Connection-Oriented DNS to Improve Privacy and Security

Liang Zhu*, Zi Hu*, John Heidemann*, Duane Wessels†, Allison Mankin†, Nikita Somaiya*
*USC/Information Sciences Institute †Verisign Labs

***Abstract*—The Domain Name System (DNS) seems ideal for connectionless UDP, yet this choice results in challenges of eavesdropping that compromises privacy, source-address spoofing that simplifies denial-of-service (DoS) attacks on the server and third parties, injection attacks that exploit fragmentation, and reply-size limits that constrain key sizes and policy choices. We propose *T-DNS* to address these problems. It uses TCP to smoothly support large payloads and to mitigate spoofing and amplification for DoS. T-DNS uses transport-layer security (TLS) to provide privacy from users to their DNS resolvers and optionally to authoritative servers. TCP and TLS are hardly novel, and expectations about DNS suggest connections will balloon client latency and overwhelm server with state. Our contribution is to show that T-DNS significantly improves security and privacy: TCP prevents denial-of-service (DoS) amplification against others, reduces the effects of DoS on the server, and simplifies policy choices about key size. TLS protects against eavesdroppers to the recursive resolver. Our second contribution is to show that with careful implementation choices, these benefits come at only modest cost: end-to-end latency from *TLS to the recursive resolver is only about 9% slower* when UDP is used to the authoritative server, and 22% slower with TCP to the authoritative. With diverse traces we show that connection reuse can be frequent (60–95% for stub and recursive resolvers, although half that for authoritative servers), and after connection establishment, experiments show that TCP and TLS latency is equivalent to UDP. With conservative timeouts (20 s at authoritative servers and 60 s elsewhere) and estimated per-connection memory, we show that *server memory requirements match current hardware*: a large recursive resolver may have 24k active connections requiring about 3.6 GB additional RAM. Good performance requires key design and implementation decisions we identify: query pipelining, out-of-order responses, TCP fast-open and TLS connection resumption, and plausible timeouts.**

I. Introduction

The Domain Name System (DNS) is the canonical example of a simple request-response protocol. DNS resolves domain names like www.iana.org into the IP addresses; rendering a single web page may require resolving several domain names, so it is desirable to minimize the latency of each query [12]. Requests and responses are typically small (originally required to be less than 512 B, and today under 1500 B as a practical matter), so a single-packet request is usually answered with a single-packet reply over UDP. Simplicity and efficiency has prompted DNS use in broader applications [78].

DNS standards have always required support for TCP, but it has been seen as a poor relative—necessary for large exchanges between servers, but otherwise discouraged. TCP is more expensive than UDP, since connection setup adds latency with additional packet exchanges, and tracking connections requires memory and computation at the server. Why create a connection if a two-packet exchange is sufficient?

This paper makes two contributions. First, we *demonstrate that DNS's connectionless protocol is the cause of a range of fundamental weaknesses in security and privacy that can be addressed by connection-oriented DNS*. Connections have a well understood role in longer-lived protocols such as ssh and HTTP, but DNS's simple, single-packet exchange has been seen as a virtue. We show that it results in weak privacy, denial-of-service (DoS) vulnerabilities, and policy constraints, and that these problems increase as DNS is used in new applications, and concerns about Internet safety and privacy grow. While prior problems have been discussed in isolation (for example, [9], [66]) and individual problems can often be worked around, taken together they prompt revisiting assumptions. We then propose T-DNS, where DNS requests should use TCP by default (not as last resort), and DNS requests from end-users should use Transport-Layer Security (TLS [21]). TCP prevents denial-of-service (DoS) amplification against others, reduces the effects of DoS on the server, and simplifies policy choices about DNSSEC key size, and that TLS protects queries from eavesdroppers to the recursive resolver.

Our second contribution is to show that *the benefits of connection-oriented DNS in T-DNS come at only modest cost*: For *clients*, end-to-end latency of T-DNS (time from a stub's request to an answer, considering all queries and caches) is only moderately more than connectionless DNS. Our models show latency increases by only 9% for TLS vs UDP-only where TLS is used just from stub to recursive-resolver, and it increases by 22% when we add TCP from recursive to authoritative. Connection reuse results in latencies almost the same as UDP once the connection is established. With moderate timeouts (20 s at authoritative servers and 60 s elsewhere), connection reuse is high for servers (85–98%), amortizing setup costs for client and server. Connection reuse for clients is lower (60–80% at the edge, but 20–40% at the root), but still results in amortized costs and lowered latencies. For *servers*, connection rates are viable for modest server-class hardware today. With conservative timeouts (20 s at authoritative servers and 60 s elsewhere) and overestimates of per-connection memory, a large recursive resolver may have 24k active connections using about 3.6 GB of RAM; authoritative servers double those needs.

TCP and TLS are well established protocols, and many DNS variations have been proposed, with TCP in the original specification, and prior proposals to use TLS, DTLS, SCTP,

and HTTP with XML or JSON. Our contribution is not protocol novelty, but *a careful evaluation of what is necessary to add established protocols to an existing ecosystem*: evaluation that shows the performance costs are modest and experiments that show the security and privacy benefits are real. With wide belief that connectionless DNS is mandatory for adequate performance, this study addresses a primary impediment to improving DNS privacy. While we evaluate our specific design, we suggest that our performance evaluation *generalizes* to most connection-like approaches to DNS, nearly all of which require some state at both ends. In addition, we identify the specific implementation choices needed to get good performance with TCP and TLS; alternative protocols for DNS encryption will require similar optimizations, and we suggest they will see similar performance.

Why: Connection-based communication is important to improve security in three ways. First, it *improves DNS privacy through the use of encryption*. We discuss alternatives in § VII-D: although some employ UDP, all effectively build connections at the application-layer to keep session keys and manage setup. DNS traffic is important to protect because hostnames are richer than already visible IP addresses and DNS queries expose application information (§ II-B3). DNS queries are increasingly vulnerable, with wireless networks, growth of third-party DNS (OpenDNS since 2006 [59] and Google Public DNS since 2009 [63]), meaning that end-user requests often cross several networks and are at risk of eavesdropping. Prior work has suggested from-scratch approaches [57], [20], [80]; we instead utilize existing standards to provide confidentiality for DNS, and demonstrate only moderate performance costs. As a side-effect, T-DNS also protects DNS queries from tampering over parts of their path.

Second, TCP *reduces the impact of denial-of-service (DoS) attacks* in several ways. Its connection establishment forces both sides of the conversation to prove their existence, and it has well-established methods to tolerate DoS attacks [26]. Lack of these methods has allowed UDP-based DNS to be exploited by attackers with amplification attacks; an anonymous attacker who spoofs addresses through a DNS server can achieve a 20× increase in traffic to its victim, a critical component of recent multi-Gb/s DoS attacks [3]. We examine performance under attack in § V.

Finally, *UDP limits on reply sizes constrains key sizes and DNS applications*. EDNS0 [18] often makes 4096 B replies possible, extending the original 512 B limit [53]. However, due to IP fragmentation [18], 1500 B is seen as an operational constraint and this limit has repeatedly affected policy choices in DNS security and applications. IP fragmentation presents several dangers: fragments require a resend-all loss recovery [42], about 8% of middleboxes (firewalls) block all fragments [79], and fragmentation is one component in a class of recently discovered attacks [32]. Of course current DNS replies strive to fit within current limits [77], but DNSSEC keys approaching 2048-bits lead to fragmentation, particularly during key rollover (§ II-B1). Finally, DNSSEC's guarantees make it attractive for new protocols with large replies, but new applications will be preempted if DNS remains limited to short replies.

How: On the surface, connection-oriented DNS seems untenable, since TCP setup requires an extra round-trip and state on servers. TCP is seen as bad for DNS, and so TLS' heavier weight handshake is impossible.

Fortunately, we show that *connection persistence*, reusing the same connection for multiple requests, amortizes connection setup. We identify the key design and implementation decisions needed to minimize overhead—query pipelining, out-of-order responses, TCP fast open and TLS connection resumption, shifting state to clients when possible. Combined with persistent connections with conservative timeouts, these optimizations balance end-to-end latency and server load.

Our key results are to show that T-DNS is feasible and that it provides a clean solution to a broad range of DNS problems across privacy, security, and operations. We support these claims with end-to-end models driven by analysis of day-long traces from three different types of servers and experimental evaluation of prototypes

II. Problem Statement

We next briefly review today's DNS architecture, the specific problems we aim to solve, and our threat model.

II-A Background

DNS is a protocol for resolving *domain names* to different *resource records* in a globally distributed database. A *client* makes a *query* to a *server* that provides a *response* of a few dozen specific types. Domain names are hierarchical with multiple *components*. The database has a common root and millions of independent servers.

Originally DNS was designed to map domain names to IP addresses. Its success as a lightweight, well understood key-to-value mapping protocol caused its role to quickly grow to other Internet-related applications [78], including host integrity identification for anti-spam measures and replica selection in content-delivery networks [13]. Recently DNS's trust framework (DNSSEC) has been used to complement and extend traditional PKI/Certificate Authorities for e-mail [29] and TLS [33].

Protocols: DNS has always run over both connectionless UDP and connection-oriented TCP transport protocols. UDP has always been preferred, with TCP used primarily for zone transfers to replicate portions of the database, kilobytes or more in size, across different servers. Responses larger than advertised limits are truncated, prompting clients to retry with TCP [76]. UDP can support large packets with IP fragmentation, at the cost of new problems discussed below.

The integrity of DNS data is protected by DNSSEC [4]. DNSSEC provides cryptographic integrity checking of positive and negative DNS replies, but not privacy. Since July 2010 the root zone has been signed, providing a root of trust through signed sub-domains.

As a Distributed System: DNS resolvers have both client and server components. Resolvers typically take three roles:

stub, recursive, authoritative. *Stub resolvers* are clients that talk only to recursive resolvers, which handle name resolution. Stubs typically send to one or a few recursive resolvers, with configuration automated through DHCP [23] or by hand.

Recursive resolvers operate both as servers for stubs and clients to authoritative servers. Recursive resolvers work on behalf of stubs to iterate through each of the several components in a typical domain name, contacting one or more authoritative servers as necessary to provide a final answer to the stub. Much of the tree is stable and some is frequently used, so recursive resolvers *cache* results, reusing them over their *time-to-live*.

Authoritative servers provide answers for specific parts of the namespace (a *zone*). Replication between authoritative peers is supported through zone transfers with notifications and periodic serial number inquiries.

This three-level description of DNS is sufficient to discuss protocol performance for this paper. We omit both design and implementation details that are not relevant to our discussion. The complexity of implementations varies greatly [68]; we describe some aspects of one operator's implementation in § IV-A.

II-B The Limitations of Single-Packet Exchange

Our goal is to remove the limitations caused by optimizing DNS around a single-packet exchange as summarized in Table I. We consider transition in § III-D.

II-B1 Avoiding Arbitrary Limits to Response Size: Limitation in payload size is an increasing problem as DNS evolves to improve security. Without EDNS [18], UDP DNS messages are limited to 512 B. With EDNS, clients and servers may increase this limit (4096 B is typical), although this can lead to fragmentation which raises its own problems [42]. Due to problematic middleboxes, clients must be prepared to fall back to 512 B, or resend the query by TCP. Evidence suggests that 5% [79] or 2.6% [35] of users find TCP impeded. Such work-arounds are often fragile and the complexities of incomplete replies can be a source of bugs and security problems [32].

Evolution of DNS and deployment of DNSSEC have pushed reply sizes larger. We studied Alexa top-1000 websites, finding that 75% have replies that are at least 738 B (data is in [86] due to space).

With increasingly larger DNS replies (for example, from longer DNSSEC keys), IP-level fragmentation becomes a risk in many or all replies. To quantify this problem, Figure 1 examines a 10-minute trace with 13.5M DNSSEC enabled responses of one server for .com. Over this real-world trace we model the effects of different key sizes by replacing current 1024-bit RSA signatures with longer ones. We model regular operation for several key sizes, showing CDFs for the size of all responses, and dots for negative responses (medians for NXD; quartiles are within 1% and so are omitted) using NSEC3 [45], and DNSKEY replies for several sizes of KSK (each row) and ZSK (different shapes, exact values).

Figure 1 shows that with a 2048-bit ZSK, 5% of DNSSEC responses and almost *all* NXDomain responses, and some

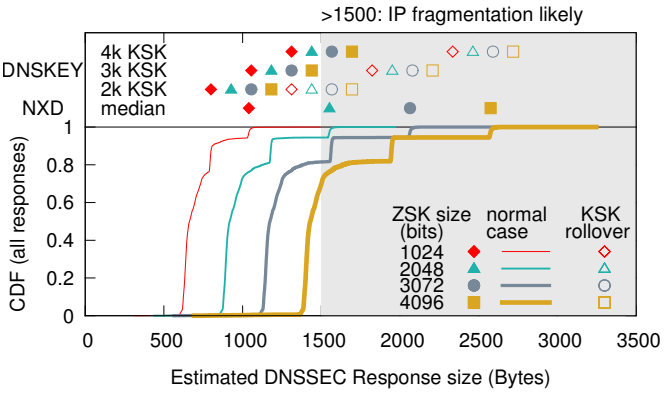


Fig. 1: Estimated response sizes with different length DNSSEC keys. Dots show sizes for DNSKEY and median for NXDomain replies. (Data: trace and modeling)

DNSKEYs during rollover will suffer IP fragmentation (shown in the shaded region above 1500 B).

This evaluation supports our claim that connectionless transport *distorts current operational and security policies*. Worries about fragmentation have contributed to delay and concern about key rollover and use of 2048-bit keys. More importantly, other designs have been dismissed because of reply sizes, such as proposals to decentralize signing authority for the DNS root which might lead to requiring TCP for root resolution [72]. For some, this requirement for TCP is seen as a significant technical barrier forcing use of shorter keys or limitations of algorithms.

Finally, size can also *preempt future DNS applications*. Recent work has explored the use of DNS for managing trust relationships (for example [60]), so one might ask how DNS would be used if these constraints to response size were removed. We examine the PGP web of trust [62] as a trust ecosystem that is unconstrained by packet sizes. Rather than a hierarchy, key authentication PGP builds a mesh of signatures, so 20% of keys show 10 or more signatures, and well connected keys are essential to connecting the graph. PGP public keys with 4 signatures exceeds 4kB, and about 40% of keys have 4 signatures or more [62]. If DNS either grows to consider non-hierarchical trust, or if it is simply used to store such information [82], larger replies will be important.

T-DNS's use of TCP replaces IP-level fragmentation with TCP's robust methods for retry and bytestream.

II-B2 Need for Sender Validation: Uncertainty about the source address of senders is a problem that affects both DNS servers and others on the Internet. Today source IP addresses are easy to spoof, allowing botnets to mount denial-of-service (DoS) attacks on DNS servers directly [36], [69], and to leverage DNS servers as part of an attack on a third party through a DNS Amplification attack [74], [48].

Work-arounds to DNS's role in DoS attacks exist. Many anti-spoofing mechanisms have been proposed, and DNS servers are able to rate-limit replies. T-DNS would greatly reduce the vulnerability of DNS to DoS and as DoS lever-

problem	**current DNS**	**with T-DNS** (why)
packet size limitations	guarantee: 512 B, typical: 1500 B	64 kB
source spoofing	spoof-detection depends on source ISP	most cost pushed back to spoofer (SYN cookies in TCP)
privacy (stub-to-recursive)	vulnerable to eavesdropping	privacy (from TLS encryption)
(recursive-to-authoritative)	aggregation at recursive	aggregation, or optional TLS

Table I: Benefits of T-DNS.

age against others. Well established techniques protect DNS servers from TCP-based DoS attacks [26], [71], and TCP's connection establishment precludes source address spoofing, eliminating amplification attacks.

We do not have data to quantify the number of DNS amplification attacks. However, measurements of source-IP spoofing shows that the number of networks that allow spoofing has been fairly steady for six years [7]. Recent measurement of distributed reflective denial-of-service (DRDoS) shows the majority of the attacks involve DNS amplification [66]. Recent reports of DoS show that DNS amplification is a serious problem, particularly in the largest attacks [3]. T-DNS suggests a long-term path to reduce this risk.

Even if TCP reduces DoS attacks, we must ensure it does not create new risks, as we show experimentally in § V. Fortunately TCP security is well studied due to the web ecosystem. We describe our approaches to DoS above, and most other known attacks have defenses. A more detailed list of TCP-specific attacks that do not apply is in our technical report [86].

II-B3 Need for DNS Privacy: Lack of protection for query privacy is the final problem. Traditionally, privacy of Internet traffic has not been seen as critical. However, recent trends in DNS use, deployment and documentation of widespread eavesdropping increase the need for query privacy [9]. First, end-user queries are increasingly exposed to possible eavesdropping, through use of third-party DNS services such as OpenDNS and Google Public DNS, and through access on open networks such as WiFi hotspots. Second, presence of widespread eavesdropping and misdirection is now well documented, for government espionage [31], censorship [2], and criminal gain [51]. Finally, ISPs have recognized the opportunity to monetize DNS typos, redirecting non-existing domain responses (NXDOMAIN hijacking), a practice widespread since 2009 (for example [52]). For both corporate or national observation or interference, we suggest that one must follow the policies of one's provider and obey the laws of one's country, but we see value in making those policies explicit by requiring interaction with the operator of the configured recursive name server, rather than making passive observation easy.

DNS is also important to keep private because it is used for many services. While protecting queries for IP addresses may seem unnecessary if the IP addresses will then immediately appear in an open IP header, full domain-names provide information well beyond just the IP address. For web services provided by shared clouds, the domain name is critical since IP addresses are shared across many services. DNS is also used for many things other than translating names to IP addresses: one example is anti-spam services where DNS maps e-mail senders to reputation, exposing some e-mail sources via DNS [46].

Although DNS privacy issues are growing, most DNS security concerns have focused on the *integrity* of DNS replies, out of fear of reply modification. The integrity of DNS replies has been largely solved by DNSSEC which provides end-to-end integrity checks.

II-C Threat Model

To understand security aspects of these problems we next define our threat model.

For *DoS attacks* exploiting spoofed source addresses, our adversary can send to the 30M currently existing open, recursive resolvers that lack ingress filtering [50].

For *query eavesdropping* and attacks on privacy, we assume an adversary with network access on the network between the user and the recursive resolver. We assume aggregation and caching at the recursive resolver provide effective anonymization to authoritative servers; if not it could enable TLS.

We also assume the operator of the recursive resolver is trusted. Although outside the scope of this paper, this requirement can be relaxed by alternating requests across several DNS providers, implementing a mix network shuffling requests from multiple users, or padding the request stream with fake queries. Similarly, privacy attacks using cache timing are outside our scope, but solved by request padding [37].

For *fragmentation attacks* due to limited packet size, we assume an off-path adversary that can inject packets with spoofed source addresses, following Herzberg and Schulman [32].

Other attacks on query integrity are already largely prevented by DNSSEC and so they are outside the scope of this paper. (T-DNS augments DNSSEC, it is not intended to replace it.)

We depend on existing mechanisms to avoid person-in-the-middle attacks on T-DNS setup of TLS as discussed in § III-B. Concurrent with our work, Shulman identified information leakage in encrypted DNS [70]. This paper seeks to close the *primary* channel; we recognize side channels remain.

T-DNS clients may set their own policy for handling *downgrade attacks*, where a request for privacy is declined. An adversary in control of the network can interfere with TLS negotiation, preventing its use. A conservative client may retry other servers or refuse to provide non-private DNS, or it may alert the user.

	system			
feature	**T-DNS**	**unbound**	**DNSCrypt**	**DNSCurve**
signaling	in-band	implicit	implicit	per-query
protocol/port	TCP/53	TCP/443	TCP/443	UDP/53
encryption	negotiable from TLS		Curve25519	
stub/recursive	yes	yes	yes	no
recursive/authoritative	yes	no	no	yes
pipelining	yes	no*	from UDP	
out-of-order replies	yes	no*	from UDP	
TCP Fast Open	yes	no	n/a	n/a
TLS resumption	yes	no	n/a	n/a

Table II: Design choices in T-DNS as compared to alternatives.

III. Design and Implementation of T-DNS

Table II lists design choices for T-DNS; next we describe in-band TLS negotiation (our protocol addition), and implementation choices that improve performance (shown in § VI). These design choice are critical to amortize the cost of connections.

III-A DNS over TCP

Design of DNS support for TCP was in the original specification [53] with later clarifications [5]. However, *implementations* of DNS-over-TCP have been underdeveloped because it is not seen today as the common case. We consider three implementation decisions, two required to to make TCP performance approach UDP.

Pipelining is sending multiple queries before their responses arrive. It is essential to avoid round-trip delays that would occur with the stop-and-wait alternative. Batches of queries are common: recursive resolvers with many clients have many outstanding requests to popular servers, such as that for .com. End-users often have multiple names to resolve, since most web pages draw resources from multiple domain names. We examined 40M web pages (about 1.4% of CommonCrawl-002 [30]) to confirm that 62% of web pages have 4 or more unique domain names, and 32% have 10 or more.

Support for *receiving* pipelined requests over TCP exists in `bind` and `unbound`. However neither *sends* TCP unless forced to by indication of reply truncation in UDP; and although explicitly allowed, we know of *no* widely used client that sends multiple requests over TCP. Our custom stub resolver supports pipelining, and we are working to bring T-DNS to the `getdns` resolver.

Out-of-order processing (OOOP) at recursive resolvers is another important optimization to avoid head-of-line blocking. TCP imposes an order on incoming queries; OOOP means replies can be in a different order, as defined and explicitly allowed by RFC-5966 [5]. Without OOOP, queries to even a small percentage of distant servers will stall a strictly-ordered queue, unnecessarily delaying all subsequent queries. (For UDP, absence of connections means all prominent DNS servers naturally handle queries with OOOP.)

We know of no DNS server today that supports out-of-order processing of TCP queries. Both `bind` and `unbound` instead resolve each query for a TCP connection before considering the next. We have implemented out-of-order processing in our DNS proxy (converting incoming TLS queries back to UDP at the server), and have a prototype implementation in `unbound`.

Finally, when possible, we wish to **shift state from server to client**. Per-client state accumulates in servers with many connections, as observed in the TIME-WAIT state overheads due to closed TCP connections previously observed in web servers [28]. Shifting TCP state with DNS is currently being standardized [83].

These implementation details are important not only to DNS; their importance has been recognized before in HTTP [54], [28]. HTTP/1.1 supports only pipelining, but both are possible in DNS and proposed HTTP/2 [56].

III-B DNS over TLS

TLS for DNS builds on TCP, with new decisions about trust, negotiation, and implementation choices.

III-B1 Grounding Trust: TLS depends on public-key cryptography to establish session keys to secure each connection and prevent person-in-the middle attacks [21]. DNS servers must be given TLS certificates, available today from many sources at little or no cost.

Client trust follows one of several current practices. We prefer DANE/TLSA to leverage the DNSSEC chain of trust [33], but other alternatives are the current public-key infrastructures (PKI) or trusted Certificate Authorities (CAs) provided out-of-band (such as from one's OS vendor or company). To avoid circular dependencies between T-DNS and DANE, one may bootstrap T-DNS's initial TLS certificate through external means (mentioned above) or with DANE without privacy.

III-B2 Upwards TLS Negotiation: T-DNS must negotiate the use of TLS. Earlier protocols selected TLS with separate ports, but IETF now encourages in-protocol upgrade to TLS to reduce port usage; this approach is the current preference for many protocols (IMAP, POP3, SMTP, FTP, XMPP, LDAP, and NNTP, although most of these do have legacy, IANA-allocated, but not RFC-standardized, ports to indicate TLS, XMPP, the most recent, being an exception). to indicate TLS). We therefore propose a new EDNS0 extension [18] to negotiate the use of TLS. We summarize our proposal below and have provided a formal specification elsewhere [34].

Our negotiation mechanism uses a new "TLS OK" (TO) bit in the extended flags of the EDNS0 OPT record. A client requests TLS by setting this bit in a DNS query. A server that supports TLS responds with this bit set, then both client and server carry out a TLS handshake [21]. The TLS handshake generates a unique session key that protects subsequent, normal DNS queries from eavesdropping over the connection.

The DNS query made to start TLS negotiation obviously is sent without TLS encryption and so should not disclose information. We recommend a distinguished query with name "STARTTLS", type TXT, class CH, analogous to current support queries [81].

Once TLS is negotiated, the client and server should retain the TLS-enabled TCP connection for subsequent requests. Either can close connections after moderate idle periods (evaluated in § IV), or if resource-constrained.

III-B3 Implementation Optimizations: Two implementation choices improve performance. TLS **connection resumption** allows the server to give all state needed to securely re-create a TLS connection to the client [67]. This mechanism allows a busy server to discard state, yet an intermittently active client can regenerate that state more quickly than a full, fresh TLS negotiation. A full TLS handshake requires three round-trip exchanges (one for TCP and two for TLS); TLS resumption reduces this cost to two RTTs, and reduces server computation by reusing the master secret and ciphersuite. Experimentally we see that resumption is $10\times$ faster than a new connection (§ VI-A).

TLS close notify allows one party to request the other to close the connection. We use this mechanism to shift TCP TIME-WAIT management to the client.

III-C Implementation Status

We have several implementations of these protocols. Our primary client implementation is a custom client resolver that we use for performance testing. This client implements all protocol options discussed here and uses either the OpenSSL or GnuTLS libraries. We also have some functionality in a version of `dig`.

We have three server implementations. Our primary implementation is in a new DNS proxy server. It provides a minimally invasive approach that allows us to test any recursive resolver. It receives queries with all of the options described here, then sends them to the real recursive resolver via UDP. When the proxy and real resolver are on the same machine or same LAN we can employ unfragmented 9 kB UDP packets, avoid size limitations and exploiting existing OOOP for UDP. It uses either the OpenSSL or GnuTLS libraries.

In the long run we expect to integrate our methods into existing resolvers. We have implemented subsets of our approach in BIND-9.9.3 and unbound-1.4.21.

III-D Gradual Deployment

Given the huge deployed base of DNS clients and servers and the complexity of some implementations [79], any modifications to DNS will take effect gradually and those who incur cost must also enjoy benefits. We discuss deployment in detail elsewhere [86] since the length of full discussion forces it outside the scope of this paper, but we summarize here.

T-DNS deployment is *technically feasible* because our changes are backwards compatible with current DNS deployments. TLS negotiation is designed to disable itself when either the client or server is unaware, or if a middlebox prevents communication. Approaches analogous to DNSSEC-trigger [55] may be used to bootstrap through temporarily interfering middleboxes, and can report long-term interference, prompting middlebox replacement or perhaps circumvention using a different port. In the meantime, individuals may select between immediately correcting the problem or operating with DNS privacy. DNS already supports TCP, so clients and servers can upgrade independently and will get better performance with our implementation guidelines. T-DNS benefits from TCP extensions like fast-open that are only very recently standardized [16], so T-DNS performance depends their deployment (Fast Open is in Linux-3.7 since December 2012). Gradual deployment does no harm; as clients and servers upgrade, privacy becomes an option and performance for large responses improves.

Motivation for deployment stems from T-DNS's privacy and DoS-mitigation. Some users today want greater privacy, making it a feature ISPs or public DNS-operators can promote. The DoS-mitigation effects of TCP allows DNS operators to reduce their amount of capacity overprovisioning to handle DoS. T-DNS's policy benefits from size require widespread adoption of TCP, but the penalty of slow adoption is primarily lower performance, so complete deployment is not necessary.

T-DNS deployment is feasible and motivations exist for deployment, but the need for changes to hardware and software suggests that much deployment will likely follow the natural hardware refresh cycle.

dataset	date	client IPs	records
DNSChanger	2011-11-15		
all-to-one		15k	19M
all-to-all		692k	964M
DITL/Level 3	2012-04-18		
cns4.lax1		282k	781M
cns[1-4].lax1		655k	2412M
DITL/B-root	2013-05-29	3118k	1182M

Table III: Datasets used to evaluate connection reuse and concurrent connections. Each is 24 hours long.

IV. Connection Reuse and Resources

Connection reuse is important for T-DNS performance to amortize setup over multiple queries (§ VI). Reuse poses a fundamental trade-off: with plentiful resources and strict latency needs, clients prefer long-lived connections. But servers share resources over many clients and prefer short-lived connections.

We next examine this trade-off, varying connection timeout to measure the *connection hit fraction*, how often an existing connection can be reused without setup, and *concurrent connections*, how many connections are active on a server at any time. We relate active connections to server resource use.

IV-A Datasets

We use three different datasets (Table III) to stand in for stub clients, recursive resolvers, and authoritative servers in our analysis. These datasets are derived from server logging (Level3) or packet capture (the others). While more data is always better, we believe our data captures very diverse conditions and more data is very unlikely to change the conclusions.

DNSChanger: DNSChanger is a malware that redirects end-users' DNS resolvers to a third party so they could inject advertising. This dataset was collected by the working group that, under government authority, operated replacement recursive resolvers while owners of infected computers were informed [51]. It includes timing of all queries from end-user

IP addresses with this malware as observed at the working group's recursive resolvers. We use this dataset to represent stub-to-recursive traffic, and select traffic to the busiest server (all-to-one) in § IV-C and the traffic from all sources to all servers (all-to-all) in § VI-D. (We know of no public sources of stub-to-recursive data due to privacy concerns.)

DITL/Level 3: Level 3 operates DNS service for their customers, and also as an open public resolver [64]. Their infrastructure supports 9 sites, each with around 4 front-end recursive resolvers, each load-balanced across around 8 back-end resolvers, as verified by the operators. We use their 48-hour trace hosted by DNS-OARC [22].

We examine two subsets of this data. We first select a random site (lax1, although we confirmed other sites give similar results). Most client IP addresses (89%) access only one site, so we expect to see all traffic for each client in the dataset (`cns[1-4].lax1`). Many clients (75%) only access one front-end at a site, so we select the busiest front-end at this site (`cns4.lax1`) to provide a representative smaller (but still large) subset. We use these Level 3 traces to represent a recursive resolver.

DITL/B-Root: This dataset was collected at the B-Root nameserver as part of DITL-2013 and is also provided through DNS-OARC. We selected B-Root because at the time of this collection it did not use anycast, so this dataset captures all traffic into one root DNS instance. (Although as one of 13 instances it is only a fraction of total root traffic.) We use this traffic to represent an authoritative server, since commercial authoritative server data is not generally accessible.

Generality: These datasets cover each class of DNS resolver (§ II-A) and so span the range of behavior in different parts of the DNS system and evaluate our design. However, each dataset is unique. We do not claim that any represents *all* servers of that class, and we are aware of quirks in each dataset. In addition, we treat each source IP address as a computer; NAT may make our analysis optimistic, although this choice is correct for home routers with DNS proxies.

IV-B Trace Replay and Parameterization

To evaluate connection hits for different timeout windows we replay these datasets through a simple simulator. We simulate an adjustable timeout window from 10 to 480 s, and track active connections to determine the number of concurrent connections and the fraction of connection hits. We ignore the first 10 minutes of trace replay to avoid transient effects due to a cold cache.

We convert the number of concurrent connections to hardware memory requirements using two estimates. First, we measure memory experimentally idle TCP connections by opening 10k simultaneous connections to `unbound` and measuring peak heap size with `valgrind`. On a 64-bit x86 computer running Fedora 18, we estimate TCP connection at 260 kB, and each TLS connection at 264 kB; to this we estimate about 100 kB kernel memory, yielding 360 kB as a very loose upper bound. Second, Google transitioned gmail to TLS with no additional hardware through careful optimizations, reporting 10 kB memory per connection with minimal CPU cost due to TLS [44]. Based on their publicly available optimizations, we use a conservative 150 kB as the per connection memory cost.

IV-C Concurrent Connections and Hit Fraction

Trace replay of the three datasets provides several observations. First we consider how usage changes over the course of the day, and we find that variation in the number of active connections is surprisingly small. When we measure counts over one-second intervals, connections vary by $\pm 10\%$ for Level 3, with slightly more variation for DNSChanger and less for B-Root (graphs omitted due to space). Connection hit fractions are even more stable, varying by only a few percent. Given this stability, Figure 2 summarizes usage with medians and quartiles.

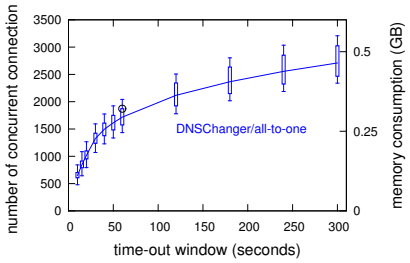
The three servers have very different absolute numbers of active connections, consistent with their client populations. All servers show asymptotic hit fractions with diminishing benefits beyond timeouts of around 100 s (Figure 2c). The asymptote varies by server: with a 120 s window, DNSChanger is at 97-98%, Level 3 at 98-99%, and B-Root at 94-96%. These fractions show that *connection caching will be very successful*. Since much network traffic is bursty, it is not surprising that caching is effective. We believe the lower hit fraction at B-Root is due to its diverse client population and its offering a relatively small zone; we expect similar results for other static DNS zones.

Recommendations: We propose timeouts of 60 s for recursive resolvers and 20 s for authoritative servers, informed by Figure 2, with a conservative approach to server load. We recommend that clients and servers not preemptively close connections, but instead maintain them for as long as they have resources. Of course, timeouts are ultimately at the discretion of the DNS operator who can experiment independently.

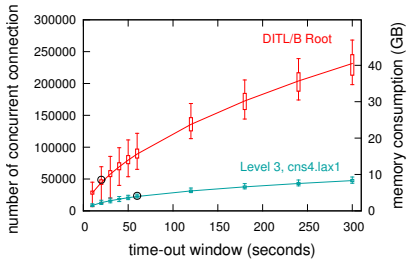
These recommendations imply server memory requirements. With 60 s and 20 s timeouts for recursive and authoritative, each DNSChanger needs 0.3 GB RAM (2k connections), Level 3 3.6 GB (24k connections), and B-Root 7.4 GB (49k connections), based on the 75%iles in Figure 2, for both user and kernel memory with some optimization, in addition to memory for actual DNS data. These values are well within current, commodity server hardware. With Moore's law, memory is growing faster than root DNS traffic (as seen in DITL [15]), so future deployment will be even easier. Older servers with limited memory may instead set a small timeout and depend on clients to use TCP Fast Open and TLS Resume to quickly restart terminated connections.

V. Performance Under Attack

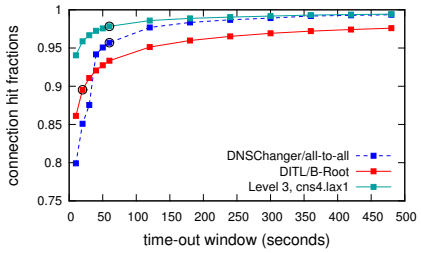
We next consider the role of DNS in denial-of-service attacks: first DNS's role in attacking others through amplification attacks, then the performance of a DNS server itself under attack. In both cases we show that TCP mitigates the problem, and that TLS does not make things worse.



(a) Median and quartiles of numbers concurrent connections. Dataset: DNSChanger



(b) Median and quartiles of numbers concurrent connections. Datasets: Level 3/cns4.lax1 and B-Root



(c) Median connection hit fractions, taken server-side. (Quartiles omitted since always less than 1%.)

Fig. 2: Evaluation of concurrent connections and connection hit fractions. Black circles show design point.

action	DNS (UDP)	TCP-SYNs no cookies	TCP-SYNs w/cookies
query	82 B	76 B	76 B
reply	200–4096 B	66–360 B	66 B
amplification	3–40×	1–6×	1×

Table IV: Amplification factors of DNS/UDP and TCP with and without SYN cookies.

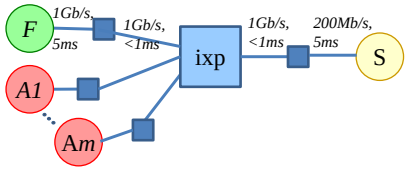


Fig. 3: Network topology for DoS attack evaluation: legitimate (F), attackers (A), and server (S).

attacker protocol	**attacker** src IP	**attacker** cookies	**foreground** protocol	**foreground** resource limit
UDP	spoofed	n/a	UDP	CPU
TCP	spoofed	no	TCP	TCP control buffers
TCP	spoofed	yes	TCP	TCP control buffers
TCP	real	yes	TCP	TCP control buffers

Table V: Limited resource for each protocol combination in tested DoS attacks.

V-A DNS: Amplifying Attacks on Others

Recently, amplification attacks use DNS servers to magnify attack effects against others [74], [48]. An attacker's botnet spoofs traffic with a source address of the victim, and the DNS server amplifies a short query into a large reply.

Table IV shows our measurements of amplification factors of three classes of attacks: DNS over UDP, and DNS over TCP without and with SYN cookies. DNS allows an attacker to turn a short UDP request into a large UDP reply, amplifying the attack by a factor of up to 40. TCP can amplify an attack as well, since a single SYN can solicit multiple SYN-ACKs attempts [43], but only by a factor of 6. With SYN cookies, TCP does not retransmit SYN-ACKs, so there is no amplification for the attacker.

DoS-prevention also requires rate limiting, which can help defuse UDP-based amplification. Such rate limiting will be important during a potential transition from UDP to TCP for DNS: wide use of TCP can allow more aggressive rate limits for TCP, as we show in § V-B, and partial use of TCP can allow more aggressive rate limiting, as we discuss next.

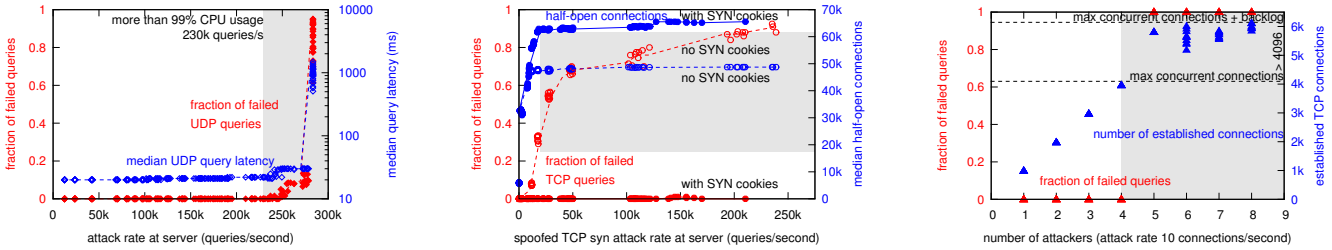
We conclude that, although TCP does not eliminate DoS attacks, full use of TCP eliminates amplification of those attacks, and partial use of TCP allows more aggressive rate limiting during transition.

V-B Direct Denial-of-Service on the DNS Server

We next consider UDP and TCP attacks designed to overwhelm the DNS server itself. While some DoS attacks overwhelm link bandwidth, UDP attacks on DNS often target server CPU usage, and TCP attacks overwhelm OS-limits on active connections. Current DNS operators greatly overprovision to absorb attacks, with best-practices recommending a factor of three [11]. We next aim to show that UDP attacks are a threat, and attacks on a naive TCP service are deadly, but a TCP service using TCP SYN cookies forces attackers to use far more resources than today.

To evaluate a DoS attack, we deploy the network shown in Figure 3 in the DETER testbed. We send foreground traffic from *F* to a DNS server *S*, then evaluate the effects of attack traffic (*A1* to *Am*) sent to the server. The traffic merges at a router (IXP, an Internet Exchange Point) and is sent to the server behind a bottleneck link. The server hosts a DNS domain with 6.5M names in `example.com`, and the attacker queries random names that exist in this domain. The server is a single-core 2.4 GHz Intel Xeon running Linux Ubuntu-14.04 (64-bit). This scenario (hardware and network speeds) represents a scaled down version of a typical deployment.

We compare several combinations of protocols for attacker and legitimate, foreground traffic (Table V). We focus on all-UDP traffic and three cases of all-TCP use to compare current DNS with proposed TCP deployments. While the future will include both TCP and UDP usage, these two "pure" cases show the limits. We use NSD-4.1.0 as our DNS server, with the OS and application configured to support either 65k or 4k TCP sessions. Since we are only testing connection initiation, we do



(a) UDP-based DNS performance under DoS attack.

(b) TCP-based DNS performance under spoofed DoS attack, with (filled circles) and without (empty circles) SYN cookies.

(c) TCP-based DNS performance with non-spoofed DoS attack.

Fig. 4: Evaluation of denial-of-service attacks on DNS servers. Dataset: testbed experiment

not use persistent connections or TCP fast-open. Foreground traffic is sent with the `dig` program. Attack traffic uses a custom-written UDP/TCP flooder or `hping3`; we vary the number of attackers and measure the observed attack rate. For each kind of attack scenario, we repeat the experiment for 10 times.

UDP-based DNS under attack: First we consider a current DNS where all traffic is UDP. A UDP receiver cannot verify source addresses, so an attacker will spoof query source addresses to defeat source-based rate-limiting or filtering. Servers must reply to both foreground and attack queries; the attacker exploits this to exhaust either host processing power or network capacity in the reverse path. In the experiment, two attackers can easily DoS the server if not network-limited (we pace the attackers to study a range of rates). In Figure 4a we see that our server handles about 230k queries/s at full capacity (CPU limited in the gray region). Both reply latency for completed queries (dashed blue line and right axis) and the number of unanswered queries (solid red line and left axis) rise dramatically under overload. These results are typical of DNS servers under DoS attacks that are not limited by network speed; they show why robust DNS operations are heavily overprovisioned.

DNS-over-TCP under attack: Next we consider three variations of a TCP SYN-flood attack in Figure 4b. Here, the attacker's goal is to exhaust the number of available TCP connections on the server (right axis), resulting in unanswered queries (left axis).

First, we allow the attacker to spoof source addresses and operate the server without SYN cookies. By default, half-open connections persist for tens of seconds, and legitimate queries must compete with attack traffic for connection slots. In Figure 4b, without SYN cookies on the server (lines with empty circles), a single attacker can easily send 60k SYN/s and consume all possible TCP connections on the server, resulting in 70% of foreground queries being dropped. With SYN cookies (lines with filled circles), all state is pushed back on the sender, so *attack traffic consumes no memory at the server* and *no* foreground replies are lost.

Finally we consider a TCP SYN-flood attack without spoofed addresses. A wise server will use SYN cookies to prevent spoofed attacks, and will rate-limit new connections to bound non-spoofed attacks. If we rate-limit to 10 new connections/s per IP address (for DNS with p-TCP, there should never be more than 1 active connection per IP), and the server has 60k TCP connection slots, then it requires 6k attackers to fully consume the server's connections. In experiment we test a scaled-down version of this scenario: attackers are limited to 10 connection/s and the server supports 4096 active connections. Figure 4c shows the outcome: 5 attackers are required to consume most connection slots, at which point *all* legitimate traffic is dropped. Although this experiment is scaled down to fit our testbed, full-scale server would support 60k connections or more. With SYN cookies against spoofers and rate limiting to 1 TCP connection per source-IP when under attack, a single server can tolerate thousands of attackers, many times what are required for a UDP attack. Large DNS providers often serve from clusters of machines, requiring even more attackers.

A final threat is attacks on TLS, with the goal of consuming server CPU [6]. Since TLS handshakes are expensive, we expect servers to adopt strict rate limits per source address, perhaps 4 TLS connections/s per IP address. A 2006 study shows that a server with a PIII-933 MHz, dual CPU can handle more than 1000 TLS connections/s with optimizations [17] (we expect this number to be much larger on current hardware), requiring an attacker with 250 machines. Since we require non-spoofed addresses, active filtering to these attacks become possible. *A DoS attacker requires more resources against an all-TLS DNS server* when compared to an all-UDP DNS server. We know that TLS-based websites survive DoS attacks, suggesting TLS-based DNS can as well.

To summarize, we show that TCP with SYN cookies and TLS greatly increase the work factor for an attacker to overwhelm the DNS server itself, compared with UDP. Our experiments use a moderate-size DNS server, but this work-factor increase means it is even harder for attacker to defeat large DNS deployments such as authoritative servers for large zones. With the overhead and performance optimizations we describe, large-size servers should find TCP and TLS both feasible and highly beneficial to the mitigation of DoS attacks.

step	OpenSSL	GnuTLS	DNSCrypt/ DNSCurve
TCP handshake	0.15 ms		none
packet handling	0.12 ms		none
crypto handshake	25.8 ms	8.0 ms	23.2 ms
key exchange	13.0 ms	6.5 ms	—
CA validation	12.8 ms	1.5 ms	—
crypto resumption	1.2 ms	1.4 ms	no support
DNS resolution	0.1–0.5 ms	same	same
crypto	∼1 ms		0.7–1.8 ms

Table VI: Computational costs of connection setup and packet processing.

VI. Client-side Latency

For clients, the primary cost of T-DNS is the additional latency due to connection setup. Using experiments, we next examine stub-to-recursive and recursive-to-authoritative query latency with TCP and TLS, highlighting the effects of pipelining and out-of-order processing. Three parameters affect these results: the *computation time* needed to execute a query, the *client-server RTT*, and the *workload*. We show that RTTs dominate performance, not computation. We study RTTs for both stub-to-recursive and recursive-to-authoritative queries, since the RTT is much larger and more variable in the second case. We consider two workloads: *stop-and-wait*, where each query is sent after the reply for the last is received, and *pipelining*, where the client sends queries as fast as possible. These experiments support modeling of end-to-end latency.

VI-A Computation Costs

We next evaluate CPU consumption of TLS. Our experiments' client and server are 4-core x86-64 CPUs, running Fedora 19 with Linux-3.12.8 over a 1Gb/s Ethernet. We test our own client and the Apache-2.4.6 web-server with GnuTLS and OpenSSL. We also measure the DNSCurve client [49], and the DNSCrypt proxy [58].

We report the median of 10 experimental trials, where each trial is the mean of many repetitions because each event is brief. We measure 10k TCP handshakes, each by setting up and closing a connection. We estimate TCP packet processing by sending 10k full-size packets over an existing connection. We measure TLS connection establishment from 1000 connections, and isolate key exchange from certificate validation by repeating the experiment with CA validation disabled. We measure TLS connection resumption with 1000 trials.

Table VI compares TLS costs: TCP setup and DNS resolution are fast (less than 1 ms). TLS setup is more expensive (8 or 26 ms), although costs of key exchange and validation vary by implementation. We see that TLS resumption is ten times faster than full TLS setup for both OpenSSL and GnuTLS.

We also examine DNSCurve and DNSCrypt cost in Table VI and find similar computation is required for their session key establishment. Their client and server can cache session keys to avoid this computation, but at the expense of keeping server state, just as T-DNS keeps TCP and TLS state. If elliptic curve cryptography has performance or other advantages, we expect it to be added to future TLS protocol suites.

Finally, prior work has reported server rates of 754 uncached SSL connections per second [8]. These connection rates sustain steady state for recursive DNS, and two servers will support steady state for our root server traces. Provisioning for peaks would require additional capacity.

Although TLS is computationally expensive, *TLS computation will not generally limit DNS*. For clients, we show (§ VI-E) that RTT dominates performance, not computation. Most DNS servers today are bandwidth limited and run with very light CPU loads. We expect server memory will be a larger limit than CPU. While our cost estimation is very promising, we are still in the progress of carrying out full-scale experimental evaluation of T-DNS under high load.

VI-B Latency: Stub-to-Recursive Resolver

We next carry out experiments to evaluate the effects of T-DNS on DNS use between stub and both local and public recursive resolvers.

Typical RTTs: We estimate typical stub-to-recursive resolver RTTs in two ways. First, we measure RTTs to the local DNS server and to three third-party DNS services (Google, OpenDNS, and Level3) from 400 PlanetLab nodes. These experiments show ISP-provided resolvers have very low RTT, with 80% less than 3 ms and only 5% more than 20 ms. Third-party resolvers vary more, but anycast keeps RTT moderate: median RTT for Google Public DNS is 23 ms, but 50 ms or higher for the "tail" of 10–25% of stubs; other services are somewhat more distant. Second, studies of home routers show typical RTTs of 5-15 ms [73].

Methodology: To estimate T-DNS performance we experiment with a stub resolver with a nearby (1 ms) and more distant (35 ms) recursive resolver (values chose to represent typical extremes observed in practice). We use our custom DNS stub and the BIND-9.9.3 combined with our proxy as the recursive. For each protocol (UDP, TCP, TLS), the stub makes 140 unique queries, randomly drawn from the Alexa top-1000 sites [1] with DNS over that protocol. We restart the recursive resolver before changing protocols, so each protocol test starts with a known, cold cache. We then vary each combination of protocol (UDP, TCP, and TLS), use of pipelining or stop-and-wait, and in-order and out-of-order processing. Connections are either *reused*, with multiple queries per TCP/TLS connection (p-TCP/p-TLS), or *no reuse*, where the connection is reopened for each query. We repeat the experiment 10 times and report combined results.

Cold-Cache Performance: Figure 5 shows the results of these experiments. We see that UDP, TCP, and TLS performance is generally similar when other parameters are held consistent (compare (a), (b), and (c), or (g), (h), and (i)). Even when the RTT is 35 ms, the recursive query process still dominates protocol choice and setup costs are moderate. The data shows that out-of-order processing is essential when pipelining is used; case (f) shows head-of-line blocking compared to (h). This case shows that while current servers support TCP, our

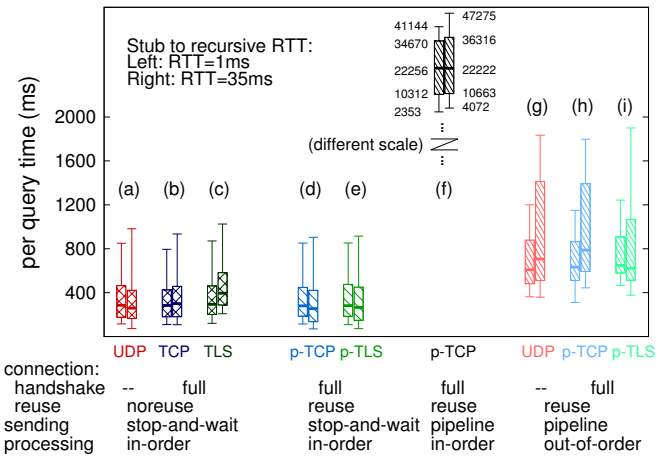


Fig. 5: Per-query response times with a cold cache for 140 unique names with different protocol configurations and two stub-to-recursive RTTs (1 ms and 35 ms). Boxes show median and quartiles. Case (f) uses a different scale.

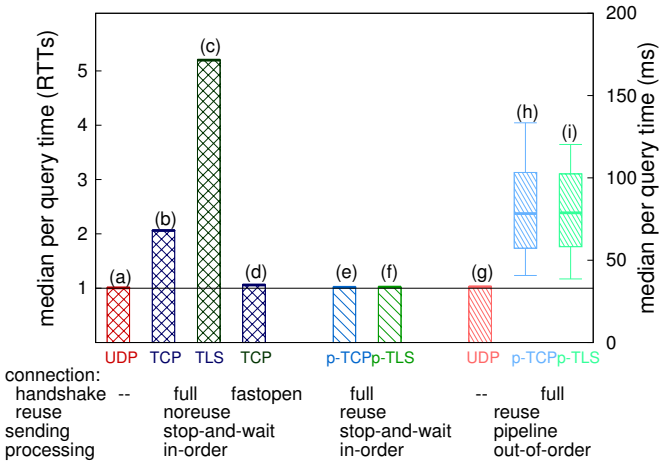


Fig. 6: Per-query response times for 140 repeated queries with different protocols, measured in RTTs (left axis) and ms (right). (Medians; boxes add quartiles.)

optimizations are necessary for high performance. Pipelining shows higher latency than stop-and-wait regardless of protocol (compare (g) with (a) or (i) with (c)). This difference occurs when 140 simultaneous queries necessarily queue at the server when the batch begins; UDP is nearly equally affected as TCP and TLS (compare (i) and (h) with (g)). Finally, we see that the costs of TLS are minimal here: comparing (c) with (b) and (a) or (i) with (g) and (h), natural variation dominates performance differences.

Warm-Cache Performance: Cold-cache performance is dominated by communication time to authoritative name servers. For queries where replies are already cached this communication is omitted and connection setup times become noticeable. For connection handling, performance of cache hits are equivalent to authoritative replies, so our recursive-to-authoritative experiments in § VI-C represent warm-cache performance with 100% cache hits. (We verified this claim by repeating our stub-to-recursive experiment, but making each query twice and reporting performance only for the second query that will always be answered from the cache.) While cache hits are expensive when they must start new connections, persistent connections *completely* eliminate this overhead (Figure 6, cases (e) and (f) compared to (a)). In addition, median TCP out-of-order pipelined connections (cases (h) and (i)) are slightly faster than UDP (case (g)) because TCP groups multiple queries into a single packet.

We conclude that *protocol choice makes little performance difference between stub and recursive* provided RTT is small and connections is not huge and connection reuse is possible. This result is always true with cold caches, where connection setup is dwarfed by communication time to authoritative name servers. This result applies to warm caches provided connections can be often reused or restarted quickly. We know that connections can be reused most of the time (§ IV-C), and TCP fast open and TLS resumption can reduce costs when they are not reused.

VI-C Latency: Recursive to Authoritative

We next consider performance between recursive resolvers and authoritative name servers. While recursives are usually near stubs, authoritative servers are globally distributed with larger and more diverse RTTs.

Typical RTTs: To measure typical recursive-to-authoritative RTTs, we use both the Alexa top-1000 sites, and for diversity, a random sample of 1000 sites from Alexa top-1M sites. We query each from four locations: the U.S. (Los Angeles), China (Beijing), the U.K. (Cambridge), and Australia (Melbourne). We query each domain name iteratively and report the time fetching the last component, taking the median of 10 trials to be robust to competing traffic and name server replication. We measure query time for the last component to represent caching of higher layers.

The U.S. and U.K. sites are close to many authoritative servers, with median RTT of 45 ms, but a fairly long tail with 35% of RTTs exceeding 100 ms. Asian and Australian sites have generally longer RTTs, with only 30% closer than 100 ms (China), and 20% closer than 30 ms (Australia), while the rest are 150 ms or more. This jump is due to the long propagation latency for services without sites physically in these countries. (We provide full data in our technical report [86].)

Methodology: To evaluate query latencies with larger RTTs between client and server, we set up a DNS authoritative server (BIND-9.9.3) for an experimental domain (example.com) and query it from a client 35 ms (8 router hops on a symmetric path) away. Since performance is dominated by round trips and not computation we measure latency in units of RTT and these results generalize to other RTTs. For each protocol, we query this name server directly, 140 times, varying the protocol in use. As before, we repeat this experiment 10 times and report medians of all combined experiments (Figure 6). Variation is usually tiny, so standard deviations are omitted except for cases (h) and (i).

Performance: Figure 6 shows the results of this experiment.

We first confirm that performance is dominated by protocol exchanges: cases (a), (b) and (c) correspond exactly to 1, 2, and 5 RTTs as predicted. Second, we see the importance of connection reuse or caching: cases (e) and (f) with reuse have *identical* performance to UDP, as does TCP fast open (case (d)).

As before, pipelining for TCP shows a higher cost because the 140 queries queue behind each other. Examination of packet traces for cases (h) and (i) shows that about 10% of queries complete in about 1 RTT, while additional responses arrive in batches of around 12, showing stair-stepped latency. For this special case of more than 100 queries arriving simultaneously, a single connection adds some latency.

We next consider the cost of adding TLS for privacy. The community generally considers aggregation at the recursive resolver sufficient for anonymity, but TLS may be desired there for additional privacy or as a policy [27] so we consider it as an option. Without connection reuse, a full TLS query always requires 5 RTTs (case (c), 175 ms): the TCP handshake, the DNS-over-TLS negotiation (§ III-B2), two for the TLS handshake, and the private query and response.

However, once established TLS performance is *identical* to UDP: cases (f) and (a) both take 1 RTT. Encryption's cost is tiny compared to moderate round-trip delays when we have an established connection. We expect similar results with TLS resumption.

Finally, when we add pipelining and out-of-order processing, we see similar behavior as with TCP, again due to how the large, batched queries become synchronized over a single connection.

We conclude that RTTs completely dominate recursive-to-authoritative query latency. We show that connection reuse can eliminate connection setup RTT, and we expect TLS resumption will be as effective as TCP fast-open. We show that TCP is viable from recursive-to-authoritative, and TLS is also possible.

VI-D Client connection-hit fractions

Connection reuse is important and § IV-C found very high reuse from the server's perspective. We next show that *client* connection-hit fractions are lower because many clients query infrequently.

To evaluate client connection hit fractions, we replay our three DNS traces through the simulator from § IV-C, but we evaluate connection hit fractions per client. Figure 8 shows these results, with medians (lines) and quartiles (bars, with slight offset to avoid overlap).

Among the three traces, the DNSChanger hit fraction exceeds Level 3, which exceeds B-Root, because servers further up the hierarchy see less traffic from any given client. We see that the top quartile of clients have high connection hit fractions for all traces (at 60 s: 95% for DNSChanger, 91% for Level 3, and 67% for B-Root). The connection hit rate for the median client is still fairly high for DNSChanger and Level 3 (89% and 72%), but quite low for B-Root (28%). Since most B-Root content can be cached, many clients only contact it infrequently and so fail to find an open connection.

These results suggest that clients making few requests will need to restart connections frequently. Fortunately TCP Fast Open and TLS Resumption allow these clients to carry the state needed to accelerate this process.

VI-E Modeling End-to-End Latency for Clients

With this data we can now model the expected *end-to-end* latency for DNS users and explore how stub, recursive and authoritative resolvers interact with different protocols and caching. Our experiments and measurements provide parameters and focus modeling on connection setup (both latency and CPU costs). Our model captures clients restarting connections, servers timing out state, and the complex interaction of stub, recursive, and authoritative resolvers. Our modeling has two limitations. First, we focus on typical latency for *users, per-query*; the modeling reflects query frequency, emphasizing DNS provisioning for common queries and reflecting queries to rare sites only in proportion to their appearance in our traces. We do not evaluate mean latency per-site, since that would be skewed by rarely used and poorly provisioned sites. Second, our models provide mean performance; they cannot directly provide a full distribution of response times and "tail" performance [19]. We are interested in using trace replay to determine a full distribution with production-quality servers, but as significant future work.

Modeling: We model latency from client to server, $L_{c\sigma}$, as the probability of connection reuse ($P^C_{c\sigma}$) and the cost of setting up a new connection ($S^C_{c\sigma}$) added to the the cost of the actual query ($Q_{c\sigma}$):

$$L_{c\sigma} = (1 - P^C_{c\sigma})S^C_{c\sigma} + Q_{c\sigma} \tag{1}$$

From Figure 6, $Q_{c\sigma}$ is the same for all methods with an open connection: about one client-server RTT, or $R_{c\sigma}$. Setup cost for UDP ($S^{C,udp}_{c\sigma}$) is 0. With the probability for TCP fast-open (TFO), $P^{TFO}_{c\sigma}$, TCP setup costs:

$$S^{C,tcp}_{c\sigma} = (1 - P^{TFO}_{c\sigma})R_{c\sigma} \tag{2}$$

We model TLS setup ($S^{C,tls}_{c\sigma}$) as the probability of TLS resumption ($P^{RE}_{c\sigma}$) and its cost $S^{C,tls_r}_{c\sigma}$, or the cost of setting up a completely new TLS connection $S^{C,tls_n}_{c\sigma}$:

$$S^{C,tls}_{c\sigma} = P^{RE}_{c\sigma}S^{C,tls_r}_{c\sigma} + (1 - P^{RE}_{c\sigma})S^{C,tls_n}_{c\sigma} \tag{3}$$

For simplicity, we assume TCP fast open and TLS resumption have the same timeout, so $P^{RE}_{c\sigma} = P^{TFO}_{c\sigma}$. Thus, $S^{C,tls_r}_{c\sigma}$ is $2R_{c\sigma} + S^{cpu_r}_{\sigma}$ (1 each for TLS negotiation and handshake) and $S^{C,tls_n}_{c\sigma}$ is $4R_{c\sigma} + S^{cpu_n}_{\sigma}$ (1 for TCP, 1 for TLS negotiation, and 2 for TLS handshake). We set $S^{cpu_n}_{\sigma}$ at 25.8 ms and $S^{cpu_r}_{\sigma}$ is at 1.2 ms(Table VI, with and without CA validation). We estimate $P^C_{c\sigma}$, $P^{RE}_{c\sigma}$ and $P^{TFO}_{c\sigma}$ from our timeout window and trace analysis (Figures 7 and 8).

To compute end-to-end latency (stub-to-authoritative, L_{sa}), we combine stub-to-recursive latency (L_{sr}) with behavior at the recursive resolver. For a cache hit (probability P^N_r) the recursive resolver can reply immediately. Otherwise it will

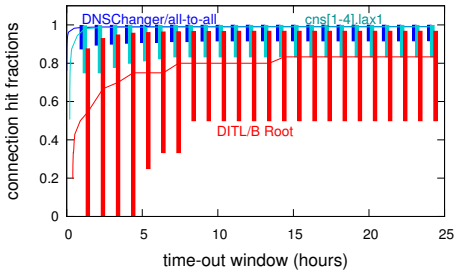


Fig. 7: Median client-side connection hit fractions with quartiles with larger time-out windows

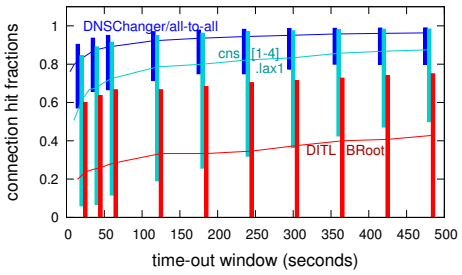


Fig. 8: Median client-side connection hit fractions with quartiles.

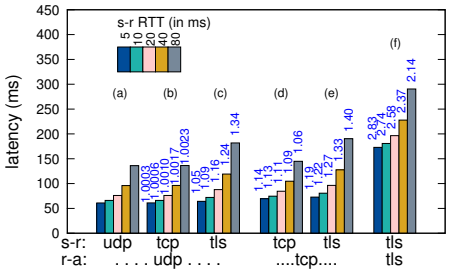


Fig. 9: End-to-end-performance as a function of protocol choice and stub-to-resolver RTT.

make several (N_r^Q) queries to authoritative resolvers (each taking L_{ra}) to fill its cache:

$$L_{sa} = L_{sr} + (1 - P_r^N) N_r^Q L_{ra} \quad (4)$$

Where L_{sr} and L_{ra} follow from Equation 1. We model recursive with the Level 3 data and authoritative as B-Root. With our recommended timeouts (60 s and 20 s), we get $P_{sr}^C = 0.72$ and $P_{ra}^C = 0.24$. We assume TCP fast open and TLS resumption last 2 hours at recursive ($P_{sr}^{RE} = P_{sr}^{TFO} = 0.9858$) and 7 h at authoritative ($P_{ra}^{RE} = P_{ra}^{TFO} = 0.8$). Prior studies of recursive resolvers suggest P_r^N ranges from 71% to 89% [37].

We determine N_r^Q by observing how many queries BIND-9.9.3 requires to process the Alexa top-1000 sites. We repeat this experiment 10 times, starting each run with a cold cache, which leads to $N_r^Q = 7.24$ (standard deviation 0.036, includes 0.09 due to query retries). We round N_r^Q to 7 in our analysis of estimated latency. Although this value seems high, the data shows many incoming queries require multiple outgoing queries to support DNSSEC, and due to the use of content-delivery networks that perform DNS-based redirection.

Scenarios: With this model we can quickly compare long-term average performance for different scenarios. Figure 9 compares six protocol combinations (each group of bars) We consider $R_{sr} = 5$ ms and $R_{sr} = 20$ ms suitable for a good U.S. or European ISP, but we report stub-to-recursive RTTs from 5 to 80 ms.

For the *local resolver*, the analysis shows that *use of TCP and TLS to the local resolver adds moderate latency*: current DNS has mean of 61 ms, and TCP is the same, and TLS is only 5.4% slower with UDP upstream. Second, we see that use of connections between recursive and authoritative is more expensive: with TLS stub-to-recursive, adding TCP to the authoritative is 19% slower and adding TLS to the authoritative more than 180% slower. This cost follows because a single stub-to-recursive query can lead to multiple recursive-to-authoritative queries, at large RTTs with a lower connection-hit fraction. However this analysis is pessimistic; the expected values underestimate possible locality in those queries.

For a *third-party resolver* ($R_{sr} = 20$ ms), the trends are similar but the larger latency to the recursive resolver raises costs: TLS to recursive (with UDP to authoritative), is 15.5% slower than UDP.

VII. Related work

Our work draws on prior work in transport protocols and more recent work in DNS security and privacy.

VII-A Siblings: DNSSEC and DANE/TLSA

DNS Security Extensions (DNSSEC) uses public-key cryptography to ensure the integrity and origin of DNS replies [4]. Since the 2010 signature of the root zone, it has provided a root of trust for DNS. DNS-based Authentication of Named Entities for TLS (DANE/TLSA) allows DNS to serve as a root of trust for TLS certificates [33]. Our work complements these protocols, addressing the related area of privacy.

Although DNSSEC protects the integrity and origin of requests, it does not address query privacy. We propose TLS to support this privacy, complementing DNSSEC. Although not our primary goal, TLS also protects against some attacks such as those that exploit fragmentation; we discuss these below.

DANE/TLSA's trust model is unrelated to T-DNS's goal of privacy. See § III-B1 for how they interact.

VII-B DNSCrypt and DNSCurve

OpenDNS has offered elliptic-curve cryptography to encrypt and authenticate DNS packets between stub and recursive resolvers (DNSCrypt [57]) and recursive resolvers and authoritative servers (DNSCurve [20]). We first observe that these protocols address only privacy, not denial-of-service nor limits to reply size.

These protocols address the same privacy goal as our use of TLS. While ECC is established cryptography, above this they use a new approach to securing the channel and a new DNS message format. We instead reuse existing DNS message format and standard TLS and TCP. Although DNSCrypt and DNSCurve are attractive choices, we believe TLS' runtime negotiation of cryptographic protocol is important for long-term deployment. We also see significant advantage in adopting existing standards with robust libraries and optimizations (such as TLS resumption) rather than designing bespoke protocols for our new application. In addition, while TLS implementations have reported recent flaws, our view is

that common libraries benefit from much greater scrutiny than new protocols. Finally, DNSCurve's mandate that the server's key be its hostname cleverly avoids one RTT in setup, but it shifts that burden into the DNS, potentially adding millions of nameserver records should each zone require a unique key.

DNSCrypt suggests deployment with a proxy resolver on the end-user's computer. We also use proxies for testing, but we have prototyped integration with existing servers, a necessity for broad deployment.

We compare DNSCrypt and DNSCurve performance in § VI-A, and features in Table II.

VII-C Unbound and TLS

We are not the first to suggest combining DNS and TLS. A recent review of DNS privacy proposed TLS [9], and NLnet Lab's `Unbound` DNS server has supported TLS since December 2011. Unbound currently supports DNS-over-TLS only on a separate port, and doesn't support out-of-order processing § III-A, and there is no performance analysis. Our work adds in-band negotiation and out-of-order processing (see Table II), and we are the first to study performance of DNS with TCP and TLS. Since the only difference is signaling TLS upgrade, our performance evaluation applies to other TLS approaches, although unbound's use of a new port avoids 1 RTT latency.

VII-D Reusing Other Standards: DTLS, TLS over SCTP, HTTPS, and Tcpcrypt

Although UDP, TCP and TLS are widely used, additional transport protocols exist to provide different semantics. Datagram Transport Layer Security (DTLS) provides TLS over UDP [65], meeting our privacy requirement. While DTLS strives to be lighter weight than TCP, it must re-create parts of TCP: the TLS handshake requires reliability and ordering, DoS-prevention requires cookies analogous to SYN cookies in TCP's handshake, and it caches these, analogous to TCP fast-open. Thus with DoS-protection, DTLS provides *no* performance advantage, other than eliminating TCP's data ordering. (We provide a more detailed evaluation of these in our technical report [86].) Applications using DTLS suffer the same payload limits as UDP (actually slightly worse because of its additional header), so it does not address the policy constraints we observe. Since DTLS libraries are less mature than TLS and DTLS offers few unique benefits, we recommend T-DNS .

TLS over SCTP has been standardized [38]. SCTP is an attractive alternative to TCP because TCP's ordering guarantees are not desired for DNS, but we believe performance is otherwise similar, as with DTLS.

Several groups have proposed some version of DNS over HTTP. Kaminsky proposed DNS over HTTP [40] with some performance evaluation [41]; Unbound runs the DNS protocol over TLS on port 443 (a non-standard encoding on the HTTPS port); others have proposed making DNS queries over XML [61] or JSON [10] and full HTTP or HTTPS. Use of port 443 saves one RTT for TLS negotiation, but using DNS encoding is non-standard, and HTTP encoding is significantly more bulky. Most of these proposals lack a complete specification (except XML [61]) or detailed performance analysis (Kaminsky provides some [41]). At a protocol level, DNS over HTTP must be strictly slower than DNS over TCP, since HTTP requires its own headers, and XML or JSON encodings are bulkier. One semi-tuned proxy shows 60 ms per query overhead [75], but careful studies quantifying overhead is future work.

Tcpcrypt provides encryption without authentication at the transport layer. This subset is faster than TLS and shifts computation to the client [8]. T-DNS's uses TLS for privacy (and DNSSEC for authentication), so tcpcrypt may be an attractive alternative to TLS. Tcpcrypt is relatively new and not yet standardized. Our analysis suggests that, since RTTs dominate performance, tcpcrypt will improve but not qualitatively change performance; experimental evaluation is future work.

The very wide use of TCP and TLS-over-TCP provides a wealth of time-tested implementations and libraries, while DTLS and SCTP implementations have seen less exercise. We show that TCP and TLS-over-TCP can provide near-UDP performance with connection caching. Because DTLS carries out the same protocol exchange as TLS (when spoof prevention is enabled), it will have the same latency. Our analysis applies directly to HTTP-based approaches, although its more verbose framing may have slightly higher overhead.

VII-E Other Approaches to DNS Privacy

Zhao et al. [85] proposed adding cover traffic (additional queries) to DNS to conceal actual queries from an eavesdropper, Castillo-Perez and Garcia-Alfaro extend this work [14]. These approaches may help protect against an adversary that controls the recursive resolver; we instead provide only communications privacy, without range queries.

Lu and Tsudick [47] identify a number of privacy threats to DNS and propose replacing it with a DHT-based system, and Zhao et al. [84] later propose DNS modifications to support their range queries [85]. Such approaches can provide very strong privacy guarantees, but such large protocol modifications pose significant deployment challenges.

VII-F Specific Attacks on DNS

As a critical protocol, DNS has been subject to targeted attacks. These attacks often exploit currently open DNS recursive name servers, and so they would be prevented with use of TLS' secure client-to-server channel. Injection attacks include the Kaminsky vulnerability [39], mitigated by changes to DNS implementations; sending of duplicate replies ahead of the legitimate reply [2], mitigated by Hold-On at the client [24]; and injection of IP fragments to circumvent DNSSEC [32], mitigated by implementation and operations changes.

Although specific countermeasures exist for each of these attacks, responding to new attacks is costly and slow. Connection-level encryption like TLS may prevent a broad class of attacks that manipulate replies (for example, [32]).

Although TLS is not foolproof (for example, it can be vulnerable to person-in-the-middle attacks), and we do not resolve all injection attacks (such as injection of TCP RST or TLS-close notify), we believe TLS significantly raises the bar for these attacks.

Similarly, recent proposals add cookies to UDP-based DNS to reduce the impact of DoS attacks [25]. While we support cookies, a shift to TCP addresses policy constraints as well as DNS, and enables use of TLS.

VIII. Conclusion

Connectionless DNS is overdue for reassessment due to privacy limitations, security concerns, and sizes that constrain policy and evolution. Our analysis and experiments show that *connection-oriented DNS addresses these problems*, and that latency and resource needs of T-DNS are manageable.

Acknowledgments

We would like to thank several that contributed data to this effort: DNS-OARC DITL program, operators of A, J, B Root name servers, Level3 and Common Crawl. Calvin Ardi extracted domain names from web pages from the Common Crawl dataset. Xun Fan helped collect data from PlanetLab. Christos Papadopoulos provided servers at CSU for our high-latency experiments. John Wroclawski, Bill Manning, and prior anonymous reviewers provided comments on the paper, many helpful. We also thank colleagues at Verisign and participants at the 2014 DNS-OARC workshop for comments and many though-provoking questions, particularly about deployment. We thank Ted Faber and Joe Touch for their discussions about TCP.

Research by Liang Zhu, Zi Hu, and John Heidemann in this paper is partially sponsored by the Department of Homeland Security (DHS) Science and Technology Directorate, HSARPA, Cyber Security Division, BAA 11-01-RIKA and Air Force Research Laboratory, Information Directorate under agreement number FA8750-12-2-0344, and contract number D08PC75599. The U.S. Government is authorized to make reprints for Governmental purposes notwithstanding any copyright. The views contained herein are those of the authors and do not necessarily represent those of DHS or the U.S. Government.

References

[1] Alexa, http://www.alexa.com/.
[2] Anonymous, "The collateral damage of internet censorship by DNS injection," *SIGCOMM Comput. Commun. Rev.*, Jun. 2012.
[3] Arbor Networks, "Worldwide infrastructure security report," Arbor Networks, Tech. Rep., Sep. 2012.
[4] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS Security Introduction and Requirements," RFC 4033, Mar. 2005.
[5] R. Bellis, "DNS Transport over TCP - Implementation Requirements," RFC 5966, Internet Engineering Task Force, Aug. 2010.
[6] V. Bernat, "SSL computational DoS mitigation," blog http://vincent.bernat.im/en/blog/2011-ssl-dos-mitigation.html, Nov. 2011.
[7] R. Beverly, R. Koga, and kc claffy, "Initial longitudinal analysis of IP source spoofing capability on the Internet," *Internet Society*, Jul. 2013.
[8] A. Bittau, M. Hamburg, M. Handley, D. Mazières, and D. Boneh, "The case for ubiquitous transport-level encryption," in *Proceedings of the 19th USENIX Conference on Security*, ser. USENIX Security'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 26–26.
[9] S. Bortzmeyer, "DNS privacy problem statement," Dec. 2013, Internet draft.
[10] ——, "JSON format to represent DNS data," Feb. 2013, work in progress (Internet draft draft-bortzmeyer-dns-json-01).
[11] R. Bush, D. Karrenberg, M. Kosters, and R. Plzak, "Root name server operational requirements," Internet Request For Comments, RFC 2870, Jun. 2000, (also Internet BCP-40).
[12] M. Butkiewicz, H. V. Madhyastha, and V. Sekar, "Understanding website complexity: Measurements, metrics, and implications," in *IMC*, Nov. 2011, pp. 313–328.
[13] M. Calder, X. Fan, Z. Hu, E. Katz-Bassett, J. Heidemann, and R. Govindan, "Mapping the expansion of Google's serving infrastructure," in *Proc. of ACM IMC*. Barcelona, Spain: ACM, Oct. 2013, p. to appear.
[14] S. Castillo-Perez and J. Garcia-Alfaro, "Evaluation of two privacy-preserving protocols for the DNS," in *Proc. of 6thIEEE International Conference on Information Technology: New Generations*. IEEE, Apr. 2009, pp. 411–416.
[15] S. Castro, D. Wessels, M. Fomenkov, and K. Claffy, "A day at the root of the Internet," *ACM Computer Communication Review*, vol. 38, no. 5, pp. 41–46, Oct. 2008.
[16] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain, "TCP Fast Open," RFC 7413, Internet Engineering Task Force, Dec. 2014.
[17] C. Coarfa, P. Druschel, and D. S. Wallach, "Performance analysis of TLS web servers," vol. 24, no. 1, pp. 39–69, Feb. 2006.
[18] J. Damas, M. Graff, and P. Vixie, "Extension mechanisms for DNS (EDNS(0))," RFC 6891, Internet Engineering Task Force, Apr. 2013.
[19] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.
[20] M. Dempsky, "DNSCurve: Link-level security for the Domain Name System," Feb. 2010, internet draft.
[21] T. Dierks and E. Rescorla, "The Transport Layer Security TLS Protocol Version 1.2," RFC 5246, Internet Engineering Task Force, Aug. 2008.
[22] DNS-OARC, https://www.dns-oarc.net/.
[23] R. Droms, "Dynamic host configuration protocol," Internet Request For Comments, RFC 2131, Mar. 1997.
[24] H. Duan, N. Weaver, Z. Zhao, M. Hu, J. Liang, J. Jiang, K. Li, and V. Paxson, "Hold-on: Protecting against on-path DNS poisoning," in *SATIN*, 2012.
[25] D. E. Eastlake, "Domain Name System (DNS) cookies," Jan. 2014, work in progress (Internet draft draft-eastlake-dnsext-cookies-04.txt).
[26] W. Eddy, "TCP SYN flooding attacks and common mitigations," Internet Request For Comments, RFC 4987, Aug. 2007.
[27] Electronic Frontier Foundation, "Encrypt the web with HTTPS everywhere," Web page https://www.eff.org/https-everywhere, Aug. 2011.
[28] T. Faber, J. Touch, and W. Yue, "The TIME-WAIT state in TCP and its effect on busy servers," INFOCOMM, 1998.
[29] T. Finch, "Secure SMTP using DNS-based authentication of named entities (DANE) TLSA records." Feb. 2014, internet draft.
[30] L. Green, "Common crawl enters a new phase," Common Crawl blog http://www.commoncrawl.org/common-crawl-enters-a-new-phase/, Nov. 2011.
[31] G. Greenwald, "NSA collecting phone records of millions of Verizon customers daily," *The Guardian*, June 2013.
[32] A. Herzberg and H. Shulmanz, "Fragmentation considered poisonous," in *Proc. of IEEE Conference on Communications and Network Security (CNS)*, Oct. 2013.
[33] P. Hoffman and J. Schlyter, "The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA," Internet Request For Comments, RFC 6698, Aug. 2012.
[34] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, and D. Wessels, "Starting TLS over DNS," Jan. 2014, work in progress (Internet draft draft-start-tls-over-dns-00).
[35] G. Huston, "A question of protocol," Talk at RIPE '67, 2013.
[36] ICANN, "Root server attack on 6 February 2007," ICANN, Tech. Rep., Mar. 2007.
[37] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, "DNS performance and the effectiveness of caching," *ACM/IEEE ToN*, vol. 10, Oct. 2002.
[38] A. Jungmaier, E. Rescorla, and M. Tuexen, "Transport Layer Security over Stream Control Transmission Protocol," RFC 3436, Internet Engineering Task Force, Dec. 2002.
[39] D. Kaminsky, "It's the end of the cache as we know it," Presentation, Black Hat Asia, Oct. 2008.

[40] ——, "The DNSSEC diaries, ch. 6: Just how much should we put in DNS?" blog post http://dankaminsky.com/2010/12/27/dnssec-ch6/, Dec. 2010.

[41] ——, "DNSSEC interlude 1: Curiosities of benchmarking DNS over alternate transports," blog post http://dankaminsky.com/2011/01/04/dnssec-interlude-1/, Jan. 2011.

[42] C. A. Kent and J. C. Mogul, "Fragmentation considered harmful," in *SIGCOMM*, Aug. 1987, pp. 390–401.

[43] M. Kührer, T. Hupperich, C. Rossow, and T. Holz, "Hell of a handshake: Abusing TCP for reflective amplification DDoS attacks," in *Proc. of USENIX Workshop on Offensive Technologies*. San Diego, CA, USA: USENIX, Aug. 2014.

[44] A. Langley, "Overclockign SSL," blog post https://www.imperialviolet.org/2010/06/25/overclocking-ssl.html, Jun. 2010.

[45] B. Laurie, G. Sisson, R. Arends, and D. Blacka, "DNS security (DNSSEC) hashed authenticated denial of existence," Internet Request For Comments, RFC 5155, Mar. 2008.

[46] C. Lewis and M. Sergeant, "Overview of best email DNS-based list (DNSBL) operational practices," Internet Request For Comments, RFC 6471, Jan. 2012.

[47] Y. Lu and G. Tsudik, "Towards plugging privacy leaks in domain name system," in *Proc. of IEEE International Conference on Peer-to-Peer Computing*. Delft, Netherlands: IEEE, Aug. 2010.

[48] J. Markoff and N. Perlroth, "Attacks used the Internet against itself to clog traffic," *New York Times*, March 2013.

[49] Matthew Dempsky, "Dnscurve client tool," https://github.com/mdempsky/dnscurve/.

[50] J. Mauch, "Open resolver project," Presentation, DNS-OARC Spring 2013 Workshop (Dublin), May 2013, https://indico.dns-oarc.net//contributionDisplay.py?contribId=24&sessionId=0&confId=0.

[51] W. Meng, R. Duan, and W. Lee, "DNS Changer remediation study," Talk at M3AAWG 27th, Feb. 2013.

[52] C. Metz, "Comcast trials (domain helper service) DNS hijacker," The Register, Jul. 2009.

[53] P. Mockapetris, "Domain names—implementation and specification," RFC 1035, Internet Engineering Task Force, Nov. 1987.

[54] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. Lie, and C. Lilley, "Network performance effects of http/1.1, css1, and png," in *SIGCOMM*, Sep. 1997.

[55] NLnetLabs, "Dnssec-trigger," web page http://www.nlnetlabs.nl/projects/dnssec-trigger/, May 2014.

[56] M. Nottingham, "HTTP/2 frequently asked questions," web page http://http2.github.io/faq/#why-is-http2-multiplexed, 2014.

[57] OpenDNS, "Dnscrypt," http://www.opendns.com/technology/dnscrypt.

[58] ——, "Dnscrypt proxy tool," https://github.com/opendns/dnscrypt-proxy.

[59] ——, "Opendns website," www.opendns.com, 2006.

[60] E. Osterweil, B. Kaliski, M. Larson, and D. McPherson, "Reducing the X.509 attack surface with DNSSEC's DANE," in *Proc. of Workshop on Securing and Trusting Internet Names (SATIN)*, Teddington, UK, Mar. 2012.

[61] M. Parthasarathy and P. Vixie, "draft-mohan-dns-query-xml-00," Sep. 2011, work in progress (Internet draft draft-mohan-dns-query-xml-00).

[62] H. P. Penning, "Analysis of the strong set in the PGP web of trust," web page http://pgp.cs.uu.nl/plot/, Jan. 2014.

[63] P. Ramaswami, "Introducing Google Public DNS," Google Official Blog http://googleblog.blogspot.com/2009/12/introducing-google-public-dns.html, Dec. 2009.

[64] S. Reifschneider, "4.2.2.2: The story behind a DNS legend," http://www.tummy.com/articles/famous-dns-server/.

[65] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security Version 1.2," RFC 6347, Internet Engineering Task Force, Jan. 2012.

[66] C. Rossow, "Amplification hell: Revisiting network protocols for DDoS abuse," in *Proc. of ISOC Network and Distributed System Security Symposium*. San Diego, California, USA: The Internet Society, Feb. 2014.

[67] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State," RFC 5077, Internet Engineering Task Force, Jan. 2008.

[68] K. Schomp, T. Callahan, M. Rabinovich, and M. Allman, "On measuring the client-side DNS infrastructure," in *IMC*, Oct. 2013.

[69] S. Sengupta, "Warned of an attack on the Internet, and getting ready," *New York Times*, p. B1, Mar. 31 2012.

[70] H. Shulman, "Pretty bad privacy: Pitfalls of DNS encryption," in *Proc. of 13th ACM Workshop on Privacy in the Electronic Society*. Scottsdale, Arizona: ACM, Nov. 2014, pp. 191–200.

[71] W. Simpson, "TCP cookie transactions (TCPCT)," Jan. 2011.

[72] A. Sullivan, "More keys in the DNSKEY RRset at ".", and draft-ietf-dnsop-respsize-nn," DNSOP mailing list, Jan. 2014, https://www.mail-archive.com/dnsop@ietf.org/msg05565.html.

[73] S. Sundaresan, N. Feamster, R. Teixeira, and N. Magharei, "Measuring and mitigating web performance bottlenecks in broadband access networks," in *Proc. of ACM IMC*. Barcelona, Spain: ACM, Oct. 2013, p. to appear.

[74] R. Vaughn and G. Evron, "DNS amplification attacks," http://isotf.org/news/DNS-Amplification-Attacks.pdf, Mar. 2006.

[75] Verisign Labs, "DNS over JSON prototype," 2014.

[76] P. Vixie, "Extension mechanisms for DNS (EDNS0)," Internet Request For Comments, RFC 1999, Aug. 1999.

[77] P. Vixie and A. Kato, "DNS referral response size issues," May 2012, internet draft.

[78] P. Vixie, "What DNS is not," *ACM Queue*, Nov. 2009.

[79] N. Weaver, C. Kreibich, B. Nechaev, and V. Paxson, "Implications of Netalyzr's DNS measurements," in *Proc. of Workshop on Securing and Trusting Internet Names (SATIN)*, Apr. 2011.

[80] W. Wijngaards and G. Wiley, "Confidential dns," Mar. 2014, internet draft.

[81] S. Woolf and D. Conrad, "Requirements for a Mechanism Identifying a Name Server Instance," RFC 4892, Internet Engineering Task Force, Jun. 2007.

[82] P. Wouters, "Using DANE to associate OpenPGP public keys with email addresses," Feb. 2014, internet draft.

[83] P. Wouters and J. Abley, "The edns-tcp-keepalive EDNS0 option," Oct. 2013, work in progress (Internet draft draft-wouters-edns-tcp-keepalive-00).

[84] F. Zhao, Fukuoka, Y. Hori, and K. Sakurai, "Analysis of existing privacy-preserving protocols in domain name system," in *IEICE TRANSACTIONS on Information and Systems*, May 2010.

[85] F. Zhao, Y. Hori, and K. Sakurai, "Analysis of privacy disclosure in DNS query," in *5th Intl. Conf. on Multimedia and Ubiquitous Engineering*. Loutraki, Greece: FTRI, Apr. 2007.

[86] L. Zhu, Z. Hu, J. Heidemann, D. Wessels, A. Mankin, and N. Somaiya, "Connection-oriented DNS to improve privacy and security (extended)," USC/Information Sciences Institute, Tech. Rep. ISI-TR-2015-695, Feb.