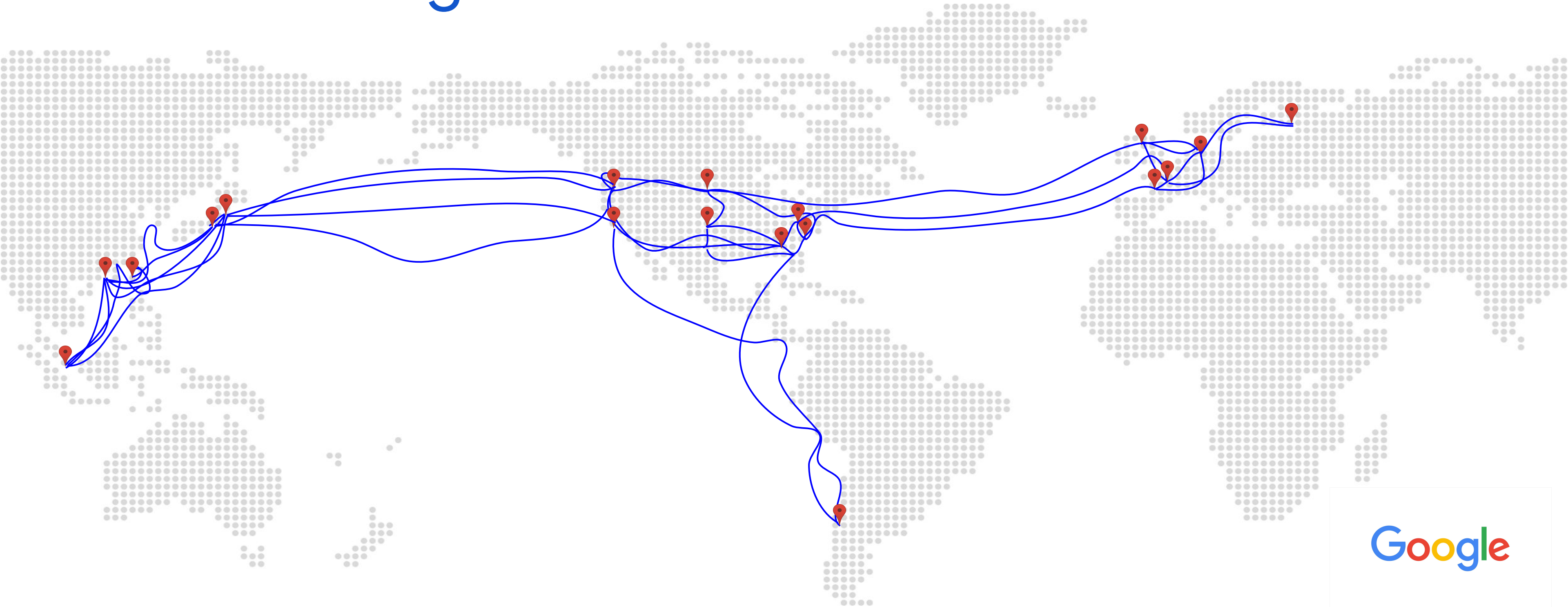
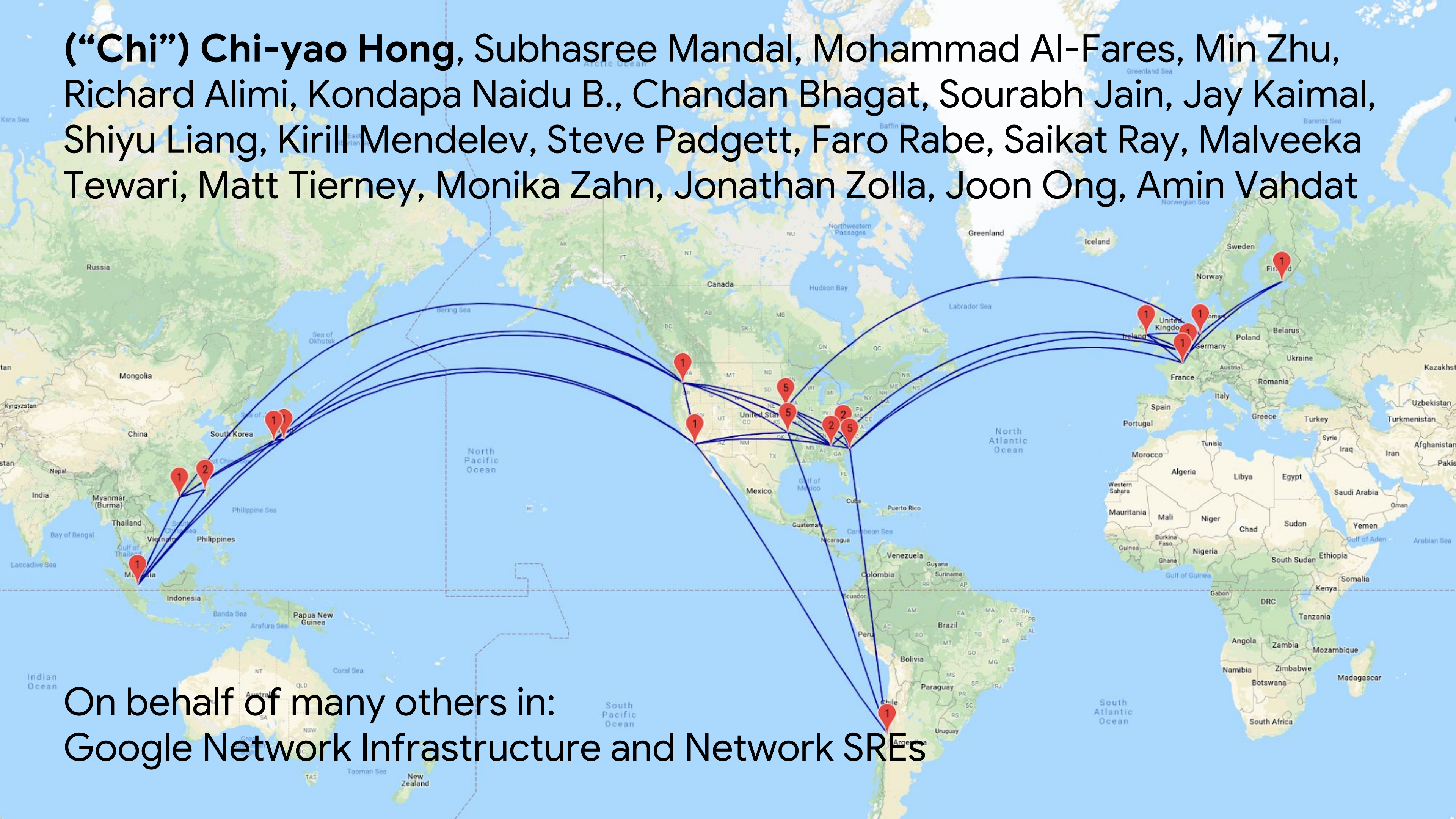
B4 and After: Managing Hierarchy, Partitioning, and Asymmetry for Availability and Scale in Google's Software-Defined WAN

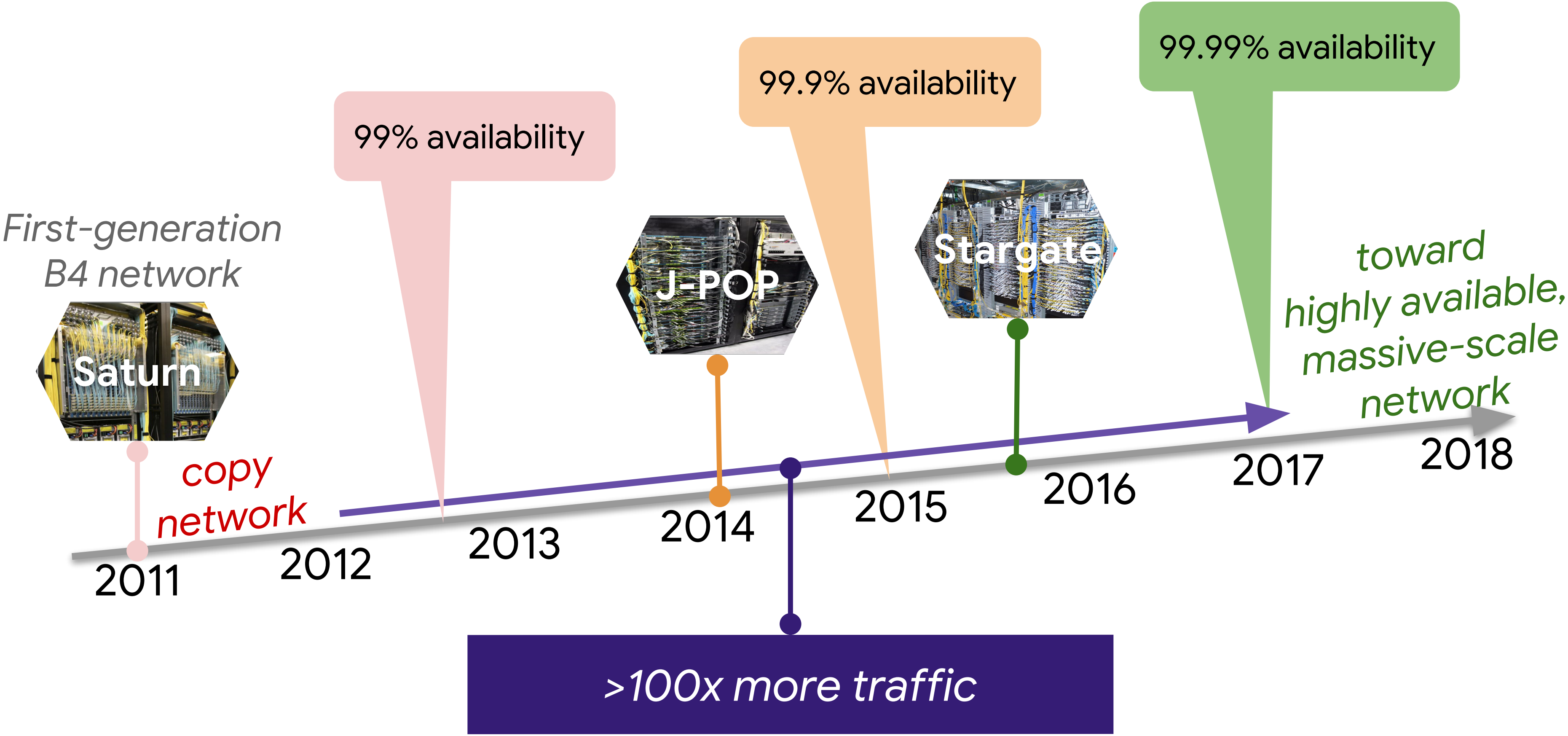


Google

(“Chi”) Chi-yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, Kondapa Naidu B., Chandan Bhagat, Sourabh Jain, Jay Kaimal, Shiyu Liang, Kirill Mendelev, Steve Padgett, Faro Rabe, Saikat Ray, Malveeka Tewari, Matt Tierney, Monika Zahn, Jonathan Zolla, Joon Ong, Amin Vahdat



On behalf of many others in:
Google Network Infrastructure and Network SREs



First-generation B4 network

Saturn

copy network

99% availability

J-POP

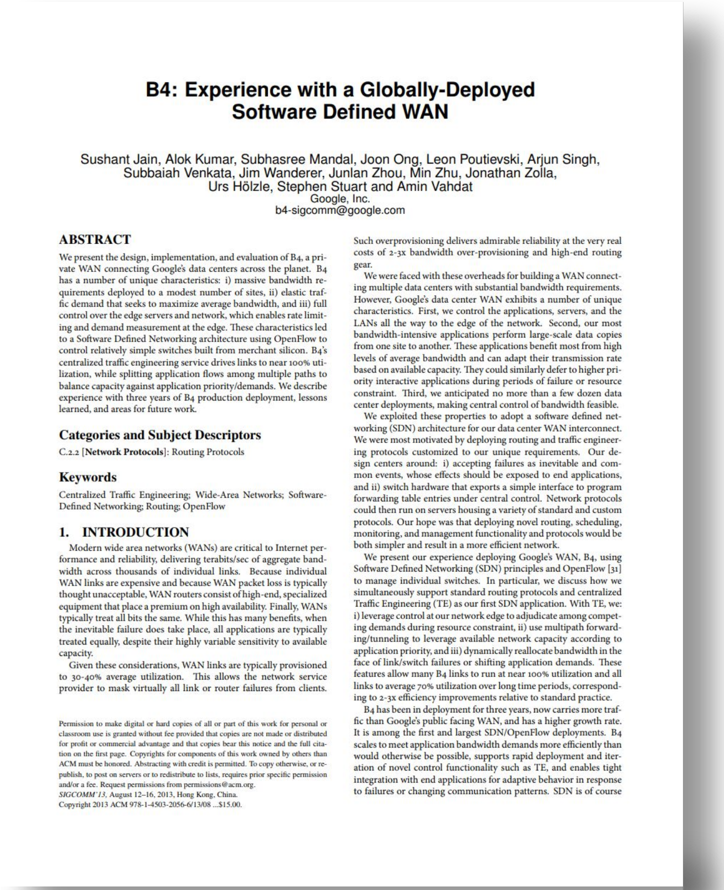
99.9% availability

Stargate

99.99% availability

toward highly available, massive-scale network

>100x more traffic



B4: Experience with a Globally-Deployed Software Defined WAN

Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh,
Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jonathan Zolla,
Urs Hölzle, Stephen Stuart and Amin Vahdat
Google, Inc.
b4-sigcomm@google.com

ABSTRACT

We present the design, implementation, and evaluation of B4, a private WAN connecting Google's data centers across the planet. B4 has a number of unique characteristics: i) massive bandwidth requirements deployed to a modest number of sites, ii) elastic traffic demand that seeks to maximize average bandwidth, and iii) full control over the edge servers and network, which enables rate limiting and demand measurement at the edge. These characteristics led to a Software Defined Networking architecture using OpenFlow to control relatively simple switches built from merchant silicon. B4's centralized traffic engineering service drives links to near 100% utilization, while splitting application flows among multiple paths to balance capacity against application priority/demands. We describe experience with three years of B4 production deployment, lessons learned, and areas for future work.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Routing Protocols

Keywords

Centralized Traffic Engineering; Wide-Area Networks; Software-Defined Networking; Routing; OpenFlow

1. INTRODUCTION

Modern wide area networks (WANs) are critical to Internet performance and reliability, delivering terabits/sec of aggregate bandwidth across thousands of individual links. Because individual WAN links are expensive and because WAN packet loss is typically thought unacceptable, WAN routers consist of high-end, specialized equipment that place a premium on high availability. Finally, WANs typically treat all bits the same. While this has many benefits, when the inevitable failure does take place, all applications are typically treated equally, despite their highly variable sensitivity to available capacity.

Given these considerations, WAN links are typically provisioned to 30-40% average utilization. This allows the network service provider to mask virtually all link or router failures from clients. Such overprovisioning delivers admirable reliability at the very real costs of 2-3x bandwidth over-provisioning and high-end routing gear.

We were faced with these overheads for building a WAN connecting multiple data centers with substantial bandwidth requirements. However, Google's data center WAN exhibits a number of unique characteristics. First, we control the applications, servers, and the LANs all the way to the edge of the network. Second, our most bandwidth-intensive applications perform large-scale data copies from one site to another. These applications benefit most from high levels of average bandwidth and can adapt their transmission rate based on available capacity. They could similarly defer to higher priority interactive applications during periods of failure or resource constraint. Third, we anticipated no more than a few dozen data center deployments, making central control of bandwidth feasible.

We exploited these properties to adopt a software defined networking (SDN) architecture for our data center WAN interconnect. We were most motivated by deploying routing and traffic engineering protocols customized to our unique requirements. Our design centers around: i) accepting failures as inevitable and common events, whose effects should be exposed to end applications, and ii) switch hardware that exports a simple interface to program forwarding table entries under central control. Network protocols could then run on servers housing a variety of standard and custom protocols. Our hope was that deploying novel routing, scheduling, monitoring, and management functionality and protocols would be both simpler and result in a more efficient network.

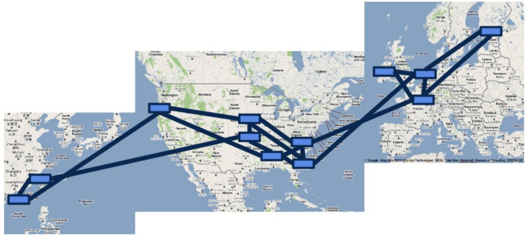
We present our experience deploying Google's WAN, B4, using Software Defined Networking (SDN) principles and OpenFlow [31] to manage individual switches. In particular, we discuss how we simultaneously support standard routing protocols and centralized Traffic Engineering (TE) as our first SDN application. With TE, we: i) leverage control at our network edge to adjudicate among competing demands during resource constraint, ii) use multipath forwarding/tunneling to leverage available network capacity according to application priority, and iii) dynamically reallocate bandwidth in the face of link/switch failures or shifting application demands. These features allow many B4 links to run at near 100% utilization and all links to average 70% utilization over long time periods, corresponding to 2-3x efficiency improvements relative to standard practice.

B4 has been in deployment for three years, now carries more traffic than Google's public facing WAN, and has a higher growth rate. It is among the first and largest SDN/OpenFlow deployments. B4 scales to meet application bandwidth demands more efficiently than would otherwise be possible, supports rapid deployment and iteration of novel control functionality such as TE, and enables tight integration with end applications for adaptive behavior in response to failures or changing communication patterns. SDN is of course

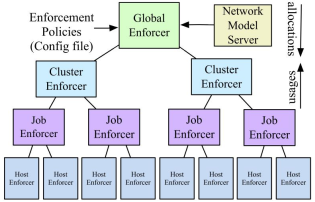
SIGCOMM'13, August 12–16, 2013, Hong Kong, China.

Previous B4 paper published in SIGCOMM 2013

Background: B4 with SDN Traffic Engineering (TE) Deployed in 2012



12-site Topology



Demand Matrix
(via Google BwE)

Site-level tunnels
(tunnels & tunnel splits)

Central
TE
Controller

Per-Site
Domain TE
Controllers

Background: B4 with SDN Traffic Engineering (TE) Deployed in 2012

Key Takeaways:

- ❏ **High efficiency**: Lower per-byte cost compared with *B2 (Google global backbone running RSVP TE on vendor gears)*
- ❏ **Deterministic convergence**: Fast, global TE optimization and failure handling
- ❏ **Rapid software iteration:** ~1 month for developing and deploying a median-size software features

But, it also comes with *new challenges*

Grand Challenge #1: High Availability Requirements

B4 initially had 99% availability in 2013

Service Class	Application Examples	Availability SLO
SC4	Search ads, DNS, WWW	99.99%
SC3	Proto service backend, Email	99.95%
SC2	Ads database replication	99.9%
SC1	Search index copies, logs	99%
SC0	Bulk transfer	N/A

Very demanding goal, given:

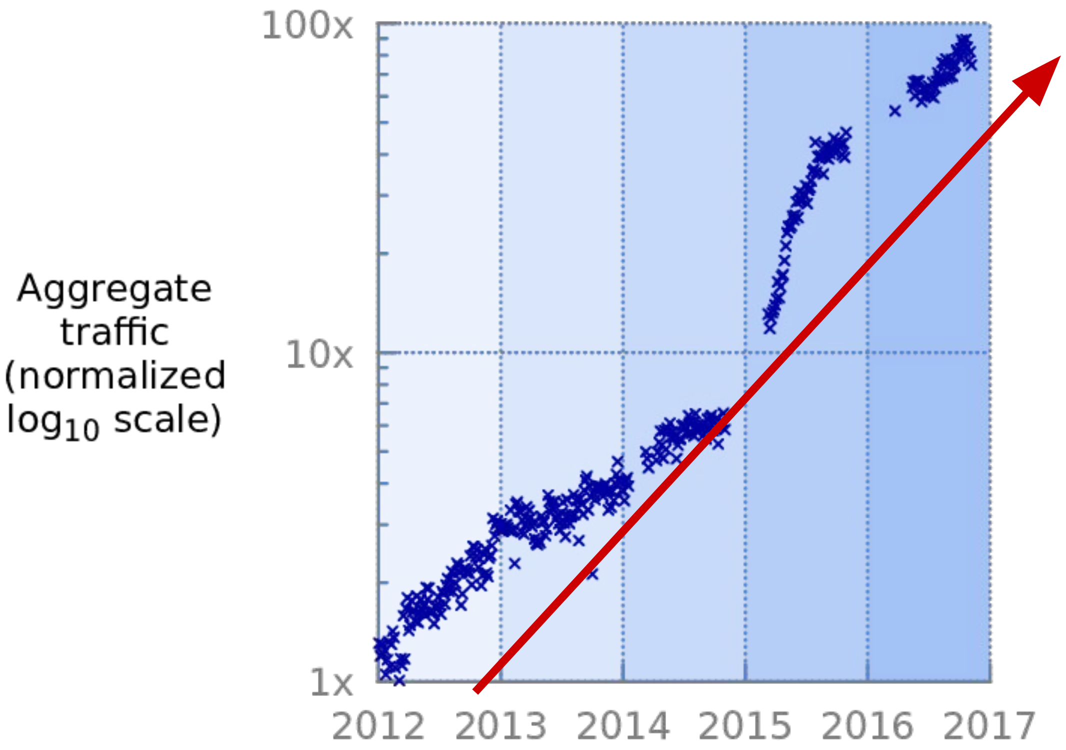
- inherent unreliability of long-haul links
- necessary management operations

B4 initially had 99% availability

Service Class	Application Examples	Availability SLO
SC4	Search ads, DNS, WWW	99.99%
SC3	Proto service backend, Email	99.95%
SC2	Ads database replication	99.9%
SC1	Search index copies, logs	99%
SC0	Bulk transfer	N/A

Grand Challenge #2: Scale Requirements

our bandwidth requirement doubled every ~9 months



traffic increased by >100x in 5 years

Grand Challenge #2: Scale Requirements

our bandwidth requirement doubled every ~9 months

Scale increased across dimensions:

- #Cluster prefixes: *8x*
- #B4 sites: *3x*
- #Control domains: *16x*
- #Tunnels: *60x*

Other challenges: No disruption to existing traffic, maintain high cost efficiency and high feature velocity

To meet these demanding requirements, we’ve had to aggressively develop many point solutions

Lessons Learned

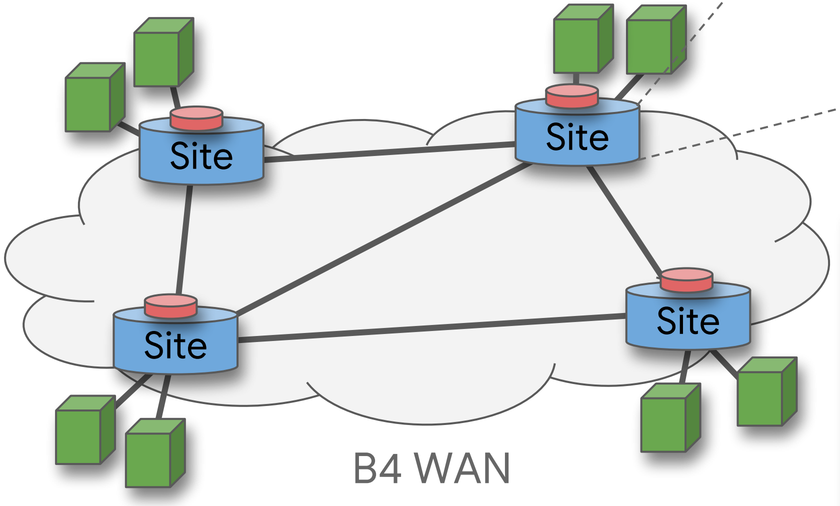
1. **Flat topology scales poorly and hurts availability**

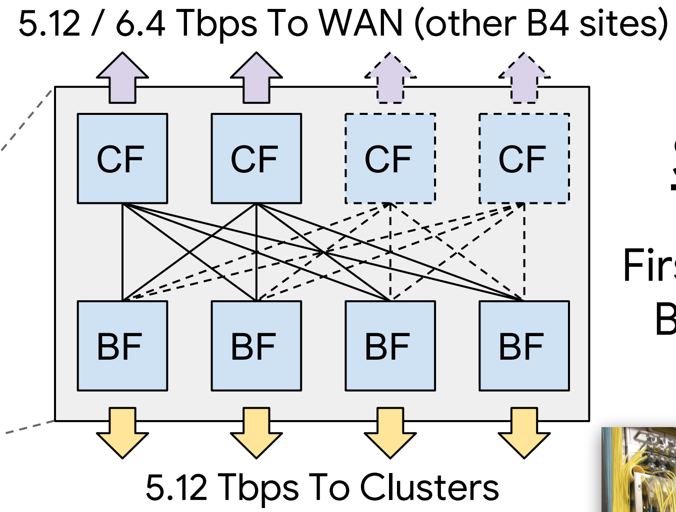
2. Solving capacity asymmetry problem in hierarchical topology is key to achieve high availability at scale

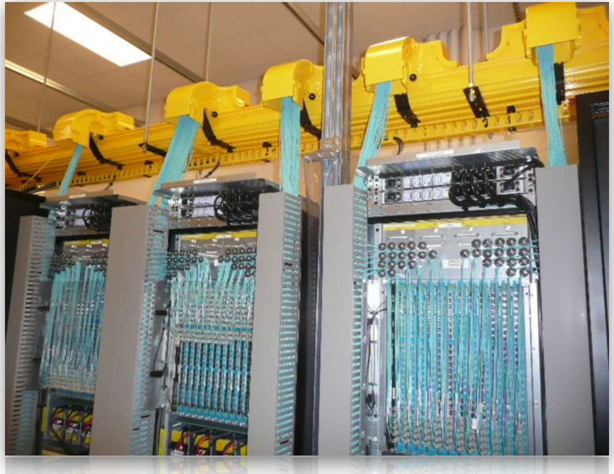
3. Scalable switch forwarding rule management is essential to hierarchical TE

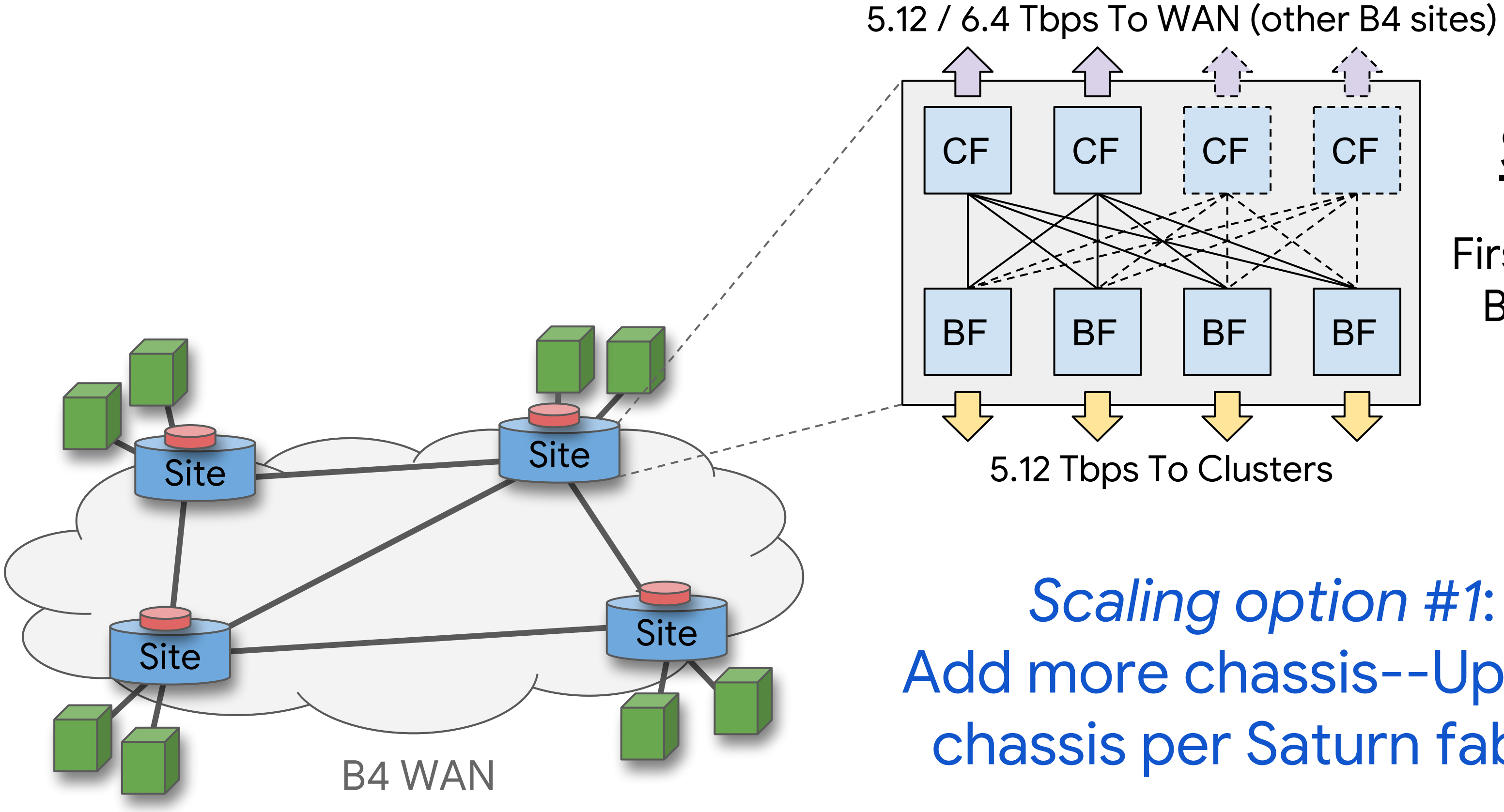
Saturn

First-generation B4 site fabric







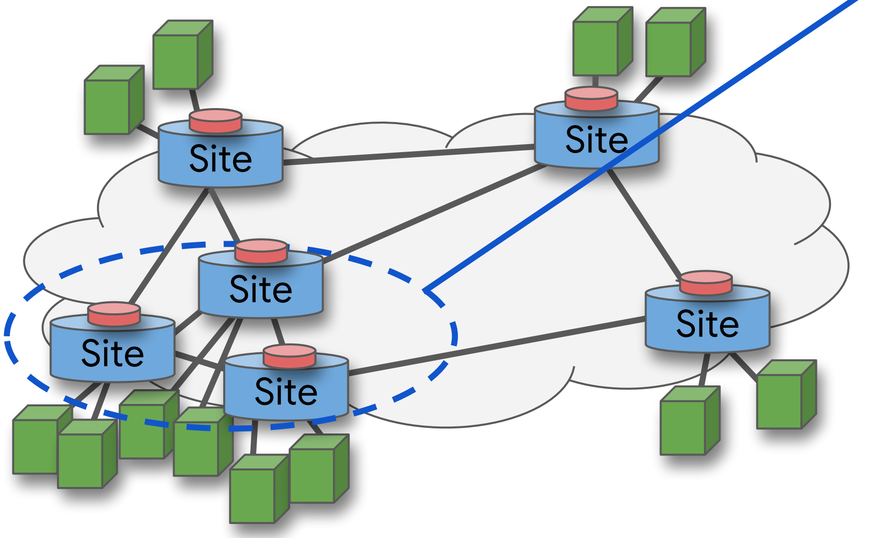


Saturn

First-generation
B4 site fabric

Scaling option #1:
Add more chassis--Up to 8 chassis per Saturn fabric

Scaling option #2: Build multiple B4 sites in close proximity

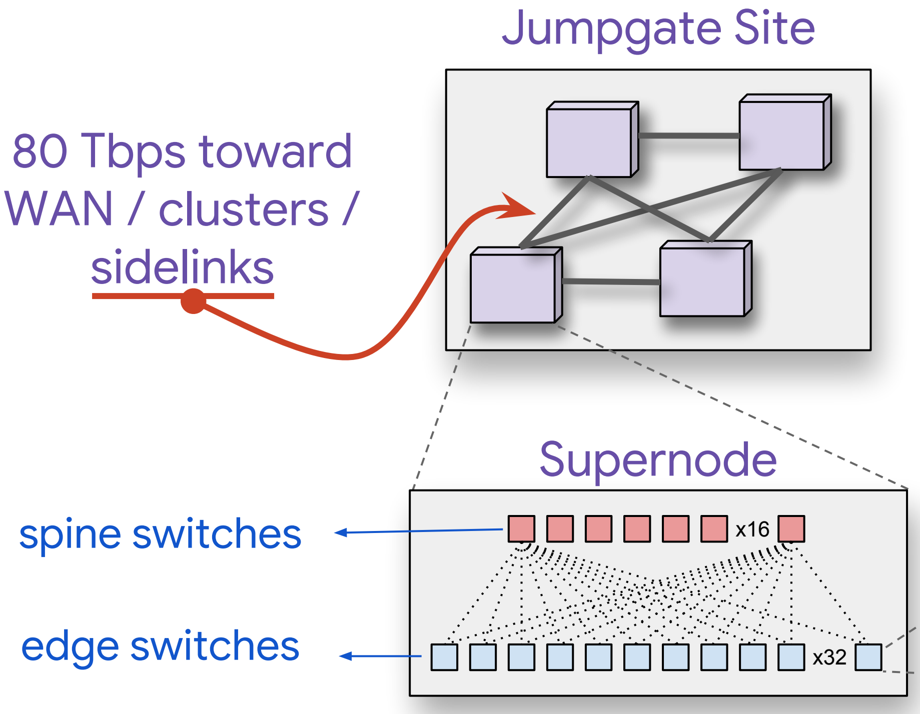


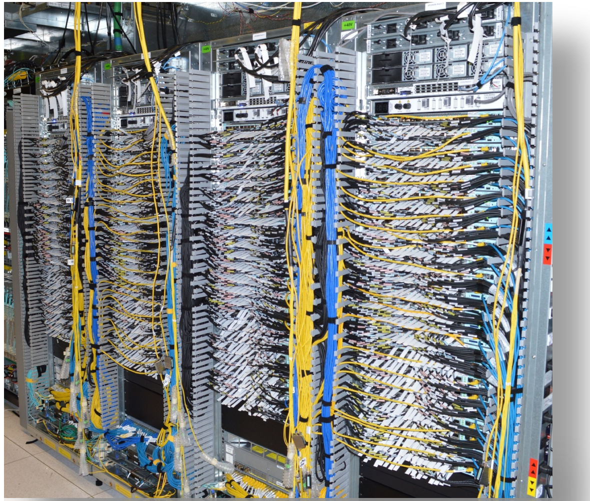
Slower central TE controller

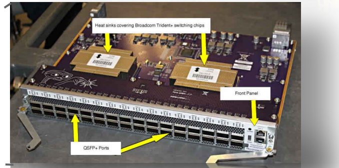
Limited switch table limit

Complicated capacity planning and job allocation

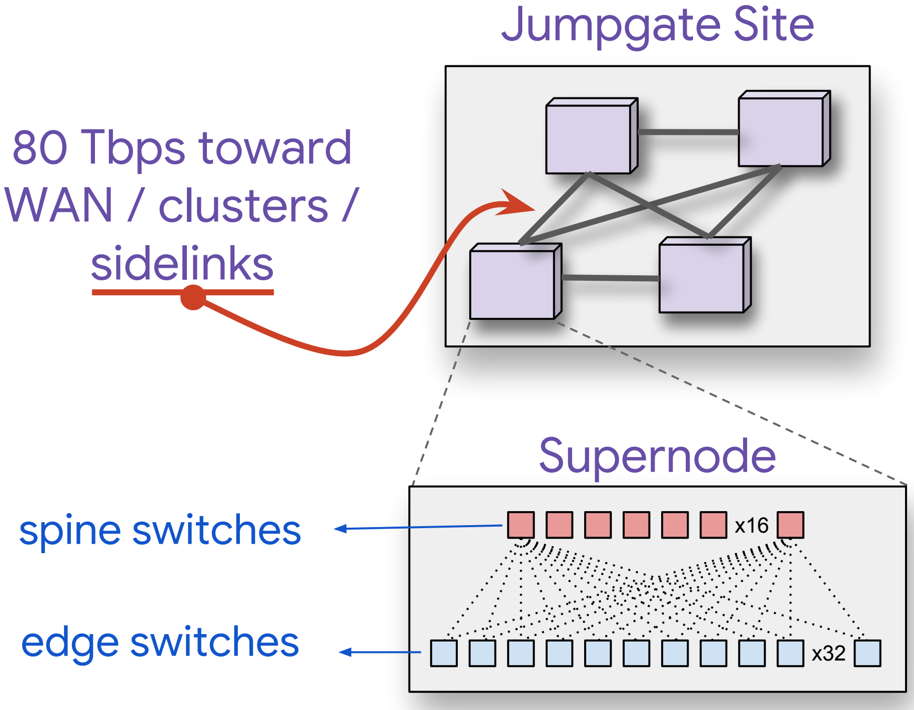
Jumpgate: Two-layer Topology







Jumpgate: Two-layer Topology



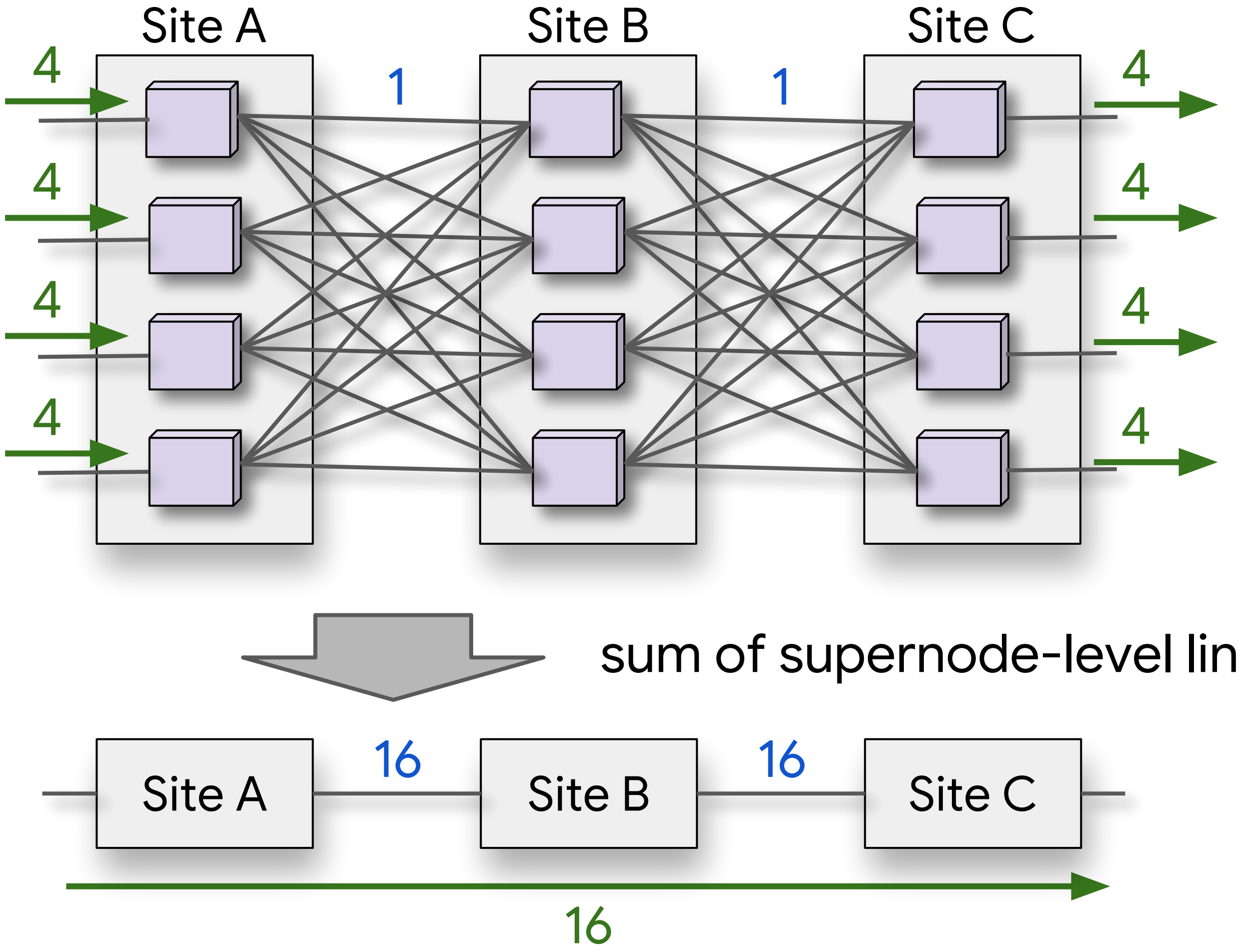
Support horizontal scaling by adding more supernodes to a site

Support vertical scaling by upgrading a supernode in place to new generation

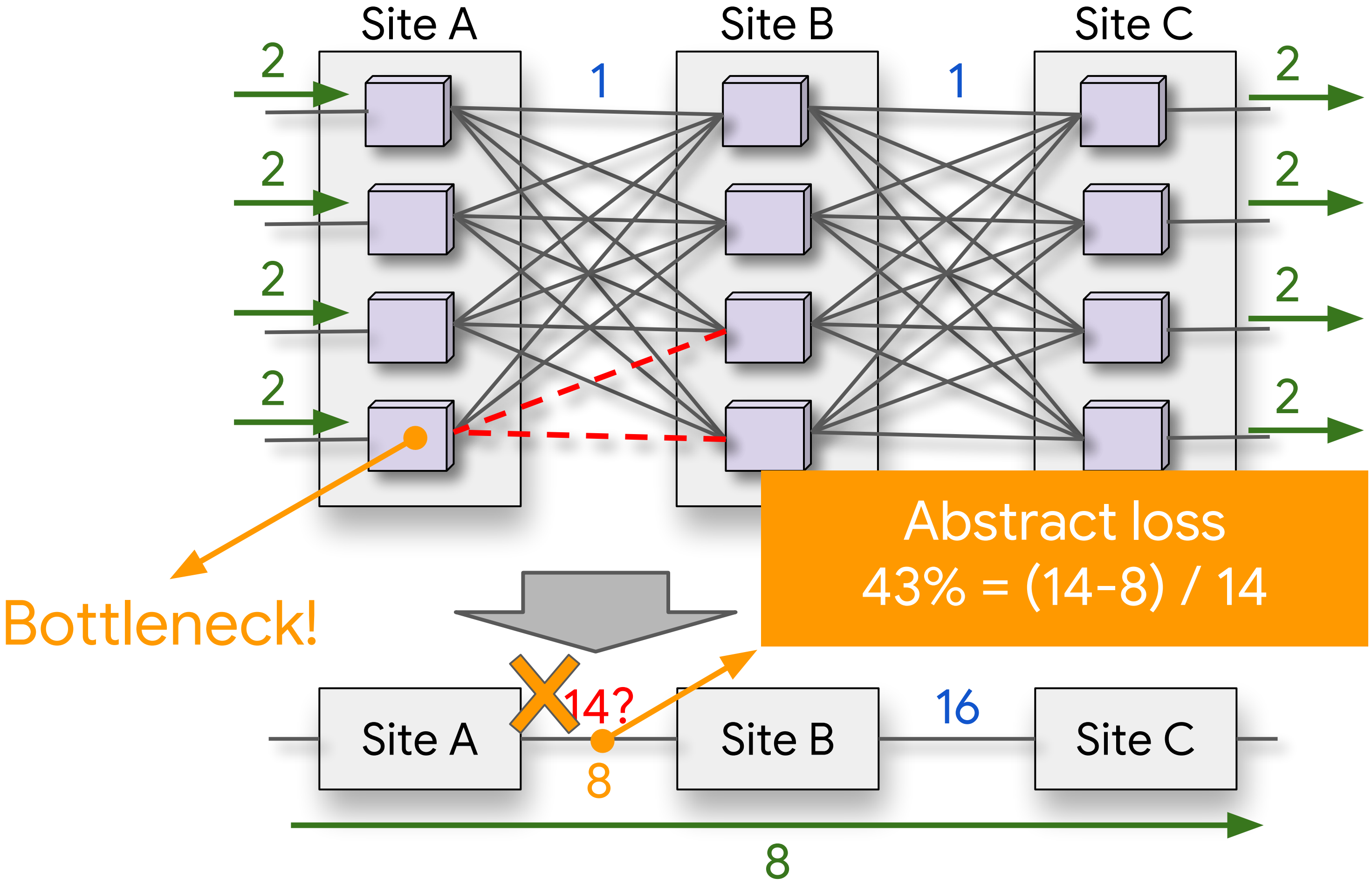
Improve availability with granular, per-supernode control domain

Lessons Learned

1. Flat topology scales poorly and hurts availability
2. **Solving capacity asymmetry problem in hierarchical topology is key to achieve high availability at scale**
3. Scalable switch forwarding rule management is essential to hierarchical TE

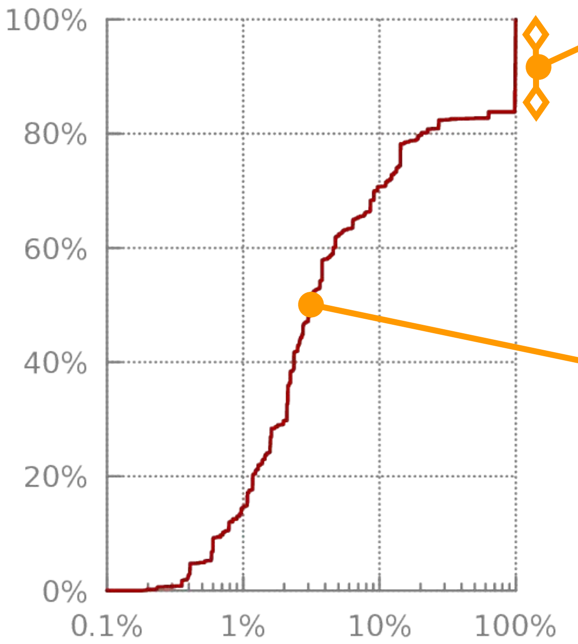


sum of supernode-level link capacity



Bottleneck!

Abstract loss
43% = (14−8) / 14



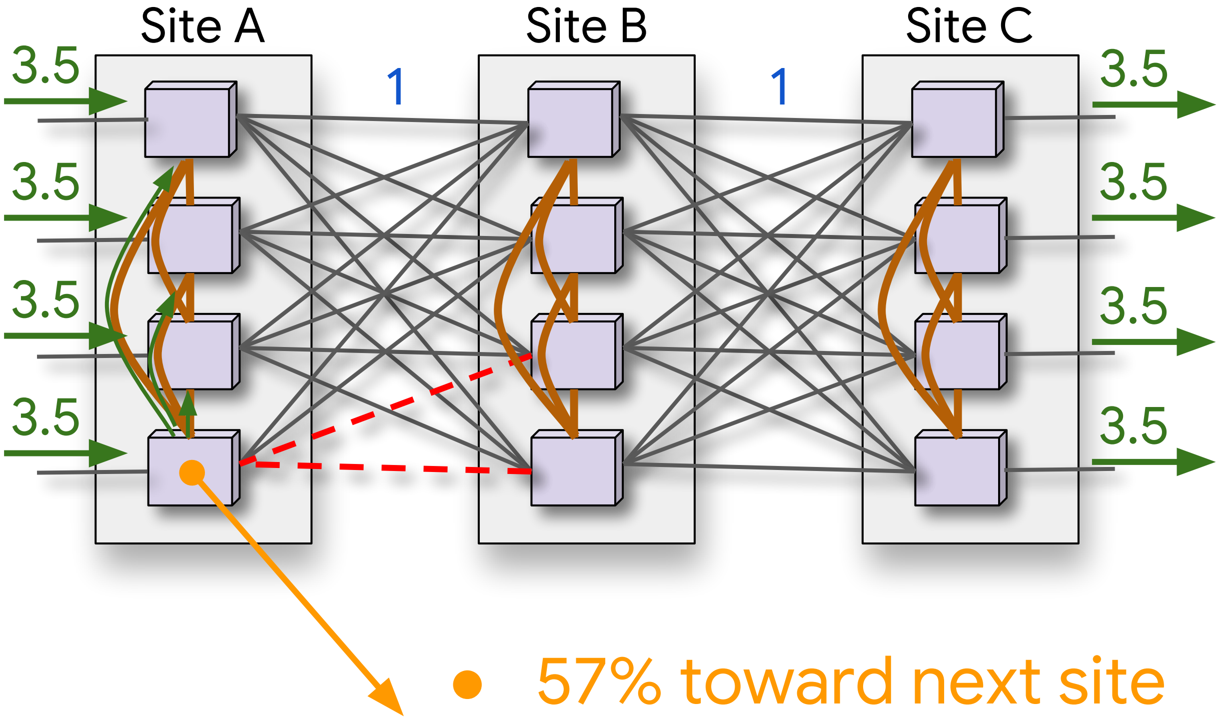
Cumulative function of site-level links and topology events

Site-level link capacity loss due to topology abstraction / total capacity [$\log_{10}$ scale]

100% capacity loss in 18% cases

2% capacity loss at median case due to striping inefficiency

Solution = Sidelinks + Supernode-level TE



- 57% toward next site
- 43% toward self site

Solution = Sidelinks + Supernode-level TE

Multi-layer TE
(Site-level & supernode-level)
turns out to be challenging!

Design Proposals

Hierarchical Tunneling

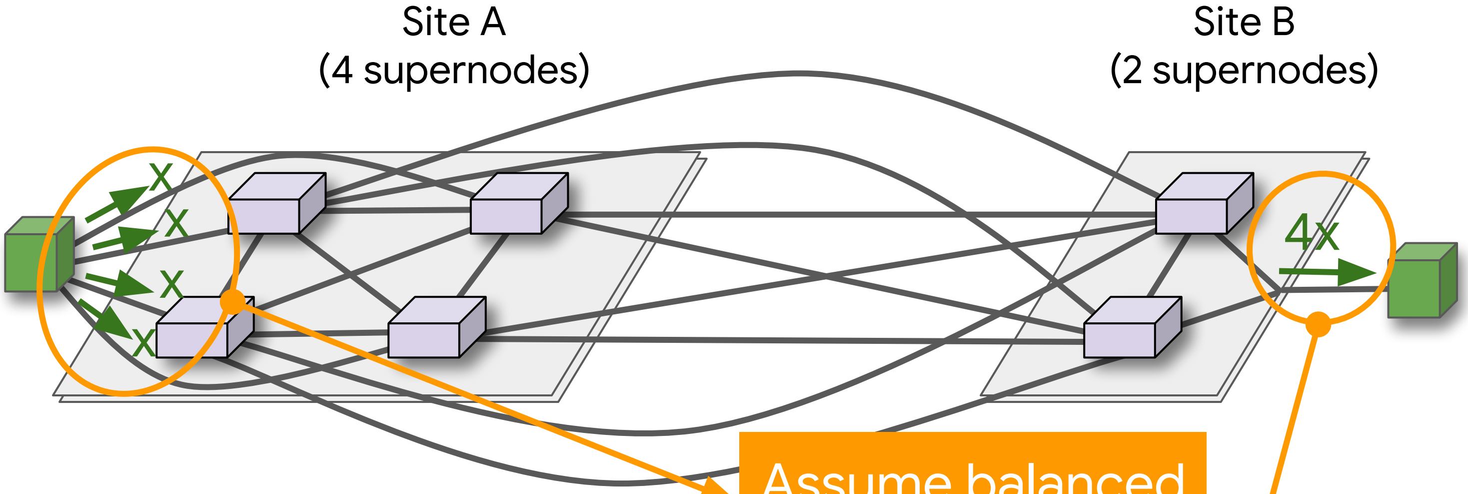
Site-level tunnels +
Supernode-level sub-tunnels

Two layers of IP encapsulation lead to inefficient hashing

Supernode-level TE

Supernode-level tunnels

Scaling challenges: Increase path allocation run time by 188x longer

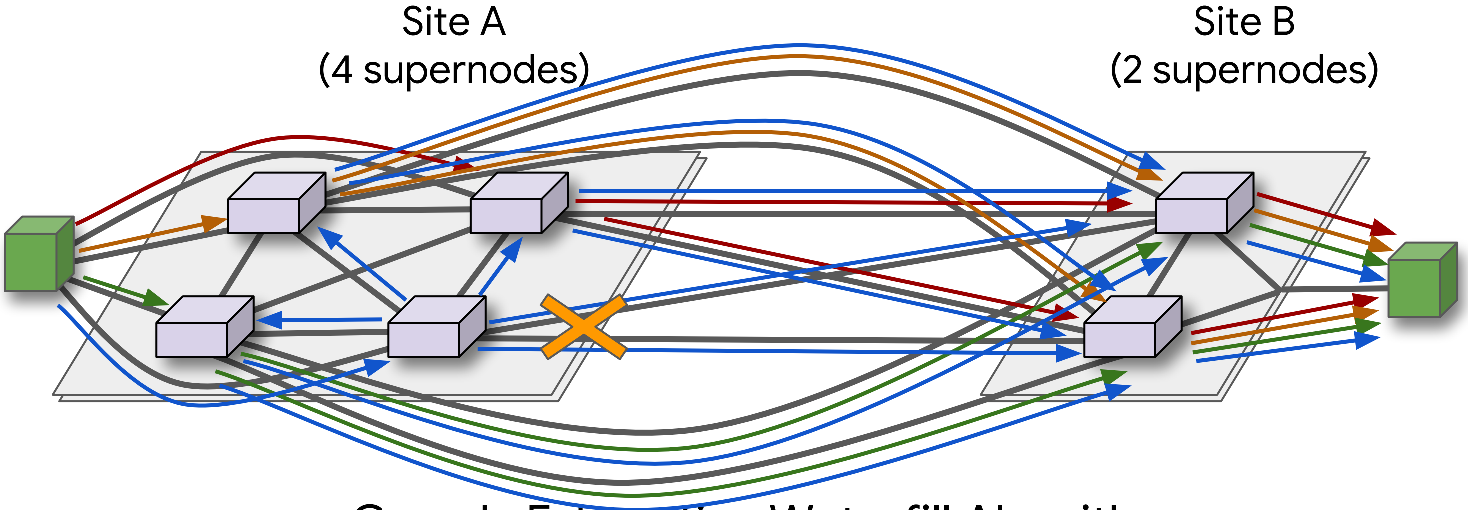


Assume balanced ingress traffic

Maximize admissible demand subject to fairness and link capacity constraint

Tunnel Split Group (TSG)

Supernode-level traffic splits;
No packet encapsulation;
Calculated per site-level link



Greedy Exhaustive Waterfill Algorithm

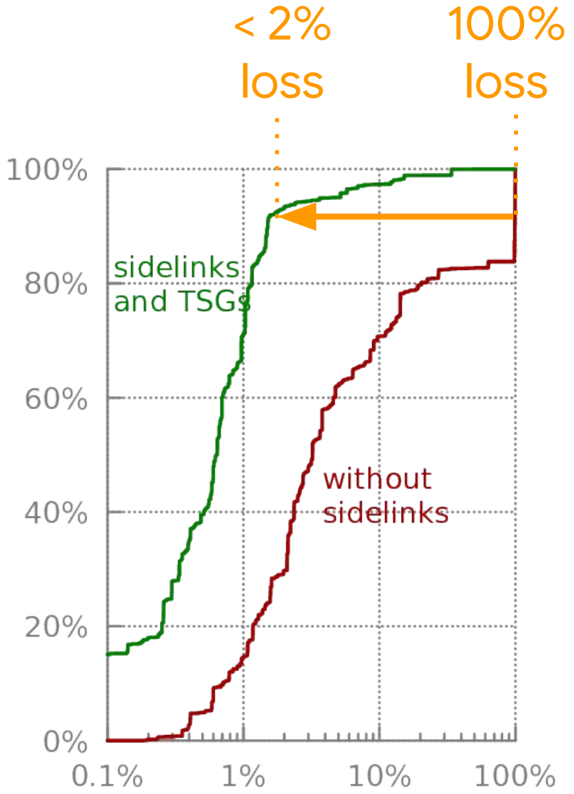
Iteratively allocate each flow on their direct path (w/o sidelinks) or alternatively on their indirect paths (w/ sidelinks on source site) until any flow cannot be allocated further

Provably forwarding loop free

Take less than 1 second to run

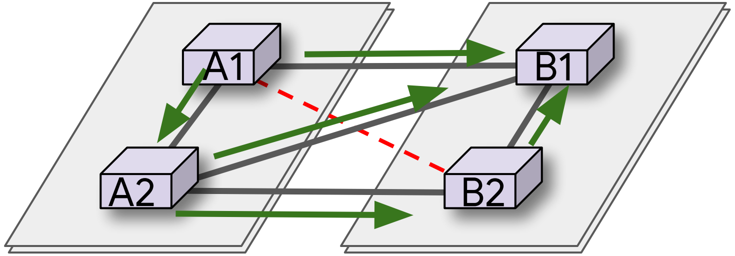
Low abstraction capacity loss

Cumulative function of site-level links and topology events

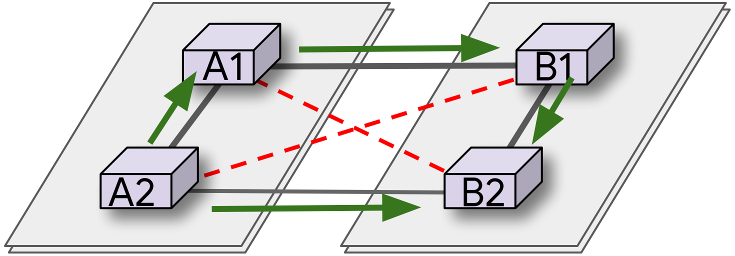


Site-level link capacity loss due to topology abstraction / total capacity [$\log_{10}$ scale]

TSG Sequencing Problem



Current TSGs



Target TSGs

Bad properties during update:

Forwarding Loop

Blackhole

Dependency Graph based TSG Update

1. Map target TSGs to a supernode dependency graph
2. Apply TSG update in reverse topological ordering*

* Share ideas with work in IGP updates:

- Francois & Bonaventure, Avoiding Transient Loops during IGP convergence in IP Networks, INFOCOM'05
- Vanbever et al., Seamless Network-wide IGP Migrations, SIGCOMM'11

Loop-free and no extra blackhole

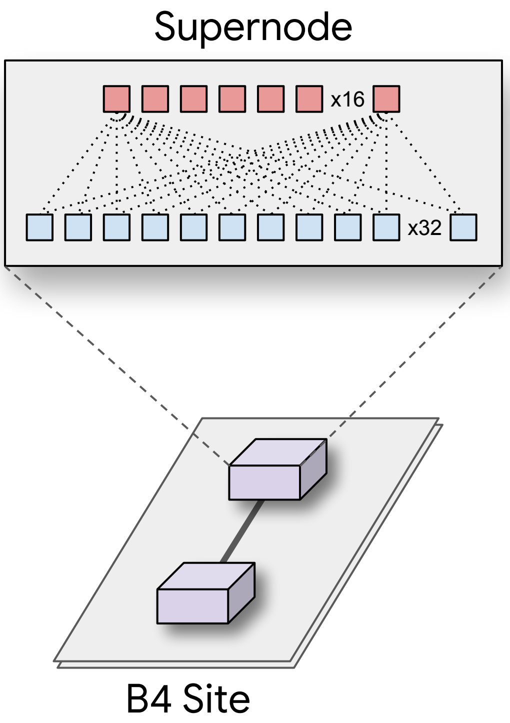
Requires no packet tagging

One or two steps in >99.7% of TSG ops

Lessons Learned

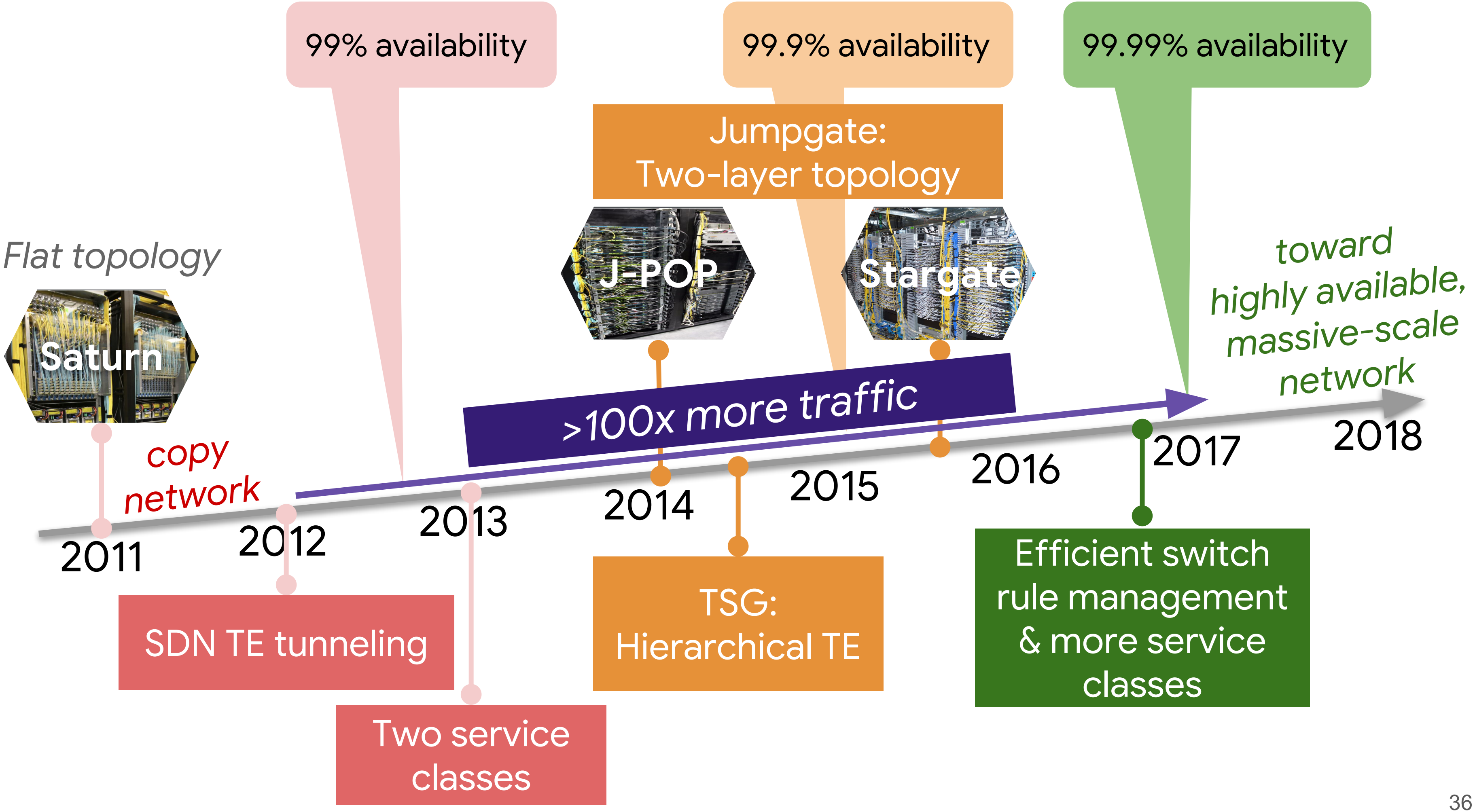
1. Flat topology scales poorly and hurts availability
2. Solving capacity asymmetry problem in hierarchical topology is key to achieve high availability at scale
3. **Scalable switch forwarding rule management is essential to hierarchical TE**

Multi-stage Hashing across Switches in Clos Supernode



1. **Ingress traffic at edge switches:**
    a. Site-level tunnel split
    b. TSG site-level split (to self-site or next-site)
2. **At spine switches:**
    a. TSG supernode-level split
    b. Egress edge switch split
3. **Egress traffic at edge switches:**
    a. Egress port/trunk split

Enable hierarchical TE at scale:
Overall throughput improved by >6%



Flat topology

Saturn

copy network

2011

2012

SDN TE tunneling

99% availability

2013

Two service classes

>100x more traffic

Jumpgate:
Two-layer topology

J-POP

2014

TSG:
Hierarchical TE

99.9% availability

2015

Stargate

2016

99.99% availability

2017

Efficient switch rule management & more service classes

toward highly available, massive-scale network

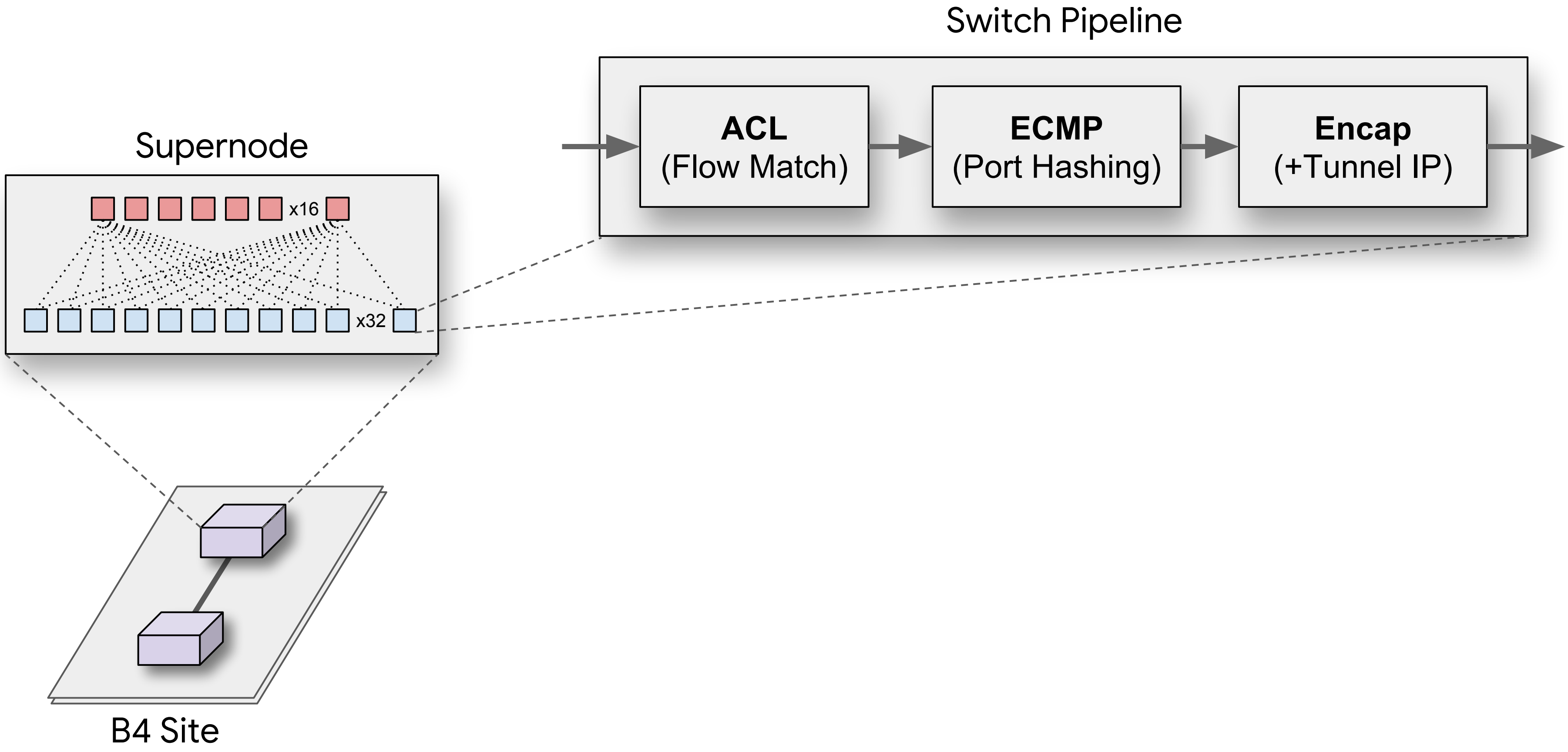
2018

Conclusions

- Highly available WAN with plentiful bandwidth offers unique benefits to many cloud services (e.g., Spanner)
- Future Work--Limit the blast radius of rare yet catastrophic failures
  - Reduce dependencies across components
  - Network operation via per-QoS canary

B4 and After: Managing Hierarchy, Partitioning, and Asymmetry for Availability and Scale in Google's Software-Defined WAN

Before	After
Copy network with 99% availability	High-available network with 99.99% availability
Inter-DC WAN with moderate number of sites	100x more traffic, 60x more tunnels
Saturn: flat site topology & per-site domain TE controller	Jumpgate: hierarchical topology & granular TE control domain
Site-level tunneling	Site-level tunneling in conjunction with supernode-level TE (“Tunnel Split Group”)
Tunnel splits implemented at ingress switches	Multi-stage hashing across switches in Clos supernode



Switch Pipeline

ACL
(Flow Match)

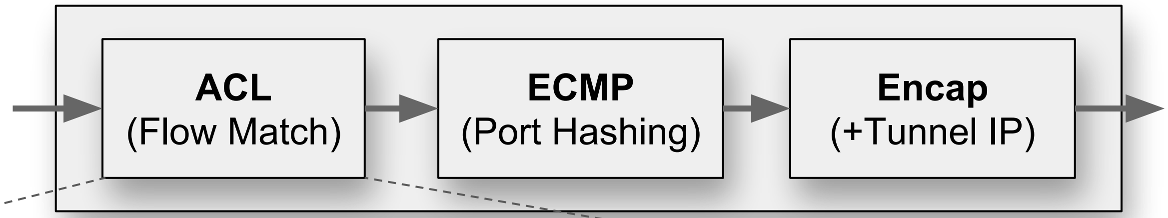
ECMP
(Port Hashing)

Encap
(+Tunnel IP)

Supernode

B4 Site

Switch Pipeline



ACL
(Flow Match)

ECMP
(Port Hashing)

Encap
(+Tunnel IP)

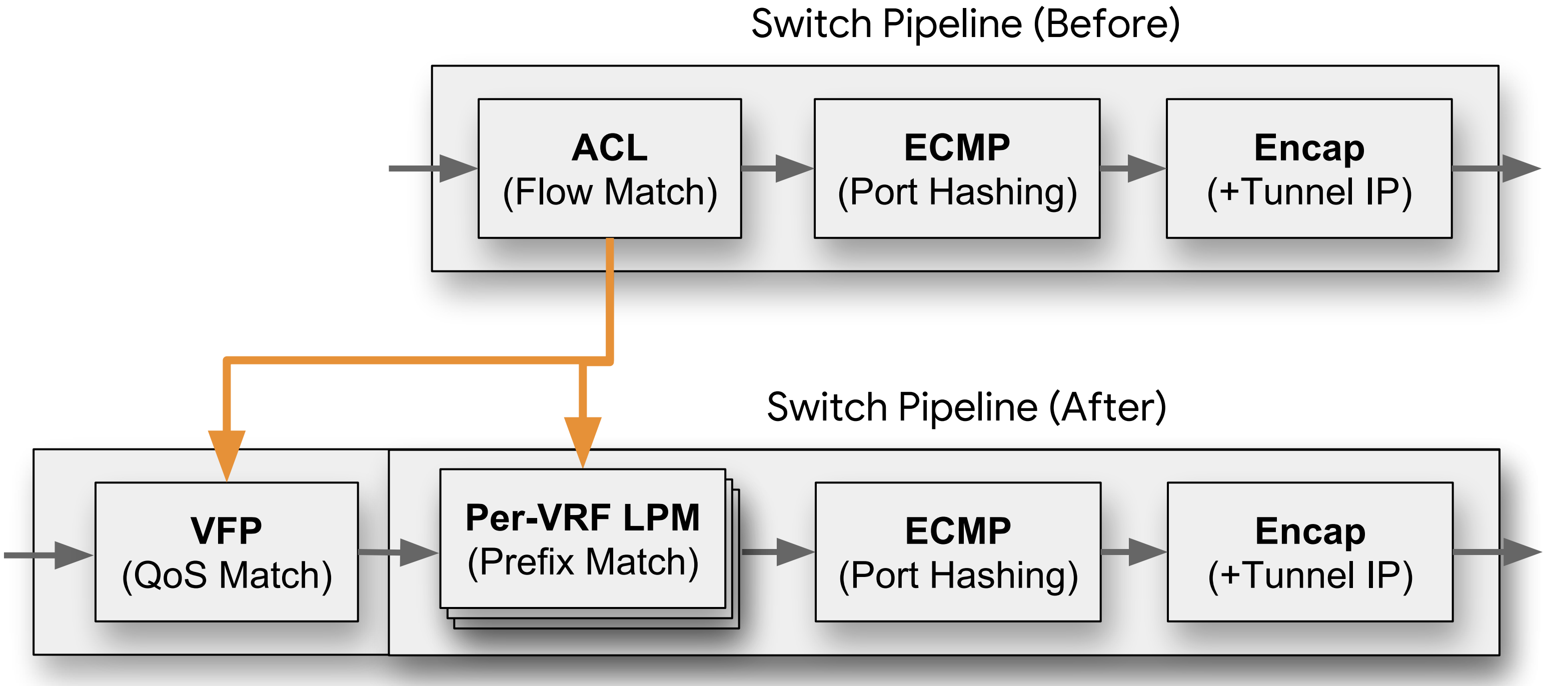
$$\text{Size(ACL)} \geq (\#\text{Sites} \times \#\text{PrefixesPerSite} \times \#\text{ServiceClasses})$$

Up to 3K entries

>16 aggregated IPv4 & IPv6 cluster prefixes

6 aggregated QoSes

Scaling bottleneck: Hit ACL table limit with ~32 sites



Switch Pipeline (Before)

ACL
(Flow Match)

ECMP
(Port Hashing)

Encap
(+Tunnel IP)

Switch Pipeline (After)

VFP
(QoS Match)

Per-VRF LPM
(Prefix Match)

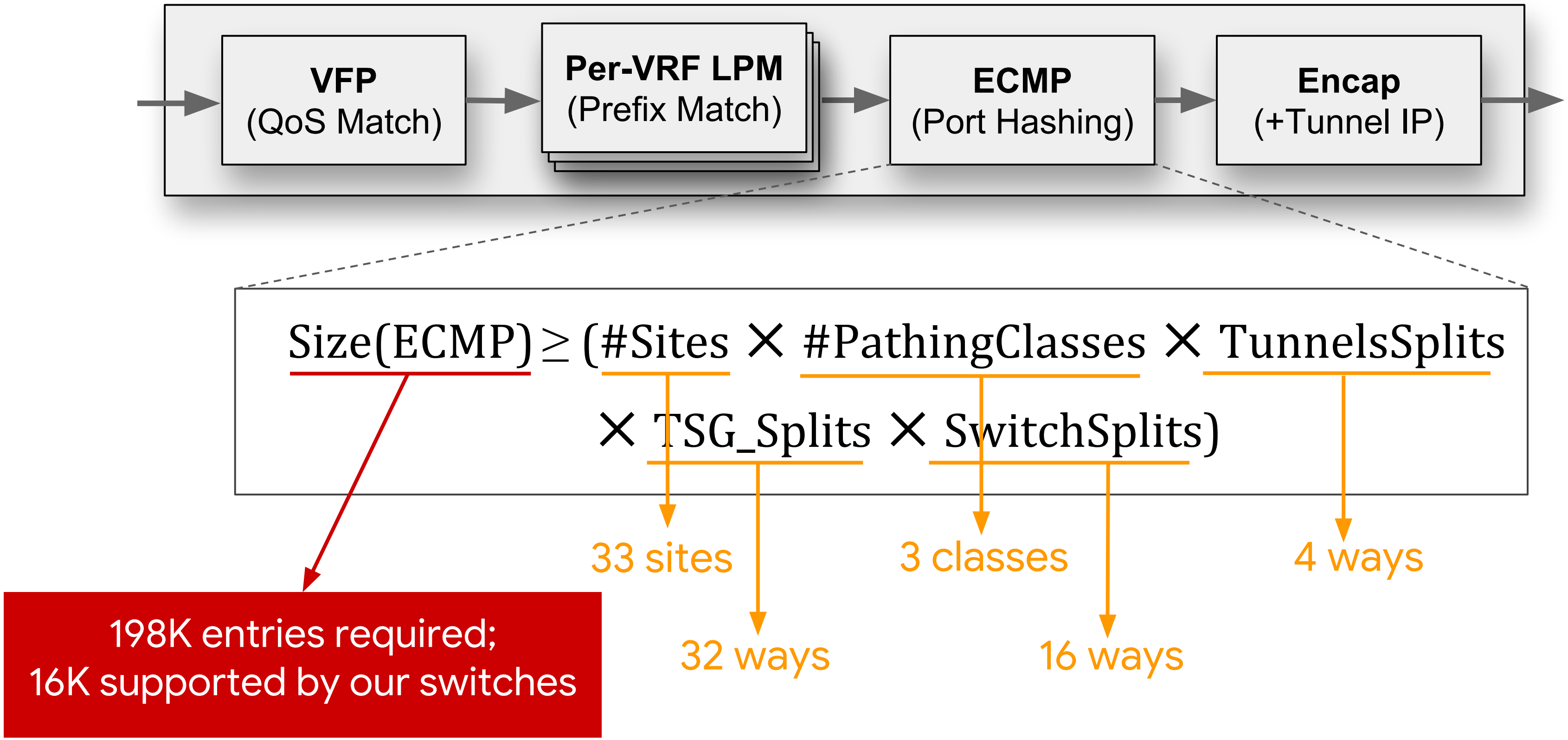
ECMP
(Port Hashing)

Encap
(+Tunnel IP)

Enable new features:
Disable per-flow tunneling

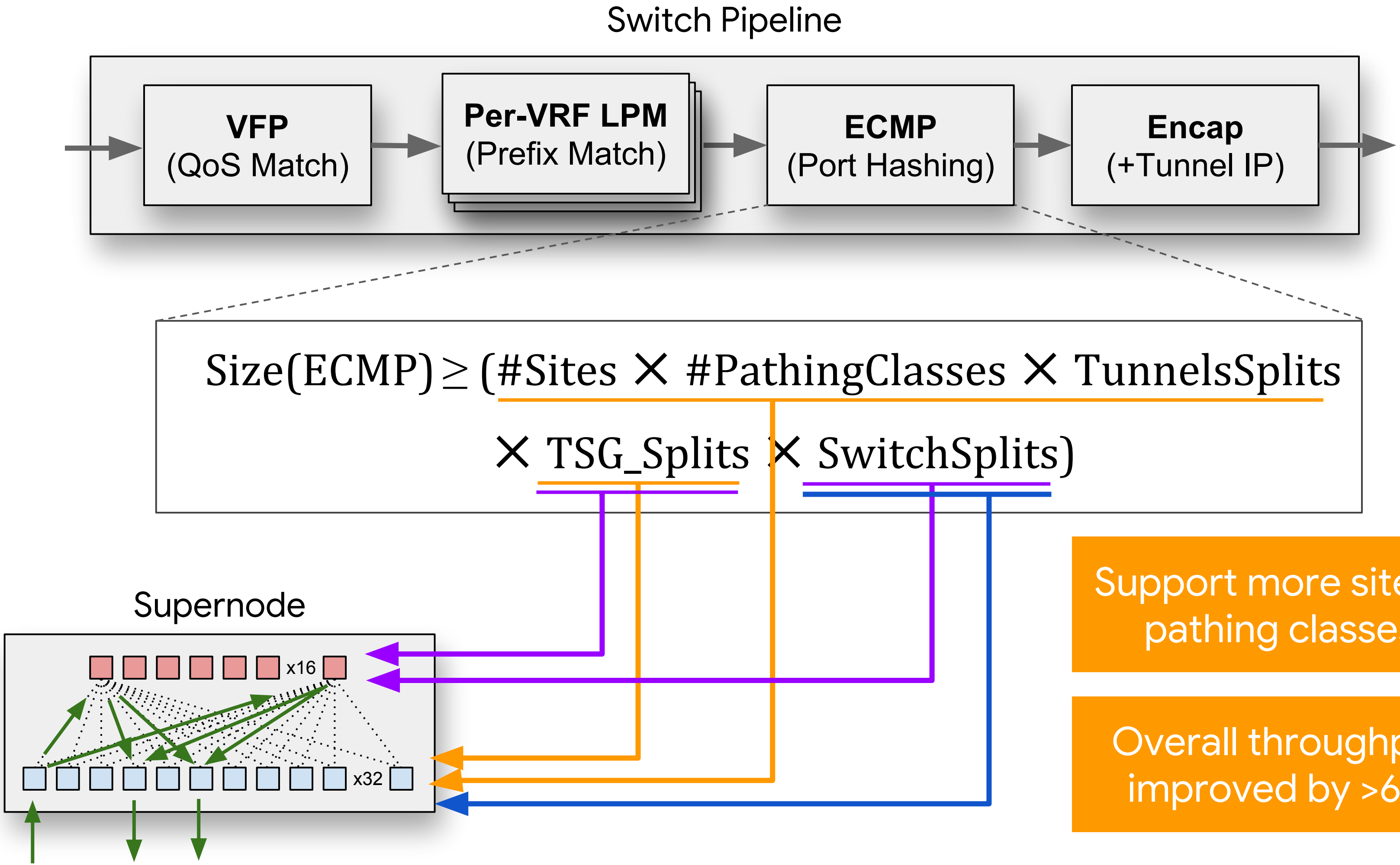
Increase #
supported sites by 60x

Switch Pipeline



$$\mathrm{Size(ECMP)} \geq (\mathrm{\#Sites} \times \mathrm{\#PathingClasses} \times \mathrm{TunnelsSplits} \times \mathrm{TSG\_Splits} \times \mathrm{SwitchSplits})$$

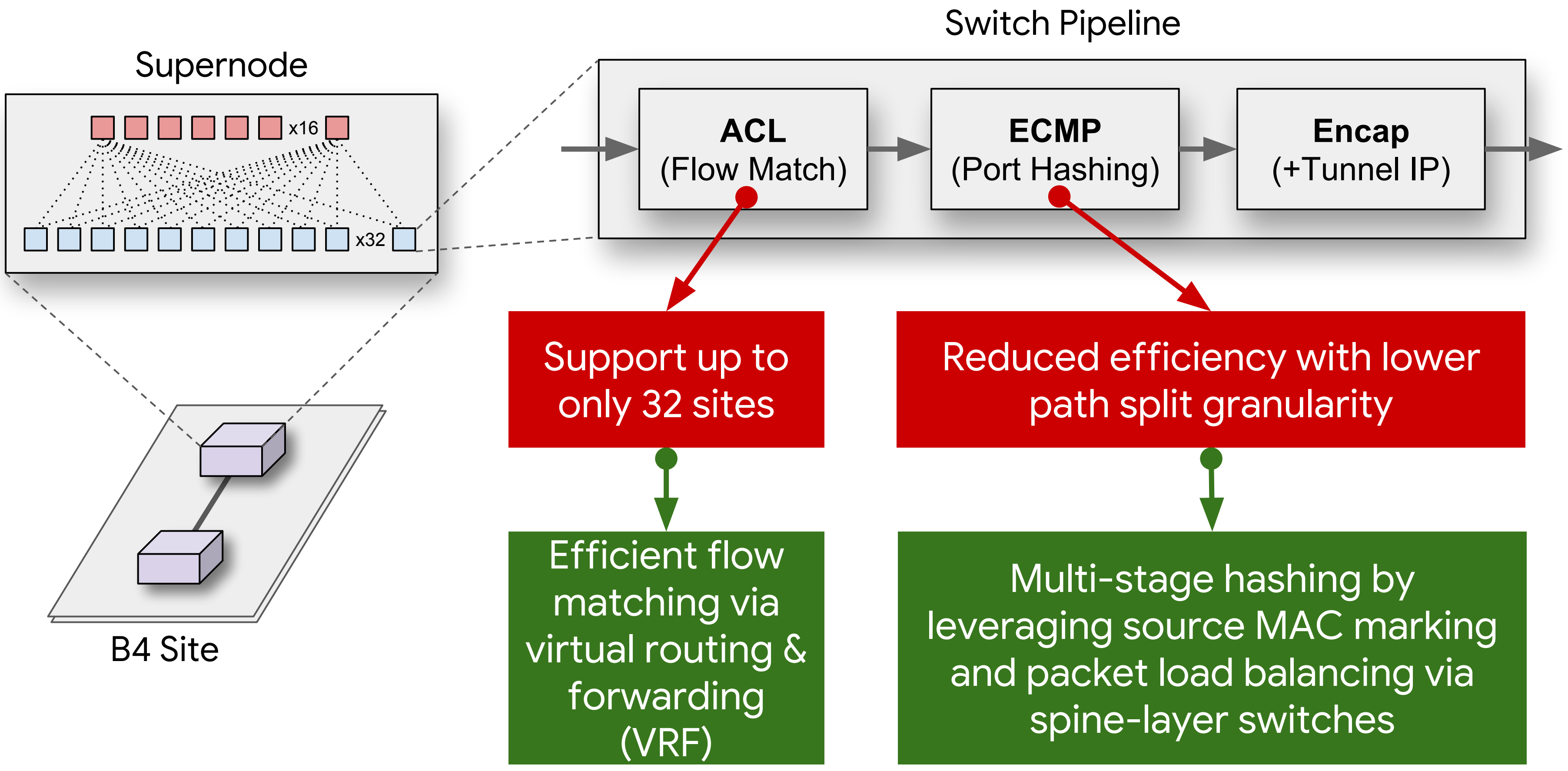
198K entries required;
16K supported by our switches



$$\text{Size(ECMP)} \geq (\#\text{Sites} \times \#\text{PathingClasses} \times \text{TunnelsSplits} \times \text{TSG\_Splits} \times \text{SwitchSplits})$$

Support more sites & pathing classes

Overall throughput improved by >6%



ACL
(Flow Match)

ECMP
(Port Hashing)

Encap
(+Tunnel IP)

Support up to only 32 sites

Efficient flow matching via virtual routing & forwarding (VRF)

Reduced efficiency with lower path split granularity

Multi-stage hashing by leveraging source MAC marking and packet load balancing via spine-layer switches