T-DNS: Connection-Oriented DNS to Improve Privacy and Security

USC/ISI Technical Report ISI-TR-688, Feb. 2014 *

Liang Zhu[1] Zi Hu[1] John Heidemann[1]
Duane Wessels[2] Allison Mankin[2] Nikita Somaiya[1]
1: University of Southern California
2: Verisign

ABSTRACT

This paper explores *connection-oriented DNS* to improve DNS security and privacy. DNS is the canonical example of a connectionless, single packet, request/response protocol, with UDP as its dominant transport. Yet DNS today is challenged by eavesdropping that compromises privacy, source-address spoofing that results in denial-of-service (DoS) attacks on the server and third parties, injection attacks that exploit fragmentation, and size limitations that constrain policy and operational choices. We propose *t-DNS* to address these problems: it combines TCP to smoothly support large payloads and 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. Traditional wisdom is that connection setup will balloon latency for clients and overwhelm servers. These are myths—our model of end-to-end latency shows *TLS to the recursive resolver is only about 21% slower*, with UDP to the authoritative server. End-to-end latency is 90% slower with TLS to recursive and TCP to authoritative. Experiments behind these models show that after connection establishment, TCP and TLS latency is equivalent to UDP. Using diverse trace data we show that frequent connection reuse is possible (60–95% for stub and recursive resolvers, although half that for authoritative servers). With conservative timeouts (20 s at authoritative servers and 60 s elsewhere) we show that *server memory requirements match current hardware*: a large recursive resolver may have 25k active connections consuming about 9 GB of RAM. We identify the key design and implementation decisions needed to minimize overhead—query pipelining, out-of-order responses, TLS connection resumption, and plausible timeouts.

1. INTRODUCTION

The Domain Name System (DNS) is the canonical example of a simple request-response protocol. A client uses DNS to translate a domain name like `www.iana.org` into the IP address of the computer that will provide service. A single web page may require resolving several domain names, so latency of each query should be minimal [9]. Historically, requests and responses are small (less than 512 B), so a single-packet request is answered with a single-packet reply over UDP.

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

This paper makes two contributions: first, we show that *connectionless DNS faces fundamental weaknesses today* stemming from the extension of DNS to many applications in a less safe Internet. We support this claim by summarizing the range of problems that stem from DNS' focus on single-packet, connectionless communication: weak privacy, security concerns, and operational constraints. While individual problems are small and can often be worked around, taken together they prompt revisiting assumptions.

These challenges prompt us to reconsider connection-oriented DNS: we propose t-DNS, where DNS requests should use TCP by default (not as last resort), and that DNS requests from end-users should use Transport-Layer Security (TLS, [12]). Our second contribution is to show that *end-to-end latency of t-DNS is only moderately more than connectionless.* Our models show latency is only 21% greater with TLS to the recursive resolver and UDP beyond, and 90% slower with TLS to the recursive resolver and TCP beyond. With moderate timeout durations (20 s at authoritative servers and 60 s elsewhere), connection reuse at servers is high (85%–98%), amortizing setup cost and leaving memory requirements are well within what is typical for servers today. Connection reuse rates for clients is lower (60–80% at the edge, but 20–40% at the root), but caching at recursive resolvers minimizes the end-to-end latency.

*Reseach 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.

The *connection rates within even modest server-class hardware today.* With conservative timeouts (20 s at authoritative servers and 60 s elsewhere), a large recursive resolver may have 25k active connections using about 9 GB of RAM, and double at an authoritative server.

Why: Connection-based communication is essential to support encryption of DNS queries for privacy[1]. Increasing use of wireless networks, and growth of third-party DNS (such as OpenDNS since 2006 [37] and Google Public DNS since December 2009 [39]) means that end-user requests are not to a server on the local network but may cross several networks and be subject to eavesdropping. Prior work has suggested a from scratch design [36, 11]; we instead an existing standards to DNS to provide confidentiality, and demonstrate only moderate performance costs. As a side-effect, it also protects DNS queries from tampering over parts of their path.

TCP's connection establishment by itself reduces vulnerabilities due to denial-of-service (DoS) attacks by forcing both sides of the conversation to prove their existence and limit the effects of source-address spoofing. TCP supports well-established techniques to tolerate DoS attacks [16]. In DNS amplification attacks an anonymous attacker gets 20:1 increase in traffic to its victim with spoofed UDP traffic to a DNS server, a critical component of recent multi-Gb/s DoS attacks [4]. TCP-based-DNS cannot be used for amplification attacks,

DNSSEC and other recent uses of DNS greatly increase response sizes. Considerations of large DNS responses over UDP reveal a number of limitations. Originally DNS limited UDP messages to 512 B [34], a limitation later relaxed by the Extension Mechanisms for DNS (EDNS0) [10]. Today most servers advertise support for 4096 B messages, but messages near or above Ethernet's 1500 MTU size are likely to see IP-level fragmentation, presenting several dangers: (1) fragments may be dropped by the network due to packet loss, increasing the chance that the entire message can not be delivered; (2) a noticeable fraction of middleboxes (firewalls) block all IP fragments; and (3) fragmentation is one component in a class of recently discovered attacks [21]. In addition, the transition from UDP to TCP for large (truncated) replies incurs a UDP round-trip (in addition to TCP connection establishment) before a full reply can be received. Even with years of support for large replies, operations still strive to live within original constraints [49].

How: On the surface, connection-oriented DNS seems untenable, since TCP doubles the round-trips and requires state on servers. If TCP is bad, 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, TLS connection resumption, shifting state to clients when possible. Combined with conservative timeouts, these 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.

[1] While DTLS provides TLS over UDP, it must implement ordered, reliable delivery for the TLS handshake [41].

2. PROBLEM STATEMENT

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

2.1 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.* Domain names are hierarchical with multiple *components.* The database has a common root and is distributed across millions of servers. Replies return resource records of a few dozen specific types.

Originally DNS was designed to map domain names to IP addresses. Its success as a lightweight, well understand key-to-value mapping protocol caused its role to quickly grow to other Internet-related applications, including host integrity identification for anti-spam measures and and replica selection in content-delivery networks [48]. Recently DNS's trust framework (DNSSEC) has been used to provide an alternative to traditional PKI/Certificate Authorities for e-mail [2] and TLS [22].

Protocols: The 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 (expected to be kilobytes or more in size). DNS truncates replies larger than advertised limits, prompting clients to retry with TCP [47]. UDP can support large packets with IP fragmentation, at the cost of new problems discussed below.

The integrity of DNS replies is protected by DNSSEC [5]. DNSSEC provides cryptographic integrity checking of positive and negative DNS replies. Since July 2010 the root zone has been signed, providing a root of trust through signed sub-domains. DNSSEC provides data integrity, but makes no attempt to protect privacy.

As a Distributed System: DNS resolvers have both client and server components. Resolvers typically take three roles: stub, recursive, authoritative (Figure 1). *Stub resolvers* (or "stubs") are clients that talk only to recursive resolvers, which handle name resolution. Stubs are typically use one or a few recursive resolvers, with configuration automated through

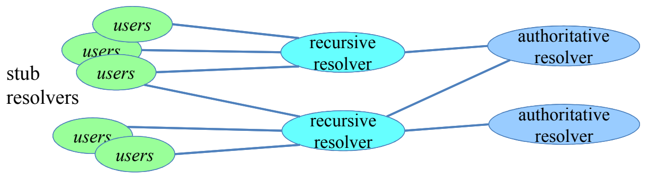


Figure 1: *Stub*, *recursive*, and *authoritative* resolvers.

DHCP [14] 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 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 (see [43], for example); we describe some of one operator's implementation in § 5.1.

2.2 Problem: 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 1. We consider transition in § 4.4.

2.2.1 Avoiding Arbitrary Limits to Response Size

Limitation in payload size is an increasing problem as DNS evolves to improve security. Without EDNS [10], 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 [29]. Due to problematic middleboxes, clients must be prepared to fall back to 512 B, or resend the query by TCP. Evidence shows that about of 2.6% web users are behind resolvers that fail to retry queries over TCP [24]. Such work-arounds are often fragile and the complexities of incomplete replies can be a source of bugs and security problems [21], including partial compromise of DNSSEC.

While 512 B has been adequate for many years, the recent deployment of DNSSEC makes it insufficient for almost any complete query. DNSSEC's cryptographic signatures add hundreds of bytes to a reply that before might only have been several dozen.

Figure 2 shows the size of responses from popular au-

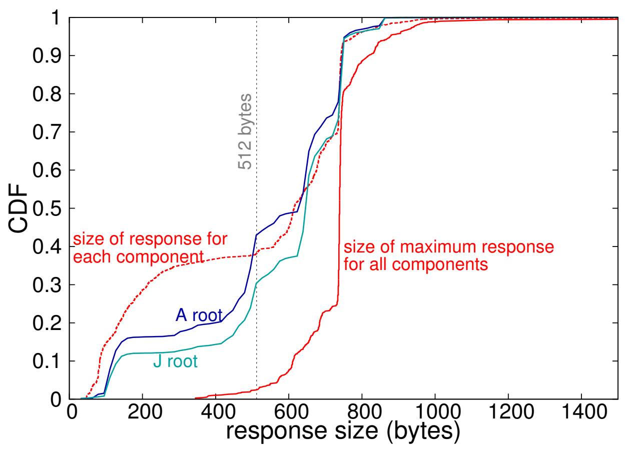


Figure 2: Response sizes from name servers of the Alexa top-1000 websites (as of 2014-01-24), and UDP response sizes out of two root servers. (Data: 2013-10-01)

thoritative name servers and from two root DNS servers. This table shows resolution of full domain names for the Alexa top-1000 websites. We show the distribution of responses for *each* component as well as the maximum seen over all components of each domain name. From this graph we see that responses today are fairly large: nearly 75% of top 1000 result in a response that is at least 738 bytes (the DNSSEC-signed reply for `.com`). Resolvers today require EDNS support for large replies.

Resolution of these domain names typically requires 600–800 B replies. Many Internet paths support 1500 B packets without fragmentation, making these sizes a good match for today's network. This result is not surprising: of course DNS use is tailored to match current constraints. However, *transient conditions stress these limits*. One example is the two methods of DNSSEC key rollover. With pre-published keys, DNSKEY responses grow, or double-signatures all signed responses are temporarily inflated. Both stretch reply sizes.

More importantly, size constraints can *distort current policy decisions*. For example, proposals to decentralize signing authority for the DNS root may replies that require TCP for root resolution [45]. For some, this requirement for TCP is seen as a significant technical barrier forcing use of shorter keys or different algorithms.

Finally, size can also *preempt future DNS applications*. Recent work has explored the use of DNS for managing trust relationships, so one might ask how DNS would be used if these constraints to response size were removed. We examine the PGP web of trust [38] 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

problem	current DNS	with t-DNS (why)
packet size limitations	guarantee: 512 B, typical: 1500 B	no limit (from TCP bytestream)
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 1: Benefits of t-DNS.

signatures or more [38]. If DNS either grows to consider non-hierarchical trust, or if it is simply used to store such information, larger replies will be important.

2.2.2 Need for Sender Validation

Uncertainty about the source 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 [25, 44], and to leverage DNS servers as part of an attack on a third party through a DNS Amplification attack [46, 30].

Work-arounds to DNS' role in DoS attacks exist. Many anti-spoofing mechanisms have been proposed, and DNS servers should rate-limit replies. T-DNS would greatly reduce the vulnerability of DNS to DoS and to participate in DoS. Well established techniques protect DNS servers from TCP-based DoS attacks [16], 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, measurement of the feasibility of source-IP spoofing show ability to spoof in the number of networks and autonomous systems has been fairly steady over the last six years [7]. Recent reports of DoS attacks show that DNS amplification is a serious problem, particularly in the largest attacks [4]. T-DNS suggests a long-term path to reduce this risk.

2.2.3 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 confirmation of widespread government eavesdropping [20] has identified the need to improve DNS privacy [8]. In addition, recent trends in DNS deployment make DNS query privacy of increasing concern. First, end-user queries are increasingly exposed to possible eavesdropping, from use of third-party DNS services such as OpenDNS and Google Public DNS, and through use of public Internet services such as WiFi hotspots and Internet cafes. Second, presence of widespread eavesdropping and misdirection is now well documented, by governments for espionage [20], censorship [3], or criminal gain [32]. Finally, ISPs have recognized the opportunity to monetize DNS typos, redirecting non-existing domain responses (NXDOMAIN hijacking), widespread since 2009 (for example [33]). 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 configured recursive name server.

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.

2.3 Threat Model

To understand security aspects of these problems we next define our threat model. For *fragmentation attacks* due to limited packet size, assume an off-path adversary that can inject packets with spoofed source addresses, following Herzberg and Schulman [21].

For *DoS attacks* exploiting spoofed source addresses, our adversary can send to the 30M currently existing open, recursive resolvers that lack ingress filtering [31].

For *query eavesdropping* and attacks on privacy, our threat model is an adversary with network access on the network path between the user and the recursive resolver. We assume aggregation at the recursive resolver is sufficient anonymization, although one could add TLS between the recursive and authoritative name server as well.

We assume traffic from the recursive resolver to authoritative name server is safe, and that the operator of that resolver is trusted. Although outside the scope of this paper, this trust requirement is 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.

We focus only on DNS privacy and assume integrity is provided by DNSSEC. However, we do secure stub-to-recursive resolver and protect the integrity non-DNSSEC zones between the user and the recursive resolver (although not beyond it). We also consider only privacy of *DNS*: an eavesdropper on the local network can observe other traffic, including IP addresses of websites.

3. RELATED WORK

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

3.1 DNS Security Extensions (DNSSEC)

The Domain Name System Security Extensions use public-key cryptography to ensure the integrity and origin of DNS replies [5]. Since the 2010 signature of the root zone it has provided a root of trust for DNS.

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.

3.2 DNSCrypt and DNSCurve

OpenDNS has proposed use elliptic-curve cryptography to encrypt and authenticate DNS packets between stub and recursive resolvers (DNSCrypt [36]) and recursive resolvers and authoritative servers (DNSCurve [11]).

These protocols address the same goals as we propose for 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. Their custom approach allows tight integration with DNS, but the implementation does not benefit from existing TLS libraries which benefit from wide study, experience, and optimization (for example, proposals for TLS resumption). TLS's suite of cryptographic protocols also aids future evolution.

In addition, the suggested DNSCrypt deployment is a proxy resolver on the end-user's computer. We too use proxies for testing, but long-term deployment needs to be integrated with existing systems. We suggest that integration of existing standards, with careful implementation choices, will provide faster deployment.

3.3 Unbound and TLS

We are not the first to suggest combining DNS and TLS. Recent review of DNS privacy proposed TLS [8], and NLnet Lab's `Unbound` DNS server has supported TLS since December 2001. Unbound currently supports DNS-over-TLS only on a separate port. To avoid this requirement we have prototyped our proposed in-band negotiation in unbound.

3.4 Other Standards: DTLS and TLS over SCTP

Although UDP, TCP and TLS are widely used, additional transport protocols exist to provide different semantics. Datagram Transport Layer Security (DTLS) implements TLS over UDP [41]. It meets our privacy requirements and avoids full TCP connection establishment. DTLS must still implement some support for reliability and ordering during its TLS handshake. In addition, DTLS requires application-level support for query reliability and retransmission. Current DNS implementations support these for small requests, but would likely require additional work to provide reliability for large non-TCP replies.

TLS over SCTP has been standardized [27]. SCTP is an attractive alternative to TCP because TCP's ordering guarantees are not desired for DNS.

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. However, these protocols deserve evaluation and comparison to TCP and TLS and we hope to explore that as future work.

3.5 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. Currently researchers employ more specific countermeasures.

The Kaminsky Vulnerability exploited a small identification space to allow injection of content into DNS caches [28]. It is unknown if this vulnerability is exploited, but it resulted in rapid deployment of countermeasures (use of the port space to increase effective ID size) in most DNS software. Another class of injection attacks "race" alternative replies ahead of the legitimate reply, sometimes for government-supported censorship [3]. Hold-On is a proposed countermeasure for this specific attack [15]. Fragmentation attacks exploit the IP-level fragmentation of large DNS queries to replace portions of DNS replies [21]; they also propose specific measures to prevent or recover from fragmentation.

Although specific countermeasures exist for each of these attacks, responding to new attacks is costly and slow. Connection-level encryption like TLS prevents a broad class of attacks that manipulate replies. Although TLS is not foolproof (for example, it can be vulnerable to person-in-the-middle attacks), it significantly raises the bar for these attacks.

3.6 Applying DNS for Trust: DANE/TLSA

Recently standardized DNS-based Authentication of Named Entities for TLS (DANE/TLSA) brings DNS to serve as a root of trust for TLS certificates [22]. Certificate authentication is necessary to avoid person-in-the-middle attacks. DANE/TLSA builds the DNSSEC-enforced trust in the DNS hierarchy to provide an alternative authentication method to current certificate authorities (CAs). It is therefore unrelated to our goal of improving the privacy of DNS communication.

There may seem to be a circular dependency between DANE/TLSA and DNS-over-TLS if DNS uses TLS for privacy, and DANE/ TLSA depends on DNS to authenticate certificates. This dependency can be broken with an external CA (many of which already exist). Alternatively, one may use DANE/TLSA without the privacy to establish an initial TLS certificate for a third-party DNS provider (like Google or OpenDNS), if disclosure of this use is acceptable.

4. DESIGN AND IMPLEMENTATION OF T-DNS

We next describe our design for in-band TLS negotiation. We also identify key implementation details that can produce large performance differences (§ 6).

4.1 DNS over TCP

Design of DNS support for TCP was in the original specification [34], and details were clarified in subsequent RFCs [6]. 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 the ability to send several queries before the 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 authoritative servers like `.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%) from CommonCrawl-002 [19] 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`, but neither sends outgoing TCP unless forced to by indication of reply truncation in UDP. Our custom stub resolver supports pipelining.

Out-of-order processing (OOOP) at recursive resolvers is another important optimization to avoid head-of-line blocking. OOOP is defined and explicitly allowed by RFC-5966 [6]. Without OOOP, queries to even a small percentage of distant servers with stall a strictly-ordered queue, unnecessarily delaying all subsequent queries. Without connections, concurrent UDP queries are naturally independent and all major DNS servers process them concurrently.

We know of no DNS server today that supports out-of-order processing of TCP queries; 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, and have a prototype implementations in unbound.

Finally, when possible, we wish to **shift state from server to client** Per-client state can be a significant cost for servers with many connections, as observed for TIME-WAIT state due to closed TCP connections as previously observed in web servers [18]. Support for shifting TCP state with DNS is currently being standardized [52].

The importance of these implementation details are not unique to DNS; they have been recognized before in HTTP [35, 18]. HTTP supports only pipelining, but both pipelining and OOOP are possible in DNS.

4.2 DNS over TLS

TLS support in DNS builds on TCP, adding new decisions about grounding trust, TLS negotiation, and implementation choices.

4.2.1 Grounding Trust

TLS depends on public-key cryptography to establish session keys to secure each connection and prevent person-in-the middle attacks [12]. DNS-over-TLS can use existing methods of certificate validation: one may have a list of known good Certificate Authorities (CAs), obtained out-of-band (such as from the vendor of ones operating system). Existing public-key infrastructures (PKI) distribute certificate authentication through a hierarchy of signatures, allowing one to begin with a shorter list of CAs. Alternatively, protocols such as DANE/TLSA leverage DNSSEC to build a chain of trust through the DNS [22]. Any of these methods are suitable for our use, although care must be taken with recursion if trust for DNS resolution depends on DNSSEC.

4.2.2 Upwards TLS Negotiation

T-DNS requires a method to negotiate the use of TLS. Early users of TLS depended on separate ports to indicate its use (for example, HTTP and HTTPS with TCP ports 80 and 443), but IETF encourages new users of TLS to negotiate its use inside current protocols, and this is the preferred mechanism for most other protocols (IMAP, POP3, SMTP, FTP, XMPP, LDAP, and NNTP, although all also have have IANA-recognized but not RFC-standardized ports to indicate TLS). We therefore propose a new EDNS0 Extension Mechanism [10] to negotiate the use of TLS. We summarize our current proposal below; a formal specification is also available [23].

To negotiate TLS, we reserve a new "TLS OK" (TO) bit in the extended flags of the EDNS0 OPT record. A client requests TLS by setting this bit and placing a DNS query. A server that supports TLS responds with the same bit set, then both client and server transition to a TLS handshake [12]. The TLS handshake generates a unique session key; following the handshake the connection is protected from eavesdropping. Normal DNS queries then continue on this connection.

The DNS query made during TLS negotiation is special and proceeds in the clear. This query should not disclose information. We recommend a distinguished query for name "STARTTLS", type TXT, class CH, analogous to current support queries [51].

Once TLS is negotiated, we expect the client and server to maintain an active, TLS-enabled TCP connection. This connection can be used for subsequent DNS requests, avoiding the significant expense of TLS setup. We expect connections to be closed after they are idle for some timeout period as proposed in § 5.

4.2.3 *Implementation Optimizations*

Two implementation options affect performance. First, TLS supports **connection resumption**, where the server passes all the state needed to re-create a TLS connection to the client [42]. 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 to two RTTs, as well as reducing server computation by reusing the master secret and ciphersuite. Experimentally we see that resumption takes less than 1 ms to resume a session § 6.1).

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.

4.3 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.

4.4 Gradual Deployment

Given the huge deployed base of DNS clients and servers, any modifications to DNS will take effect gradually. In general, our changes are backwards compatible with current DNS deployments. TCP is already supported for DNS, so clients and servers can upgrade independently. Our performance improvements require server-side changes and can be deployed as servers operators see greater TCP use. Privacy requires TLS support at both client and server, but clients can fall back on non-TLS DNS. Only connection establishment's role in DoS mitigation requires widespread deployment to be effective. However, incremental deployment can proceed at servers most vulnerable to use in DNS amplification and could be accompanied by rate-limiting on UDP queries, providing some reduction in harm.

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

5. CONNECTION REUSE AND SERVER-SIDE RESOURCE CONSUMPTION

The purpose of DNS is get a reply for a query. UDP is well matched to this goal, while t-DNS adds the delay of connection setup before a query can be made. When we examine latency in § 6, we will show that *connection reuse* is essential to amortizing the cost of setup over several queries. This need brings a fundamental trade-off: clients prefer long-lived connections, because for them resources are plentiful and latency is precious. For servers, however, resources are shared across many clients, and so even modest state per idle connection can add up, encouraging short-lived connections.

In this section we examine this trade-off using traces from from different real-world DNS deployments. We simulate different connection time-out periods to understand *connection hits* and misses. A *connection hit* is when a new query B arrives within the time-out window of a prior query A. In this case, B will be sent on the same connection as A without suffering connection setup. (A connection hit should not be confused with a DNS cache hit, which is when the answer for a query can be found in the cache of a stub or recursive resolver.) A connection miss is a query that finds no active connection, either because it is the first to a server, or a prior connection timed out and was torn down.

We evaluate connection reuse with two metrics: the *connection hit fraction* indicates how many queries are connection hits and not misses. We also consider the number of *concurrent connections* as the total number of clients with non-timed-out connections to a server at any time. We relate this count to memory usage at the server, and plot worst-case number of concurrent connections for every second.

5.1 Datasets

We use three different datasets (Table 2) for our trace analysis to stand in for stub clients, recursive resolvers, and authoritative servers.

DNSChanger: DNS changer was malware that redirected 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 name servers while owners of infected computers were informed [32]. It contains the timing of all queries made from end-user IP addresses that were compromised by this malware as captured at recursive resolvers run by the working group. We believe each IP address in this dataset represents an individual computer, and so we use it to represent stub-to-recursive resolver traffic. We use the traffic to the busiest server (all-to-one) in § 5.3 and the traffic from all the sources to all the servers (all-to-all) in § 6.4.

DITL/Level 3: Level 3 operates DNS service for their customers, and also as an open public resolver [40]. Their infrastructure supports 9 sites, each with typically 4 front-end recursive resolvers, each load-balanced across typically 8 back-end resolvers. We confirmed this architecture with the operators. They have provided a 48-hour dataset from this infrastructure to DNS-OARC [13].

We examined two subsets of this data. We first selected a random site (lax1, although we confirmed other sites give similar results). Most client IP addresses (89%) access only one site, so by examining this site we see all traffic for most clients in the dataset `cns[1-4].lax1`. Many clients (75%) only access one front-end at a site, so we also consider smaller subset of this data, selecting the busiest front-end at this site (`cns4.lax1`). Since it is representative, we use this still large subset (2.4B records) to speed processing. We use these Level 3 traces to represent a recursive resolver.

DITL/B-Root: This dataset was taken at the B-Root nameserver as part of DITL-2013 and is provided through DNS-OARC. We selected B-Root because at the time of this collection it employed only a single site, so this dataset captures all client traffic to this root instance. (Although as one of 13 instances it is only a fraction of total root DNS traffic). We use this traffic to represent an authoritative server.

Generality: These datasets cover each class of DNS resolver (Figure 1) and so cover very different behavior seen at different parts of the DNS system. 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. Nevertheless, we believe the diversity here is broad enough to evaluate our design.

5.2 Trace Replay and Parameterization

To evaluate connection hits for different timeout windows we replay these datasets through a simple simulator. The simulator implements the DNS server cache and an adjustable timeout window; from that we find out the number of concurrent connections and the fraction of connection hits. We study both short (10 through 60 s) and longer (120 to 480 s) window sizes. We ignore results for the first 10 minutes of trace replay to avoid transient effects due to a cold cache.

To convert numbers of concurrent connections to hardware constraints, we estimate both memory and CPU consumption per connection. We determine both experimentally. We measure the memory consumption for idle TCP/TCP connection by opening 10k simultaneous connections from a custom client to `unbound`, then measure peak heap size with `valgrind`. Measured on a computer running 64-bit Fedora 18 Linux OS, we see each TCP connection consume 260 kB, and each TLS connection 264 kB. To this, we estimate kernel buffers at about 100 kB, so we approximate the per-connection memory as 360 kB.

5.3 Concurrent Connections and Hit Fraction

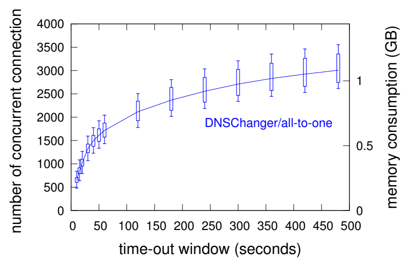
Trace replay of the three datasets provides several observations. First consider how usage changes over the course of the day, finding that variation in the number of active connections is surprisingly small. When we measure counts over one-second intervals, connections vary by ±10% for Level 3, with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

Next, we see that the three servers have *very* different absolute numbers of active connections consistent with their client populations (Figure 3a DNSChanger: for this dataset, a few thousand uncorrected users; Figure 3b: Level 3: many thousand customers per site and B-Root: potentially any global recursive resolver).

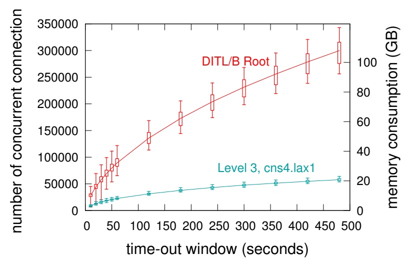
For all of our traces, Figure 3c shows connection hit fractions converge on an asymptote, showing diminishing benefits beyond a timeout windows of 100 s or so. The asymptote varies by server: with a 120 s window, DNSChanger is at 97-98%, Level 3 at 98-99%, and B-Root at 94-96%. These fractions show that *connection caching will be very successful.* We know that much network traffic is bursty and self-similar, so it is not surprising that caching is so effective.

Finally, when one compares the authoritative server (B-Root) with the recursive resolvers, we see the ultimate hit fraction is considerably smaller, consistently several percent lower for a given timeout. We believe the lower hit fraction for B-Root is due to its diverse client population and the small set of queries for which it is authoritative. We expect this result will hold for servers that provide static DNS zones. (DNS servers providing dynamic content, such as blackhole lists are likely to show different trends.)

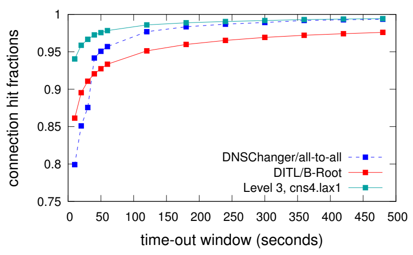
Recommendations: We propose timeouts of 60 s for recursive resolvers and 20 s for authoritative servers, informed by the data with a conservative approach to server load. We recommend that clients and servers not



(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%.)

Figure 3: Evaluation of concurrent connections and connection hit fractions.

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 and providers can experiment independently.

Using our estimated memory consumption, we can translate these recommendations into memory requirements. With 60 s for recursive resolvers and 20 s for authoritative servers, each DNSChanger server would need 0.7 GB RAM, Level 3's 9 GB, and B-Root 18 GB, based on the 75%iles in Figure 3. (These values are in addition to memory required to cache actual DNS data.) Although large, these values are well within what is typical for current server hardware.

An alternative for servers with limited memory is to set a small timeout and depend on TCP Fast Open and TLS Resume to quickly restart terminated connections.

6. CLIENT-SIDE LATENCY

For clients, the primary cost of t-DNS is the potential for additional latency due to connection setup.

Using experiments, we next examine stub-to-recursive and recursive-to-authoritative 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 latency*, and the *workload*. We first study computation time and show that, even with TLS, latency is much more significant than computation.

For latency, we consider both *stub-to-recursive* queries, where latency is usually constant and small, and *recursive-to-authoritative* queries, where latency is often large and variable. We highlight the effects of latency with two different workloads: *start and wait* (s+w) queries, where queries are send sequentially; each query is sent after the reply for the last is received, and *pipelining*, where the client sends queries as fast as possible.

Finally, we combine these experimental results with estimates of client hit rate to model the end-to-end effects on observed client latency.

6.1 Computation Costs

step	**OpenSSL**	**GnuTLS**
TCP handshake processing	0.15 ms	
TCP packet handling	0.12 ms	
TLS connection establishment	25.8 ms	13.1 ms
key exchange	13.0 ms	—
CA validation	12.8 ms	—
TLS connection resumption	—	0.7 ms
DNS resolution (from [50])	0.1–0.5 ms	

Table 3: Computational costs of connection setup and packet processing.

We next experimentally evaluate CPU consumption of TLS. The client and the server are 4-core x86-64 CPUs, running Fedora 19 with the Linux-3.12.8 kernel, connected by 1Gb/s Ethernet LAN. We test our own client and server for GnuTLS; for OpenSSL we use the Apache-2.4.6 webserver.

Because each event is short we estimate times by repeating each action many times and dividing. (We report the median of 10 trials.) 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. Our GnuTLS server uses only anonymous authentication so there we cannot evaluate CA validation. We measure GnuTLS connection resumption with 1000 trials. OpenSSL does not support TLS resumption.

Our results are in Table 3. TCP setup and DNS resolution are small (less than 1 ms). TLS setup is much more expensive (13 or 26 ms), with both key exchange and CA validation equally costly. TLS resumption, however, is only a few times more expensive than TCP.

Although TLS is computationally expensive, *TLS computation will not generally limit DNS.* Most DNS servers are bandwidth limited with only light CPU loads. We

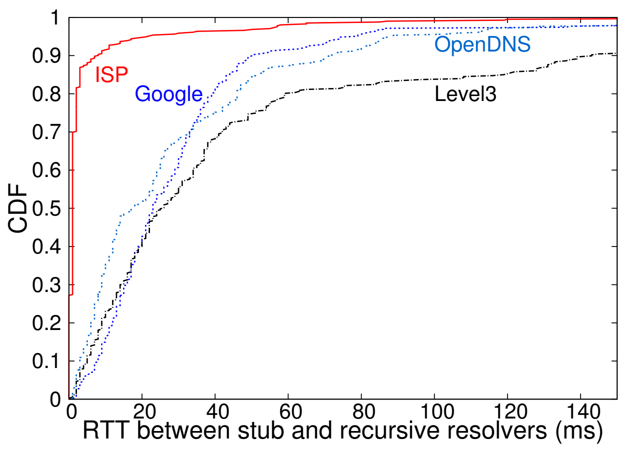


Figure 4: Measured RTTs from stub and recursive resolver from 400 PlanetLab nodes, for the ISP-provided resolver and three third-party resolvers.

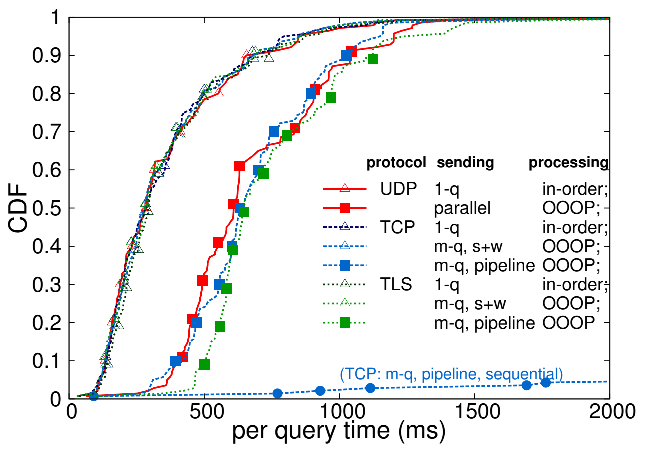


Figure 5: Median query response times for 140 different names with different protocol configurations with 1 ms client-to-server latency.

expect server memory will be a larger limit than CPU. For clients, we show (§ 6.5) that latency dominates performance, not computation.

6.2 Latency: Stub-to-Recursive Resolver

We first estimate what latencies to expect for stub-to-recursive, then compare protocol alternatives.

6.2.1 Typical Stub-to-Recursive RTTs

Stubs typically talk to a few (one to three) recursive resolvers. Recursive resolvers are usually provided as a service by ones ISP, and so they are typically nearby in the network. Alternatively, some users use third-party resolvers. These may have a higher round-trip-time (RTT) from the stub, but provide a very large cache and user population.

We latency between stub and recursive resolvers across 400 PlanetLab nodes to their local (ISP-provided) resolver, and also to three third-party DNS services (Google, OpenDNS, and Level 3). For each case, we issue the same query 7 times, each after the previous reply, and report the median result. We expect the first query to place the result in the cache, and report the median to suppress noise from interfering traffic.

Figure 4 shows the CDF of these latency measurements. This data confirms that the ISP-provided recursive resolver almost always has very low latency (80% less than 3 ms). Only a few stragglers have moderate latency (5% above 20 ms). For third-party resolvers, we see more variation, but most have fairly low latency due to distributed infrastructure. Google Public DNS provides median latency of 23 ms and the others only somewhat more distant. The tail of higher latency here affects more stubs, with 10–25% showing 50 ms or higher.

PlanetLab nodes are primarily hosted at academic sites and so likely have better-than-typical network connectivity. These observed third-party DNS latency may be lower than typical. However, with this sample of more than 300 organizations, this data provide some diversity in geography and configuration of local DNS.

6.2.2 TCP connection setup: stub-to-recursive

We next evaluate the costs of TCP connection setup for stub-to-recursive queries. Since the RTT between stub and recursive is usually small, we expect overall query latency to be dominated by the recursive-to-authoritative resolution; the (stub-to-recursive) TCP handshake should add minimal overhead.

We simulate a typical stub and recursive resolver by operating both on the same network (RTT is about 1 ms). We use our custom DNS client and the BIND-9.9.3 server with our modifications. The stub then makes 140 unique queries, randomly drawn from the Alexa Top-1000 sites [1]. We repeat these requests with DNS over different protocols: UDP, TCP and TLS. For each protocol, we evaluate different sending policies on the client side: UDP: single-query (1-q); parallel. TCP: single-query per TCP (1-q);multiple-query per TCP with start and wait (s+w); multiple-query per TCP connection (m-q) with pipelining (pipeline); and also different processing policies on the server side: in-order: queries are processed sequentially. Out-of-order processing (OOOP): queries are processed concurrently.

We report median latency across 10 trials of this experiment to avoid bias due to outliers and cross-traffic. We restart the recursive resolver before each run so all occur with a cold cache.

Figure 5 shows the results of these experiments. We first consider *UDP compared to TCP*: the left set of lines with triangles, corresponding with sequential queries (s+w) made with UDP, single-query and multi-query TCP. The lines are nearly on top of each other, demonstrating that *with small client-server RTTs, TCP setup* *time is irrelevant*; it is dwarfed by overall cost of resolving a new name.

To consider pipelining, sending multiple queries before the replies return. In Figure 5 the lines marked with squares indicate pipelining. First, we see that per-query resolution times are actually higher with pipelining than when done sequentially. This delay occurs because all 140 queries arrive at the server at nearly the same time, so they queue behind each other as they are processed by our 4-core computer. Second, we see that the delay with multi-query TCP (light blue, dashed line with filled circles at the bottom) is horrible, increasing linearly. The reason for this delay is that while both BIND-9 and Unbound can process multiple queries from UDP concurrently(OOOP), they process queries from the same TCP connection sequentially(in-order), which causes head of line blocking: later queries get blocked by previous ones. While correct, current resolvers *are not optimized for high-performance TCP* query handling.

DNS specifications require support for out-of-order queries and responses, even though current implementations do not process queries this way (see prior discussion in § 4.1). Here we approximate native resolvers support for out-of-order TCP queries by placing a proxy resolver on the same computer as the real resolver. The proxy receives queries from the stub over TCP, then forwards them to recursive resolver over UDP. This approach leverages current native out-of-order UDP processing and incurs no fragmentation since UDP is sent inside the same machine (over the loopback interface). The proxy then returns replies over TCP to the stub, but in whatever order the recursive resolver generates results. The light blue, dashed line with filled squares in Figure 5 shows the effects of this improvement: TCP and UDP performance are again equivalent. In fact, for about half of the cases *multi-query TCP shows equivalent performance to UDP* (within measurement noise).

6.2.3 TLS privacy: stub-to-recursive

Connection establishment for TLS is much more expensive than TCP, requiring additional round trips and computation to establish a session key. We repeat our experiments from the prior section, this time comparing UDP with TLS. For consistency with our out-of-order experiments, we place our proxy resolver on the same machine recursive resolver.

The dotted lines in Figure 5 show TLS performance. Although the TLS handshake adds 3 RTTs to the query, this cost is negligible relative to the cost of the recursive-to-authoritative query (ten vs. hundreds of ms). With sequential queries, TLS performance is almost the same as UDP and TCP (lines with triangles in the left overlap with each other). With both pipelining and out-of-order processing, TLS performance is only slightly slower than UDP (the light green, dotted squares are only 10–50 ms behind the red, solid squares).

6.2.4 Overall Stub-to-Recursive

In summary, this section shows that when the stub and recursive resolvers are close to each other the extra packet exchanges add very small latency to the query, and even the TLS connection setup cost is dwarfed by the costs involved in making distributed DNS queries to authoritative name servers. Second, minimizing connection setup requires reusing connections, and we showed that head-of-line blocking in the TCP processing of current resolver implementations adds significant latency. Current resolvers have most of the machinery to fix this problem, and our experiments show out-of-order processing allows DNS performance with both TCP and TLS to be very close to that of simple UDP. In short, *the cost of connections between nearby stub and recursive resolvers is in the noise.*

6.3 Latency: Recursive Resolver to Authoritative Server

We next turn to latency we expect between the recursive resolvers and authoritative name servers. While stubs query only a few, usually nearby recursive resolvers, authoritative servers are distributed around the globe and so the recursive/authoritative round-trip times are both larger and more diverse.

6.3.1 Typical Recursive-to-Authoritative RTTs

To estimate typical recursive-to-authoritative RTTs, we again turn to the Alexa top 1000 sites. We query each from four locations: our institution in Los Angeles (`isi.edu`), and PlanetLab sites in China (`www.pku.edu.cn`), UK (`www.cam.ac.uk`), and Australia (`www.monash.edu.au`). These sites show the effects of geographic diversity; we do not claim they are representative.

For each site we query each domain name. We use `dig +trace` to resolve each domain component from the root to the edge, including DNSSEC where possible. We report the median of 10 repetitions of the query time of the last step to estimate of best-case recursive-to-authoritative RTTs. This method represents performance as if higher layers were already cached by the recursive resolver, and median provides some robustness to competing traffic and random selection from multiple name servers.

Figure 6 shows the results of this experiment. We see that the U.S. and UK sites are close to many authoritative servers, with median RTT of 45 ms, but it also has a fairly long tail, with 35% more than 100 ms. The Chinese site has generally longer RTTs, with only 30% responding in 100 ms. While many large sites operate Asian mirrors, many don't. The Australian site shows a sharp shift with about 20% of sites less than 30 ms, while the remaining 150 ms or longer. This jump is due to the long propagation latency for services without sites physically in Australia.

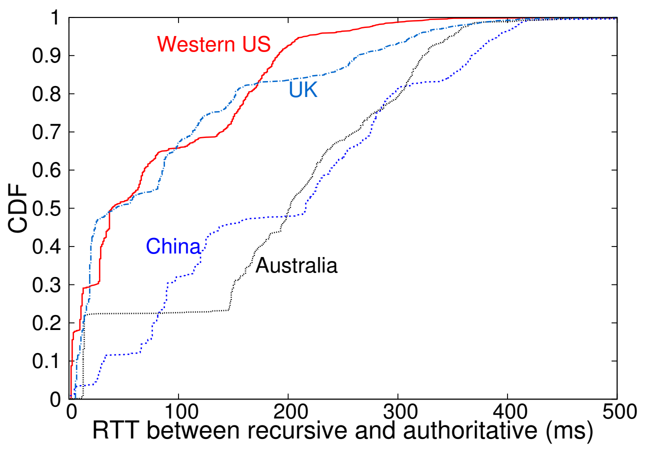


Figure 6: RTT between recursive resolver and authoritative name server

Overall, the important difference compared to stub-to-recursive RTTs is that while a few authoritative servers are close (RTT < 30 ms), many will be much further.

6.3.2 TCP connection setup: recursive-to-authoritative

With noticeably larger RTTs to authoritative servers compared to the stub/recursive RTTs, we expect to see a much higher overhead for connection negotiation with TCP and TLS.

To evaluate query latencies with larger RTTs between client and server, we set up a DNS authoritative server for an experimental domain and queried it from a client 35 ms (8 router hops on a symmetric path) away. We operate a BIND-9.9.3 server as the authoritative name server at one site. We then query this name server directly, 140 times, while varying the protocol in use. As before, we repeat this experiment 10 times and report the median observed value. Standard deviations are very small and so are not shown.

We first compare UDP (UDP:1-q), single-query per TCP (TCP: 1-q), and multiple-query per TCP (TCP: m-q), the lines in Figure 7 marked with open triangles. We see that all queries made by single-query-per-TCP with in-order processing (TCP:1-q,in-order) take about 70 ms, exactly two RTTs, due to TCP's connection establishment followed by the request and response. Both UDP and multiple-query-per-TCP with start and wait (TCP:m-q,s+w) take 35 ms, one RTT per query. This difference shows the *importance in reusing TCP connections for multiple queries* to avoid connection setup latency, highlighting the need for good connection hit ratios (§ 5).

We next consider pipelining multiple queries over a single TCP connection and supporting out-of-order processing (TCP:m-q, pipelining, OOOP). Basic UDP already supports both of these. To match our prior experiment we implement these options for TCP with a

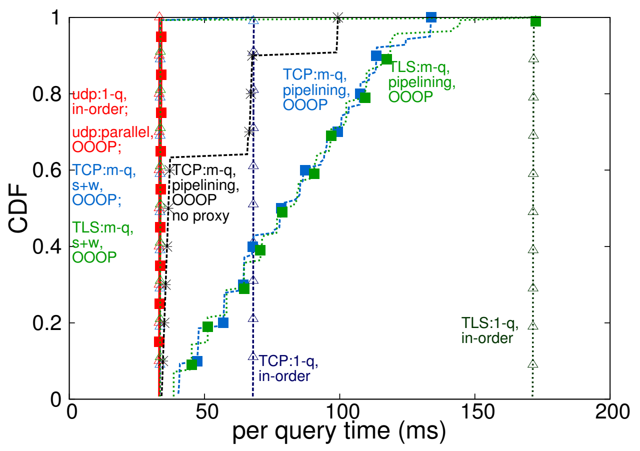


Figure 7: Median per-query response times for 140 repeated queries to the same name with a 35 ms RTT between client and server with different protocol configurations with 1 ms client-to-server latency.

proxy server running on the same computer as the authoritative server, and we plot these results as the line with filled squares in Figure 7. In this case, 10% of the queries complete with performance similar to UDP, while the other queries take slightly longer, in steps. We examined packet traces and verified each step is a single TCP packet with 12 or 13 responses. Thus the delay is due to synchronization overhead as all 140 responses, processed in parallel, are merged into a single TCP connection in our proxy. When we remove our proxy and use unbound directly (the black dotted "TCP:m-q, pipelining, OOOP, no proxy" line) performance is limited by the 8-packet TCP initial window. For this special case of more than 100 queries arriving simultaneously, a single connection can add some latency.

6.3.3 TLS privacy: recursive-to-authoritative

Next we consider the addition of TLS. Use of TLS from recursive-to-authoritative is a policy decision; one might consider aggregation at the recursive resolver to provide sufficient anonymity, or one might employ TLS on both hops as a policy matter (for example, as with HTTPS Everywhere [17]). Here we consider the effects on latency of full use of TLS.

In Figure 7, the green, dotted lines show TLS usage. Without pipelining (the line on the right: TLS:1-q, in-order), TLS always takes 175 ms (5 round trips). This corresponds to one round trip to setup TCP, one to negotiate DNS-over-TLS (§ 4.2.2), two for the TLS handshake, and then the final private query.

However, once established, the TLS connection can easily be reused. If we reuse the existing TLS connection and send queries without pipelining (TLS:m-q, s+w, OOOP), TLS performance is *identical* to UDP

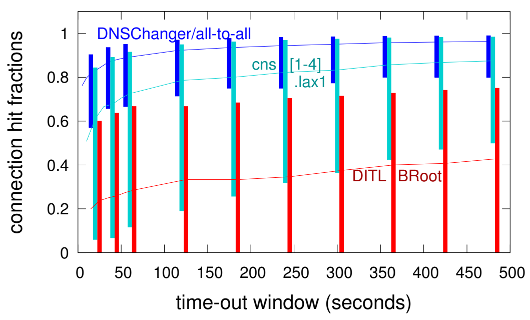


Figure 8: Median client-side connection hit fractions with quartiles.

with a mean latency of one RTT, except for the first TLS query. This result shows that the expense of encryption is tiny compared to moderate round-trip delays, when we have an established connection.

Finally, when we add out-of-order processing, we see similar stair-stepped behavior as with TCP, again due to synchronization over a single connection and our unoptimized proxy. The light-green, dotted line with filled squares shows connection reuse, pipelining, and out-of-order processing; with this combination TLS performance again roughly equivalent to TCP, within measurement noise.

6.3.4 Overall Recursive-to-Authoritative

This section showed that *round-trip latency* dominates performance for queries from recursive resolvers to authoritative name servers. Latency is incurred in connection setup, with TCP adding one additional RTT and TLS three more. This latency is very expensive, but it can be largely eliminated by connection reuse.

6.4 Client connection hit fractions

Our experiments that show query latency evaluate the benefits of connection reuse, but they don't address how likely a client will find an existing connection. In § 5.3 we reported connection hit fractions from the server's perspective and found them very high because frequent queriers often have open connections. We next see that *client connection hit fractions* are lower, because there are many clients that query infrequently.

To evaluate client connection hit fractions, we replay our three DNS traces through the simulator from § 5.3, 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 some clients (those making few requests) will need to restart connections frequently. This observation indicates the importance to TCP Fast Open and TLS Resumption, mechanisms that allow these clients to assume the burden of retaining state for this process.

6.5 Modeling End-to-End Latency for Clients

With this data we can now model the expected (average) *end-to-end* latency for DNS users, exploring the interaction of stub, recursive and authoritative resolvers as we vary the protocols in use.

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

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

From Figure 7 we know that $Q_{c\sigma}$ the same for all methods when the connection is open: it is about one client-server RTT, or $R_{c\sigma}$. Setup $S^C_{c\sigma}$ is 0 for UDP, $R_{c\sigma}$ for TCP, and $4R_{c\sigma} + S^{cpu}_{\sigma}$ for TLS (1 for TCP, 1 for TLS neogitation, and 2 for TLS handshake), where S^{cpu}_{σ} is computationally approximated at 25.8 ms (Table 3). We can estimate $P^C_{c\sigma}$ given a timeout window from client-side trace analysis (Figure 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 make several (N^Q_r) queries to authoritative resolvers (each taking L_{ra}) to fill its cache:

$$L_{sa} = L_{sr} + (1 - P^N_r)N^Q_r L_{ra} \qquad (2)$$

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 $P^C_{sr} = 0.72$ and $P^C_{ra} = 0.24$. Prior studies of recursive resolvers suggest P^N_r ranges from 71% to 89% [26].

We determine N^Q_r by feeding the Alexa top-1000 sites into BIND-9.9.3 and observing how many outgoing queries emerge. We repeat this experiment 10 times, starting each run with a cold cache. The mean value of $N^Q_r = 7.24$ (standard deviation 0.036) including 0.09 for query retries. We round N^Q_r 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 Lookaside Validation). Some sites that use content-delivery networks

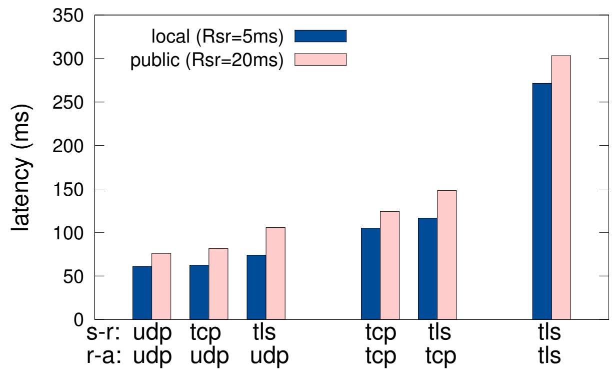


Figure 9: Modeling end-to-end-performance for local ($R_{sr} = 5\,\text{ms}$) and third-party ($R_{sr} = 20\,\text{ms}$) recursive resolvers with different protocols. ($R_{ra} = 40\,\text{ms}$, $P_r^N = 0.8$, $N_r^Q = 7$)

require several queries as they 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 deployed at stub-to-recursive and recursive-to-authoritative (each a group of bars). We consider two locations for the recursive resolver: ISP-provided with R_{sr} = 5ms, and a third-party resolver with R_{sr} = 20ms (median for U.S. or Europe).

First we consider the local resolver. From the model we see that *use of TCP and TLS to the local resolver adds moderate latency*: current DNS has mean of 61 ms, while TCP is 62 ms and TLS only 74 ms (21% worse), and using UDP upstream. Second, we see that use of connections between recursive and authoritative is very expensive: TCP is 116 ms (1.9×), and TLS is 272 ms (4.46×), with TLS to recursive. The recursive resolver must make several queries to the authoritative servers, at large RTTs, with a much lower connection hit fraction. The extra round-trips for connection setup show in the TCP/TLS case.

When we turn to a third-party resolver, we see a similar trend, but the higher latency between stub and recursive raises the cost of TCP and TLS. For example TLS to recursive (with UDP to authoritative), is 39% slower than UDP.

7. CONCLUSION

This paper shows that *connectionless DNS has outlived its stay.* We summarized the range of problems that have accumulated due to DNS' focus on single-packet, connectionless communication: privacy limitations, security concerns, and operational constraints. Although work-arounds exist for individual problems, DNS is too useful and today's Internet too dangerous to be limited to one packet.

Surprisingly, we show that *end-to-end latency of connection-oriented DNS approaches connectionless.* Our models show TLS to the recursive resolver is only about 21% slower with UDP to the authoritative server, and 90% slower with TLS to recursive and TCP to authoritative. This performance depends on the design and implementation we describe.

8. ACKNOWLEDGMENT

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 webpages 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 and Bill Manning provided comments on the paper.

9. REFERENCES

[1] Alexa. `http://www.alexa.com/`.

[2] E. Allman, J. Callas, M. Delany, M. Libbey, J. Fenton, and M. Thomas. DomainKeys Identified Mail (DKIM) signatures. RFC 4871, May 2007.

[3] Anonymous. The collateral damage of internet censorship by DNS injection. *SIGCOMM CCR*, June 2012.

[4] Arbor Networks. Worldwide infrastructure security report. Technical report, Arbor Networks, Sept. 2012.

[5] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033, Mar. 2005.

[6] R. Bellis. DNS Transport over TCP - Implementation Requirements. RFC 5966, Aug. 2010.

[7] R. Beverly, R. Koga, and kc claffy. Initial longitudinal analysis of IP source spoofing capability on the Internet. *Internet Society*, July 2013.

[8] S. Bortzmeyer. DNS privacy problem statement. Internet draft, Dec. 2013.

[9] M. Butkiewicz, H. V. Madhyastha, and V. Sekar. Understanding website complexity: Measurements, metrics, and implications. In *IMC*, pages 313–328, Nov. 2011.

[10] J. Damas, M. Graff, and P. Vixie. Extension mechanisms for DNS (EDNS(0)). RFC 6891, Apr. 2013.

[11] M. Dempsky. DNSCurve: Link-level security for the Domain Name System. Internet draft, Feb. 2010.

[12] T. Dierks and E. Rescorla. The Transport Layer Security TLS Protocol Version 1.2. RFC 5246,

Aug. 2008.

[13] DNS-OARC. `https://www.dns-oarc.net/`.

[14] R. Droms. Dynamic host configuration protocol. RFC 2131, Mar. 1997.

[15] 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.

[16] W. Eddy. TCP SYN flooding attacks and common mitigations. RFC 4987, Aug. 2007.

[17] Electronic Frontier Foundation. Encrypt the web with HTTPS everywhere. Web page `https://www.eff.org/https-everywhere`, Aug. 2011.

[18] T. Faber, J. Touch, and W. Yue. The TIME-WAIT state in TCP and its effect on busy servers. INFOCOMM, 1998.

[19] L. Green. Common crawl enters a new phase. Common Crawl blog `http://www.commoncrawl.org/common-crawl-enters-a-new-phase/`, Nov. 2011.

[20] G. Greenwald. NSA collecting phone records of millions of Verizon customers daily. *The Guardian*, June 2013.

[21] A. Herzberg and H. Shulmanz. Fragmentation considered poisonous. IEEE-CNS, Oct. 2013.

[22] P. Hoffman and J. Schlyter. The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA. RFC 6698, Aug. 2012.

[23] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, and D. Wessels. Starting TLS over DNS. Work in progress (Internet draft draft-start-tls-over-dns-00), Jan. 2014.

[24] G. Huston. A question of protocol. Talk at RIPE '67, 2013.

[25] ICANN. Root server attack on 6 February 2007. Technical report, Mar. 2007.

[26] J. Jung, E. Sit, H. Balakrishnan, and R. Morris. DNS performance and the effectiveness of caching. *ACM/IEEE ToN*, 10, Oct. 2002.

[27] A. Jungmaier, E. Rescorla, and M. Tuexen. Transport Layer Security over Stream Control Transmission Protocol. RFC 3436, Dec. 2002.

[28] D. Kaminsky. It's the end of the cache as we know it. Presentation, Black Hat Asia, Oct. 2008.

[29] C. A. Kent and J. C. Mogul. Fragmentation considered harmful. In *SIGCOMM*, Aug. 1987.

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

[31] 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`.

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

[33] C. Metz. Comcast trials (domain helper service) DNS hijacker. The Register, July 2009.

[34] P. Mockapetris. Domain names—implementation and specification. RFC 1035, Nov. 1987.

[35] 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*, Sept. 1997.

[36] OpenDNS. Dnscrypt. `http://www.opendns.com/technology/dnscrypt`.

[37] OpenDNS. Opendns website. `www.opendns.com`, 2006.

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

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

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

[41] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347, Jan. 2012.

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

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

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

[45] 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`.

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

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

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

[49] P. Vixie and A. Kato. DNS referral response size issues. Internet draft, May 2012.

[50] D. Wessels and G. Sisson. Root zone augmentation and impact analysis. Presentation, NANOG 47, 2009.

[51] S. Woolf and D. Conrad. Requirements for a Mechanism Identifying a Name Server Instance. RFC 4892, June 2007.

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

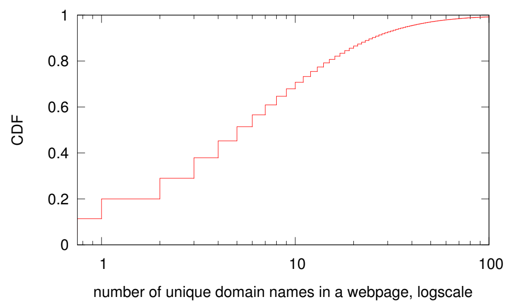


Figure 10: CDF of number of unique hostnames per web page. (Dataset: 40584944 page sample from CommonCrawl-002 [19]).

APPENDIX

A. DOMAIN NAMES PER WEB PAGE

To demonstrate the need for pipelining DNS queries for end-users (§ 4.1), we examined about 40M web pages (about 1.4%) from a sample of CommonCrawl-002 [19]. The sample is selected arbitrarily, so we do not expect any bias. We count the number of unique domain names per page.

Figure 10 shows the results: to confirm that 62% of web pages have 4 or more unique domain names, and 32% have 10 or more.

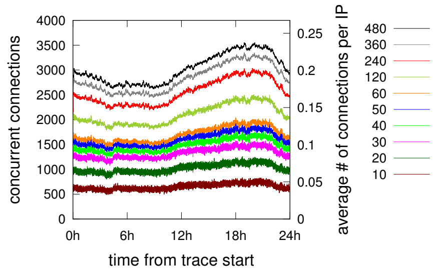
B. ADDITIONAL DATA FOR SERVER-SIDE LATENCY

Figure 11 and shows the number of connections over the day for all three datasets, Figure 12 shows the hit fraction over the day for all three datasets, expanding on the data in Figure 3.

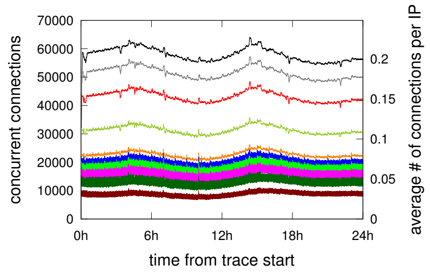
Figure 13 summarizes the data in Figure 12 by quartiles.

C. ADDITIONAL DATA FOR CLIENT-SIDE LATENCY

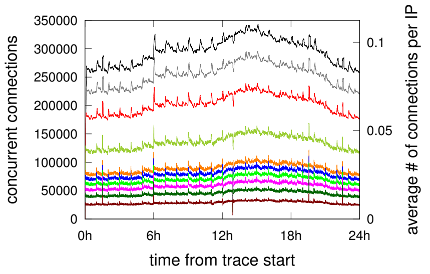
Figure 14 shows the data that underlies Figure 8.



(a) Dataset: DNSChanger

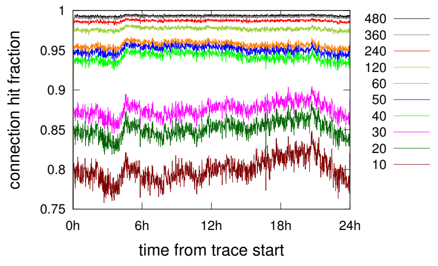


(b) Dataset: Level 3, cns4.lax1.

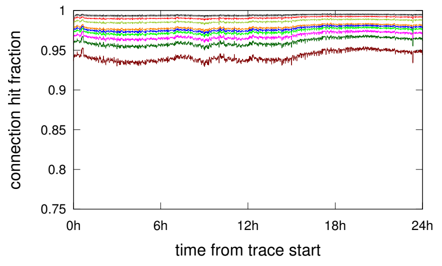


(c) Dataset: B Root

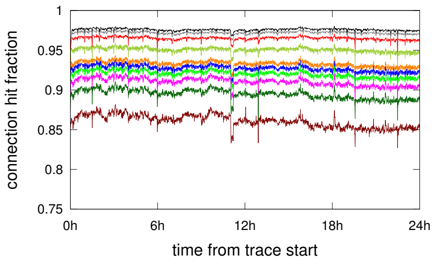
Figure 11: The number of concurrent connections given by different time-out window sizes.



(a) Dataset: DNSChanger

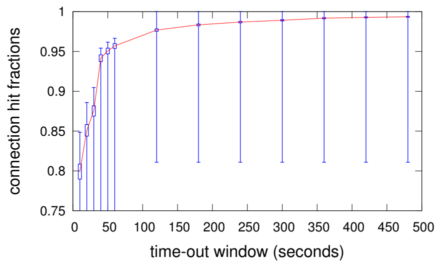


(b) Dataset: Level 3, cns4.lax1.

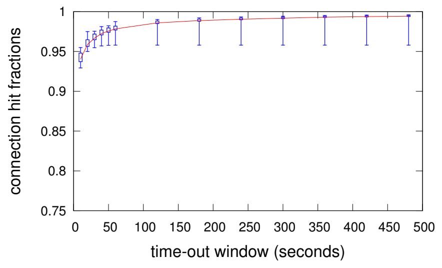


(c) Dataset: B Root

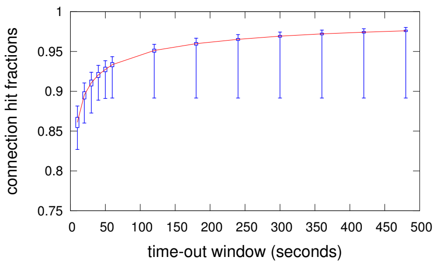
Figure 12: Server-side hit ratio (connection reuse) of queries given by different time-out window sizes



(a) Dataset: DNSChanger, all-to-one

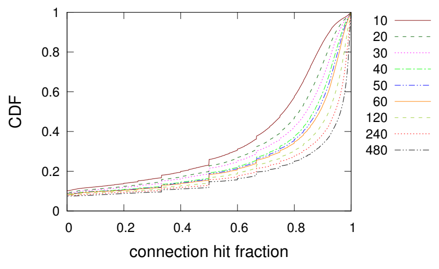


(b) Dataset: Level 3, cns4.lax1.

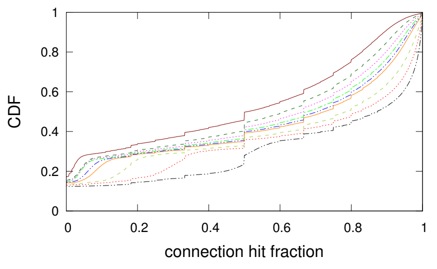


(c) Dataset: B Root

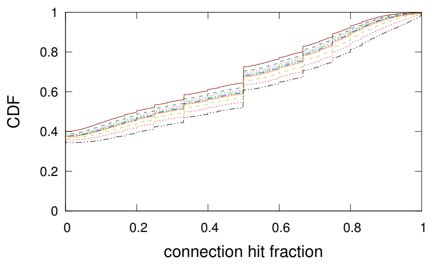
Figure 13: Quartile plot of server-side connection hit fraction.



(a) Dataset: DNSChanger



(b) Dataset: Level 3, cns[1-4].lax1.



(c) Dataset: B Root

Figure 14: CDF of client-side connection hit fraction