Advanced Computer Networks

Data Center Networking

Lin Wang

Period 2, Fall 2022

VU VRIJE UNIVERSITEIT AMSTERDAM

Course outline

Essentials

- Introduction (history, principles)
- Networking basics
- Network transport

Data center networking

- **Data center networking**
- Data center transport
- Software defined networking
- Programmable data plane

Network innovations

- Networking data structures
- Network monitoring
- Programmable switch architecture
- In-network computing - applications
- In-network computing - hardware
- Machine learning for networking

Guest lecture

- Fernando Ramos (University of Lisbon)

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

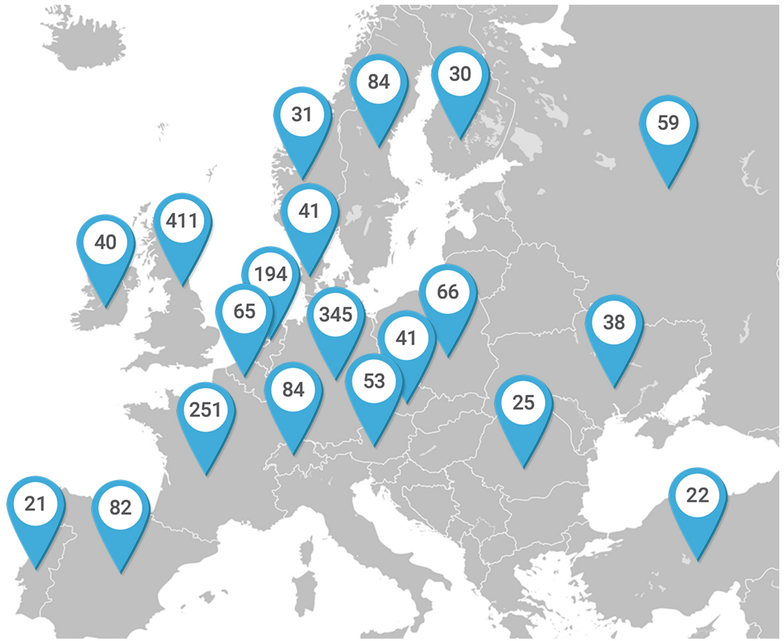
- Multiple independent users, resource isolation
- Amortize the cost of the shared infrastructure

Flexible service management

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

What is behind cloud computing?

Large-scale data centers



Data center distribution in Europe

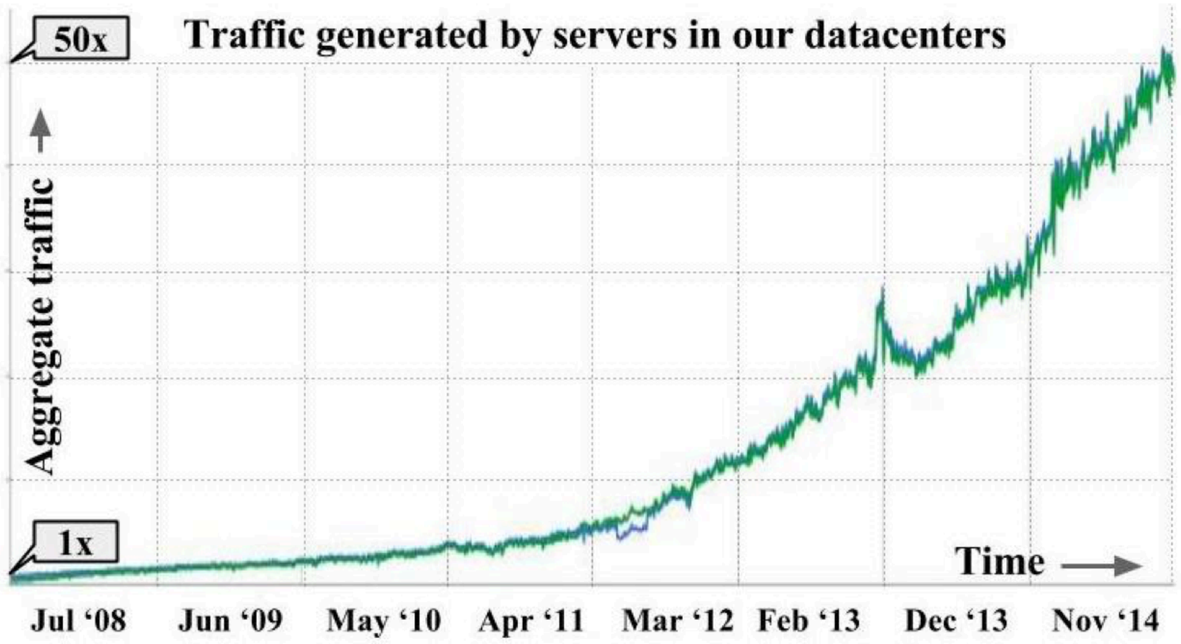
Equinix’s AM4 data center @ Science Park

How does a data center look inside?

Equinix’s AM4 data center @ Science Park

Well organized & interconnected racks of servers!

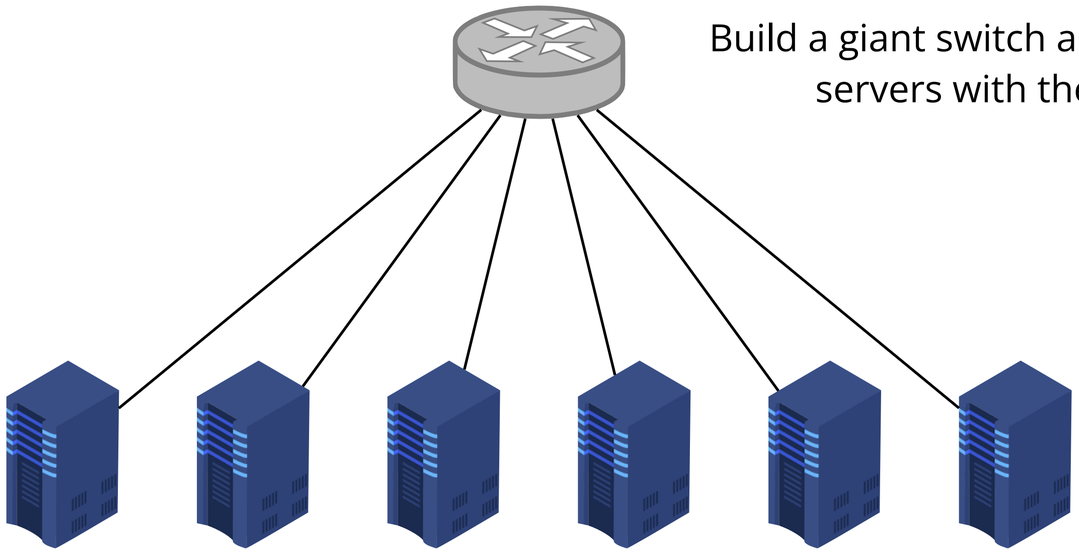
Aggregated server traffic in Google’s data centers



ACM SIGCOMM 2015

How to interconnect servers efficiently in a data center?

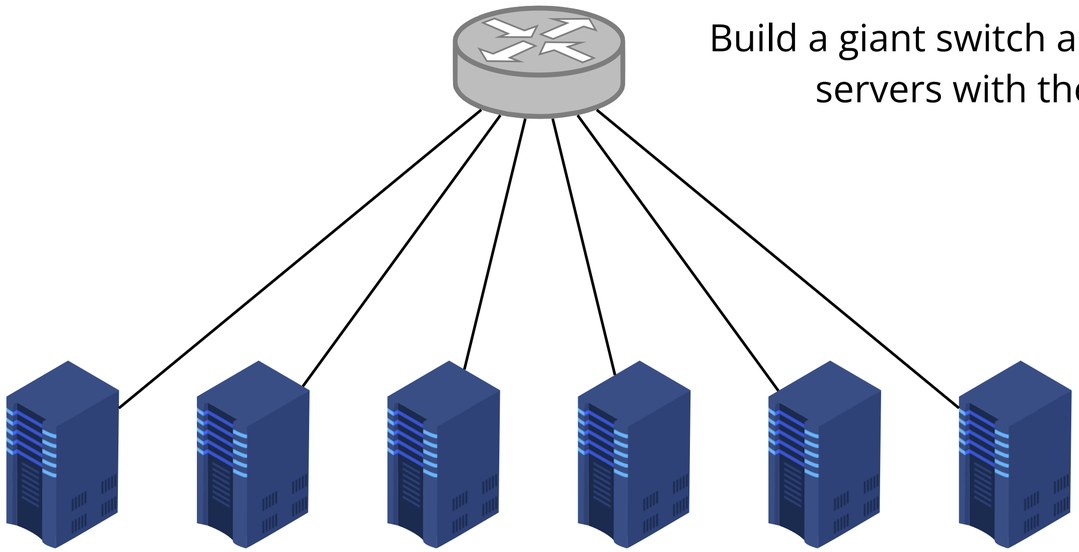
How to connect these servers in a data center?



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?

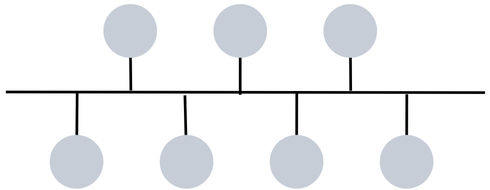


Build a giant switch and connect all servers with the switch

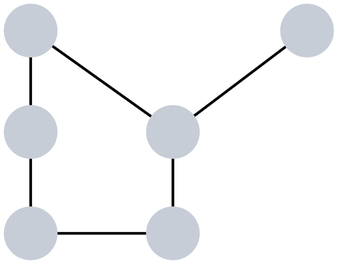
Limited port density, broadcast storms, isolation…

A dedicated network for the data center

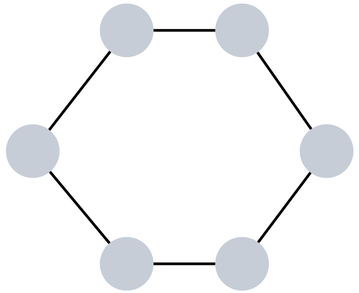
Line



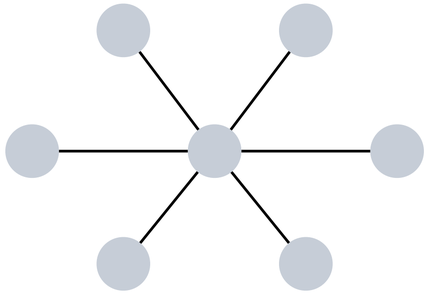
Bus



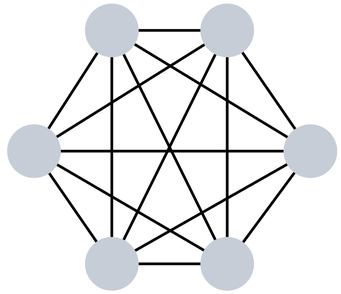
Mesh



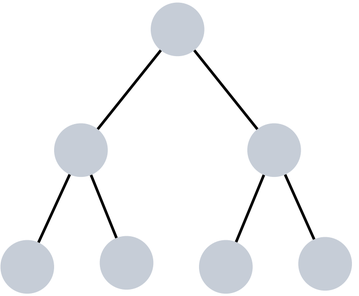
Ring



Star

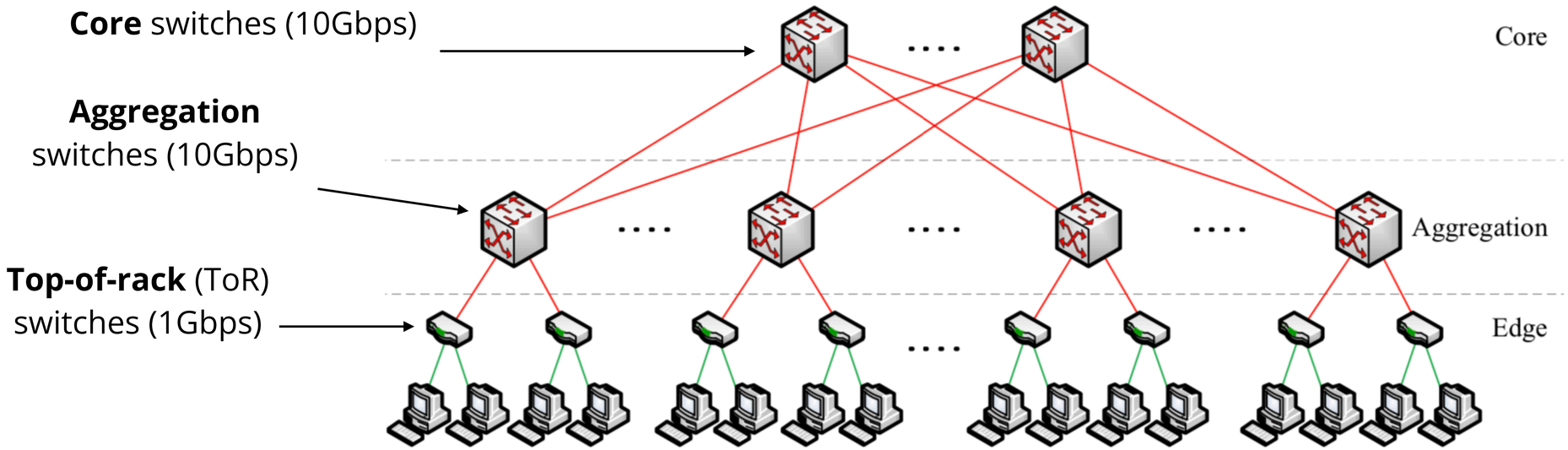


Fully connected



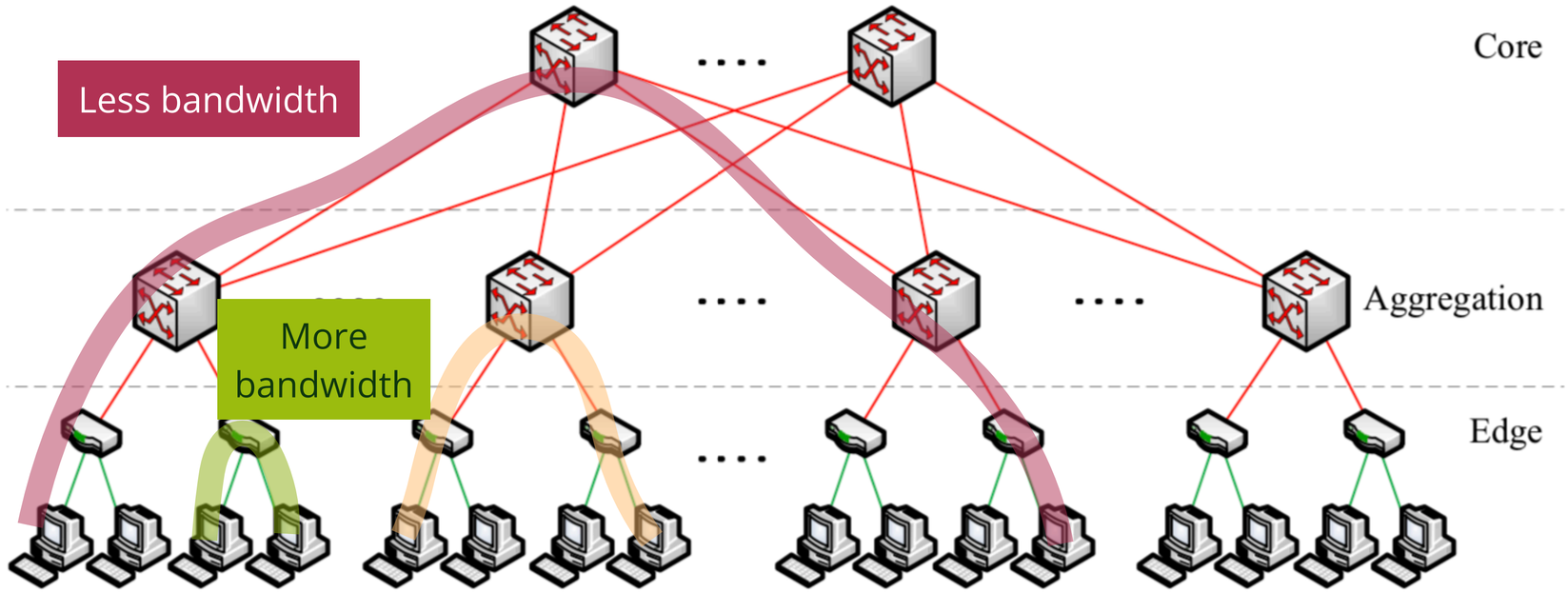
Tree

A tree-based data center network



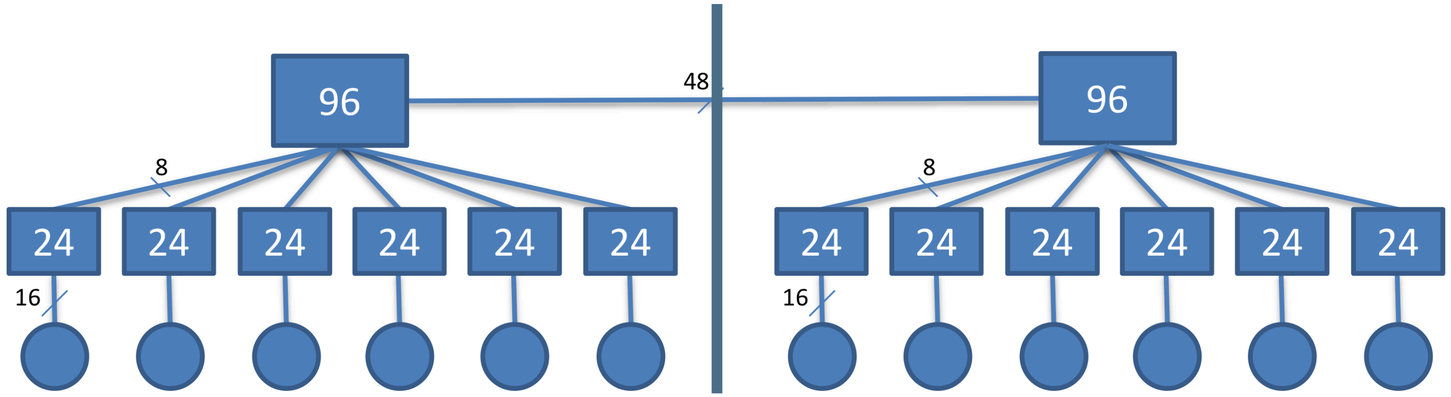
What if ToR switches go for 10Gbps or beyond?

Bottleneck in tree-based networks



How to quantitatively measure the connectivity?

Network performance metrics



Bisection width	The minimum number of links cut to divide the network into two halves
Bisection bandwidth	The minimum bandwidth of the links that divide the network into two halves
Full bisection bandwidth	Nodes in one half can communicate simultaneously with nodes in the other half, at their full uplink capacity

Oversubscription ratio

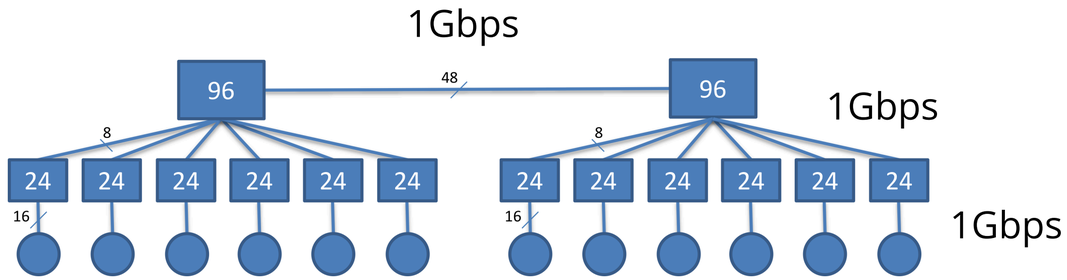
Definition

- Ratio of worst-case required aggregate bandwidth to the total uplink bandwidth of a network device
- 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 oversubscription ratio is 2.5:1 to 8:1



What is the oversubscription ratio in the above topology?

Oversubscription ratio

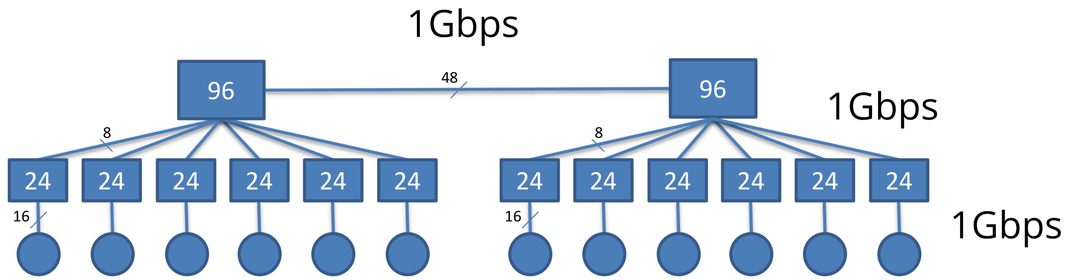
Definition

- Ratio of worst-case required aggregate bandwidth to the total uplink bandwidth of a network device
- 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 oversubscription ratio is 2.5:1 to 8:1



Oversubscription ratio at the aggregation layer: 16 x 6 / 48 = 2:1

Oversubscription ratio at the core layer: 8 x 6 / 48 = 1:1

Factors behind data center network designs

Commoditization in the data center

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

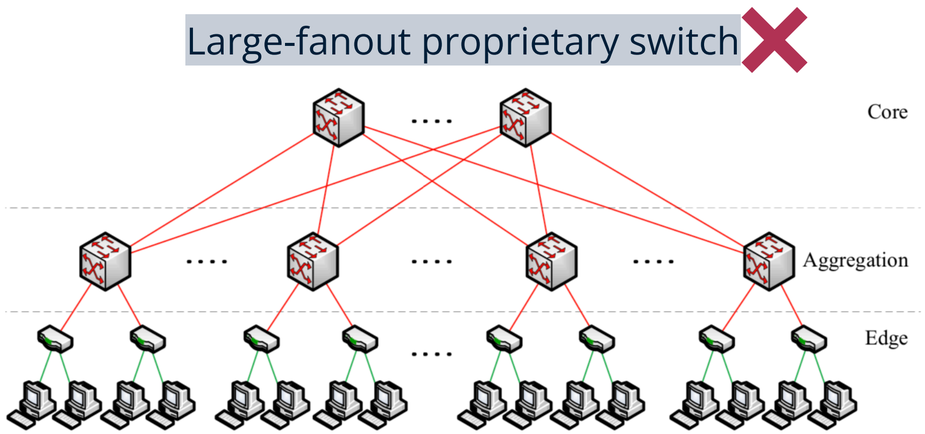
Data center is **not** a “small Internet”

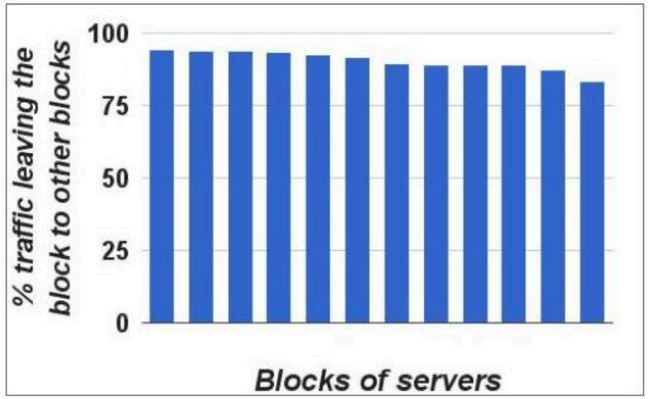
- One admin domain, not adversarial, limited policy routing, etc…

Bandwidth is often the bottleneck

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

Large-fanout proprietary switch ✗

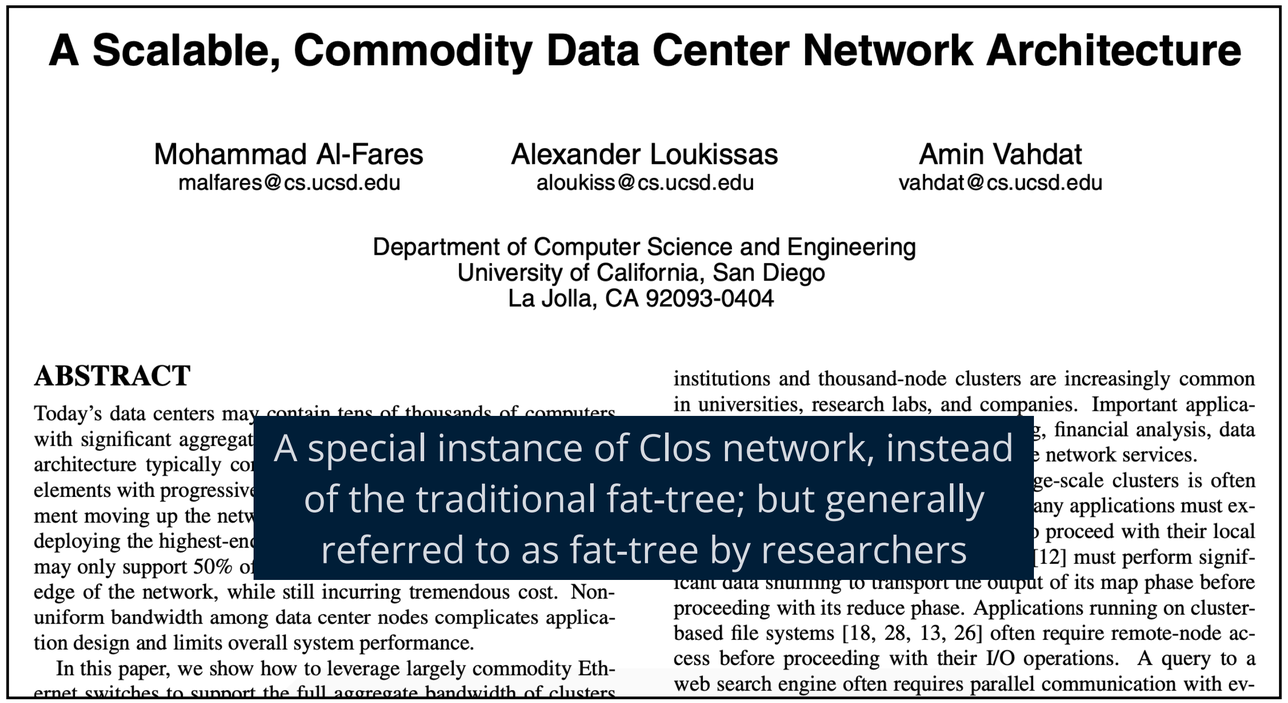




Low traffic locality

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 aggregat architecture typically co elements with progressive ment moving up the netw deploying the highest-en may only support 50% o 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

A special instance of Clos network, instead of the traditional fat-tree; but generally referred to as fat-tree by researchers

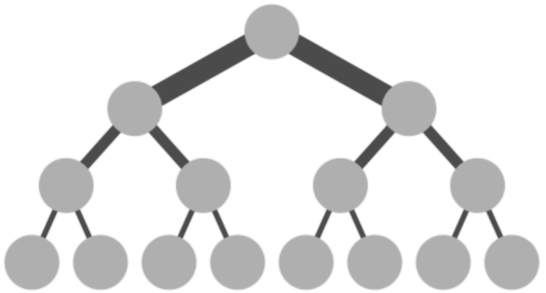
institutions and thousand-node clusters are increasingly common in universities, research labs, and companies. Important applica- g, financial analysis, data e network services. ge-scale clusters is often any applications must ex- proceed with their local 12] must perform signif- icant data shuffling 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-

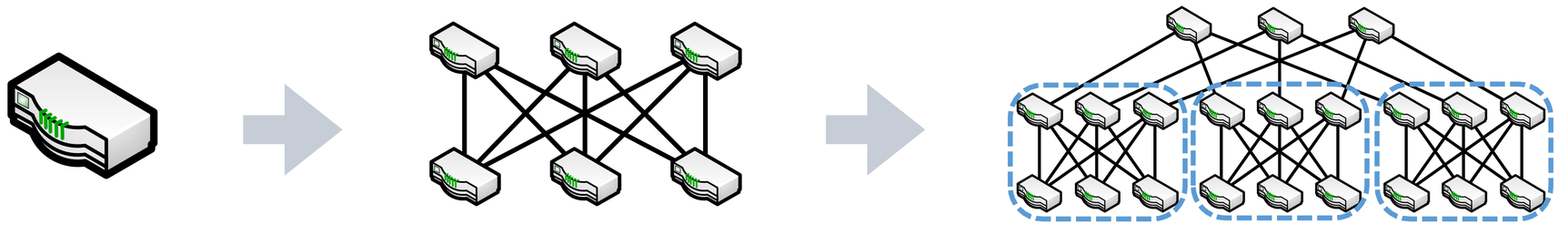
ACM SIGCOMM 2008

Fat-tree 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
- Invented in 1938 by Edson Erwin and formalized by Charles Clos in 1953
- Fat-tree was proposed by Charles Leiserson in 1985, which means a different topology (shown in the right side)





Fat-tree: design goals

Scalable interconnection bandwidth

- **Full bisection bandwidth** between all pairs of hosts (what is the expected oversubscription ratio?)

Economies-of-scale

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

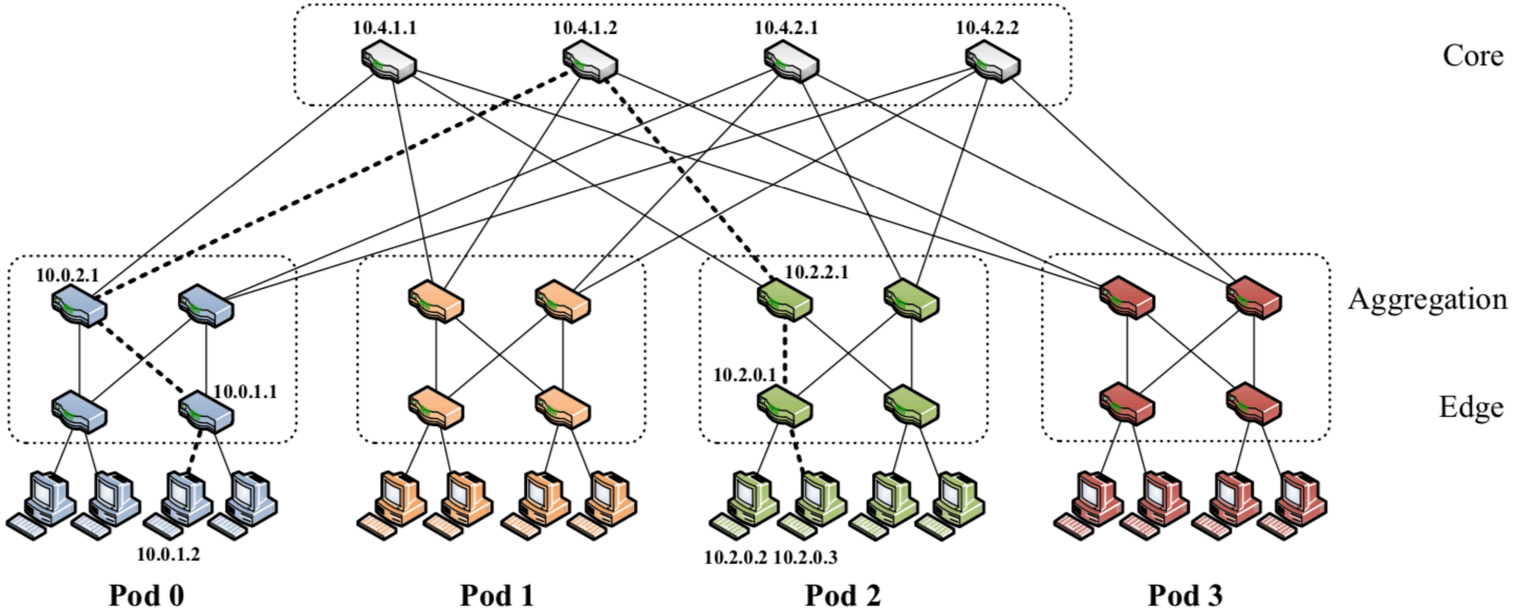
Compatibility

- Support Ethernet and IP without host modifications

Easy management

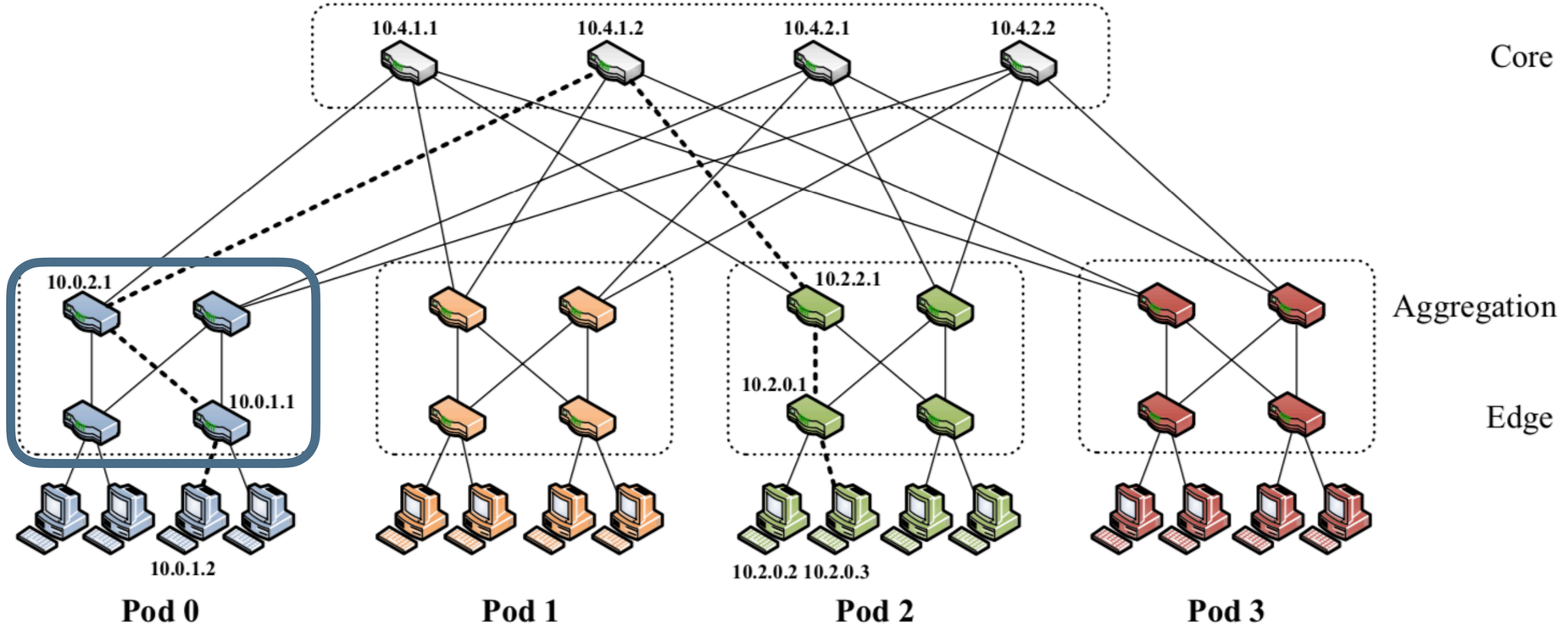
- Modular design, avoid manual management

Fat-tree example



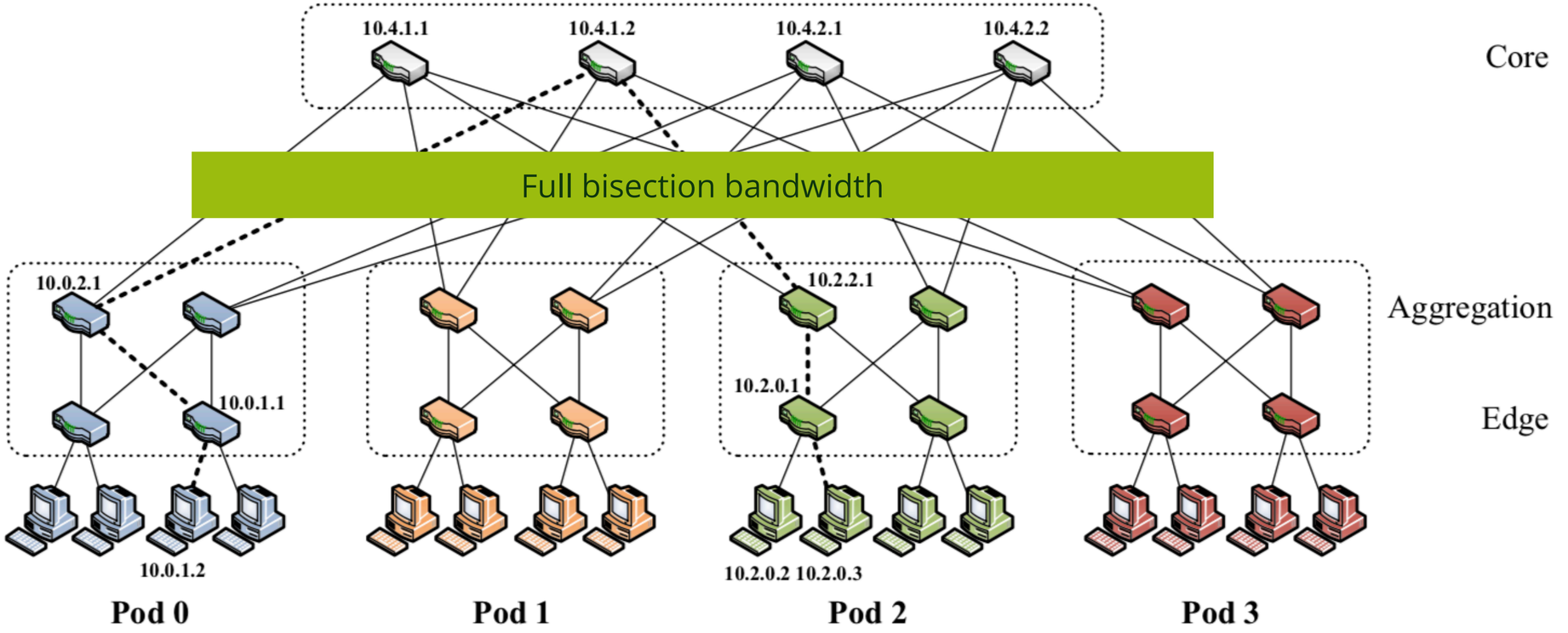
A fat-tree network built from 4-port **identical** switches

Fat-tree example

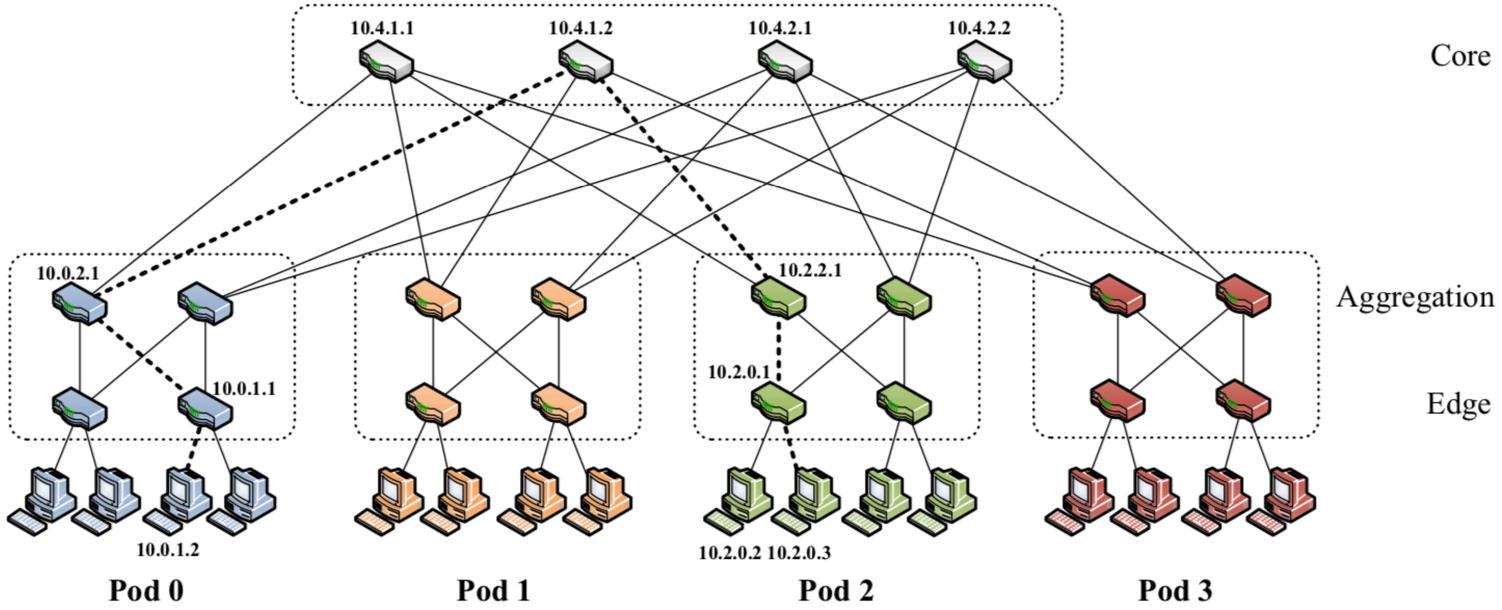


Support 16 hosts organized into 4 pods:
- Each pod is 2-ary 2-tree
- Full bandwidth among hosts directly connected to the pod

Fat-tree example

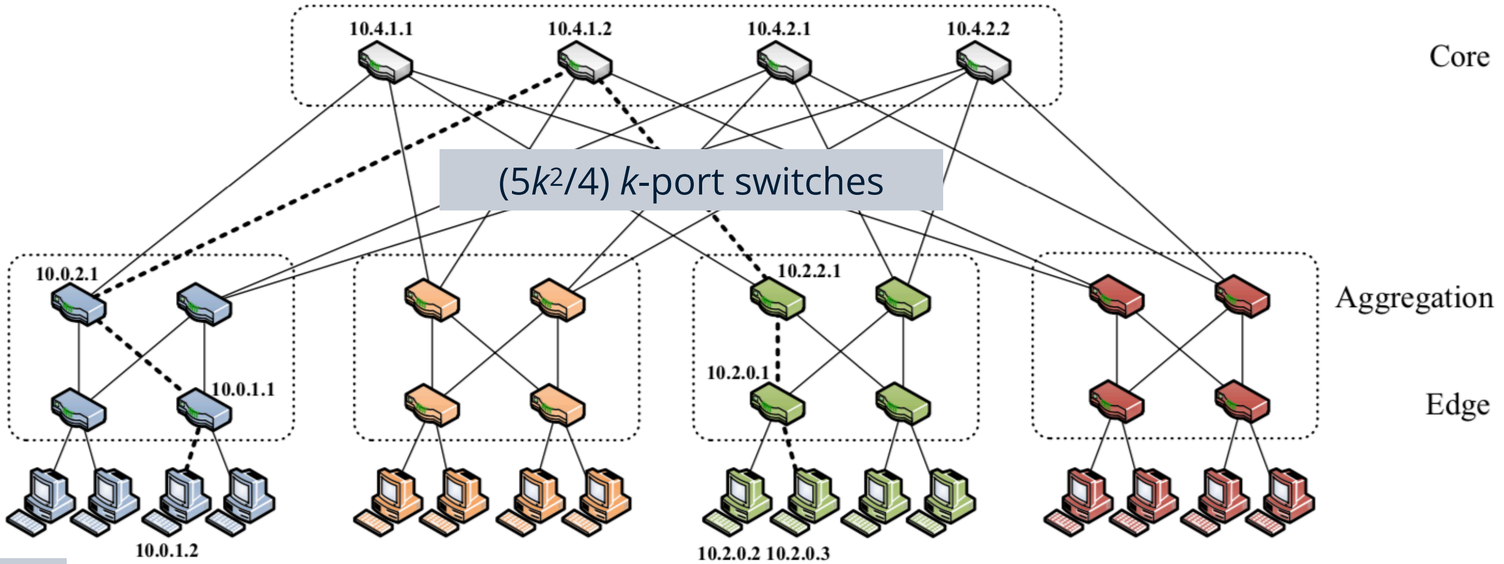


Fat-tree scalability



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

Fat-tree scalability



Why this has not been done before?

Recall fat-tree was proposed in 1985!!

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

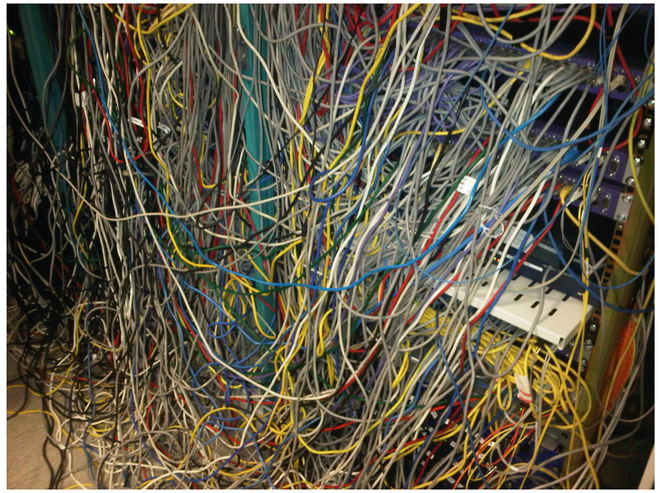
- Existing routing and forwarding protocols do not work for fat-tree
- Scalability challenges with millions of end points

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

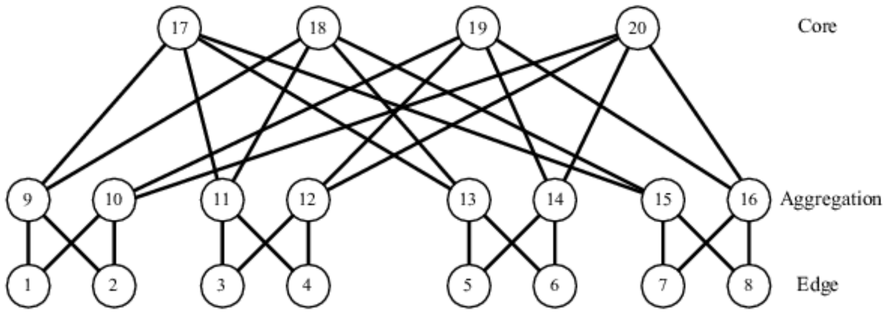


Challenges with fat-tree

Backward compatible with IP/Ethernet

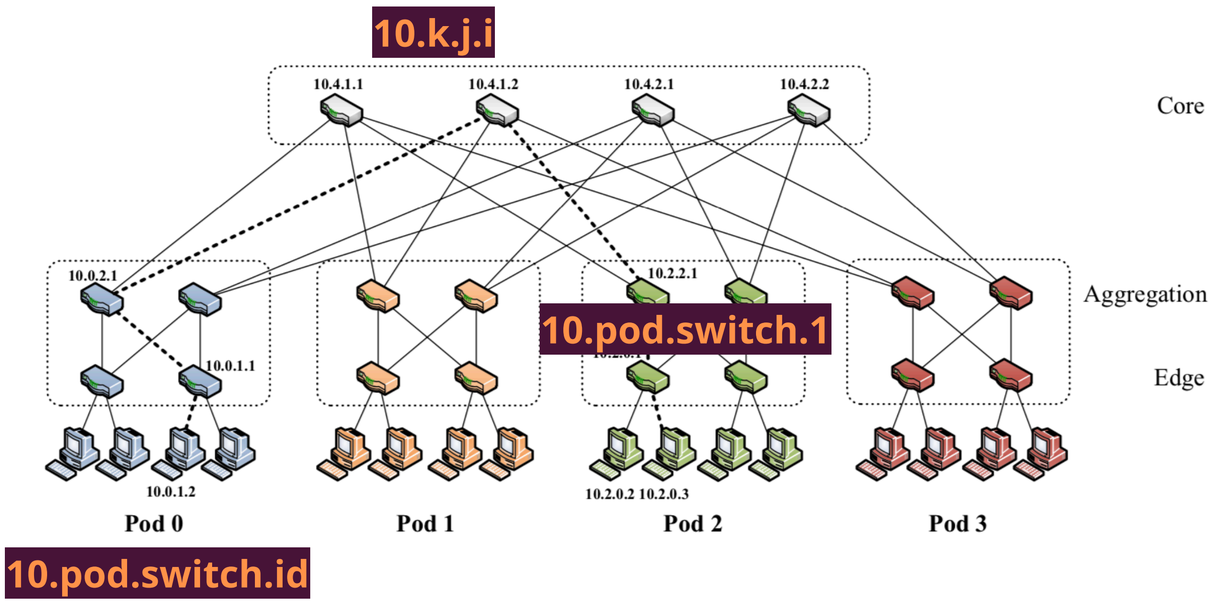
- Routing algorithms (such as OSPF) will naively choose a single shortest path to use between subnets
- Leads to bottleneck quickly
- $(k/2)^2$ shortest paths available, should use them all equally

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

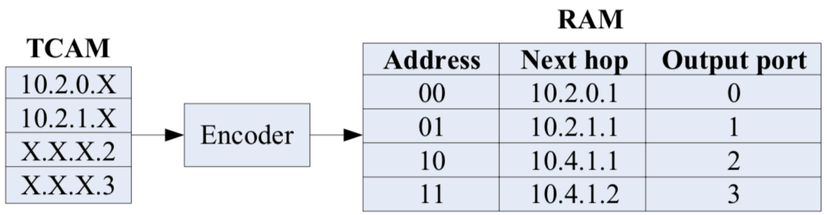
- Prefixes used for forwarding intra-pod traffic
- Suffixes used for forwarding inter-pod traffic

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

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

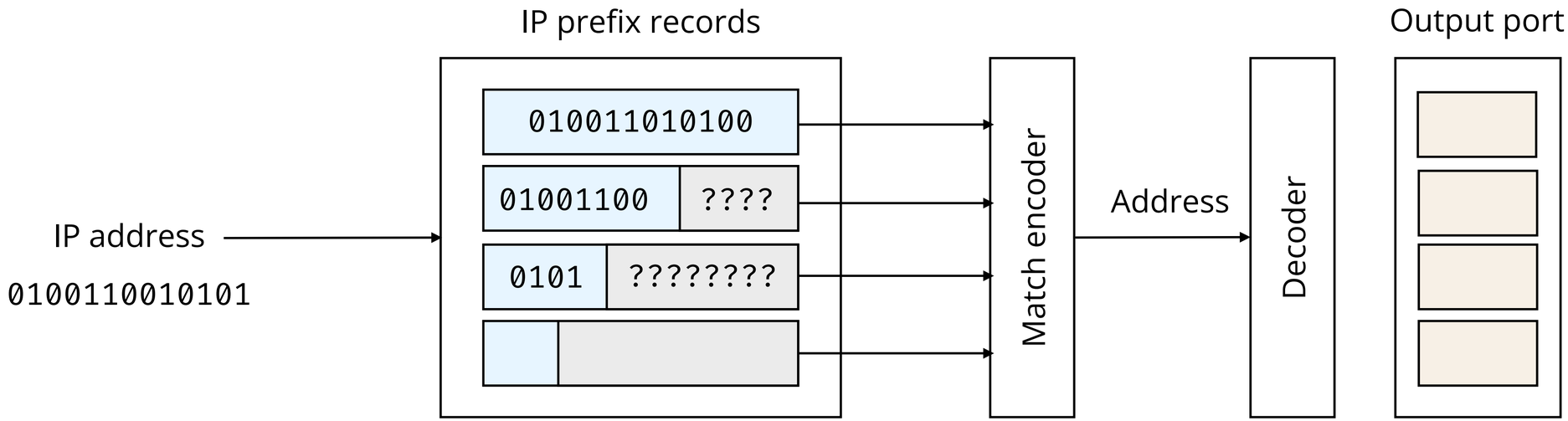


TCAM-based implementation

TCAM

A hardware device that supports to match on a set of records in constant time (one iteration)

- CAM supports only two states (0/1) in each bit position: widely used in switches for MAC address matching
- TCAM extends CAM by allowing for 3 states (0/1/?) in each position: useful for IP prefix matching
- Disadvantages: expensive, power-consuming



Why IP prefix, not IP addresses?

Routing

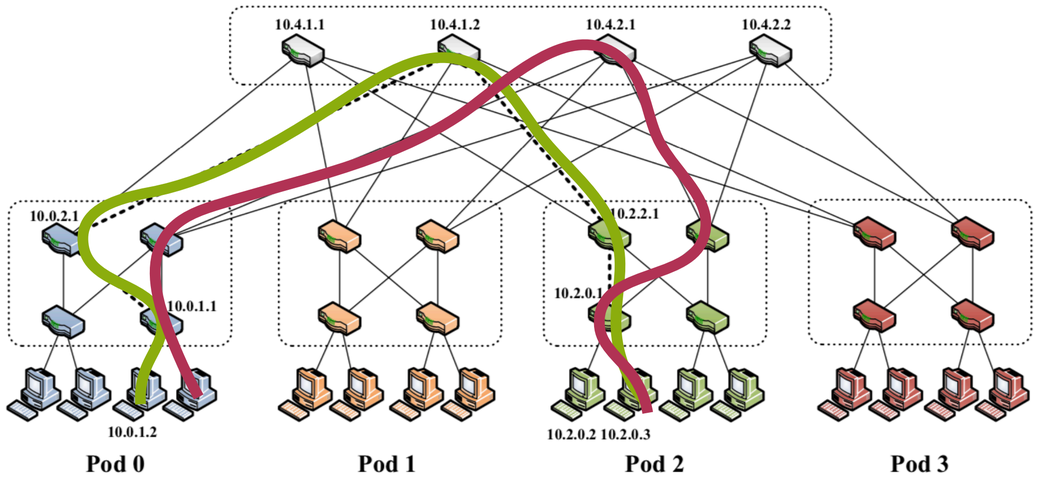
Prefixes in two-level lookup table prevent intra-pod traffic from leaving the pod

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

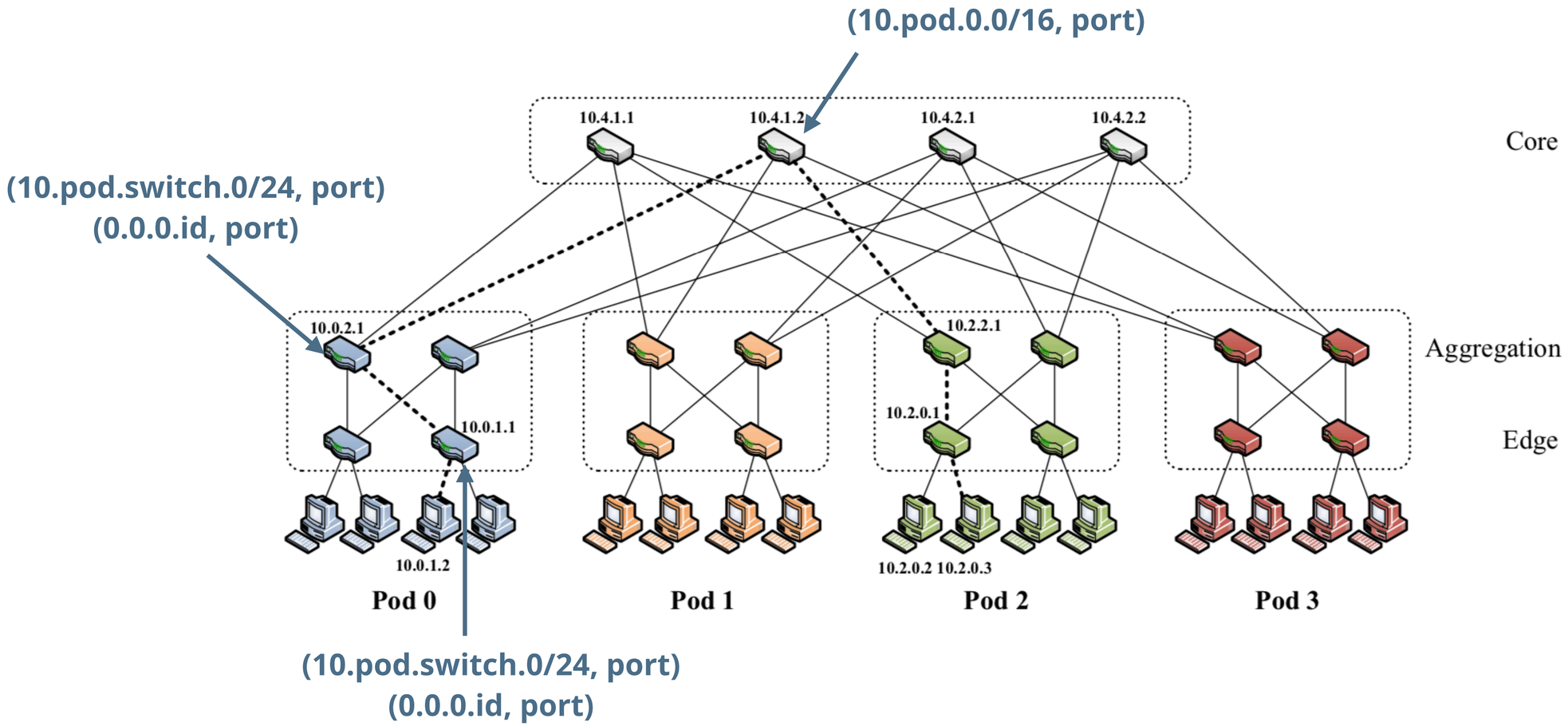
Each host-to-host communication has **a single static path**

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

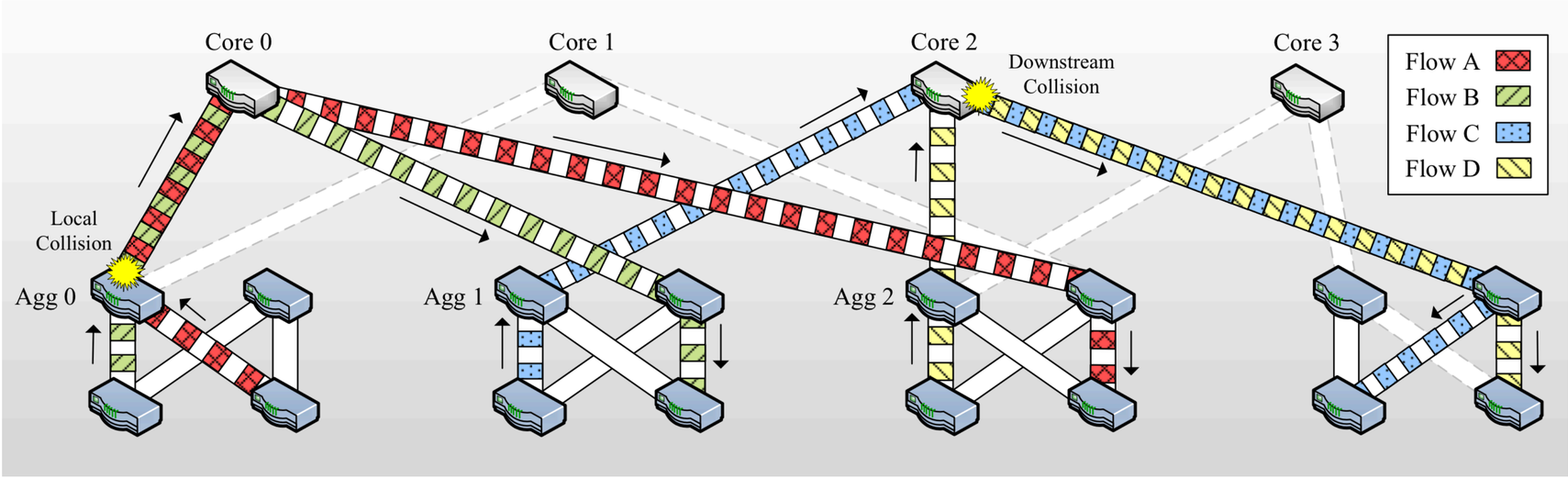


Routing example

10.0.1.2 → 10.2.0.3

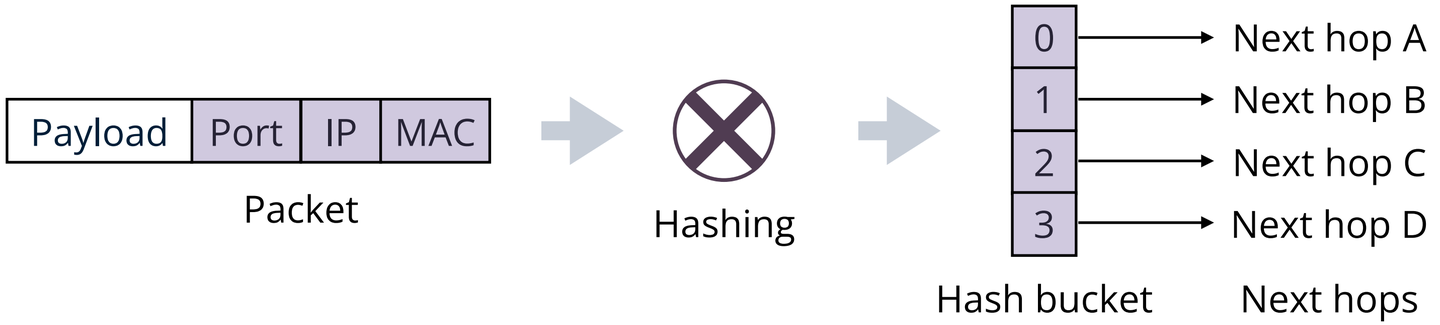


Flow collision



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

Solutions to flow collisions

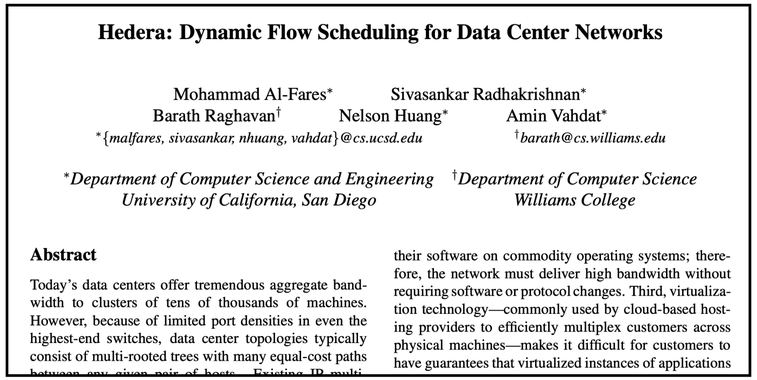


Equal-cost multi-path (ECMP)

- 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

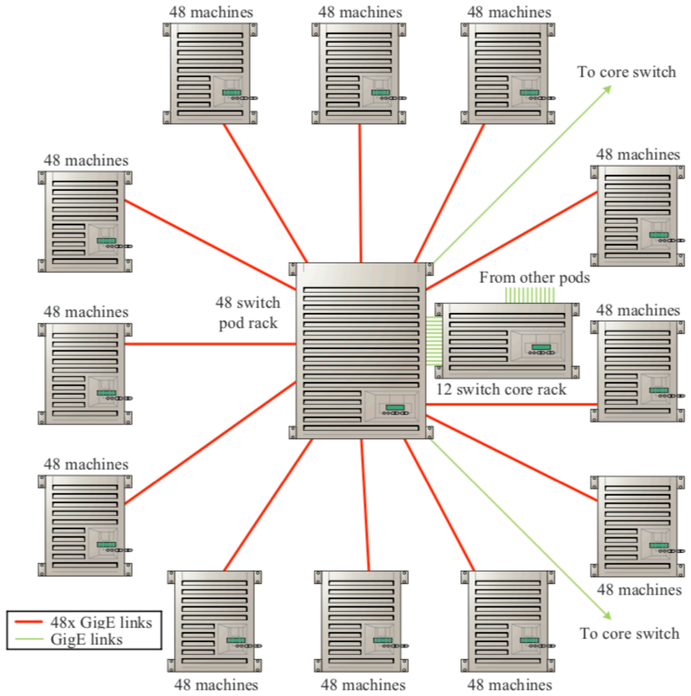
Abstract

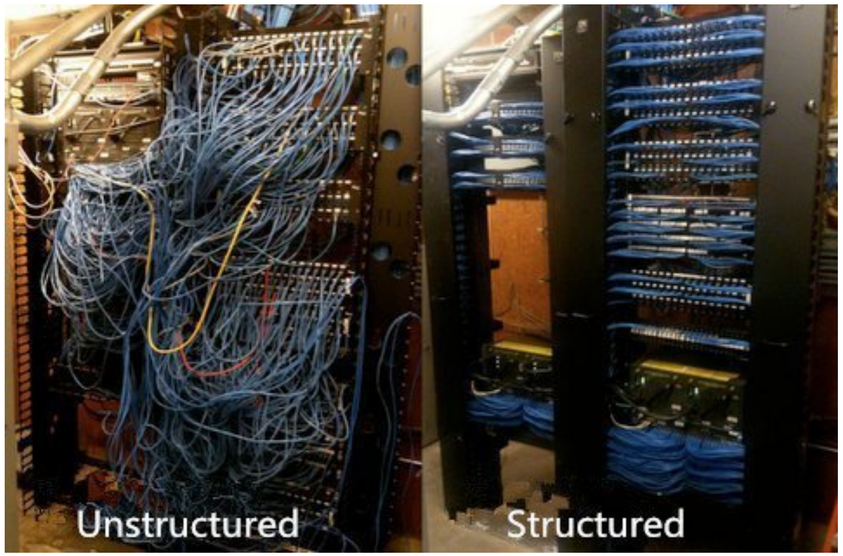
Today's data centers offer tremendous aggregate bandwidth to clusters of tens of thousands of machines. However, because of limited port densities in even the highest-end switches, data center topologies typically consist of multi-rooted trees with many equal-cost paths between any given pair of hosts.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

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

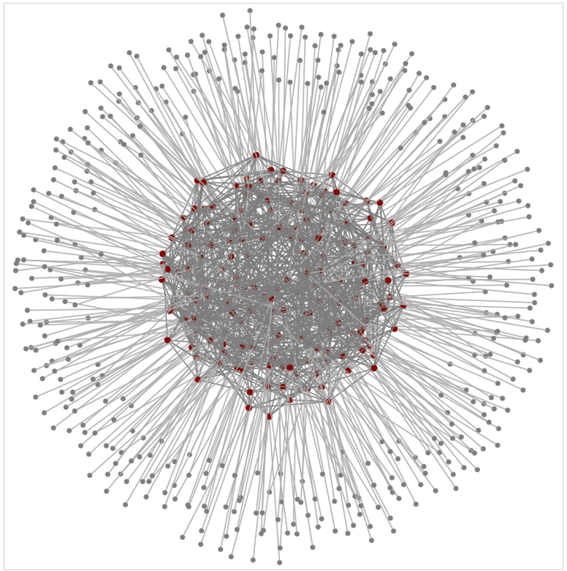




Organize switches into pod racks leveraging the regular structure of fat-tree

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

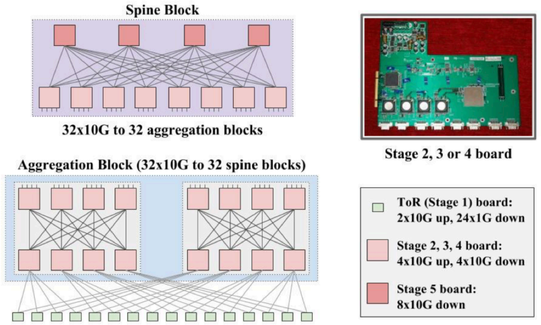
Industry experience indicates that the ability to incrementally expand data centers is essential. However, existing high-bandwidth network designs have rigid structure that interferes with incremental expansion. We present Jellyfish, a high-capacity network interconnect which, by adopting a random graph 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]).

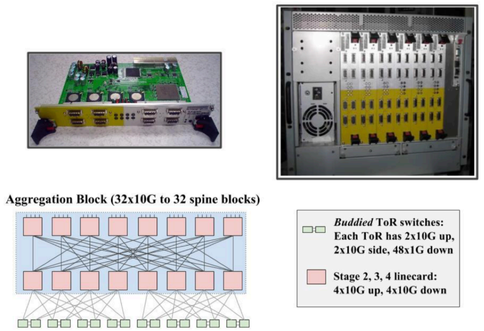
Do current high-bandwidth data center network proposals allow incremental growth? Consider the fat-tree

USENIX NSDI 2012

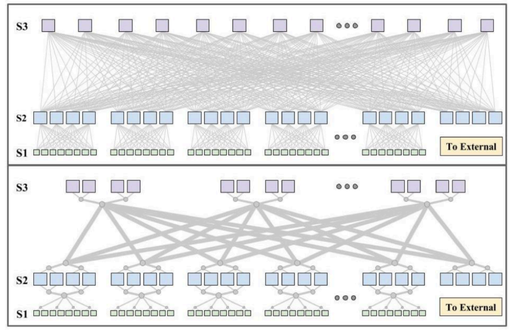
Google’s data center network evolution



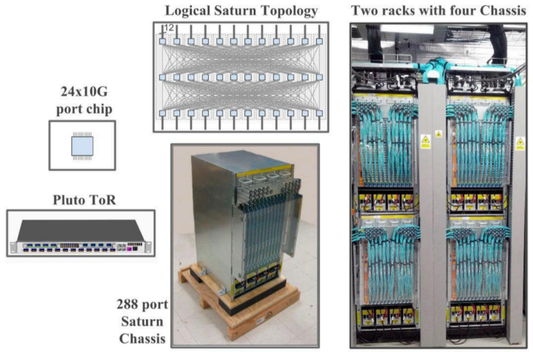
Firehose 1.0 (never in production)



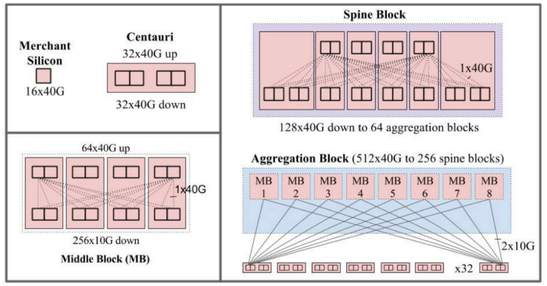
Firehose 1.1 (first production Clos, bag on the side)



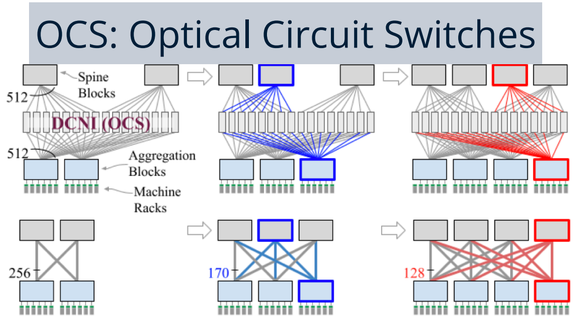
Watchtower (inter-cluster networking, depop)



Saturn (first 10G possible between servers)



Jupiter, 2015 (uniform bandwidth, incremental deployment)



Jupiter, 2022 (direct connect, reconfigurable)

Factors driven the evolution

Motivation

- **Bandwidth demands** in the data center are doubling 12-15 months
- **Cost and operational complexity** become prohibitive (expensive high-end switches, autonomous individual switch configuration)
- **Availability requirements** not strict in data centers (due to abundant, cheap bandwidth)
- **Interoperability** is not a big concern (single-operator)

Design principles

- **Clos topologies** (can scale to nearly arbitrary size, in-built path diversity and redundancy)
- **Merchant silicon** (general purpose, commodity priced, exponential growth in bandwidth capacity)
- **Centralized control protocols** (to replace distributed protocols)

Google’s data center network evolution

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 the general, but complex, decentralized network routing and management protocols supporting arbitrary deployment scenarios were overkill for single-operator, pre-planned datacenter networks. We built a centralized control mechanism based on a global configuration pushed to all datacenter switches. Third, modular hardware design coupled with simple, robust software allowed our design to also support inter-cluster and wide-area networks. Our datacenter networks run at dozens of sites across the planet, scaling in capacity by 100x over ten years to more than 1Pbps of bisection bandwidth.

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 can deliver higher quality results by accessing more data on the critical path of individual requests. Finally, constellations of co-resident applications often share substantial data with one another in the same cluster; consider index generation, web search, and serving ads.

Ten years ago, we found the cost and operational complexity associated with traditional datacenter network architectures to be prohibitive. Maximum network scale was limited by the cost and capacity of the highest end switches available at any point in time [24]. These switches were engineering marvels, typically recycled from products targeting wide area deployments. WAN switches were differentiated with hardware sup-

ACM SIGCOMM 2015

Jupiter Evolving: Transforming Google’s Datacenter Network via Optical Circuit Switches and Software-Defined Networking

Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, Rishi Kapoor, Stephen Kratzer, Nanfang Li, Hong Liu, Karthik Nagaraj, Jason Ornstein, Samir Sawhney, Ryohei Urata, Lorenzo Vicisano, Kevin Yasumura, Shidong Zhang, Junlan Zhou, Amin Vahdat
Google
sigcomm-jupiter-evolving@google.com

ABSTRACT

We present a decade of evolution and production experience with Jupiter datacenter network fabrics. In this period Jupiter has delivered 5x higher speed and capacity, 30% reduction in capex, 41% reduction in power, incremental deployment and technology refresh all while serving live production traffic. A key enabler for these improvements is *evolving Jupiter from a Clos to a direct-connect topology among the machine aggregation blocks*. Critical architectural changes for this include: A datacenter interconnection layer employing Micro-Electro-Mechanical Systems (MEMS) based Optical Circuit Switches (OCSes) to enable dynamic topology reconfiguration, centralized Software-Defined Networking (SDN) control for traffic engineering, and automated network operations for incremental capacity delivery and topology engineering. We show that the combination of traffic and topology engineering on direct-connect fabrics achieves similar throughput as Clos fabrics for our production traffic patterns. We also optimize for path lengths: 60% of the traffic takes direct path from source to destination aggregation blocks, while the remaining transits one additional block, achieving an average block-level path length of 1.4 in our fleet today. OCS also achieves 3x faster fabric reconfiguration compared to pre-evolution Clos fabrics that used a patch panel based interconnect.

KEYWORDS

Datacenter network, Software-defined networking, Traffic engineering, Topology engineering, Optical circuit switches.

ACM Reference Format:

Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, Rishi Kapoor, Stephen Kratzer, Nanfang Li, Hong Liu, Karthik Nagaraj, Jason Ornstein, Samir Sawhney, Ryohei Urata, Lorenzo Vicisano, Kevin Yasumura, Shidong Zhang, Junlan Zhou, Amin Vahdat Google sigcomm-jupiter-evolving@google.com . 2022. Jupiter Evolving: Transforming Google’s Datacenter Network via Optical Circuit Switches and Software-Defined Networking. In *Proceedings of ACM Conference (SIGCOMM’22).* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3544216.3544265

1 INTRODUCTION

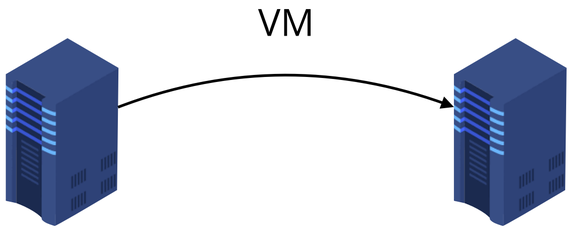
Software-Defined Networking and Clos topologies [1, 2, 14, 24, 33] built with merchant silicon have enabled cost effective, reliable building-scale datacenter networks as the basis for Cloud infrastructure. A range of networked services, machine learning workloads, and storage infrastructure leverage uniform, high bandwidth connectivity among tens of thousands of servers to great effect.

While there is tremendous progress, managing the heterogeneity and incremental evolution of a building-scale

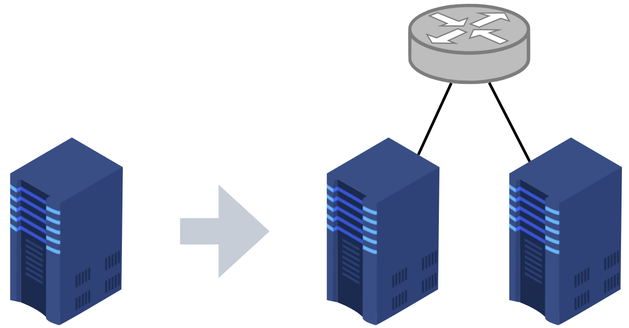
ACM SIGCOMM 2022

How to achieve flexible management of data center networks?

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

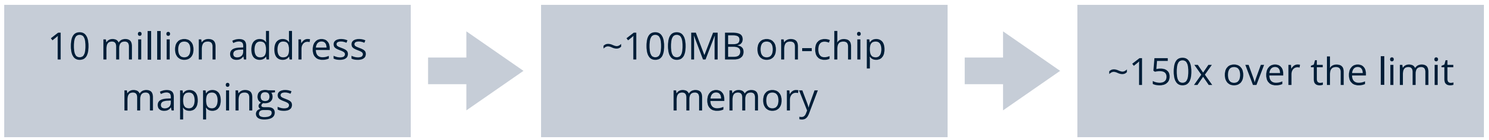
Switch state: L2 vs L3

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

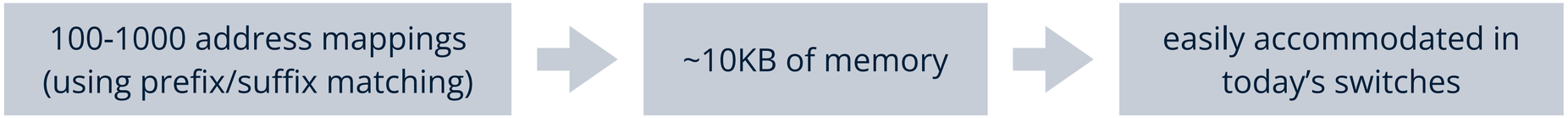
- Can store 32-64K forwarding entries

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

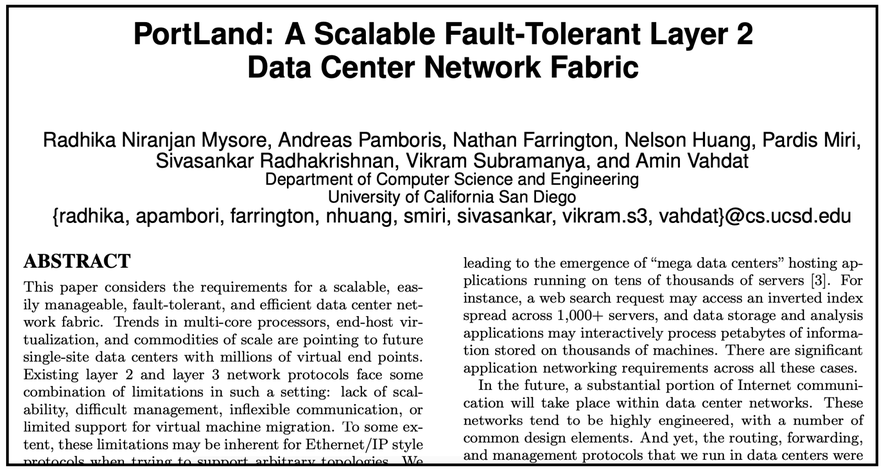
Main idea: separate node location from node identifier

- 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

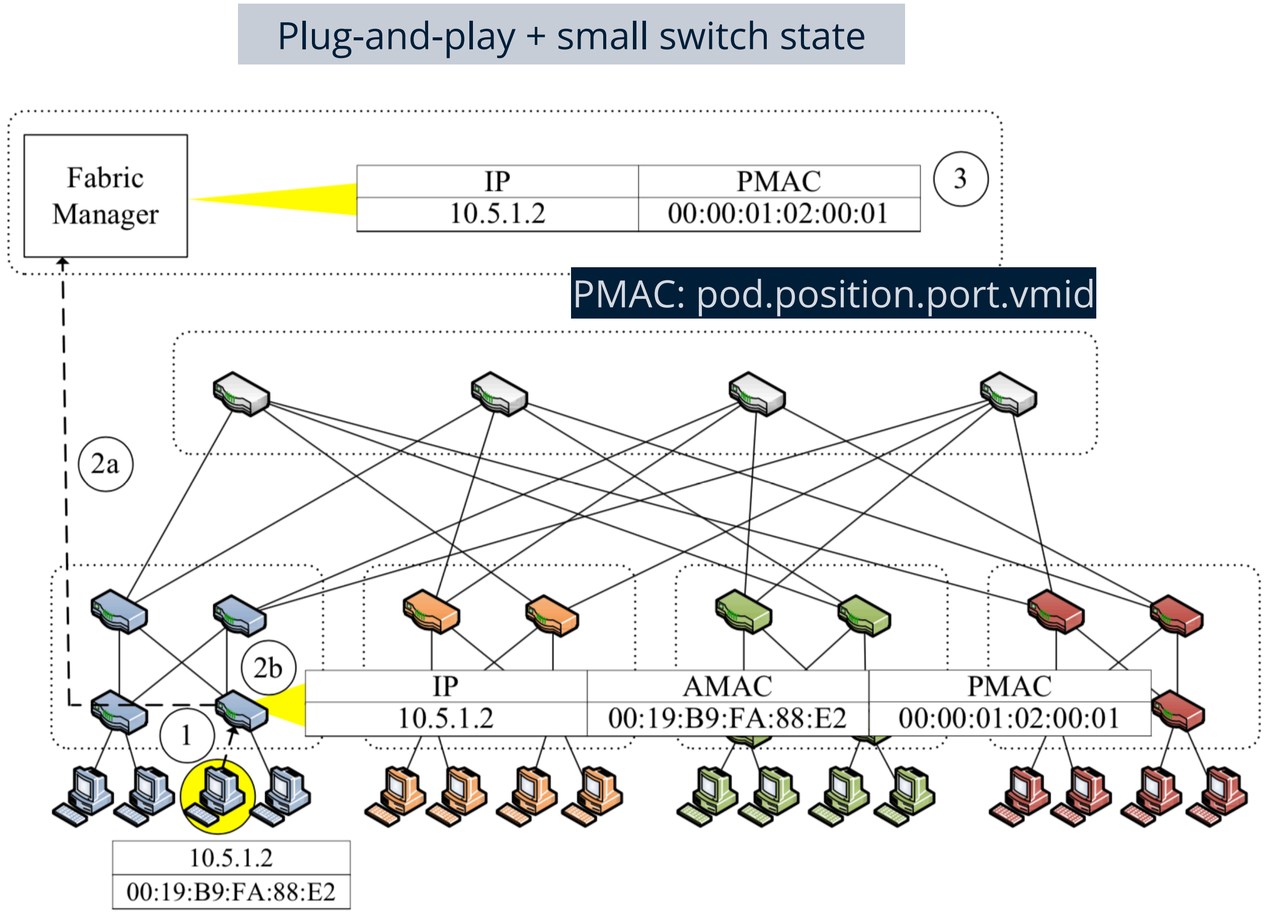
This paper considers the requirements for a scalable, easily manageable, fault-tolerant, and efficient data center network fabric. Trends in multi-core processors, end-host virtualization, and commodities of scale are pointing to future single-site data centers with millions of virtual end points. Existing 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



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

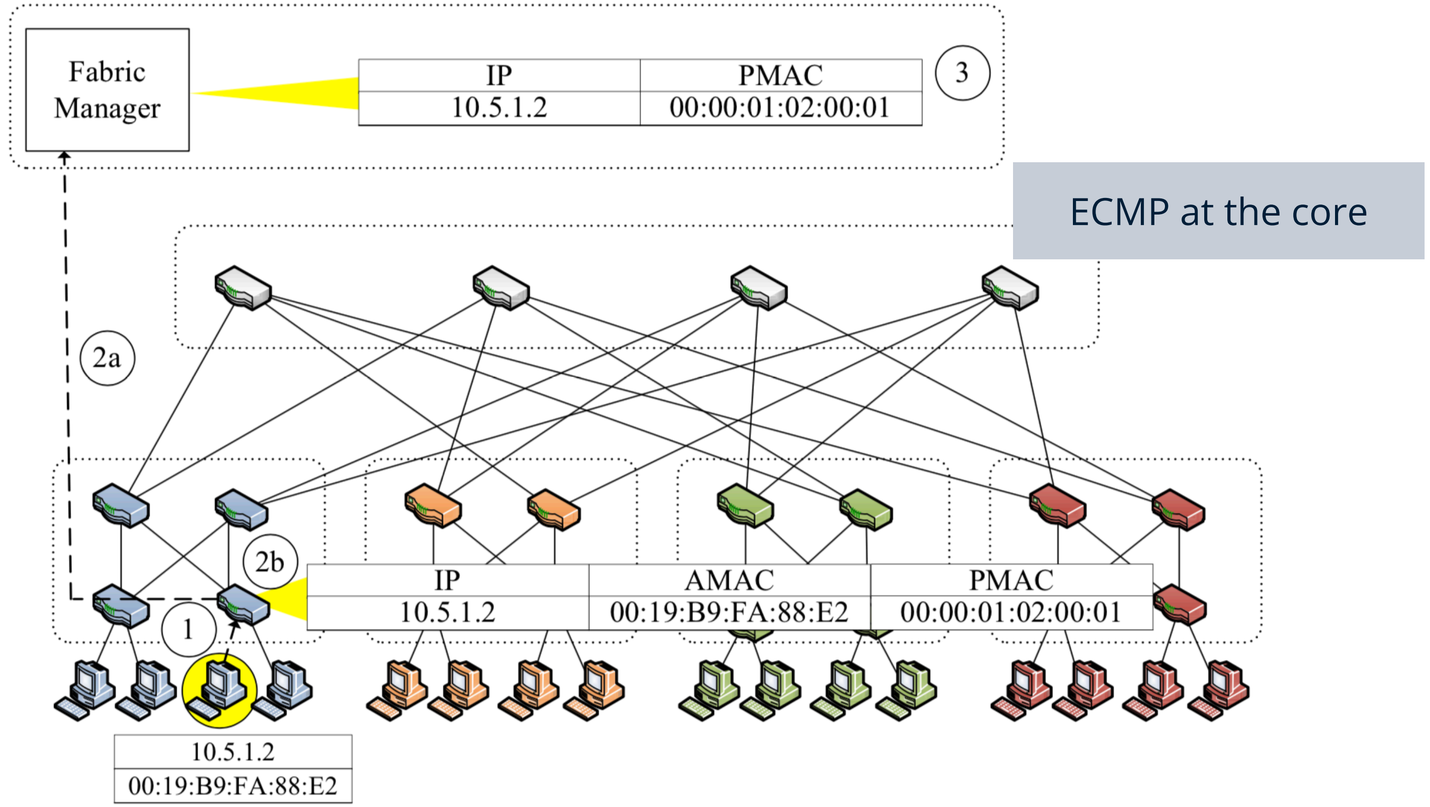
- Tree-level/role: based on neighbor identity
- Pod number: fetch from the Fabric manager
- Position number: aggregation switches help ToR switches choose unique position number

Fabric manager

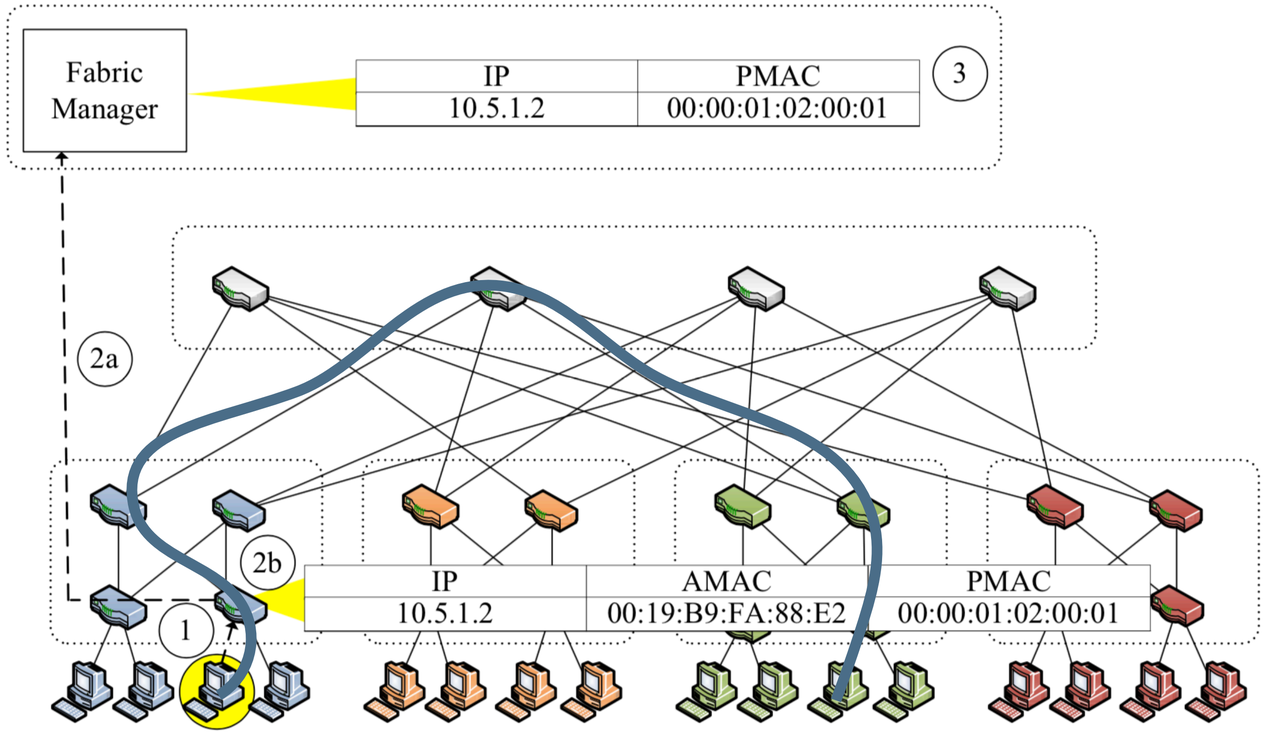
ARP mappings

Network map

Soft state: no need for manual configurations

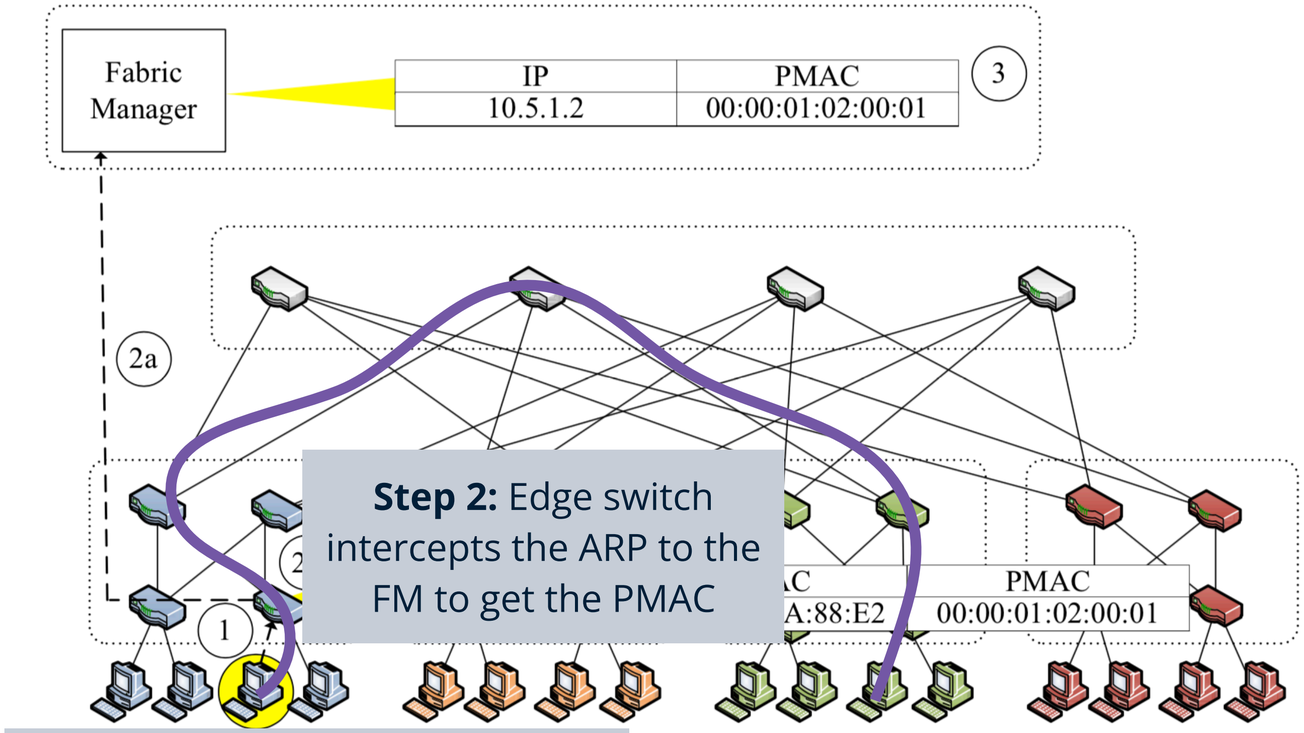


PortLand workflow

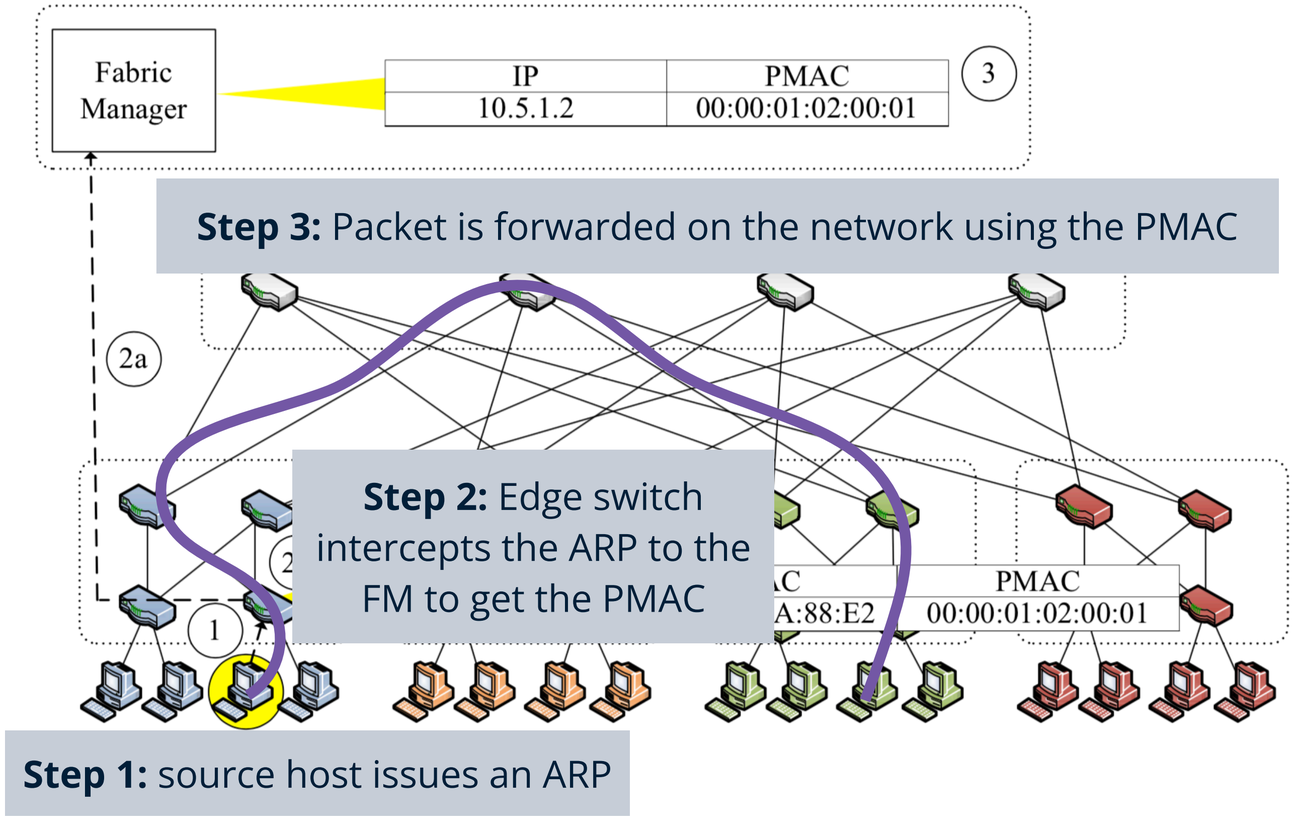


Step 1: source host issues an ARP

PortLand workflow



PortLand workflow

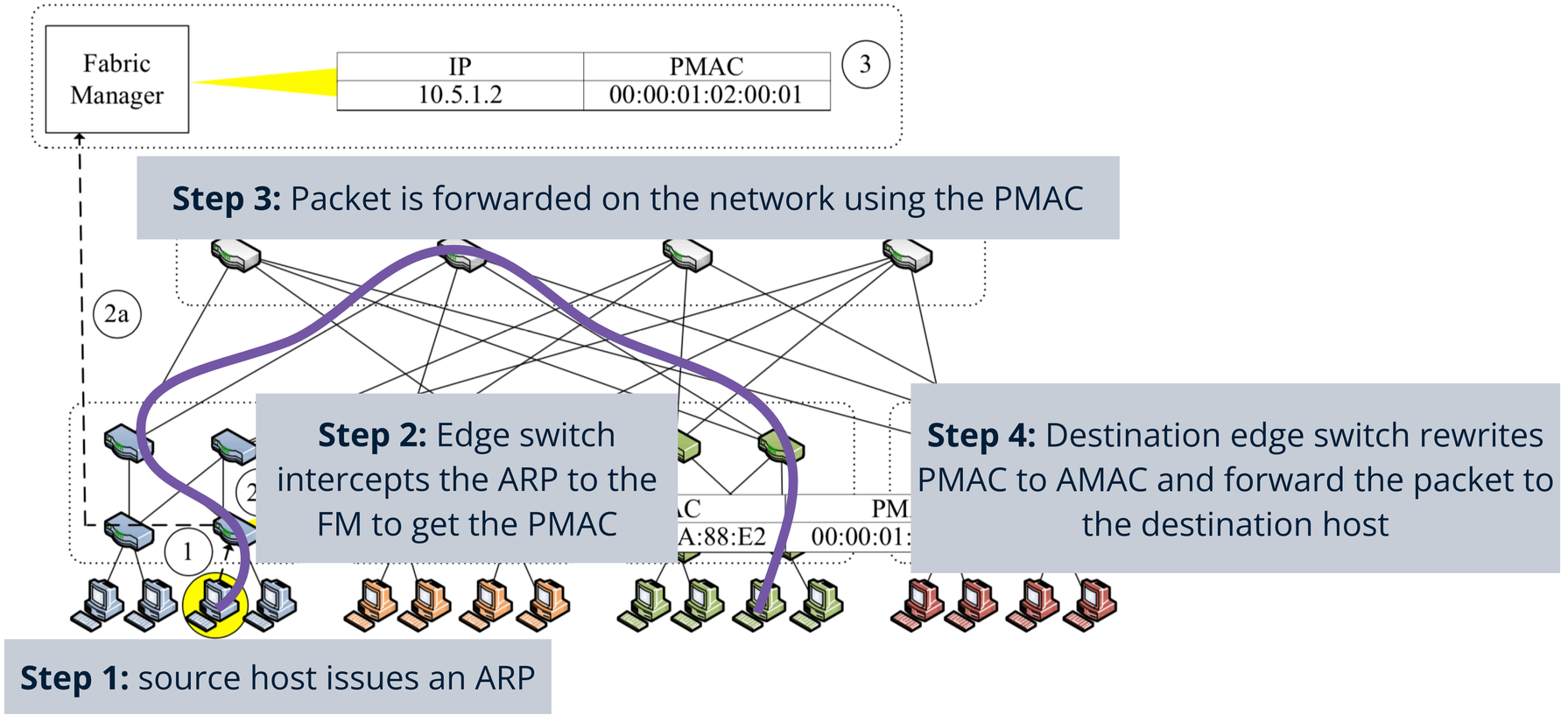


Step 3: Packet is forwarded on the network using the PMAC

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

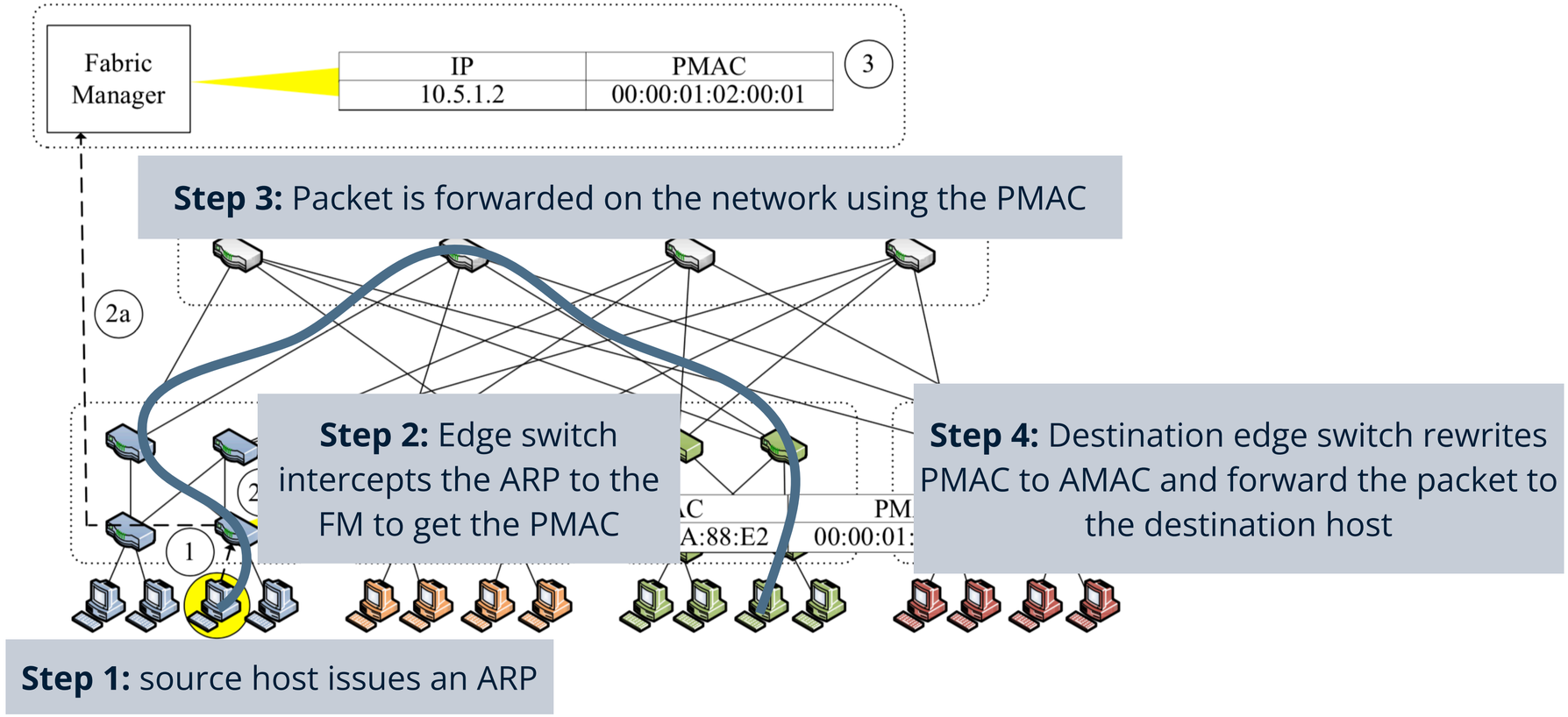
Step 1: source host issues an ARP

PortLand workflow



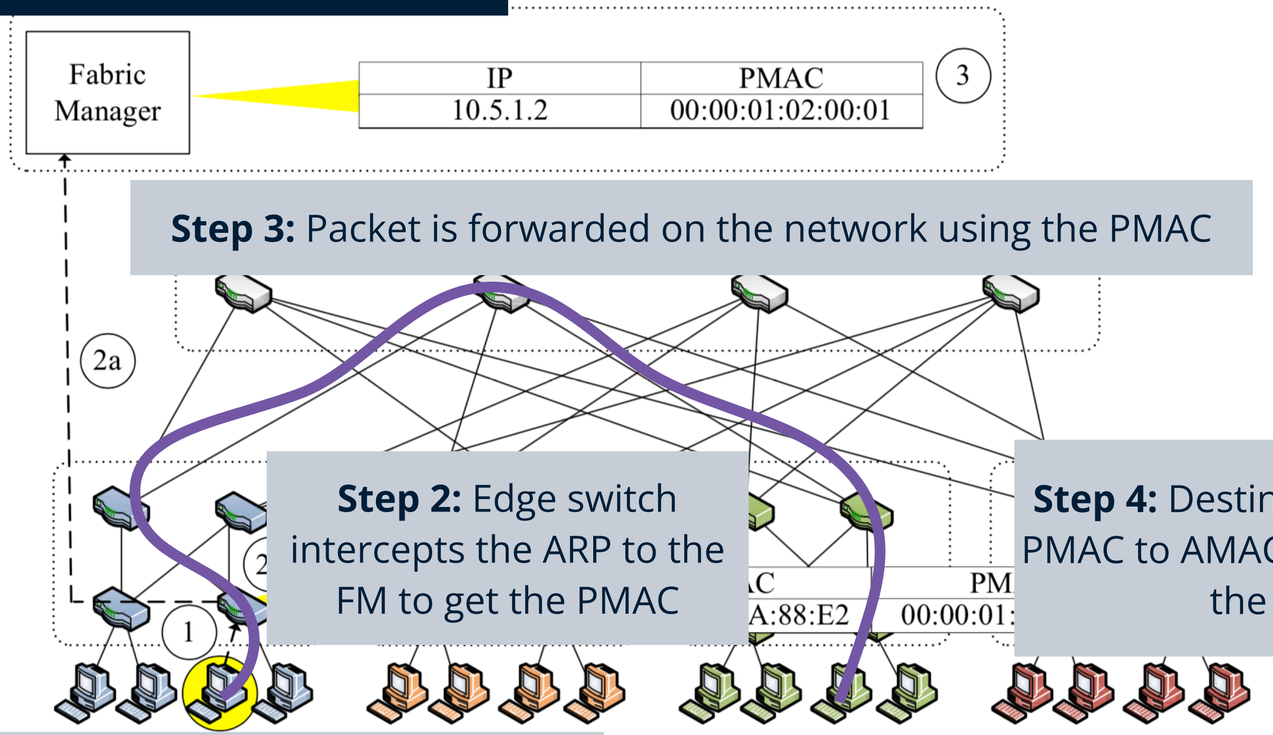
PortLand workflow

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



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

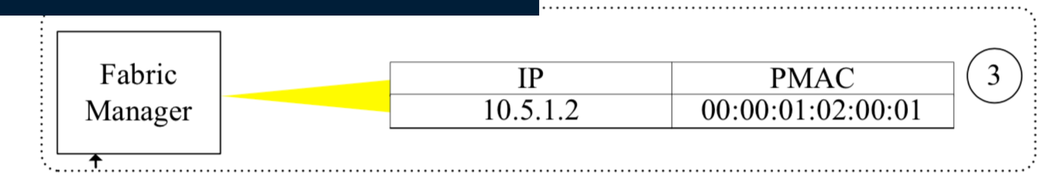
Step 4: Destination edge switch rewrites PMAC to AMAC and forward the packet to the destination host

Step 1: source host issues an ARP

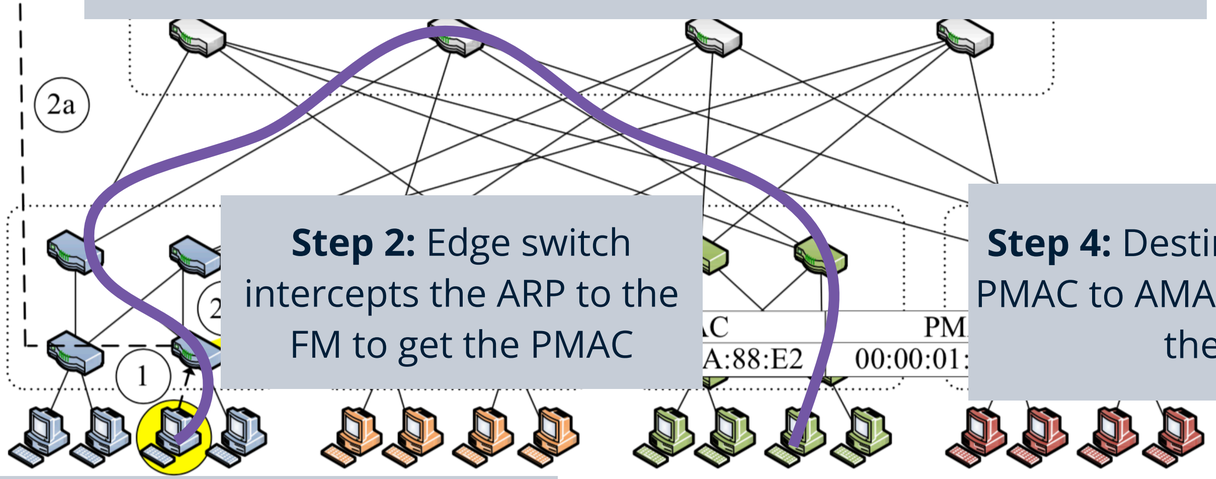
PortLand workflow

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



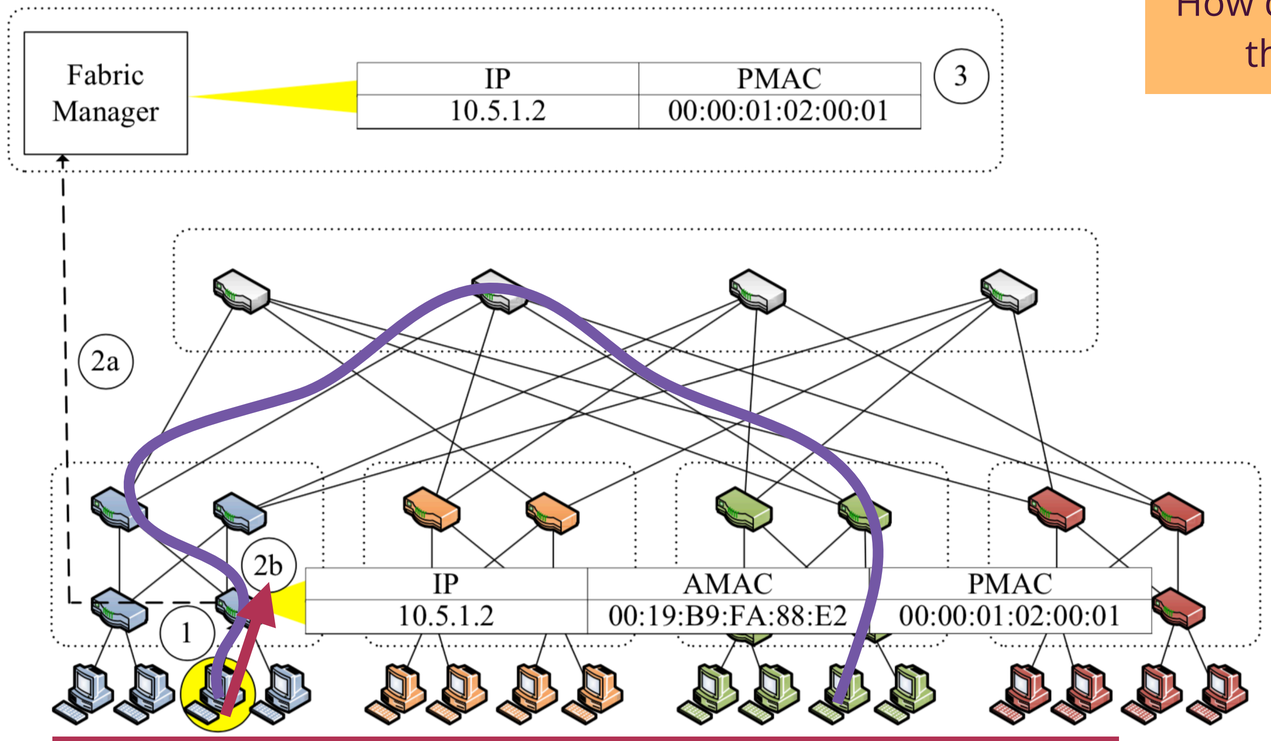
Step 2: Edge switch intercepts the ARP to the FM to get the PMAC

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?

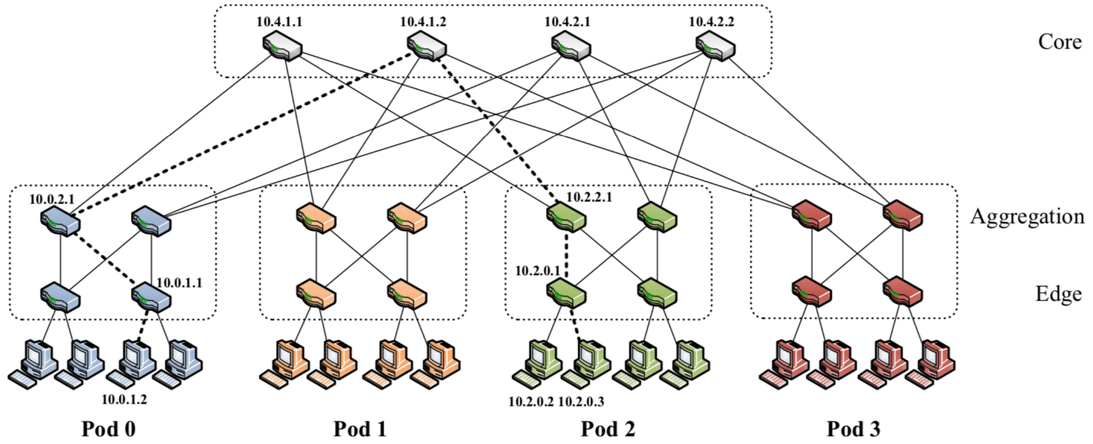


Recall learning switch: an IP-PMAC entry is forwarded to the FM every time the edge switch sees a new IP

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