DNS Encryption: Operational Experience and Insights
OARC 32 San Francisco Feb 8/9 2020

Ralf Weber - Principal Architect
Mark Dokter - Product Manager
Bruce Van Nice - Product Marketing
Agenda / Topics

- Common question from Network Operators
- Observations
- Analysis
- Testing
Network Operator Question:
“How does DoT/DoH change our DNS infrastructure capacity model?”

.. well.. that depends on what the clients are doing..

Available sample data sources

- Firefox DoH: Multiple clients/roaming locations
- Residential CPE’s with DoT forwarding capabilities
  - pfSense on ALIX apu4b4 (~35 devices)
  - Stubby on a Raspberry PI (~15 devices)
  - AVM Fritz! Box 7590 (~10 devices)
Client 1: Firefox DoH

- **Session reuse:**
  - Total queries 268485
  - Total connections 62177
  - Avg queries per unique connection 4.32

- **Observations:**
  - Generally well behaved
  - Pro: Server side snafu (expired cert) did not cause any interruption of user experience
  - Con: .. cert validation failure not noticeable to end user
Client 2: Residential CPE (pfSense) DoT

- **Session reuse:**
  - None - new TLS session per query
  - Min 0.29 / Max 1.22 second conversation duration seen (on server side packet captures)

- **Observations: Some unexpected results**
  - ~2% of all queries (70851 of 3472626) timed out
  - Timeouts represented “mostly” unnoticeable user impact
  - Specifying different/multiple upstream forwarders and upgrading to latest in repo version of resolver did not fix issue
  - Typically between 30-120 TCP states in TIME_WAIT (on CPE)
Client 3: Residential CPE (Stubby) DoT

- Session reuse:
  - Total queries 182772
  - Total connections 50594
  - Avg queries per unique connection 3.61

- Observations
  - No issues
  - Not necessarily representative of average user setup
  - By default, stricter default configuration than other clients (fail closed)

```bash
root@server:~# cat /etc/stubby/stubby.yml
resolution_type: GETDNS_RESOLUTION_STUB
dns_transport_list:
  - GETDNS_TRANSPORT_TLS
tls_authentication: GETDNS_AUTHENTICATION_REQUIRED
#tls_authentication: GETDNS_AUTHENTICATION_NONE
tls_query_padding_blocksize: 128
edns_client_subnet_private : 1
idle_timeout: 10000
listen_addresses:
  - 127.0.0.1
  - 0::1
round_robin_upstreams: 1
upstream_recursive_servers:
  - address_data: 18.189.255.38
tls_auth_name: "dns-live-demo.nominum.cloud"
```

root@server:~#
Client 4: Residential CPE (AVM Fritz!Box) DoT

DNS over TLS (DoT)

- Encrypted name resolution in the internet (DNS over TLS)
- Force a certificate check for encrypted name resolution in the internet
  - Only allow servers that are fully validated.
  - This setting should be disabled only if the identity of the server is known. Otherwise MITM attacks cannot be prevented.
- Allow fallback to non-encrypted name resolution in the internet.
  - Allow a fallback to non-encrypted DNS traffic if all encrypted servers fail.
  - Attention: If this setting is disabled, a complete DNS failure can result.

- Session reuse:
  - Total queries 286811
  - Total connections 53500
  - Avg queries per unique connection 5.36

- Observations
  - No issues
  - Fallback options clearly exposed on UI
# Client Behaviours - Cert validation failures

## New operational workflows to consider
- Certs: Obtain, rotate/automate, oops?
- Troubleshooting this is hard

<table>
<thead>
<tr>
<th>Client</th>
<th>Version tested</th>
<th>Failure Mode (default)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox (DoH)</td>
<td>72.0.2</td>
<td>Fail Open</td>
</tr>
<tr>
<td>Chrome (DoH)</td>
<td>X</td>
<td>Fail Open</td>
</tr>
</tbody>
</table>
| kdig / curl / getdns    | 2.6.5 / 0.1 / 1.6.0 | Opportunistic mode: Fail Open  
Strict mode: Fail Closed |
| Android 9 Private DNS   | G950WVLS7CSK1  | Automatic mode: Fail Open  
Private DNS hostname: Fail Closed |
| CPE (pfsense)           | 2.4.4          | Fail closed until CPE restart              |
| CPE (AVM Fritzbox)      | 7.19-74093     | Fail open*                                 |
| Stubby on Raspberry PI  | 0.2.2          | Fail closed  
(Dependent on “tls_authentication” param) |
### Other observations - Server idle timeouts

<table>
<thead>
<tr>
<th>Provider</th>
<th>DoT idle timeout (seconds)</th>
<th>DoH idle timeout (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloudFlare</td>
<td>1.1</td>
<td>15</td>
</tr>
<tr>
<td>Google</td>
<td>60</td>
<td>240</td>
</tr>
<tr>
<td>Quad9</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>NextDNS</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>British Telecom</td>
<td>X</td>
<td>10</td>
</tr>
<tr>
<td>Comcast</td>
<td>10</td>
<td>618-655</td>
</tr>
<tr>
<td>Cox</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Deutsche Telekom</td>
<td>600</td>
<td>10</td>
</tr>
</tbody>
</table>
DNS Log Analysis

- **Methodology**
  - Logged queries include timestamp and unique IP/port combination
  - Server timeout is 10 seconds
  - If there is more than 10 seconds (I actually used 60 to be save) between two queries with same source IP/Port it is counted as a new connection
  - As clients don’t seem to drop connections the actual connections length is 10 seconds after the last packet (not shown in graphs)

- **Graphs**
  - Log scale (to show the area where most data is)
  - Numbers show percentage of connections
Connection time distribution

Number of connections

Connection length (seconds)
Query count distribution
Off to the races

- DNS over 53 sizing is simple
  - Primarily about throughput
  - Understood values. Fixed ~0.25 qps per subscriber, Mobile ~0.15 qps (both rising)

- TCP/DoT/DoH has more variables
  - How many connections?
  - How long is connection lifetime?
  - How many queries per connection?
  - How CPU intensive is connection setup vs established state queries

- Have to do some assumptions here for a lab test
Test Setup

- **Hardware**
  - Real Hardware – 40 core Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.00GHz
  - 64GB of memory (not enough ;-)
  - Intel 10GB NIC
  - One similar machine as client for UDP testing
  - 12 machines I could borrow from QA (Thanks guys) as DoH clients

- **Tools**
  - Dnsperf for UDP
  - Python3 (to write my own test scripts)
DNS over UDP/53 tuning and testing

- On the server
  ```
sysctl -w net.core.rmem_max=524288
sysctl -w net.core.wmem_max=524288
/sbin/ethtool -N eth0 rx-flow-hash udp4 sdfn
  ```

- On the client run `dnsperf`
  ```
dnsperf -s dohtest -d one.q -l 5 -T 8 -c 8 -S 1 -q 200
DNS Performance Testing Tool
Nominum Version 2.1.1.0.d

[Status] Command line: dnsperf -s dohtest -d one.q -l 5 -T 8 -c 8 -S 1 -q 200
[Status] Sending queries (to dohtest) over UDP
[Status] Started at: Sat Feb 8 03:57:29 2020
[Status] Stopping after 5.000000 seconds
1581134250.831940: 604876.413248
1581134251.832960: 621347.225830
1581134252.833940: 650431.577054
1581134253.834939: 623151.471680
[Status] Testing complete (time limit)
```
DNS over HTTPS testing

- Python script
  - Use python multithreading to open a HTTPs connection and send a query every 4 seconds
    - 0.25 qps
    - 5 packets across a 20 second connection
  - Works reliably for ~1000 concurrent threads
  - Add more instances for more connections
  - Did not really work (at least not for the 5 to 6 digit numbers I was aiming for)
  - What is up
DNS over HTTPS (TCP really) tuning

- TCP connections
  - Each connection requires a source port
    - Per default there are only 30k allowed
    - 65536 / 64512 is the maximum you can have
    - For reliability only do 50k connections per machine
      - Need more client machines – QA to the rescue!
  - Each connection requires a filehandle
    - Couldn’t push above 1048576 in /etc/security/limits.conf
      - Have another root terminal open when you change this file
Detailed TCP tuning

- **Client**
  - sysctl net.ipv4.ip_local_port_range="1025 65535"

- **Server**
  - sysctl net.ipv4.tcp_fin_timeout = 20
  - sysctl net.ipv4.tcp_tw_reuse = 1
  - sysctl fs.file-max=4194304
  - /etc/security/limits.conf
    - soft nofile 1048576
    - hard nofile 1048576
Can we crash the test setup?

- Stuff to figure out
  - How fast can we create new connections
  - How many active connections can we serve
  - What will be the CPU load
  - When will it break
Connection stress diagram
Crashtest takeaways

- It didn’t crash
- Connection setup is far more CPU intensive than just answering
- It had limits
  - ~7000 new DoH connections per second
  - 460000 active connections on a 64GB machine
- It scaled well
  - Could push additional 100k UDP without any other impact to the above
Closing thoughts

- Well behaved clients critical to successfully scaling server side operations
- Early days. Ecosystem is still evolving
- DNS servers now also need to tune for TCP workloads
- Lots of active connections needs lots of memory
- Server side multithreading important
- UDP and TLS workloads are cumulative
- Ensure there are sufficient CPU cores to handle the combined workload