DNS over TCP and TLS

draft-hzhwm-dprive-start-tls-for-dns-00

John Heidemann and Sara Dickinson

Joint work with Liang Zhu, Zi Hu, Duane Wessels, Allison Mankin, Willem Toorop

USC/ISI, Verisign Labs, and Sinodun in collaboration with NLnet Labs, getdns

IETF91 / 11 November 2014
Our Goals

• DNS protocol changes
  – encouraging TCP
  – STARTTLS to initiate TLS
• implementation choices for good performance
• performance study to confirm costs
  – client latency: only modestly more
  – server memory: well within current hardware
Why DNS over TCP and TLS

• here: protecting privacy
  – encrypt stub-to-recursive queries

• use of TCP helps in other regards
  – defanging DoS
    • prevent attacks on the DNS server: use existing TCP anti-DoS (SYN cookies)
    • reducing attacks on others: TCP avoids amplification attacks
  – relaxing limits of UDP packet sizes: TCP
Protocol Changes: Goals

- minimize change
- reuse existing approaches
- follow IETF norms

(as boring as possible)
Protocol Changes: Goals

• minimize change
• reuse existing approaches (as boring as possible)
• follow IETF norms

• implications:
  – reuse TLS: Transport Layer Security
  – add a STARTTLS-like “upgrade”
  – dedicated port too, if that is acceptable under IANA Port Review (RFC 6335)
  – innovation: careful implementation
SMTP before STARTTLS

C & S: open TCP connection
    S: 220 mail.imc.org SMTP service ready
C: EHLO mail.example.com
    S: 250-mail.imc.org hi, extensions are: -8BITMIME -STARTTLS DSN

problem: cleartext mail is snoop-able (fix: TLS)

C: MAIL FROM:<sender@mail.example.com>
    S: 250 2.1.0 <sender@mail.example.com>... Sender OK
C: RCPT TO:<destination@mail.example.com>
    S: 250 2.1.5 <destination@mail.example.com>
C: <send mail contents>

DNS over TCP and TLS
**SMTP with STARTTLS (RFC-3207)**

C & S: open TCP connection

S: 220 mail.imc.org SMTP service ready

C: EHLO mail.example.com

S: 250-mail.imc.org hi, extensions are: -8BITMIME -STARTTLS DSN

C: STARTTLS

S: 220 Go ahead

C & S: <negotiate a TLS session with a new session key, in binary>

C: EHLO mail.example.com

S: 250-mail.imc.org hello, extensions are: -8BITMIME DSN

C: MAIL FROM:<sender@mail.example.com>

S: 250 2.1.0 <sender@mail.example.com>... Sender OK

C: RCPT TO:<destination@mail.example.com>

S: 250 2.1.5 <destination@mail.example.com>

C: <send mail contents>

**prologue: in clear (no privacy here)**

**transition to TLS**

**contents now private**

this example: SMTP; idea used for IMAP, POP3, FTP, XMPP, LDAP, NNTP…
Our STARTTLS for DNS
(draft-hzhwm-dprive-start-tls-for-dns-00)

C & S: open TCP connection

C: QNAME="STARTTLS", QCLASS=CH, QTYPE=TXT
   with the new TO bit set in EDNS options

   S: RCODE=0, TXT="STARTTLS", with the TO bit set

C & S: <negotiate a TLS session, get new session key, in binary>

   contents now private

C: <send actual query>

   S: <reply to actual query>

pros: no new port (from IANA, or in firewalls)
cons: extra RTT; middleboxes may not like encrypted traffic
      (other signaling approaches are possible)
Protocol Details

• keeps standard DNS framing before and after TLS upgrade
  – allows easy retrofit to existing resolver software
• use dummy query to avoid leaking information
• i-d says TO bit is only signaling
• pre-IANA, we use STARTTLS QNAME and no TO bit in our implementations
Our Goals

• DNS protocol changes
  – encouraging TCP
  – STARTTLS add TLS

• **implementation choices for good performance**

• performance study to confirm costs
  – client latency: only modestly more
  – server memory: well within current hardware
Careful Implementation Choices

- problem: no tuning of DNS TCP for queries (until now!)
  - see draft-dickinson-dnsop-5966-bis-00 (on DNSOP agenda today)
- connection reuse (or restart)
  - persistent connections
  - TCP fast open
  - TLS resumption
- query pipelining
- query reordering (out-of-order processing)

_details in Sara’s talk, and supplemental slides_
Our Goals

• DNS protocol changes
  – encouraging TCP
  – STARTTLS add TLS
• implementation choices for good performance
• performance study to confirm costs
  – client latency: only modestly more
  – server memory: well within current hardware

Connection Reuse Helps? (YES!)

what fraction of queries find open TCP connections?

method: replay 3 traces: recursive (DNSChanger, Level3) and authoritative (B-Root)

(graph shows medians, quartiles are tiny)

120s timeout => >94% connection reuse (reuse is effective!)

we propose 20s/60s (conservative) => still >85% connection reuse

conclusion: connection reuse is often helpful

DNS over TCP and TLS
Cost of Connection Reuse? (ok!)

how many connections?
how much memory?

**method**: replay same 3 traces (here we show 2 biggest)

experimental estimate of memory: 360kB/connection (very conservative)

(graph shows medians and quartiles)

120s timeout => 16 to 40GB RAM

we propose 20s/60s (conservative) => 3.6GB from study for recursive (L3), 7.4GB for root (B)

**conclusion**: connection reuse is often helpful and it’s not too costly (easy to add server parallelism if needed)
Latency: CPU Cost

- we used micro-benchmarks to study CPU cost

<table>
<thead>
<tr>
<th>step</th>
<th>OpenSSL</th>
<th>GnuTLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP handshake processing</td>
<td>0.15 ms</td>
<td></td>
</tr>
<tr>
<td>TCP packet handling</td>
<td>0.12 ms</td>
<td></td>
</tr>
<tr>
<td>TLS connection establishment</td>
<td>25.8 ms</td>
<td>8 ms</td>
</tr>
<tr>
<td>key exchange</td>
<td>13.0 ms</td>
<td>6.5 ms</td>
</tr>
<tr>
<td>CA validation</td>
<td>12.8 ms</td>
<td>1.5 ms</td>
</tr>
<tr>
<td>TLS connection resumption</td>
<td>1.2 ms</td>
<td>1.4 ms</td>
</tr>
<tr>
<td>DNS resolution (from [52])</td>
<td>0.1–0.5 ms</td>
<td></td>
</tr>
</tbody>
</table>

TLS setup is noticeable, but RTT (40-100+ms) more impt.
Latency: Stub to Recursive

TCP and TLS vs. UDP? effects of implementation choices?
with short (1ms, left) and medium (35ms, right) RTTs

method: live experiments of random 140 names from Alexa top 1000; stub-recursive RTT=1ms
(graph shows medians and quartiles)

TCP and TLS: as fast as UDP
(why? 1ms RTT is ~free)
Latency: Stub to Recursive

TCP and TLS vs. UDP? effects of implementation choices?

with short (1ms, left) and medium (35ms, right) RTTs

method: live experiments of random 140 names from Alexa top 1000

(graph shows medians and quartiles)
End-to-End Latency: Methodology

• controlled experiments are hard
  – variable stub query timing
  – caching at recursive resolver
  – different RTTs (many stubs and authoritatives)

• approach: model expected latency
  – i.e., just averages
  – median connection reuse from trace replay
  – other parameters from experiments
End-to-End Latency: Results

Protocol choices: stub-recursive and recursive-authoritative

Method: modeling; vary stub-recursive RTT; assumes all optimizations (TCP FO, TLS resumption, pipelining, OOOP)

(Graph shows expected values, plus slowdown relative to case (a), UDP/UDP)

TLS (s-r, 60s t.o.) + UDP (r-a)
5 to 34% slower: modest cost -> most benefit
Our Goals

• DNS protocol changes
  – encouraging TCP
  – STARTTLS add TLS

• **implementation choices for good performance**

• performance study to confirm costs
  – client latency: only modestly more
  – server memory: well within current hardware
T-DNS Implementation Project Recap

• Aim: Running T-DNS code!
• People: Verisign Labs, Sinodun, NLnet Labs, getdns team, USC-ISI, …..

• Implementation Website:
  https://portal.sinodun.com/wiki/display/TDNS/T-DNS+Project+Homepage

• Past Presentations:
  DNSE at IETF89
  DNS-OARC Spring 2014 Workshop
  https://indico.dns-oarc.net//contributionDisplay.py?contribId=11&confId=19
Implementation Status

- initial prototyping
  - [http://www.isi.edu/ant/software/index.html](http://www.isi.edu/ant/software/index.html)
  - digit: t-DNS client queries
  - (also client and server-side proxies; supports full protocol and cert authentication, but not for production use)

- current phase: targeting production software
  - LDNS (drill) / Unbound / NSD (NLnet Labs)
  - getdns ([http://getdnsapi.net/](http://getdnsapi.net/))

- next phase includes BIND

- implementation notes
  - current code uses only dummy query (qname=STARTTLS, CH/TXT) to negotiate
    - use of TO bit pending IANA allocation
  - TLS-1.1 or better only (not SSL) as per UTA BCP
  - work-in-progress, still to do: certificate authentication

DNS over TCP and TLS
Performance and Functionality

• current focus: functionality
  – T-DNS (TLS)
  – TCP Fast open (reduces latency)
  – TCP connection re-use, and pipelining
  – query reordering (out-of-order processing)
Query Pipelining

send several queries immediately (not stop-and-wait)

connection reuse without pipelining

width=0.5cm height=0.5cm

connection reuse with pipelining

width=0.5cm height=0.5cm

DNS over TCP and TLS

pipelining matters:
62% of web has 4+ domain names
(dataset: common crawl)
Out-of-Order Processing

process queries on same connection in parallel

out-of-order processing
queries run in parallel

reply as soon as possible (maybe reorder)

out-of-order matters:
avoid head-of-line blocking

in-order (only)

out-of-order (only)

(stub)

(recursive)

(authoritative) (for Q1)

(authoritative) (for Q2)

q1, q2

q1

q2

a1

a2

a1

a2

q2 delayed waiting for a1 (+1 RTT)
# Current Status (Detailed)

<table>
<thead>
<tr>
<th>Software</th>
<th>digit</th>
<th>LDNS</th>
<th>getdns</th>
<th>Unbound</th>
<th>NSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>client</td>
<td>client (drill)</td>
<td>stub</td>
<td>recursive*</td>
<td>server</td>
</tr>
<tr>
<td>TLS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T-DNS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TFO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conn reuse</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pipelining</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Dark Green**: Latest stable release supports this
- **Light Green**: Patch available
- **Yellow**: Patch in progress, or requires building a patched dependancy
- **Grey**: Not applicable or not planned

* *getdns in its recursive mode uses libunbound*
Demo Time

• patched version of *drill* querying patched *Unbound*
  – regular DNS UDP/TCP query
  – DNS query over TLS (dedicated port)
  – T-DNS (STARTTLS upgrade on TCP port 53)
  – [connection reuse/pipelining]
  – [TCP Fast Open]
    • STARTTLS goes in SYN; Linux only

• wireshark screenshots at the end
T-DNS Next Steps

• more information:
  – tech report ISI-TR-2014-693
    www.isi.edu/~johnh/PAPERS/Zhu14b/
• code:
  – client, client & server proxies, unbound patch
  – http://tdns.net
  – http://www.isi.edu/ant/software/
  – interoperability meeting tonight—come by for demo
  – working to get patches upstream
  – Bind implementation will begin next
• i-d for WG to consider adopting
  – draft-hzhwm-dprime-start-tls-for-dns-00
Appendices
Wireshark Screenshots
Supplemental Slides
TCP Query

3-way handshake

session take down
TCP Query

DNS query and response

TCP ACKs
TCP connection re-use

<table>
<thead>
<tr>
<th>No.</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>55991</td>
<td>53 TCP</td>
<td>[SYN]</td>
<td>68</td>
<td>55991-53 Seq=0 Win=65535 Len=0 MSS=16344 WS=32 TVal=908459743 TSecr=0</td>
</tr>
<tr>
<td>2</td>
<td>53</td>
<td>55991 TCP</td>
<td>[SYN, ACK]</td>
<td>68</td>
<td>53-55991 Seq=0 Ack=1 Win=65535 Len=0 MSS=16344 WS=32 TVal=908459743</td>
</tr>
<tr>
<td>3</td>
<td>55991</td>
<td>53 TCP</td>
<td>[ACK]</td>
<td>56</td>
<td>55991-53 Seq=1 Ack=1 Win=408288 Len=0 TVal=908459743 TSecr=908459743</td>
</tr>
<tr>
<td>5</td>
<td>55991</td>
<td>53 DNS</td>
<td>[Standard query]</td>
<td>87</td>
<td>0x8141 A example.com</td>
</tr>
<tr>
<td>7</td>
<td>53</td>
<td>55991 DNS</td>
<td>[Standard query response]</td>
<td>151</td>
<td>0x8141 A 93.184.216.119</td>
</tr>
<tr>
<td>9</td>
<td>55991</td>
<td>53 DNS</td>
<td>[Standard query]</td>
<td>87</td>
<td>0x9b5 A sinodun.com</td>
</tr>
<tr>
<td>11</td>
<td>53</td>
<td>55991 DNS</td>
<td>[Standard query response]</td>
<td>148</td>
<td>0x9b5 A 88.98.24.67</td>
</tr>
<tr>
<td>13</td>
<td>55991</td>
<td>53 DNS</td>
<td>[Standard query]</td>
<td>91</td>
<td>0xc980 A <a href="http://www.example.com">www.example.com</a></td>
</tr>
<tr>
<td>15</td>
<td>53</td>
<td>55991 DNS</td>
<td>[Standard query response]</td>
<td>155</td>
<td>0xc980 A 93.184.216.119</td>
</tr>
<tr>
<td>17</td>
<td>55991</td>
<td>53 TCP</td>
<td>[FIN, ACK]</td>
<td>56</td>
<td>55991-53 Seq=98 Ack=287 Win=408000 Len=0 TVal=908459744 TSecr=908459743</td>
</tr>
<tr>
<td>20</td>
<td>53</td>
<td>55991 TCP</td>
<td>[FIN]</td>
<td>56</td>
<td>55991-53 Seq=287 Ack=99 Win=408192 Len=0 TVal=908459744 TSecr=908459743</td>
</tr>
</tbody>
</table>

Multiple DNS queries-responses on same TCP connection
TCP pipelining (getdns 0.1.5)

Multiple DNS queries sent together:
- responses processed when they arrive
- each query in own packet here
- could have multiple queries in one packet
TLS (port 443) - handshake

- certificate
- cipher spec
- session ticket
TLS (port 443) - DNS query

- Wireshark can decrypt if given the key

Encrypted DNS query
T-DNS - STARTTLS dummy query

- STARTTLS query
- Server is T-DNS aware and enabled -> STARTTLS response
T-DNS - TLS handshake

Server supports STARTTLS - let's do a TLS handshake

<table>
<thead>
<tr>
<th>No.</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>56014</td>
<td>53</td>
<td>TCP</td>
<td>68</td>
<td>56014-53 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS=32 TSval=908742506 TSecr=0</td>
</tr>
<tr>
<td>2</td>
<td>53</td>
<td>56014</td>
<td>TCP</td>
<td>68</td>
<td>56014-53 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16344 WS=32 TSval=908742506 TSecr=908742506</td>
</tr>
<tr>
<td>3</td>
<td>56014</td>
<td>53</td>
<td>TCP</td>
<td>56</td>
<td>56014-53 [ACK] Seq=1 Ack=1 Win=408288 Len=0 TSval=908742506 TSecr=908742506</td>
</tr>
<tr>
<td>5</td>
<td>56014</td>
<td>53</td>
<td>TLSv1.2</td>
<td>95</td>
<td>Continuation Data</td>
</tr>
<tr>
<td>7</td>
<td>53</td>
<td>56014</td>
<td>TLSv1.2</td>
<td>116</td>
<td>Continuation Data</td>
</tr>
<tr>
<td>9</td>
<td>56014</td>
<td>53</td>
<td>TLSv1.2</td>
<td>363</td>
<td>Client Hello</td>
</tr>
<tr>
<td>11</td>
<td>53</td>
<td>56014</td>
<td>TLSv1.2</td>
<td>1148</td>
<td>Server Hello, Certificate, Server Hello Done</td>
</tr>
<tr>
<td>13</td>
<td>56014</td>
<td>53</td>
<td>TLSv1.2</td>
<td>374</td>
<td>Client Key Exchange, Change Cipher Spec, Finished</td>
</tr>
<tr>
<td>15</td>
<td>53</td>
<td>56014</td>
<td>TLSv1.2</td>
<td>282</td>
<td>New Session Ticket, Change Cipher Spec, Finished</td>
</tr>
<tr>
<td>17</td>
<td>56014</td>
<td>53</td>
<td>TLSv1.2</td>
<td>116</td>
<td>Application Data</td>
</tr>
<tr>
<td>19</td>
<td>53</td>
<td>56014</td>
<td>TLSv1.2</td>
<td>87</td>
<td>Application Data</td>
</tr>
<tr>
<td>21</td>
<td>53</td>
<td>56014</td>
<td>TLSv1.2</td>
<td>178</td>
<td>Application Data</td>
</tr>
<tr>
<td>23</td>
<td>56014</td>
<td>53</td>
<td>TLSv1.2</td>
<td>116</td>
<td>Application Data</td>
</tr>
</tbody>
</table>

Secure sockets Layer

TLSv1.2 Record Layer: Application Data Protocol: http

Content Type: Application Data (23)
Version: TLS 1.2 (0x0303)
Length: 55

Encrypted Application Data: 4318fb5af26bf806363f1dca5583503851edafec23

Frame (116 bytes) Decrypted SSL data (31 bytes)

Payload is encrypted application data (ssl.app_data), 55 bytes
Packets: 59 · Displayed: 28 (47.5... · Profile: Default
T-DNS - DNS query

- Encrypted DNS query
  - Wireshark can decrypt if given the key

Decompression and SSL compression:

Secure sockets layer

- TLV1.2 Record Layer: Application Data Protocol: http
  - Content Type: Application Data (23)
  - Version: TLS 1.2 (0x0303)
  - Length: 55

Encrypted Application Data: 4318fb5af26bfb06363f11dca5583503851edafec234c189

Frame (116 bytes) Decrypted SSL data (31 bytes)
T-DNS - Fallback to TCP

<table>
<thead>
<tr>
<th>No.</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>56020</td>
<td>53</td>
<td>TCP</td>
<td>68</td>
<td>56020-53 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS=32 TVal=908901711 TSecr=0</td>
</tr>
<tr>
<td>2</td>
<td>53</td>
<td>56020</td>
<td>TCP</td>
<td>68</td>
<td>53-56020 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16344 WS=32 TVal=908901711</td>
</tr>
<tr>
<td>3</td>
<td>56020</td>
<td>53</td>
<td>TCP</td>
<td>56</td>
<td>56020-53 [ACK] Seq=1 Ack=1 Win=408288 Len=0 TVal=908901711 TSecr=908901711</td>
</tr>
<tr>
<td>4</td>
<td>56020</td>
<td>53</td>
<td>TCP</td>
<td>95</td>
<td>Standard query 0x7e2b _TXT STARTTLS</td>
</tr>
<tr>
<td>7</td>
<td>53</td>
<td>56020</td>
<td>DNS</td>
<td>114</td>
<td>Standard query response 0x7e2b _TXT</td>
</tr>
<tr>
<td>9</td>
<td>56020</td>
<td>53</td>
<td>DNS</td>
<td>87</td>
<td>Standard query 0x8141 A example.com</td>
</tr>
<tr>
<td>11</td>
<td>53</td>
<td>56020</td>
<td>DNS</td>
<td>151</td>
<td>Standard query response 0x8141 A 93.184.216.119</td>
</tr>
<tr>
<td>13</td>
<td>56020</td>
<td>53</td>
<td>DNS</td>
<td>87</td>
<td>Standard query 0x9a5b A sinodun.com</td>
</tr>
<tr>
<td>15</td>
<td>53</td>
<td>56020</td>
<td>DNS</td>
<td>148</td>
<td>Standard query response 0x9a5b A 88.98.24.67</td>
</tr>
<tr>
<td>17</td>
<td>56020</td>
<td>53</td>
<td>DNS</td>
<td>91</td>
<td>Standard query 0xc980 A <a href="http://www.example.com">www.example.com</a></td>
</tr>
<tr>
<td>19</td>
<td>53</td>
<td>56020</td>
<td>DNS</td>
<td>155</td>
<td>Standard query response 0xc980 A 93.184.216.119</td>
</tr>
<tr>
<td>21</td>
<td>56020</td>
<td>53</td>
<td>TCP</td>
<td>56</td>
<td>56020-53 [FIN, ACK] Seq=137 Ack=345 Win=4070 Len=0 TVal=908903455 TSecr=908903455</td>
</tr>
</tbody>
</table>

STARTTLS: type TXT, class CH
- Name: STARTTLS
- Type: TXT (Text strings) (16)
- Class: CH (0x0003)
- Time to live: 0
- Data length: 7
- TXT Length: 6
- TXT: NO_TLS

NO_TLS response - fall back to TCP (on same connection)
Supplemental Slides

- stresses on UDP-only DNS
- secure DNS relationship to TLS
- details about performance optimizations
- recursive-to-authoritative performance
- getdns
UDP Packet Size Limits

- for >25 years, *policy* decisions forced by UDP packet sizes
  - number of root servers: all fit in 512B
  - DNSsec: required EDNS
  - crypto algorithm and key size
- partial fix: EDNS0 deployment (10+ years, since 1999)
- but packet size lurks
  - keysizes
  - IPv6 records
  - certs in DNS (for DANE)

 dns over tcp and TLS
Doesn’t DNSsec already “Secure DNS”?

A: yes, but…

• DNSsec is about *query integrity*
  – that is: if you are told X, is X true?
  – it signs answers; signatures prove X is true

• DNSsec does *nothing* for privacy and DoS
  – *everything* sent in the clear: *no privacy*
  – nothing about DoS
  – large signatures stress UDP size limits

=> need DNSsec’s integrity *and* T-DNS’ privacy
Latency in DNS/TLS

C & S: open TCP connection

C: QNAME="STARTTLS", QCLASS=CH, QTYPE=TXT with the new TO bit set in EDNS options

   S: RCODE=0, TXT="STARTTLS" with the TO bit set

C & S: <negotiate a TLS session with a new session key, in binary>

TLS handshake: +2 RTTs

C: <send actual query>

   S: <reply to actual query>

query: 1 RTT

TCP 3wh: +1 RTT

STARTTLS: +1 RTT

DNS over TCP and TLS
Connection Reuse

• basic idea:
  reuse connection -> no setup cost

• secondary idea:
  if must close, client keeps state to restart quickly
Connection Reuse

• basic idea: reuse connection -> no setup cost
  – persistent connections (in client and server)

• secondary idea: if must close, client keeps state to restart quickly
  – TCP fast open: client has cookie to send data in 3wh
    • draft-ietf-tcpm-fastopen-08: in Linux-3.6, default 3.13
  – TLS resumption (RFC-5077): client keeps
    • RFC-5077: in OpenSSL and GnuTLS
Query Pipelining

send several queries immediately (not stop-and-wait)

before pipelining

with pipelining

pipelining matters:
62% of web has 4+ domain names
(dataset: common crawl)
(Digression) DNS Resolution: stub -> recursive -> authoritative

**stub**
- at end-user
- generates queries, recursive does work

**recursive**
- at user’s ISP or public DNS in a CDN
- converts user query to many authoritatives; caches replies

**authoritative**
- at provider (maybe replicated)
- has actual answers

Q: A **www.example.com**? -> rec

Q: A **www.example.com**? -> .
Q: A **www.example.com**? -> .com
Q: A **www.example.com**?
- -> example.com
A: 192.0.52.1

A: see NS for .com
A: see NS for example.com
A: 192.0.52.1
Latency: Recursive to Authoritative

TCP and TLS vs. UDP? effects of implementation choices? 

*with long RTT (=35ms)*

**method**: live experiments of random 140 names, each repeated 10x; recursive-authoritative RTT=35ms

New connections are expensive (RTTs exactly as predicted!)
Latency: Recursive to Authoritative

TCP and TLS vs. UDP? effects of implementation choices?

*with long RTT (=35ms)*

**method**: live experiments of random 140 names, each repeated 10x; recursive-authoritative RTT=35ms

(new connections are expensive (RTTs exactly as predicted!))

(reusing connections avoids much overhead)

(DNS over TCP and TLS)
getdns

• getdns API ([http://getdnsapi.net/](http://getdnsapi.net/))
  – modern, asynchronous DNS API specification
  – API originally by Paul Hoffman
  – created by and for application developers

• getdns is the first (and currently only) implementation of this specification

• open source C implementation developed and maintained in collaboration by NLnet Labs, Verisign Labs, and No Mountain Software

DNS over TCP and TLS