The road to DNSRocks

Why would you write your own authoritative DNS server?

Balint Csergo
Production Engineer

Meta

Agenda

01 History

02 TinyDNS

03 DNSRocks with CDB backend

04 DNSRocks with RocksDB

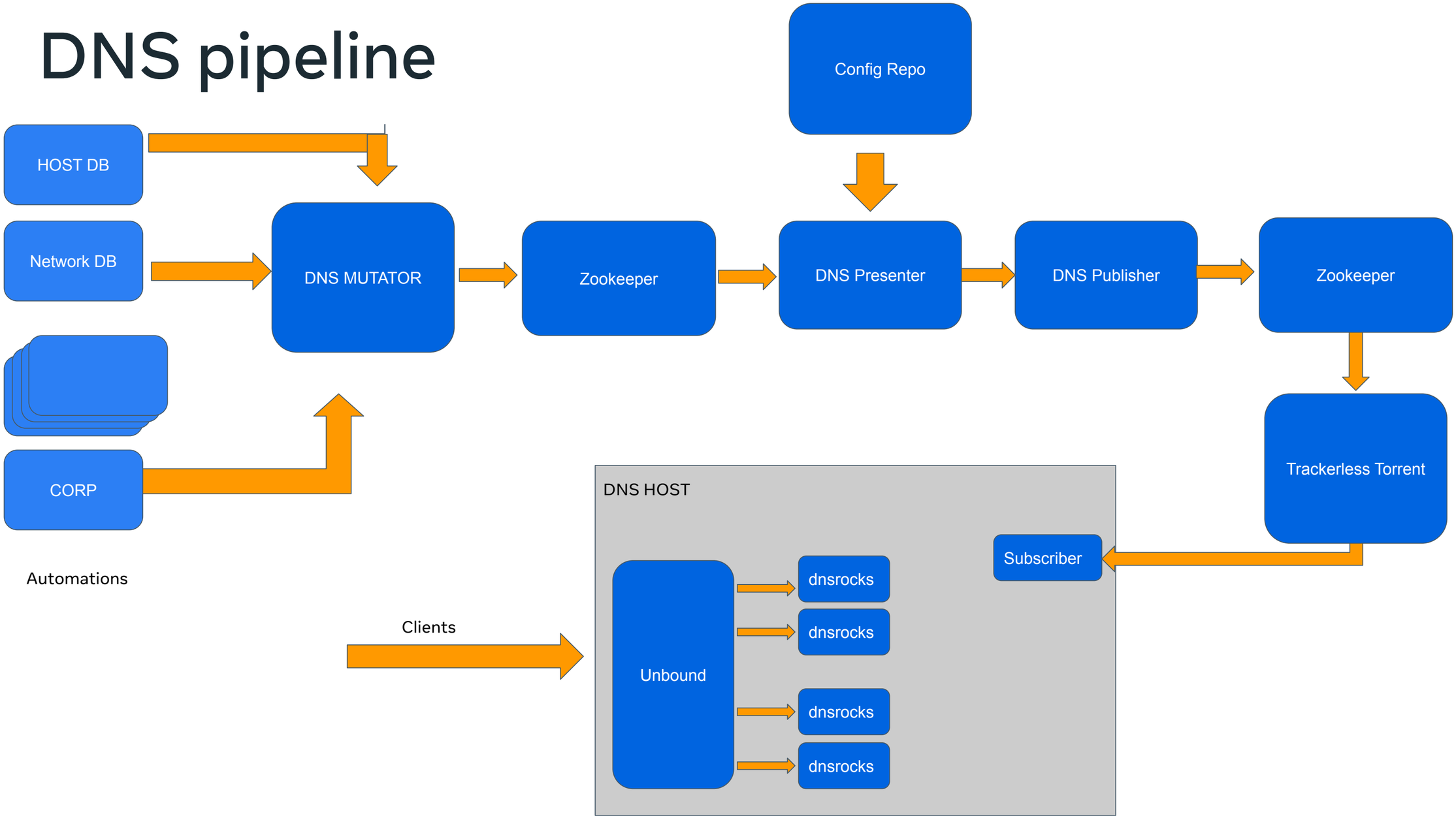
05 DNSRocks OpenSource

01 History

What do we need from a DNS server?

- Resolver IP based “views”
- EDNS Client Subnet “views”
- Simple to generate views
- Simple to configure
- Simple to deploy DB updates
- Query Logs
- Health metrics
- Latency requirements

DNS pipeline



01 History

TinyDNS

We have been using TinyDNS (with patches)

- IPv6 support
- EDNS Client Subnet location matching
- Multiple Map support

DNSRocks with CDB backend

We start running our own implementation of an authoritative DNS server. We are still using a TinyDNS compatible format by default

DNSRocks with RocksDB

We migrate from CDB to a RocksDB backend to support data growth

Open Sourcing DnsRocks

Split off proprietary bits and provide useful Open source alternatives

Switch to upstream dependencies where it made sense

02 TinyDNS

What was great about TinyDNS?

- Simple
- Efficient
- Very fast
- Line-based zone format
- Distributing data.cdb is simple

What was not so great about TinyDNS?

- Antiquated hard to read C code
- Not easy to extend
- Simplicity comes with trade-offs
- Lack of tests and modern programming paradigms
- Lot of global/static variables
- Hard for engineers to ramp up
- Single threaded

Open Source Alternatives

	Resolver View Supported	ECS View Supported
BIND	✅	❌
Knot	❌ (Until 2.7.0)	❌ (Until 2.7.0)
NSD	❌	❌
PowerDNS	✅(geoip backed)	✅(geoip backed)

03 DNSRocks with CDB Backend

Roll your own

- Go is a modern language
- miekg/dns is an amazing library, we've built on top of that
- We reuse a lot of plumbing from CoreDNS
- Go tooling makes building/running/testing easy
- Good unit test coverage was a must
- Faster iteration speed
- Works with CDB, compatible with internal tooling
- Feature parity with TinyDNS, so rollout and A/B testing was "simple"
- A/B testing enabled us to profile our code, and optimize hotspots

04 DNSRocks with RocksDB

What’s good about CDB

- speed: Data access is very very fast
- It’s a constant database, it can be mmapped
- We know how to work with it already
- It’s a very simple data structure, basically a hashmap

What’s bad about CDB

- It’s a constant database, we need to redeploy the whole view on every change, which causes constant recompilation, and increased propagation time
- This constant DB rewrite leads to SSD wear, we needed to store the DB in Ramdisk
- It’s a 32 bit hashmap. 4 GB is the maximum size of a cdb database

RocksDB

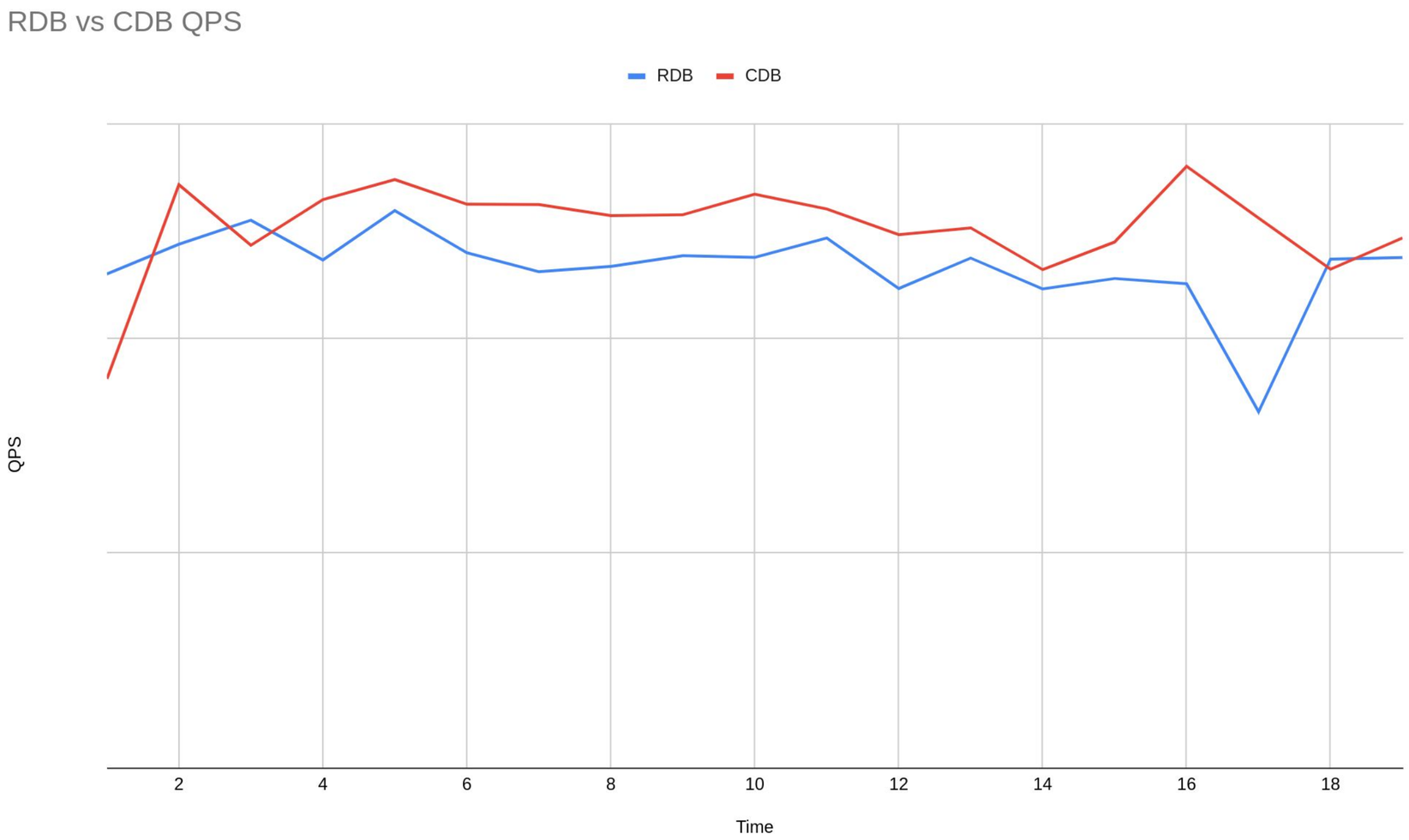
- speed: RocksDB being mutable means we don’t recompile anything. Incremental updates are possible and applying changes takes milliseconds.
- SSD wear: RocksDB is flash-ready, which meant we’ve been be able to move back to SSDs and free up some precious RAM
- 32-bit limitations of CDB no longer apply, shards can get as big as we need. 4 GB isn’t enough for everybody :)

RocksDB

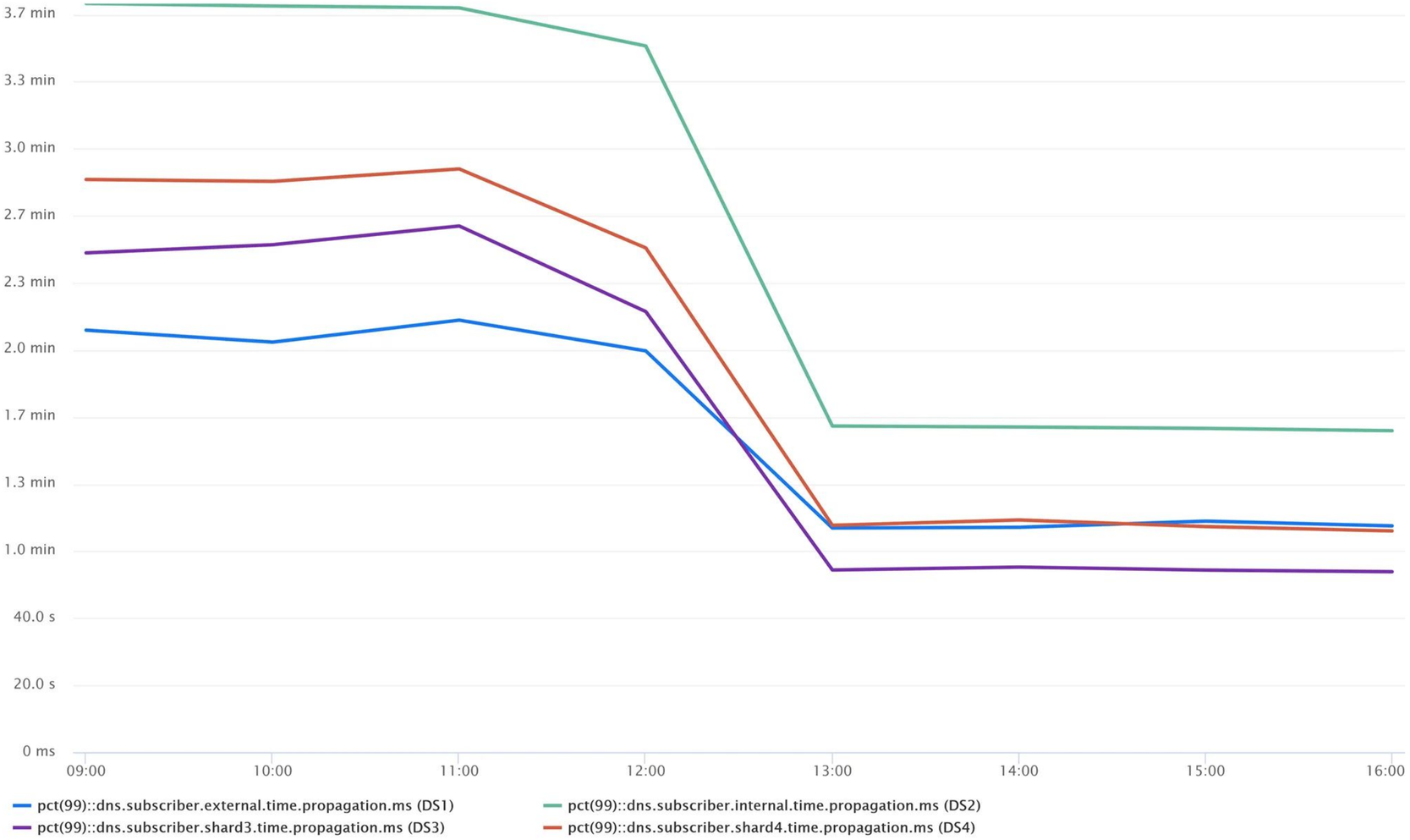
The steps we took:

- Writing CGO wrapper around RocksDB so it could be used in Go
- Adding alternative backend support in DNSRocks
- Creating native go parser and compiler from TinyDNS configuration format to CDB and RocksDB
- Rethink how we do maps to better utilize RocksDB’s findClosest

DNS server performance comparison



DNS propagation time wins on Rocksdb



05 DNSRocks Open Source

DNSRocks Open Source

The steps we took:

- Split off proprietary pieces of code
- Provide reasonable OSS alternatives (Prometheus metrics + dnstap logger)
- Switch to upstream dependencies wherever possible
- CI pipeline on github
- Documentation

Thanks

- Manu Bretelle (@chantra)
- Yaroslav Kolomiiets (@yarikk)
- Oleg Obleukhov (@leoleovich)
- Andrei Lukovenko (@alukovenko)
- Alex Bulimov (@abulimov)
- Pablo Mazzini (@pmazzini)
- Patrick Cullen (@t3lurid3)
- Vadim Fedorenko (@vvfedorenko)
- Trinity Mirell (@trynity)
- Maxime Montinet
- Jinyuan Feng
- Ivan Laktyunkin
- Yang Yu

04 Questions


Meta