Rebuilding the airplane in-flight... (safely)
What is our plane in-flight?
• Modern Intelligent DNS

  • Modern DNS is no longer just a static phone book
  • DNS is the **first entry point** to any online application
  • If it's down or slow, so is your application
  • It’s a great time to make an **intelligent routing decision**
Our airplane: DNS+Global Traffic Management

- 25 global POPs, more dedicated single tenant
- Anycasted IP space
- US+EU avg latency < 12ms
- Global avg latency < 45ms

- in the critical path of application delivery
- 100% uptime SLA
- tens of billions of queries per day
- custom software stack and platform
So why did the plane need a rebuild?
Why the plane needed a rebuild

- Original platform built in **Python**
  - Scaled horizontally
  - Did smart things with cache locality, local load balancing across processes
- Extreme **platform growth** since start in 2014
  - Platform pressures
  - Scaling economics: horizontal was too wide
- Need **DNSSEC** with online signing
- Forecast serving tens of billions of queries a day globally, each one of which could include **custom filters** and/or DNSSEC signing
Ok, so rewrite. Requirements?

- Something we could **base our future tech on** for years to come
- **Improve performance** (QPS throughput) at least **10x**
- Support **DNSSEC** with online signing
- Maintain (reimplement) **full custom feature set** like filter chains, record level traffic statistics, near instant propagation times, ALIAS, Pulsar
- Ability to rollout across platform with **no downtime**
- Limit faults, **verify correctness** (esp. of intelligent features)
Our Plan

- **Ground-up rewrite** in a more performant language
- Optimize for **features, scale, reliability**
- **Team of 3-5** senior engineers
- 6 month project in **two phases**
- **Safe deployment** plan
Execution
A server is born: trex
• **Beginnings**
  
  • Choosing a **new language**
    - compiled language: C/C++ or go?
    - benchmarks and clean interface to OS told us **C++14/17**
  
  • Choosing a **server framework**: single thread, async io
    - looked at seastar.io, wangle, others
    - went with libuv + uvw
  
  • Get **code base** and **build pipeline** setup

Jan 2017
Trex Server Architecture
Flamethrower

- Complementary tool for performance and functional testing
- Originally built to ensure we hit our performance target and prove software stack (same as trex)
- Can push ~70k-100k QPS per core
- Can simulate various types of traffic, including DoS attacks

Jan 2017
Flamethrower

```
aweyrick@twisted:/workspace/flamethrower/cmake-build-relwithdebinfo (master)$ ./flame 192.168.0.54 -l 10
flaming target "192.168.0.54" (192.168.0.54) on port 53 with 10 concurrent generators, each sending 10 queries every 1ms on protocol udp
query generator [static] contains 1 record(s)
  0.954615s: send: 70100, avg send: 70100, recvv: 19186, avg recv: 19186, min/avg/max resp: 4.52353/166.805/234.571ms, in flight: 50906, timeouts: 0
stoping, waiting up to 3s for in flight to finish...
12.9827s: send: 0, avg send: 71827, avg recvv: 4724, avg recv: 20324, min/avg/max resp: 138.394/0/239.662ms, in flight: 0, timeouts: 203864
```

---
run id : 931428e0190bc542
runtime : 12.9827 s
total sent : 718300
total rcvd : 207982
min resp : 4.52353 ms
avg resp : 186.196 ms
max resp : 239.662 ms
avg r qps : 20324
avg s qps : 71827
avg pkt : 37 bytes
tcp conn. : 0
timeouts : 518318 (71.0452%)
bad recv : 0
net errors : 0
responses : NOERROR: 207982
Flamethrower

 NS1.
2 phased release plan

- Why?
  - Get MVP out fast, then iterate
  - Gain operational experience
  - Prove the technology chain
- Phase the first: front-end proxy, 3 mo, out in 3-4 POPs
- Phase the second: full replacement, 3-4 mo, sitewide deploy
First phase

Jan 2017
First phase results

- **Success**, deployed to 3 POPs on time
- Met goals of **proving tech stack** and **laying foundation**
- Learned some **operational lessons**
  - e.g. number of processes, throughput for different types of real traffic
- But... **limited practical benefit** on the platform as a proxy

Apr 2017
• Second phase
• **Road to Phase 2**

  - **Full, optimized query pipeline**, better RFC compliance
  - Port all **filter chains** and other special sauce
  - **DNSSEC** online signing
  - Plan for **full site deployment**

Apr 2017
Second phase results

- Code execution **took 6-7 months** instead of 3-4
- No major architectural setbacks; devil, as usual, in the **details**
- **More unknowns** and variables in this phase
- Bringing on devs with no C++ background has **ramp up time**
- Summer conferences, PTO and vacations affected schedule
- On to the deploy...

Nov 2017
Safe, Sitewide Deploy
**Deploy Strategy**

- **Layers** of protection and rollback
- Slow **roll out period** targeting 2-3 months
  - Utilize our **anycasted** edge network for fault isolation
  - Canary deploys, sending traffic to a percentage of POPs and total cores
- Prove **operational reliability**
  - Detailed operational metrics and central exception logging
- Prove **correctness**
  - Tee testing to compare traffic output ensuring correctness

Dec 2017
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)
• Walled Castle: SuperPOPs

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

(not to scale, illustrative only)
• **Maintenance and Deployments**

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

(Not to scale, illustrative only)

- Usually, all are up providing best performance
• Fault Isolation and Rollback

• Anycast gives us **natural fault isolation** by being able to turn down a deployed POP in case of problems

• Traffic is **redistributed automatically** to remaining POPs running old server

• In worst case scenarios, always able to **fall back to SuperPOPs** running old server software (until final switchover complete)
Central crash logging

- With thousands of processes handling tens of thousands of QPS per process and running all around the world, 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

Jan 2018  
Mar 2018
• **Tee testing for Correctness**
  
  • 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, ELK, packetbeat, custom report scripts (python)

Jan 2018  |  Mar 2018
• **Tee testing**

![Diagram of Tee testing process with IP Tables, Tnsnr, Trexd, and ELK integration.](image_url)
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 (trex) 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 a duplicate response. 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

Jan 2018

Mar 2018
Tee testing
POP rollout

- Had to understand **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
Finishing the deploy: SuperPOP Cutover

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 😎

Mar 2018
• Final Results

• Blew past our 10x goal for **20-40x QPS perf increase**
• **Smoother operations** as we ingest sudden spikes in smaller POPs
• Online **DNSSEC** signing in place
• **No downtime**, outage or customer disruption during deployment
• 15 months total
Lessons and tips

- Phased plan was a win: make it real soon, get it out and iterate
- Canary deploys, know what metrics to watch
- Have layers of fall back plans
- Tools for comparing packet-level results were a huge win

Shout out to our amazing team of engineers @NS1 who made this happen
Thank you.