DNS-over-QUIC

More than a year with DoQ

Andrey Meshkov
CTO and Co-Founder of AdGuard
am@adguard.com
@ay_meshkov

Intro

DNS-based products by AdGuard

- AdGuard DNS — public DNS resolver
- AdGuard Home — DNS server for personal use with content blocking capabilities
- AdGuard apps provide DNS filtering and encryption capabilities (DoH/DoT/DNSCrypt)
- We added DoQ to each of them: https://adguard.com/en/blog/dns-over-quic.html

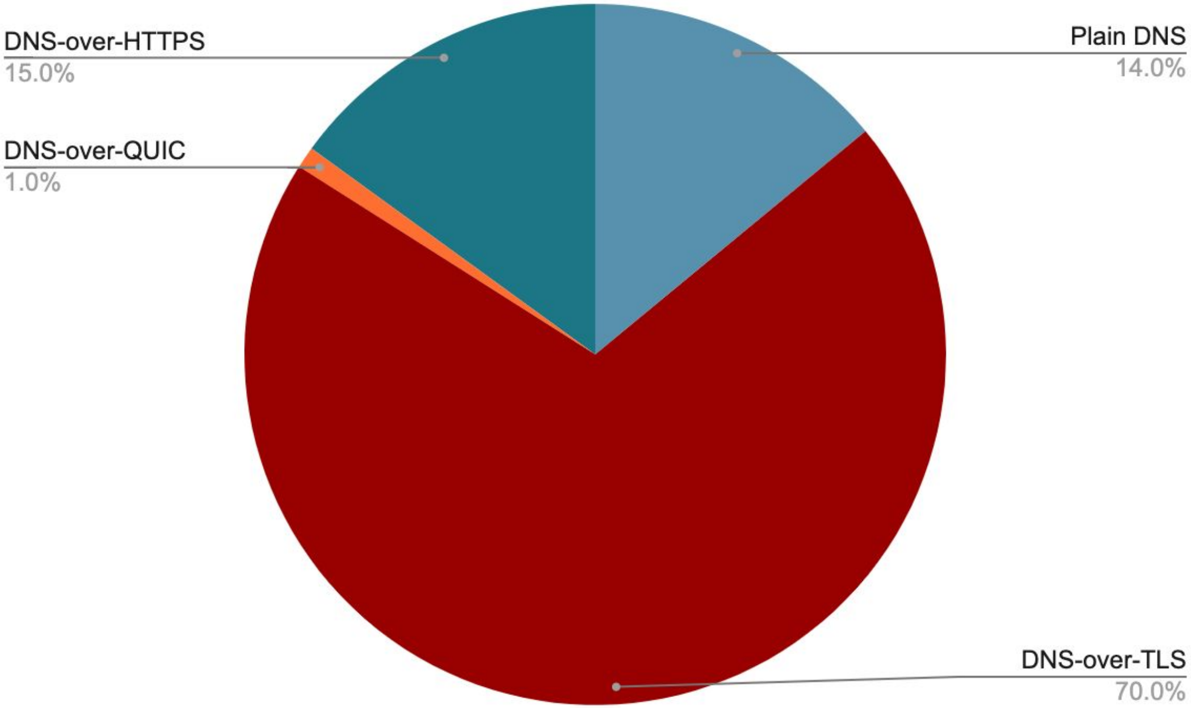
AdGuard DNS

- Public DNS resolver with the focus on content blocking
- The first beta was launched in the end of 2016
- Officially released in December, 2018
- Open-source
  https://github.com/AdguardTeam/AdGuardDNS
- Most of the clients are mobile devices

AdGuard DNS

Avg 1M+ RPS

- DNS: 14%
- DoT: 70%
- DoH: 15%
- DoQ: 1%

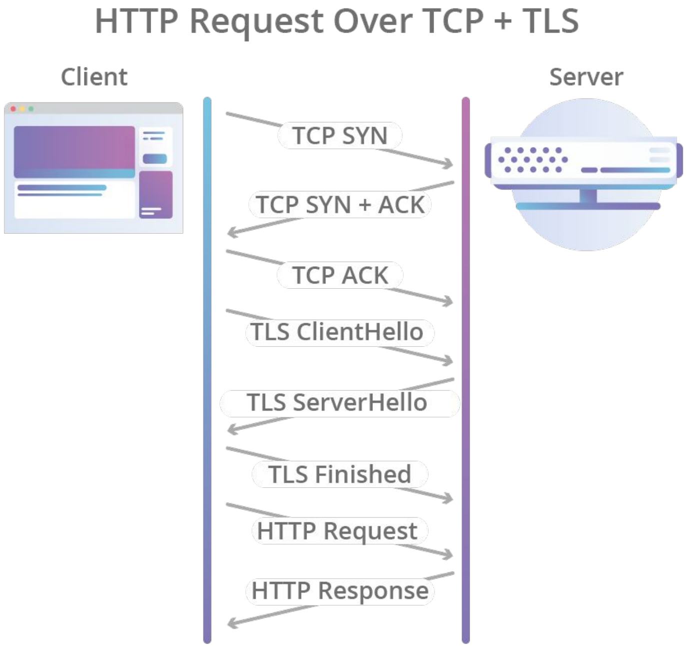


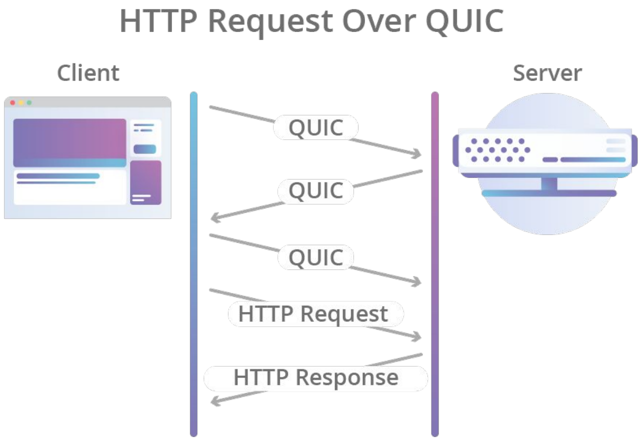
QUIC

What is QUIC? Basically, this is reinventing TCP over UDP, but with some cool stuff built-in.

- Built-in encryption (TLS v1.3)
- Faster handshake compared to TCP+TLS
- Multiplexing (+solving head-of-line blocking)
- Connection migration

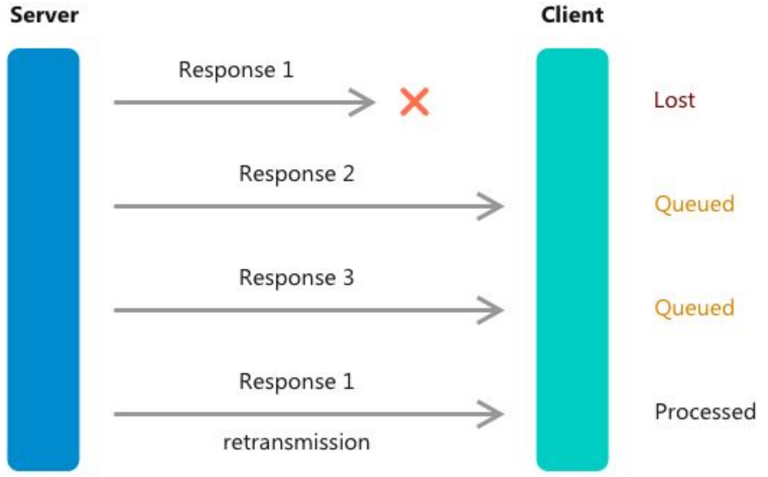
Faster Handshake



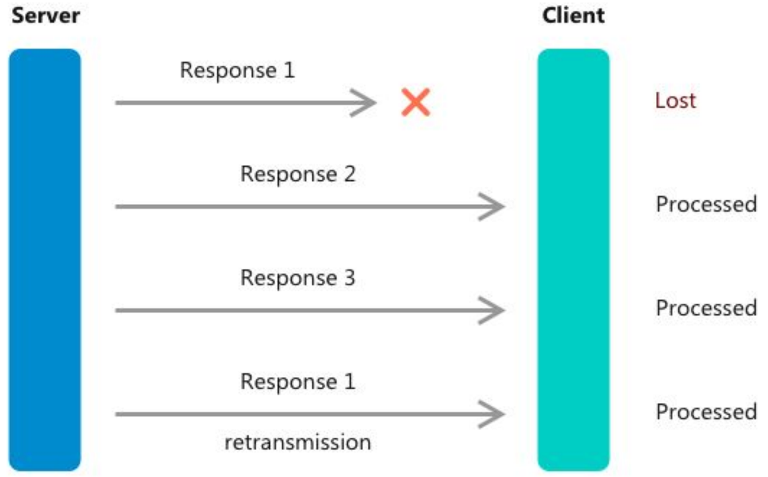


Images from https://blog.cloudflare.com/the-road-to-quic/

Head-Of-Line Blocking

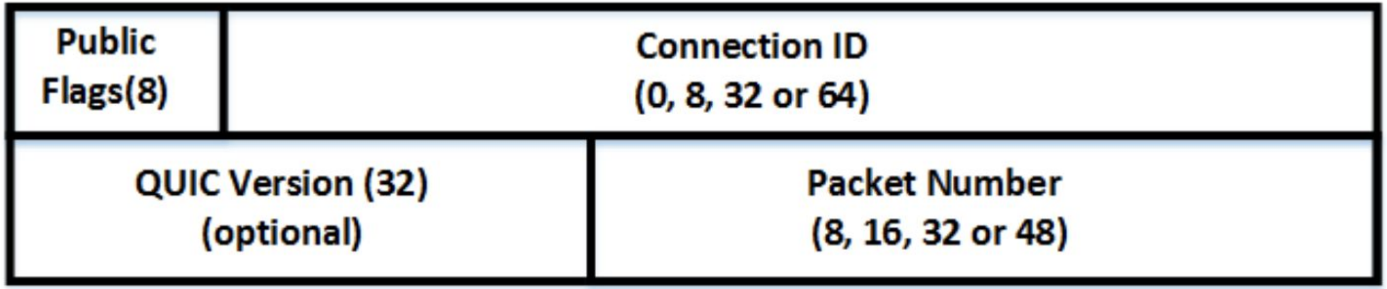


HTTP/2 head-of-line blocking: a single TCP packet loss will, all queries/responses have to wait



QUIC - every DNS query/response is a new QUIC stream

Connection Migration



Public Flags(8)	Connection ID (0, 8, 32 or 64)
QUIC Version (32) (optional)	Packet Number (8, 16, 32 or 48)

QUIC packet header

- Endpoints can use “Connection ID” to track connections
- This makes it possible to continue using the same connection when network change occur (i.e. Wi-Fi <-> Cellular)

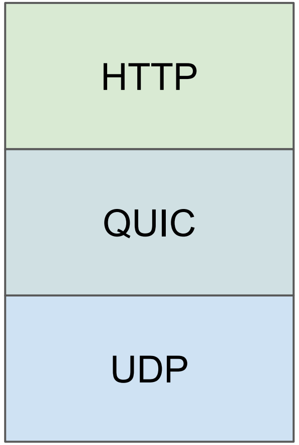
DoQ vs Plain DNS

- Encryption
- No limit on DNS messages size
- Built-in protection against amplification

DoQ vs DNS-over-HTTP/3

- Both DoQ and DoH3 use QUIC as an underlying transport
- HTTP/3 adds HTTP on top of it
- HTTP adds almost zero value
- It adds more data-points that can be used for fingerprinting clients

  Examples:
  - HTTP headers order
  - TLS properties
  - ETag tracking

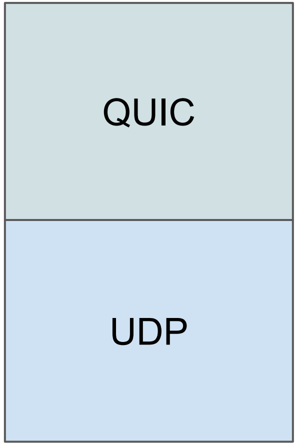


HTTP

QUIC

UDP

DoH3 stack



QUIC

UDP

DoQ stack

Our experience with DoQ

- DoQ connections are more “stable” than DoH/DoT
- DoQ is heavier on CPU than DoT, same as DoH
- DoQ is a good fit for mobile thanks to faster handshake

Performance

QUIC connections seem to be more “stable” than DoT and DoH.

Metric: *DNS queries / TLS handshakes*

- DoT: ~9 queries per connection
- DoH: ~14 queries per connection
- DoQ: ~30 queries per connection

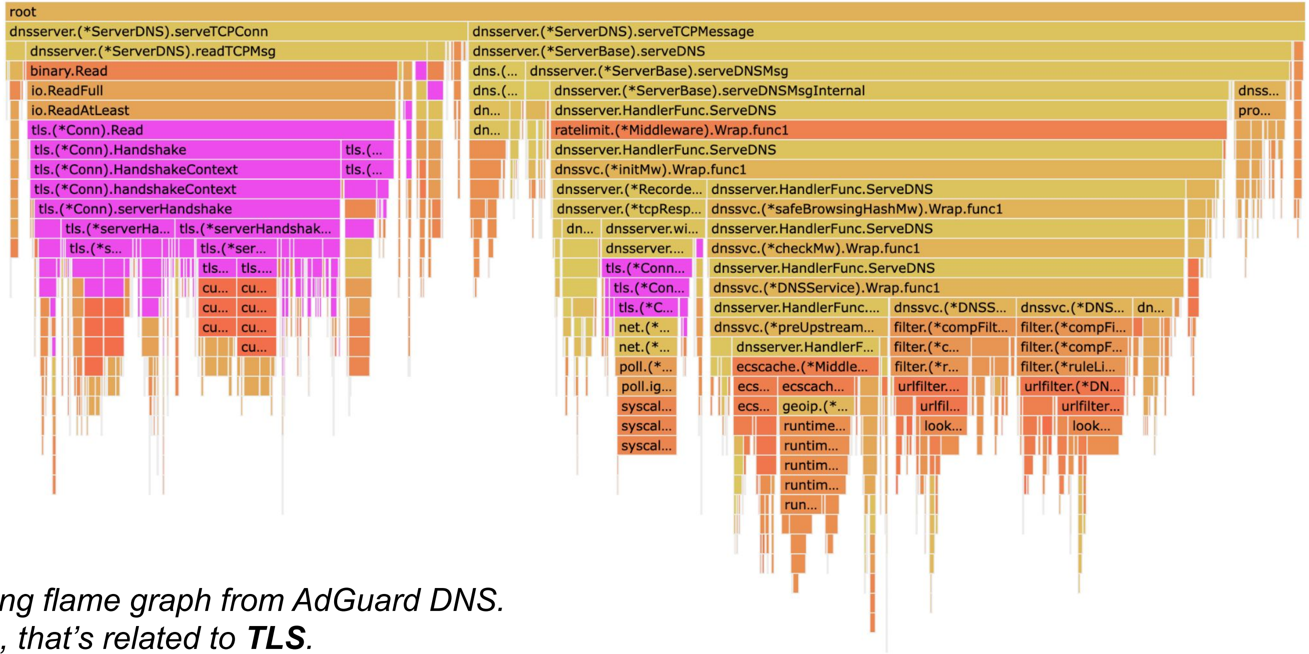
Handshake is the heaviest and slowest part so, generally, fewer handshakes means better performance.

CPU usage

Metric: *Time spent on AdGuard DNS filtering / Time spent in the protocol-specific code*

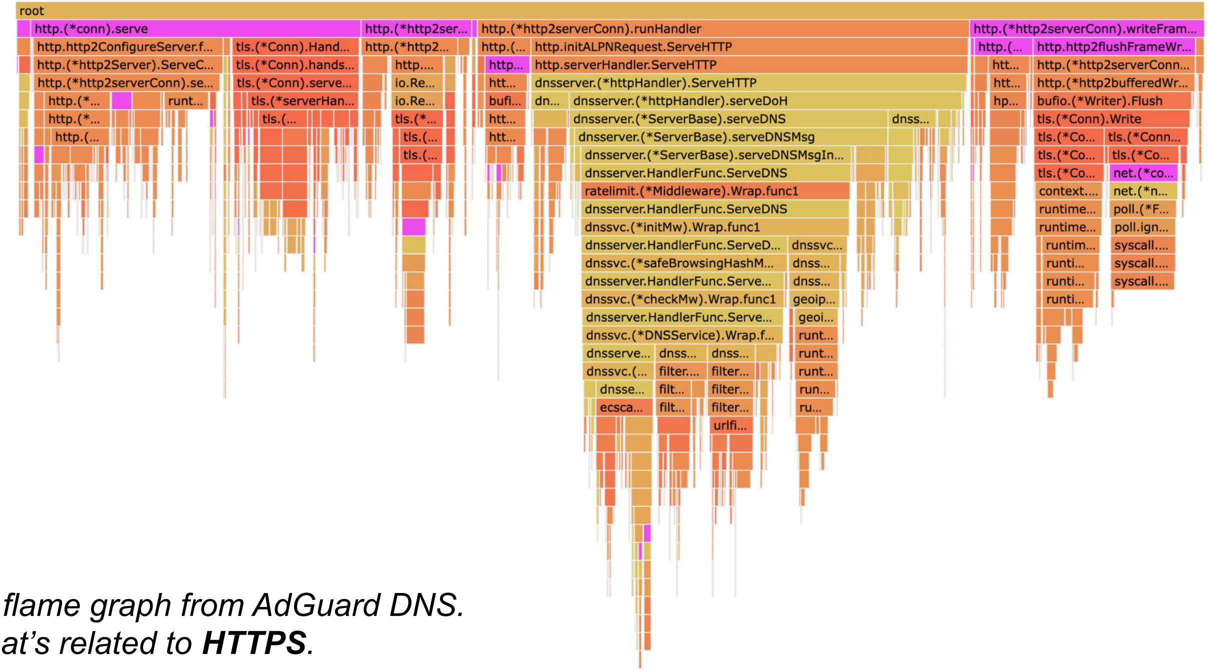
1. Processing of a single DNS query involves cryptoprotocol-related code AND internal logic of AdGuard DNS (working with DNS messages, DNS cache, content blocking, etc).
2. On a flame graph we can see how much time was spent in each part of the code.

CPU usage - DoT



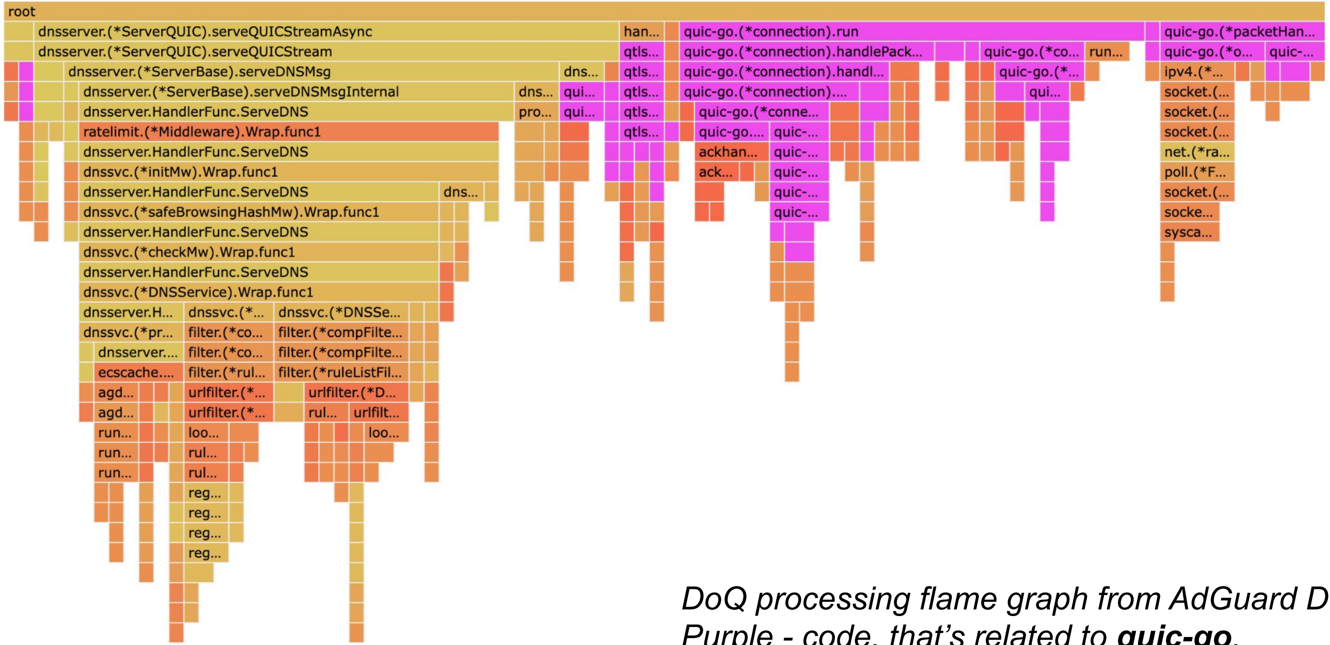
DoT processing flame graph from AdGuard DNS.
*Purple - code, that’s related to **TLS**.*

CPU usage - DoH



DoH processing flame graph from AdGuard DNS.
Purple - code, that's related to ***HTTPS****.*

CPU usage - DoQ



DoQ processing flame graph from AdGuard DNS.
Purple - code, that's related to ***quic-go****.*

CPU usage

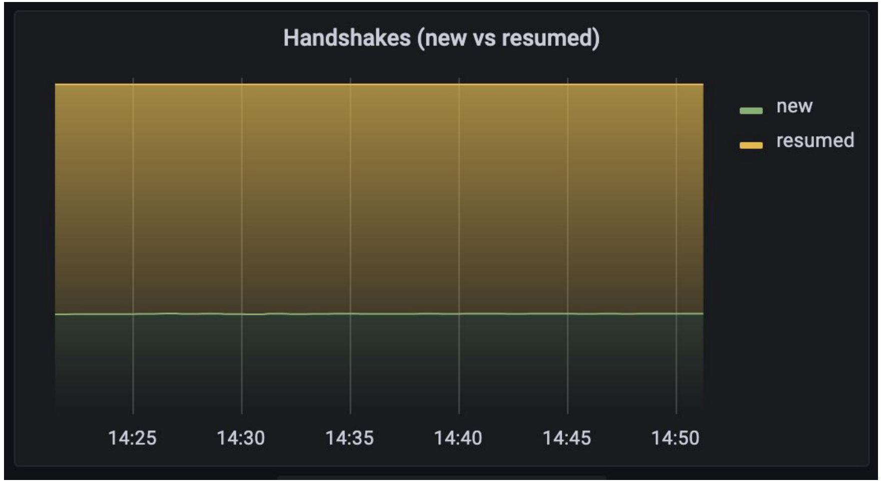
QUIC is heavier on CPU than DoT. Same as DoH.

Metric: *Time spent on AdGuard DNS filtering / Time spent in the protocol-specific code*

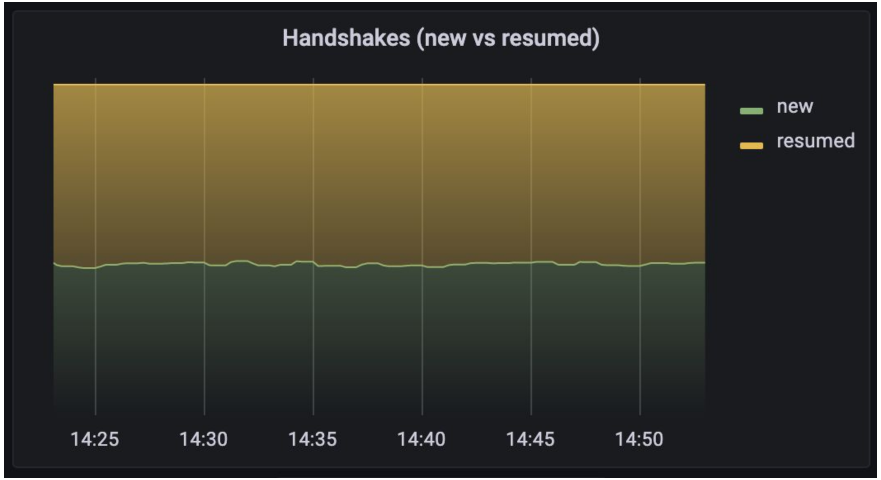
- DoT: ~**40%** of the time was spent in TLS-related code
- DoH: ~**60%** of the time was spent in HTTP-related code
- DoQ: ~**60%** of the time was spent in QUIC-related code

Note, that it **does not** mean with DoQ a single query is slower! It just requires more CPU time overall (on async operations), but processing of a single query is very fast.

TLS Session Resumptions



TLS session resumptions (DNS-over-TLS)

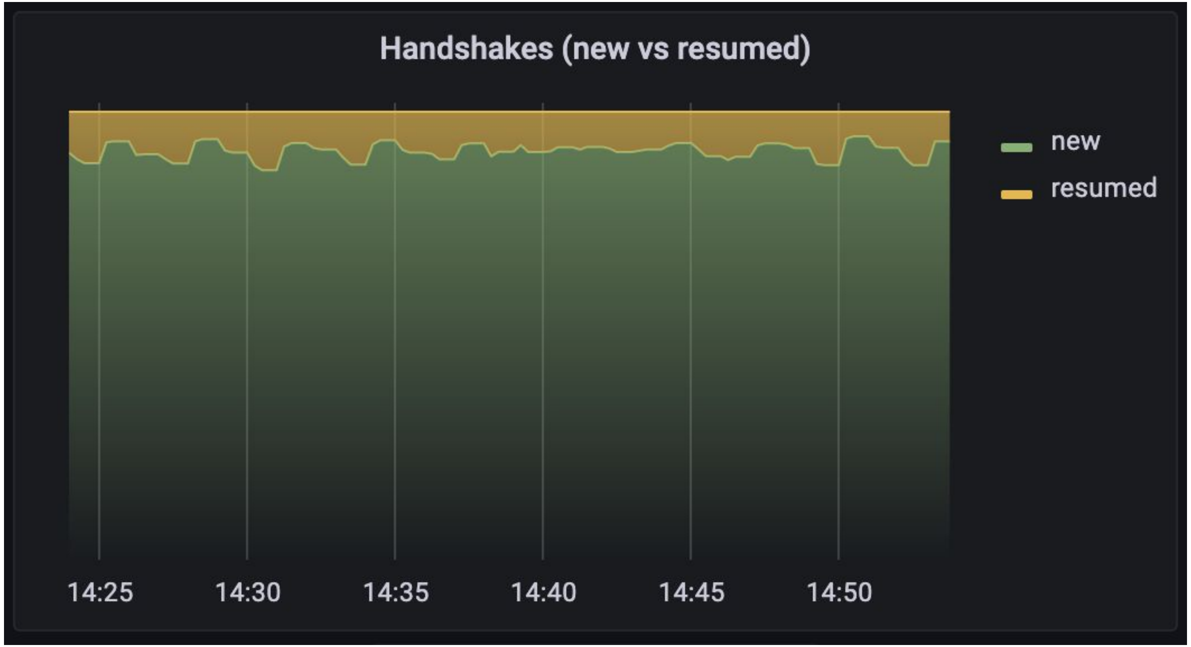


TLS session resumptions (DNS-over-HTTPS)

TLS Session Resumptions (DoQ)

Overall, the share of resumed sessions is very small for DoQ.

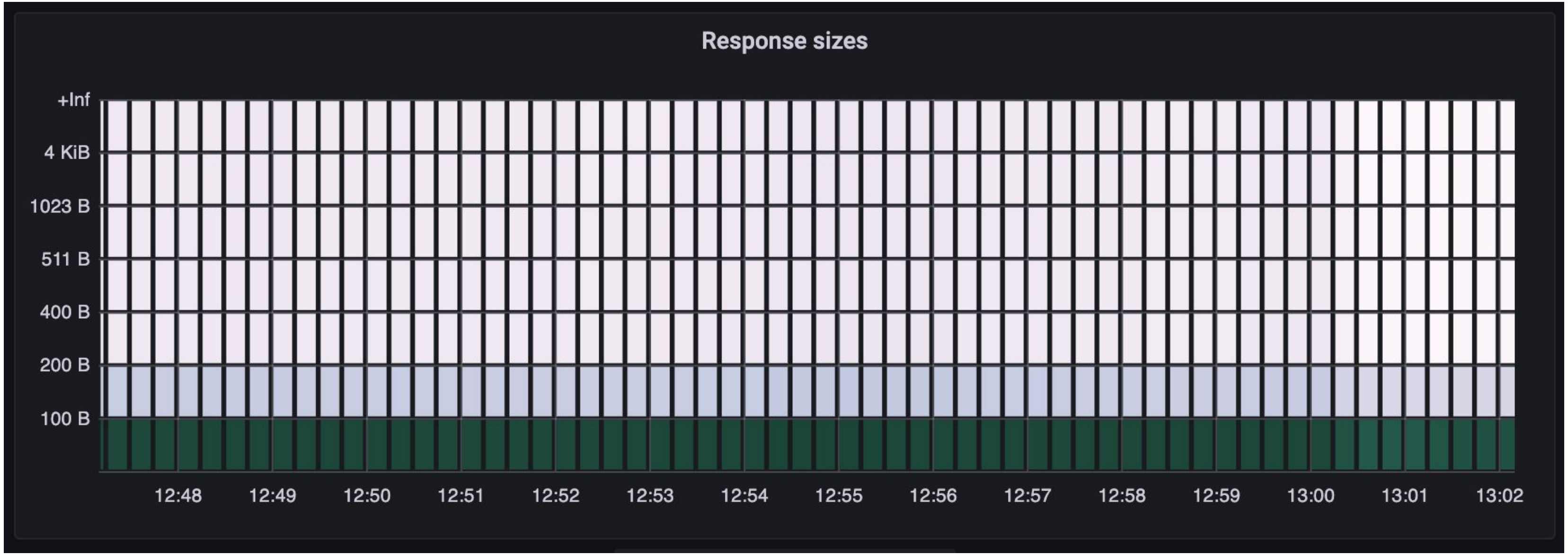
We are yet to figure out what’s the problem here.



Mildly interesting insights

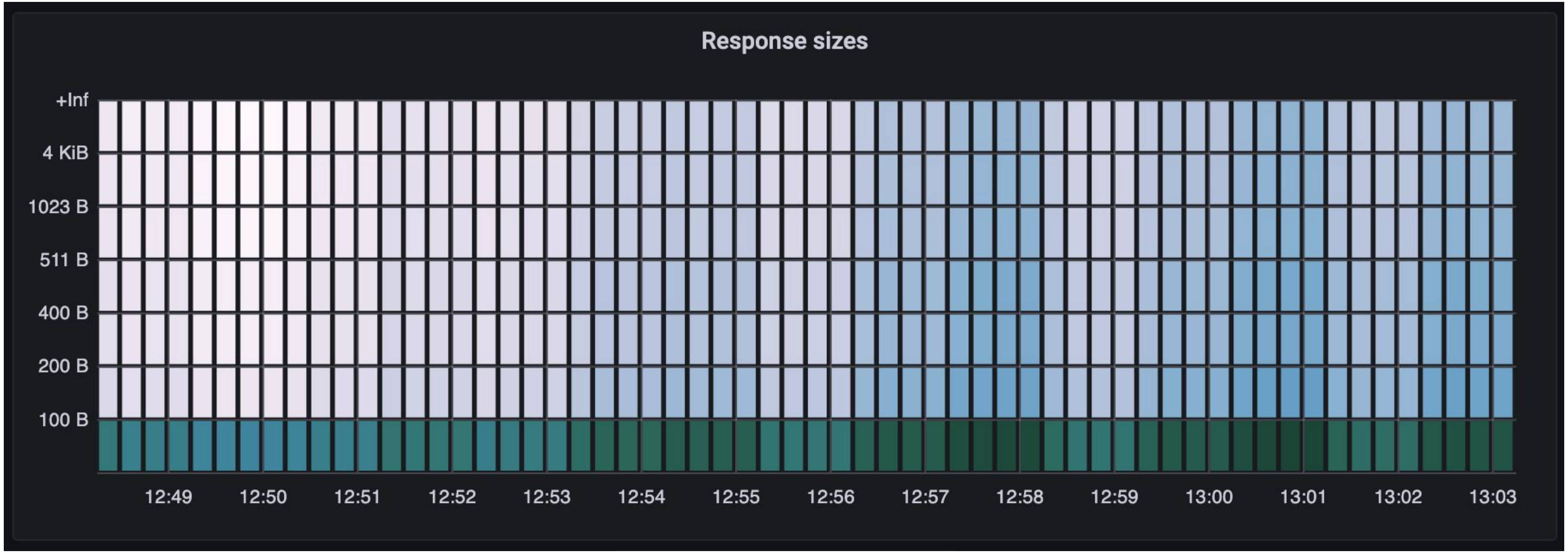
- Request sizes are pretty much the same for all protocols
- Response sizes distribution for DoQ is similar to DoH
- DoQ and DoH clients prefer IPv4 not as often as DoT clients
- Invalid DNS messages
- TLS versions

Response sizes



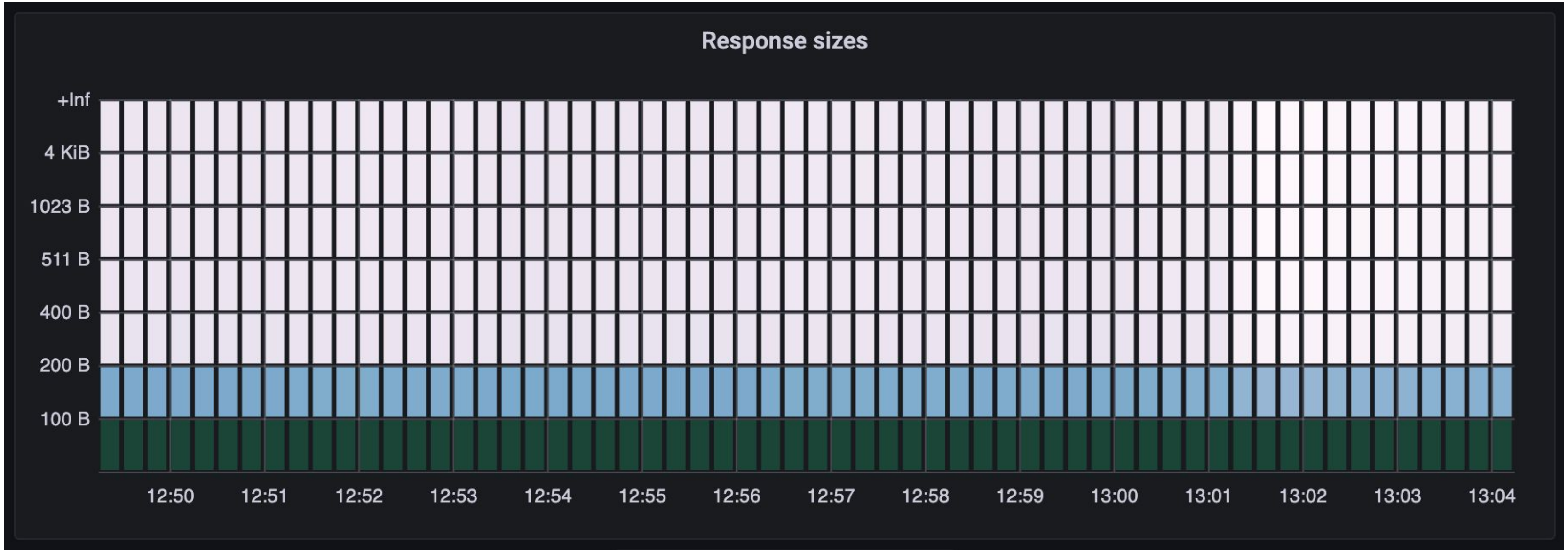
Plain DNS over UDP

Response sizes



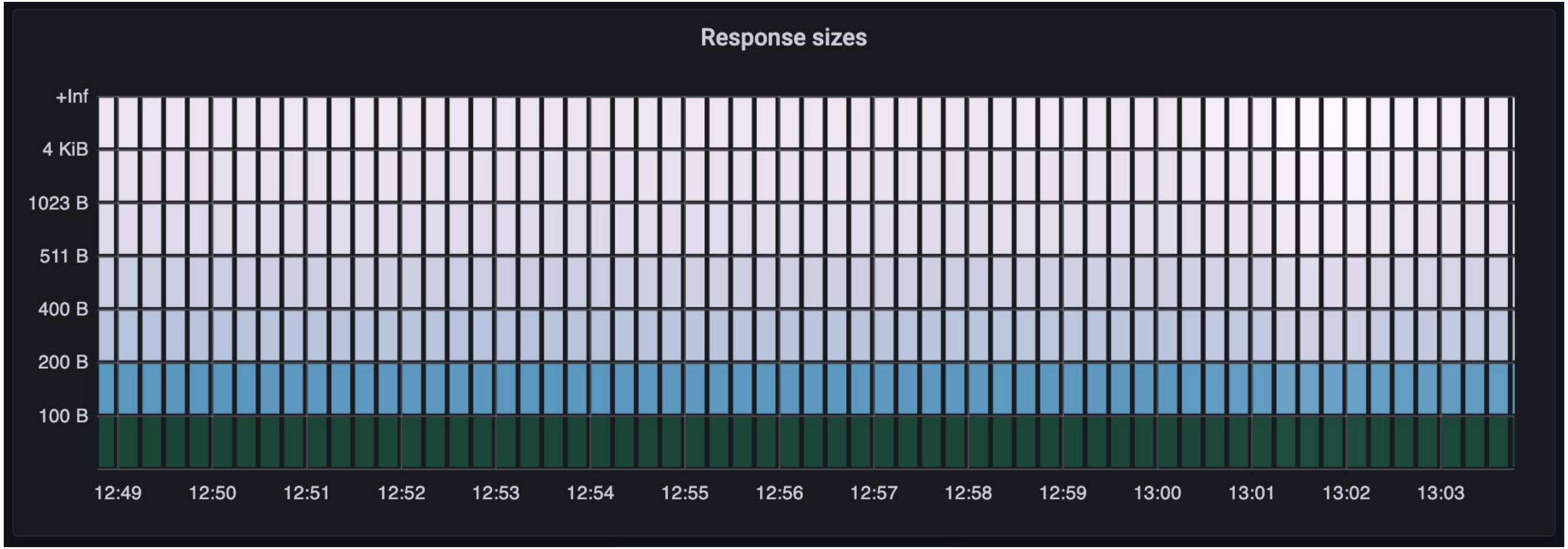
Plain DNS over TCP

Response sizes



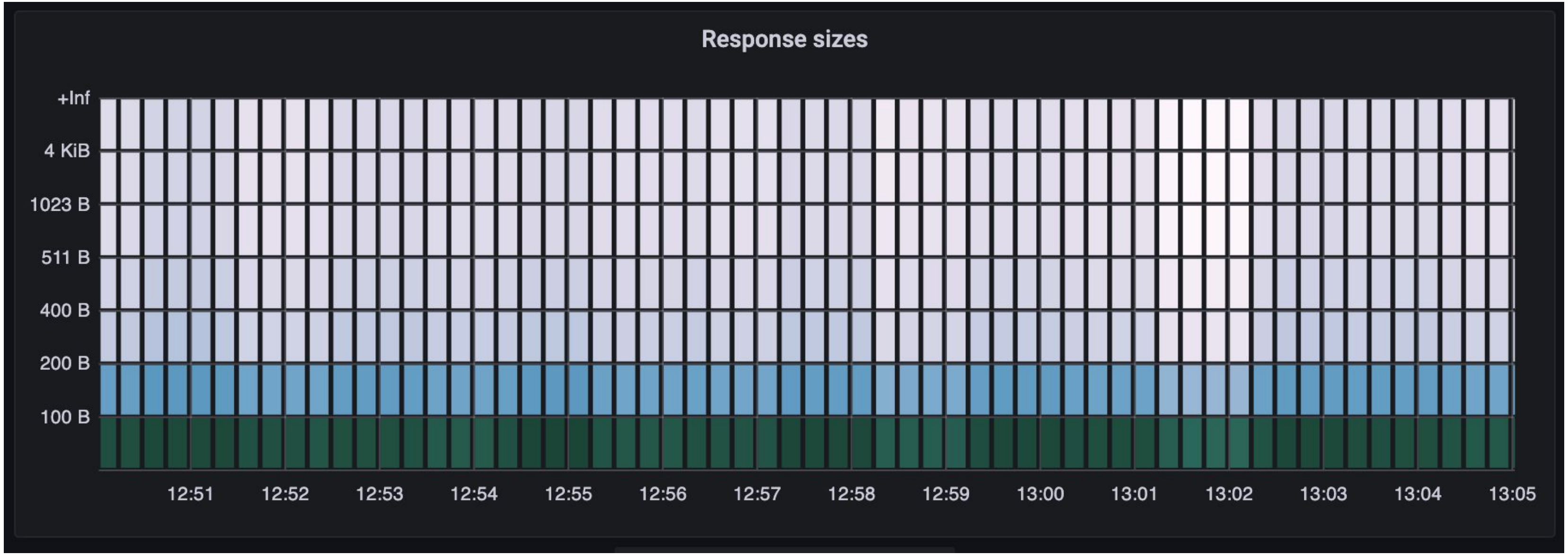
DNS-over-TLS

Response sizes



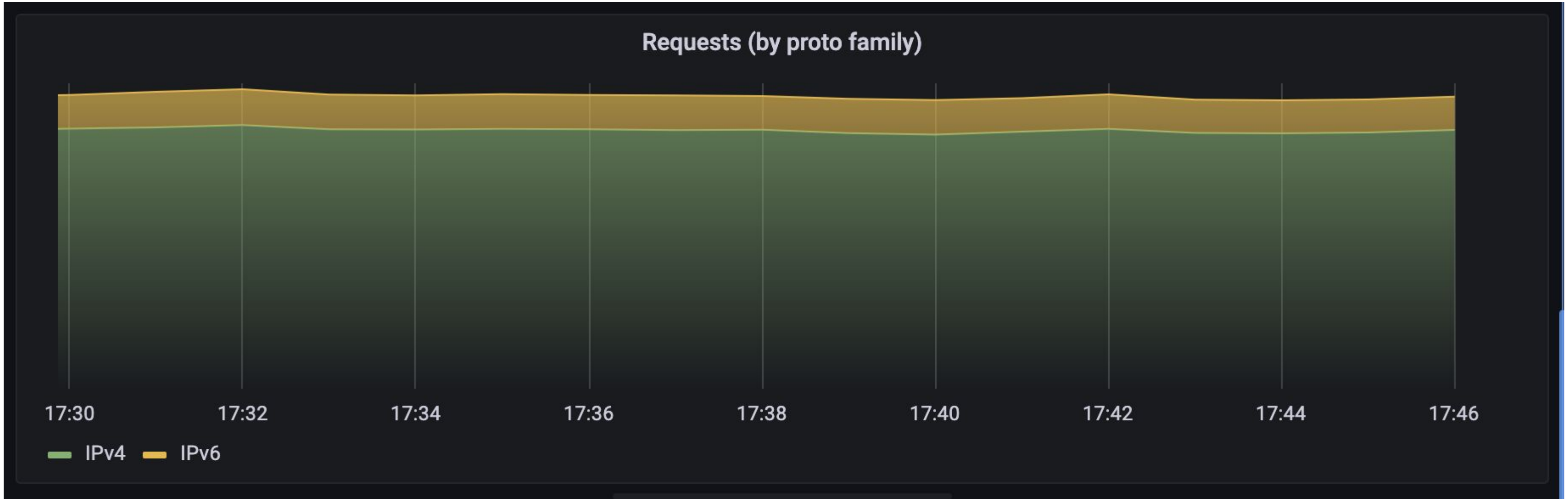
DNS-over-HTTPS

Response sizes



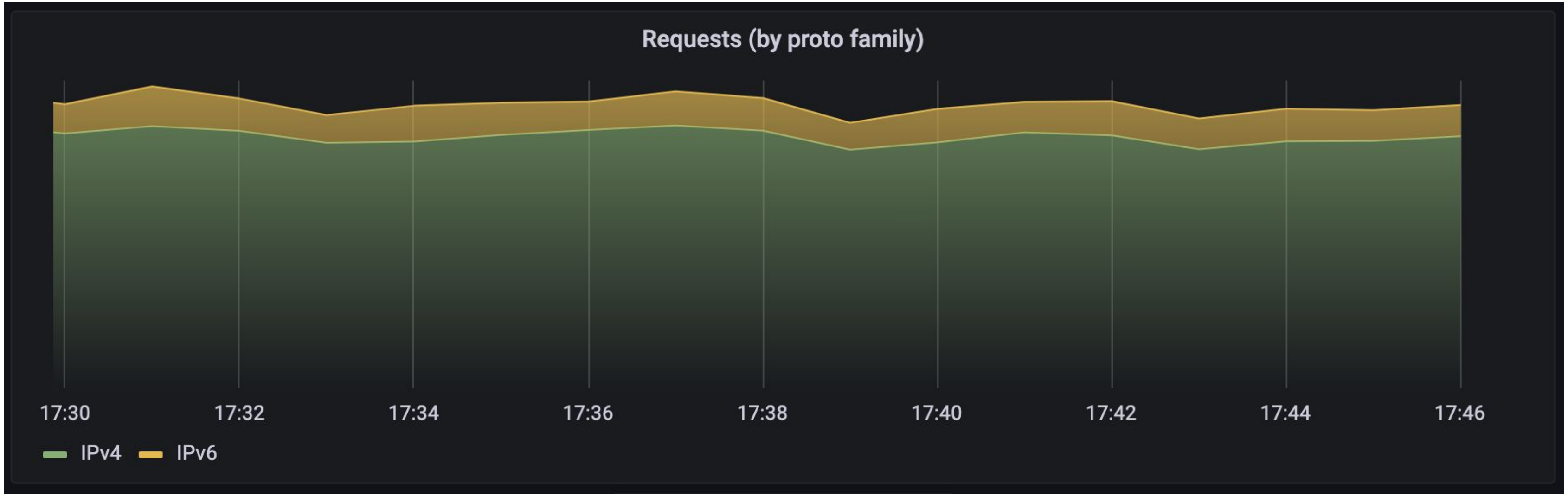
DNS-over-QUIC

IPv4 vs IPv6



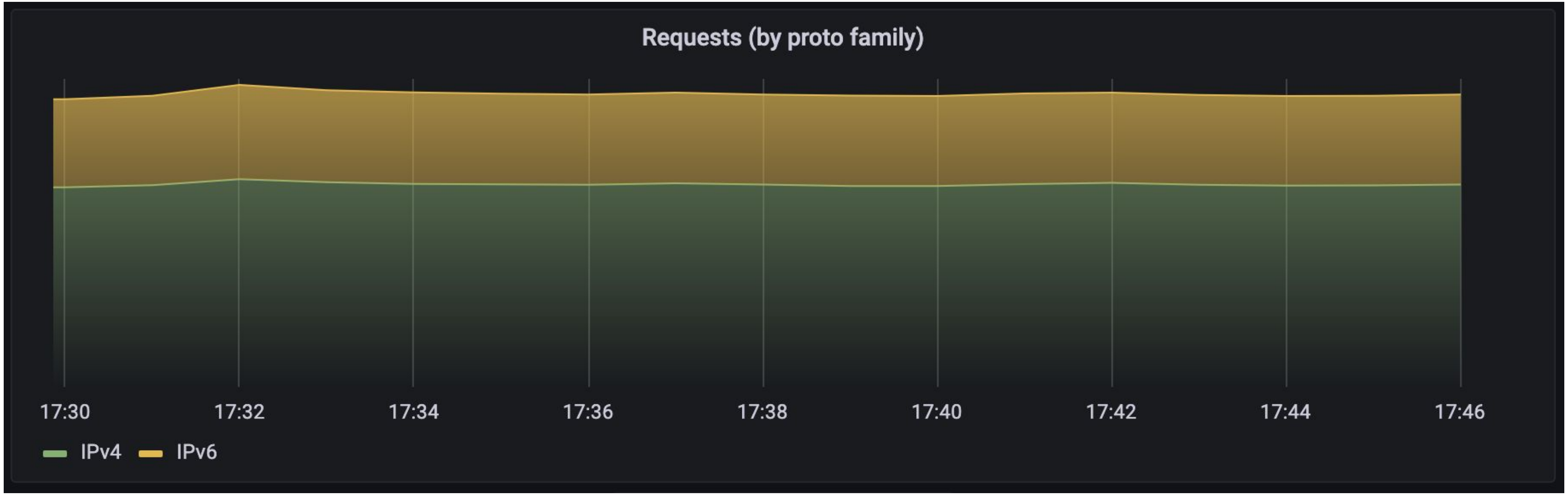
DNS-over-HTTPS

IPv4 vs IPv6



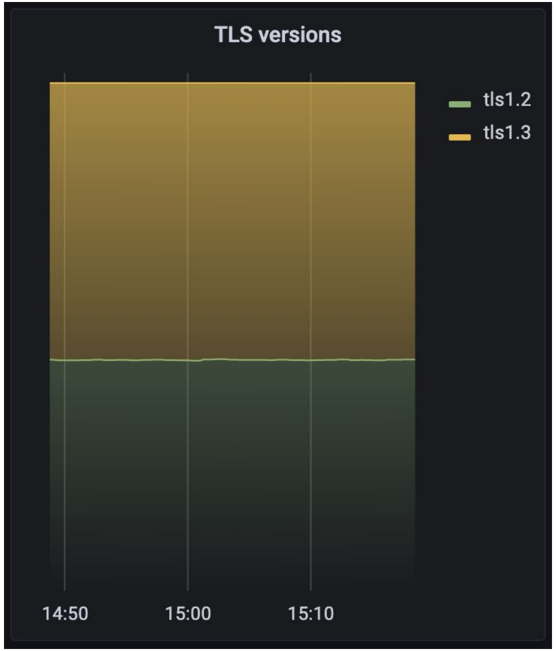
DNS-over-QUIC

IPv4 vs IPv6

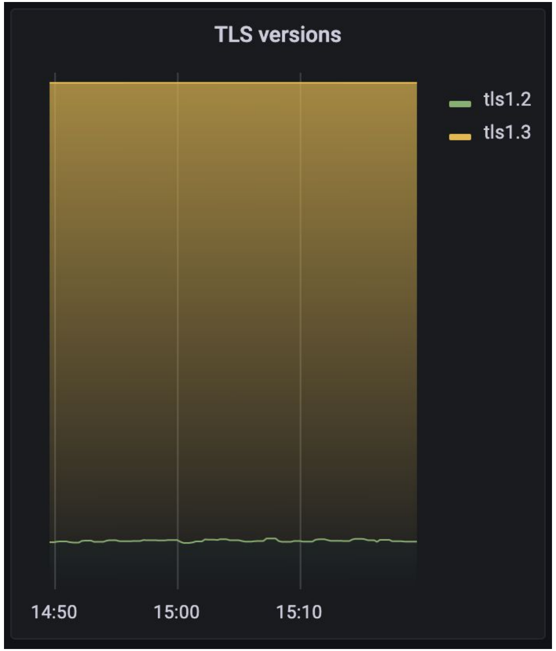


DNS-over-TLS

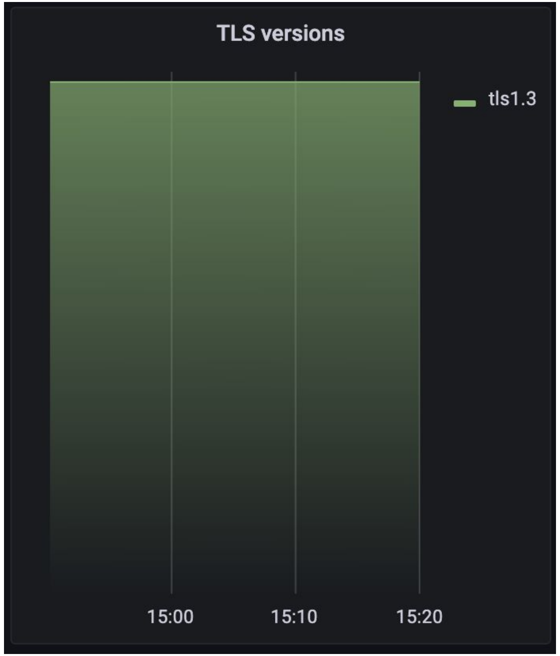
TLS versions



DNS-over-TLS

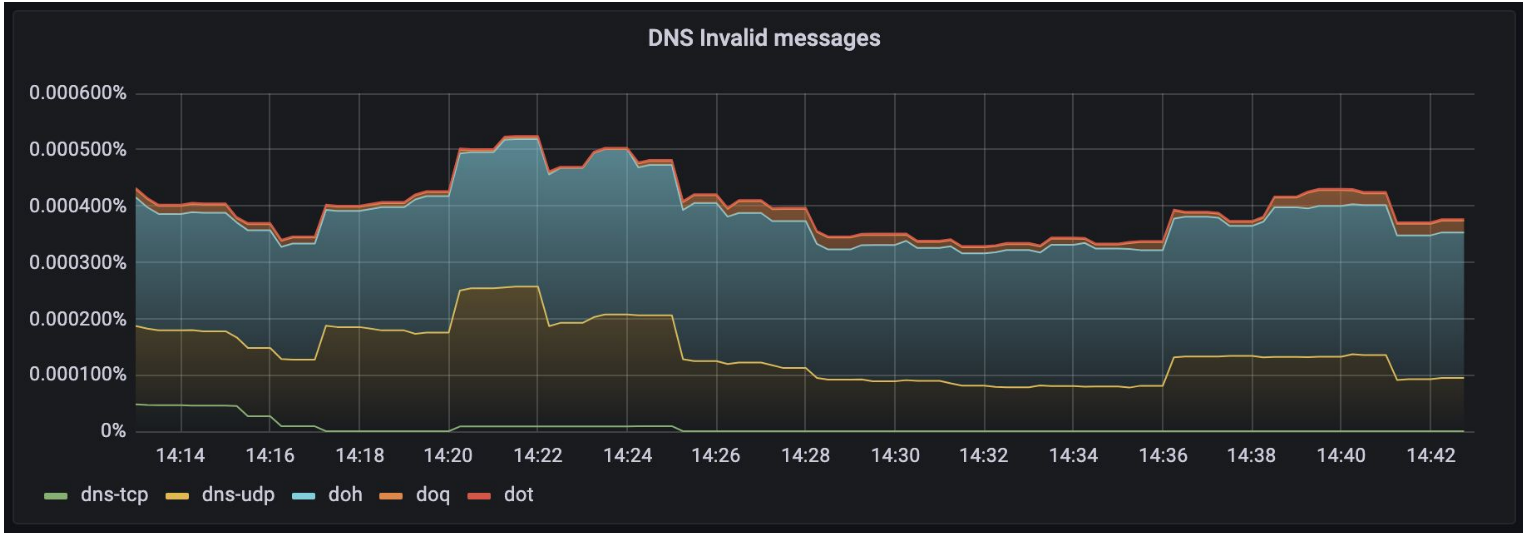


DNS-over-HTTPS



DNS-over-QUIC

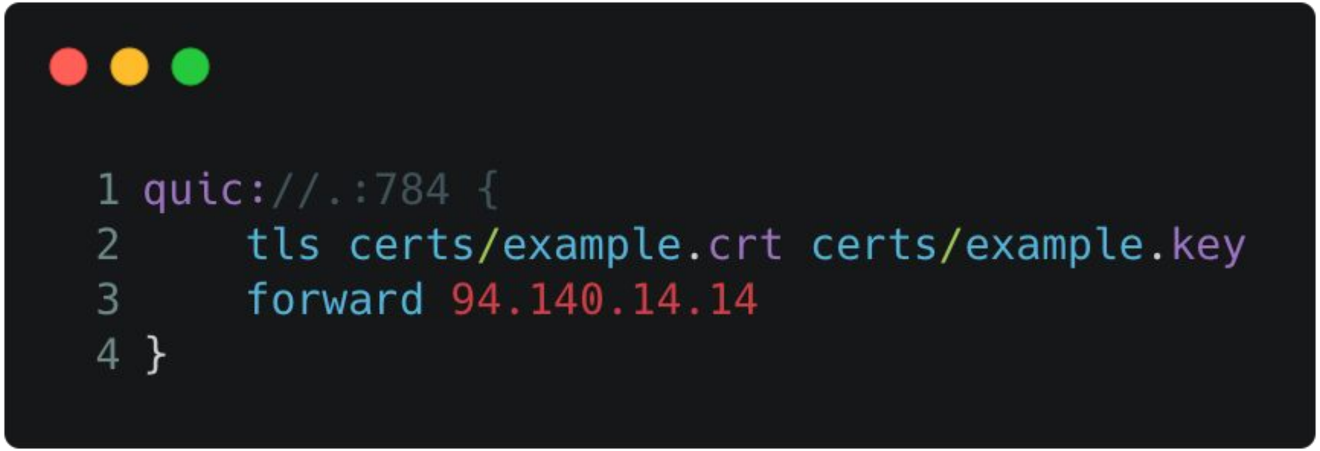
Invalid DNS queries



Queries, that we cannot parse

DoQ Server-Side Implementations

- CoreDNS fork **(deprecated, we don’t use it anymore)**: https://github.com/AdguardTeam/coredns



```
quic://.:784 {
    tls certs/example.crt certs/example.key
    forward 94.140.14.14
}
```

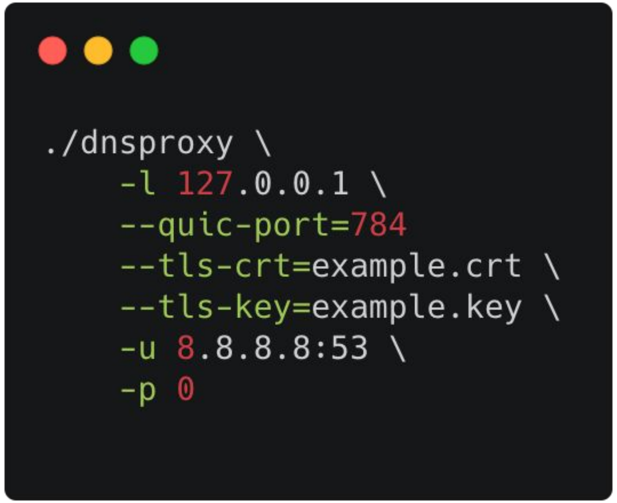
Sample CoreDNS configuration

DoQ Server-Side Implementations

- AdGuard DNS: **coming soon**
- We’re going to open the code under AGPL in the following weeks.
- The part of the code that implements pure DNS server (with DoQ support) will be then moved to a separate library with a permissive license.

DoQ Server-Side Implementations

- dnsproxy: https://github.com/AdguardTeam/dnsproxy

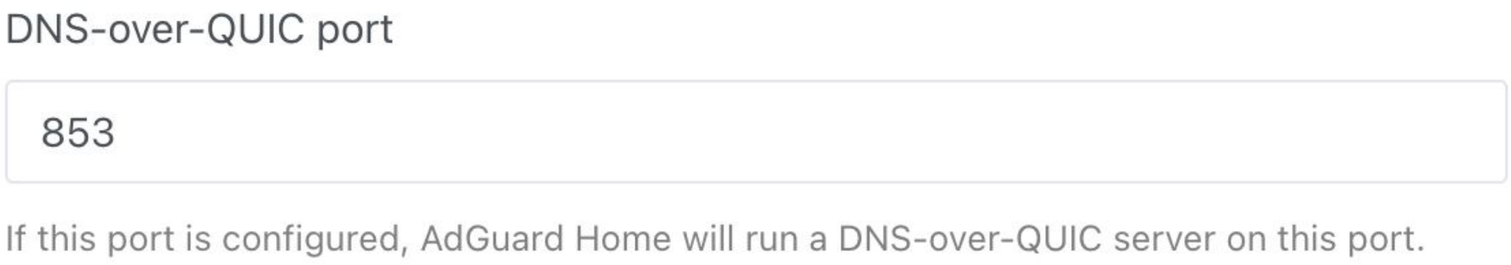


```
./dnsproxy \
    -l 127.0.0.1 \
    --quic-port=784
    --tls-crt=example.crt \
    --tls-key=example.key \
    -u 8.8.8.8:53 \
    -p 0
```

Running dnsproxy as a DoQ server forwarding queries to 8.8.8.8

DoQ Server-Side Implementations

- AdGuard Home: https://github.com/AdguardTeam/AdGuardHome



DNS-over-QUIC port

853

If this port is configured, AdGuard Home will run a DNS-over-QUIC server on this port.

DoQ Client-Side Implementations

- dnsproxy (written in Golang, can be used as a library): https://github.com/AdguardTeam/dnsproxy
- AdGuard Home (written in Golang, uses dnsproxy internally): https://github.com/AdguardTeam/AdGuardHome
- DnsLibs (library, written in C++): https://github.com/AdguardTeam/DnsLibs
- dnslookup (simple nslookup-like util, supports DoQ/DoH/DoT/DNSCrypt): https://github.com/ameshkov/dnslookup

QUIC Implementations

- Golang: **quic-go**
  https://github.com/lucas-clemente/quic-go
- C++: **ngtcp2**
  https://github.com/ngtcp2/ngtcp2
- Rust: **quiche**
  https://github.com/cloudflare/quiche

Thank you!

Questions?

Andrey Meshkov
am@adguard.com
@ay_meshkov