Advanced Network Programming

Cloud Networking

Lin Wang

Fall 2020, Period 1

Part 2: network infrastructure

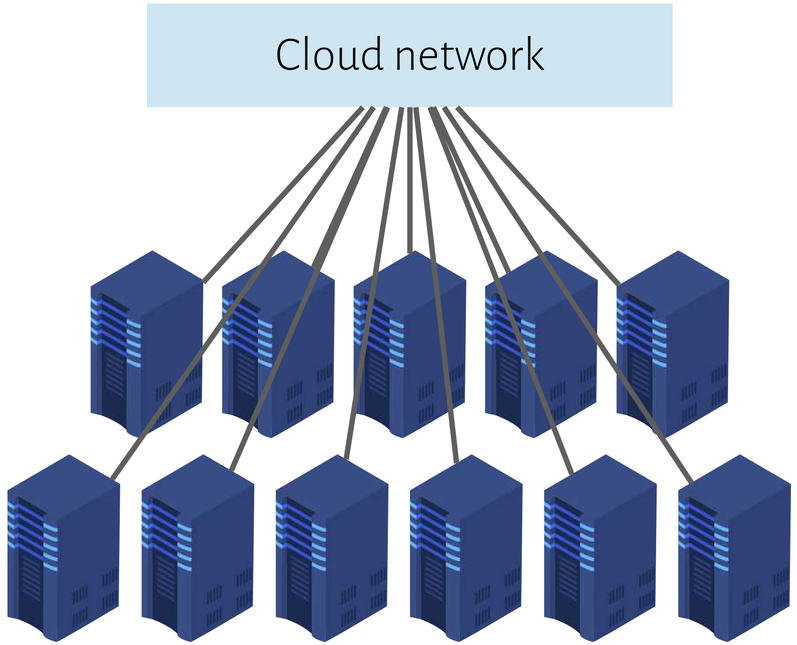
Lecture 7: Network forwarding and routing

Lecture 8: Software defined networking

Lecture 9: Programmable data plane

Lecture 10: Cloud networking

Lecture 11: Beyond networking



Cloud computing

Elastic resources

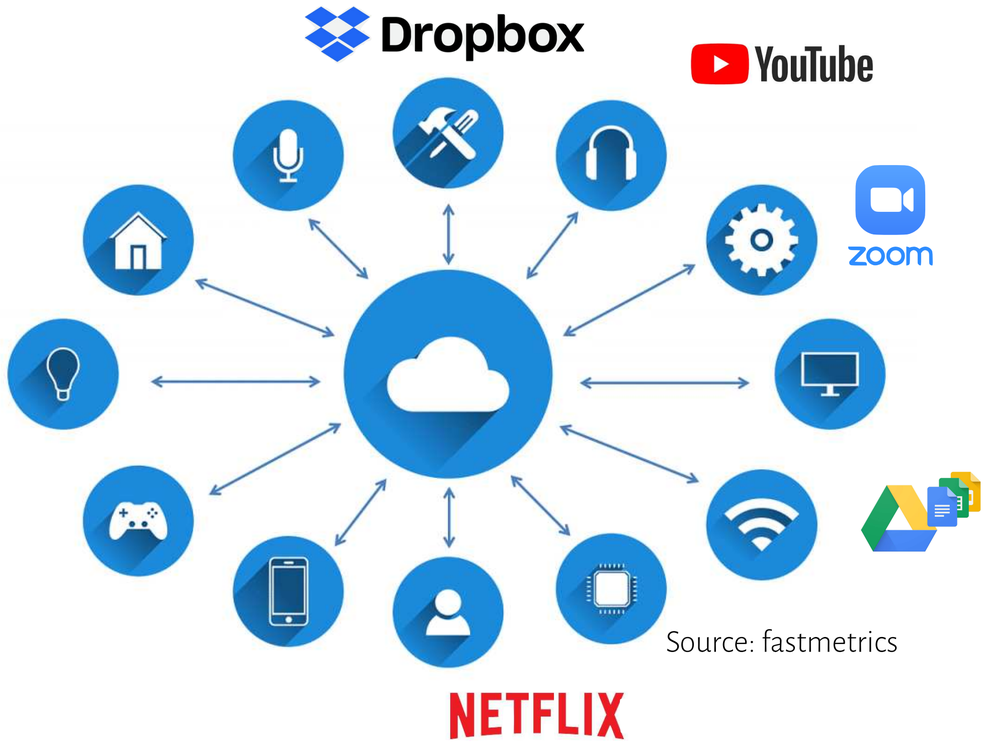
- Expand and contract resources
- Pay-per-use, infrastructure on demand

Multi-tenancy

- Multiple independent users, resource isolation
- Amortize the cost of the shared infrastructure

Flexible service management

- Resilience: isolate failures of server and storage
- Workload migration: move work to other locations



Source: fastmetrics

What is behind cloud computing?

Large-scale data centers

Data center distribution in Europe

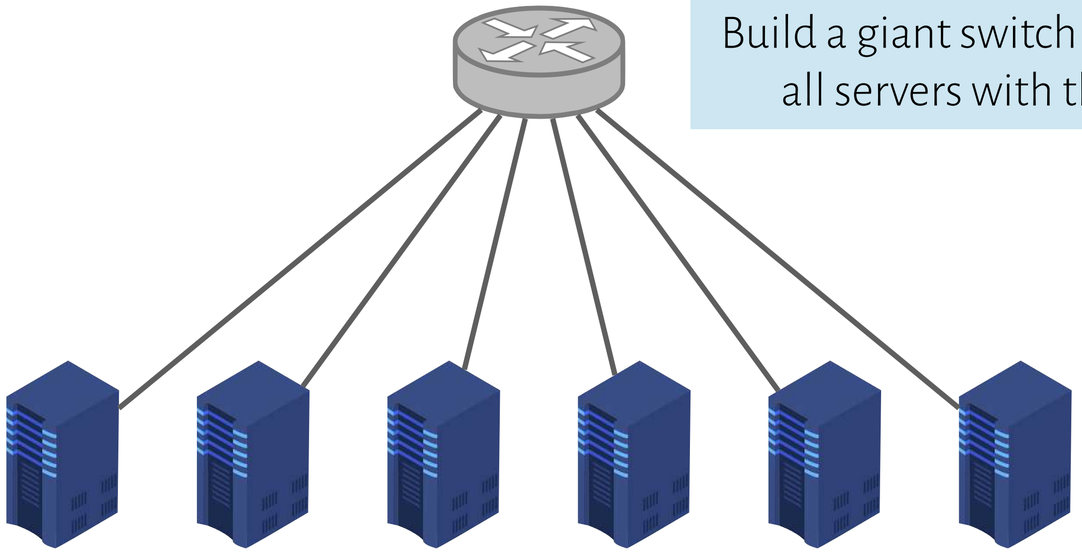
Equinix’s AM4 data center @ Science Park

How does a data center look inside?

Equinix's AM4 data center @ Science Park

Well organized & interconnected racks of servers!

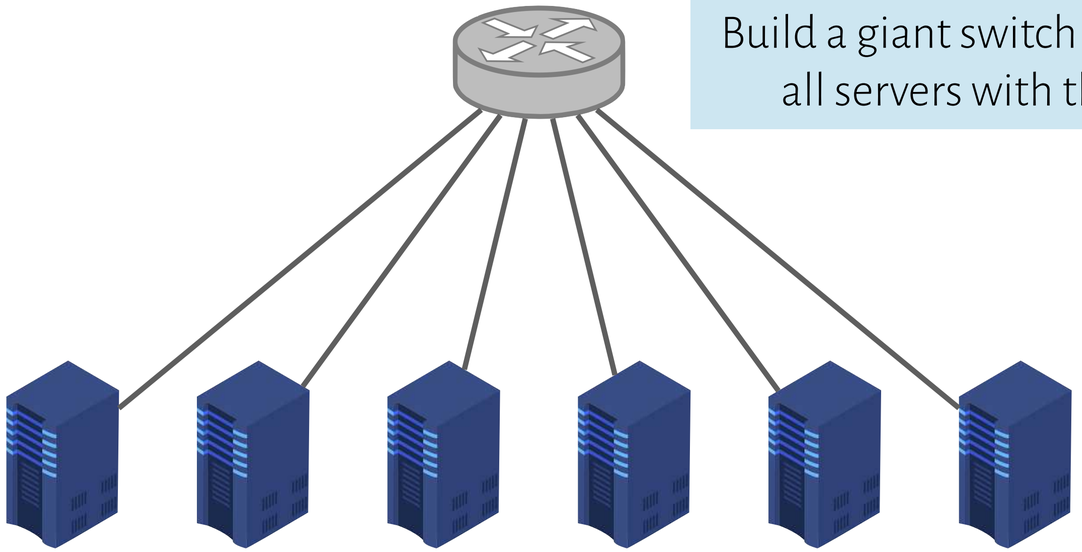
How to connect these servers in a data center?



Build a giant switch and connect all servers with the switch

What problems can you think of with such a design?

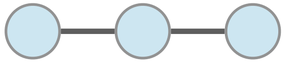
How to connect these servers in a data center?



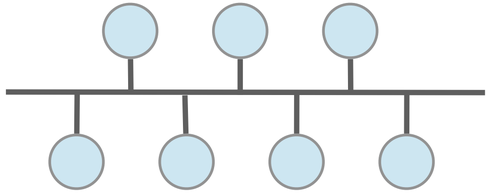
Build a giant switch and connect all servers with the switch

Switches have a **limited port density** and cannot scale to huge number of servers (recall the switching fabric design)

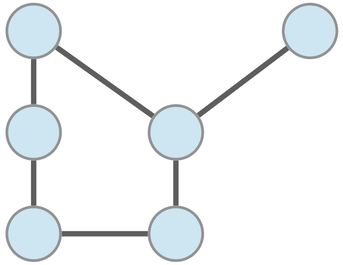
A dedicated network for the data center



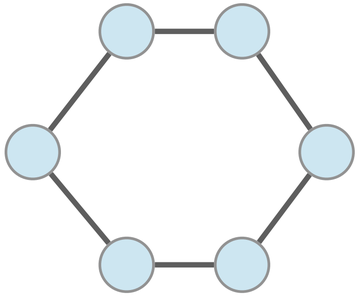
Line



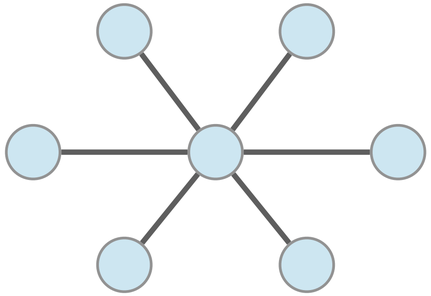
Bus



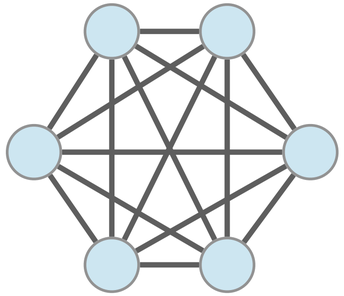
Mesh



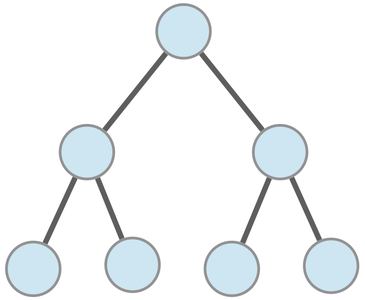
Ring



Star



Fully connected

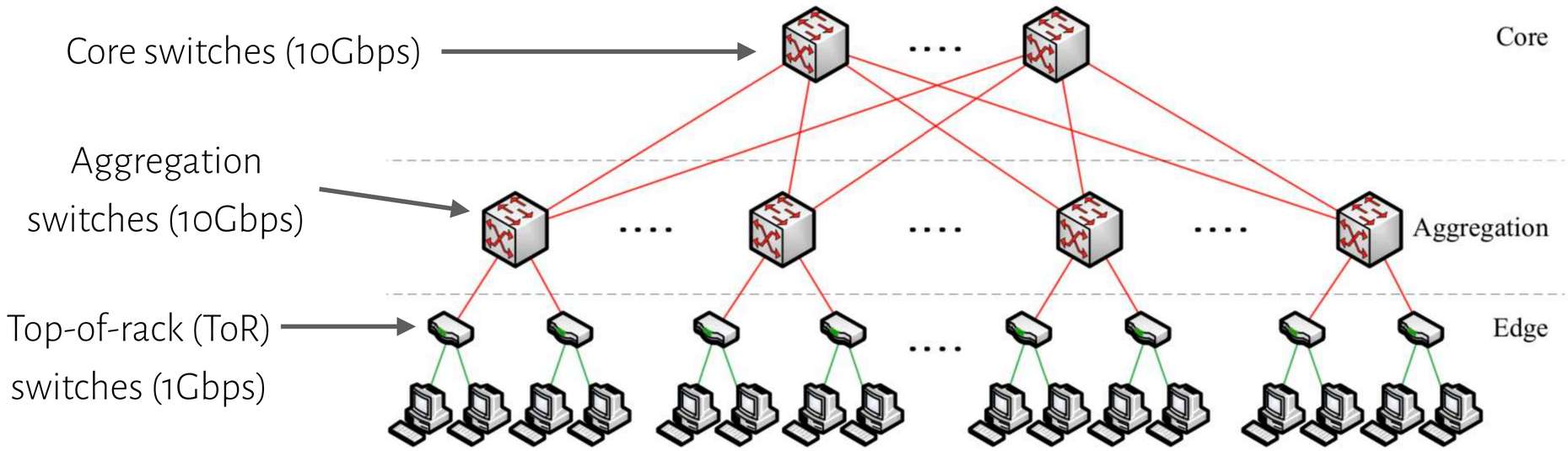


Tree

Tradeoff between connectivity and complexity

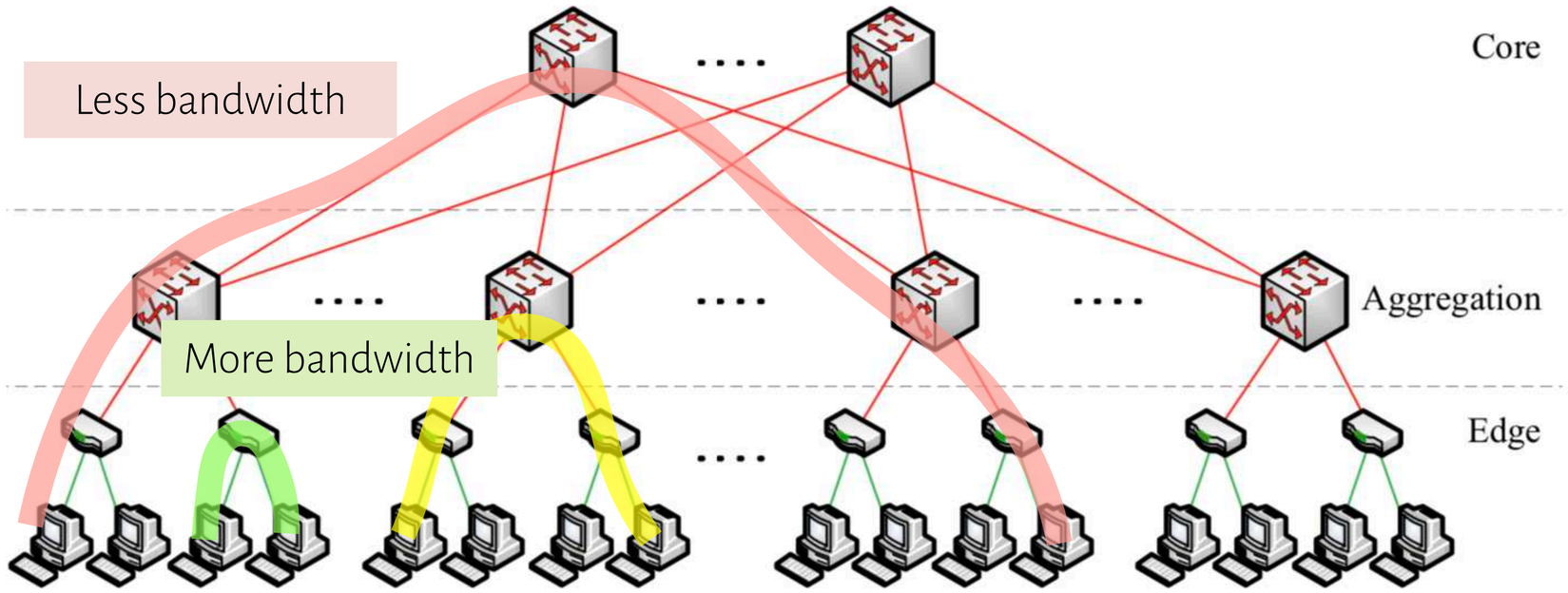
A typical data center network architecture

A 3-tier tree architecture



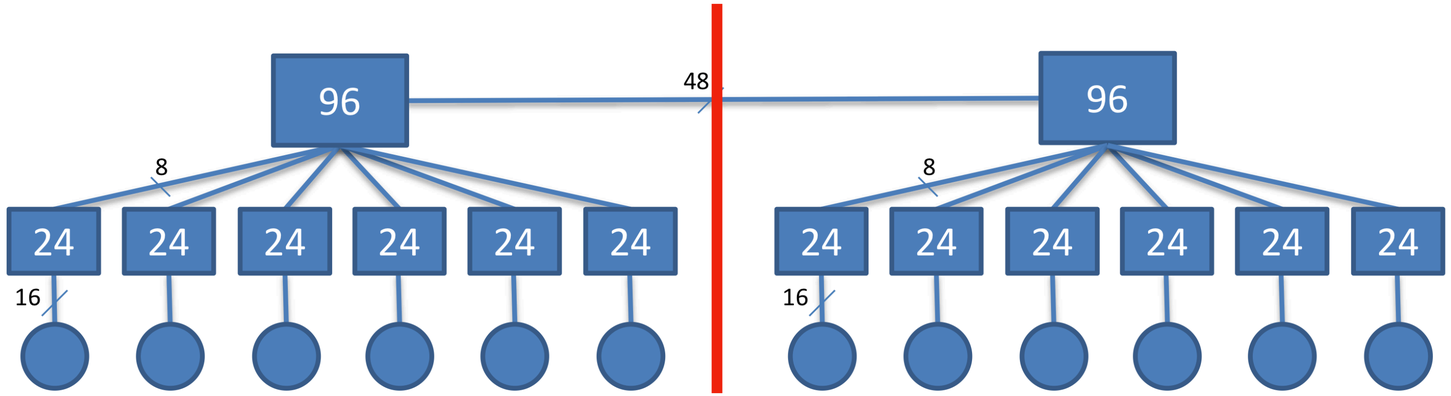
What if ToR switches go for 10Gbps or beyond?

Bottleneck in tree networks



How to quantitatively measure the connectivity?

Bisection bandwidth



Bisection width	The minimum number of links cut to divide the network into two halves
Bisection bandwidth	The total bandwidth of the above links
Full bisection bandwidth	One half of nodes can communicate simultaneously with the other half of nodes

Oversubscription

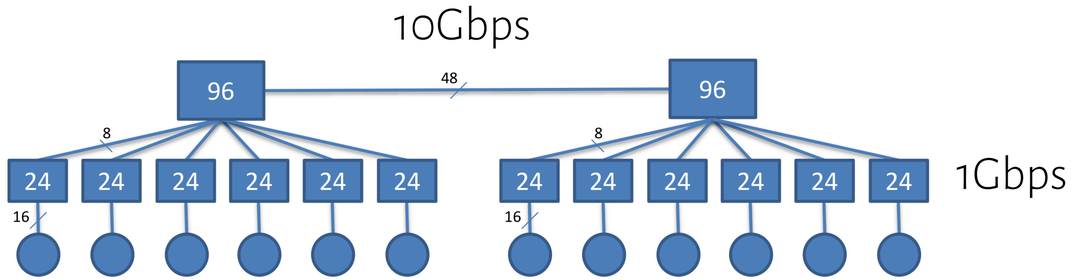
Definition

- Ratio of worst-case required aggregate bandwidth among end-hosts to the total bisection bandwidth of the network topology
- Ability of hosts to fully utilize its uplink capabilities

Examples

- 1:1 → All hosts can use full uplink capacity
- 5:1 → Only 20% of host bandwidth may be available

Typical data center subscription ratio is 2.5:1 to 8:1



What is the oversubscription ratio of the above topology?

Questions?

Motivation for data center network design

Commoditization in the data center

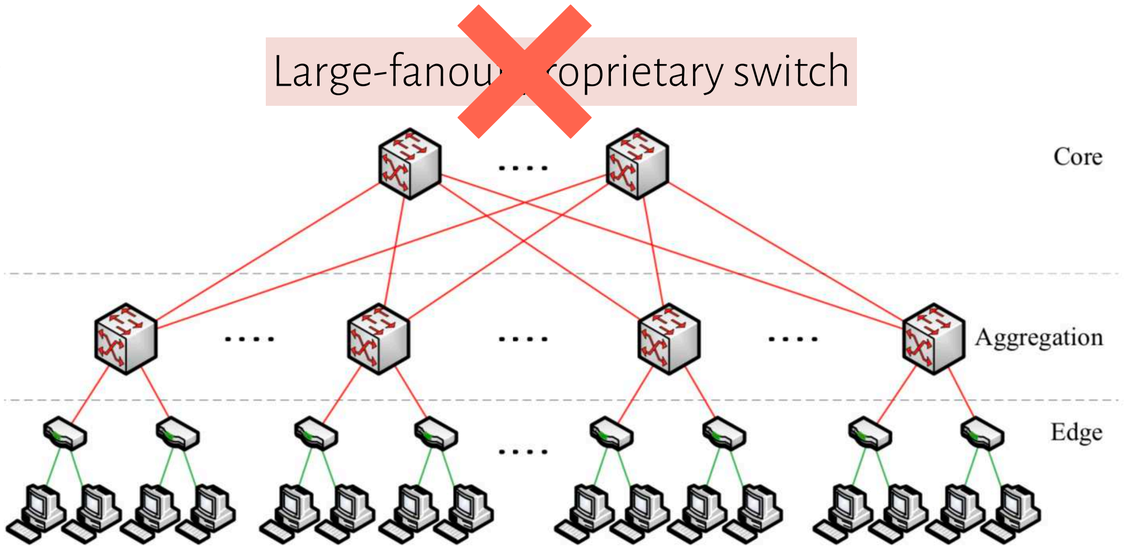
- Inexpensive, commodity servers and storage devices
- But the network is still highly specialized (using large-fanout proprietary switches)

Data center is **not** a "small Internet"

- One admin domain, not adversarial, limited policy routing, etc…

Bandwidth is often the bottleneck

- Data-intensive workloads (big data, graph processing, machine learning)



Fat-tree

Expand the tree topology with a “fat” root to increase the root connectivity

A Scalable, Commodity Data Center Network Architecture

Mohammad Al-Fares
malfares@cs.ucsd.edu

Alexander Loukissas
aloukiss@cs.ucsd.edu

Amin Vahdat
vahdat@cs.ucsd.edu

Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0404

ABSTRACT

Today’s data centers may contain tens of thousands of computers with significant aggregate bandwidth requirements. The network architecture typically consists of a tree of routing and switching elements with progressively more specialized and expensive equipment moving up the network hierarchy. Unfortunately, even when deploying the highest-end IP switches/routers, resulting topologies may only support 50% of the aggregate bandwidth available at the edge of the network, while still incurring tremendous cost. Non-uniform bandwidth among data center nodes complicates application design and limits overall system performance.

In this paper, we show how to leverage largely commodity Ethernet switches to support the full aggregate bandwidth of clusters

institutions and thousand-node clusters are increasingly common in universities, research labs, and companies. Important applications classes include scientific computing, financial analysis, data analysis and warehousing, and large-scale network services.

Today, the principle bottleneck in large-scale clusters is often inter-node communication bandwidth. Many applications must exchange information with remote nodes to proceed with their local computation. For example, MapReduce [12] must perform significant data shuffling to transport the output of its map phase before proceeding with its reduce phase. Applications running on cluster-based file systems [18, 28, 13, 26] often require remote-node access before proceeding with their I/O operations. A query to a web search engine often requires parallel communication with ev-

ACM SIGCOMM 2008

Fat-tree: design goals

Scalable interconnection bandwidth

- **Full bisection bandwidth** (1:1 oversubscription ratio) between all pairs of hosts

Economies-of-scale

- Price/port constant with number of hosts
- Must leverage commodity merchant silicon

Compatibility

- Support Ethernet and IP without host modifications

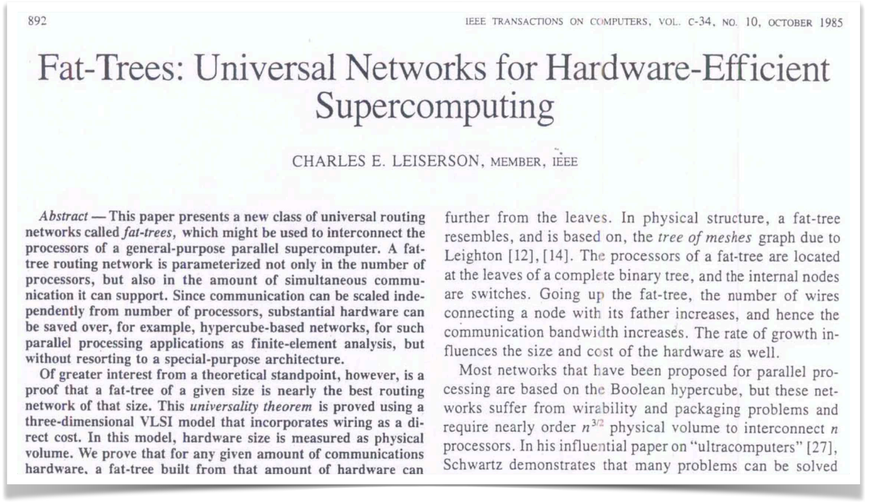
Easy management

- Modular design, avoid manual management

Fat-tree topology

A special instance of the **Clos topology**

- Clos networks are originally designed for telephone switches
- Emulate a single huge switch with many smaller switches
- Proposed in 1953 by Charles Klos
- Fat-tree was proposed by Leiserson in 1985



892 IEEE TRANSACTIONS ON COMPUTERS, VOL. C-34, NO. 10, OCTOBER 1985

Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing

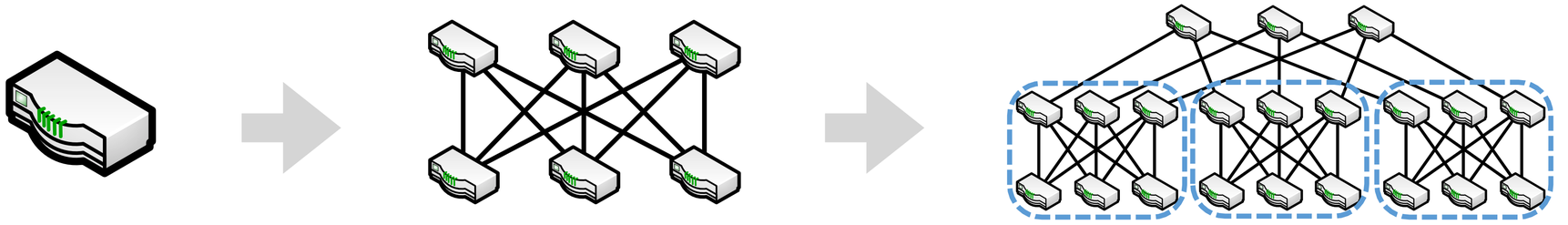
CHARLES E. LEISERSON, MEMBER, IEEE

Abstract — This paper presents a new class of universal routing networks called *fat-trees*, which might be used to interconnect the processors of a general-purpose parallel supercomputer. A fat-tree routing network is parameterized not only in the number of processors, but also in the amount of simultaneous communication it can support. Since communication can be scaled independently from number of processors, substantial hardware can be saved over, for example, hypercube-based networks, for such parallel processing applications as finite-element analysis, but without resorting to a special-purpose architecture.

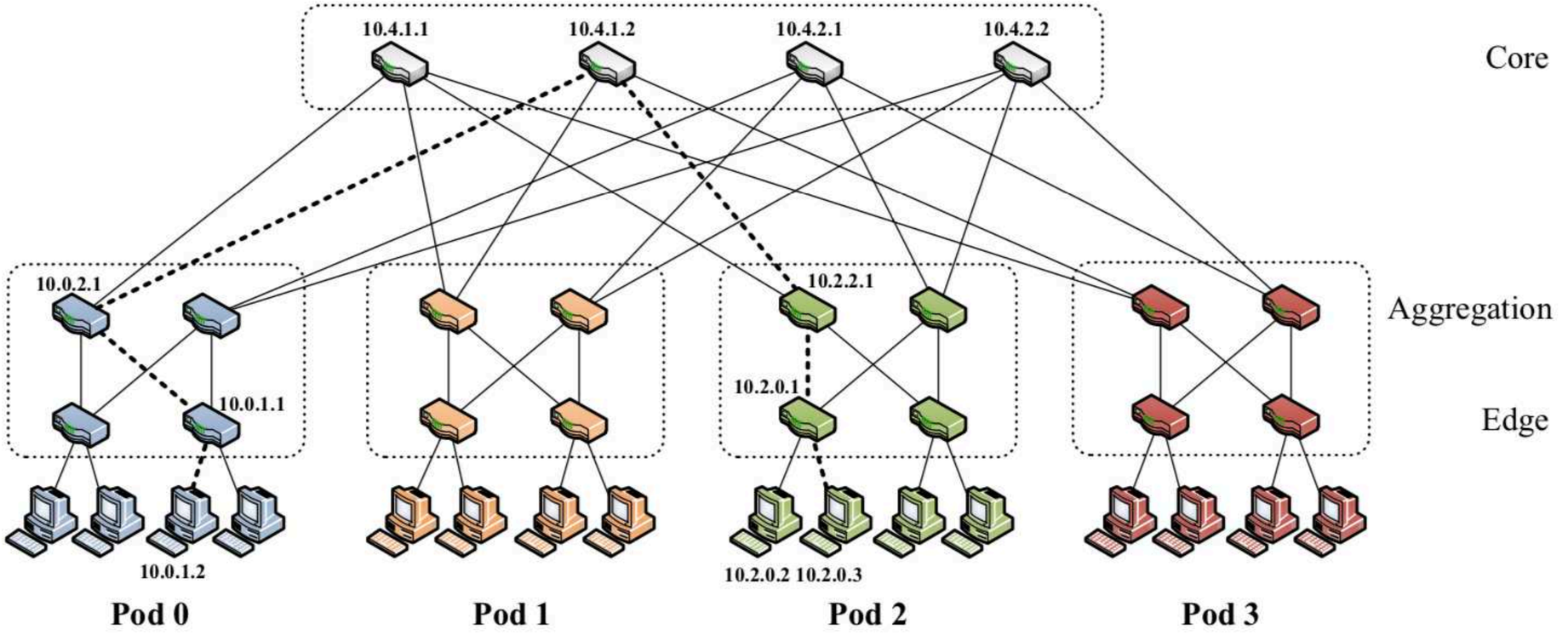
Of greater interest from a theoretical standpoint, however, is a proof that a fat-tree of a given size is nearly the best routing network of that size. This *universality theorem* is proved using a three-dimensional VLSI model that incorporates wiring as a direct cost. In this model, hardware size is measured as physical volume. We prove that for any given amount of communications hardware, a fat-tree built from that amount of hardware can further from the leaves. In physical structure, a fat-tree resembles, and is based on, the *tree of meshes* graph due to Leighton [12], [14]. The processors of a fat-tree are located at the leaves of a complete binary tree, and the internal nodes are switches. Going up the fat-tree, the number of wires connecting a node with its father increases, and hence the communication bandwidth increases. The rate of growth influences the size and cost of the hardware as well.

Most networks that have been proposed for parallel processing are based on the Boolean hypercube, but these networks suffer from wirability and packaging problems and require nearly order $n^{3/2}$ physical volume to interconnect n processors. In his influential paper on "ultracomputers" [27], Schwartz demonstrates that many problems can be solved

IEEE TOC 1985



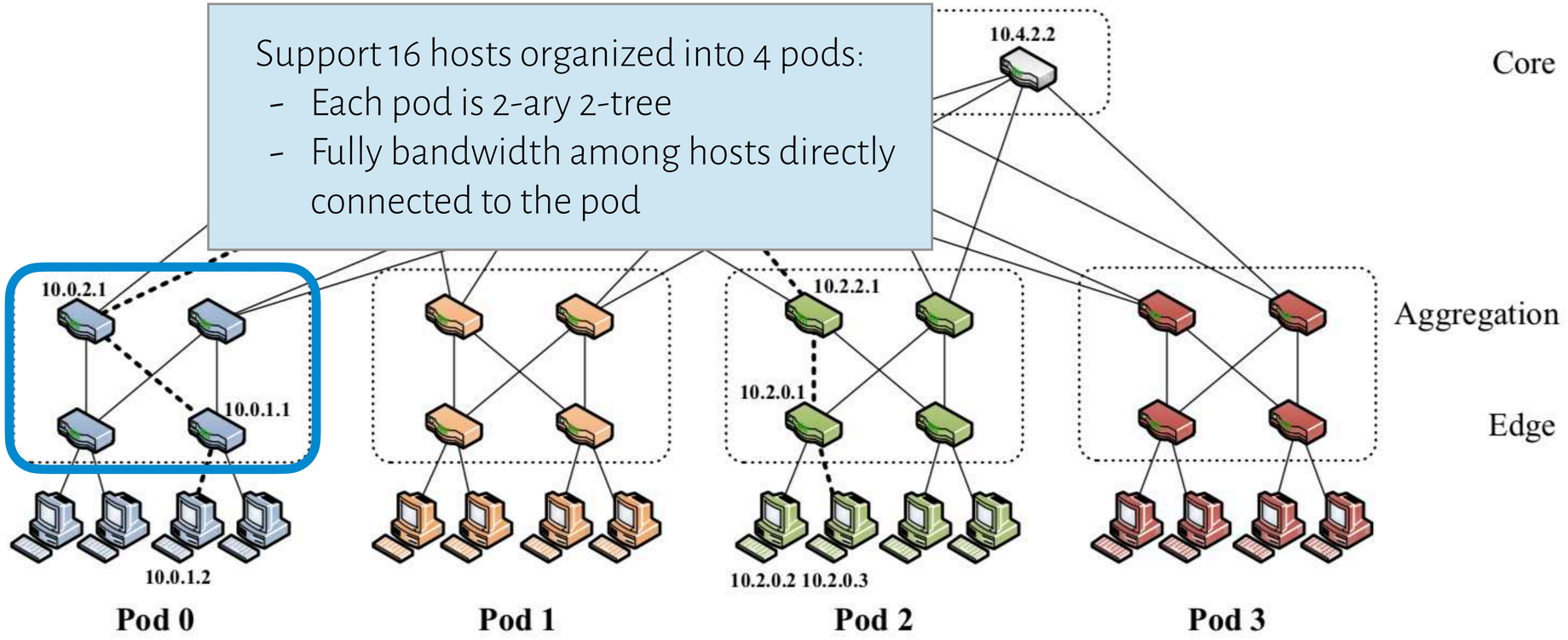
Fat-tree example



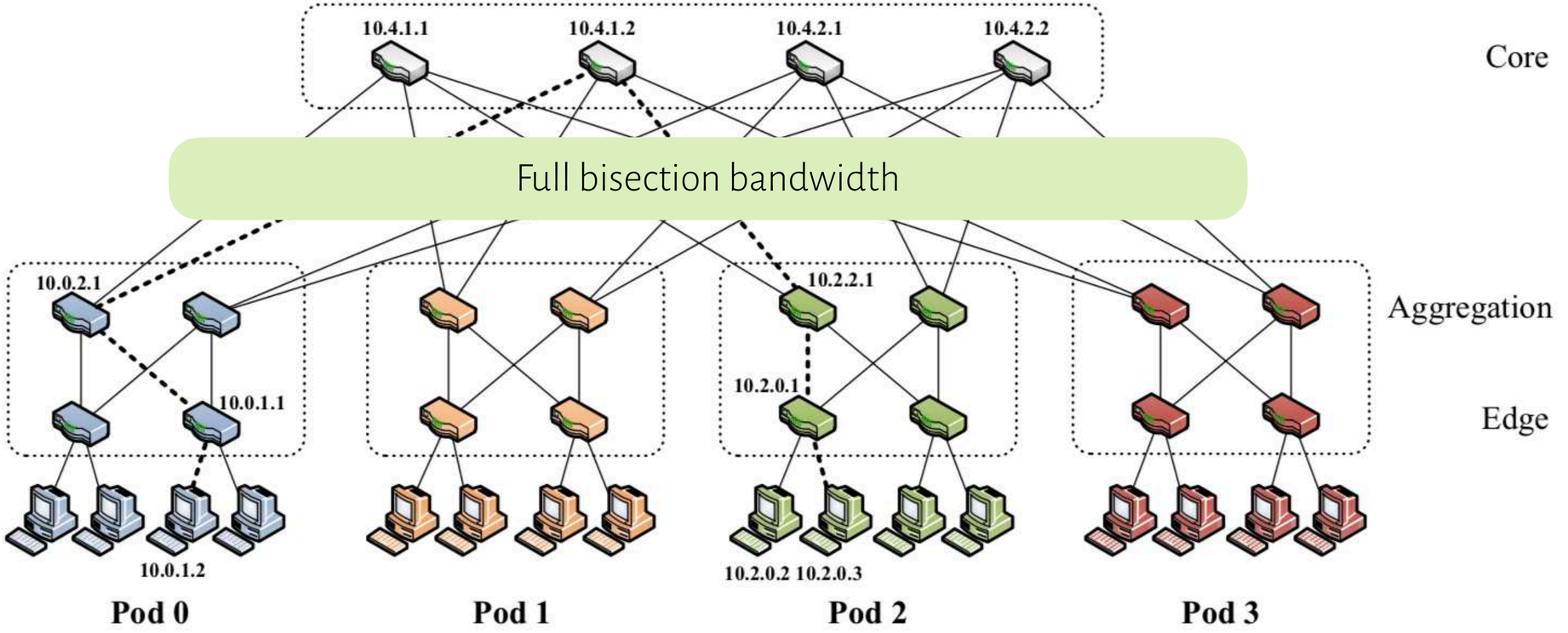
A fat-tree network built from 4-port **identical** switches

Fat-tree example

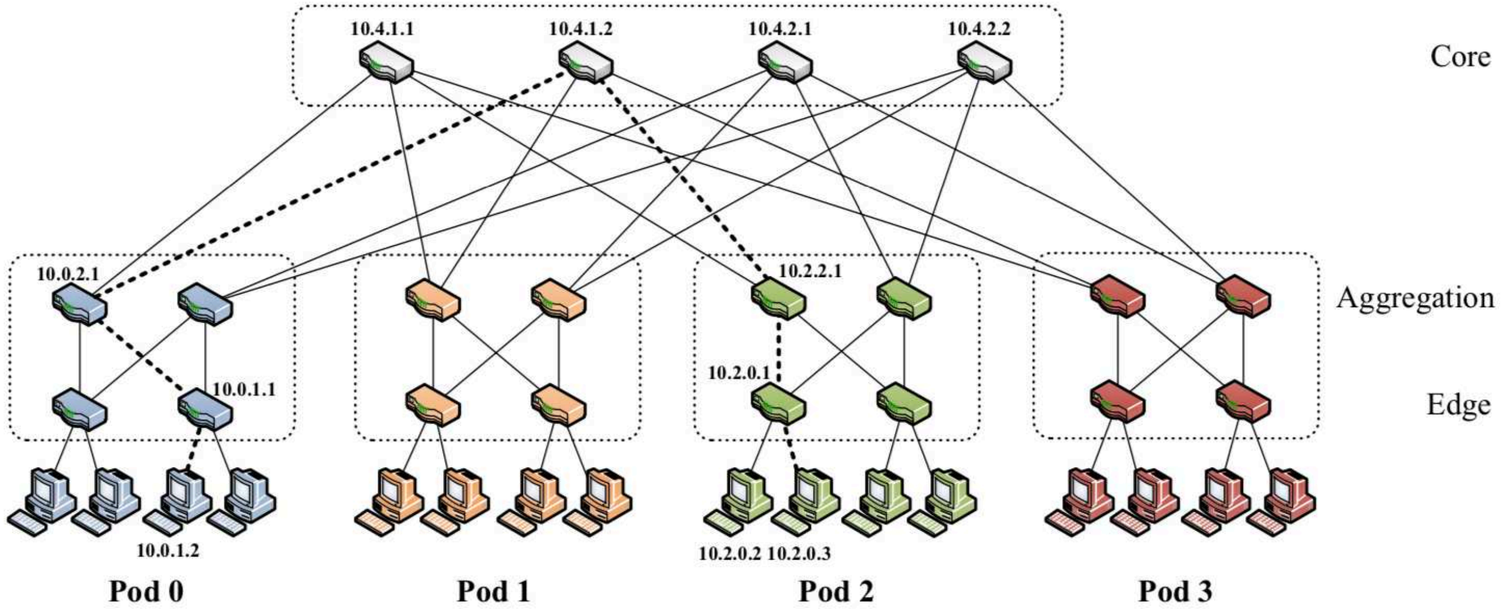
Support 16 hosts organized into 4 pods:
- Each pod is 2-ary 2-tree
- Fully bandwidth among hosts directly connected to the pod



Fat-tree example

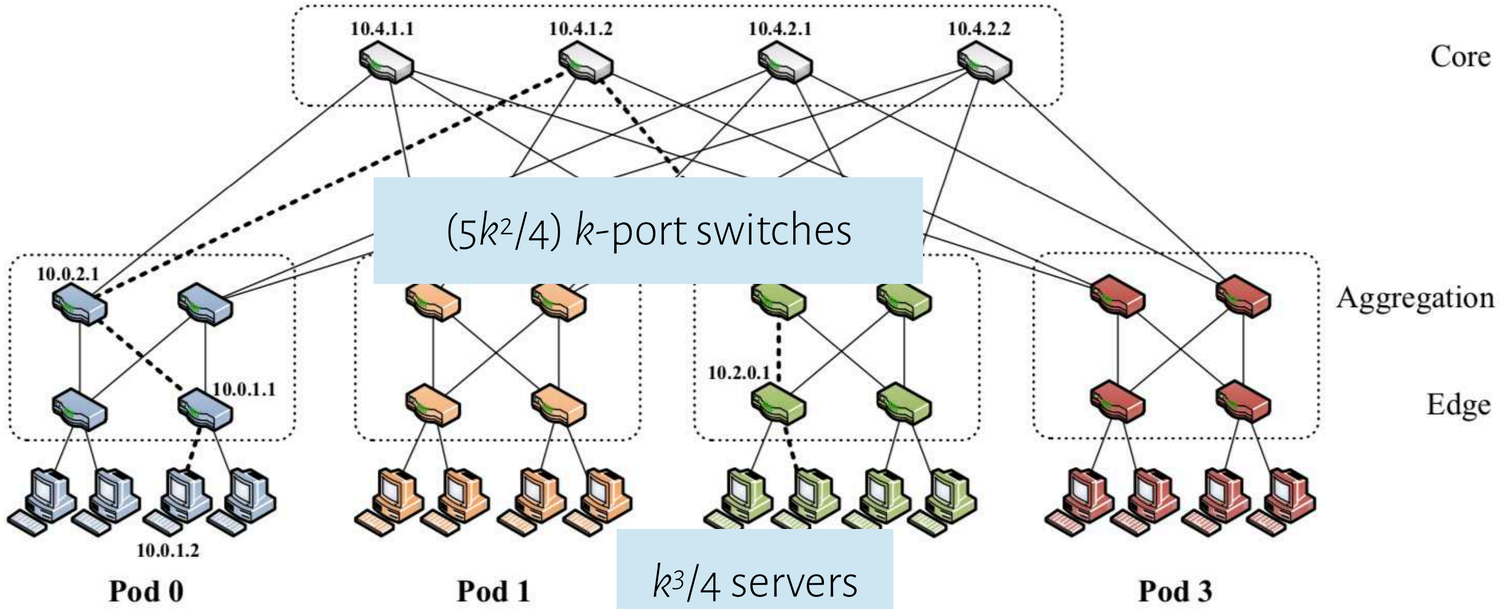


Fat-tree scalability



Suppose we use ***k*-port switches**, **how many servers** can we interconnect with fat-tree, and **how many switches** are needed?

Fat-tree scalability



$(5k^2/4)$ k-port switches

$k^3/4$ servers

Fat-tree can scale to any link capacity at the edge: 10Gbps, 40Gbps, 100Gbps, …

Why this has not been done before?

Recall fat-tree was proposed in 1985!!

Needs to be **backward compatible** with IP/Ethernet

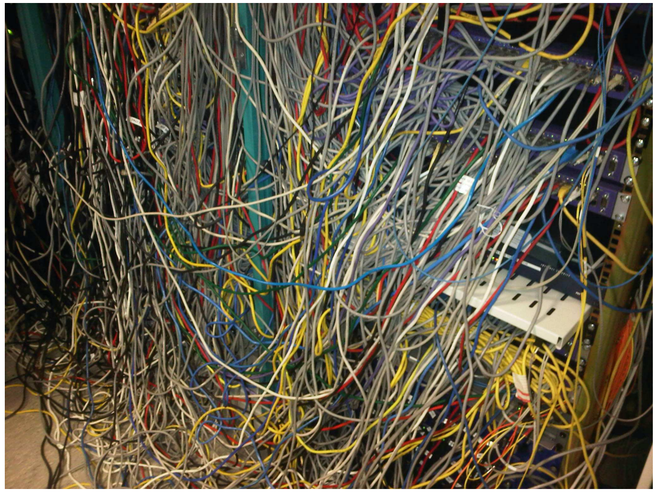
- Existing routing and forwarding protocols do not work for fat-tree
- Scalability challenges with millions of end points

Management

- Thousands of individual elements that must be programmed individually

Cabling explosion at each level of fat-tree

- Tens of thousands of cables running across the data center

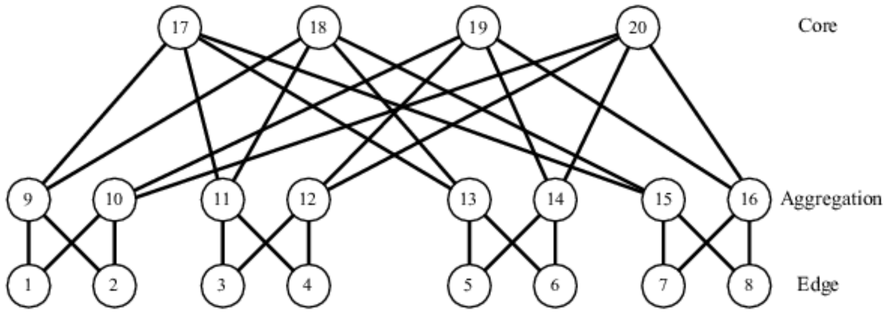


Challenges with fat-tree

Backward compatible with IP/Ethernet

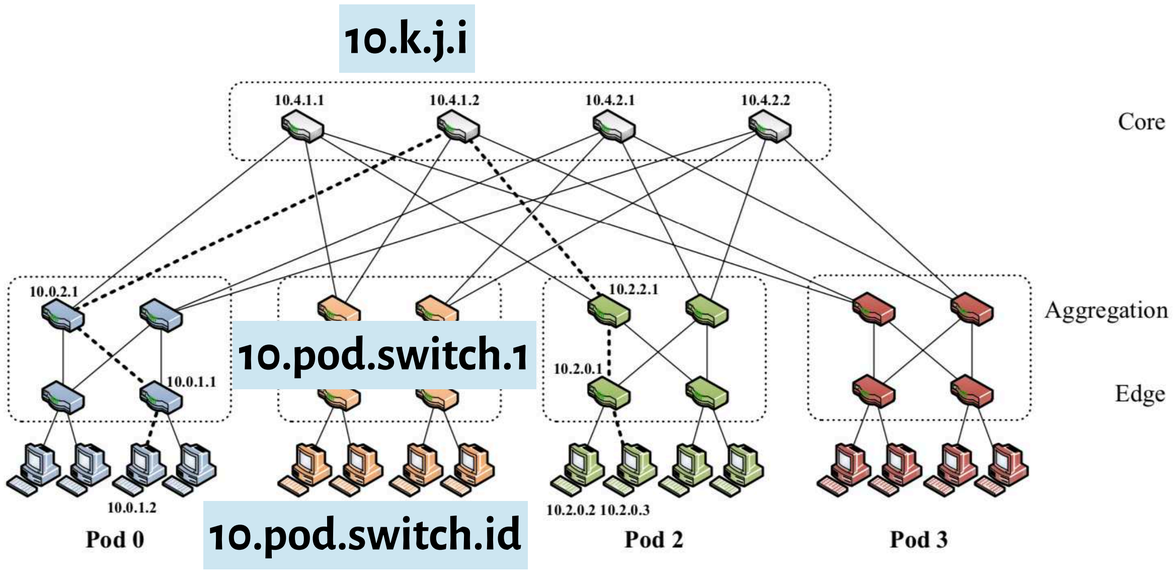
- Routing algorithms (such as OSPF2) will naively choose a single shortest path to use between subnets
- Leads to bottleneck quickly
- $(k/2)^2$ shortest paths available, should use them all equally

Complex wiring due to lack of high-speed ports



Hints: take advantage of the **regularity of the fat-tree structure** to simplify protocol design and improve performance

Addressing in fat-tree



Use 10.0.0.0/8 private address block

Pod switches: **10.pod.switch.1**

- Pod and switch between [0, *k*-1] based on the position

Core switches: **10.k.j.i**

- i and j denote core positions in $(k/2)^2$ core switches

Hosts: **10.pod.switch.id**

- ID in [2, (k/2)+1]
- $k < 256$, does not scale indefinitely

Why?

Forwarding

Two-level lookup table

Host IP: **10.pod.switch.id**

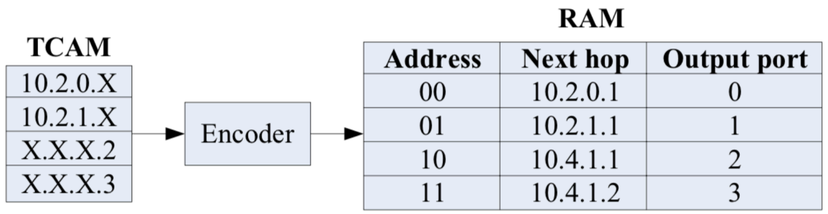
- Prefixes used for forwarding intra-pod traffic
- Suffixes used for forwarding inter-pod traffic

Hosts in the same pod are forwarded based on the IP prefix

Prefix	Output port
10.2.0.0/24	0
10.2.1.0/24	1
0.0.0.0/0	

Suffix	Output port
0.0.0.2/8	2
0.0.0.3/8	3

Hosts in different pods are forwarded based on the host ID



TCAM-based implementation

Routing

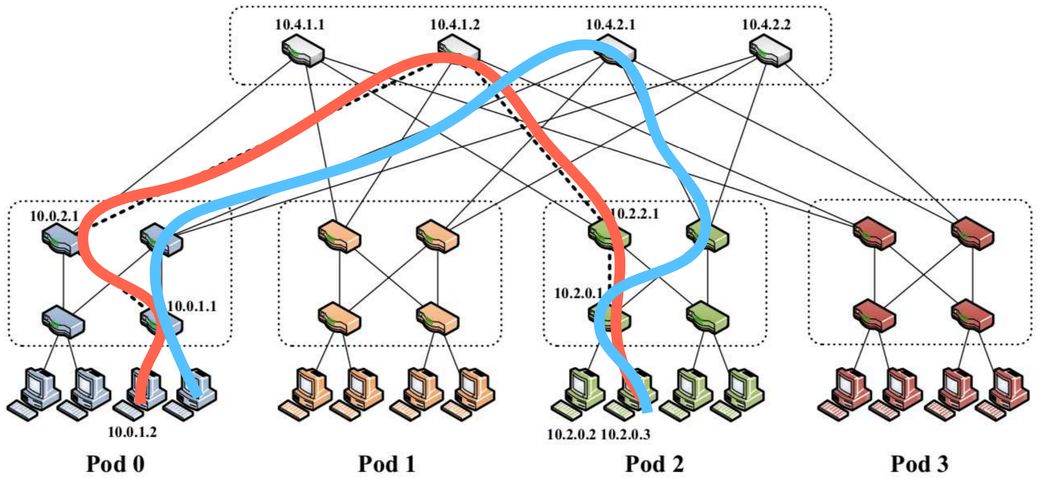
Prefixes in two-level lookup table prevent intra-pod traffic from leaving the pod

Inter-pod traffic is handled by suffix table

- **Suffixes** based on host IDs, ensuring spread of traffic across core switches
- Prevent packet reordering by having static path

Each host-to-host communication has **a single static path**

- Not perfect, but better than having a single static path between two subnets (as in OSPF)



Routing

Core switches contain **[10.pod.0.0/16, port]** entries

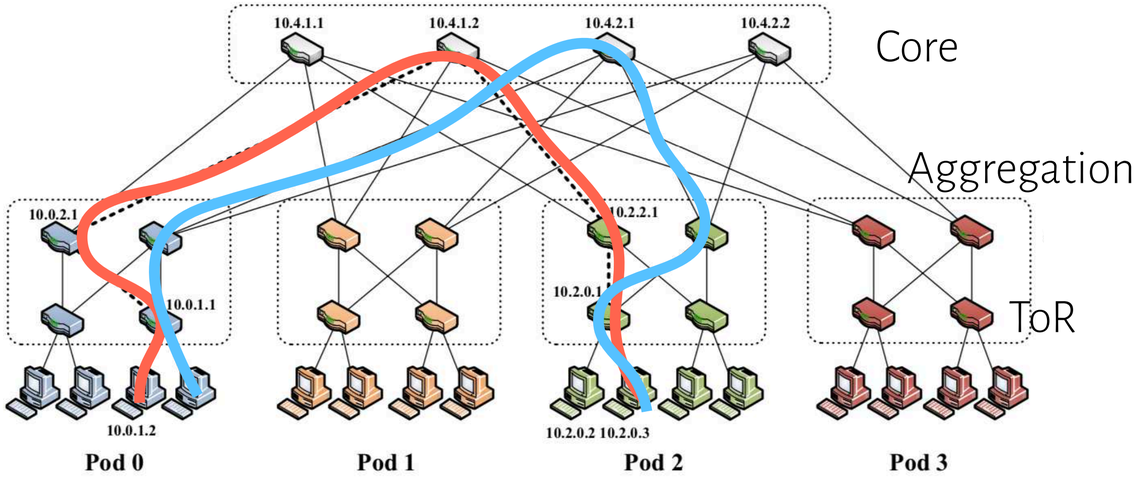
- Statically forward inter-pod traffic on specific port

Aggregation switches contain **[10.pod.switch.0/24, port]** entries

- Switch value is the edge switch number

Assume a **central entity** with full knowledge of the topology generates these routing tables

- Also responsible for detecting switch failures and re-routing traffic

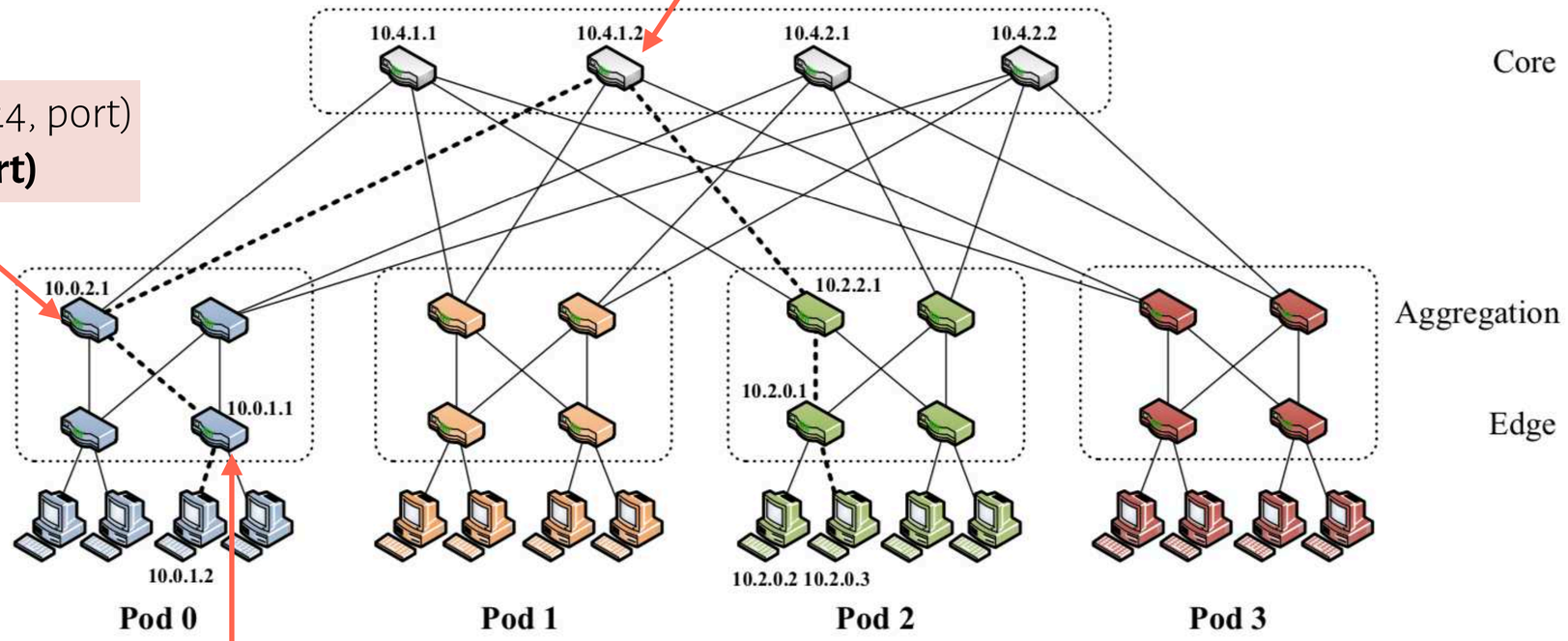


Routing example

10.0.1.2 → 10.2.0.3

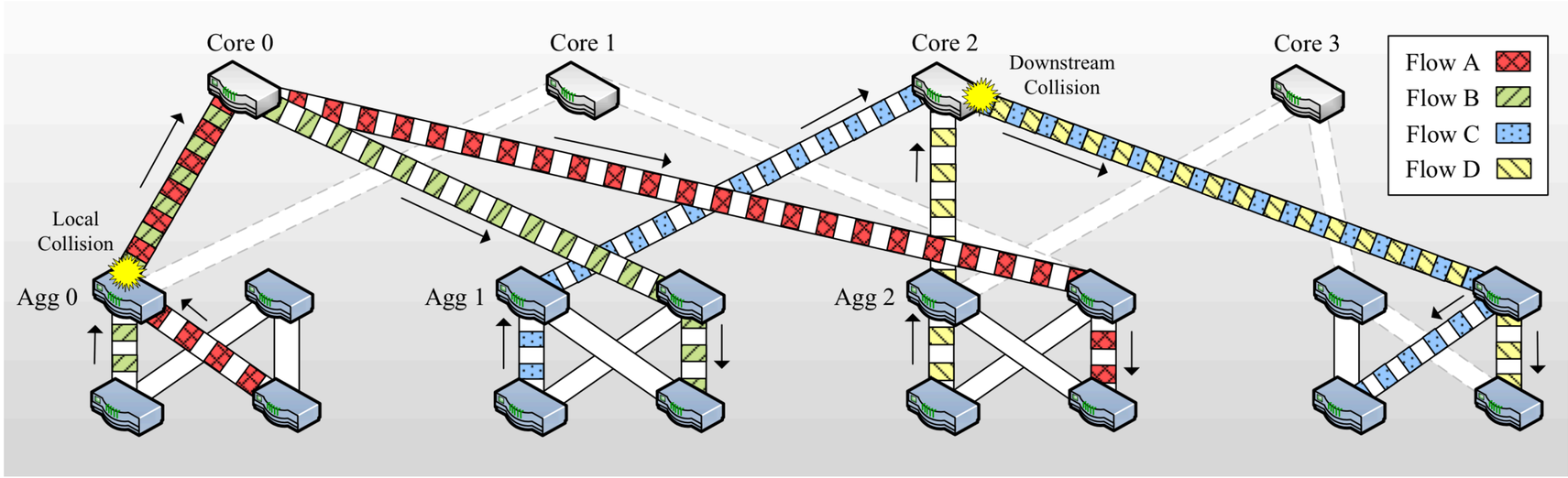
(10.pod.0.0/16, port)

(10.pod.switch.0/24, port)
(0.0.0.id, port)



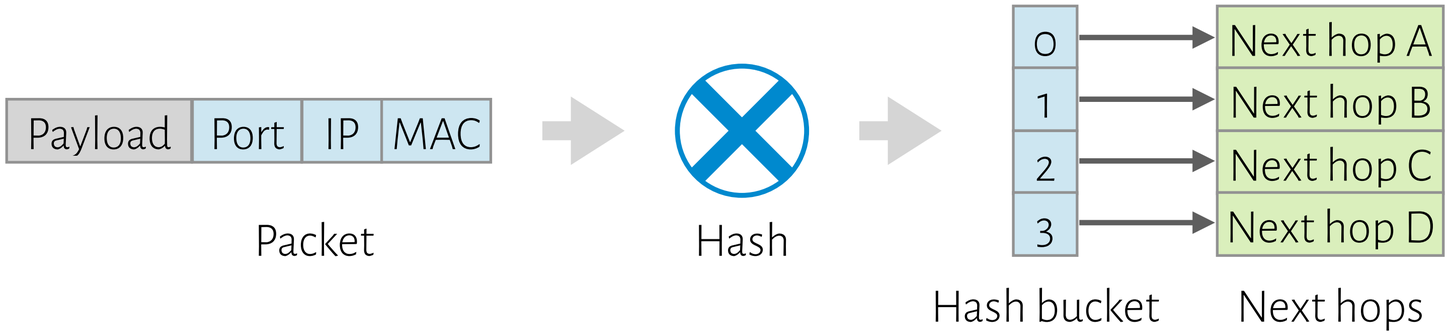
(10.pod.switch.0/24, port)
(0.0.0.id, port)

Flow collision



Hard-coded traffic diffusion can lead to bad collisions
→ performance bottleneck

Solutions to flow collisions

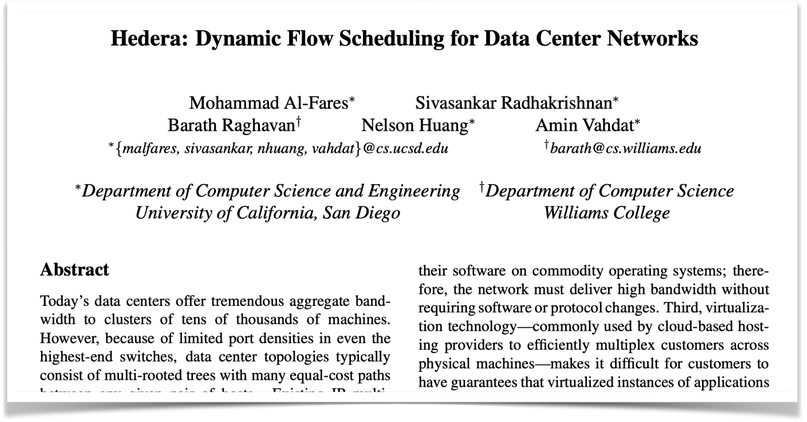


Equal-cost multi-path (ECMP)

- Static path between end-hosts → **static path for each flow**

Flow scheduling

- Have a centralized scheduler to assign flows to paths (leveraging SDN)



Hedera: Dynamic Flow Scheduling for Data Center Networks

Mohammad Al-Fares* Sivasankar Radhakrishnan*
Barath Raghavan† Nelson Huang* Amin Vahdat*
*{malfares, sivasankar, nhuang, vahdat}@cs.ucsd.edu †barath@cs.williams.edu

*Department of Computer Science and Engineering
University of California, San Diego
†Department of Computer Science
Williams College

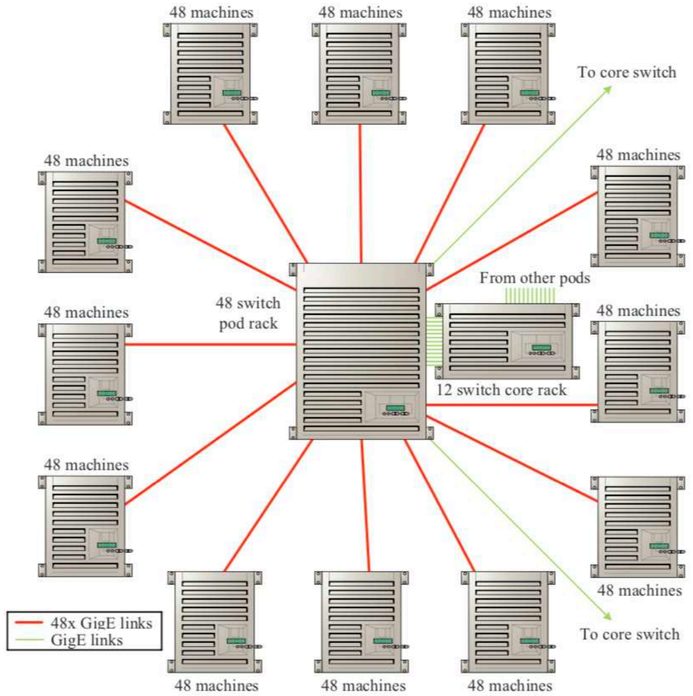
Abstract

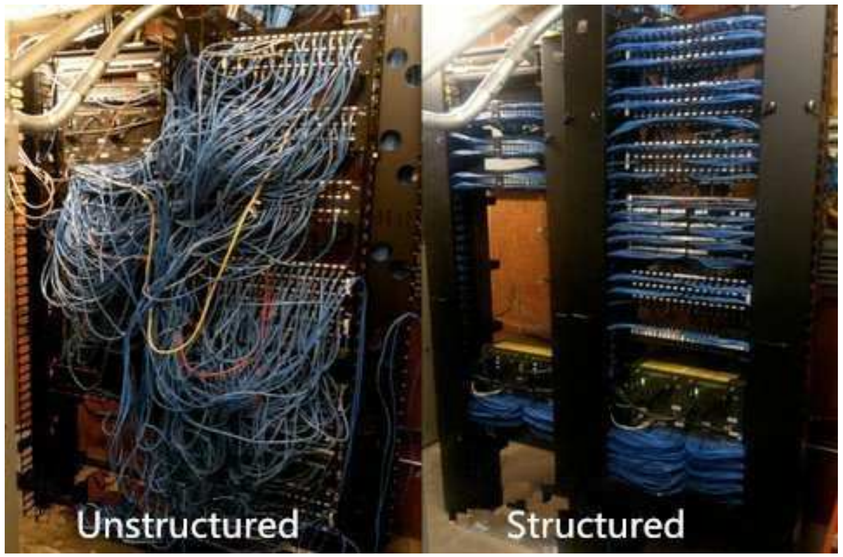
Today's data centers offer tremendous aggregate bandwidth to clusters of tens of thousands of machines. However, because of limited port densities in even the highest-end switches, data center topologies typically consist of multi-rooted trees with many equal-cost paths

their software on commodity operating systems; therefore, the network must deliver high bandwidth without requiring software or protocol changes. Third, virtualization technology—commonly used by cloud-based hosting providers to efficiently multiplex customers across physical machines—makes it difficult for customers to have guarantees that virtualized instances of applications

USENIX NSDI 2010

Fat-tree cabling solution

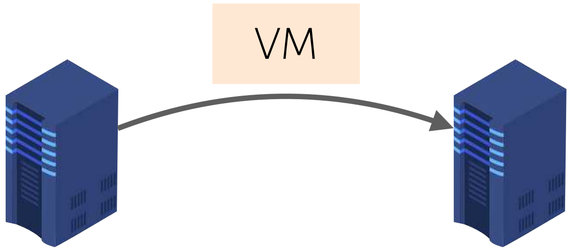




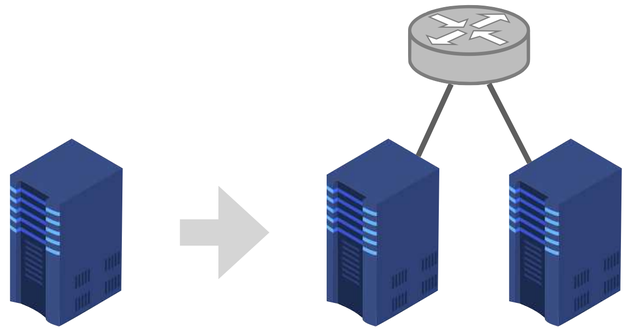
Organize switches into pod racks leveraging the regular structure of fat-tree

Questions?

Unaddressed issues in fat-tree



No support for seamless VM migration: IP addresses are location-dependent and migration would break the TCP connection



Plug-and-play not possible: IP addresses have to be pre-assigned to both switches and hosts

It seems that the location-dependent IP address is the culprit. Any ideas from what you have learned?

L2 vs L3 data center network fabric

Technique	Plug-and-play	Scalability	Small switch state	Seamless VM migration
Layer 2: flat MAC addresses	+	−	−	+
Layer 3: IP addresses	−	+−	+	−

L2 vs L3 data center network fabric

Technique	Plug-and-play	Scalability	Small switch state	Seamless VM migration
Layer 2: flat MAC addresses	+	− Broadcast	−	+
Layer 3: IP addresses	− Location-dependent addresses mandate manual configuration	+−	+	− IP endpoint changes

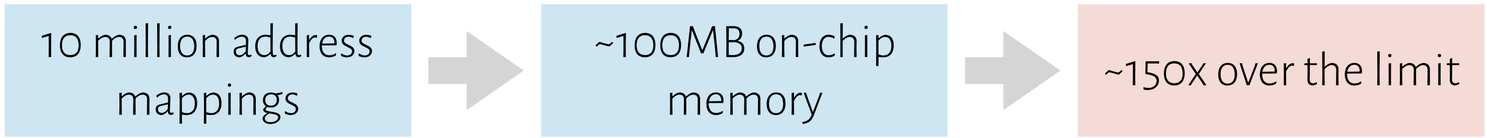
Switch state: L2 vs L3

Commodity switches have ~640KB of low latency, power hungry, expensive on chip memory (e.g., TCAM)

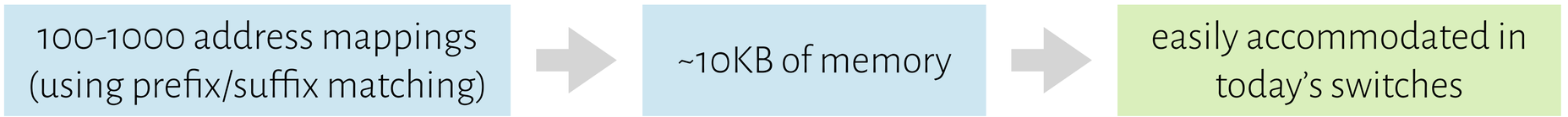
- Can store 32-64K forwarding entries

In a data center with **500K servers**, there could be **10 million virtual endpoints** that need to be addressed

- Flat address (MAC address)



- Hierarchical address (IP address)



PortLand

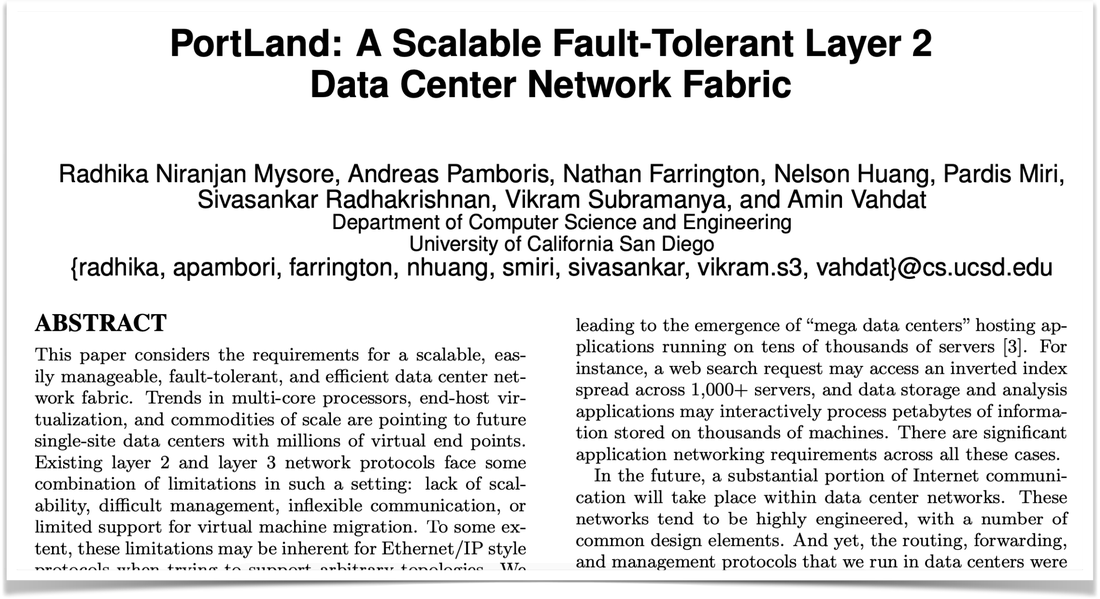
Main idea: separate node location from node identifier

- Host IP: node identifier
- Pseudo MAC (PMAC): node location

Fabric manager

- Maintains IP → PMAC mapping for ARP
- Facilitates fault tolerance

PMAC sufficient for positional forwarding



PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric

Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat
Department of Computer Science and Engineering
University of California San Diego
{radhika, apambori, farrington, nhuang, smiri, sivasankar, vikram.s3, vahdat}@cs.ucsd.edu

ABSTRACT

This paper considers the requirements for a scalable, easily manageable, fault-tolerant, and efficient data center network fabric. Trends in multi-core processors, end-host virtualization, and commodities of scale are pointing to future single-site data centers with millions of virtual end points. Existing layer 2 and layer 3 network protocols face some combination of limitations in such a setting: lack of scalability, difficult management, inflexible communication, or limited support for virtual machine migration. To some extent, these limitations may be inherent for Ethernet/IP style protocols when trying to support arbitrary topologies. We

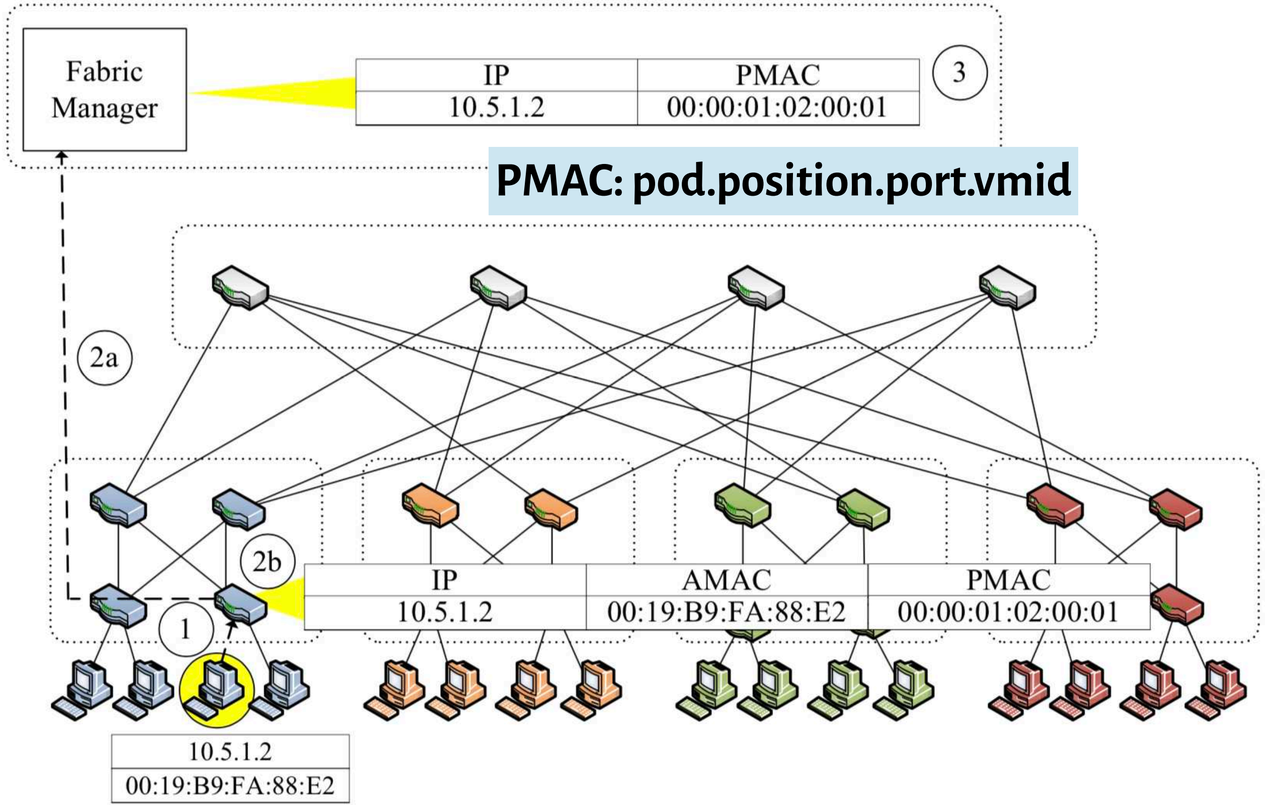
leading to the emergence of "mega data centers" hosting applications running on tens of thousands of servers [3]. For instance, a web search request may access an inverted index spread across 1,000+ servers, and data storage and analysis applications may interactively process petabytes of information stored on thousands of machines. There are significant application networking requirements across all these cases.

In the future, a substantial portion of Internet communication will take place within data center networks. These networks tend to be highly engineered, with a number of common design elements. And yet, the routing, forwarding, and management protocols that we run in data centers were

ACM SIGCOMM 2009

PortLand design

Plug-and-play + small switch state



Switches self-discover location by exchanging **Location Discovery Messages (LDMs):**

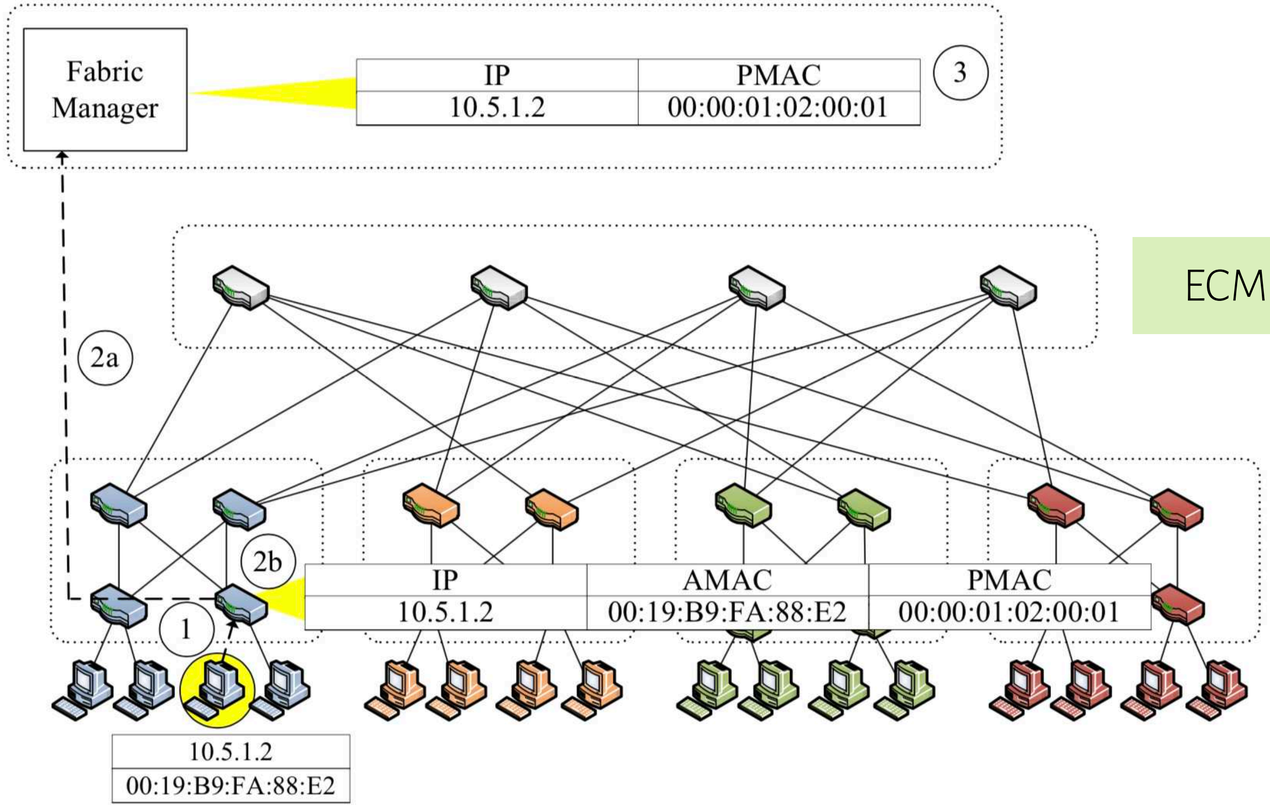
- Tree-level/role: based on neighbor identify
- Pod number: fetch from the Fabric manager
- Position number: aggregation switches help ToR switches choose unique position number

Fabric manager

ARP mappings

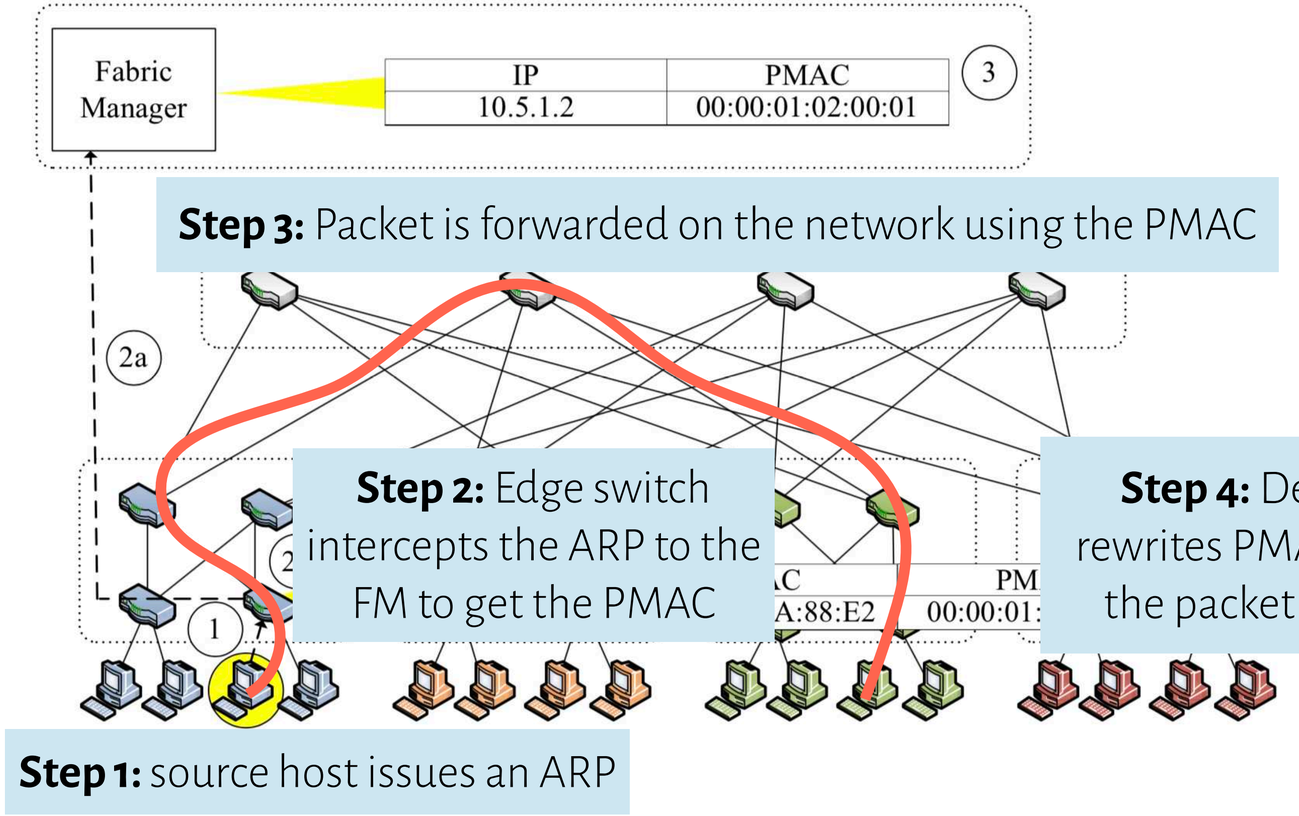
Network map

Soft state: no need for manual configurations



ECMP at the core

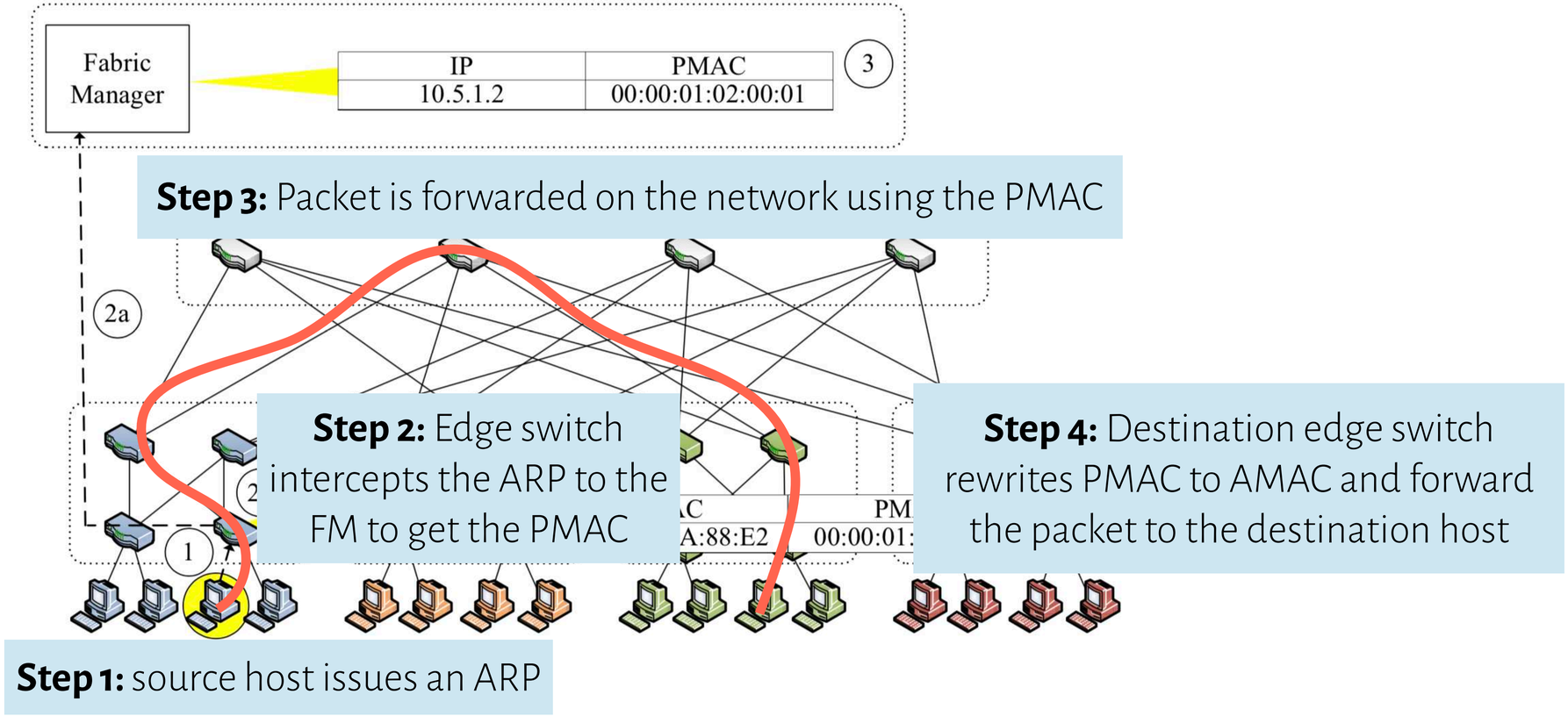
PortLand workflow



PortLand workflow

Hardware support:

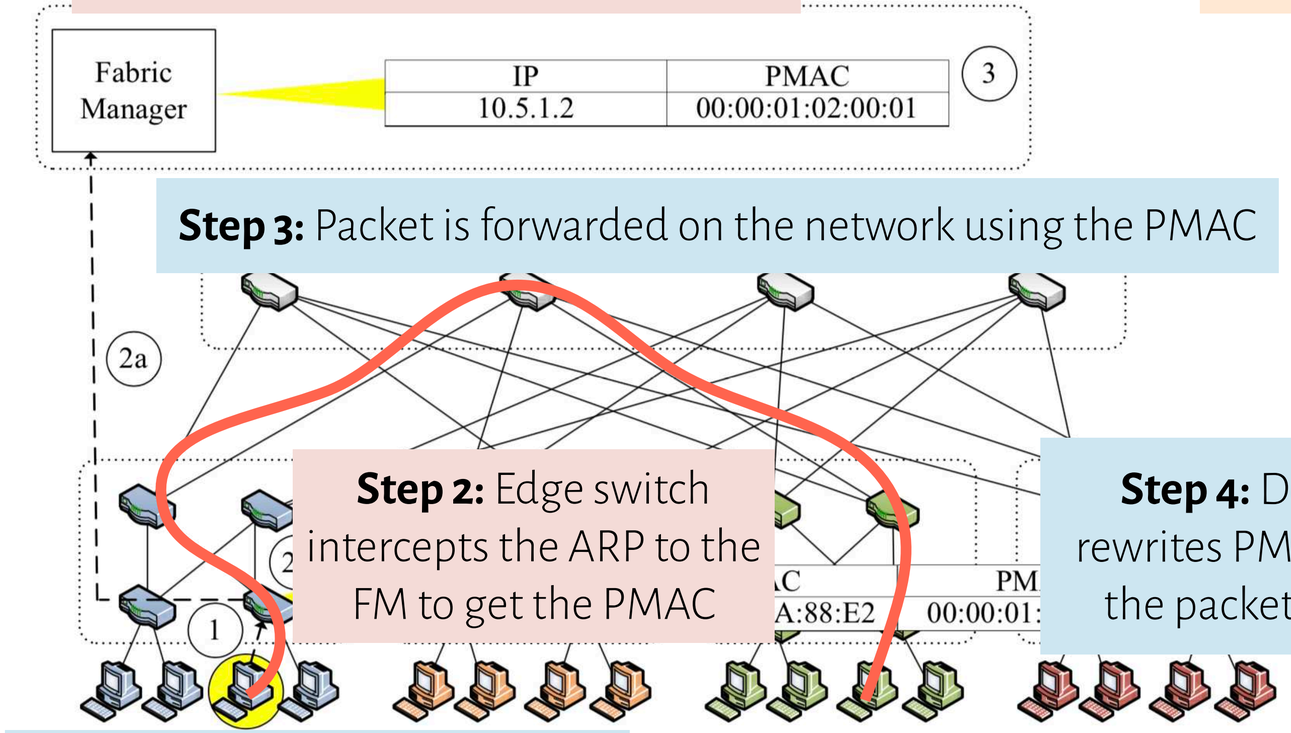
- No modification needed for hosts
- PMAC <> AMAC translation on edge switches
- Other switches forward based on prefix-matching on PMAC



PortLand workflow

In case of no matching entry, the FM broadcasts the ARP request

How does the FM populate the IP-PMAC table?



Step 3: Packet is forwarded on the network using the PMAC

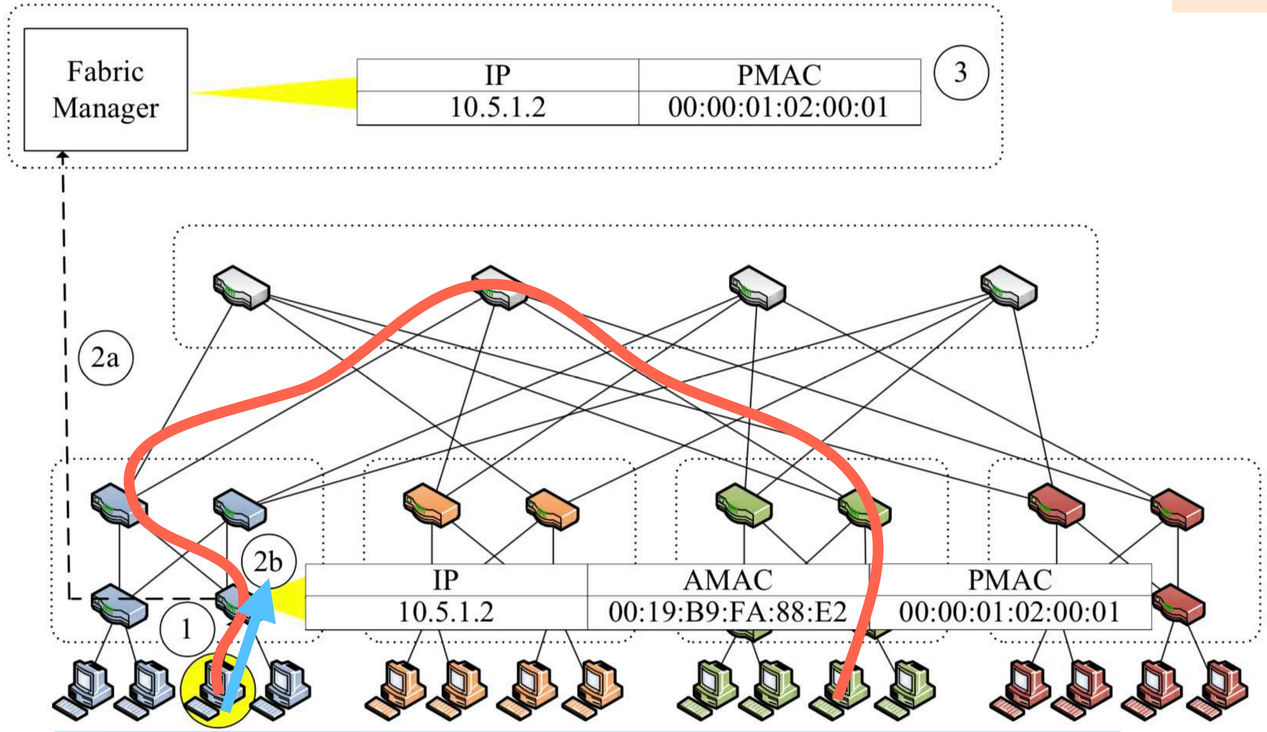
Step 2: Edge switch intercepts the ARP to the FM to get the PMAC

Step 4: Destination edge switch rewrites PMAC to AMAC and forward the packet to the destination host

Step 1: source host issues an ARP

PortLand workflow

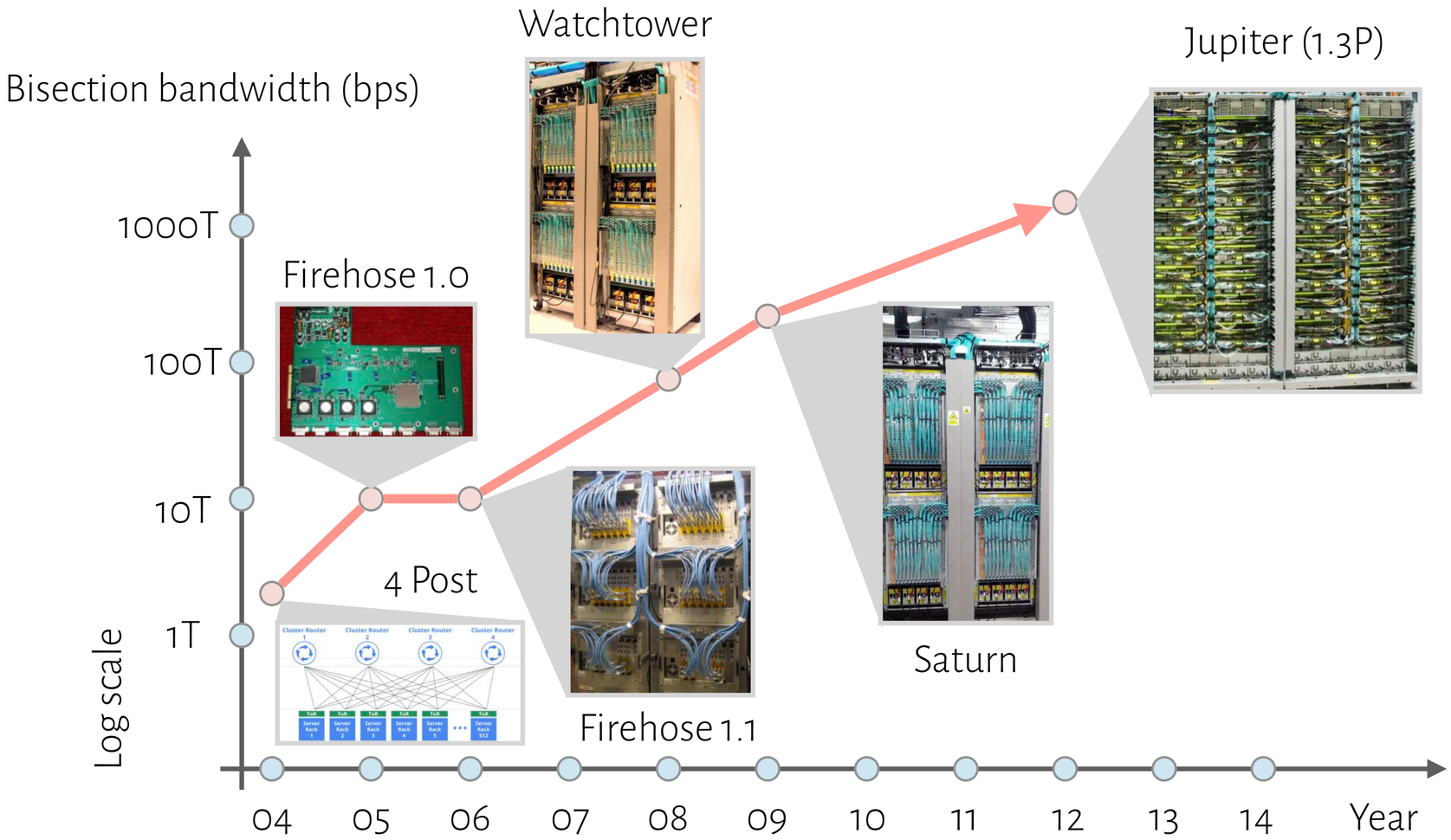
How does the FM populate the IP-PMAC table?



Recall learning switch: an IP-PMAC entry is forwarded to the FM every time the edge switch sees a new IP

Questions?

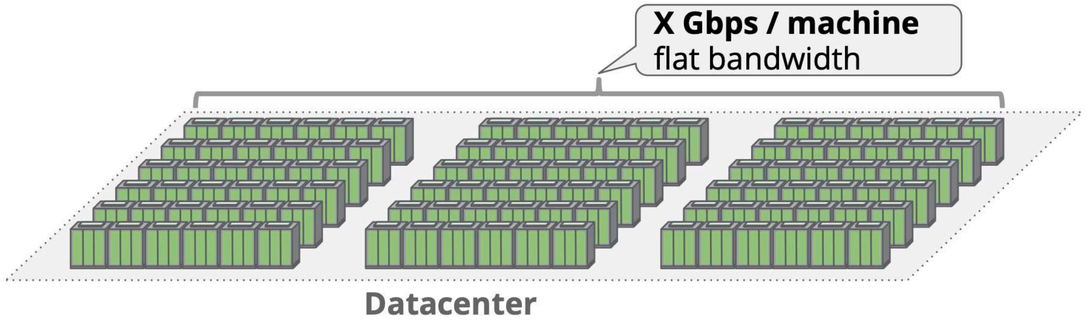
Evolution of Google's cloud networking

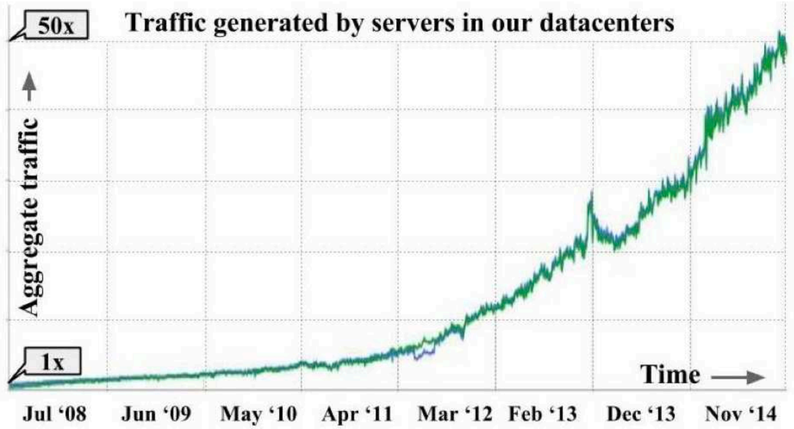


Jupiter

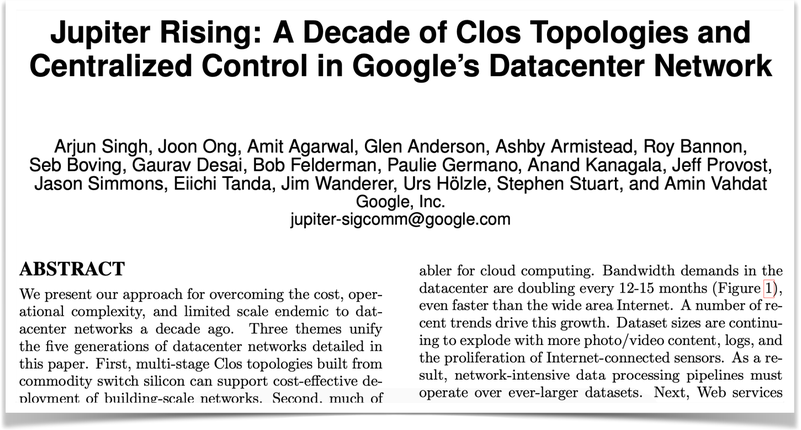
Challenge: flat bandwidth profile across all servers

- Simple job scheduling (remove locality)
- Save significant resources via better bin-packing
- Allow application scaling





Traffic in Google data center



Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network

Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat
Google, Inc.
jupiter-sigcomm@google.com

ABSTRACT

We present our approach for overcoming the cost, operational complexity, and limited scale endemic to datacenter networks a decade ago. Three themes unify the five generations of datacenter networks detailed in this paper. First, multi-stage Clos topologies built from commodity switch silicon can support cost-effective deployment of building-scale networks. Second, much of

abler for cloud computing. Bandwidth demands in the datacenter are doubling every 12-15 months (Figure 1), even faster than the wide area Internet. A number of recent trends drive this growth. Dataset sizes are continuing to explode with more photo/video content, logs, and the proliferation of Internet-connected sensors. As a result, network-intensive data processing pipelines must operate over ever-larger datasets. Next, Web services

ACM SIGCOMM 2015

Ideas behind Jupiter

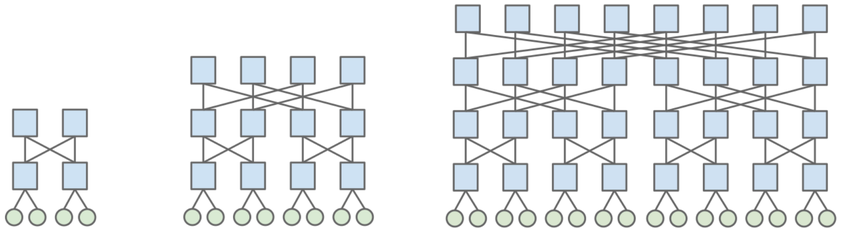
Merchant silicon: general purpose, commodity priced, off the shelf switching components

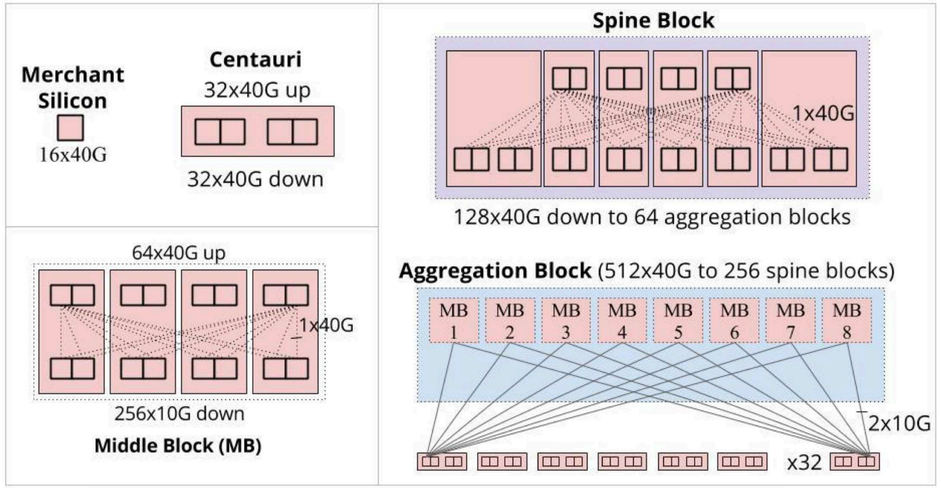
Clos topologies: Accommodate low radix switch chips to scale nearly arbitrarily by adding stages

Centralized control / management: e.g., SDN-based

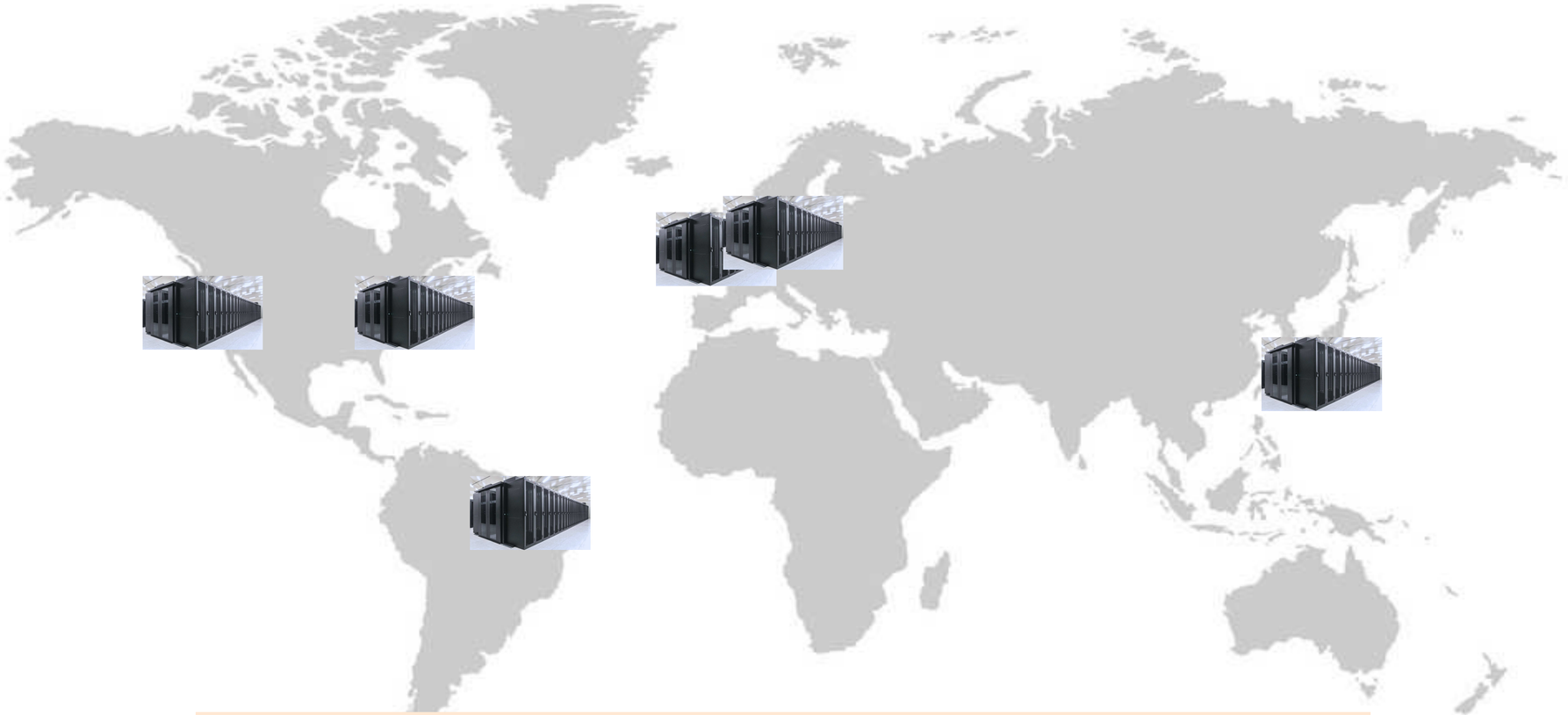
→ Scales out building wide 1.3Pbps

- Enables 40Gbps to hosts
- External control servers
- OpenFlow





Data center backbone



How to interconnect the geographically distributed data centers?

Why a data center backbone?

Data centers deployed across the world

- Serve content with **geographic locality**
- **Replicate** content for fault tolerance

Need a network to connect these data centers to one another

- Not on the public Internet, why?
- **Cost-effective** network for high volume traffic
- Application-specific variable in **SLO**
- **Bursty/bulk** traffic (not smooth/diurnal)

Google data center backbone

Two separate backbones

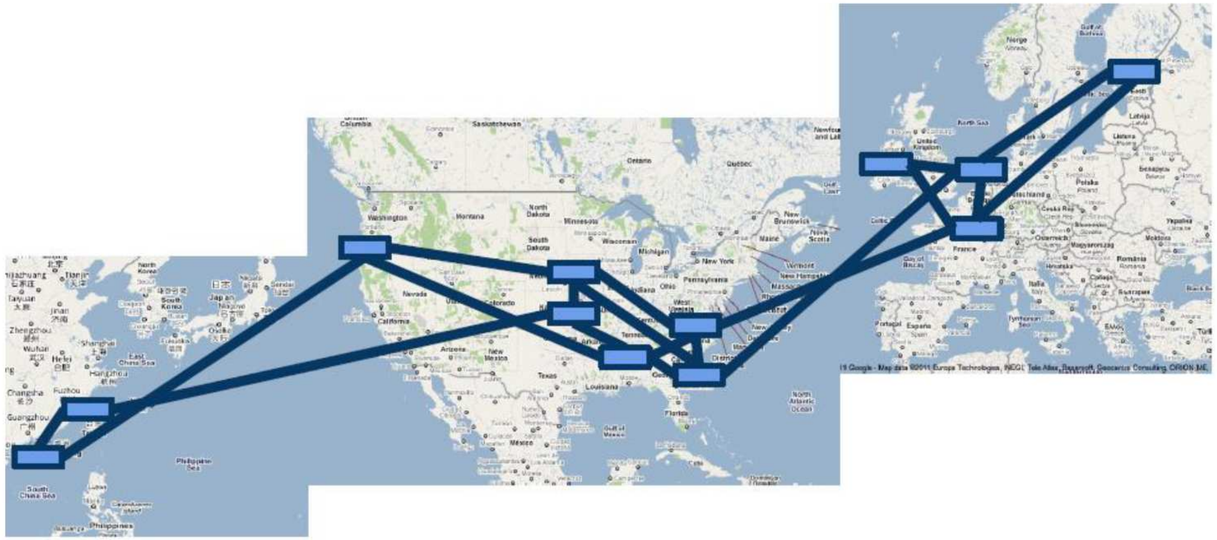
- **B2:** Carries Internet facing traffic → growing faster than the Internet
- **B4:** Inter-data center traffic → more traffic than B2, growing faster than B2

Unlike compute and storage, networking cost/bit does not naturally decrease with size

- Quadratic **complexity** in pairwise interactions and broadcast overhead of all-to-all communication requires more expensive equipment
- **Manual** management and configuration of individual elements
- Complexity of automated configuration to deal with **non-standard vendor configuration** APIs

SDN to the rescue

B4: Google's software defined WAN



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

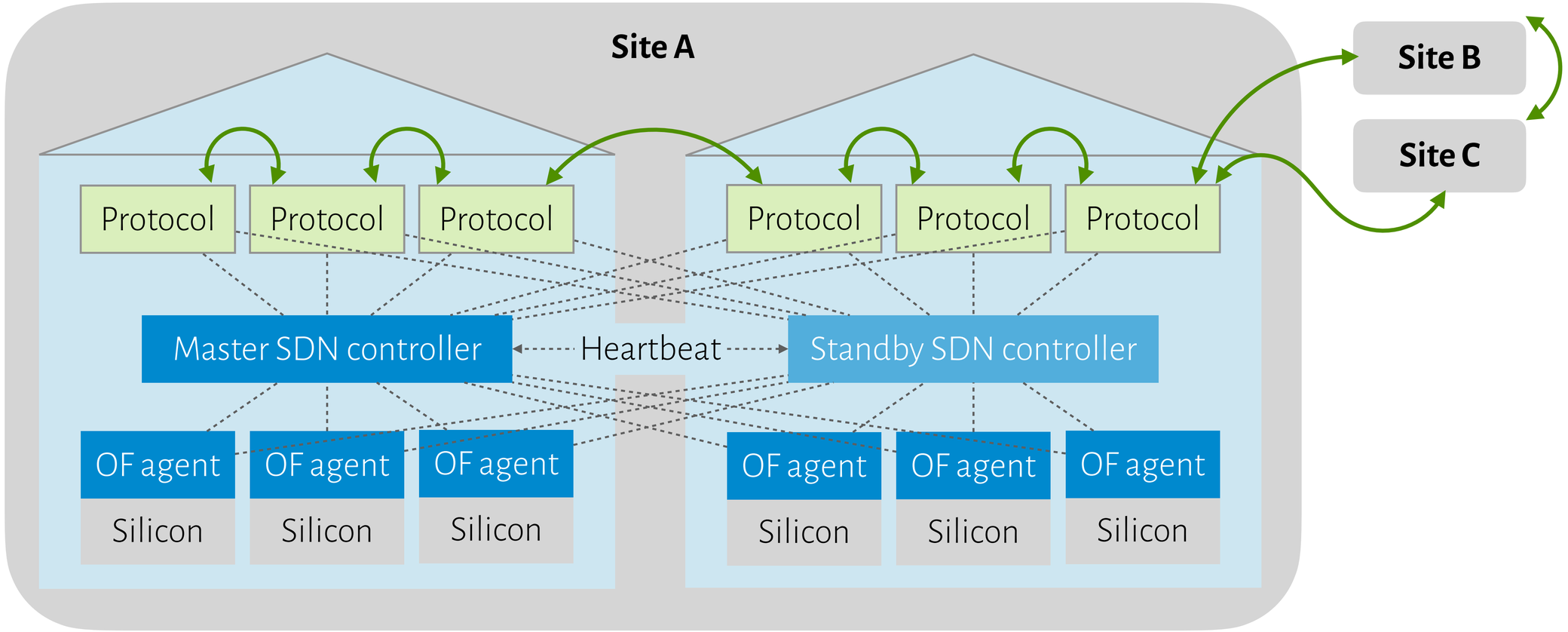
Such overprovisioning delivers admirable reliability at the very real costs of 2-3x bandwidth over-provisioning and high-end routing gear.

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 pri-

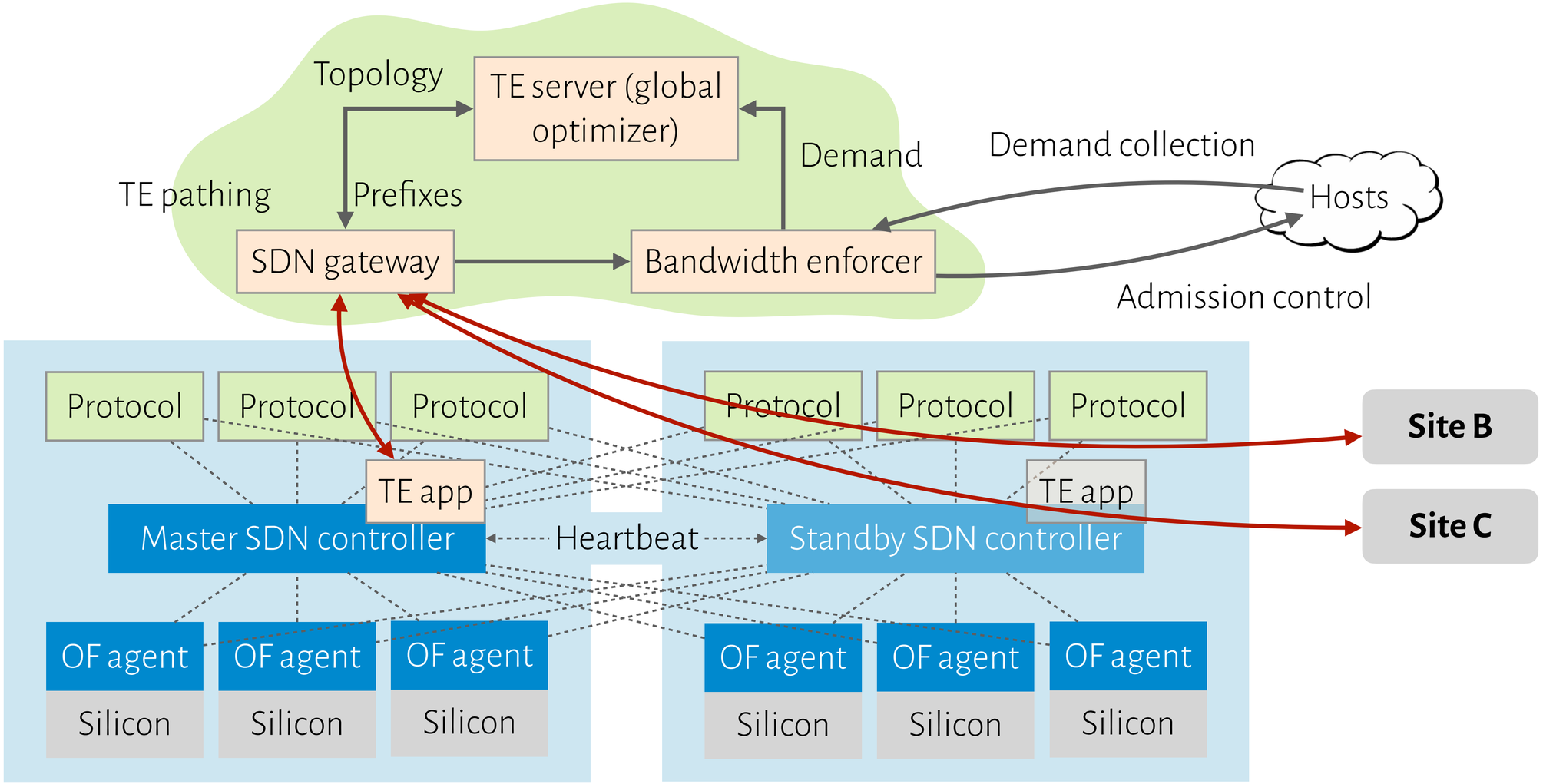
ACM SIGCOMM 2013

B4 site: SDN architecture

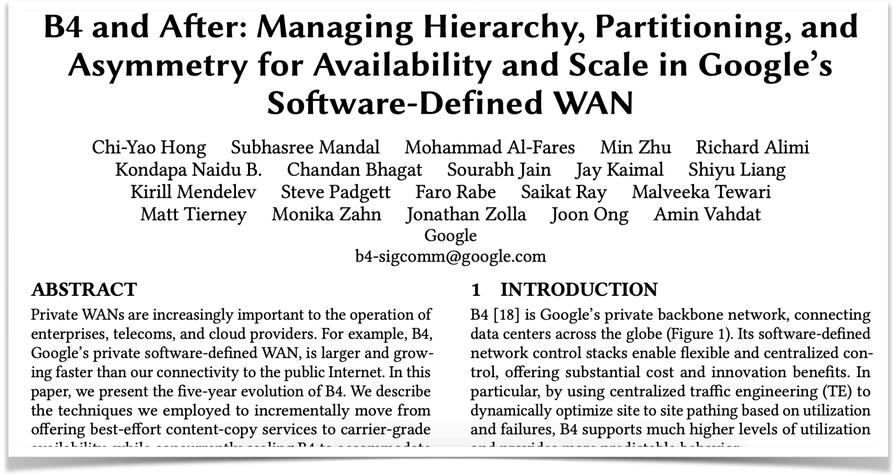
Traditional WAN integrated with SDN: still speaks legacy protocols like BGP



B4 site: traffic engineering control plane



More about Google's cloud networking suite



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

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
Google
b4-sigcomm@google.com

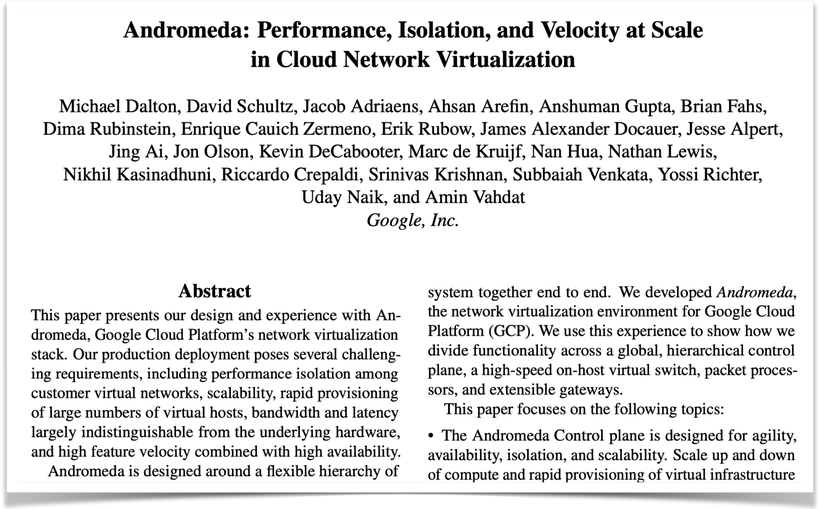
ABSTRACT

Private WANs are increasingly important to the operation of enterprises, telecoms, and cloud providers. For example, B4, Google's private software-defined WAN, is larger and growing faster than our connectivity to the public Internet. In this paper, we present the five-year evolution of B4. We describe the techniques we employed to incrementally move from offering best-effort content-copy services to carrier-grade

1 INTRODUCTION

B4 [18] is Google's private backbone network, connecting data centers across the globe (Figure 1). Its software-defined network control stacks enable flexible and centralized control, offering substantial cost and innovation benefits. In particular, by using centralized traffic engineering (TE) to dynamically optimize site to site pathing based on utilization and failures, B4 supports much higher levels of utilization

ACM SIGCOMM 2018



Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization

Michael Dalton, David Schultz, Jacob Adriaens, Ahsan Arefin, Anshuman Gupta, Brian Fahs, Dima Rubinstein, Enrique Cauich Zermeno, Erik Rubow, James Alexander Docauer, Jesse Alpert, Jing Ai, Jon Olson, Kevin DeCabooter, Marc de Kruijf, Nan Hua, Nathan Lewis, Nikhil Kasinadhuni, Riccardo Crepaldi, Srinivas Krishnan, Subbaiah Venkata, Yossi Richter, Uday Naik, and Amin Vahdat
Google, Inc.

Abstract

This paper presents our design and experience with Andromeda, Google Cloud Platform's network virtualization stack. Our production deployment poses several challenging requirements, including performance isolation among customer virtual networks, scalability, rapid provisioning of large numbers of virtual hosts, bandwidth and latency largely indistinguishable from the underlying hardware, and high feature velocity combined with high availability.

Andromeda is designed around a flexible hierarchy of system together end to end. We developed *Andromeda*, the network virtualization environment for Google Cloud Platform (GCP). We use this experience to show how we divide functionality across a global, hierarchical control plane, a high-speed on-host virtual switch, packet processors, and extensible gateways.

This paper focuses on the following topics:

- The Andromeda Control plane is designed for agility, availability, isolation, and scalability. Scale up and down of compute and rapid provisioning of virtual infrastructure

USENIX NSDI 2018

Cloud networking research landscape

Architecture

Topology, addressing, rack-scale computing (Fat-tree, BCube, DCell, VL2, PortLand, Jellyfish)

Transport

Congestion control, flow scheduling (DCTCP, pFabric, JumpQueue, HPCC)

RDMA

Congestion control, flow control (TIMELY, GFC, DCQCN, IRN)

Energy efficiency

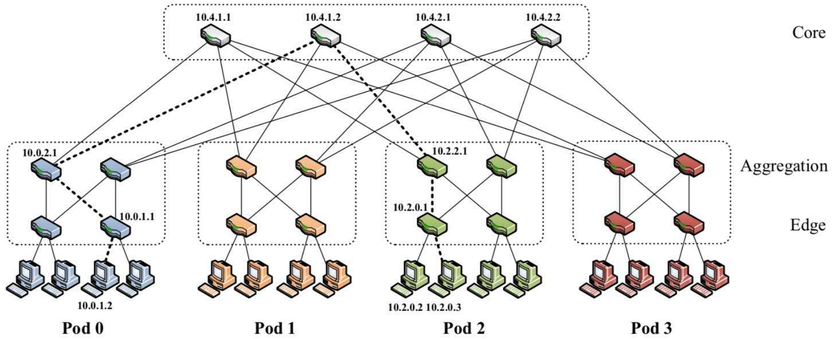
Traffic engineering, flow scheduling (ElasticTree, Flattened-Butterfly)

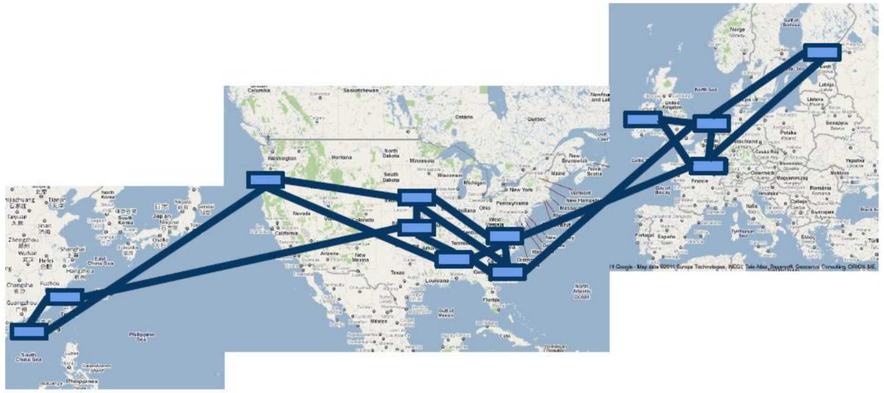
Questions?

Summary

Lecture 10: Cloud networking

- Data centers
- Data center network architecture
- Fat-tree architecture
- L2 vs. L3 addressing
- PortLand architecture
- Google's cloud networking
- Google's WAN B4





Next week: beyond networking

Let us talk about **video**

Video is almost
58% of the total downstream volume of traffic on the internet