Taking the Edge off with Espresso

Scale, Reliability and Programmability for Global Internet Peering

KK Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, Victor Lin, Colin Rice, Brian Rogan, Arjun Singh, Bert Tanaka, Manish Verma, Puneet Sood, Mukarram Tariq, Matt Tierney, Dzevad Trumic, Vytautas Valancius, Calvin Ying, Mahesh Kallahalla, Bikash Koley, Amin Vahdat and many others.

Problem Statement

Egress Terabits/sec of traffic to our Internet peers

- High-def video, cloud traffic, etc.

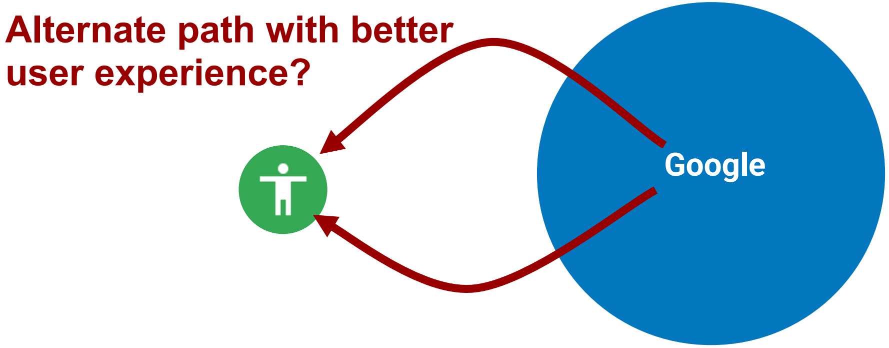
Problem Statement

Egress Terabits/sec of traffic to our Internet peers

- High-def video, cloud traffic, etc.

1. Optimize traffic per-customer and per-application

- e.g., optimal video quality, or differentiated service for cloud
- Problem: Constrained by BGP shortest path and lack of application awareness



Problem Statement

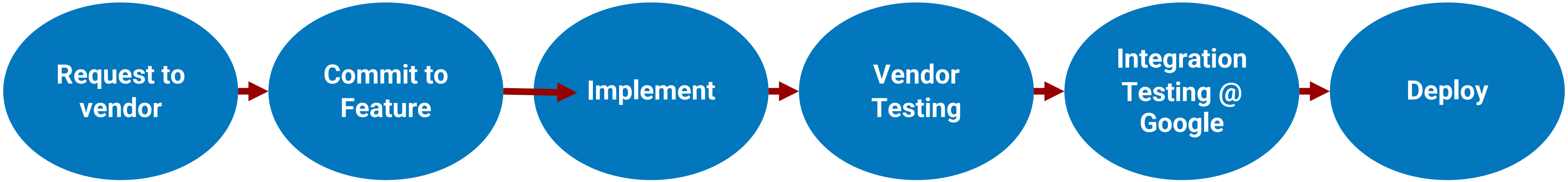
Egress Terabits/sec of traffic to our Internet peers

- High-def video, cloud traffic, etc.

2. Deliver new features quickly

- Problem: router-vendor feature cycles and qualification take many years

Novel L2 VPN?



Espresso: Google's SDN Peering Edge

Our previous experience with SDN

- B4 [SIGCOMM 2013] and Jupiter [SIGCOMM 2015]
- Enable flexible traffic engineering
- Increase feature velocity

SDN is only suited for walled gardens?

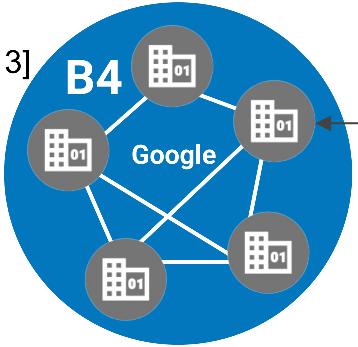
Peering edge requires interoperability with heterogeneous peers.

Agenda

- Problem Statement
- Espresso in Context
- Design Principles
- Architecture Overview
- Results
- Conclusion

Espresso in Context

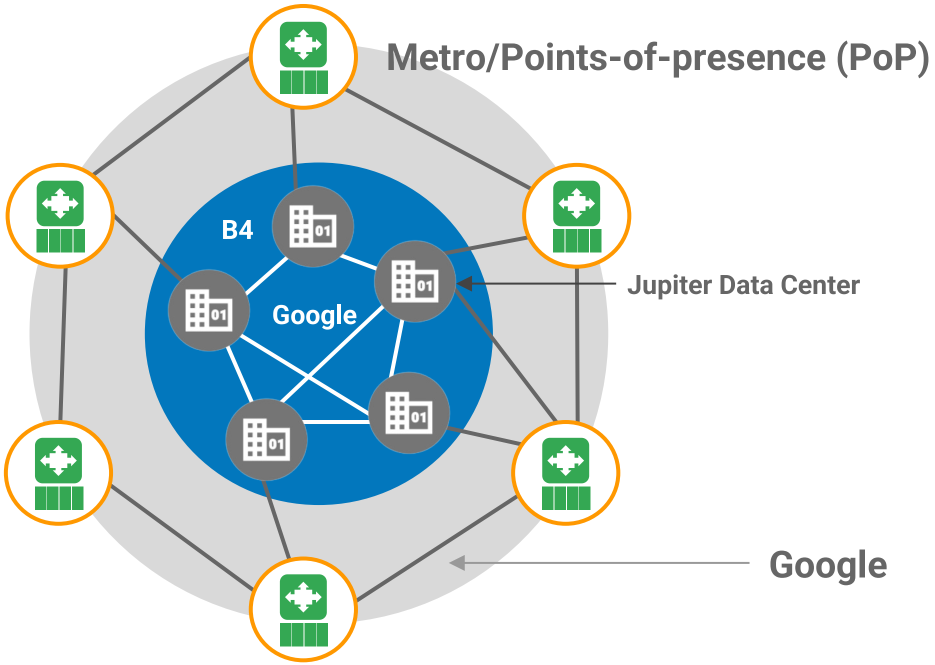
[SIGCOMM 2013]



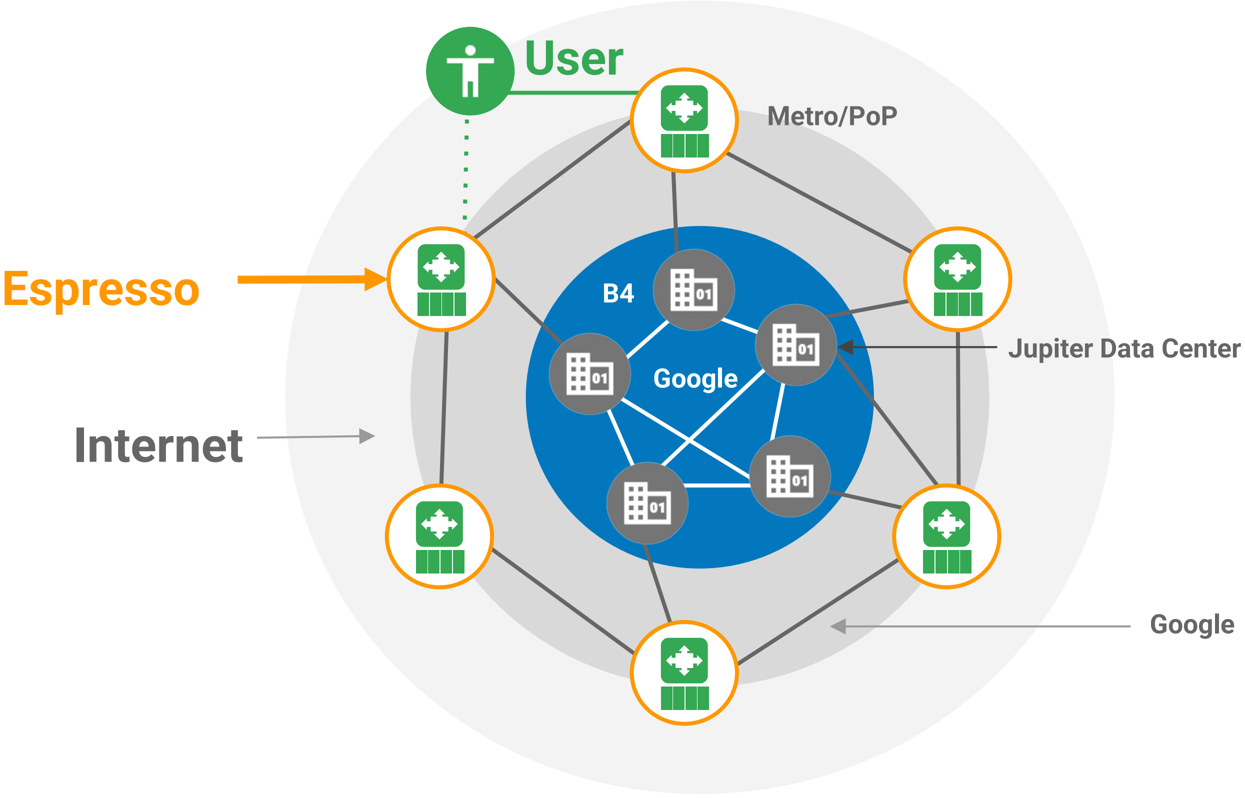
Jupiter Data Center

[SIGCOMM 2015]

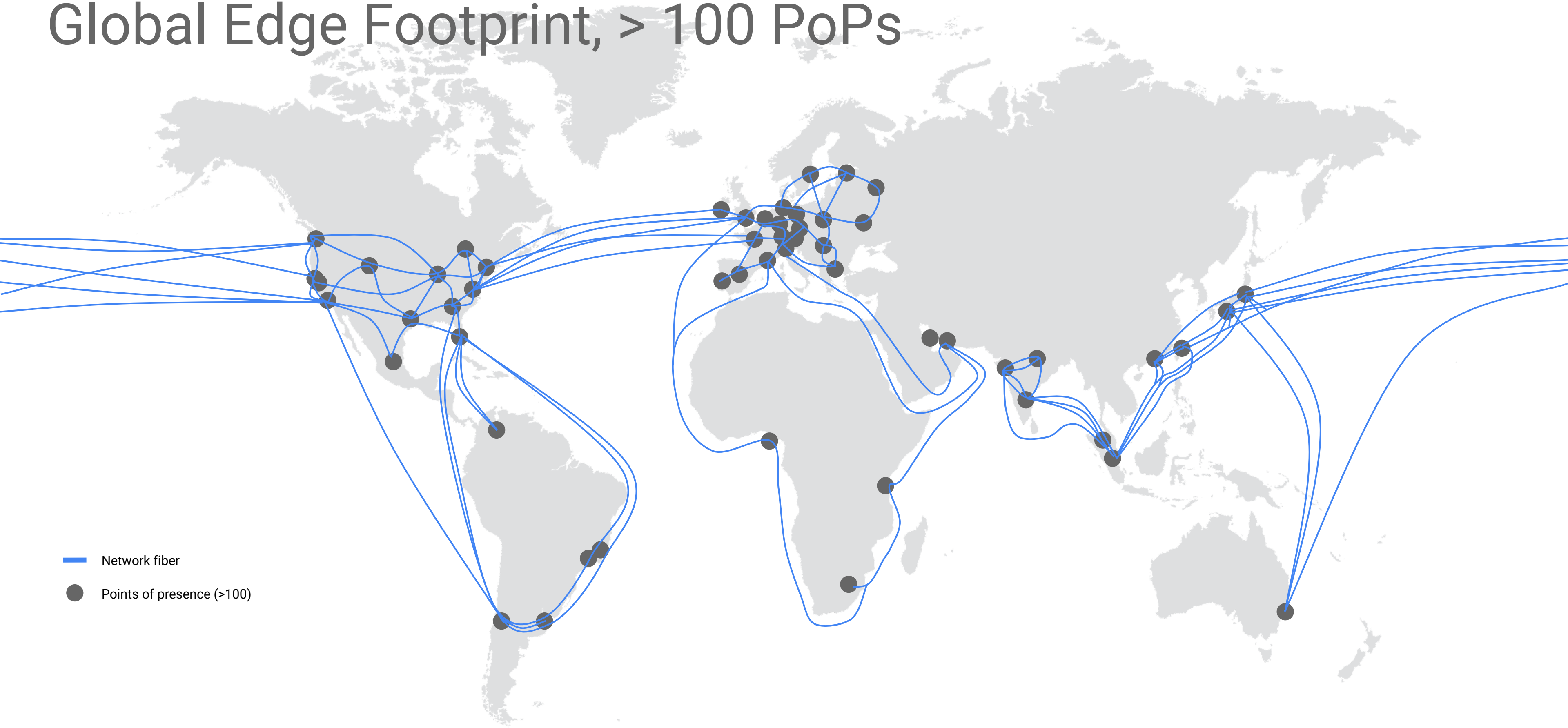
Espresso in Context



Espresso in Context



Global Edge Footprint, > 100 PoPs



Agenda

- Problem Statement
- Espresso in Context
- Design Principles
- Architecture Overview
- Results
- Conclusion

Espresso's Design Principles

1. Hierarchical control plane
    - Global optimization while local control plane provide fast reaction.
2. Fail static
    - Local control plane continues to function without global controller failure.
3. Software programmability
    - Externalize features into software to exploit commodity servers for scale.
4. Testability
5. Manageability

Espresso’s Design Principles

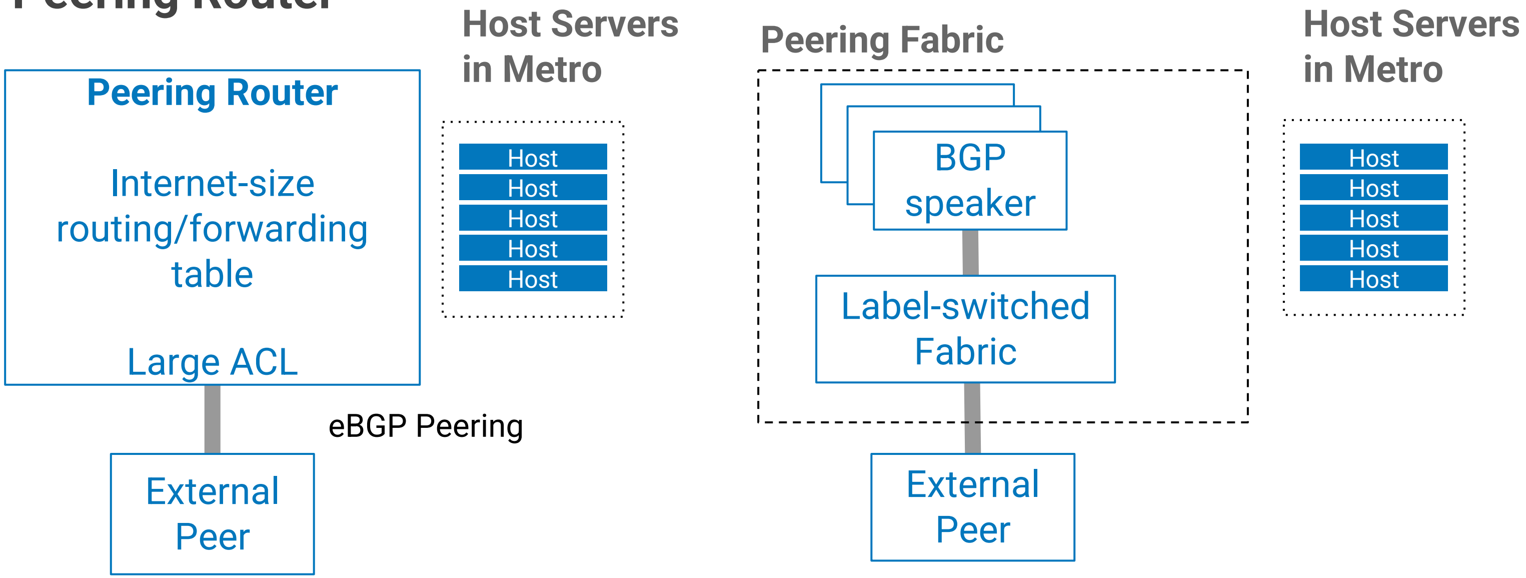
1. **Hierarchical control plane**
    - Global optimization while local control plane provide fast reaction.
2. **Fail static**
    - Local control plane continues to function without global controller failure.
3. **Software programmability**
    - Externalize features into software to exploit commodity servers for scale.
4. Testability
5. Manageability

Hierarchical control plane
Fail static
Software programmability

Architecture: Externalizing BGP

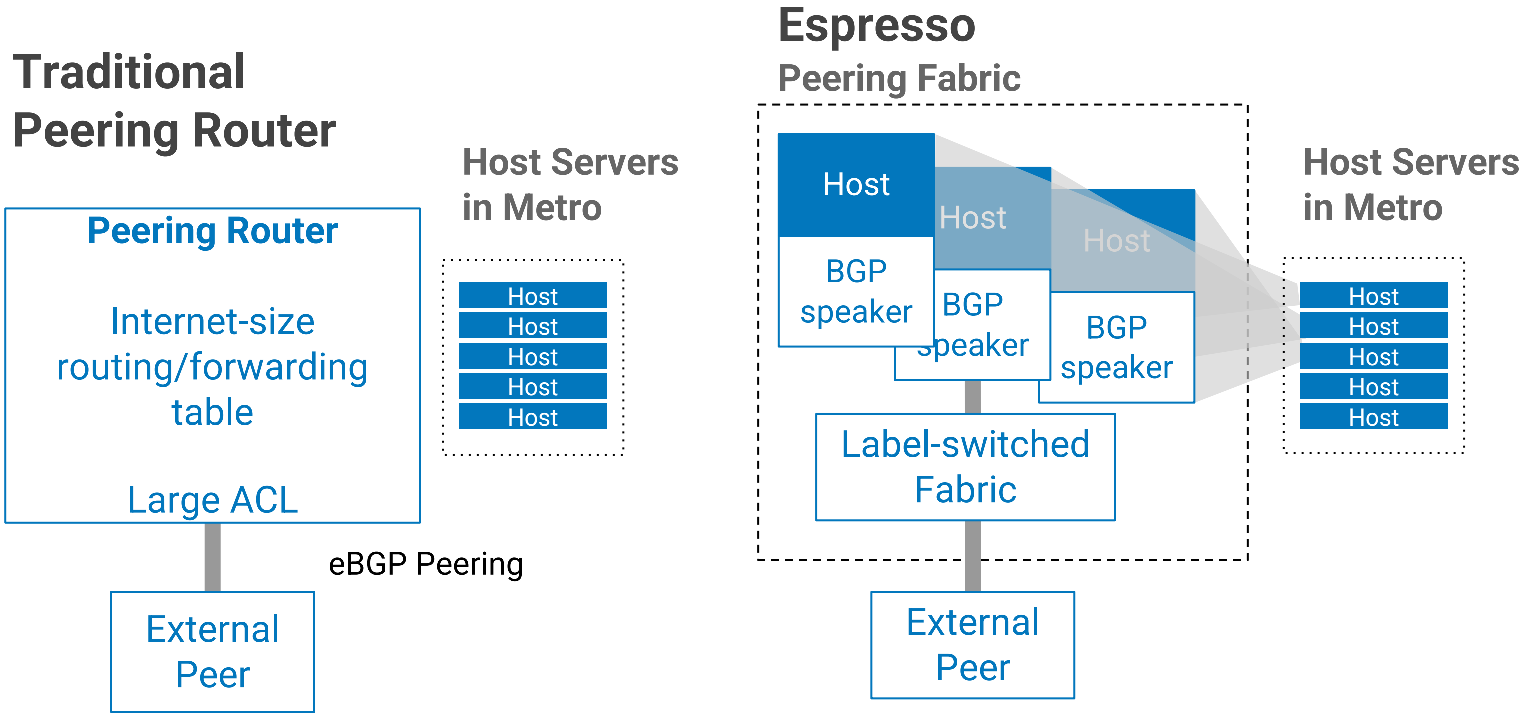
Traditional Peering Router

Espresso



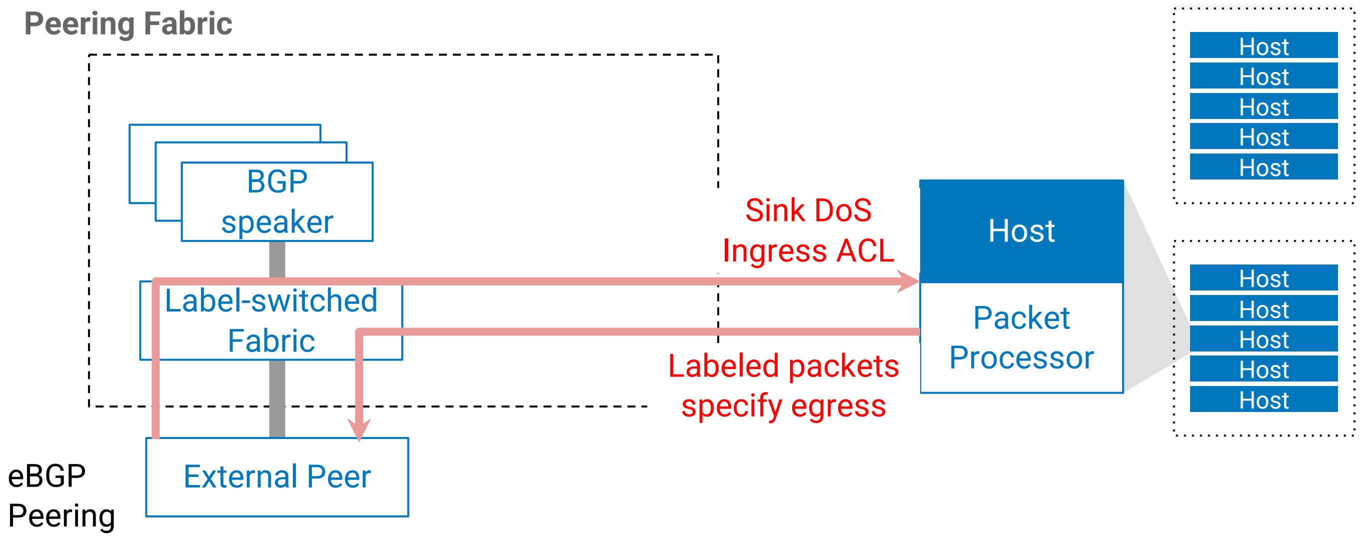
Hierarchical control plane
Fail static
Software programmability

Architecture: Reliability and Scale of BGP



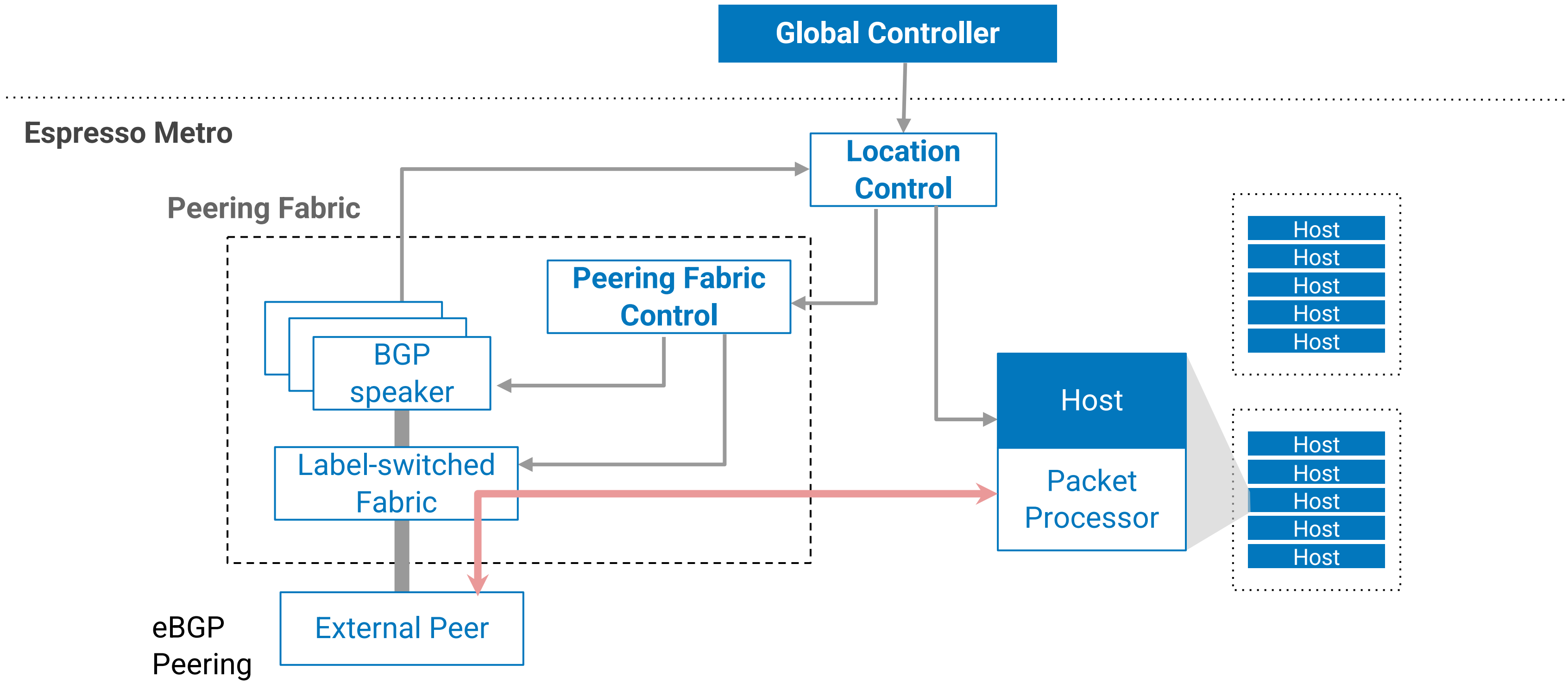
Architecture: Externalize Packet Processing

Host-based packet processor allows flexible packet processing, including ACL and handling of DoS.



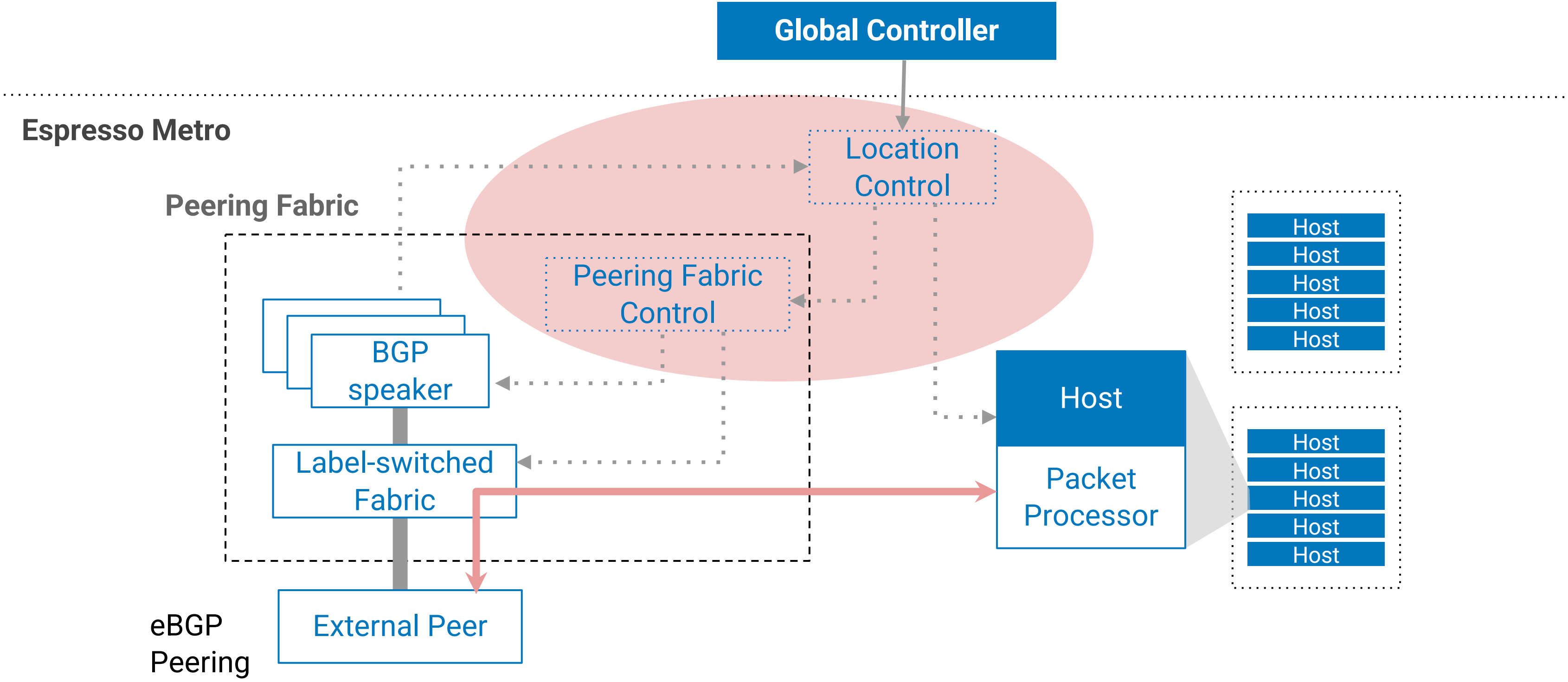
Architecture: Hierarchical Control

Hierarchical control plane
Fail static
Software programmability

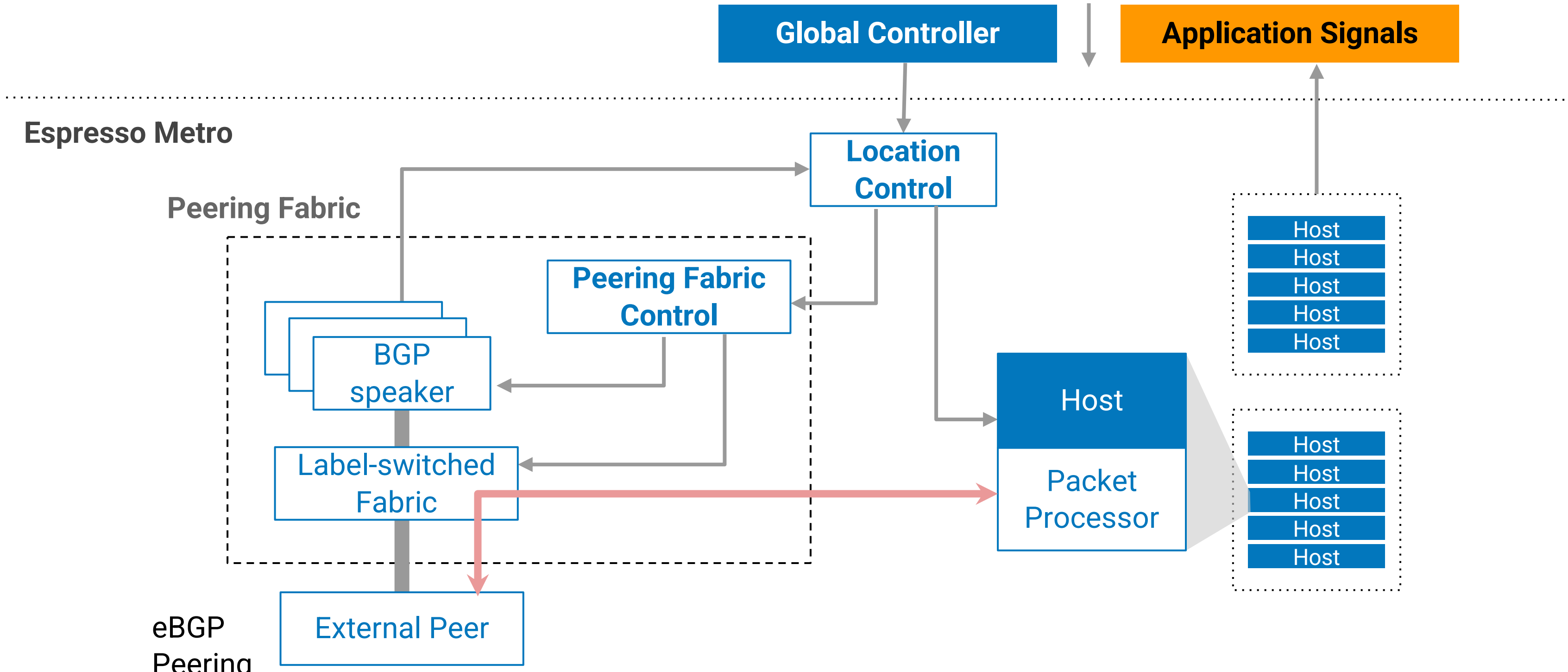


Architecture: Fail Static

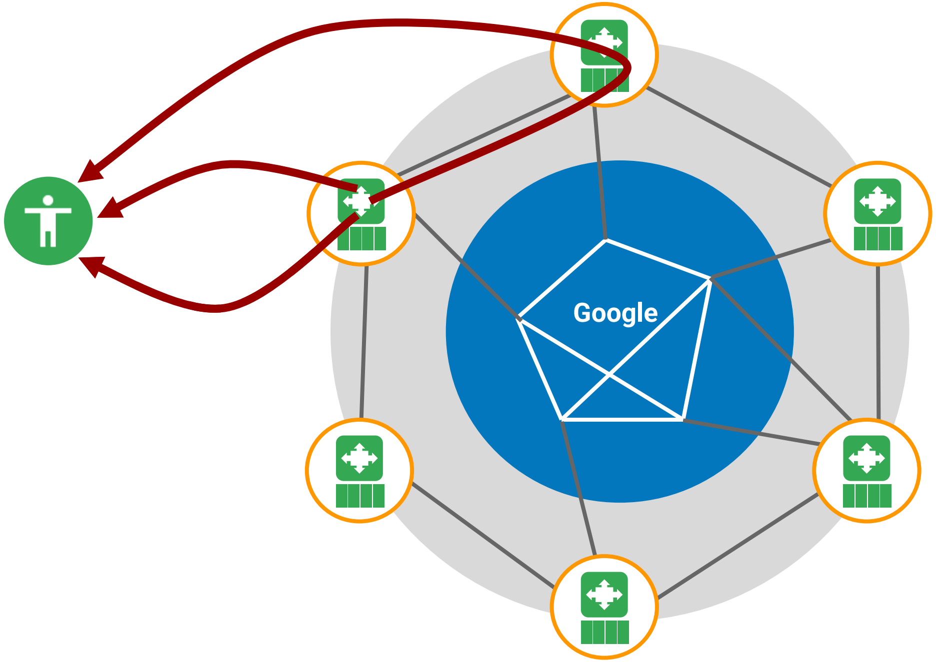
Hierarchical control plane
Fail static
Software programmability



Architecture: Application Aware Routing



Using User's Best Path, not BGP's



- Serve **13% more traffic** than BGP best path in application aware manner.
- Helps capacity-constrained ISPs by overflowing demand to alternate paths within local metro and **also via remote metros.**

Improvements in End User Experience

Client ISP	Change in mean time between rebuffers (MTBR)	Change in Mean Goodput
A	10 → 20 min	2.25 → 4.5 Mbps
B	4.6 → 12.5 min	2.75 → 4.9 Mbps
C	14 → 19 min	3.2 → 4.2 Mbps

Provide significant improvements to end-user experience.

Release Velocity

Component	Average Velocity (days)
Local Controller	11.2
BGP speaker	12.6
Peering Fabric Controller	15.8

> 50× more frequently than with traditional peering routers.

Novel L2 VPN delivered 6× faster via incremental rollout.

Conclusion

~~SDN is only suited for walled gardens..~~

Espresso demonstrates that

- Traditional peering architecture can evolve to exploit SDN
- SDN's value is in flexibility and feature velocity

Conclusion

Local view
Connectivity based optimization
Slow evolution
Costly

Global view
Application signals-based optimization
Rapid deploy-and-iterate
75% Cheaper

What are the hard parts the authors didn’t talk about?

- Difficulties deriving from many moving parts
  - Many more control- and data-plane entities
  - How to synchronize them? How to reconcile the data upon failures and recoveries?
- What else?

Your questions

- To what extent does the need to cooperate and provide interoperability with ISP in the context of edge peering affect or constrain the ability of even an entity as large and prolific as Google that sits squarely in the content provider and infrastructure world from innovating on the edge?
- When the data plane needs to be upgraded or changed, how can we provide availability?
- MPLS switches seem vital to Espresso, so a more detailed explanation in this context would be nice.
- What are the implications of Google’s total ownership of the assignment of service classes to different applications and clients? Are there any different ramifications now that it’s in the domain of edge networking, or is this still similar to their full ownership of their own datacenter networking?

Your questions

- How does the GC get information to make global optimizations? Does it have to use similar strategies to Flow Event Telemetry and LightGaurdian? How do they make sure this happens fast enough?
- How does this affect the overall effectiveness of DDoS protection? What does the "finer resolution" of mitigation in the paper refer to, specifically?
- What are example applications for higher priority traffic? Is it based on latency, such as maybe responding to voice input may need faster response?
- The authors describe the "big red button", an extreme design choice to provide a switch for operators to quickly shut down the system. Are such switches common on industry-scale peering architectures? I imagine that they're controversial given the fallout from using the button, and also the danger of human error in using this button at the wrong time.

Backup Slides

Related Work: EdgeFabric

EdgeFabric	**Espresso**
Relieves congestion in metro/PoP	Global traffic optimization
Peering routers	MPLS-label switches ● Substantial cost reduction ● Feature velocity
Routing based on RIB/FIB	Application-level control over traffic

Hierarchical control plane
Fail static
Software programmability

Architecture: Application Aware Routing