X_405082

Advanced Computer Networks

Data Center Networking

Lin Wang (lin.wang@vu.nl)

Period 2, Fall 2020

Course outline

Warm-up

- Fundamentals
- Forwarding and routing
- Network transport

Data centers

- **Data center networking** 👈
- Data center transport

Programmability

- Software defined networking
- Programmable forwarding

Video

- Video streaming
- Video stream analytics

Networking and ML

- Networking for ML
- ML for networking

Mobile computing

- Wireless and mobile

Learning objectives

How to build a **high-performance** data center network?

How to achieve **flexible management** in a data center network?

Cloud computing

Elastic resources

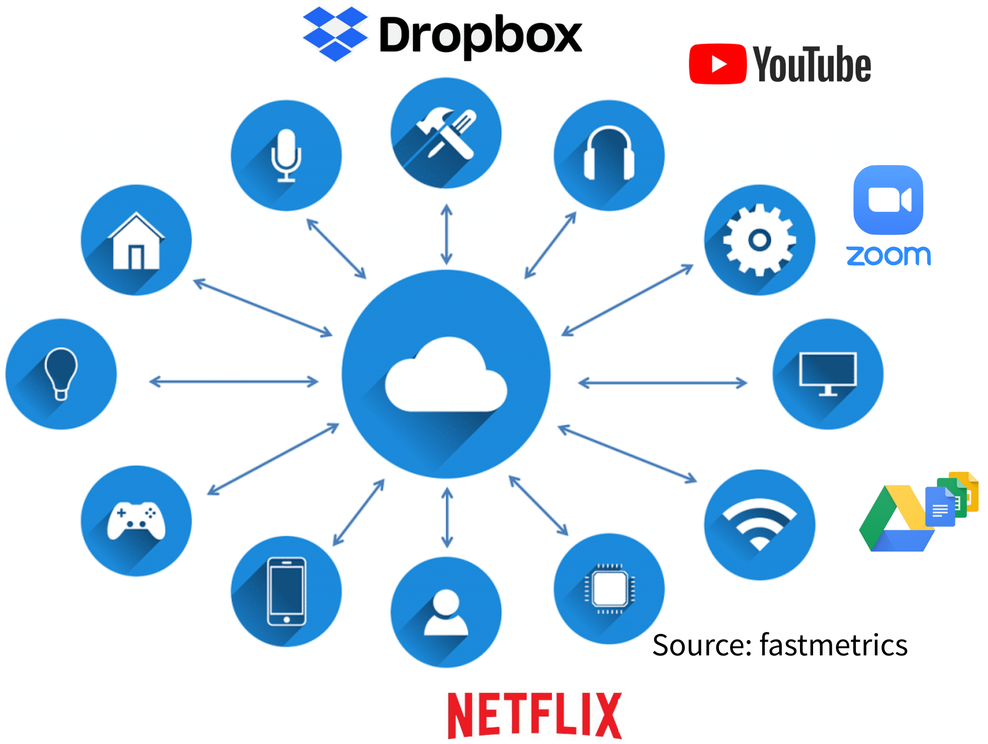
- Expand and contract resources
- Pay-per-use, infrastructure on demand

Multi-tenancy

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

Flexible service management

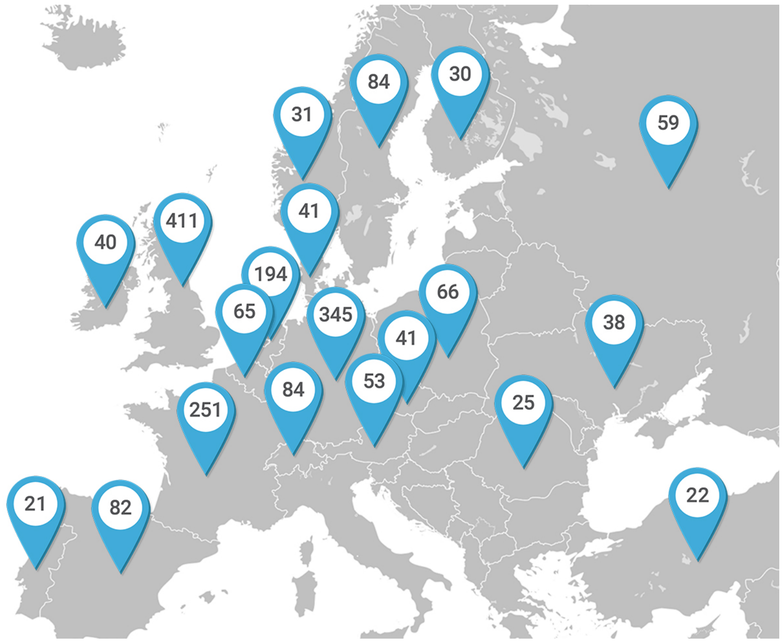
- Resilience: isolate failures of server and storage
- Workload migration: move work to other locations



Source: fastmetrics

What is behind cloud computing?

Large-scale data centers



Data center distribution in Europe

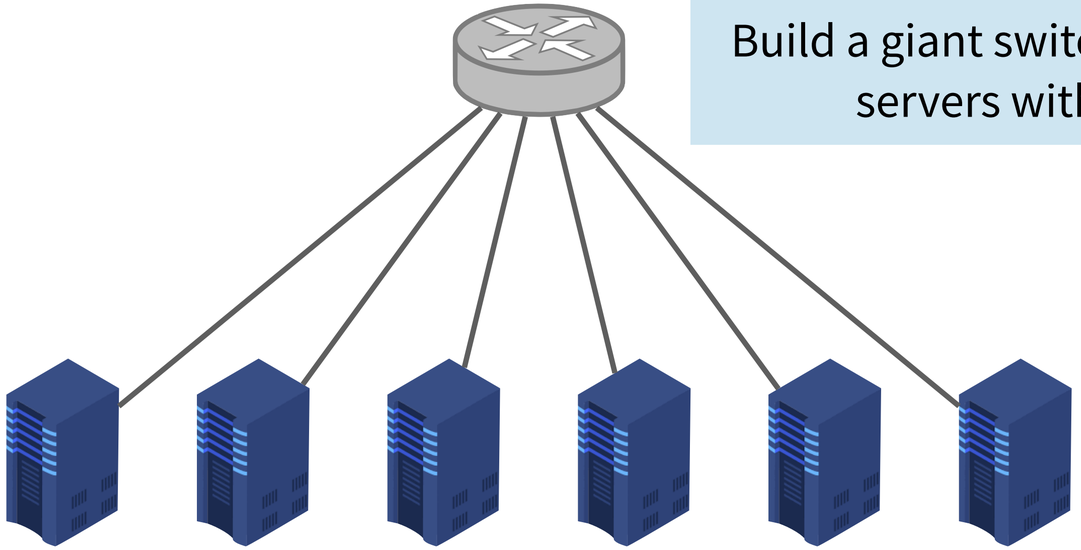
Equinix’s AM4 data center @ Science Park

How does a data center look inside?

Equinix’s AM4 data center @ Science Park

Well organized & interconnected racks of servers!

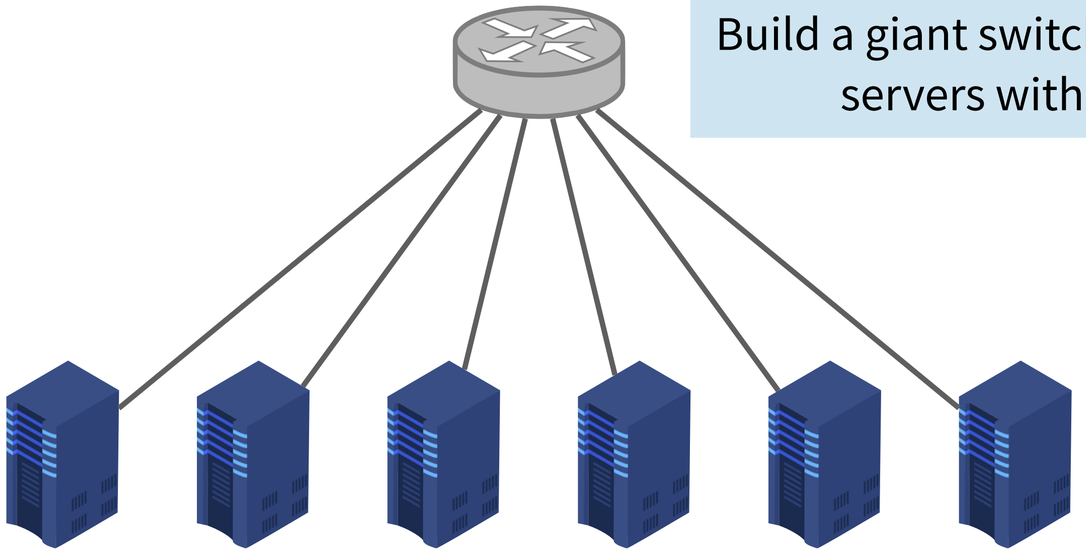
How to connect these servers in a data center?



Build a giant switch and connect all servers with the switch

What problems can you think of with such a design?

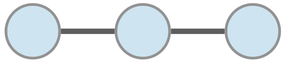
How to connect these servers in a data center?



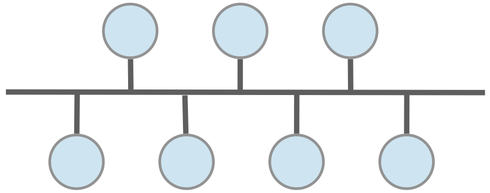
Build a giant switch and connect all servers with the switch

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

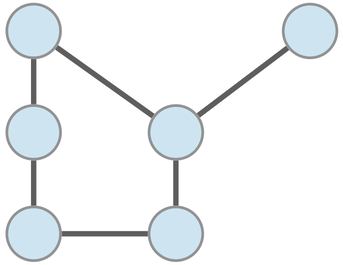
A dedicated network for the data center



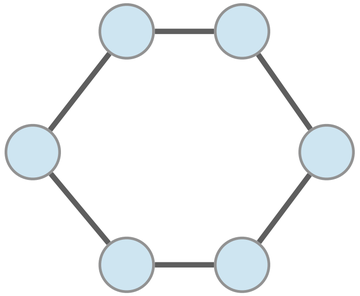
Line



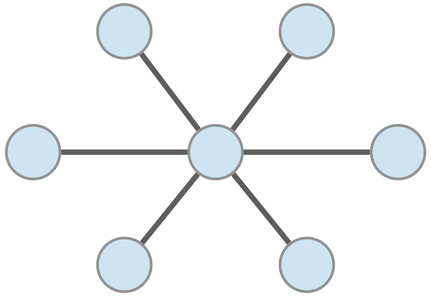
Bus



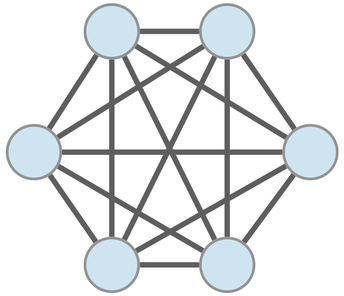
Mesh



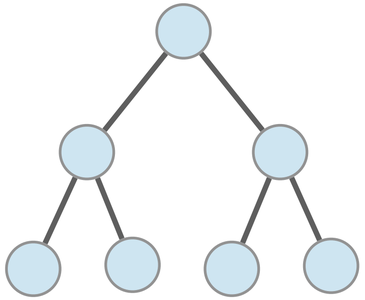
Ring



Star



Fully connected

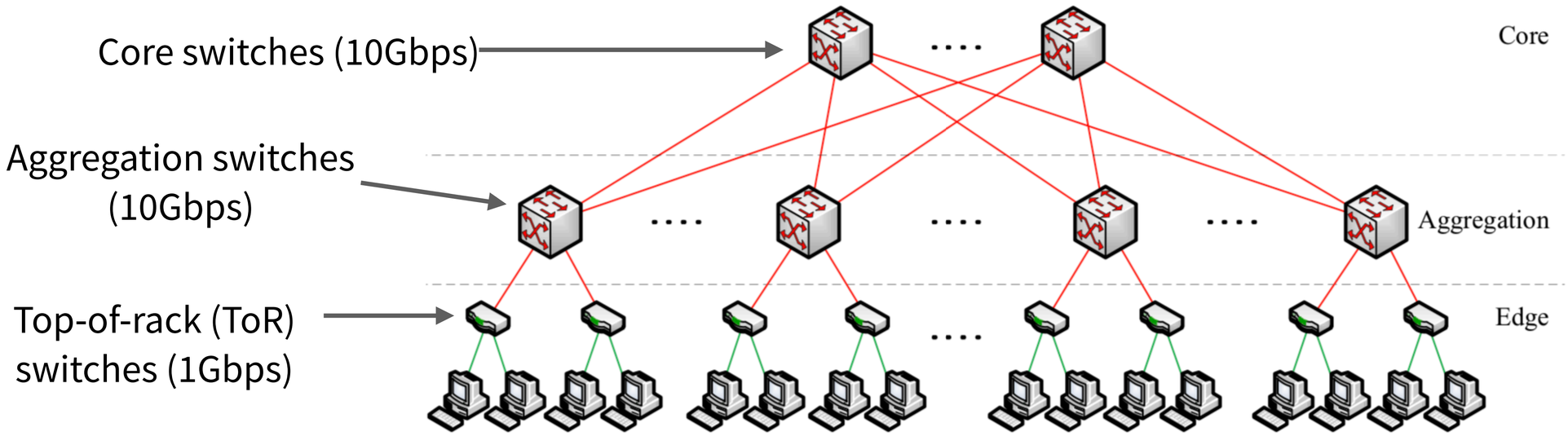


Tree

Tradeoff between connectivity and complexity

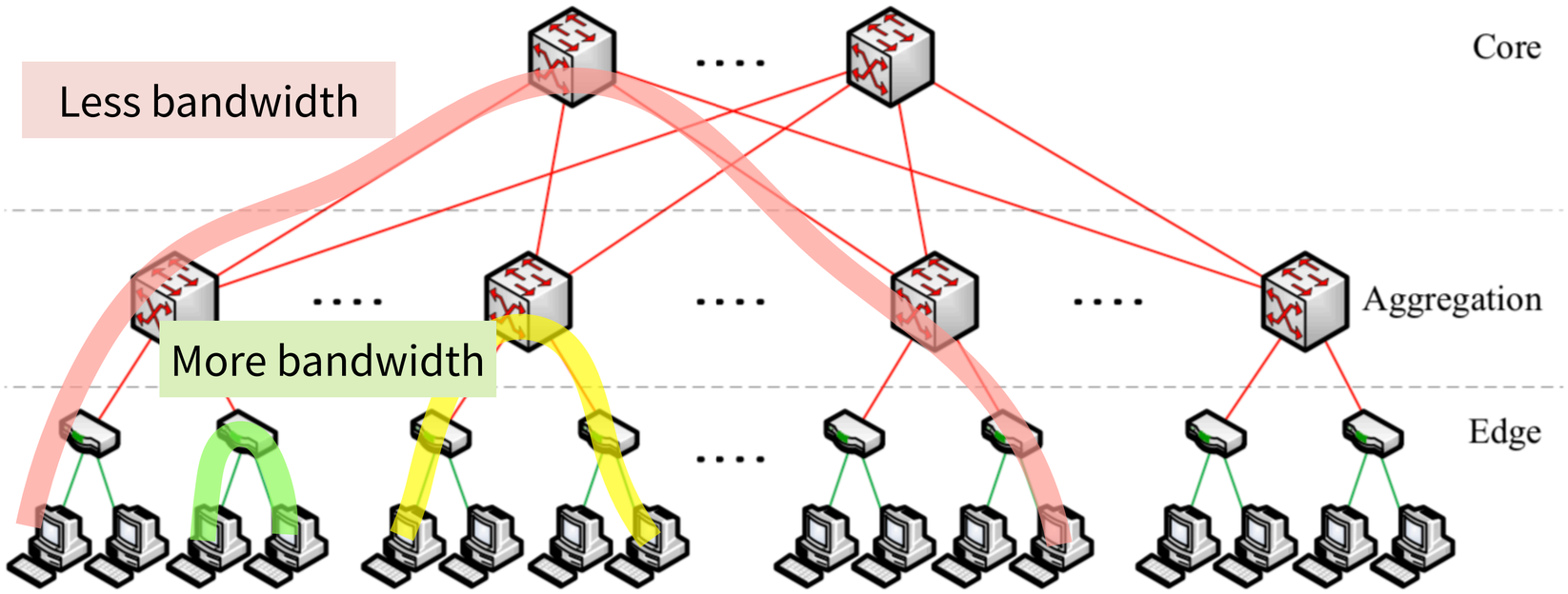
A typical data center network architecture

A 3-tier tree architecture



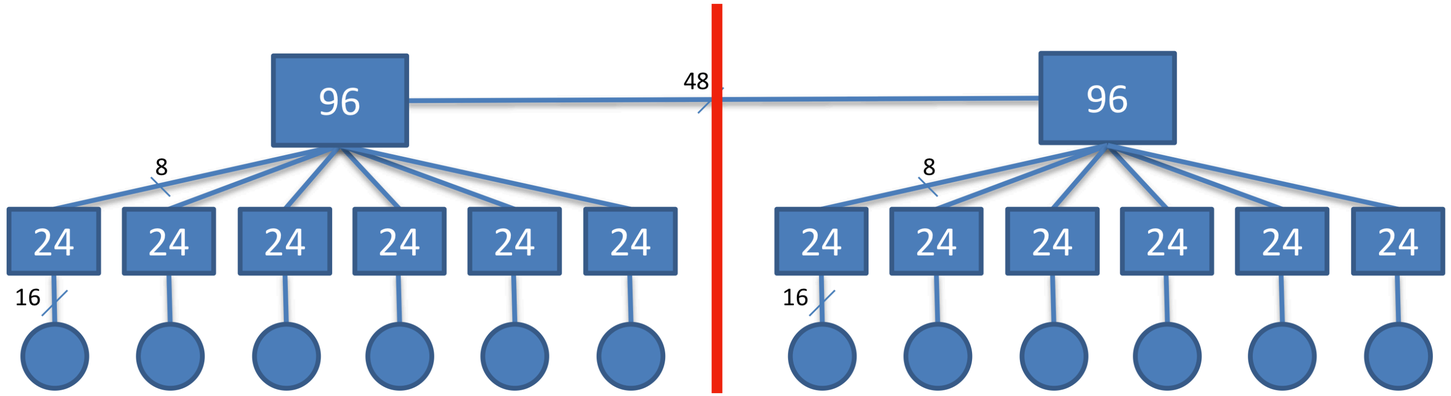
What if ToR switches go for 10Gbps or beyond?

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

Oversubscription

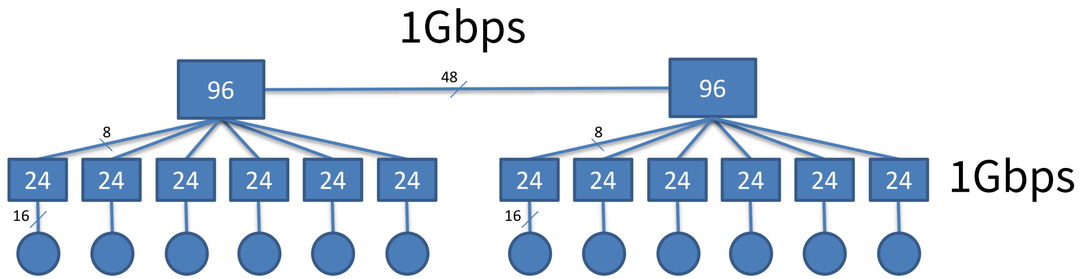
Definition

- 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

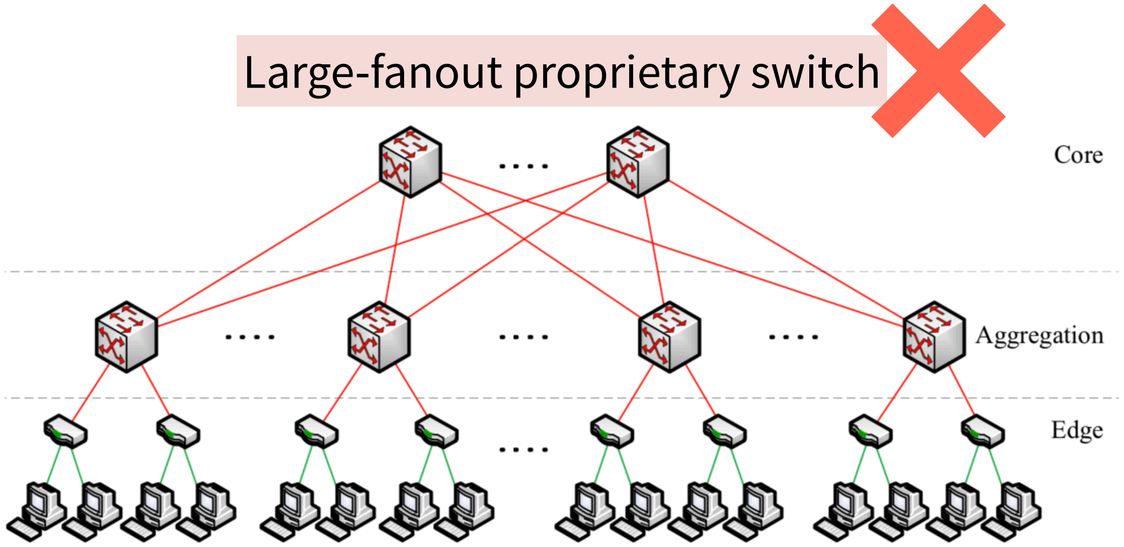
- Inexpensive, commodity servers and storage devices
- But the network is still highly specialized (using large-fanout proprietary switches)

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

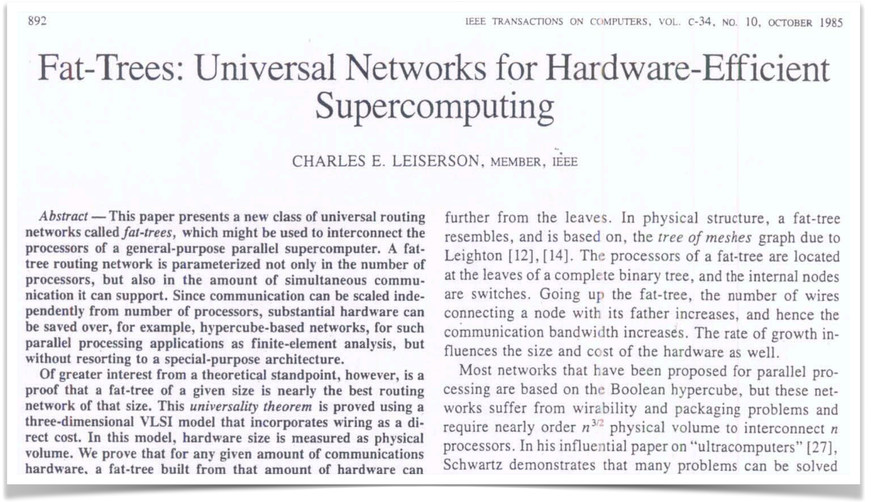
Easy management

- 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

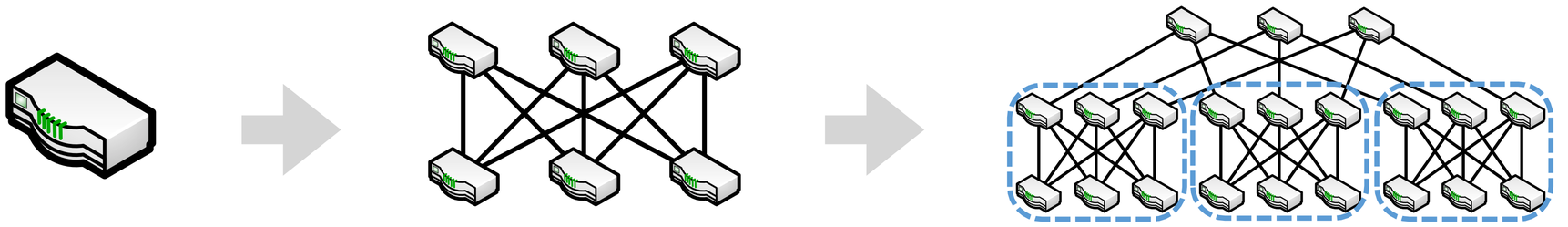
CHARLES E. LEISERSON, MEMBER, IEEE

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.

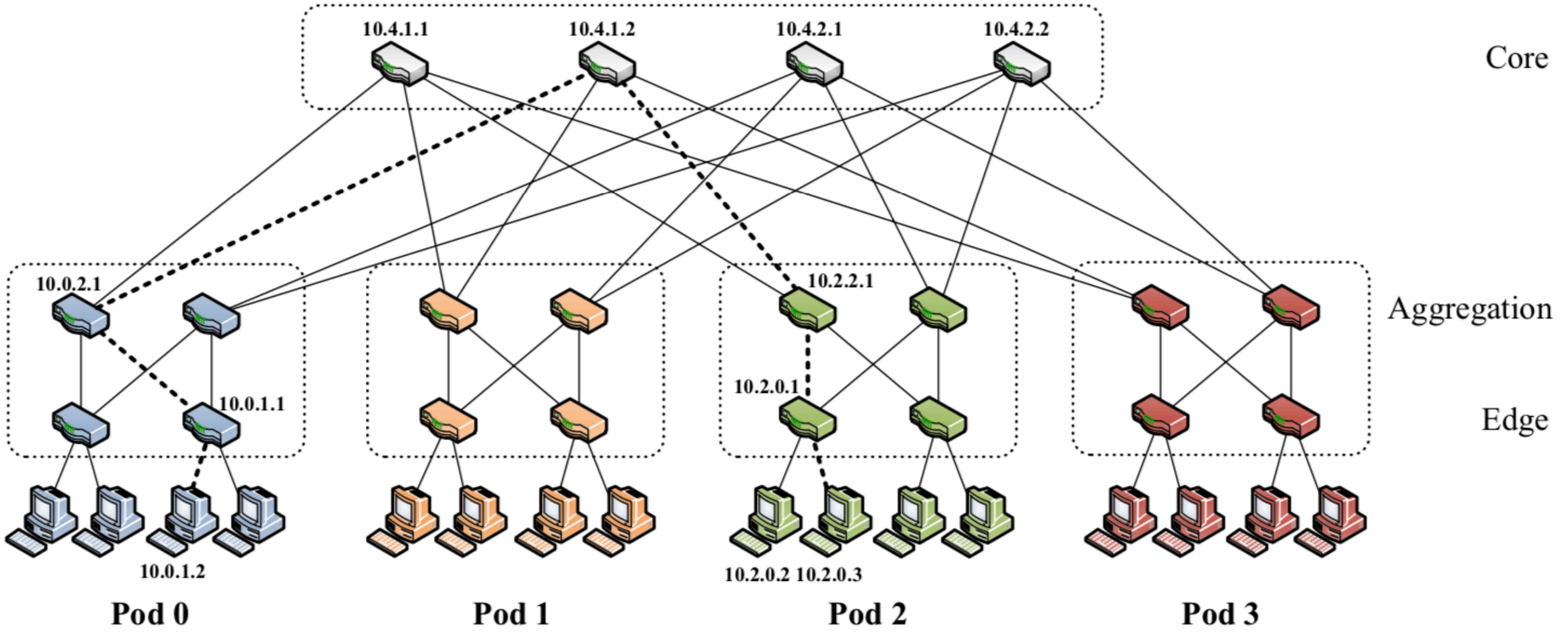
Of greater interest from a theoretical standpoint, however, is a proof that a fat-tree of a given size is nearly the best routing network of that size. This *universality theorem* is proved using a three-dimensional VLSI model that incorporates wiring as a direct cost. In this model, hardware size is 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



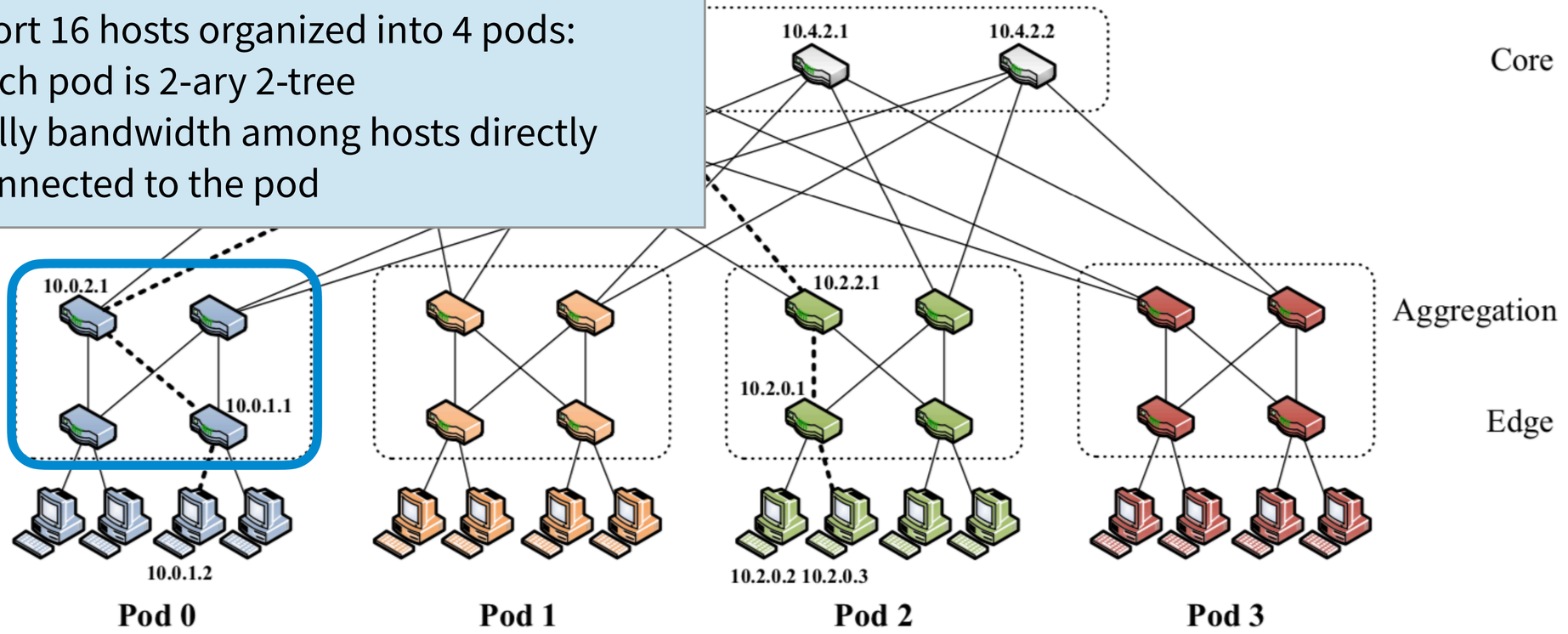
Fat-tree example



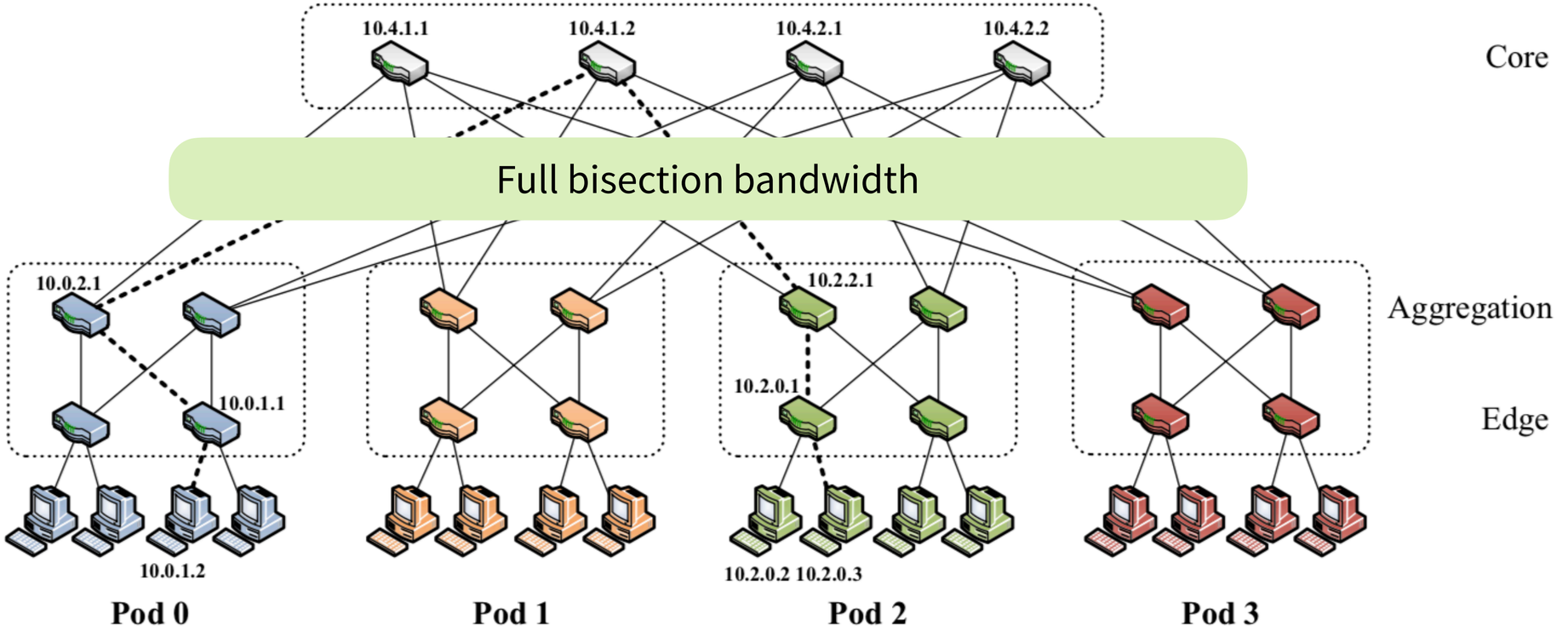
A fat-tree network built from 4-port **identical** switches

Fat-tree example

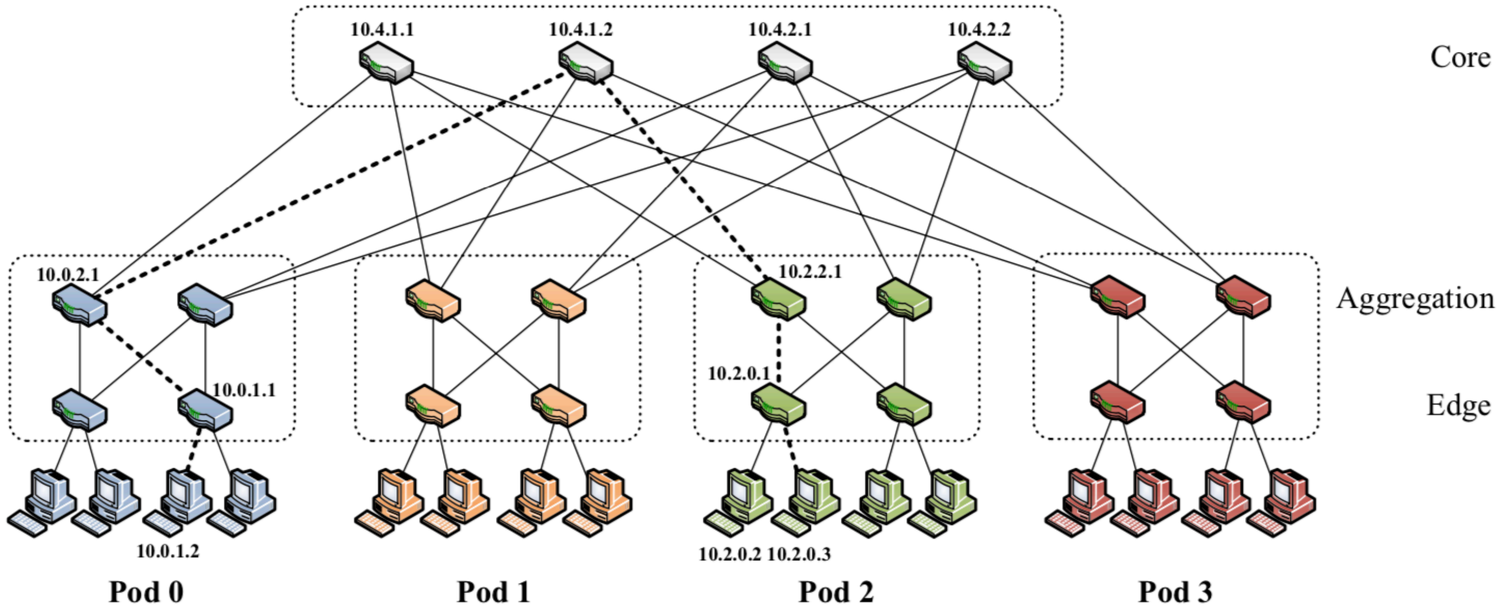
Support 16 hosts organized into 4 pods:
- Each pod is 2-ary 2-tree
- Fully bandwidth among hosts directly connected to the pod



Fat-tree example



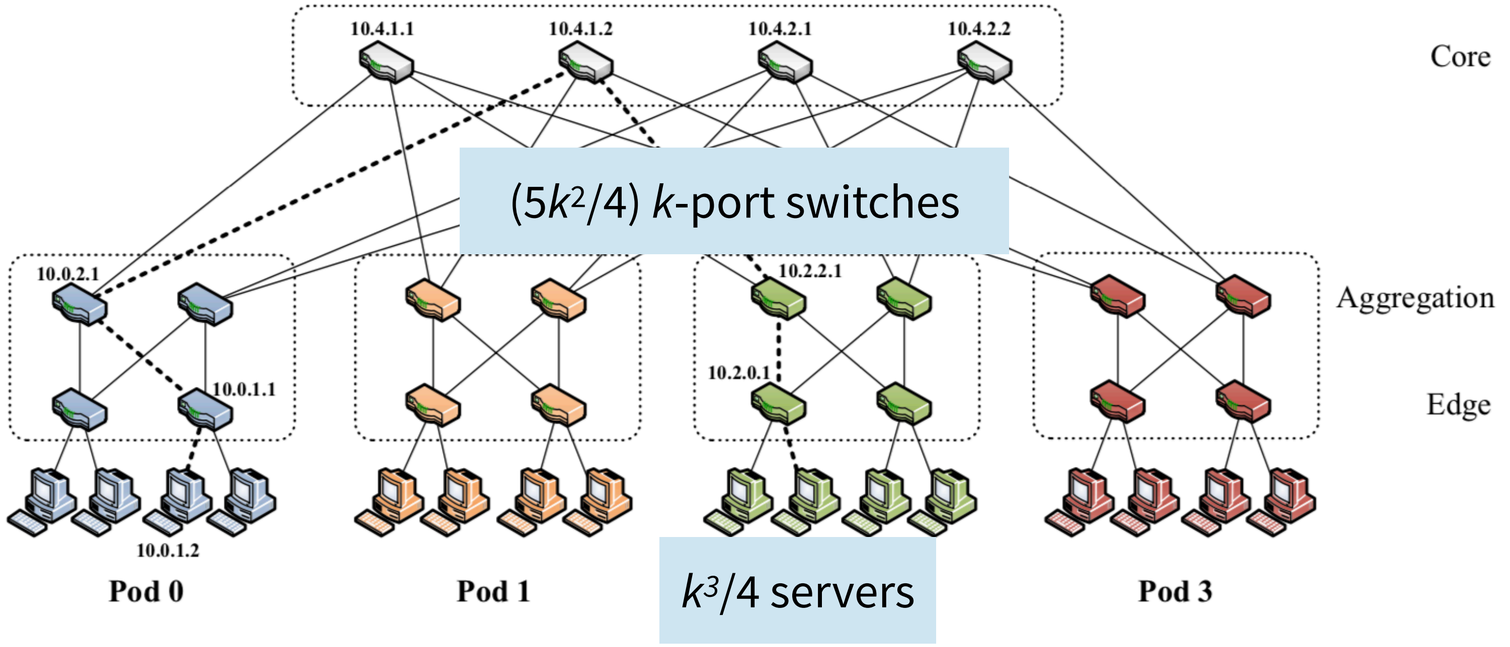
Fat-tree scalability



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

More generally, how to construct a fat-tree topology? (See Lab2)

Fat-tree scalability



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

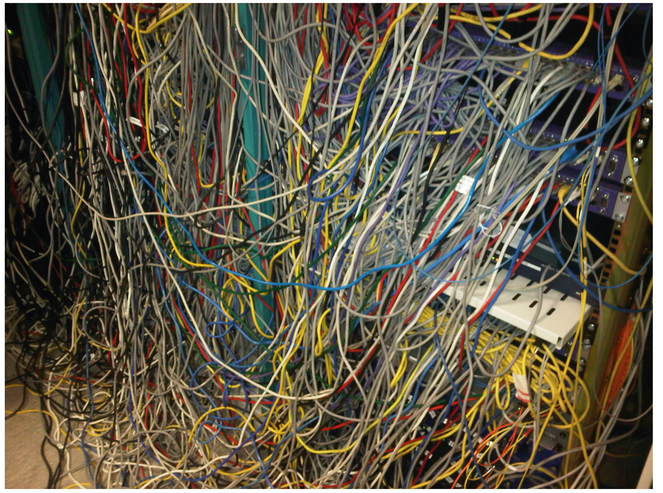
- Existing routing and forwarding protocols do not work for fat-tree
- Scalability challenges with millions of end points

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

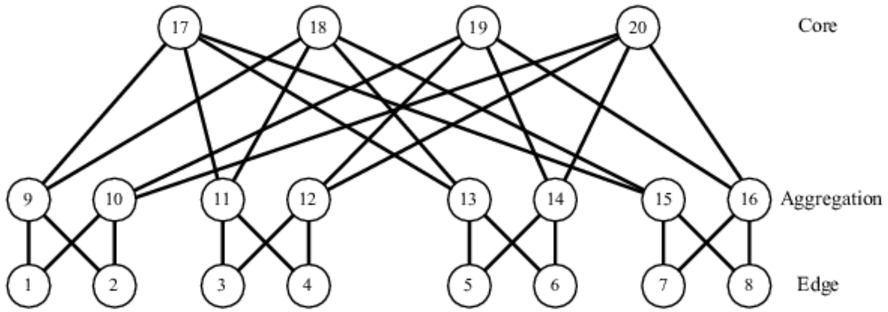


Challenges with fat-tree

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**

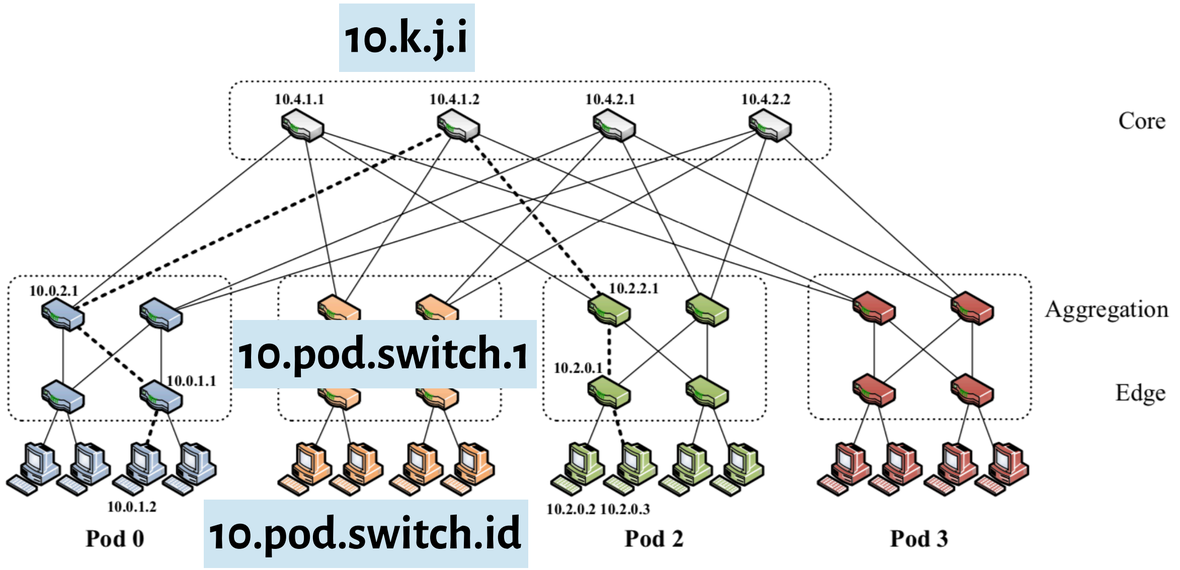
- Pod and switch between [0, *k*-1] based on the position

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



Forwarding

Two-level lookup table

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

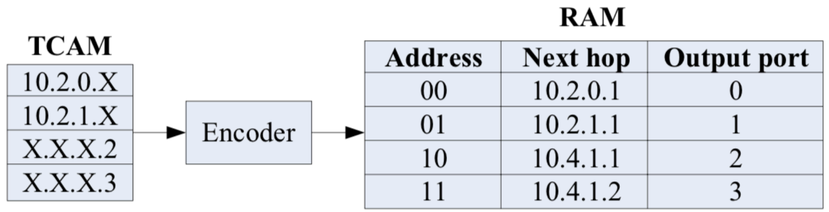
- Prefixes used for forwarding intra-pod traffic
- Suffixes used for forwarding inter-pod traffic

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

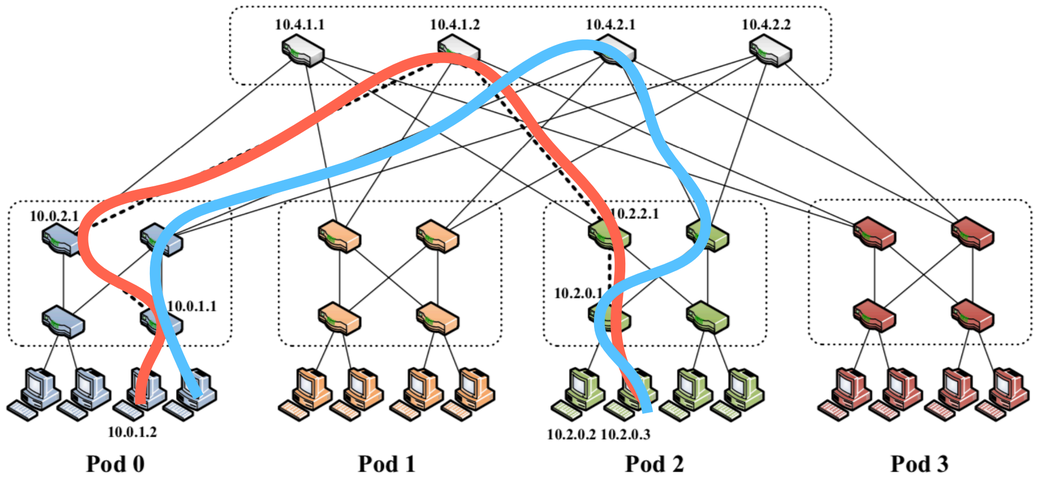
Prefixes in two-level lookup table prevent intra-pod traffic from leaving the pod

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

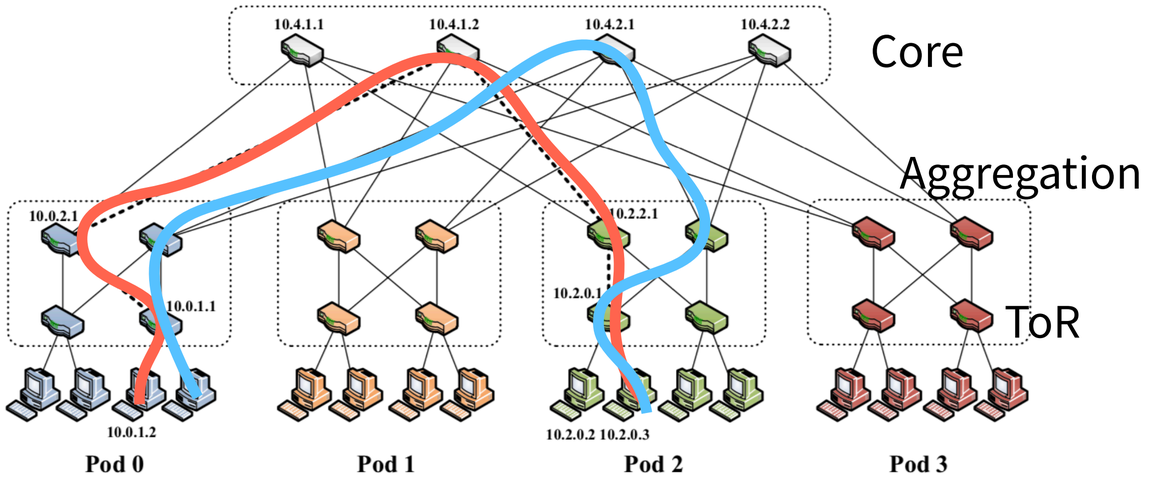
- Statically forward inter-pod traffic on specific port

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

- Switch value is the edge switch number

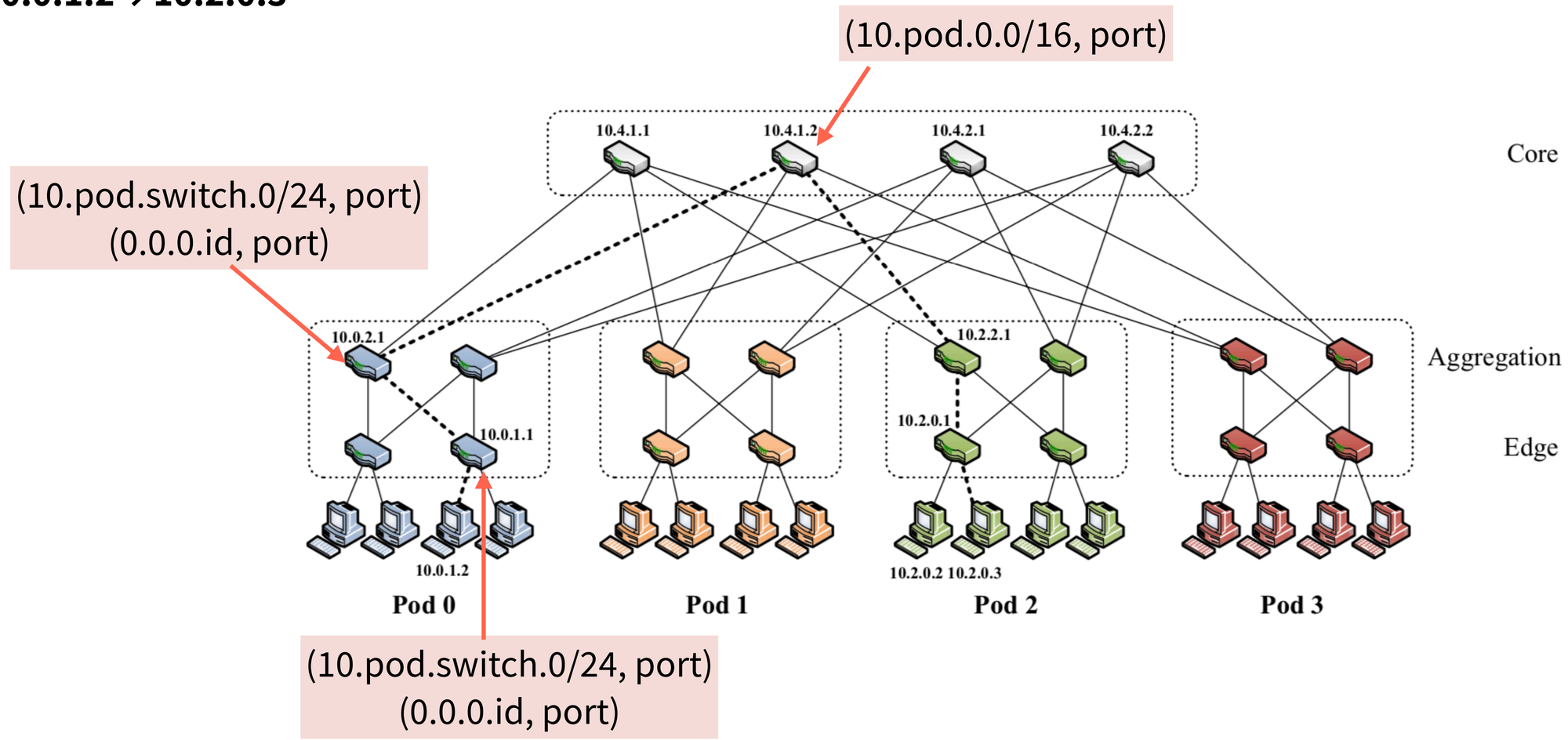
Assume a **central entity** with full knowledge of the topology generates these routing tables

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

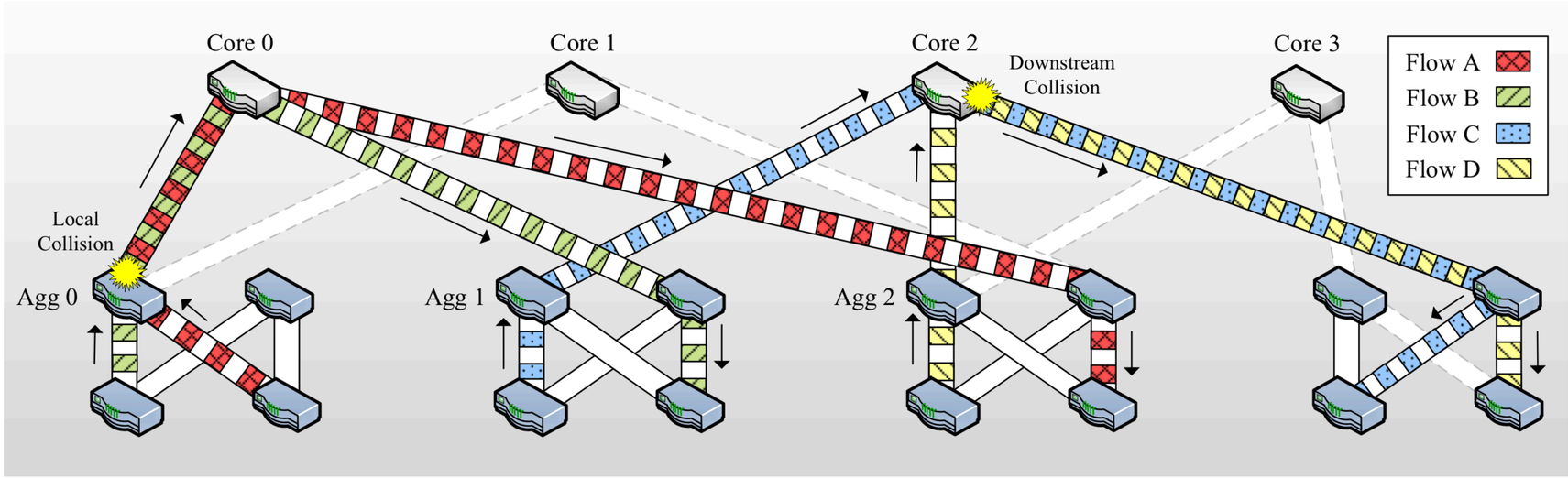


Routing example

10.0.1.2 → 10.2.0.3

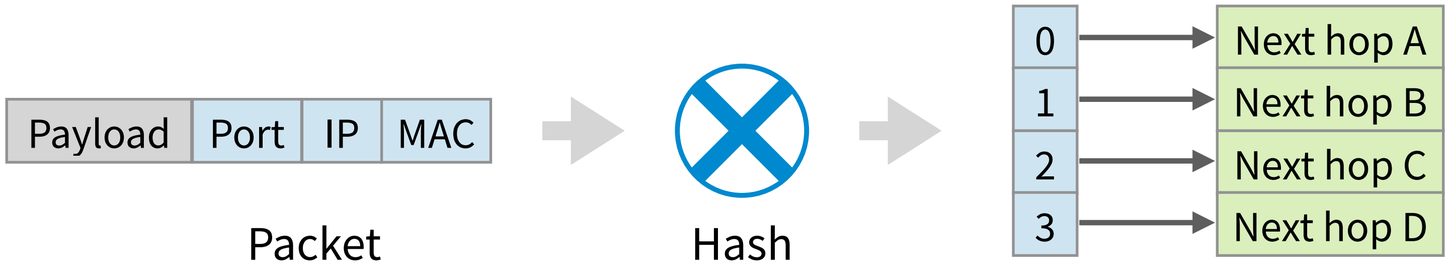


Flow collision



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

Solutions to flow collisions

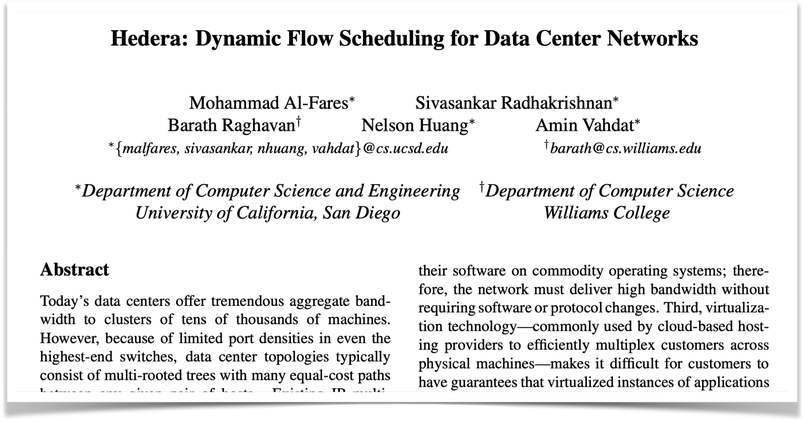


Equal-cost multi-path (ECMP)

- 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

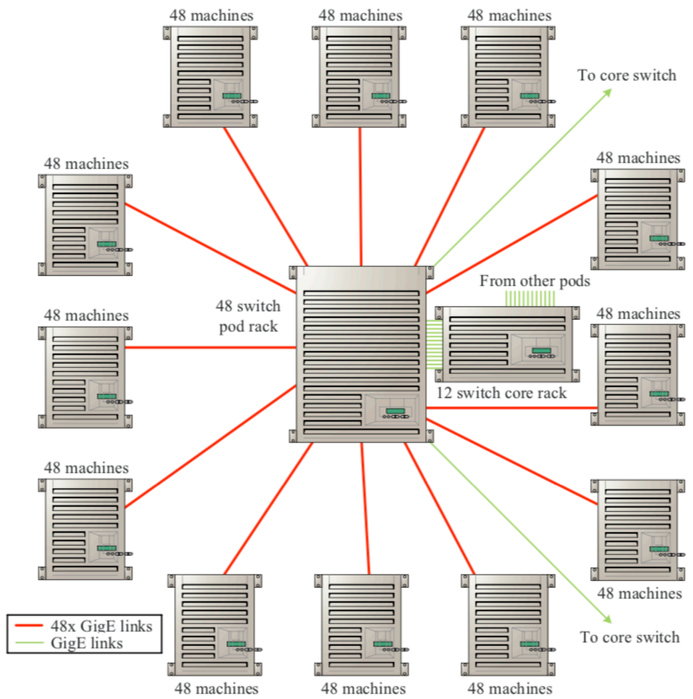
Abstract

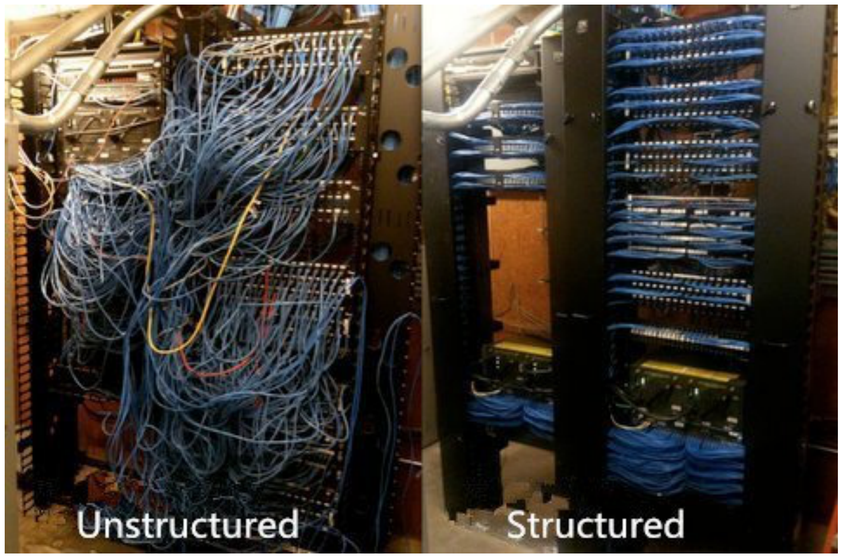
Today's data centers offer tremendous aggregate bandwidth to clusters of tens of thousands of machines. However, because of limited port densities in even the highest-end switches, data center topologies typically consist of multi-rooted trees with many equal-cost paths

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

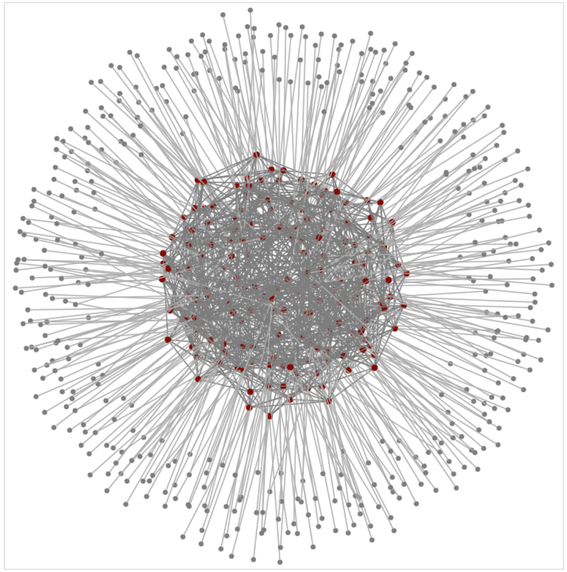


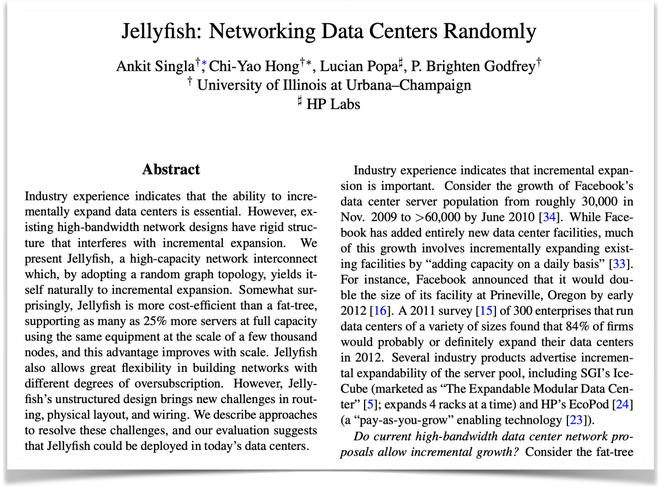


Organize switches into pod racks leveraging the regular structure of fat-tree

Fat-tree is quite regular, can we take the other extreme?

Can we general a completely random topology for the data center network? How would that topology perform compared with fat-tree? (See Lab2)





Jellyfish: Networking Data Centers Randomly

Ankit Singla[†], Chi-Yao Hong[†], Lucian Popa[♯], P. Brighten Godfrey[†]
[†] University of Illinois at Urbana–Champaign
[♯] HP Labs

Abstract

Industry experience indicates that the ability to incrementally expand data centers is essential. However, existing high-bandwidth network designs have rigid structure that interferes with incremental expansion. We present Jellyfish, a high-capacity network interconnect which, by adopting a random graph topology, yields itself naturally to incremental expansion. Somewhat surprisingly, Jellyfish is more cost-efficient than a fat-tree, supporting as many as 25% more servers at full capacity using the same equipment at the scale of a few thousand nodes, and this advantage improves with scale. Jellyfish also allows great flexibility in building networks with different degrees of oversubscription. However, Jellyfish's unstructured design brings new challenges in routing, physical layout, and wiring. We describe approaches to resolve these challenges, and our evaluation suggests that Jellyfish could be deployed in today's data centers.

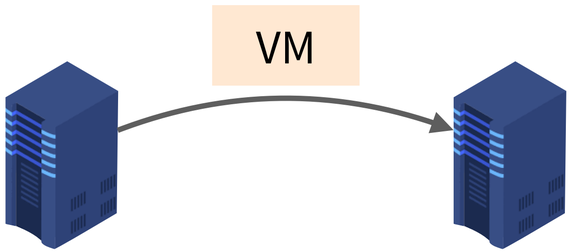
Industry experience indicates that incremental expansion is important. Consider the growth of Facebook's data center server population from roughly 30,000 in Nov. 2009 to >60,000 by June 2010 [34]. While Facebook has added entirely new data center facilities, much of this growth involves incrementally expanding existing facilities by "adding capacity on a daily basis" [33]. For instance, Facebook announced that it would double the size of its facility at Prineville, Oregon by early 2012 [16]. A 2011 survey [15] of 300 enterprises that run data centers of a variety of sizes found that 84% of firms would probably or definitely expand their data centers in 2012. Several industry products advertise incremental expandability of the server pool, including SGI's IceCube (marketed as "The Expandable Modular Data Center" [5]; expands 4 racks at a time) and HP's EcoPod [24] (a "pay-as-you-grow" enabling technology [23]).

Do current high-bandwidth data center network proposals allow incremental growth? Consider the fat-tree

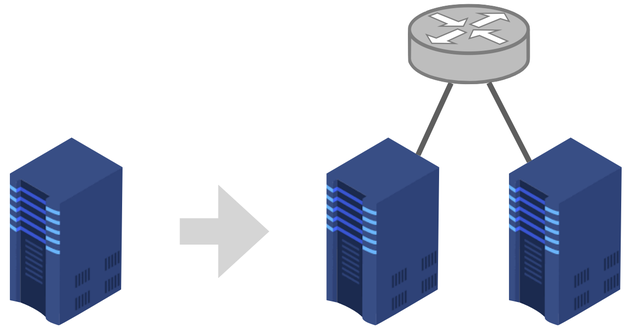
USENIX NSDI 2012

Questions?

Unaddressed issues in fat-tree: management flexibility



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	− Cannot do prefix-based matching as in IP routing	+
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)

10 million address mappings → ~100MB on-chip memory → ~150x over the limit

- Hierarchical address (IP address)

100-1000 address mappings (using prefix/suffix matching) → ~10KB of memory → easily accommodated in today's switches

PortLand

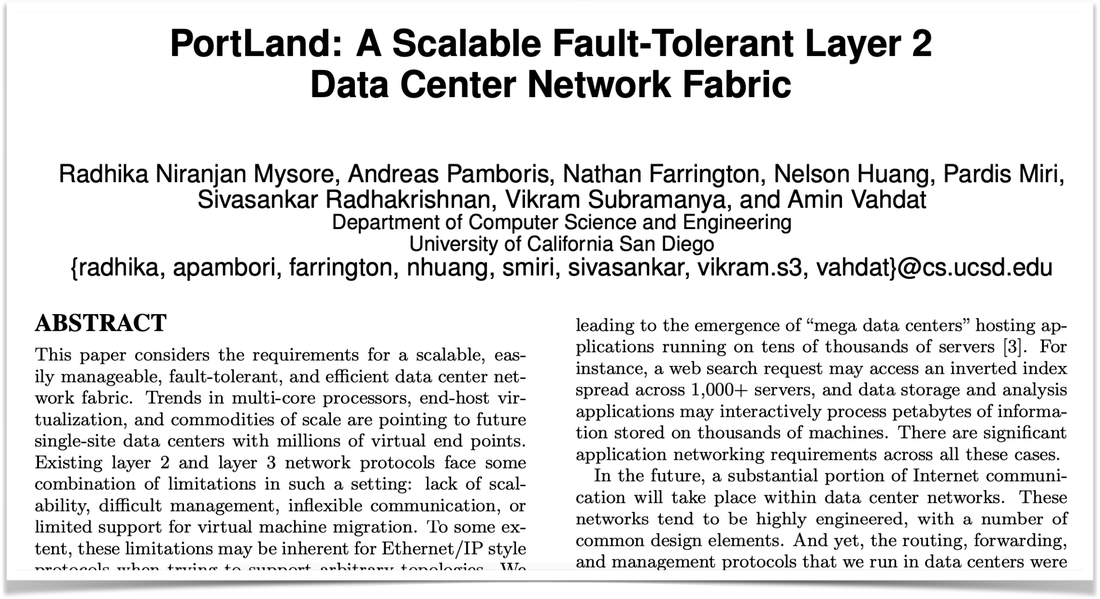
Main idea: separate node location from node identifier

- 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

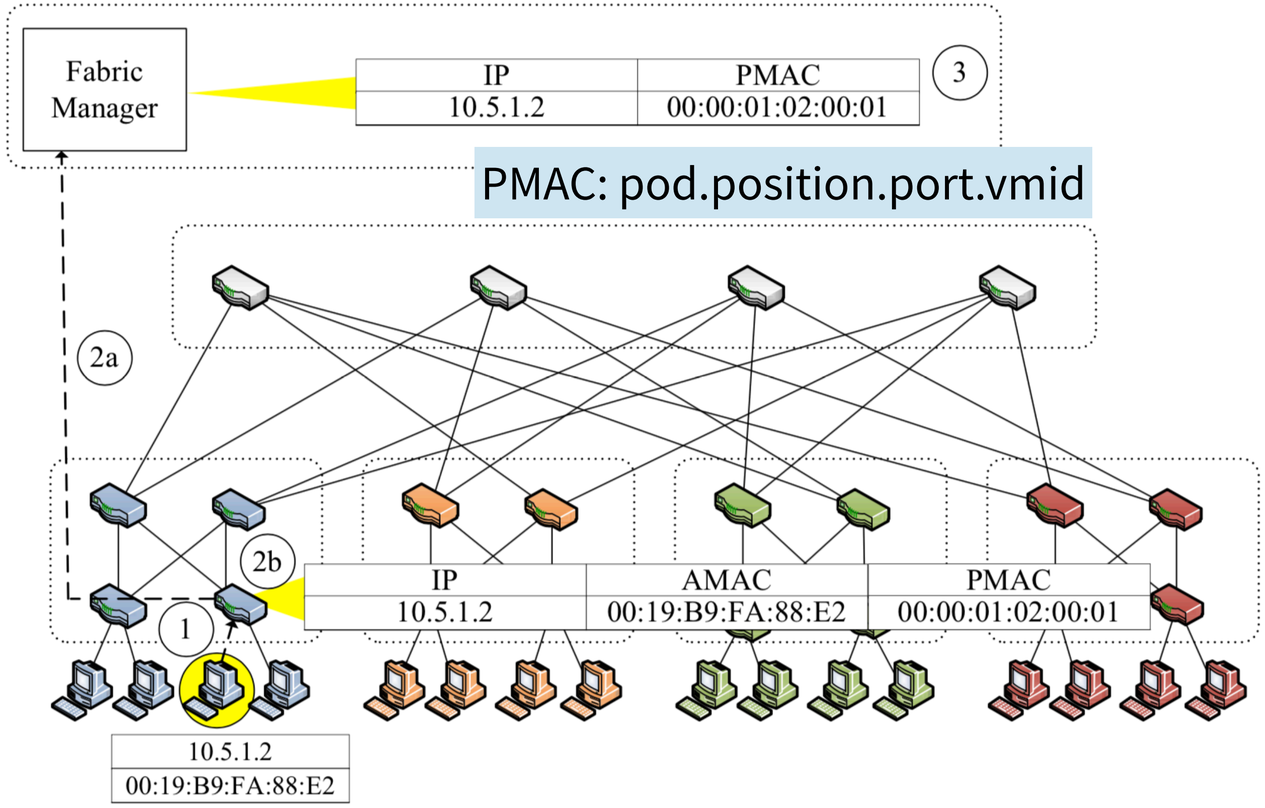
leading to the emergence of "mega data centers" hosting applications running on tens of thousands of servers [3]. For instance, a web search request may access an inverted index spread across 1,000+ servers, and data storage and analysis applications may interactively process petabytes of information 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):**

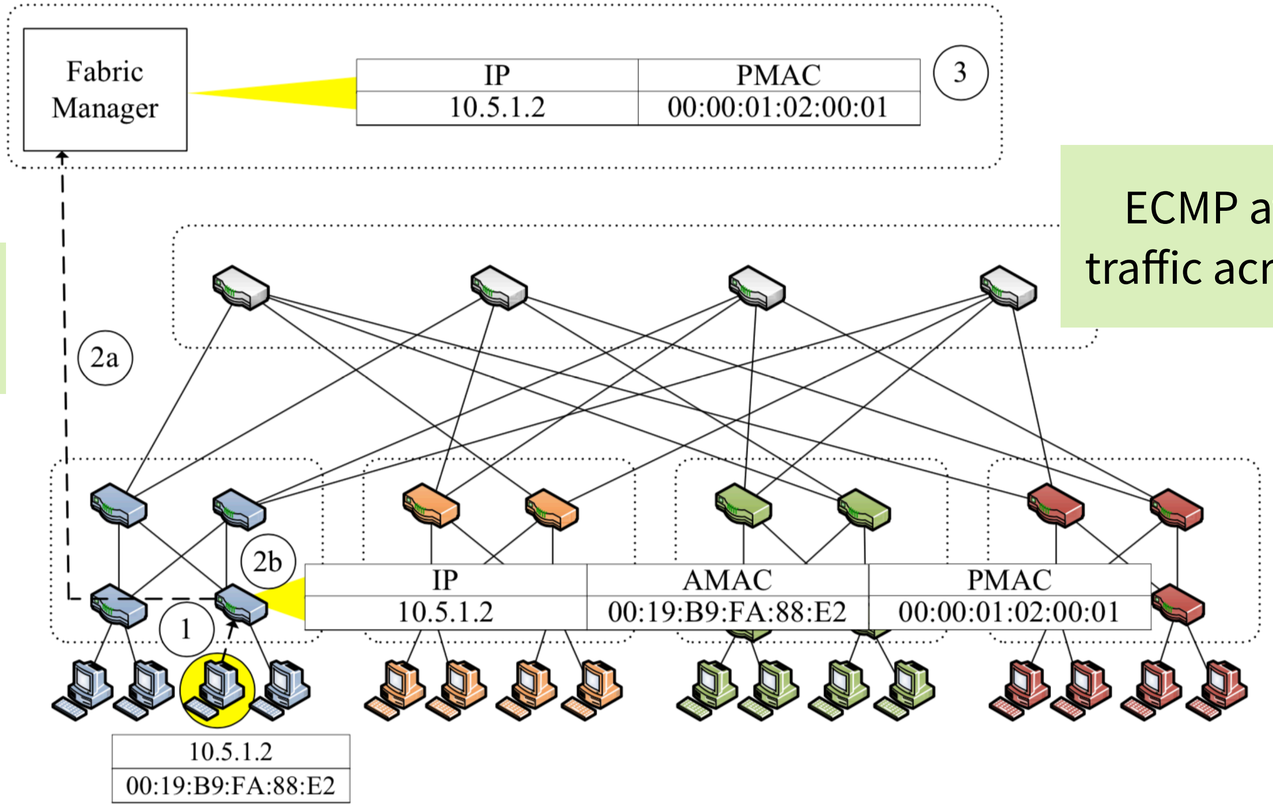
- Tree-level/role: based on neighbor identify
- Pod number: fetch from the Fabric manager
- Position number: aggregation switches help ToR switches choose unique position number

Fabric manager

ARP mappings

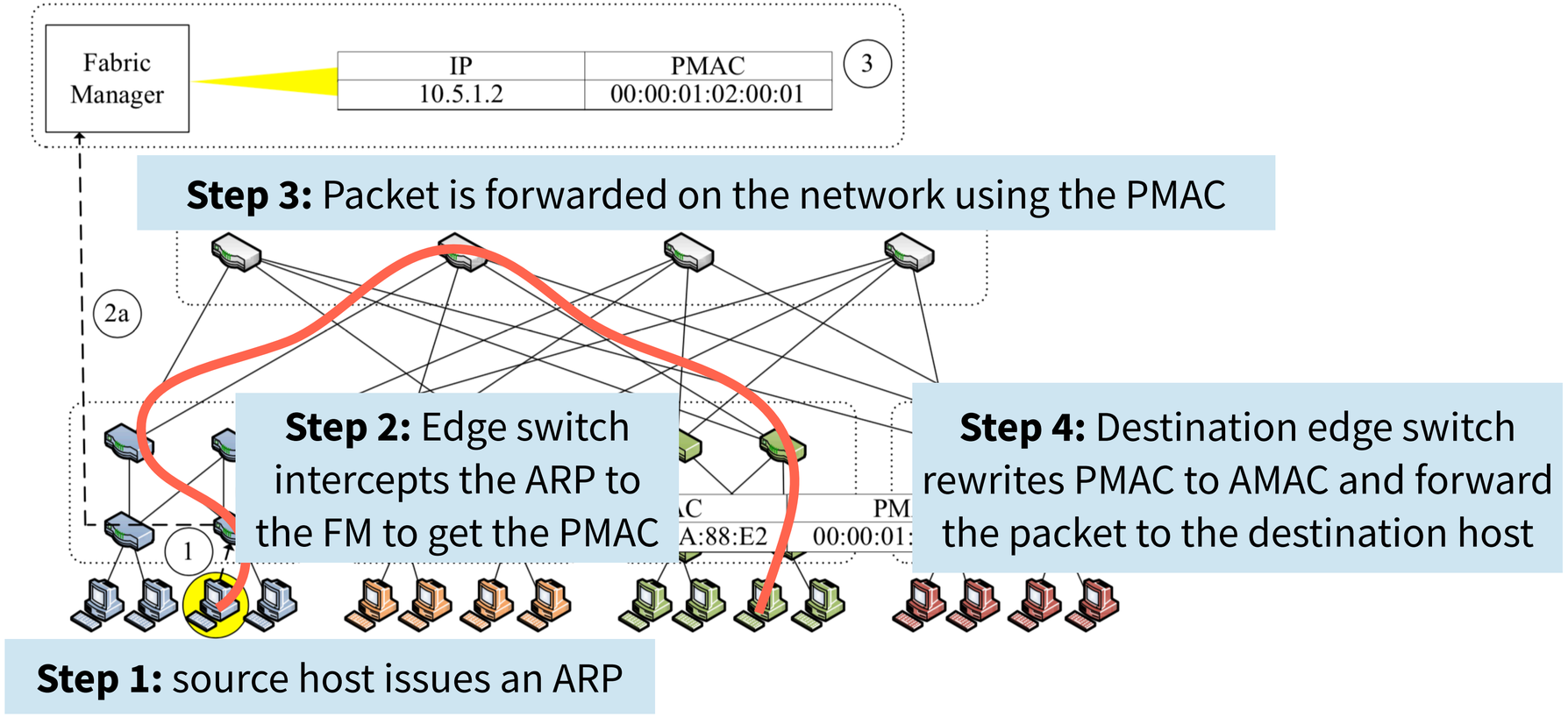
Network map

Soft state: no need for manual configurations



ECMP at the core to spread traffic across the core switches

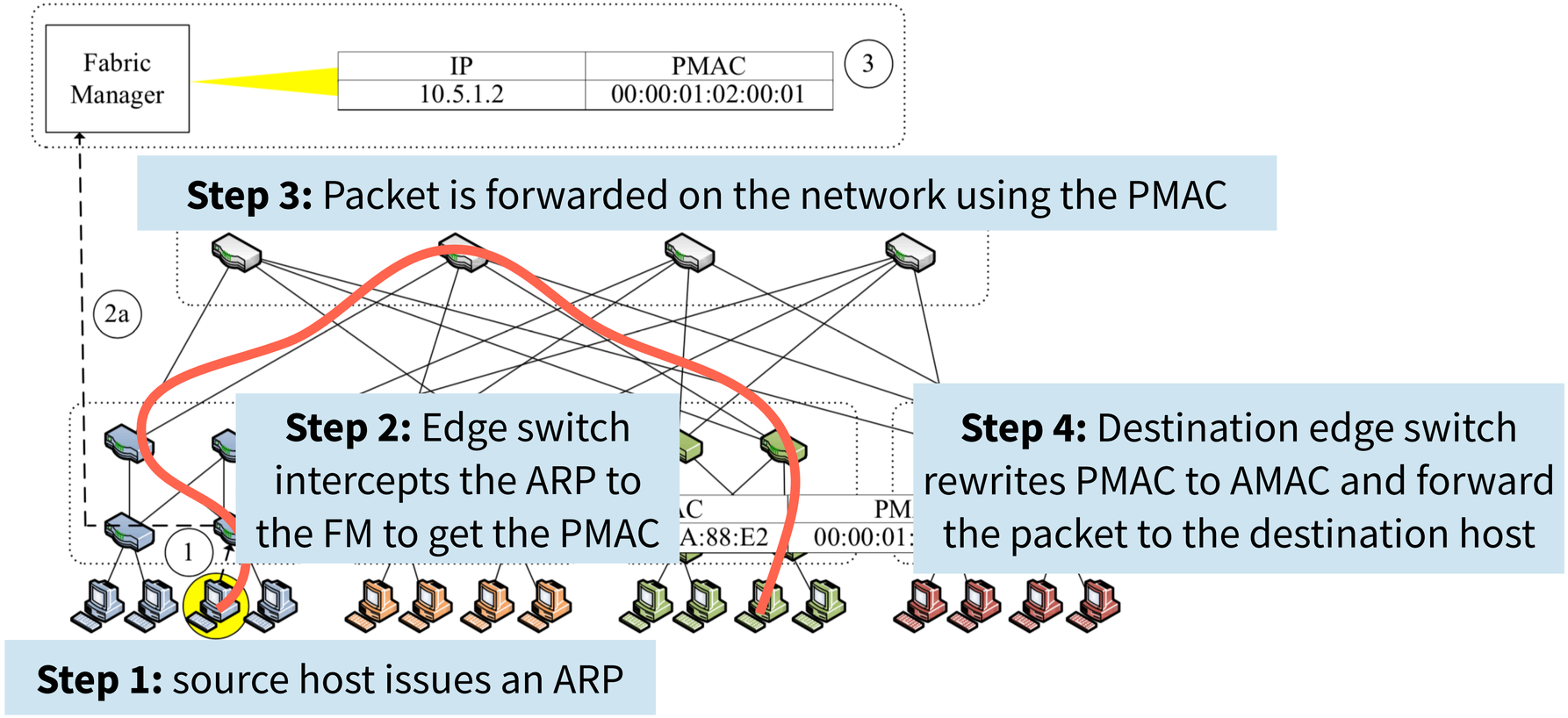
PortLand workflow



PortLand workflow

Hardware support:

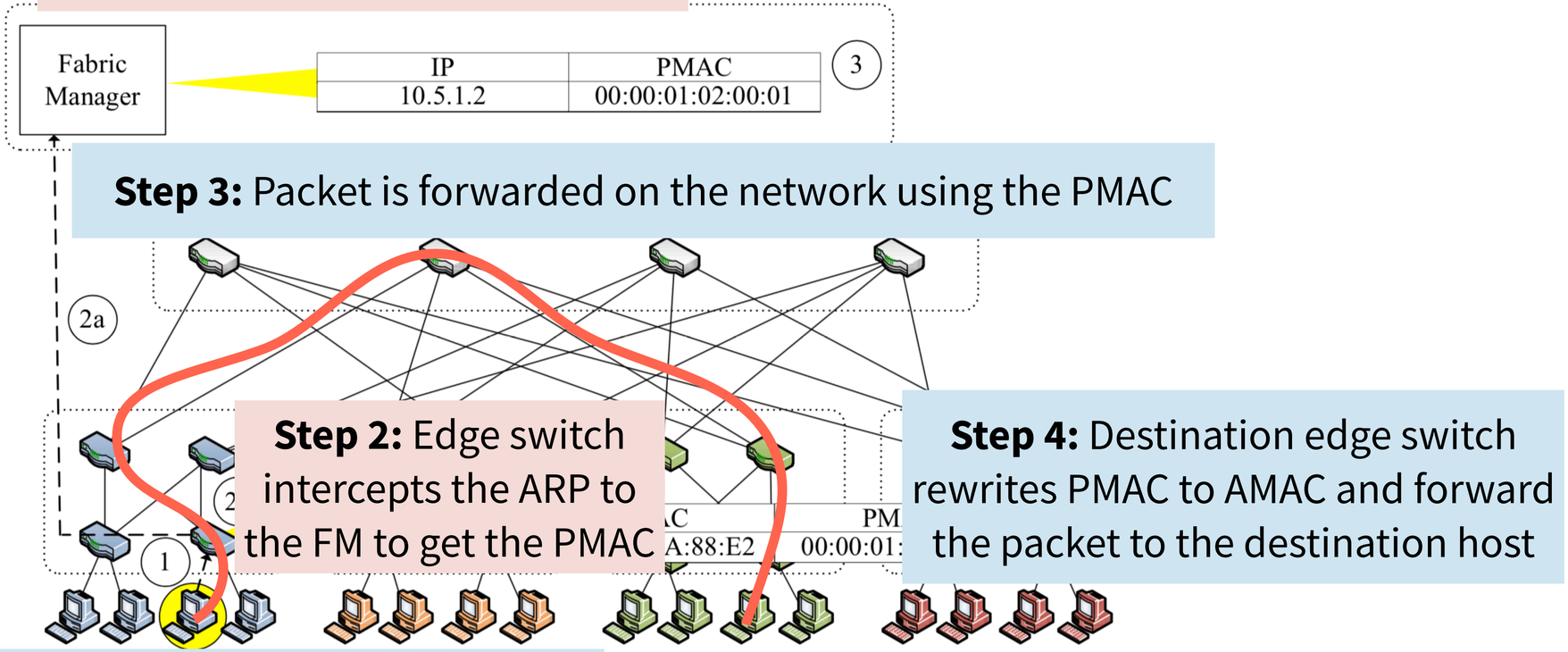
- No modification needed for hosts
- PMAC <> AMAC translation on edge switches
- Other switches forward based on prefix-matching on PMAC



PortLand workflow

How does the FM populate the IP-PMAC table?

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



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

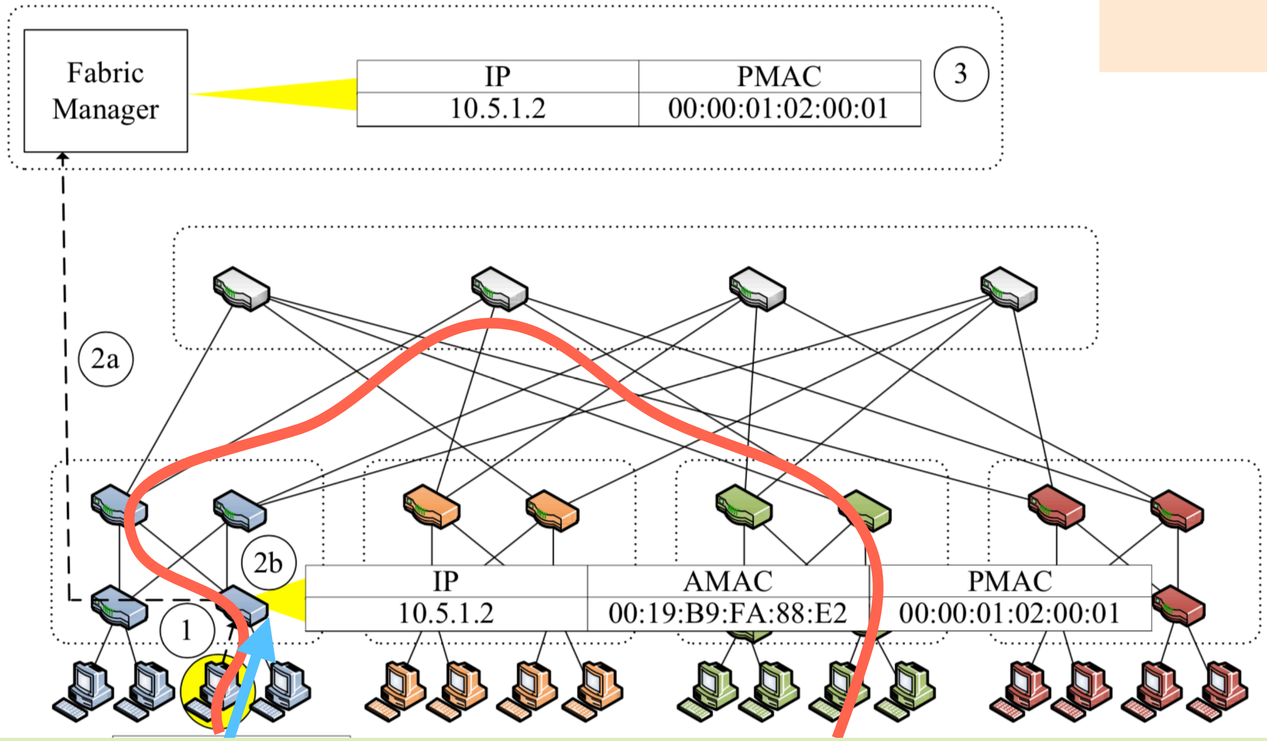
Step 2: Edge switch intercepts the ARP to the FM to get the PMAC

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

Summary

Data center networking

- Topology, performance (bisection bandwidth, over-subscription ratio)
- Architecture design: fat-tree
- Routing in fat-tree
- L2 vs. L3 addressing for data center networking
- PortLand design
- Forwarding and routing in PortLand

Next time: data center transport