Google

NetInfra

Under the covers: the networks of a cloud provider

Jeff Mogul

Google Network Infrastructure

mogul@google.com

Goal for today: understand something about Cloud networks

“Cloud” is a fuzzy concept, but whatever it means, it involves networks

- Lots of networks, and different kinds

Agenda

- What do I mean by “Cloud”?
- What is a virtual network? (Just briefly)
- ~~How do cloud providers create virtual networks?~~ (Nick will tell you, soon)
- How do we build a scalable, reliable data-center network?
- How do we build scalable, reliable WANs?

Disclaimers & Credits

- Not all cloud providers do things exactly the same way
- There are a lot of things about Google that I can't tell you
- I've borrowed a lot of material from other (mostly) Google talks:
  - *B4: Experience with a Globally-Deployed Software Defined WAN*
    - by lots of Googlers; talk by Amin Vahdat, at SIGCOMM 2013
  - a talk by Amin Vahdat at the *Open Networking Summit* (ONS 2014)
    - Recording is on YouTube (search for "Vahdat ONS 2014")
  - *Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network*
    - by Arjun Singh and many others, SIGCOMM 2015
  - *The Rise of Cloud Computing Systems*
    - Jeff Dean's talk from the SOSP '15 *History Day* Workshop

What’s a cloud, and why?

Before "cloud": large systems

Various companies built large systems, because they had to:

- Very resource-intensive interactive services, such as
  - Search (Google)
  - Electronic commerce (Amazon)
  - Email (AOL, HotMail, GMail, etc.)
- We learned how to make these scalable, reliable, and cheap:
  - "Scale out" (lots of cheap systems, networked together) instead of "Scale up"
  - Use low-reliability but cheap HW; gain reliability through distributed systems
  - Provide a high-level view of the resource pool, rather than "here are a lot of parts"
    - Via frameworks such as MapReduce/Hadoop, GFS/HBase, and many others
    - Design pattern: centralized master for control, thousands each of workers & clients
    - Framework maps computation/storage automatically onto a large cluster of machines

The genesis of cloud

- A few companies had mastered large scale-out systems
- Most other users were struggling with the basics:
  - power, cooling, machine repair, upgrades, patches, network plumbing, etc. ...
  - … none of which differentiates you from your competition
  - Jeff Bezos called this “undifferentiated heavy lifting”
- For many decades, people had a vision of “computing as a utility”
  - But mostly this was impossible to get off the ground, because it cost too much
- The “Aha!” moment: we can bring scale-out computing to everyone
  - Hide the messy stuff behind simplified interfaces
  - Don’t try to solve everyone’s problems at once
  - Ruthlessly focus on cost, by leveraging economies of scale

There's more than one kind of cloud

- Infrastructure-as-a-Service (IaaS)
  - Provider offers virtual computers/containers, storage devices, and networks
  - Customer provides all the software, from the operating system to the applications
  - Examples: Amazon EC2, Google GCE
- Platform-as-a-Service (PaaS)
  - Provider manages high-level building blocks, makes them reliable and scalable
  - Customer writes code/scripts to glue these together (perhaps w/some IaaS)
  - Examples: Google Dataflow (big-data analytics-as-a-service)
- Software-as-a-Service (SaaS)
  - Provider creates and runs the applications
  - Users access applications via Web browser or apps
  - Examples: Salesforce.com (CRM), Gmail, Google Docs
- Many cloud "tenants" will use both IaaS and PaaS at the same time

Typical characteristics of a cloud system

- Most of the code and data lives within the provider's infrastructure
  - And the users can be anywhere on the Internet
  - Some businesses use cloud processing to augment "on-premises" legacy systems
- The provider manages all of the physical infrastructure
  - Customer can usually ignore HW failures, SW upgrades, diesel generators, etc.
- You only pay for what you use
  - By the hour, by the Gbyte, by the query, etc.
  - And as computers get cheaper, you get to pay less

SLIs, SLOs, SLAs, and nines

Are you getting what you paid for?

- Service Level Indicator (SLI): a carefully-defined measurement
  - e.g.: round-trip latency between two VMs, or service uptime
- Service Level Objective (SLO): a goal, based on one or more SLIs
  - e.g.: 99.9% of RPCs have a round-trip latency below 500 microseconds
  - e.g.: my storage service is available for use 99.95% of the time, over one month
- Service Level Agreement (SLA): an SLO with consequences
  - e.g.: if you don’t meet the latency SLO, you have to refund double what I paid you

Availability SLA is often stated in terms of “nines”

- For example, “5 nines” means that the SLA guarantees 99.999% uptime
- … which is 5.26 minutes of downtime per year, or 864 milliseconds per day

Why customers like using the cloud

- Often cheaper than managing their own systems
  - Cloud providers can exploit economies of scale
  - Also, converts “capital expense” (CapEx) costs to “operating expense” (OpEx)
- If they need to grow quickly, they can
  - They can shrink quickly, too
- They can rapidly launch new applications
  - Through flexible resources and a growing set of PaaS components

But: some customers are nervous about the cloud:

- Can I trust the provider to be reliable and honest?
- Will the government get a warrant to snoop on my data?
- Can I predict how large my monthly bills will be?
- What if I want to switch to a different provider?

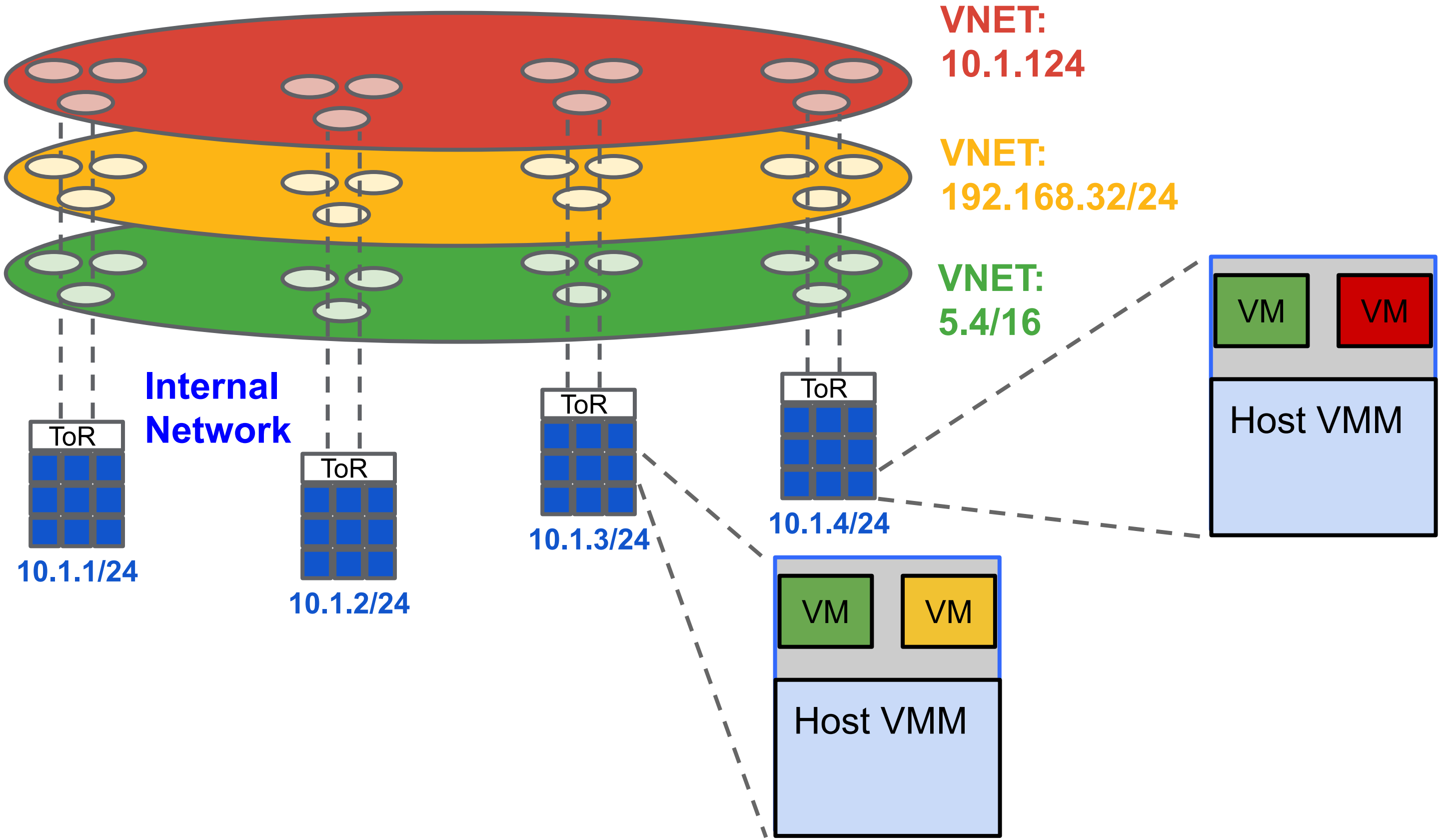
Virtual networks

What’s a “virtual network”?

A virtual network is to a real network as a virtual machine is to a real computer:

- In both cases, an abstraction that
  - preserves the important aspects from the user’s point of view
  - hides the boring details of the underlying “real” implementation
  - allows the provider to efficiently allocate resources among tenants
  - supports *isolation* (security and performance) between the tenants
- Typically, an IaaS tenant has a number of VMs connected by a virtual network
  - The provider maps this structure onto its underlying real network
  - This mapping is seldom 1:1
- PaaS tenants also usually have virtual networks, connecting to PaaS services

VMs and virtual networks in action



What do we need from virtual networks?

- Almost all IaaS and PaaS cloud tenants need to connect multiple things:
  - VMs [or “containers”, but for simplicity, I will ignore that in this talk]
  - PaaS services
  - Internet users
  - On-premises systems
- Cloud tenants want:
  - Predictable, high performance and availability
  - Flexible scaling and re-arrangement of their virtual networks
- A cloud provider needs to:
  - Enforce isolation between tenants, and protect them and itself against attacks
  - Meet its SLAs for network availability and performance
  - Collect billing-related information

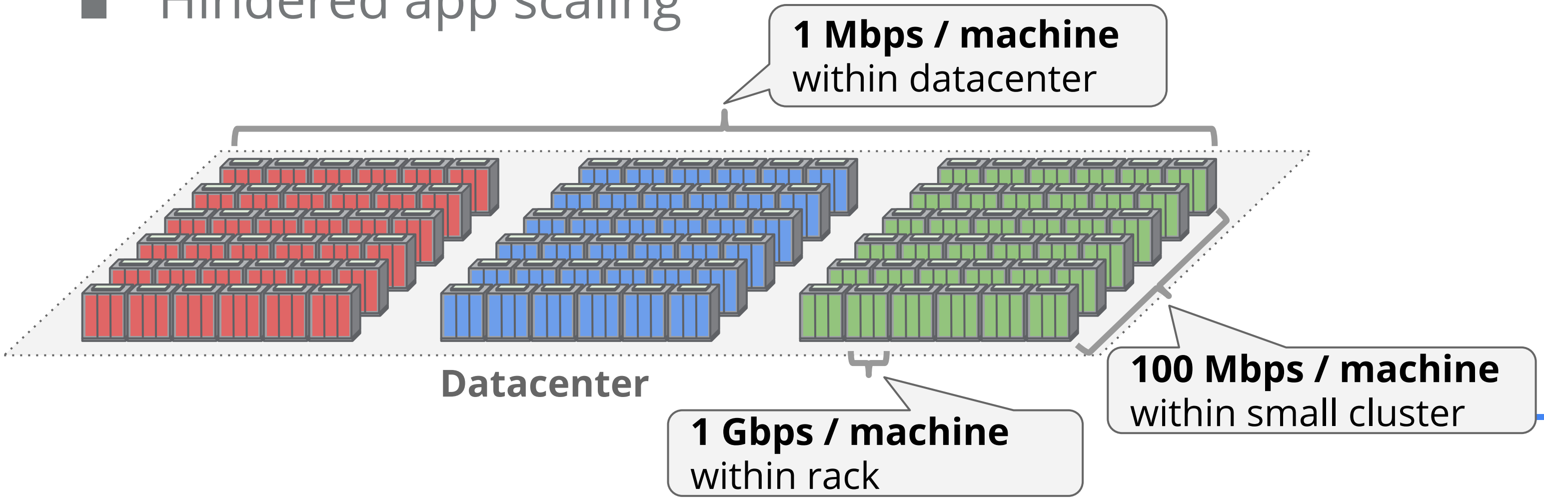
Data center networks

Our datacenters are big ...


Google
google.com/datacenters

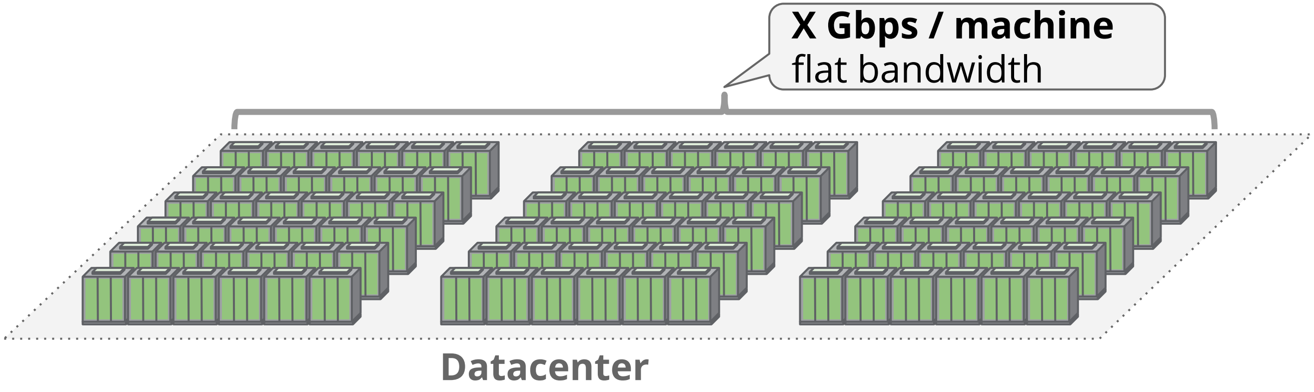
Grand challenge for datacenter networks

- Tens of thousands of servers, interconnected in clusters
- 10 years ago: *Islands of bandwidth* were a bottleneck for Google
    - Engineers struggled to optimize for bandwidth locality
    - “Stranded” compute/memory resources
    - Hindered app scaling



Grand challenge for datacenter networks

- **Challenge: Flat bandwidth profile across all servers would**
  - Simplify job scheduling, by removing the need for locality
  - Save significant resources, via better bin-packing
  - Allow better application scaling



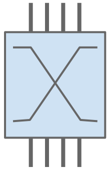
Motivation

- **Traditional network architectures**
  - Cost prohibitive
  - Could not keep up with our bandwidth demands
  - Operational complexity of “box-centric” deployment

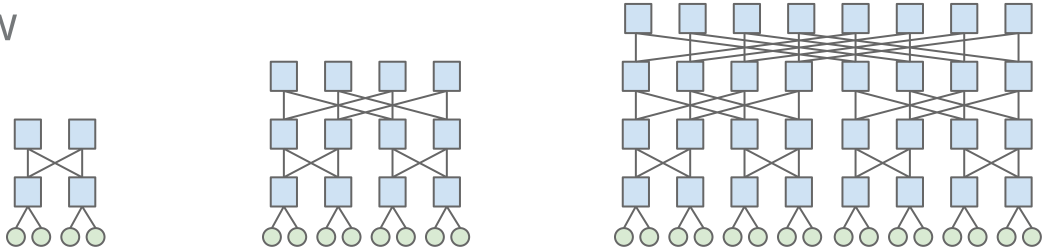
- **Opportunity: A datacenter is a single administrative domain**
  - One organization designs, deploys, controls, operates the n/w
  - ...And often also the servers

Three pillars that guided us

Merchant silicon: General purpose, commodity priced, off the shelf switching components

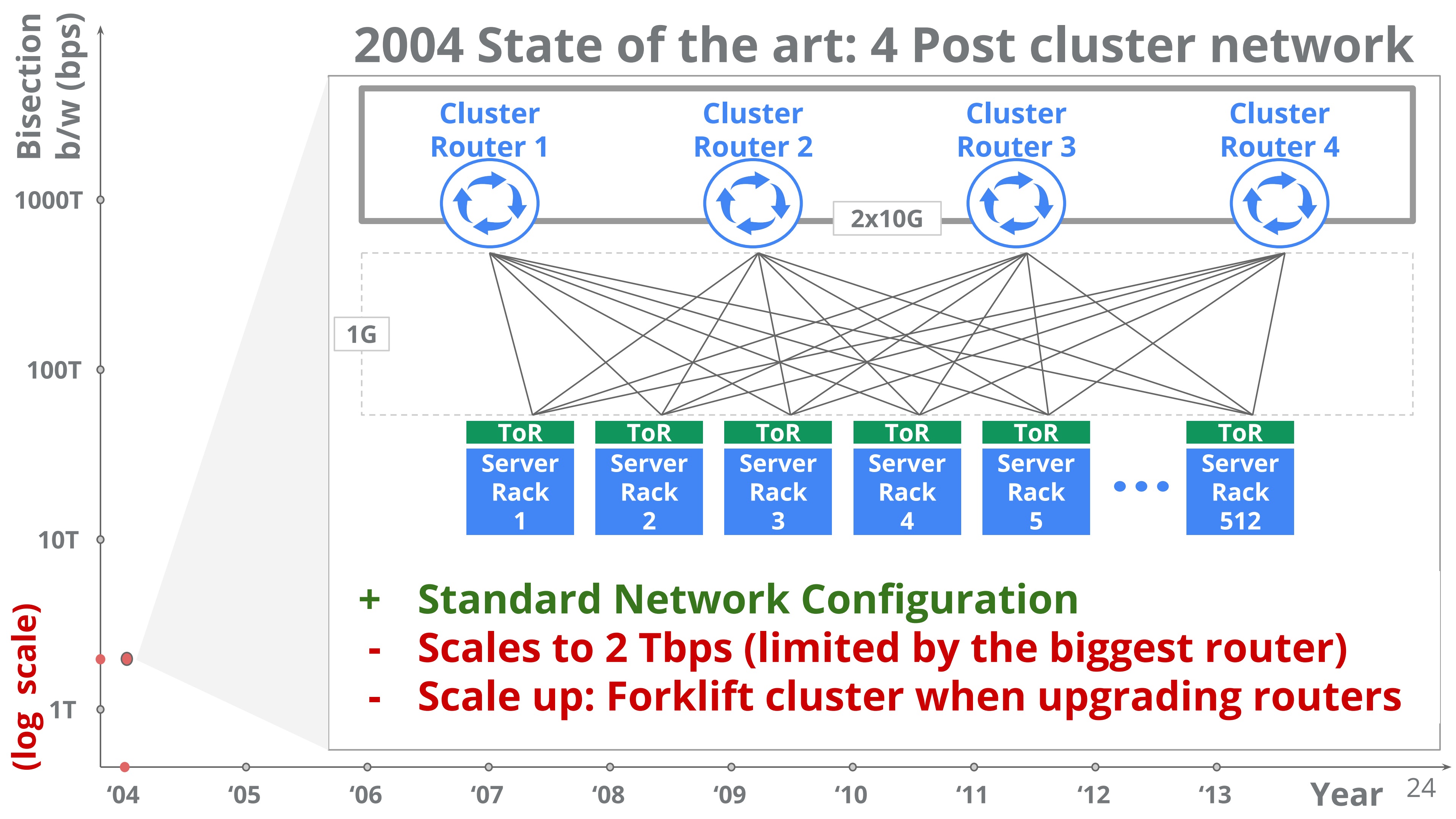


Clos topologies: Accommodate low radix switch chips to scale nearly arbitrarily by adding stages



Centralized control / management

2004 State of the art: 4 Post cluster network

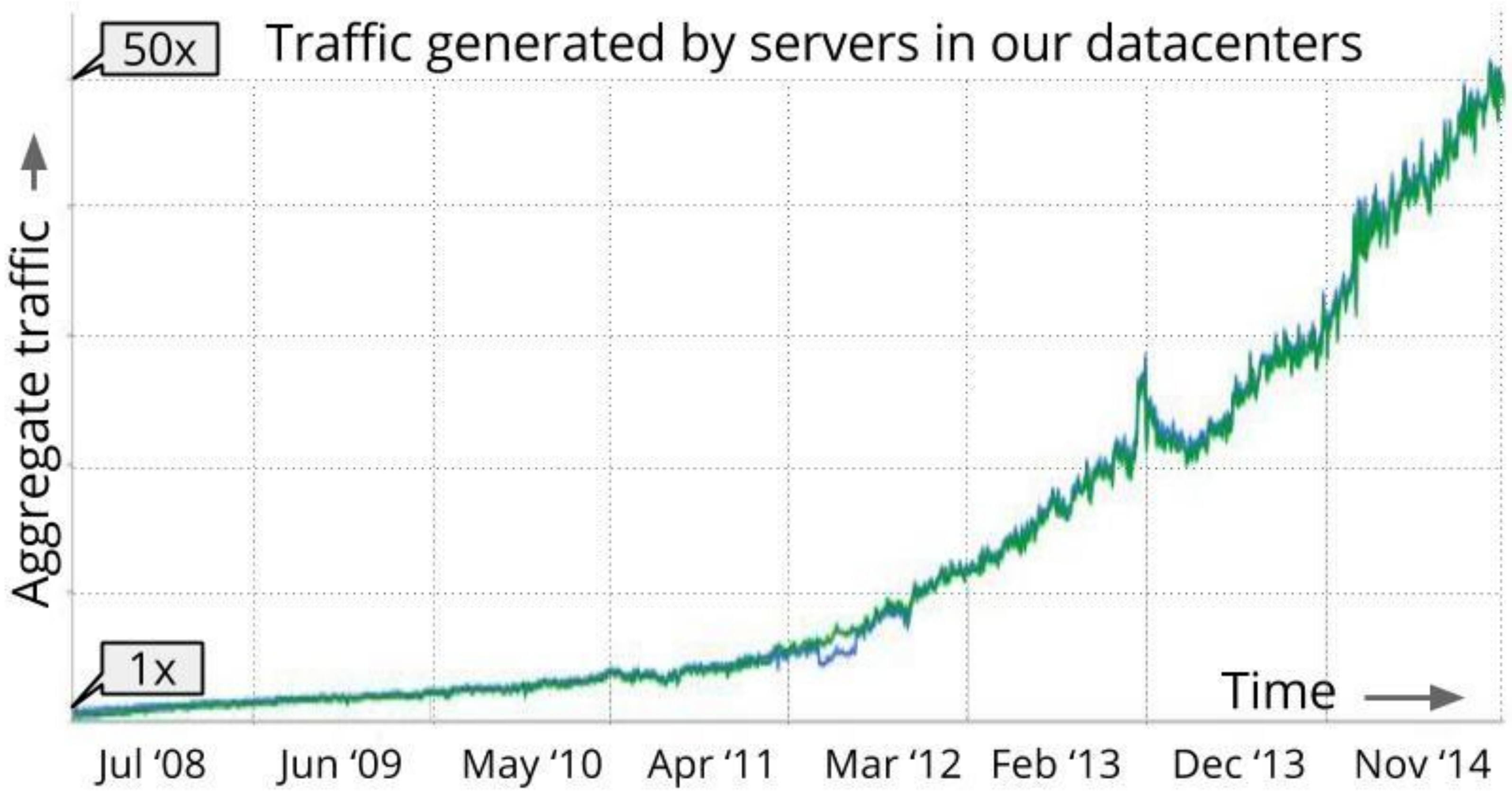


\+ Standard Network Configuration

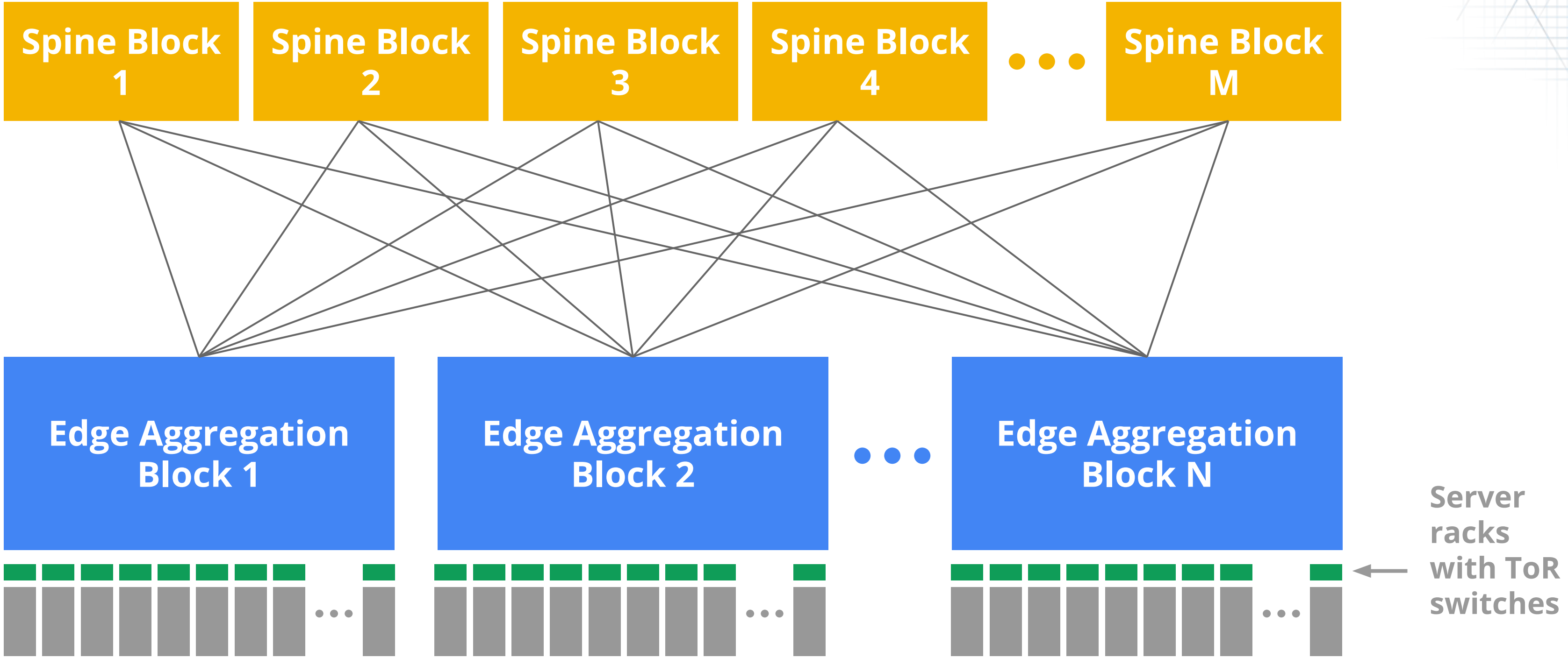
\- Scales to 2 Tbps (limited by the biggest router)

\- Scale up: Forklift cluster when upgrading routers

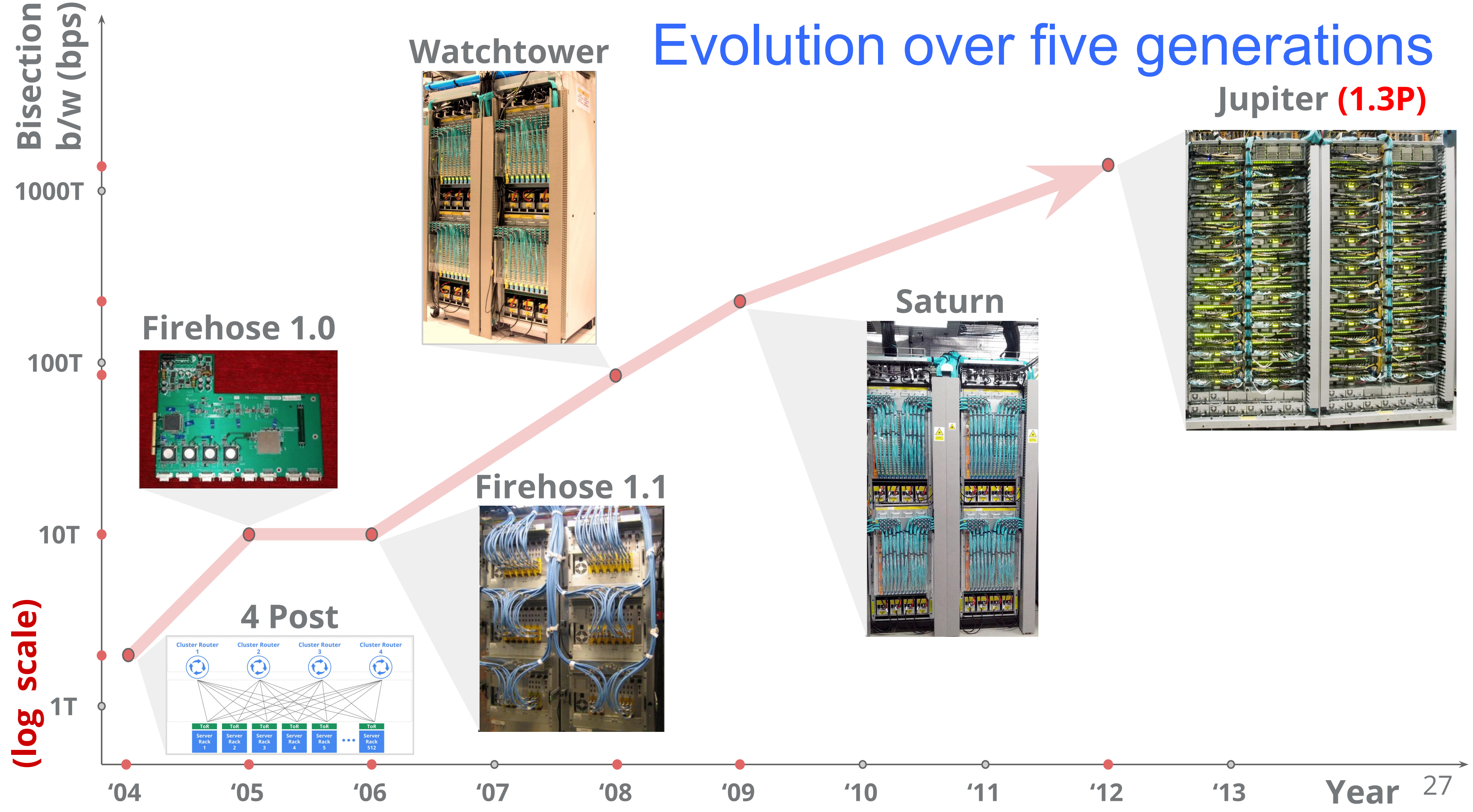
DCN bandwidth growth demanded much more



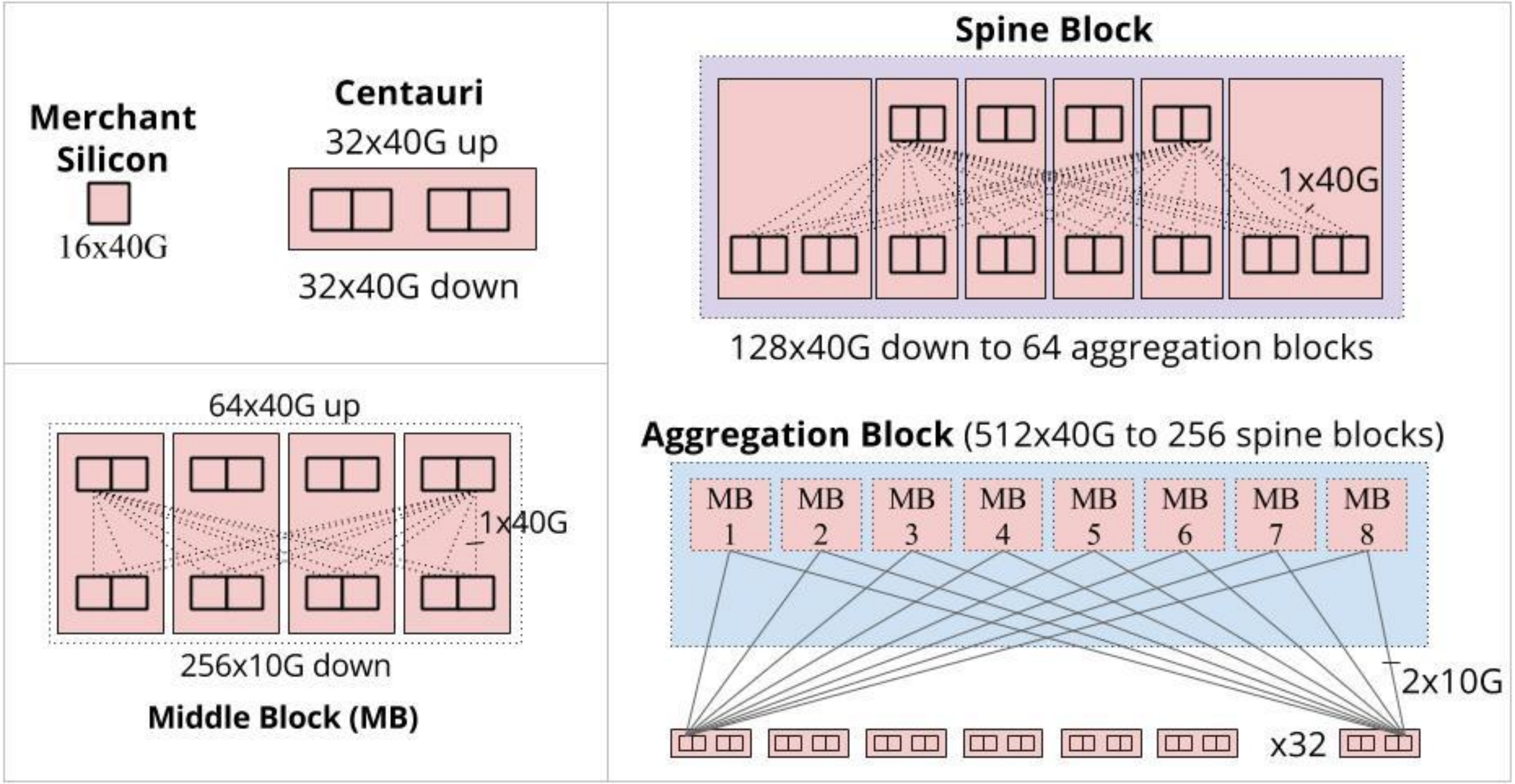
Basic pattern for Google's Clos networks



Evolution over five generations



Jupiter building blocks



40G to hosts; Scales out to 1.3 Pbps

Challenges we faced in building our own solution

- **Topology and deployment**
  - Introducing our network to production
  - Unmanageably high number of cables/fiber
  - Cluster-external burst b/w demand
- **Control and management**
  - Operating at huge scale
  - Routing scalability / routing with massive multipath
  - Interop with external vendor gear
- **Performance and reliability**
  - Small on-chip buffers
  - High availability from cheap/less reliable components

Challenges we faced in building our own solution

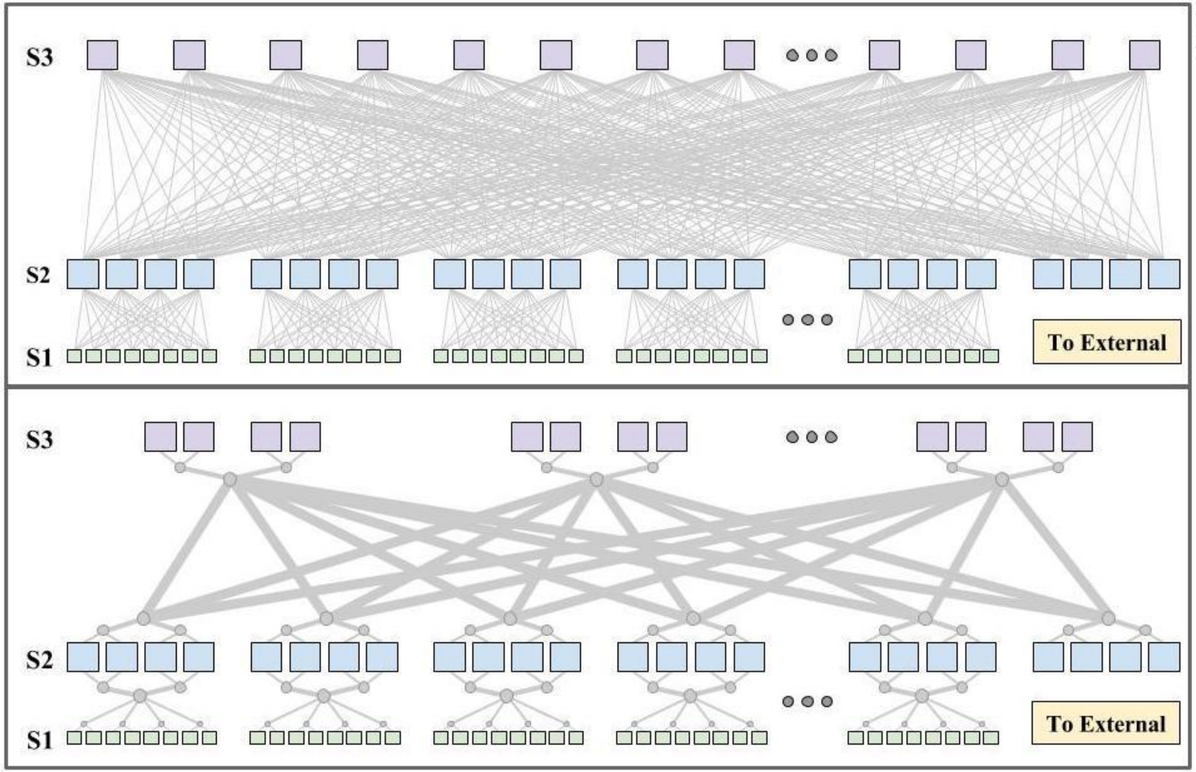
- **Topology and deployment**
  - Introducing our network to production
  - Unmanageably high number of cables/fiber
  - Cluster-external burst b/w demand
- **Control and management**
  - Operating at huge scale
  - Routing scalability / routing with massive multipath
  - Interop with external vendor gear
- **Performance and reliability**
  - Small on-chip buffers
  - High availability from cheap/less reliable components

I only have time for some of these -- see "*Jupiter Rising*" paper for others

Cable management

Without bundling:

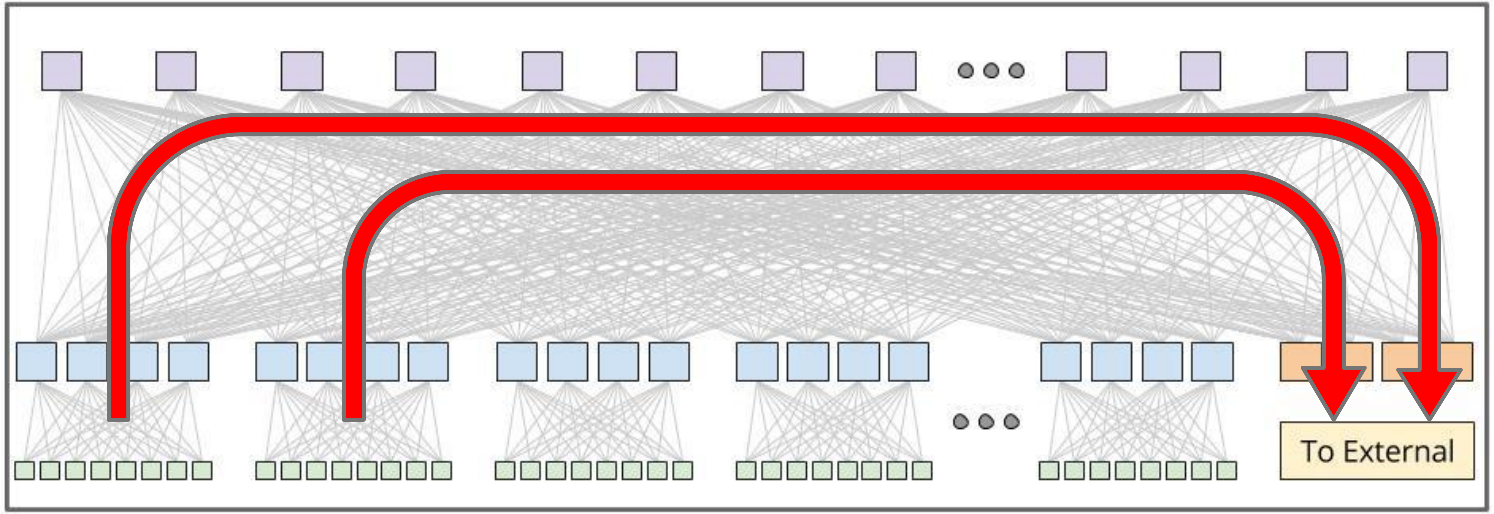
With bundling:



+ Cable bundling saves 40% TCO

+ 10x reduction in *fiber runs* to deploy

Connecting to other networks



+ **Connect externally via border routers**
+ **Massive external burst b/w**
    + **Enables (e.g.) cross-cluster MapReduce**
+ **No need to keep old "cluster routers"**

High reliability from cheap, low-reliability components?

- Exploit the redundancy in multi-path (Clos) network topologies
- Design topologies for diversity
    - e.g., don’t connect a “redundant” pair of links to the same two switch chips
- Implement only the features we need
- Learn, from our outages, how to build a reliable control plane
- Test the control plane in a virtualized testbed (at-scale testing is expensive!)

Wide-area networks

Wide Area Networks are different

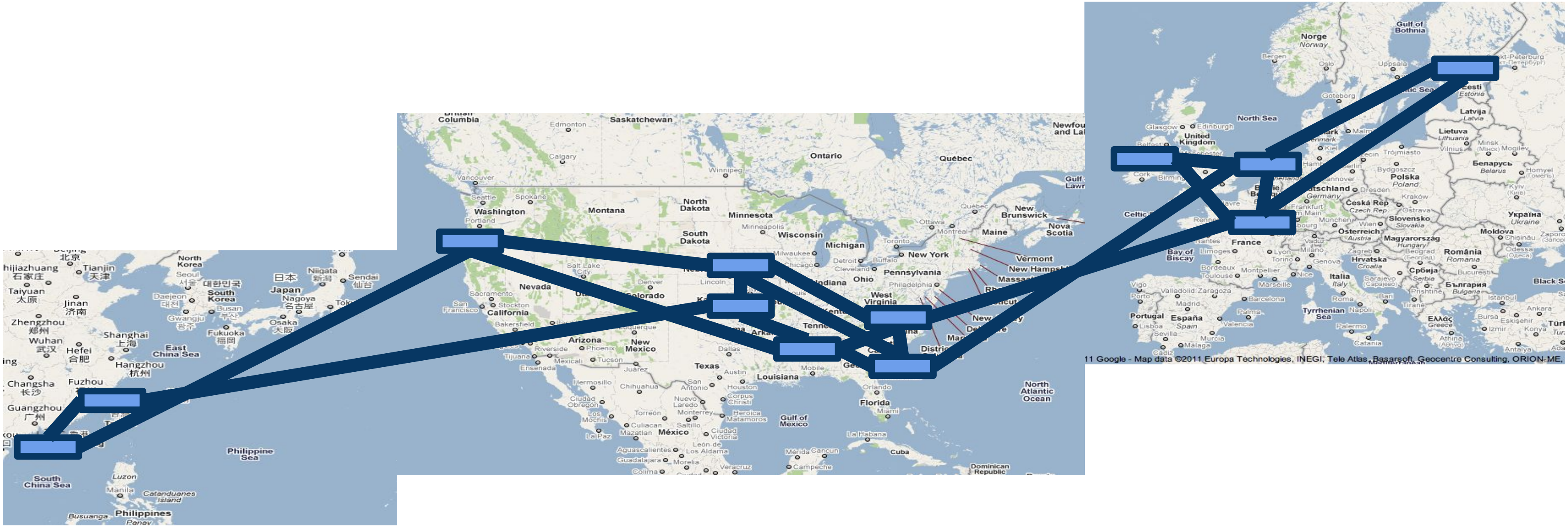
- All kinds of clouds (IaaS/PaaS/SaaS) need WAN connectivity
- WANs face challenges that data-center networks do not:
  - WAN links are much more expensive
  - WAN links are more vulnerable
    - Backhoes, farmers, sharks, drunken hunters, spy agencies, etc.
  - WAN topologies are highly irregular
    - Constrained by geography
    - Some paths are much more expensive or vulnerable than others
  - WANs have lots of “edge” issues
    - Boundaries between routing domains
    - Boundaries between owners
    - Equipment often in remote and/or constrained locations (“POPs”)
    - Every link and every port might need a unique configuration

Google maintains two distinct WAN networks

- User-facing WAN:
    - Connects our datacenters to Internet peers (ISPs)
        - Must interface with a wide variety of equipment from many vendors
        - Connects to POPs all over the world
    - Connects our datacenters to our own world-wide edge caches (CDN)
    - Carries requests from users, and responses to them
- Datacenter-to-Datacenter WAN (“B4”):
    - Connects only between our datacenters
    - Used for distributed storage, copying large datasets, replication of user data, etc.

I’ll tell you about B4, because it’s the more technically interesting one.

B4's world-wide scope



B4 design goals

B4's design is motivated by somewhat different goals (vs. our other WAN):

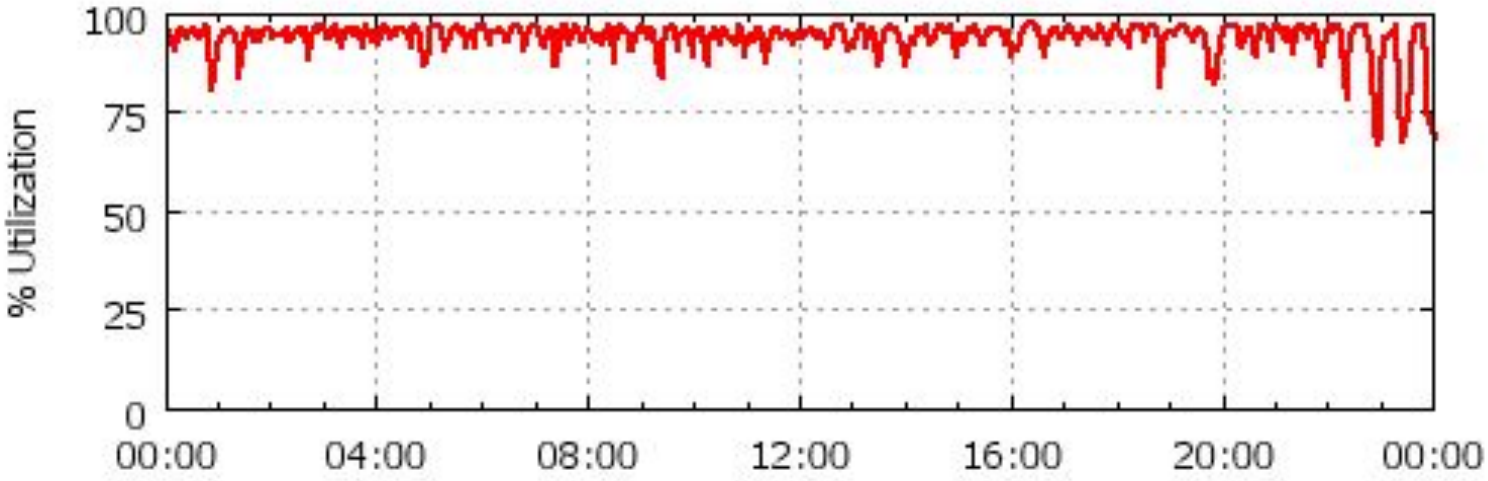
- Large bandwidth demands, but tolerant of occasional reductions
- Relatively few sites
- We control all of the end-point applications and operating systems
- We prefer to lower costs, instead of using overprovisioning to achieve:
    - Low packet drop rates
    - Low rates of link failures

B4 design principles

Traditional approach	B4 Approach
Traditional routers	Simpler switches, + SDN controllers
Manage 1000s of individual boxes	Manage the network as a whole
Distributed, non-deterministic routing protocols	Logically-centralized control, with traffic engineering
All packets are equally important	Allocate resources based on application-level priorities
TCP flows regulated by “fair share” mechanisms	Measure demands, and shape flows, at the endpoints

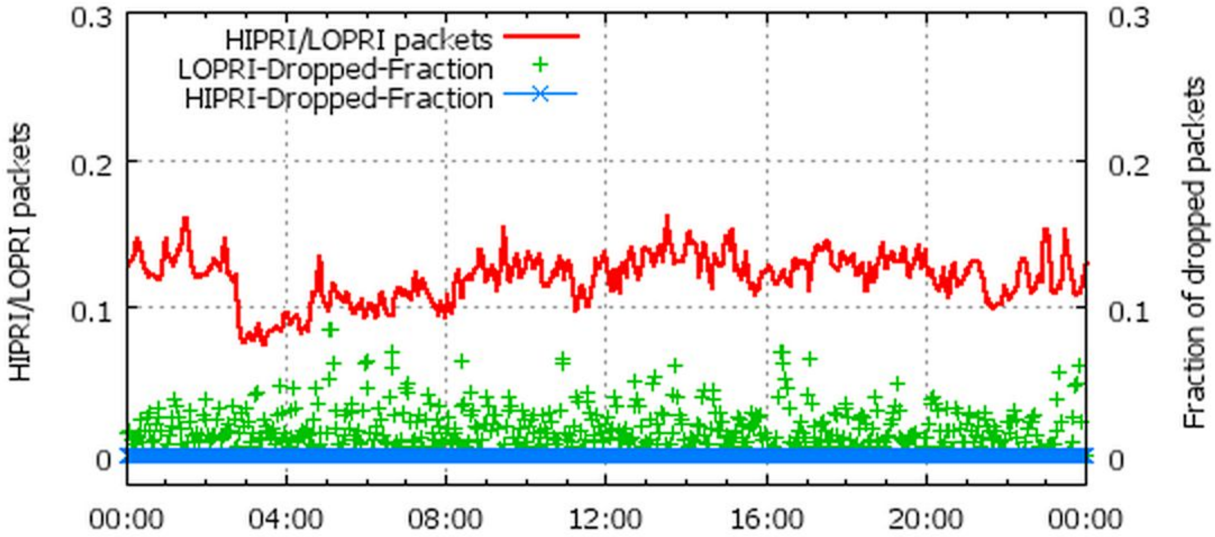
This allows us to run the network at 100% utilization

- 100% utilization? That's crazy!
    - In a traditional network, this would lead to horrible packet loss
- But it's really cost-effective, and so we made it work



How to live with 100% utilization

- Treat high-priority traffic and low-priority traffic differently
  - Hi-pri: strive for zero loss, and shortest-path (lowest-latency) routing
  - Lo-pri: applications must tolerate loss, latency, and variable capacity
  - Most of the traffic on B4 is lo-pri



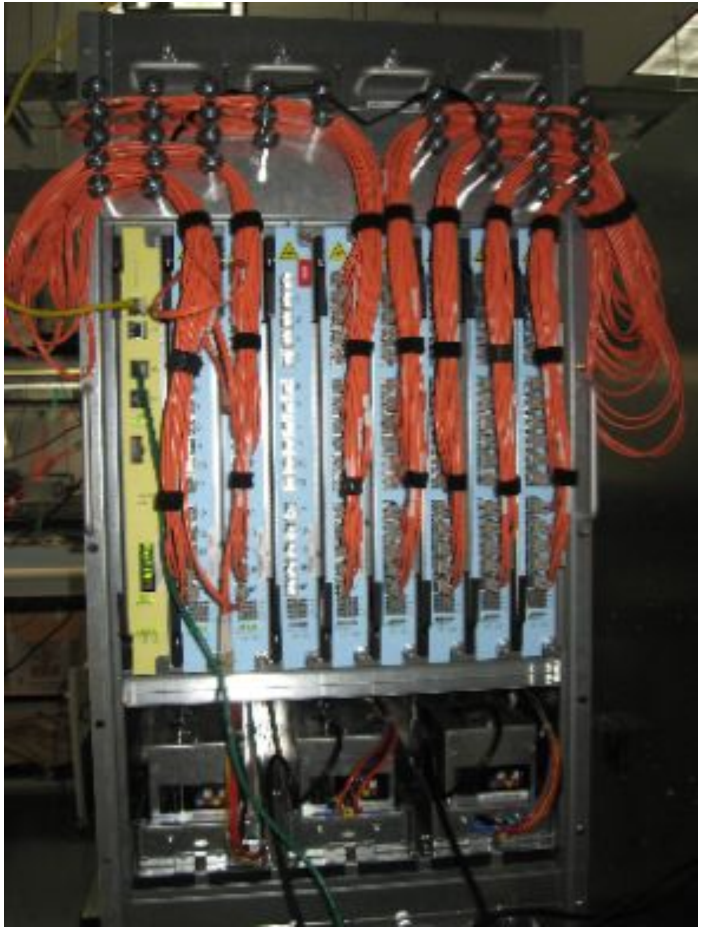
Google

Building the B4 network

- Our own SDN-controllable switches, built from merchant silicon
  - Far fewer features than commercial routers -- so much cheaper
    - Smaller buffers, smaller routing tables, less HW-based fault tolerance
  - Shortest-path routing for hi-pri traffic
  - Tunnelled, traffic-engineered routing for lo-pri traffic
- SDN controllers that carefully manage the traffic-engineered tunnels
  - See SIGCOMM 2013 paper for details on the TE algorithms and protocols
- “Bandwidth Enforcer” to shape traffic flows at all end hosts
  - See SIGCOMM 2015 paper for more details on BwE
- Specialized applications that can tolerate high loss rates and latency
  - and that can adapt transmission rates to variable bandwidth capacity

B4 hardware

- Built from merchant silicon
  - 100s of ports of nonblocking 10GE
- OpenFlow support
- Open-source routing stacks for BGP, ISIS
- Multiple chassis per site
  - Fault tolerance through redundancy
  - Scales to multiple Tbps



A word from our sponsor ...

Short links for jobs

Internships:

- PhD SW Eng students: deadline Feb 7 2020
- Undergrad/MS SW Eng: deadline Dec 13 2019
- Hardware Eng: deadline January 31 2020
- Freshmen/soph “STEP”: deadline was Nov 1
- Use google.com/students to find out more

Full-time jobs:

- g.co/research/networks for our team specifically
- careers.google.com more generally

If you apply: please let me know (mogul@google.com) so I can keep track of you

Questions? stanfordstudents@google.com