Is large-scale DNS over TCP practical?

Baptiste Jonglez
PhD student, Univ. Grenoble Alpes, France

14-18 May 2018
RIPE 76

DNS over UDP

Issues with DNS over UDP

Well-known issues:

- no source address validation: huge DDoS attacks by reflection/amplification
- requires IP fragmentation for large messages (DNSSEC)
- no privacy
- unreliable transport

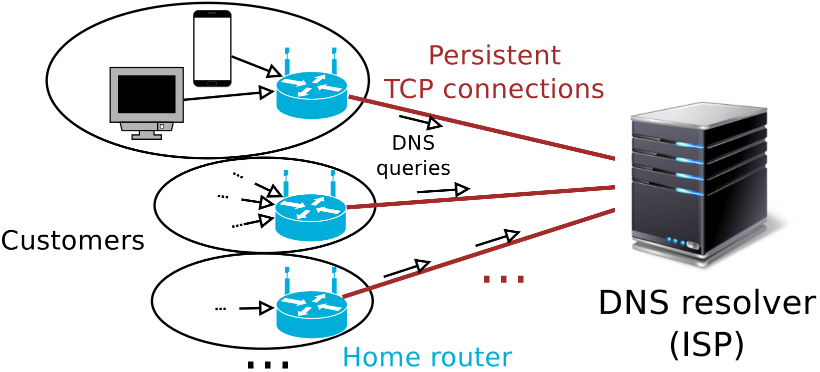
Each individual issue can be worked-around: RRL, Ed25519 for DNSSEC, DTLS or DNSCurve for privacy...

Solution to all four problems: TCP+TLS (see DPRIVE, RFC 7858)

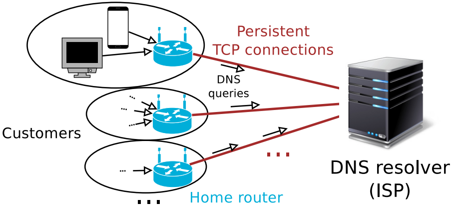
Switch to DNS over TCP by default?

Scenario

Use **persistent TCP** connections between home routers and recursive resolvers, for **all customers** of an ISP.



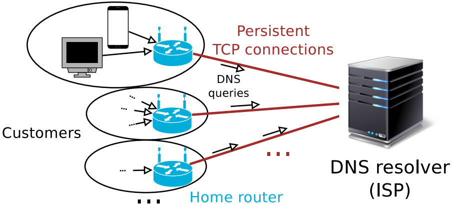
Switch to DNS over TCP by default?



Why is this situation useful to study?

- The resolver could stop supporting UDP queries (no reflection attacks possible!)
- First step towards TCP+TLS for privacy
- Using persistent TCP connections can improve responsiveness on lossy networks (not shown here)

Switch to DNS over TCP by default?



Objection

"But wait, our recursive resolvers won't handle that much load!"

Goals: performance analysis

- Develop a **methodology** to measure resolver performance
- Experiment with **lots of clients** (millions) to assess whether a recursive resolver can handle that much TCP connections
- See if **resolver performance** depends on the number of clients

Challenges

Why would performance depend on the number of clients?

- performance of `select`-like event notification facilities (bitmap of file descriptors, linear search)
- the kernel has to manage millions of timers (retransmission on each TCP connection)
- memory usage, CPU cache

Experimental challenges

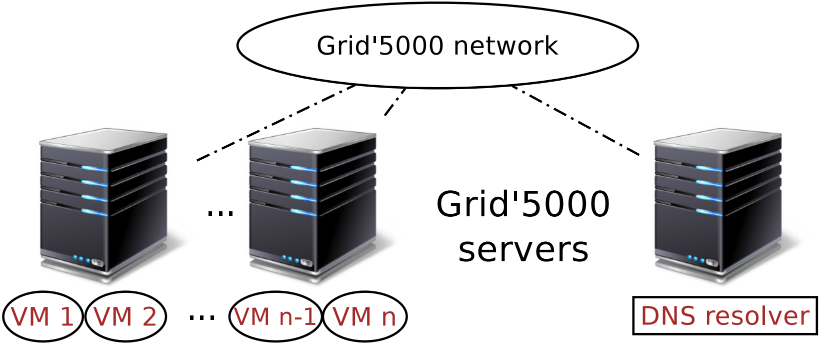
Practical challenges

- How to spawn millions of DNS clients?
- Realistic query generator?

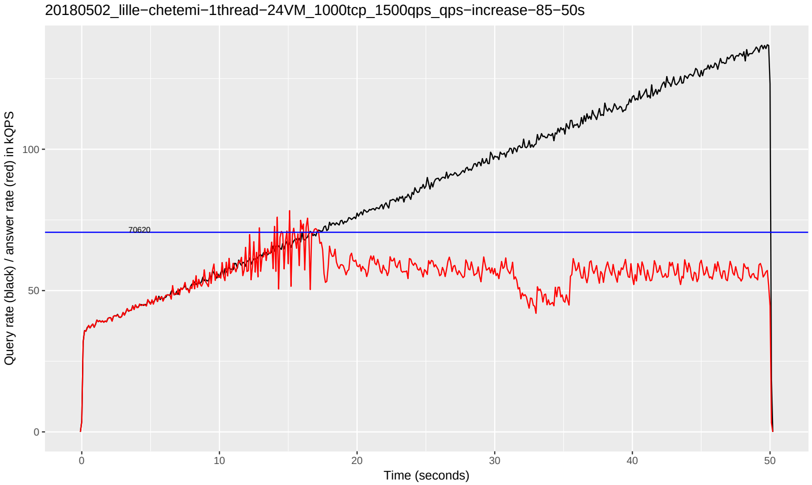
Solution

- Use **Grid'5000**: a "Hardware-as-a-Service" research platform, with lots of powerful servers: 32 cores, 128 GB RAM, 10G NICs;
- One dedicated server for `unbound` on Linux, everything served from cache;
- Lots of Virtual Machines acting as clients;
- On each VM, open 30k persistent TCP connections towards the server and send DNS queries with custom client in C with libevent;

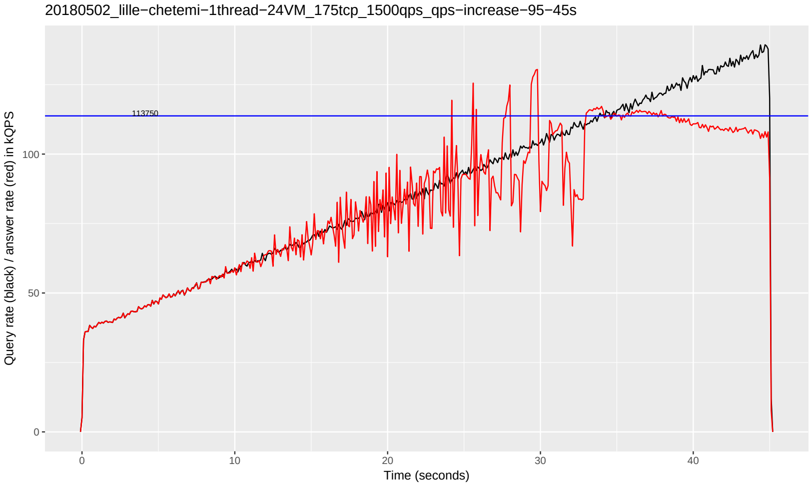
Experimental setup: high-level



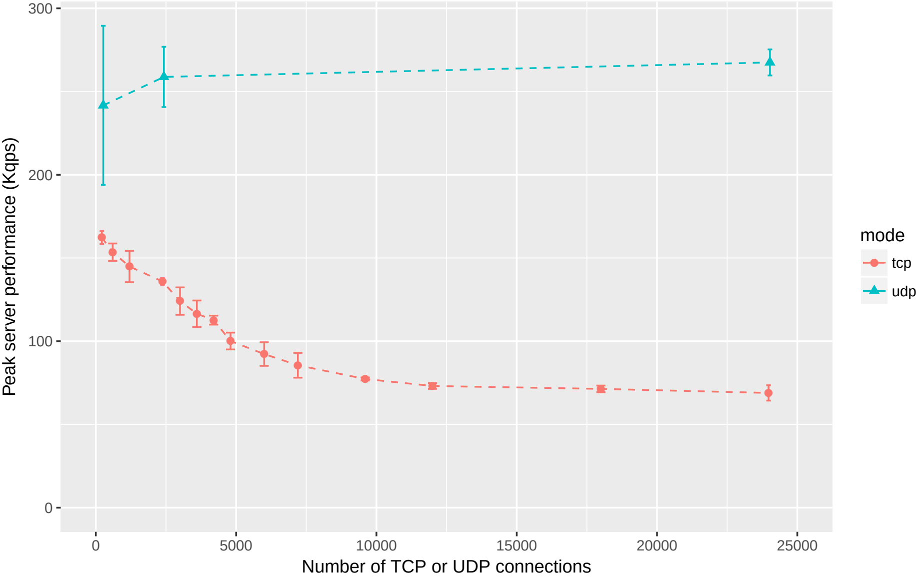
Methodology: how to measure performance?



Methodology: how to measure performance?



UDP/TCP comparison



UDP/TCP comparison

Interpretation

Resolver performance analysis

- settings: unbound runs on 1 thread
- UDP performance does not really depend on the number of clients, as expected (stateless)
- performance over TCP is good with very few clients, but then drops rapidly
- it then reaches a plateau: stable 50k to 60k qps even for 6.5 million TCP clients!

Hypotheses for performance drop

- more clients → lower query rate per client, so less potential for aggregation (in TCP, `select()`, ...)
- TCP data structures may not fit anymore in CPU cache?

Large-scale experiment

Result

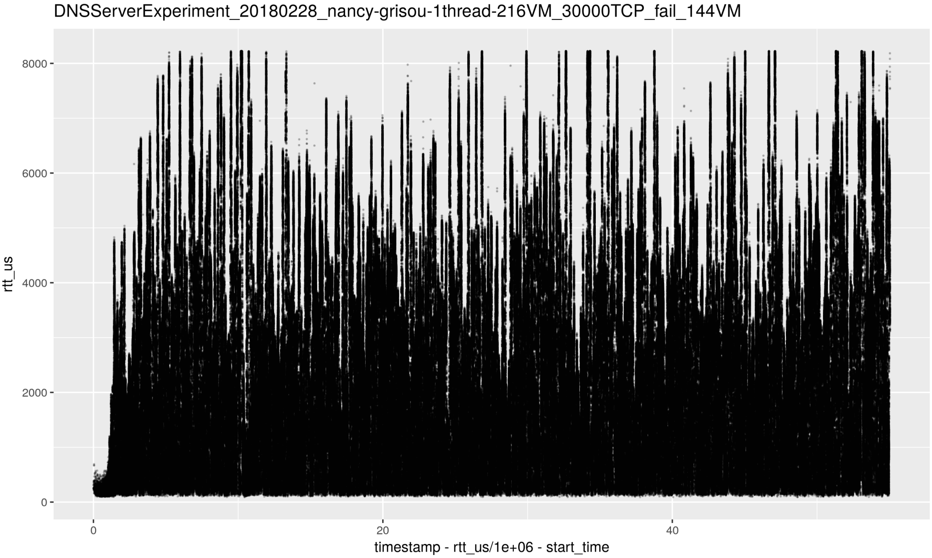
Experimented with up to **6.5 million** TCP clients:

- required 216 client VM running on 18 physical machines
- each VM opened 30k TCP connections to resolver
- server had 128 GB of RAM, peak usage: 51.4 GB (kernel + unbound)
- server performance: around 50k queries per second

Memory usage breakdown per connection: 4 KB for unbound buffer, 3.7 KB for the rest (unbound, libevent, kernel)

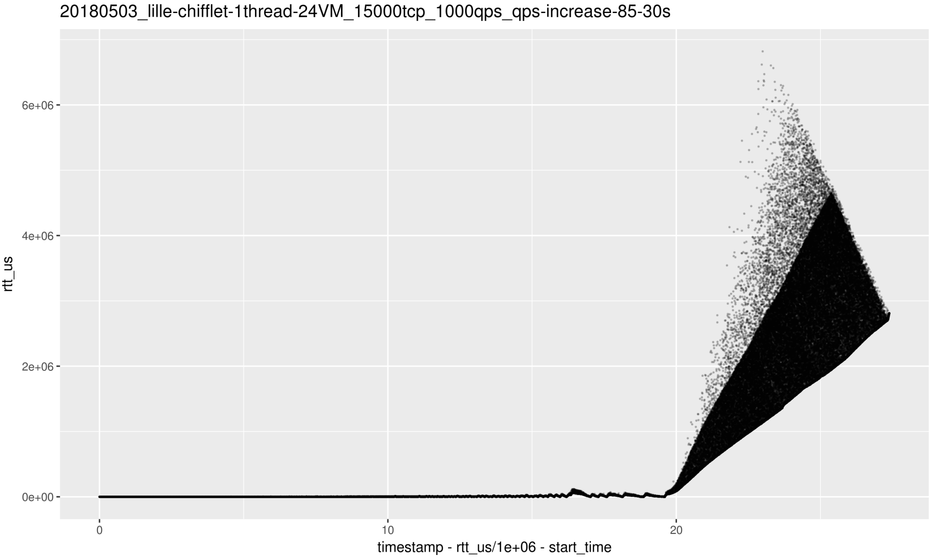
What about client query delay?

Medium-high load: 43 kQPS from 4.3M TCP clients

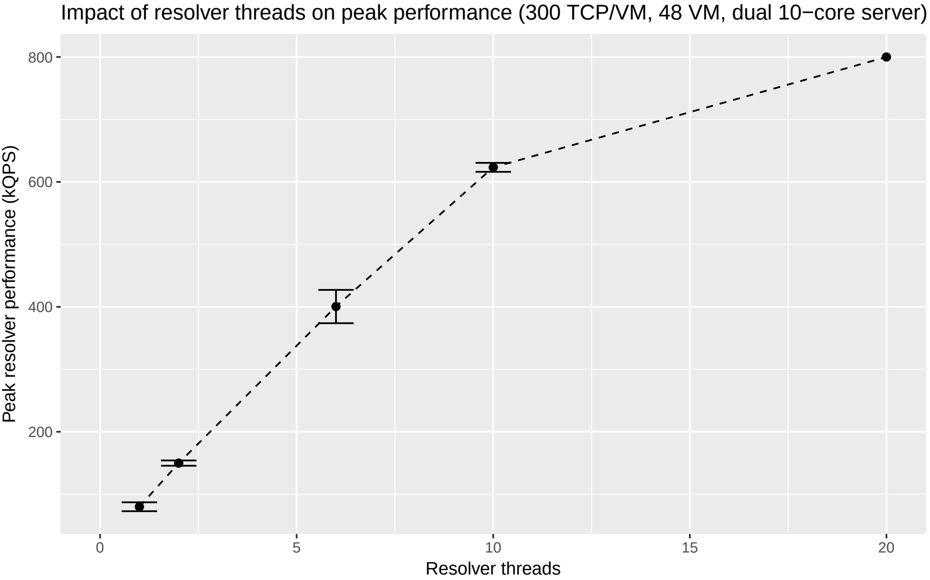


What about client query delay?

Increasing to overload, 360k TCP clients



What about multi-threading?



Assumptions and outlooks

Some assumptions we made

- everything was served from **static zone** in unbound (= cache)
- we currently open **all** TCP connections beforehand → cost of **client churn**? what about TLS?
- client queries modelled as **Poisson** processes → any better model?
- could we somehow experiment with **constant query rate** per client?

Setup

Detailed setup

- Linux 4.9 (Debian stretch)
- Unbound 1.6.7, with 4 KB of buffer per TCP connection, and no disconnection timeout
- custom libevent-based client: `https://github.com/jonglezb/tcpscaler`
- experiment orchestration: `https://github.com/jonglezb/dns-server-experiment`
- Grid'5000: `https://www.grid5000.fr`
- Hardware details (mostly used Chetemi, Chifflet, Grisou): `https://://www.grid5000.fr/mediawiki/index.php/Hardware`

Conclusions

DNS-over-TCP is feasible on a large scale

- with 6 million TCP clients, unbound can still handle around 50k queries per second per CPU core
- apparently unlimited number of TCP clients (requires OS tweaking and enough RAM)

Remaining work

- better understanding of the server performance drop
- measure impact of client churn
- performance when not serving from DNS cache?
- apply methodology to more recursive resolver software
- experiment with TLS, QUIC, SCTP

Bonus slides

Aside: unreliable transport?

Queries or responses can be lost.

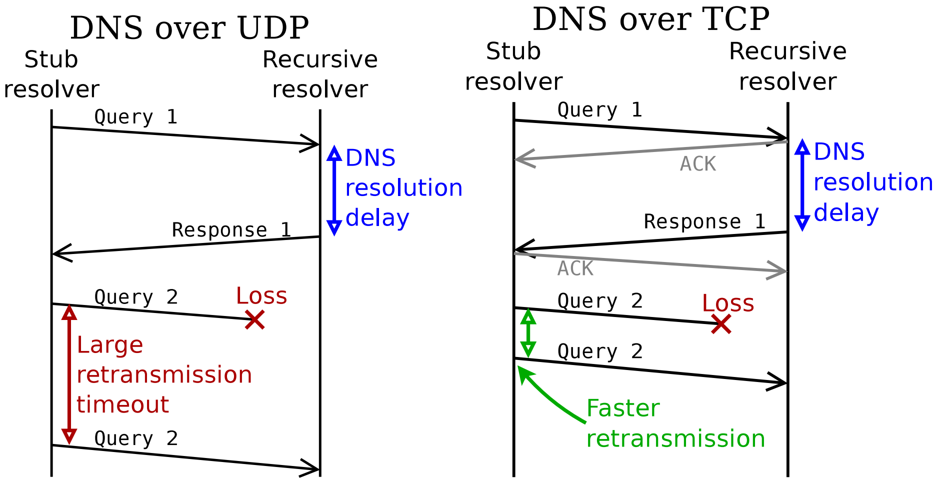
Retransmission timeout

Large retransmission timeout when a DNS query is lost!

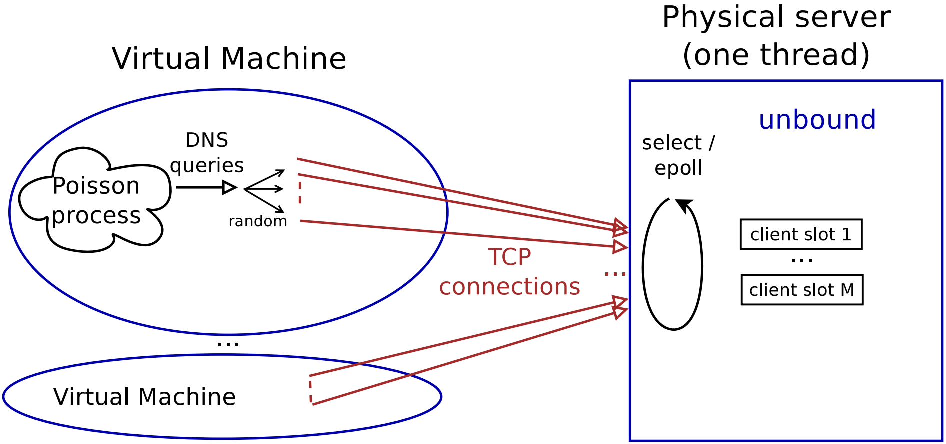
Retransmission timeouts in stub resolvers:

- **Linux/glibc:** 5 seconds, configurable down to 1 second
- **Android/bionic:** identical (but there is a local cache)
- **Windows:** 1 second (since Vista)

Why not just lower retransmission timeouts?



Experimental setup, details



Experimental setup, more details

Setup

- all queries are answered directly by unbound (100% cache hit)
- unbound was modified to allow infinite connections (very large timeout)
- everything scripted with `execo`, fully reproducible:
  `https://github.com/jonglezb/dns-server-experiment`
  `https://github.com/jonglezb/tcpscaler`

Gotcha

- generating queries according to a fast Poisson process is tricky!
- `epoll()` has very low timeout resolution compared to `poll()` or `select()`...
- Linux has several limits regarding the number of file descriptors, but they can all be configured at runtime (thanks Google...)