Google Cloud

Google SDN Peering: An Early Engagement Case Study

Murali Suriar, msuriar@google.com
On behalf of Google Technical Infrastructure and Network Infrastructure SRE

August 30, 2017

Who am I?

- Murali Suriar
- Seven years at Google*
  - Network Engineer, Dublin
  - SRE, London
    - Initially working on proxies/load balancing
    - Currently running SDN control systems
- @msuriar on Github, Twitter, IRC

* = minus a brief stint on a boat

Today's talk

- What is SDN?
- A brief history of SDN at Google
- An overview of Espresso (SDN internet peering)
- SRE early engagement with the Espresso dev team

What is SDN?

Traditional networking

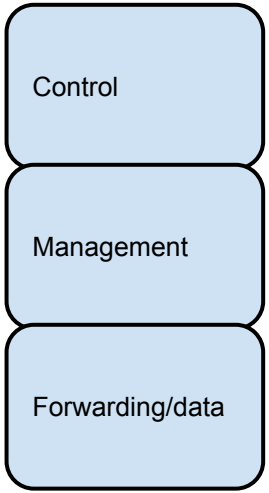
- Common protocols and standards (mostly).
- Proprietary/vertically integrated implementations.

An aside - why hardware?

- IP networking all about **packets per second** (pps).
- Weird standards.

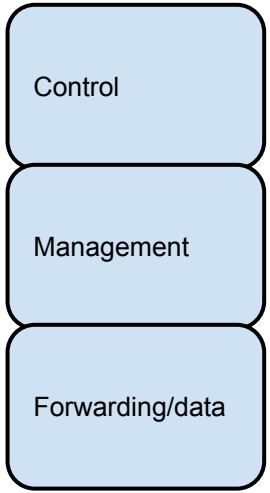
Planes of a switch/router

Planes of a switch/router ("swouter")



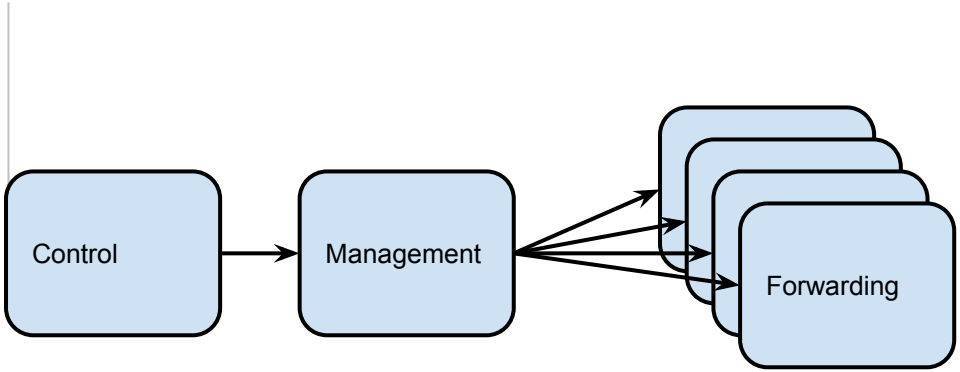
Planes of a switch/router ("swouter")

- Control plane scales with protocol/network complexity.
- Network vendors use long-term supported hardware.
- Long depreciation cycles lead to underpowered control plane.



The dream of SDN

- Create standard for programming the forwarding plane.
- Separate control plane from network devices.



Complexities of SDN

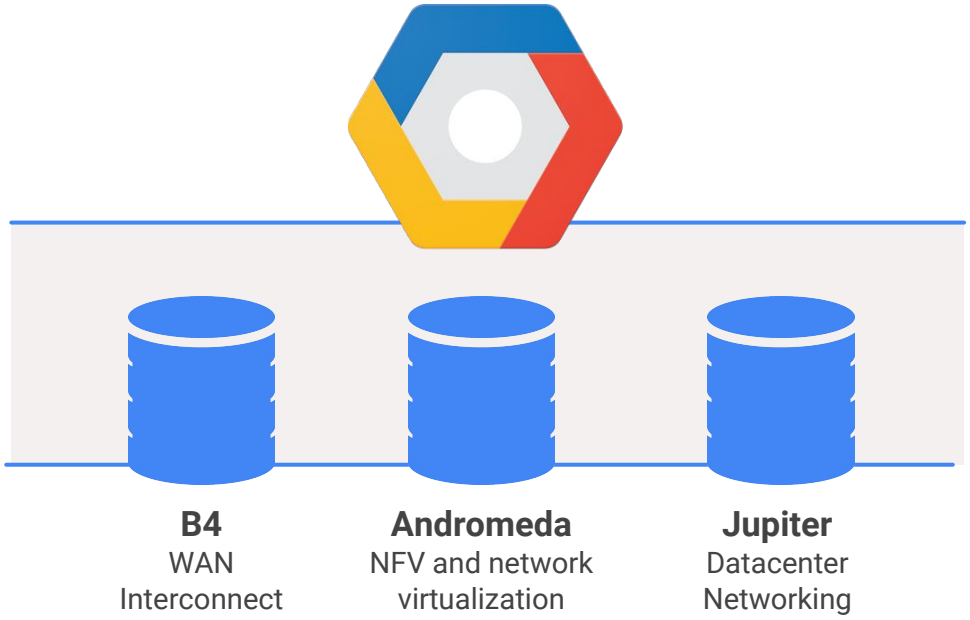
- Need a new network to connect control and data plane together.
- Network engineers need to learn about running binaries and managing machines.
- Or sysadmins/SREs need to learn about networking.

New failure modes of SDN

- Less shared fate between control plane and data plane.
- Single controller outage has (potentially) large impact on data plane.
- Increased latency in reacting to some classes of failures.

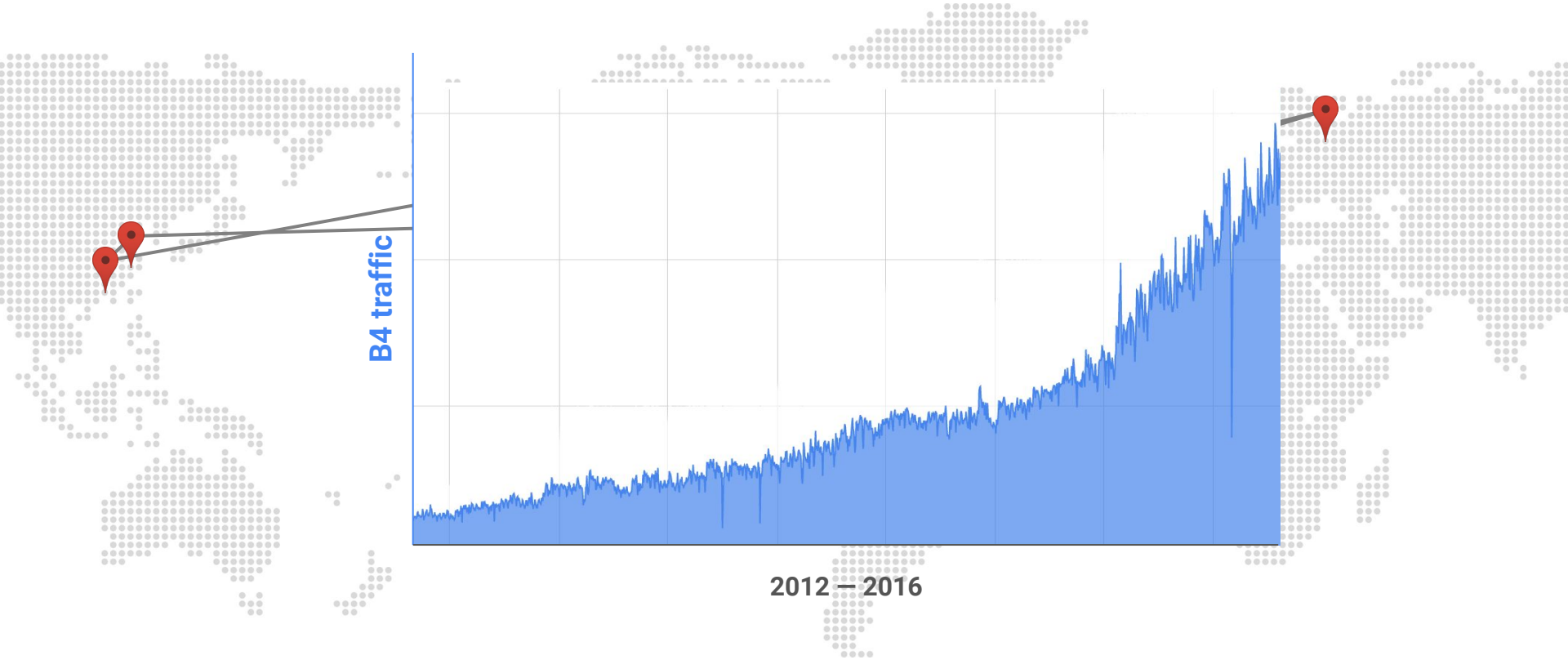
A brief history of SDN at Google

The Pillars of SDN @ Google



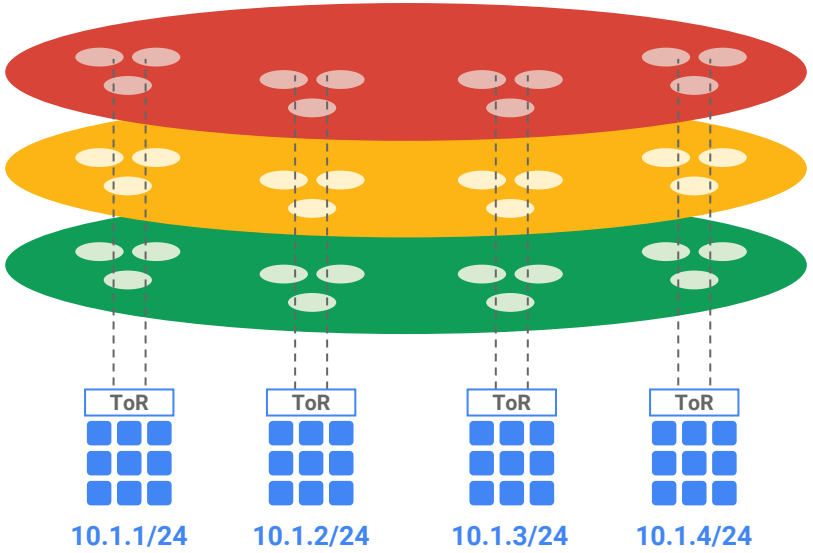
B4: Google's Software Defined WAN

B4: From Copy Network to Business Critical



Andromeda

Google Infrastructure Services



VNET: 10.1.1/24

VNET: 192.168.32/24

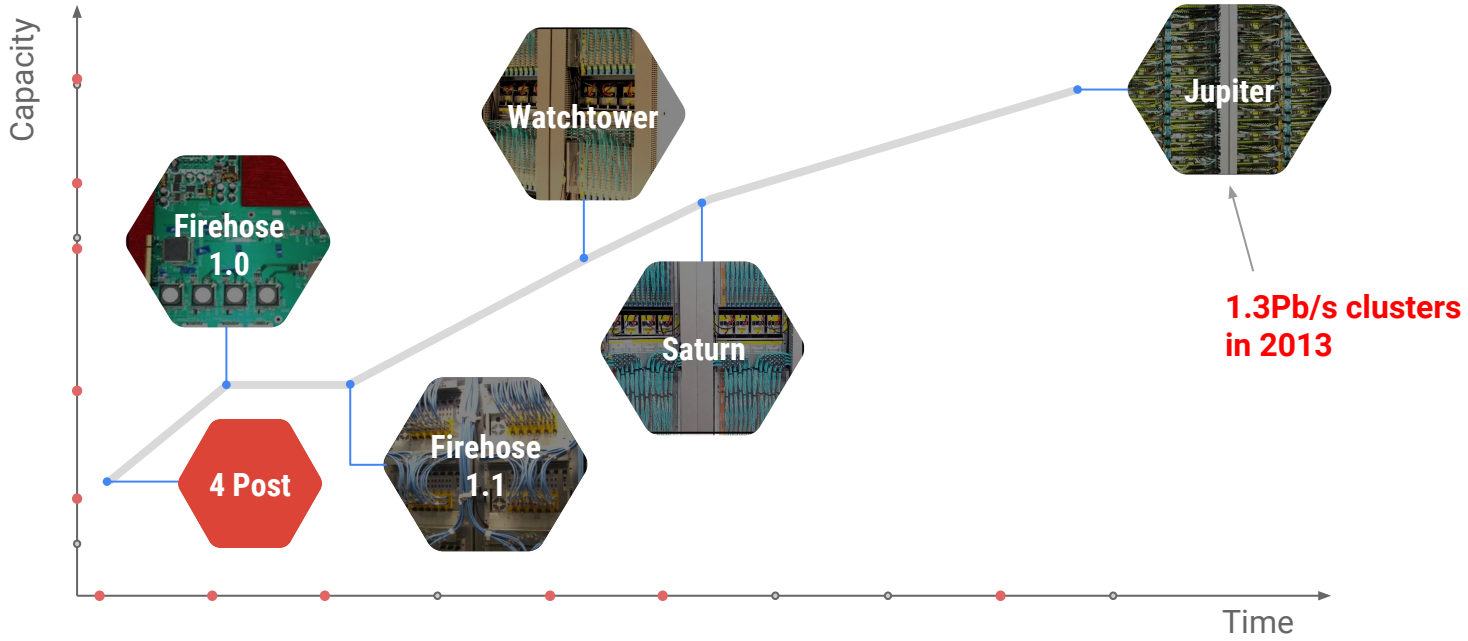
VNET: 5.4/16

Internal Network

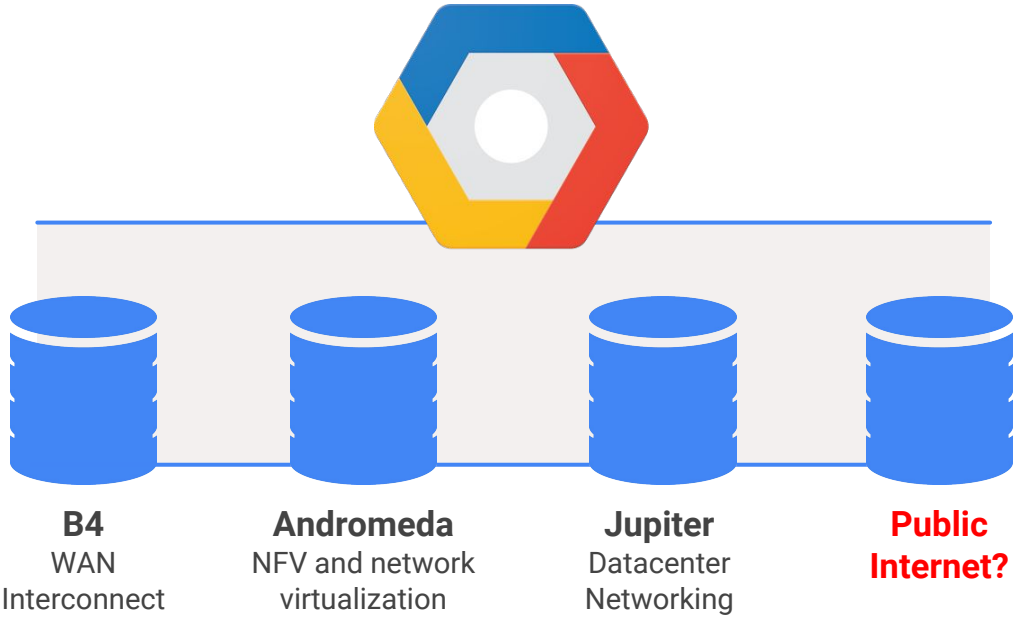
- Load Balancing
- DoS
- ACLs
- VPN
- NFV

Google Datacenter Network Innovation

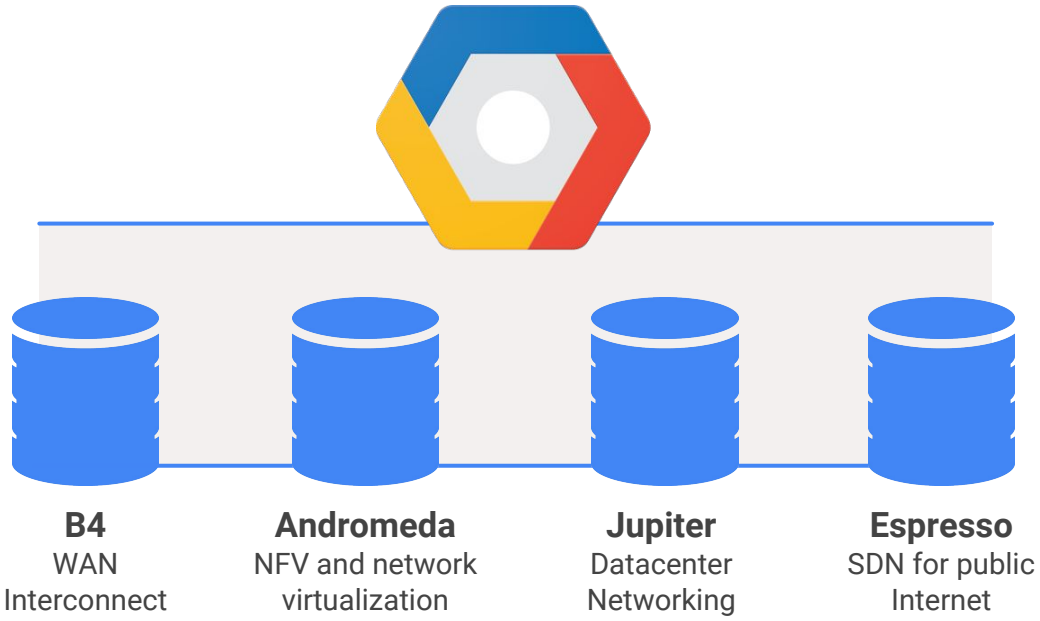
And hardware scale that we could not buy



The Pillars of SDN @ Google

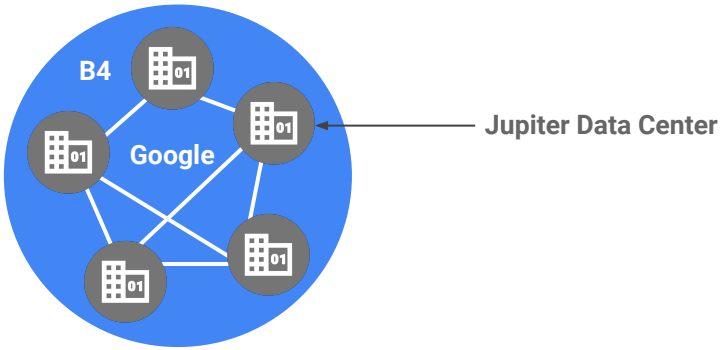


The Pillars of SDN @ Google

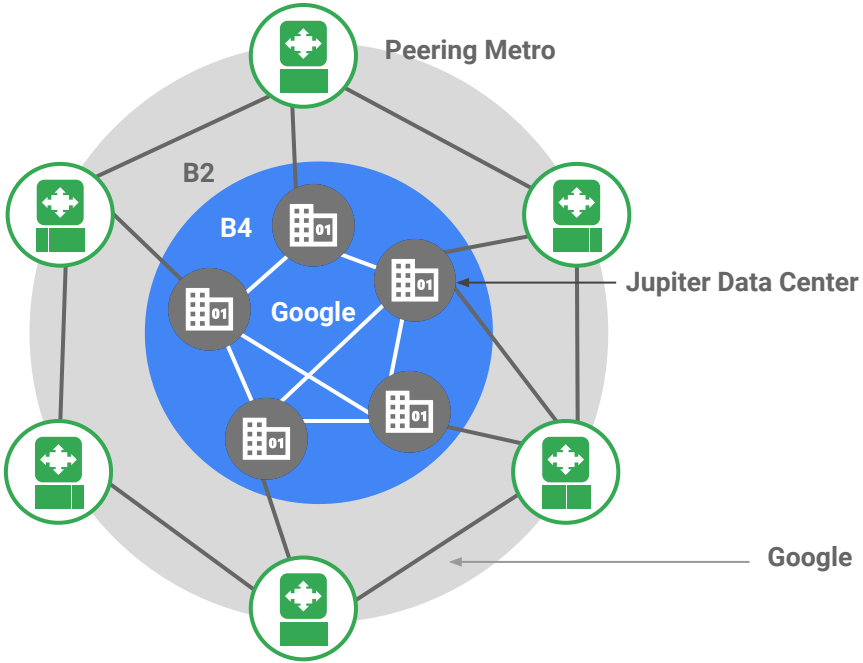


Enter Espresso

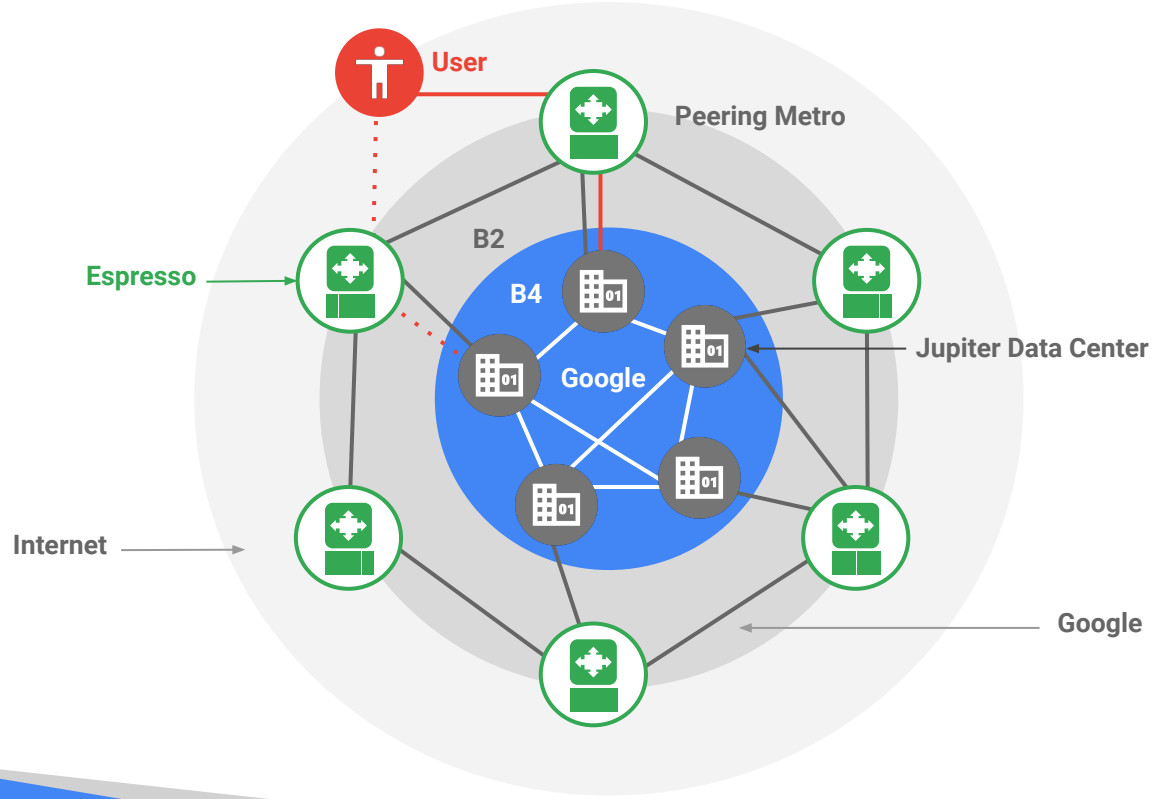
Espresso in Context



Espresso in Context



Espresso in Context

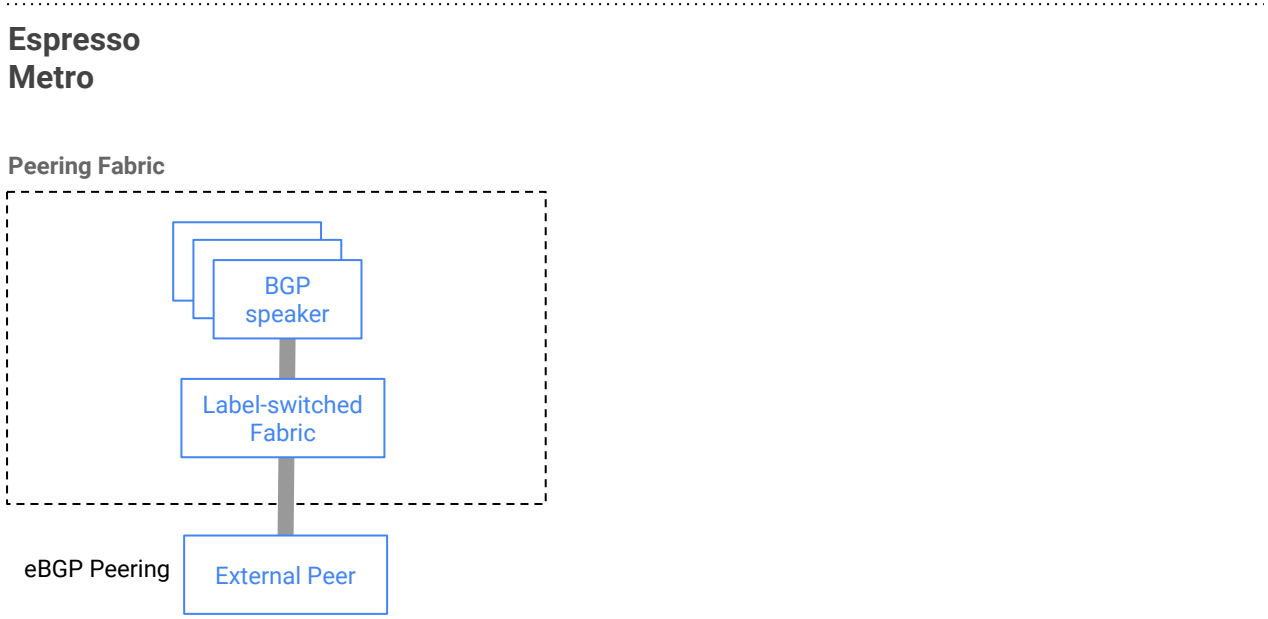


Espresso: Before and After

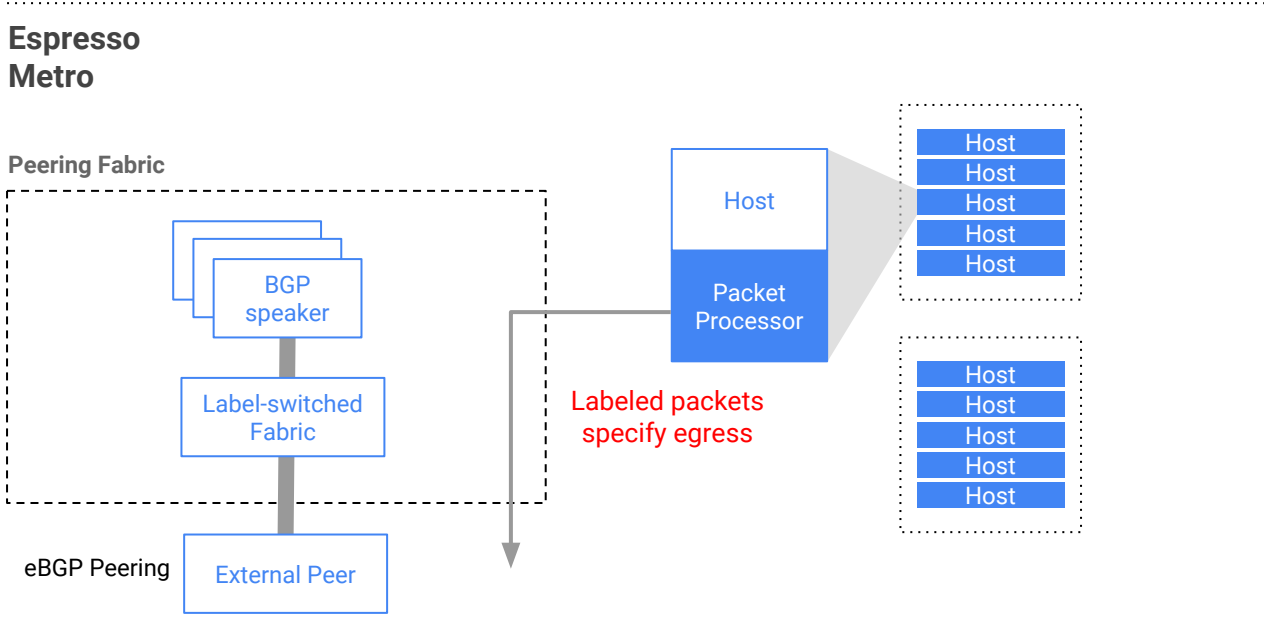
Local view
Connectivity first
Coarse fault recovery

Per-metro and global view
Application signals
Real-time optimization

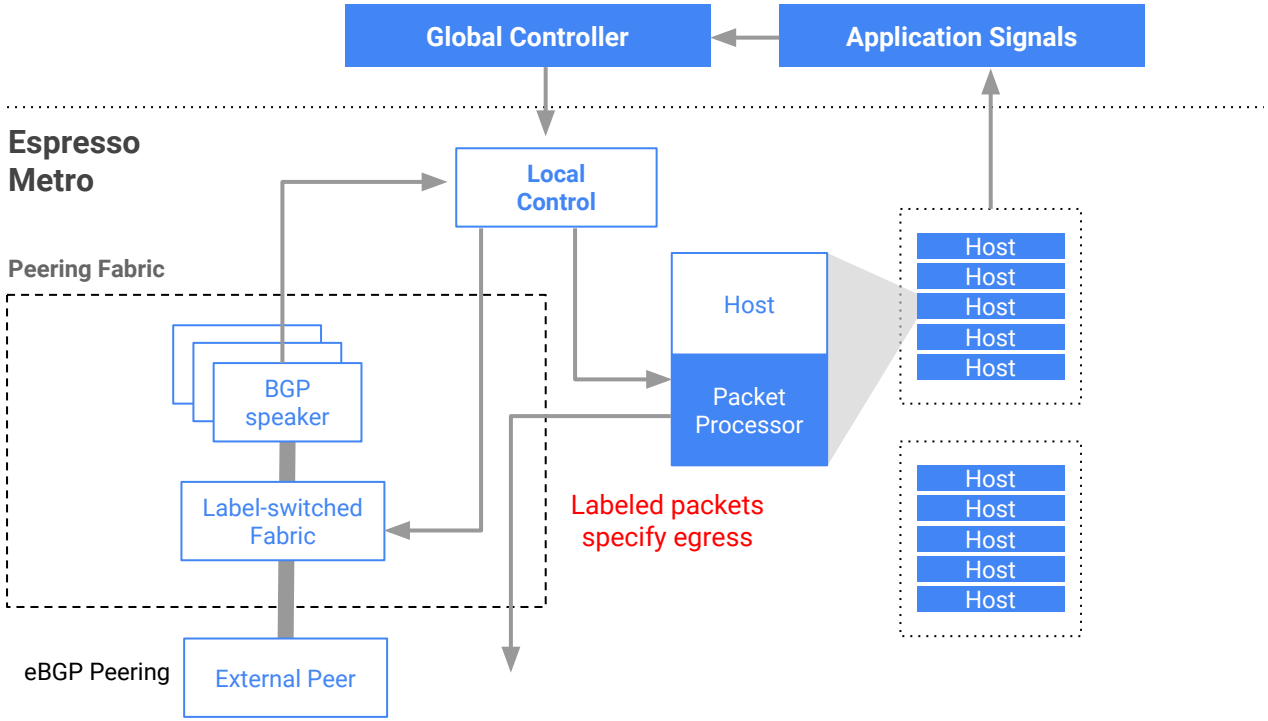
Espresso Architecture Overview



Espresso Architecture Overview



Espresso Architecture Overview



SRE for Espresso

Complexities of Espresso

- Large set of distributed systems.
- Many teams, different skill sets.
- Massive, top to bottom change.
- How do we contain and direct all of this so we make progress?

Espresso team

- Cross functional team
  - Network engineers
  - SREs
  - Developers
  - Testers
  - ...

Espresso team

- Responsible for supporting Espresso from inception to production.
- Set up testing infrastructure.
- Set up job control, monitoring.
- Oncall when Espresso shipped its first bytes.
- Eventually spun down and handed off oncall to permanent teams.

Test/release infrastructure

- Unit tests on **everything**.
- Some software integration tests.
- Automated hardware integration tests.
- CD pipeline cutting a release every night from latest green commit and deploying to hardware testbeds.

Production environment

- Reused/adapted standard building blocks.
  - Borg
  - Chubby
  - ~~Prometheus~~Borgmon
- Had a post lab, prod-parallel testbed which paged Espresso oncall.

"I have a question…"

"Do you know how to let a Borg job SSH into a production machine?"

"Yes. I'm not going to tell you how, though. What are you trying to do?"

(SSH is almost never used for system to system communication at Google; we prefer RPCs.)

"I have a question…"

"I want to save some binary data to disk, then log in, copy it off, and then get it into Dremel."

"So… you want to save some structured (ProtoBuf?) logs into Dremel."

"Yes."

(It turns out Google has an existing toolkit to solve precisely this problem.)

Monitoring/alerting

- Lots of possible points of failure:
  - Peering Fabric.
  - Packet processing on hosts.
  - Software (Local controller, BGP speakers).
  - Global control plane.
- How to tell what's broken?

Monitoring/alerting

- Lots of possible points of failure
  - Peering Fabric.
  - Packet processing on hosts.
  - Software (Local controller, BGP speakers).
  - Global control plane.
- How to tell what's broken?

Monitoring/alerting

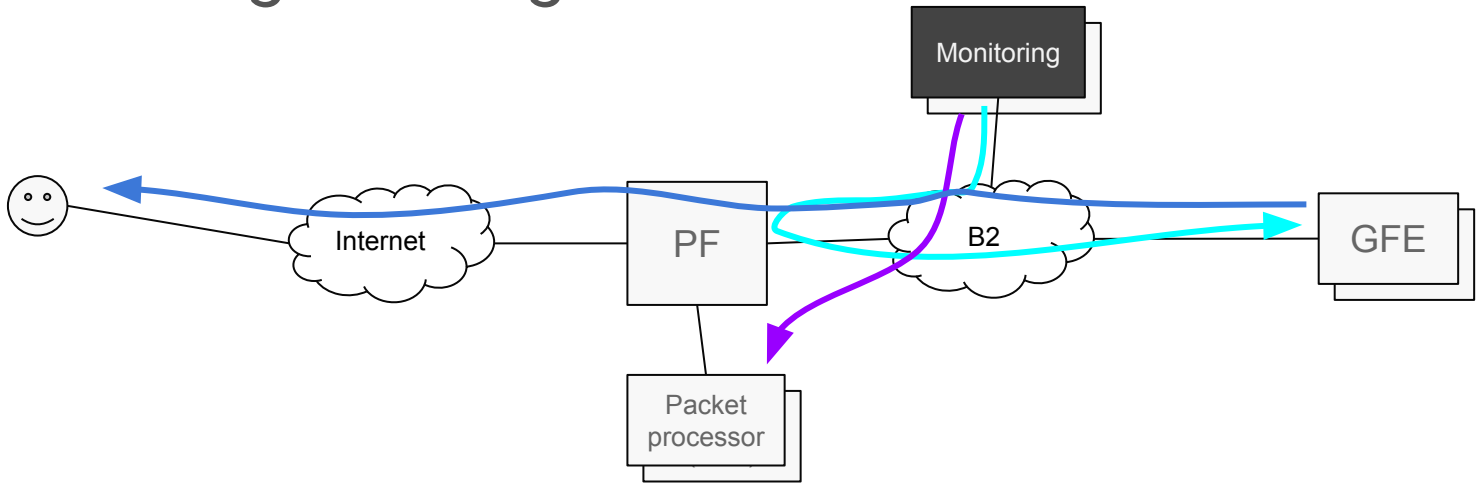
"Network devices have counters everywhere. If we page on the drop counters, that'll catch all the failures we see with traditional peering devices?"

"Oooor… we could build some blackbox probing infrastructure to catch failures which don't show up in counters?"

Monitoring/alerting

- Built a couple of high signal, **symptom based alerts**
  - Black box prober, doing end to end test of control- and dataplane.
- Used lots of whitebox telemetry to help point to root cause.
  - ALL THE GRAPHS.

Monitoring/alerting



1. Blackbox realtime monitoring of PF availability + encap + decap + GFE reachability.
2. Greybox realtime monitoring of pocket processor ACL: decap + ACL-is-blocking + ACL-is-permitting.
3. Passive loss/blackhole monitoring.

Introspection

- Alerting/monitoring tells you **something** is broken.
- How do find out what exactly is causing you to be paged?

Introspection tools

- Google has standard HTTP endpoints for debugging.
  - "Show me the important things about this binary."
  - "Packet processor, what do you know about 192.0.2.1?"
- Custom traceroute-like tools for debugging dataplane.

What broke?

- Most common failure mode: control plane breakage.
- Example: Local controller OOM on new version.
  - No traffic impact. (Fail static.)
  - Caught in first production canary.
  - Added regression test.

What broke?

- SDN management.
- Example: accidentally disabled non-SSH access to Peering Fabrics.
  - No traffic impact. (Fail static)
  - Used SSH access to restore SDN management.
  - Added more conservative canarying for device management changes.

Comprehensibility

- Complex system needed an architecture diagram.
- Espresso architecture doc has:
  - All components.
  - What talked to what.
  - Links to individual design docs.
  - (Later) Who was oncall for what.

Oncall

- Everyone in Espresso team in the oncall rotation:
  - SREs.
  - Developers.
  - Network engineers.
- Some people never oncall before.
- Some people already oncall for other stuff.
- Needed to account for all of this in oncall practices.

Oncall

- Initially Espresso team oncall for all Espresso deployments.
- Then only for a couple of sites where we were testing new features.
- Eventually spun down and handed off to many existing teams.

Summary

What did early engagement get us?

- Dev familiarity with production.
  - When you're paged by a bug, you fix it faster.
- Broad knowledge across lots of disciplines.
- Significant design changes:
  - Reusing more production infrastructure.
  - Symptom based monitoring.

Lessons learned

- Design for testability.
- Reuse whatever you can.
- System architecture diagrams are great.
- Focus on a few, high signal, symptom based alerts.
- Lots of white box telemetry to aid with root causing.

Thank You!