Rebuilding the airplane in-flight... (safely)
• **We rewrote business-critical software**

Here’s how it went.

**Spoiler alert:** It’s been a huge success!
The What, Why & How
What We Do

www.foo.com?
www.foo.com?
ISP’s nameserver

NS1
Authoritative for foo.com

www.foo.com?
www.foo.com?

lookup

database
lookup | up | geo_target | select
www.foo.com?

lookup | up | geo_target | select

database

geo data

monitoring data
Our custom nameserver (nsoned)

www.foo.com?

lookup | up | geo_target | select
• **Why a Rebuild?**

  • **Success!**
    - Extreme growth putting the pressure on MVP
  • **Needed better performance**
    - We scale horizontally
    - Economics of horizontal scale with inefficient software is threatening to the business... Just like the risks of a large rewrite.
  • **A big new feature**
    - DNSSEC (didn’t want to build into old server)
  • **Wanted a platform built for evolution**
How ~ Requirements

- Improve performance (QPS) by at least 10x
- Address correctness
- Drop-in replacement
  - So old and new versions could run side-by-side
- Zero-downtime deployment
• **How ~ The Plan**

  • Decided to use C++14
  • Previously created an architecture plan based on POCs
  • Start with team of 4 senior engineers
  • Two-phase execution plan
    • Phase I: Running old and mini-new side-by-side, 3 months
    • Phase II: Do everything else, 3-4 months
  • Okay, let’s go!
• Trex!
Execution
It Did Not Go As Planned

- Phase I landed smoothly, on-time
  - 3 months, as estimated!
- Phase II DELAYED
  - Took 8 months, not the 3-4 months we estimated.
- So what happened with Phase II?
Phase II

- Phase II felt like the “atom” of the project
- We declined to break it down further
  - Taking time to do hard planning vs. keep executing
- We needed additional developers but added them too late
- Plus, we were hiring aggressively!
  - Additional time taken from senior devs
  - The same folks working on Trex
Phase II

- Phase II felt like the “atom” of the project
- We declined to break it down further
  - Taking time to do hard planning vs. keep executing
- We needed additional developers but added them too late
- Plus, we were hiring aggressively!
  - Additional time taken from senior devs
  - The same folks working on Trex

So... pretty pedestrian stuff.
I’ll be praying for pedestrian failures from now on.
It Has Been a Huge Success

• Exceeded our performance goals
• Delivered DNSSEC
• Survived the pain of late delivery
• There was much rejoicing
  • Yay!
It Has Been a Huge Success

- Exceeded our performance goals
- Delivered DNSSEC
- Survived the pain of late delivery
- There was much rejoicing
  - Yay!

We met our requirements.

Still could have been a Second System!
• Second-System Effect
Second-System Effect

“The second is the most dangerous system a person [sic] ever designs.”

“the tendency is to over-design, ... using all the ideas and frills that were cautiously sidetracked”

-- Fred Brooks, TMM
I Define It As...

The tendency of movies after The Matrix to **focus on the thing that didn’t matter**.
Why Are Second-Systems Hard?

Original system was not shooting (only) for success. It was striving for existence!

There is a simplicity in that that is essential.

And that’s hard to replicate.
We Avoided It...

- Experienced team helps
- Simple mantra: Correct first, Optimal second
- Quite often, correct is optimal
- If not then you have a correct solution to start from!
Safe, Sitewide Deploy
Deploy Strategy: Layers of protection

1. Slow **roll out period**, targeting 2-3 months
   - Utilize our anycasted edge network
   - **Canary** deploys, sending traffic to a percentage of POPs and total cores

2. Prove **correctness**
   - **Tee testing** to compare traffic and ensure correctness/compatibility

3. Prove **operational reliability**
   - Detailed operational **metrics** and central exception **logging**
Global Load Balancer: Anycast

- Network space is announced in multiple locations
- BGP announcements dictate where traffic flows
- Well distributed edge network promotes low global latency

(not to scale, illustrative only)
Traffic distribution follows announcements

- Any combination of POPs may be brought down for maintenance: traffic automatically redistributed
- Which POPs are up affects latency, not functionality
- Usually, all are up providing best performance

(not to scale, illustrative only)
Traffic Bastions: SuperPOPs

- SuperPOPs have enough capacity to maintain global traffic load
- Extra network uplink and capacity to scrub under attack

(not to scale, illustrative only)
How anycast helped our deploy

1. Decide which POPs to convert to new server
2. Bring down BGP announcement
3. While announcement is down, anycast gives us transparent traffic redistribution to remaining POPs
4. Deploy new code, test it, bring BGP announcement back up

In worst case scenarios, always able to fall back to SuperPOPs running old server, until final switchover is complete.
Worst Case Fallback Scenario

- Turn off all traffic to POPs that have been converted to new server
- SuperPOP handle global traffic on old server

(not to scale, illustrative only)
Op Reliability: Metrics

- Metrics and Health Checks
  - Queries: protocol, transport, error, rcode, dnssec
  - CPU, Memory, Network
  - Packet queue drops, listening ports, response time, ...
- Store, Search and Dashboarding
  - Elastic, Kibana, Grafana
- Alerting
  - Elastalert, Custom
Op Reliability: Central crash logging

- Crashes can be **hard to reproduce** because of the many variables
- We need detailed information in the case of a crash

backtrace.io gives us:

- excellent C++ support, detailed stack traces
- information about what was in memory
- efficiently sent to central server for visualization and searching
Proving Correctness: Tee testing

- There is **no way to fully automate test coverage** of all filter chain combinations
  - some filters are nondeterministic
  - therefore, some differences in results are expected and ok
- The variables of **different geographic regions**
- Tools we used
  - iptables
  - elastic, kibana, packetbeat
  - custom report scripts (python)
Tee testing overview

- traffic
- iptables
- nsoned
- trexd
- Elastic
- Packet Capture
- Core
- POP
- control
- test
- Packet Capture
- tooling
- Python
• **Tee testing**

**Control Server** *(nsoned) 10.10.251.51*

```
-A PREROUTING -p udp -m udp --dport 53 -j TEE --gateway 10.10.251.52
-A PREROUTING -p tcp -m tcp --dport 53 -j TEE --gateway 10.10.251.52
```

**Test Server** *(trexd) 10.10.251.52*

```
# this is to forward TEEing traffic to bitbucket host (.53) so it doesn't go back to clients and make
# them see two responses. we can't just drop this traffic because we need to capture it in
# packetbeat too

/sbin/ip route add default via 10.10.251.53 dev em1 table 10000
/sbin/ip rule add from 10.10.251.0/24 lookup 10000
```
**Tee testing**
Our deploy strategy in action

- Early in rollout, we had a report of minor geotargeting misdirection in a region where we had deployed new code
- The procedure:
  - down the BGP announcement in POP, traffic is redirected automatically
  - identify and fix the issue (EDNS client subnet)
  - deploy, tee-test, examine results, gain confidence
  - bring the POP back up
• Rollout Recap

• Understanding **global distribution of traffic**
• **Start with smaller POPs**, do intensive **correctness testing**
• But can’t go strict low to high traffic: **must mix in different regions**
• Collect, visualize, alert on **detailed operational metrics**
• Ability to **canary** at granularity of individual nodes across POPs
• **Increase percentage** of traffic to new server as we gain confidence
1. Split traffic in SuperPOPs
2. Burn in (1w)
3. Turn down traffic to old servers, run Supers at half capacity in standby
4. Burn in (1w)
5. Reprovision old, bring up full capacity with trex 😎
Takeaways
Final Results

- Blew past our 10x goal for **20-40x QPS perf increase**
- **Smother operations** as we’ve scaled platform
- Online DNSSEC signing in place
- **No downtime** during deployment
- 15 months total
Thank you.