.

[18] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson, “Netalyzr: Illuminating the edge network,” in *Internet Measurement Conference (IMC)*, 2010.

[19] M. Luckie and B. Stasiewicz, “Measuring Path MTU Discovery Behavior,” in *Internet Measurement Conference (IMC)*, 2010.

[20] L. Mamakos, K. Lidl, J. Evarts, D. Carrel, D. Simone, and R. Wheeler, “A Method for Transmitting PPP Over Ethernet (PPPoE),” IETF, RFC 2516, February 1999.

[21] Messaging Anti-Abuse Working Group (MAAWG), “Overview of DNS Security - Port 53 Protection,” http://www.maawg.org/sites/maawg/files/news/MAAWG_DNSPort53V1.0_2010−06.pdf, June 2010.

[22] V. Paxson, “An Analysis of Using Reflectors for Distributed Denial-of-Service Attacks,” *Computer Communication Review*, vol. 31, 2001.

[23] “Root DNSSEC,” http://www.root−dnssec.org.

[24] P. Vixie, “Extension Mechanisms for DNS (EDNS0),” RFC 2671 (Proposed Standard), Internet Engineering Task Force, Aug. 1999.

[25] M. Wong and W. Schlitt, “Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1,” http://www.ietf.org/rfc/rfc4408.txt, IETF, RFC 4408, April 2006.