facebook

Building and Deploying a new Nameserver

Manu Bretelle

Production Engineer

DNS OARC 31 / 2019-10-31

DNS@FB History

DNS@FB

History

- Using tinydns since 2012
- IPv6 support
- CIDR based location matching
- EDNS Client Subnet location matching
- Multiple Map support

DNS@FB

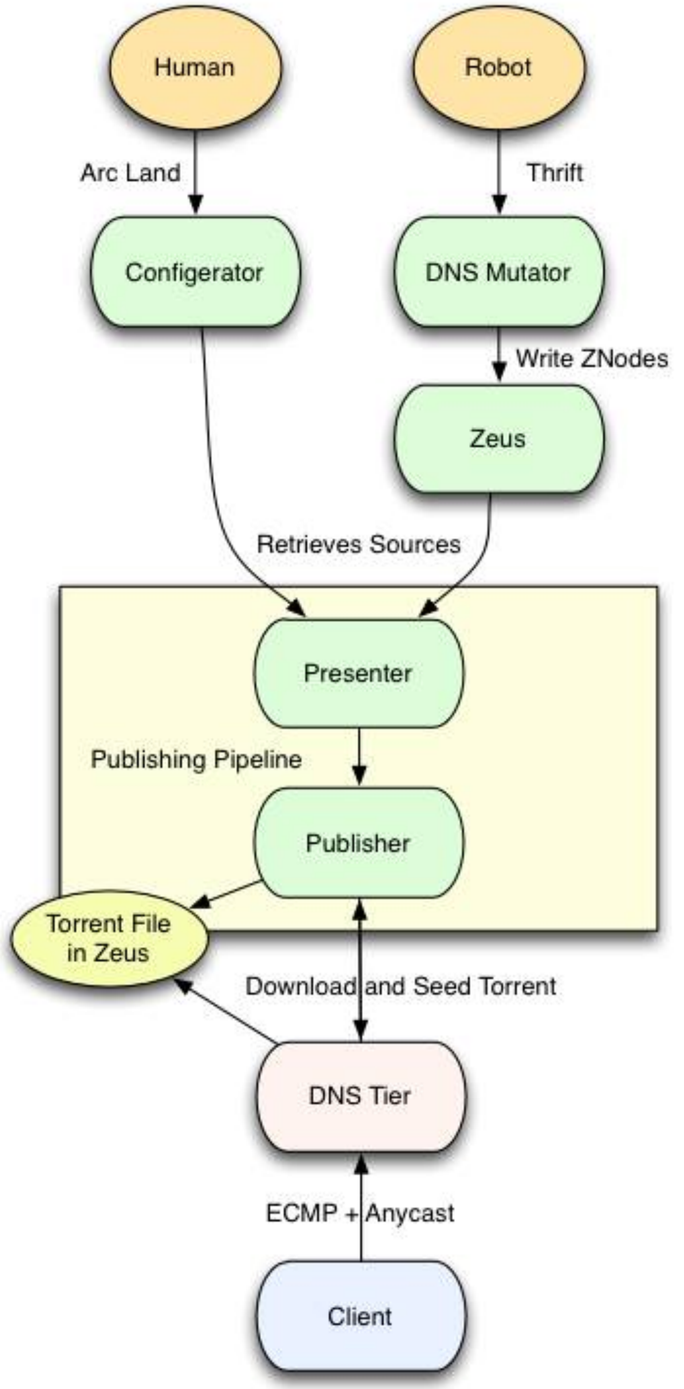
Feature Requirements

- Resolver-based “views”
- EDNS Client Subnet “views”

DNS@FB

Operational Requirements

- Simple to generate views
- Simple to configure
- Simple to deploy DB updates
- Query Logs
- Service Health Metrics
- Easy to integrate in our infrastructure.



DNS@FB

What's great about tinydns?

- Simple
- Efficient
- opinionated
- line-based zone format
- distributing data.cdb is simple

DNS@FB

Why moving off tinydns?

- 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

Searching alternatives

Open Source

January 2018

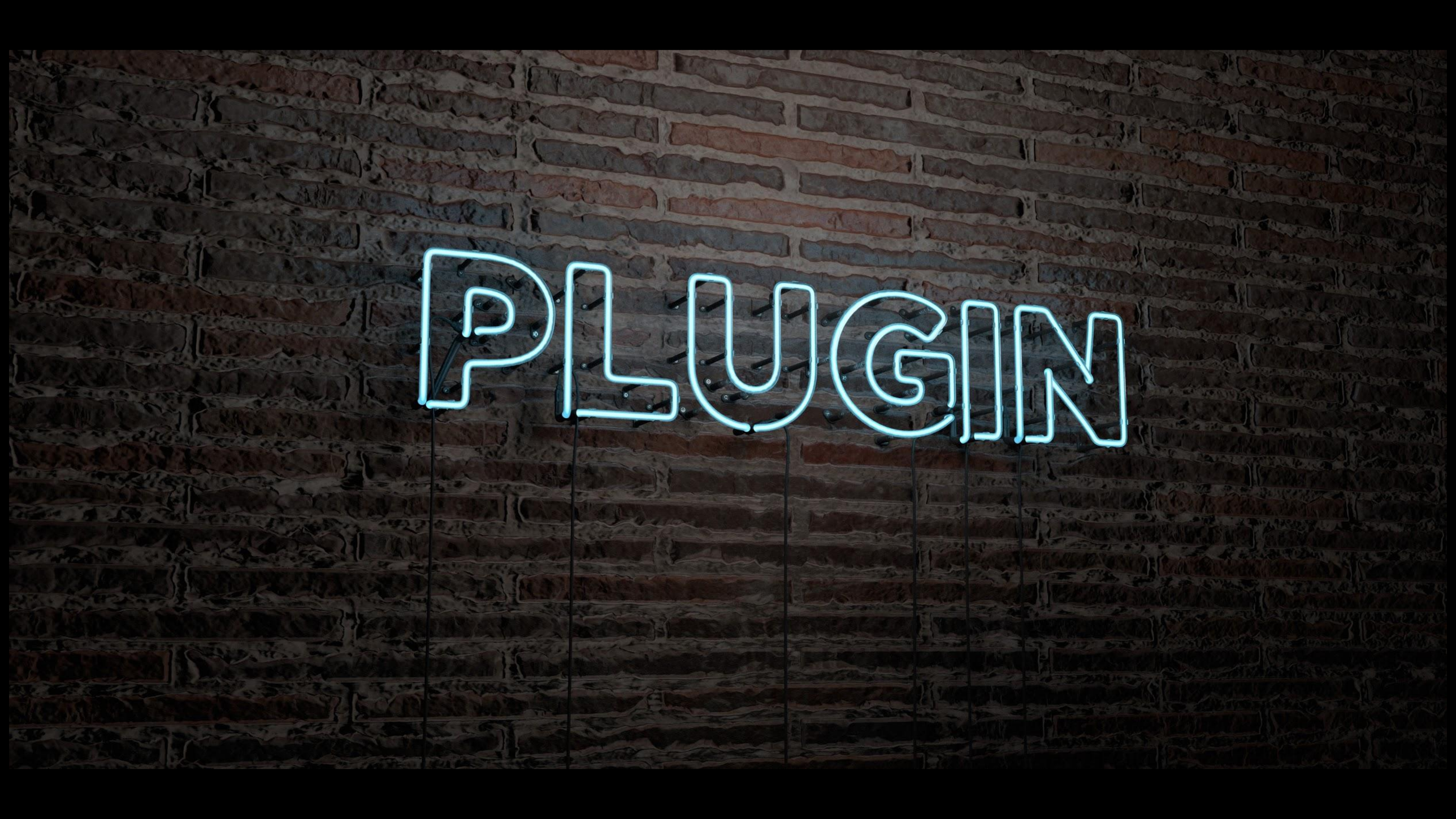
	Resolver View	ECS View
BIND	Yes	No
Knot	No (until 2.7.0)	No (until 2.7.0)
NSD	No	No
PowerDNS	Yes (geoip backend)	Yes (geoip backend)

What IF?

Operation

Had they been available solutions, we would have needed to make sizeable changes to our existing pipeline.
Deploying a .cdb works well for us
we have plenty of tooling available that does sanity checking and operational work for us.

What then?


PLUGIN

Plug your own logic in

aka building a module

- CoreDNS
- Knot DNS
- PowerDNS

Picking up a solution

Building a Proof of Concept

- How easy to get started with it?
  - What is the learning curve?
  - How easy to build/test?
- How familiar are engineers with the language/library?
- How easy would it be to integrate in-house?

Picking up a solution

CoreDNS

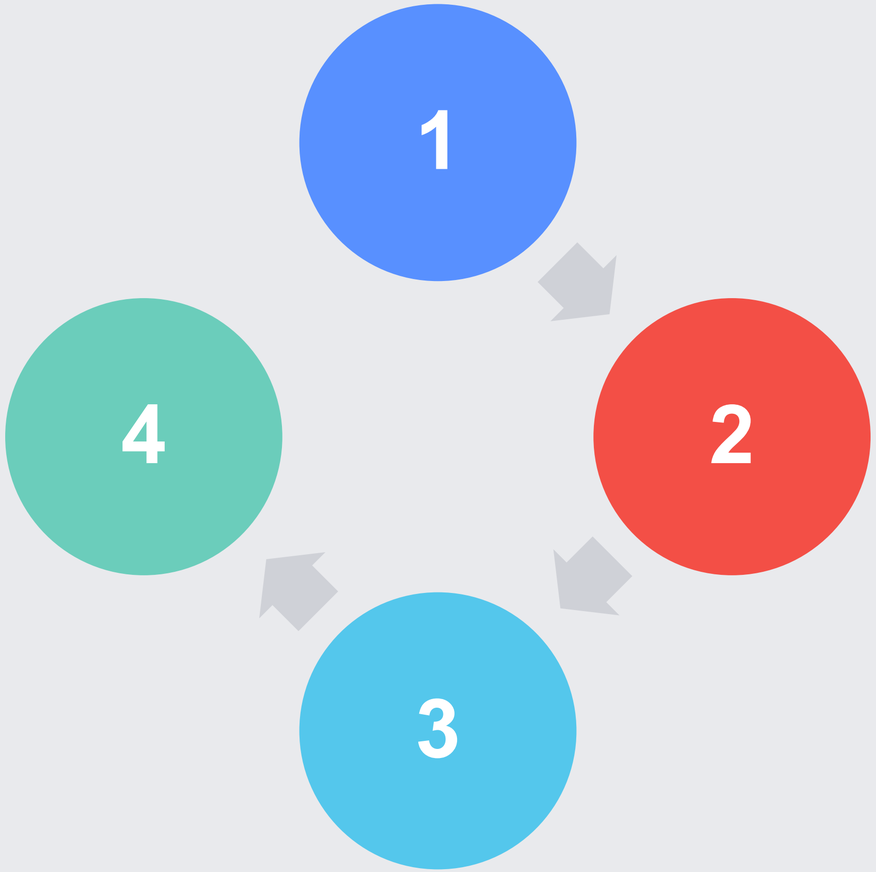
- Excellent plugin tutorial
- miekg/dns:
  - already used in our DNS pipeline
  - simple to use
- Go tooling makes building/running/testing easy

Development

From Proof of Concept To Prod

PoC

Get something working to demonstrate feasibility.



Compatibility

Support feature parity. same input, same output

Integration

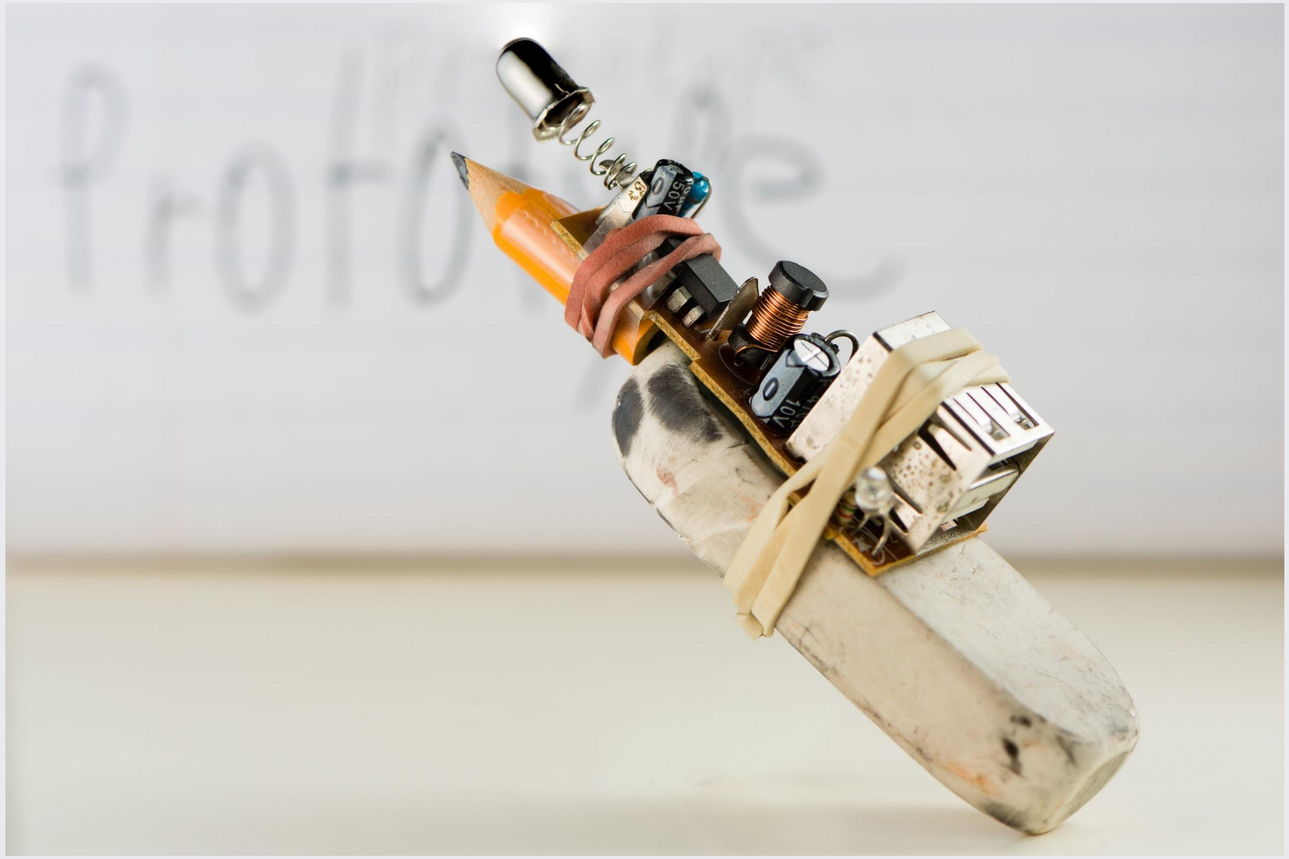
Make it work internally, remove unnecessary dependencies.

Production-ize

Logging, metrics, refactoring, performance.

Proof of Concept

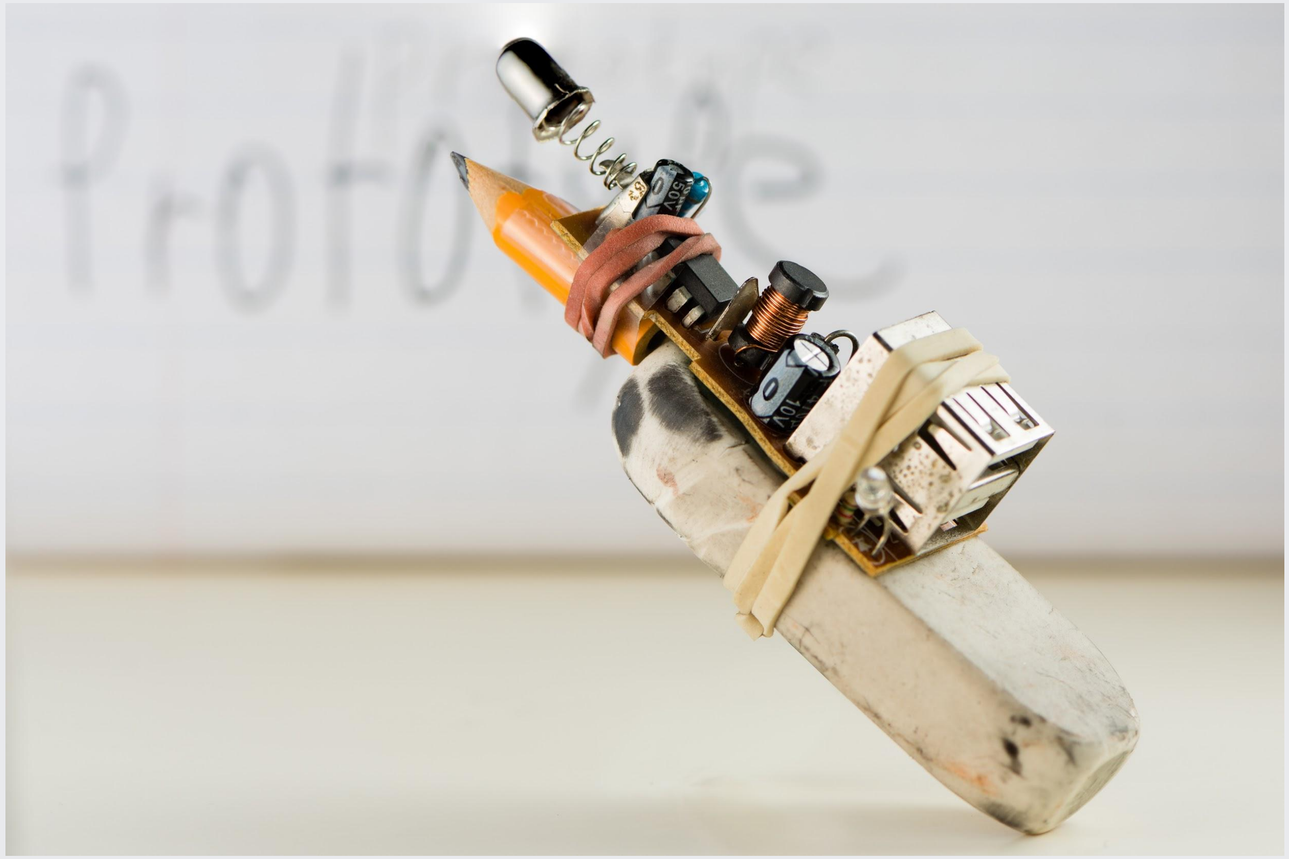
aka get some answers



- 3 weeks times
- Read from CDB
  - Map/Query matching
  - Location for Resolver/ECS
  - Unpack RDATA from CDB
- unittests, unittests, unittests, unittests
- dirty^Wquick hacks

Proof of Concept

aka get some answers



This is still a POC, but it does support (as in answer somehow correctly) some of the basic A/AAAA queries. Quick benchmark shows it is 3x slower than TinyDNS on a single core,

Compatibility

aka “actually works correctly”

- 2 weeks time
- Feature parity with tinydns
- Validate replaying queries against tinydns and fbdns
- Find bugs, fix them, unittest them, rinse and repeat
- Would work if taking prod traffic

Integration

aka “use internal toolchain”

- 2 weeks time
- Get rid of CoreDNS dependencies
- Build standalone UDP/TCP listening layer
- Reuse CoreDNS’s ServeDNS entrypoint and helpers

Production-ize

aka “prime time”

- 1 month time
- Query Logging
- Metrics
- refactoring
- profile, find hotspots, optimize
- rinse/repeat

Deployment

Deployment Strategy

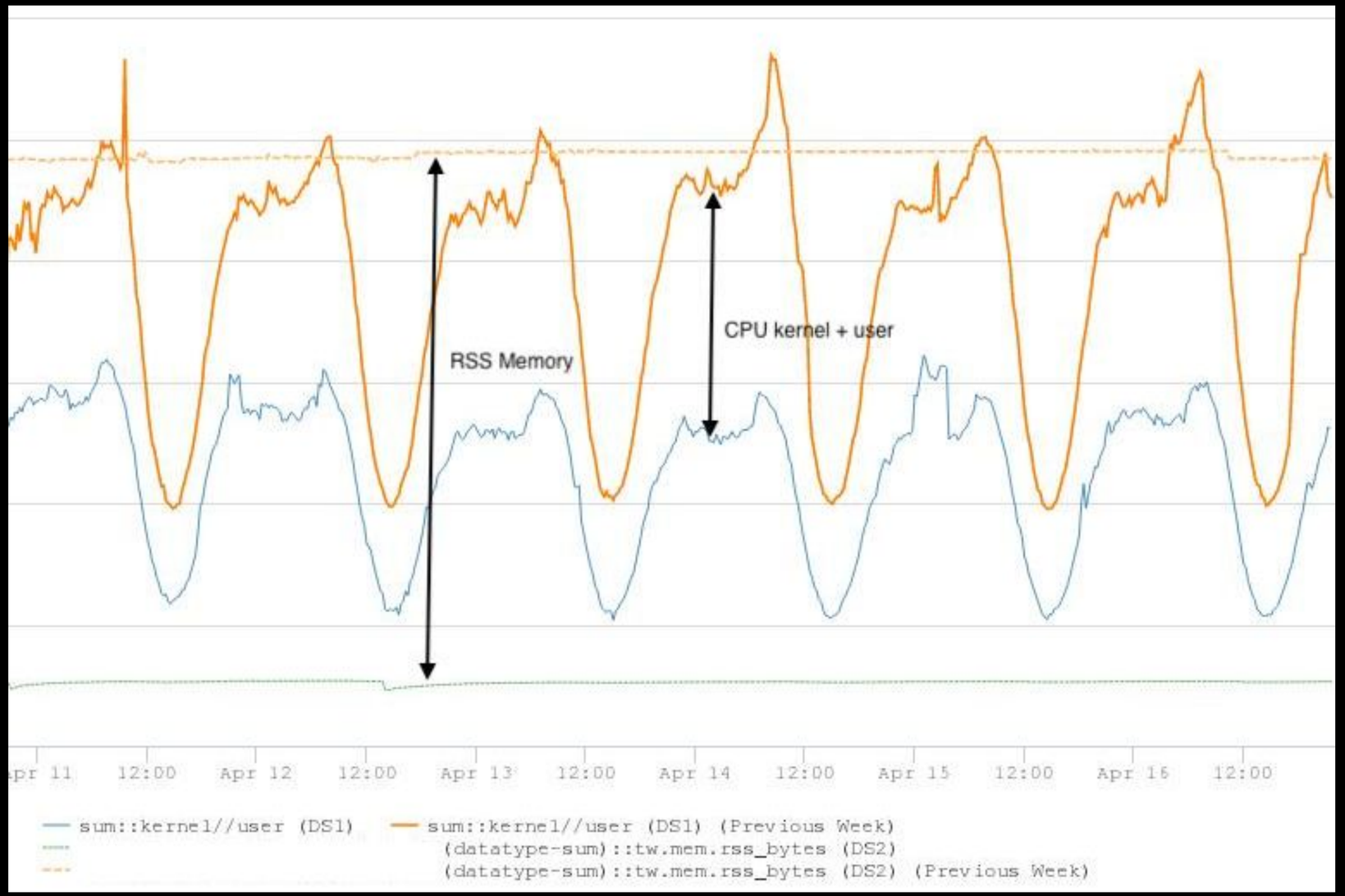
- Focus on `b` nameservers
- Leave `a` alone
- Start small
- Push early in the week
- Build confidence
- Deploy to more POPs

Test the water

Slow Start

- Couple of hosts in a cluster
- Eventually a full cluster
- Let it bake 1 week
- Look for traffic change
- Check performance

How does it compare?



Ramp it up

- 7 days later
- 1 more cluster
- Let it bake
- Look for traffic change

It’s working great!

gogogogogogogo

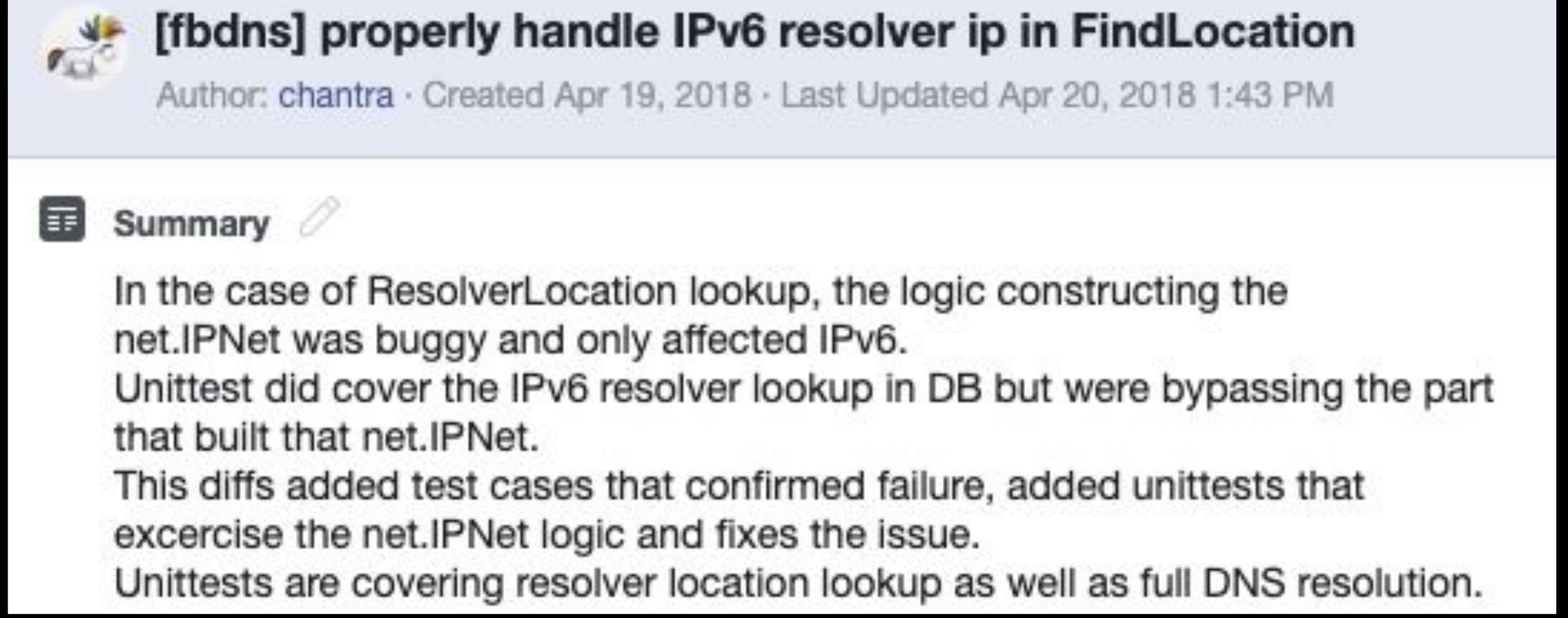
- 2 days later
- Deploy to more clusters at 12pm
- Later that day, employees started reporting issues
- Revert to tinydns
- Everyone is happy
- Troubleshoot, unittests, fix

Didn’t you have validating tests?

Yes…. but

- Source IP is hard to spoof if you want to receive the answer.
- Tests validated answers using ECS

What Happened?



[fbdns] properly handle IPv6 resolver ip in FindLocation

Author: chantra · Created Apr 19, 2018 · Last Updated Apr 20, 2018 1:43 PM

Summary

In the case of ResolverLocation lookup, the logic constructing the net.IPNet was buggy and only affected IPv6.
Unittest did cover the IPv6 resolver lookup in DB but were bypassing the part that built that net.IPNet.
This diffs added test cases that confirmed failure, added unittests that excercise the net.IPNet logic and fixes the issue.
Unittests are covering resolver location lookup as well as full DNS resolution.

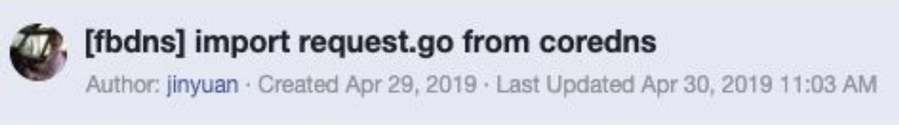
Next Monday!

Just keep rolling!

- Quickly back to where we were
- Over 2 weeks, rolled to 100% of `b` nameservers
- `a` nameservers
  - stayed on tinydns 5 more months
  - migrated to fbdns over 2 weeks

1 year later….

Pay the tech debt

- 2018-03-02 copy coredns/coredns:request/request.go
- 2018-03-21 coredns report “High CPU Load” GH#1625
- 2018-03-22 coredns fix in GH#1629
- 2019-04-30 

- 2019-05-01 

Fixing this

Clean the tech debt

- get rid of copy/paste:
  - use proper dependency management
  - avoid forks, work with upstream
- Keep third-party up-to-date
- Invest in regression detection
- Invest in continuous (staged) rollout
- Invest in Fuzzing

Thank You

- miekg/dns and CoreDNS
- BIND, Knot, NSD and PowerDNS
- gitlab.isc.org/isc-projects/DNS-Compliance-Testing
- github.com/DNS-OARC/dnsperf

facebook

Questions