DNS: Old solution for modern problems

Thomas Jackson & Rauf Guliyev
Traffic SRE
LinkedIn
What is it that “Traffic SRE” does

• Responsible for:
  • Global PoPs
  • Proxies
  • “Fixing” it

• Basically boils down to:
  • Get traffic from the user to the correct frontend in the correct DC as fast as possible

• What do we use?

traffic server™
Why service discovery?
Traffic is easy!

- Obviously hosts are where they are, how hard could it be?
- Load balancing is easy!
- Routing is easy!
Why service discovery?
What does our environment really look like?

- Multi-tenant
  - LinkedIn
  - Slideshare
  - Lynda
  - Etc.
- Legacy
- Lots of microservices: new ones all the time
- Dynamically changing scale
Why service discovery?
Isn’t this problem already solved?

• Common solutions:
  • Frameworks: rest.li / thrift / etc.
    • Cons: Request/Response + DynamicDiscovery (solves more than we want…)
  • custom (usually zk-based) solution
    • Cons: doesn’t scale to all platforms, all services, etc.

• Common problems
  • fairly high barrier to entry (meaning FOSS and legacy won’t easily integrate)

• In a large-scale, multi-tenant environment relying on any single solution can cause some problems
Picking a solution

Why do we need another service discovery thing

How standards proliferate:
(see: a/c chargers, character encodings, instant messaging, etc)

SITUATION:
There are 14 competing standards.

14?! RIDICULOUS! We need to develop one universal standard that covers everyone's use cases. YEAH!

SITUATION:
There are 15 competing standards.

https://xkcd.com/927/
Picking a solution
How do you pick a solution

1. Gather requirements
2. Find past/possible/passable solutions
3. Do it! (pick a solution)
Picking a solution: How do you pick a solution

Gather requirements

• What do we need?
  • reliable service discovery
  • Differentiate between hosts that “want” traffic and ones that don’t (OOR hosts)

• What do we want?
  • A single solution
  • Easily debuggable solution
  • Require no/little work
Picking a solution: How do you pick a solution

1. Find past/possible/passable solutions

- Multiple sources of truth (primarily internal topology, rest.li, range)
- Closest solution in existence was rest.li
- Idea for using DNS directly in ATS
Picking a solution: How do you pick a solution

Do it! (pick a solution)

- DNS: the original name -> IP solution

- We decided to go with dns-discovery and we are here to talk about it -- so it must have worked right? ;)

- And DNS is awesome, SRV records anyone?
Example SRV response

$ dig srv _http._tcp.profile.linkedin.com
...

;; ANSWER SECTION:
:http._tcp.profile.linkedin.com. 3 IN SRV 0 0 8080 profile1.linkedin.com.
:http._tcp.profile.linkedin.com. 3 IN SRV 0 0 8080 profile2.linkedin.com.
:http._tcp.profile.linkedin.com. 3 IN SRV 0 0 8080 profile3.linkedin.com.
...

;; ADDITIONAL SECTION:
profile1.linkedin.com. 3600 IN A 10.136.148.97
profile2.linkedin.com. 3600 IN A 10.136.148.98
profile3.linkedin.com. 3600 IN A 10.136.148.219
...

;; Query time: 13 msec
Architecture
How do we design infrastructure?

- Set your runtime priorities
- Create service contract
- While (!requirements_met)
  - Create design
  - Understand the design
    - How will it fail (hopefully NOT fail, but it is inevitable)?
    - How will it scale?
    - How will it be extended?
- Document it!
Architecture
Set your runtime priorities

• Terms:
  • Availability: will it respond to a query?
  • Consistency: will all of them respond with the same thing?
  • Accuracy: will the response be the same as the data source?

• For this particular product: availability -> consistency -> accuracy
Architecture
Service Contract

• Eventually consistent data across the cluster
• Best-Effort consistency to data source
• Best-Effort "real" status
• Best-Effort "host" resolution

• Client is responsible for
  • Following DNS RFC (for failover)
  • Load balancing
Architecture

V0.1

DNS resolver

Source of truth
Architecture

V0.2

DNS resolver

In-Memory cache

Source of truth
Architecture

V0.3

In-Memory cache

DNS resolver

Gossip

Source of truth
Architecture

V0.4

In-Memory cache
Eventual persistence
DNS resolver
Gossip

Source of truth
Source of truth
Source of truth
Architecture

V0.5

- In-Memory cache
- Eventual persistence
- DNS resolver
- Gossip
- Plugins
- Source of truth
- Source of truth
- Source of truth
Architecture

V0.6

- In-Memory cache
- Eventual persistence
- DNS resolver
- Gossip
- Distributed Healthchecking
- Plugins
- Source of truth
- Source of truth
- Source of truth
YOU HAVE A QUESTION

I HAVE AN ANSWER.
Architecture

Heading off some common questions

• The source of truth should be able to handle all lookups!
  • Even if we assume that all sources of truth could (which is a fairly large "if") we don't necessarily want it to-- as there are potential tradeoffs made for accuracy over reliability (even though they may be small)

• But we can make everything use X instead of using this
  • This works in theory, but in practice something is always outside of "everything" (LBs, FWs, acquisitions, etc.)

• What about availability?
  • We use BGP to announce the same anycast IP address from multiple hosts
Implementation

Why is SRE building this anyways?

• This is a key piece of infrastructure, and at first we were the only users on the roadmap
• Infrastructure should be written by those who support it-- and have to handle the phone calls when it breaks
• Because we can, E - engineers 😊
Implementation

Prototype – in Python

• 1 week to get to staging

• Python ran into some serious scale problems in staging
  • Able to do 800 healthchecks/s per host – which isn’t a lot!
  • Curse you GIL!!!
Implementation
First cut – in Golang

• ~1 day

• Basically, rough around the edges but lightweight and extremely quick thanks to golang (and goroutines)!
Implementation

Bake time

• Test function, failure, and scale
• DNS is EASY to test!
• Golang makes concurrency testing easy
  • go build –race
• We weren’t even the first users! (we were third!!)
Implementation

Problems while baking

• Concurrency
  • Deadlocks—channels are blocking!

• Concurrency
  • RWMutex -- can’t recursively acquire
Productionalizing

Monitoring/Alerting guidelines

• Metrics
  • Service health
    • Gororoutines
    • CPU/Memory usage
    • Gossip
    • Pacemaker delay
    • BGP
  • Plugin health
    • Number of lookups, loads, response latency, etc.
    • Customer focused metrics (how do we perceive your service)

• Alerts
  • All of the metrics you care about 😊
  • Tune thresholds as you go
Outcomes

• Significantly reduced complexity
  • ~2k unique DNS service names
• Dramatic decrease in convergence time (~3s instead of 1+ day)
• Ubiquitous service discovery
  • Even curl works with it!!
  • Other people are already using it!
• Leverage existing DNS infrastructure
  • We get ~800 QPS of the ~25k total QPS
• Self-supporting community
GREAT SUCCESS!