eecs489 COMPUTER NETWORKS

Lecture 24:
Cloud Computing and
Data Center Networking

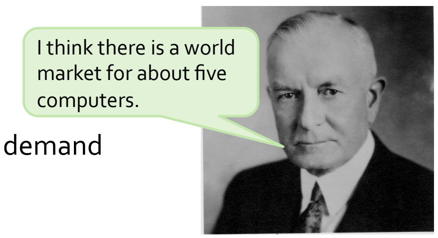
Utility Computing

August 2006: Amazon Elastic Compute Cloud, EC2+S3
- first successful IaaS offering

IaaS == Infrastructure as a Service
- swipe your credit card, and spin up your VM

Provides utility computing:
- computing resources as a metered service ("pay as you go")
- ability to dynamically provision virtual machines

Why utility computing?
- cost: CAPEX vs. OPEX
- scalability: "infinite" capacity
- elasticity: scale "out" (or in) on demand



[Joshi&Lagar-Cavilla, Lin]

Evolution into PaaS

Platform as a Service (PaaS) is higher level
- simpleDB (relational tables)
- simple queue service
- elastic load balancing
- flexible payment service

Azure Joyent

PaaS diversity (and lock-in)
- Amazon's Elastic Beanstalk (upload your JAR)
- Microsoft Azure: .NET, SQL
- Google AppEngine: python, java, GQL, memcache
- Heroku: ruby, python, node.js, php, java
- Joyent: node.js and javascript

heroku

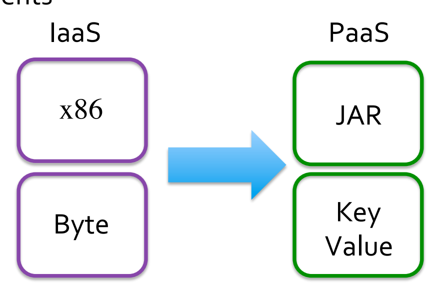
[Joshi&Lagar-Cavilla]

IaaS vs. PaaS

Hardware-centric vs. API-centric

Never care about drivers again
- or sys-admins, or power bills

You can scale if you have the money
- you can deploy on two continents
- and ten thousand servers
- and 20TB of storage



[Joshi&Lagar-Cavilla]

Your New Concerns

App provider:

- how will I horizontally scale my application
- how will my application deal with distribution
  - latency, partitioning, concurrency
- how will I guarantee availability
  - failures will happen
  - dependencies are unknown

Cloud provider:

- how will I maximize multiplexing?
- can I scale and provide performance guarantees?
- how can I diagnose infrastructure problems?

[Joshi&Lagar-Cavilla]

From Cloud-User's POV

Cloud is like the IP layer

- it provides a best-effort substrate
- is cost-effective
- is on-demand
- provides compute and storage infrastructure

But you have to build your own reliable service

- fault tolerance
- availability, durability, QoS

[Joshi&Lagar-Cavilla]

Everything as a Service

Utility computing = Infrastructure as a Service (IaaS)

- why buy machines when you can rent cycles?
- examples: Amazon's EC2, Rackspace

Platform as a Service (PaaS)

- give me a nice API and take care of the maintenance, upgrades, …
- example: Google AppEngine, Heroku

Software as a Service (SaaS)

- Just run it for me!
- example: Gmail, GoogleDocs, Salesforce, Adobe's Creative Cloud, Microsoft's Office365

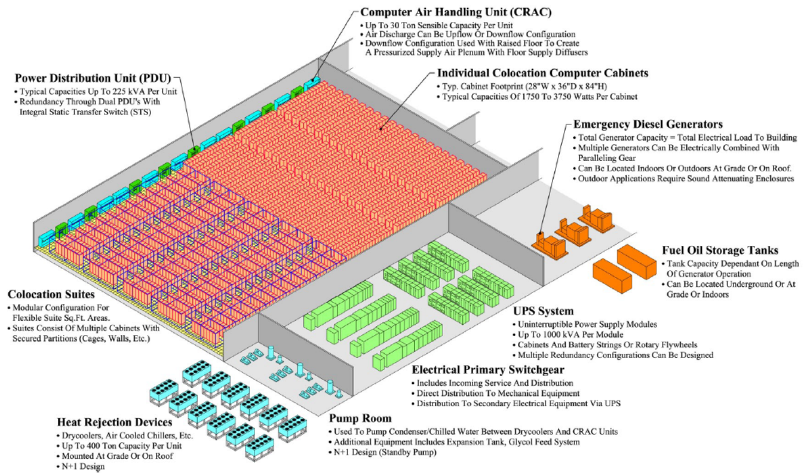
[Lin]

Cloud Computing: Summary

NIST's definition: services accessed over a standardized network with the following characteristics:

- on-demand self-service: a customer can order compute resources without any human interaction with provider
- resource pooling: provider's physical and virtual resources pooled to serve multiple customers dynamically
- rapid elasticity: resources appear unlimited and can be scaled up or down rapidly
- measured service: metered usage (and billing)
- broad network access: available over the Internet, platform independent: mobile, laptops, tablets

Anatomy of a Datacenter



Source: Barroso and Urs Hölzle (2009)

How Much Power Needed?

- 0.0003 kWh to answer a typical Google search
- 0.05 kW to use a laptop for an hour
- 0.1 kW to run a ceiling fan for an hour
- 1.1 kW to use a coffee maker for an hour
- How much power is 30 MW?
  - 6,000 average homes with central air (~5 kW/home)
  - 300 fast food restaurants
  - 45 large retail stores
  - 37 grocery stores
  - 30 large home improvement stores
  - 1.5 Sears Towers
  - 1 computer data center

Source: IEEE Spectrum and Google

Data Center Networks

Tens to hundreds of thousands of hosts, often closely coupled, in close proximity:

- e-commerce (e.g., Amazon)
- content servers (e.g., NetFlix, YouTube, Apple, Microsoft)
- search engines, data mining (e.g., Google)

Challenges:

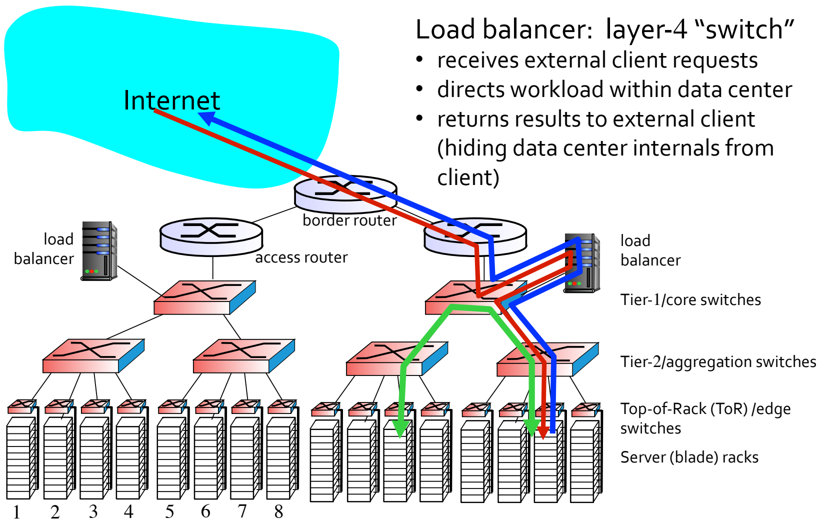
- multiple applications, each serving massive numbers of clients
- managing/balancing load, avoiding processing, networking, data bottlenecks

Inside a 40-ft Microsoft container, Chicago data center

Data Center Networks

Load balancer: layer-4 "switch"

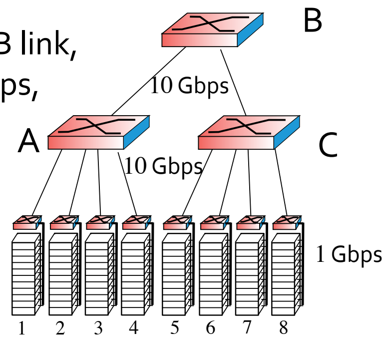
- receives external client requests
- directs workload within data center
- returns results to external client (hiding data center internals from client)



Potential Network Bottleneck

Host – ToR: 1 Gbps
ToR – Tier 2 and Tier 1 – Tier 2: each 10 Gbps
10 hosts on rack 1 each talk to a different host on rack 5
Similarly between racks 2 – 6, 3 – 7, and 4 – 8

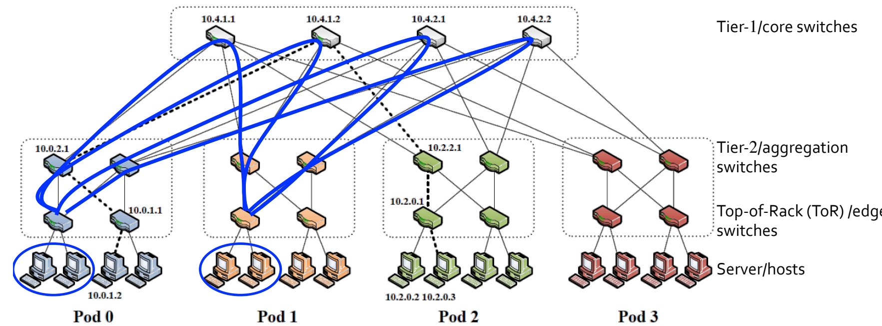
40 flows share the 10 Gbps A – B link, each gets only $10/40 = 250$ Mbps, only ¼ of the 1 Gbps host – ToR capacity



Fat-tree Topology with $k = 4$

Rich interconnection among switches, a.k.a. Clos network

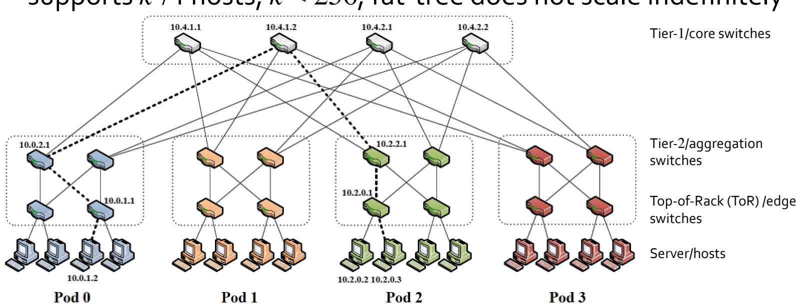
- increased throughput between racks
  Equal Cost Multi-Path (ECMP) routing
- increased reliability via redundancy
- originally intended for data center with off-the-shelf parts



Fat-tree Architecture

k-ary fat-tree: three-layer topology

- k pods, each consists of $(k/2)^2$ hosts and two layers of switches, each layer has $k/2$ k-port switches
- each ToR switch connects to $k/2$ hosts and $k/2$ Tier-2 switches
- each Tier-2 switch connects to $k/2$ ToR and $k/2$ Tier-1 switches
- $(k/2)^2$ Tier-1 switches: each connects to all k pods
- supports $k^3/4$ hosts, $k < 256$, fat-tree does not scale indefinitely



[Beyer]

Cost Analysis

	Hierarchical design			Fat-tree		
Year	**10 GigE**	**Hosts**	**Cost/ GigE**	**GigE**	**Hosts**	**Cost/ GigE**
2002	28-port	4,480	\$25.3K	28-port	5,488	\$4.5K
2004	32-port	7,680	\$4.4K	48-port	27,648	\$1.6K
2006	64-port	10,240	\$2.1K	48-port	27,648	\$1.2K
2008	128-port	20,480	\$1.8K	48-port	27,648	\$0.3K

Maximum possible cluster size with all hosts capable of fully utilizing uplink capacity

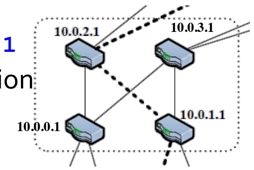
Hierarchical design uses higher-speed, and more expensive, switches higher up in the hierarchy (scale up)

Addressing in Fat-tree

Use `10.0.0.0/8` private addressing block

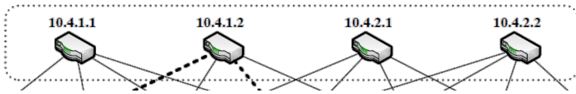
Pod switches have address `10.pod.switch.1`

- pod and switch in range $[0, k\text{-}1]$, based on position



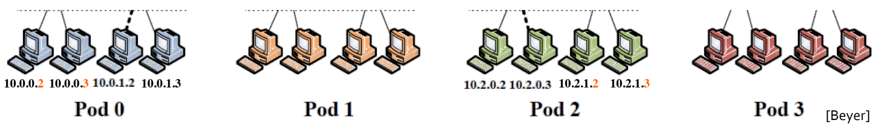
Tier-1 switches have address `10.k.i.j`

- i and j denote switch position in $(k/2)^2$ Tier-1 switches



Hosts have address `10.pod.switch.ID`

- ID in range $[2, (k/2)+1]$, for $k = 4$, ID can only be 2 or 3

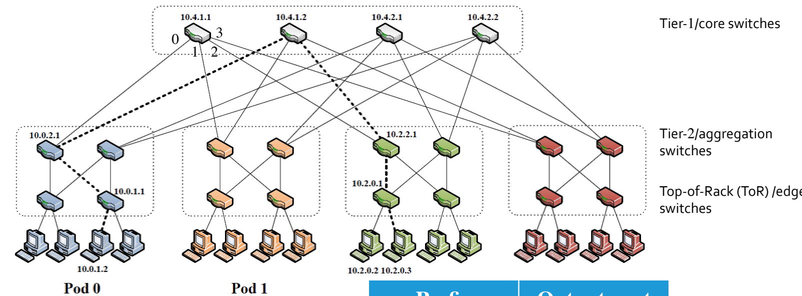


[Beyer]

Forwarding in Fat-tree

Tier-1 switches contain (`10.pod.0.0/16, port`) entries

- statically forwards inter-pod traffic on specified port



- `10.4.1.1`'s routing table:

Prefix	Output port
10.0.0.0/16	0
10.1.0.0/16	1
10.2.0.0/16	2
10.3.0.0/16	3

[Beyer]

Tier-2's Two-Level Lookup Table

Prefix table contains (`10.pod.switch.0/24, port`) entries

- switch value is the ToR switch number
- used for forwarding intra-pod traffic

Suffix table used for forwarding inter-pod traffic

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

Recall: for $k = 4$, host ID can only be 2 or 3

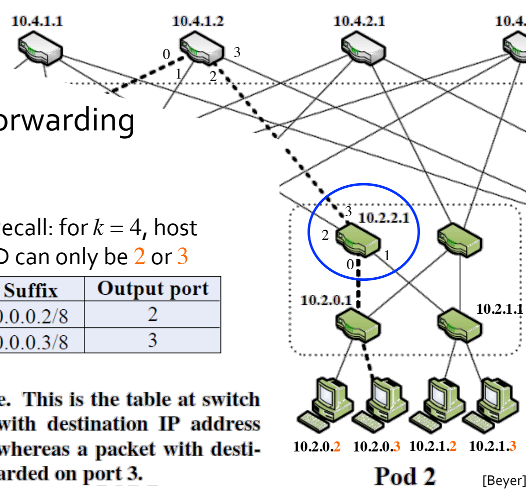


Figure 4: Two-level table example. This is the table at switch 10.2.2.1. An incoming packet with destination IP address 10.2.1.2 is forwarded on port 1, whereas a packet with destination IP address 10.3.0.3 is forwarded on port 3.

[Beyer]

Tier-2's Forwarding Algorithm

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

Prefix table prevents intra-pod traffic from leaving pod

Suffix table for inter-pod traffic based off host IDs:

- ensures spread of traffic across Tier-1 switches
- prevents packet reordering by assigning a single static path for each host-to-host communication
  - better than having a single path between subnets

[Beyer]

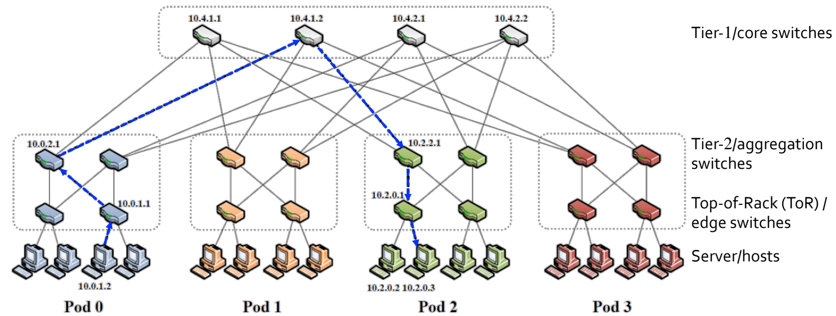
ToR Switch's Forwarding

Inter-rack traffic relies on switch's original backward learning algorithm

Assumes forwarding tables generated by a central controller with full knowledge of topology

- central controller also responsible for detecting switch failures and re-routing traffic
- and for answering ARP and DHCP requests

Fat-tree Routing Example



Packets from source `10.0.1.2` to destination `10.2.0.3` take the dashed path

[Beyer]

Two-Level Lookup Implementation

Implemented in hardware using a TCAM

- TCAM: Ternary (0, 1, don't care) Content-Addressable Memory
- can perform parallel lookups across table
- stores don't care bits, suitable for variable length prefixes

Prefixes preferred over suffixes

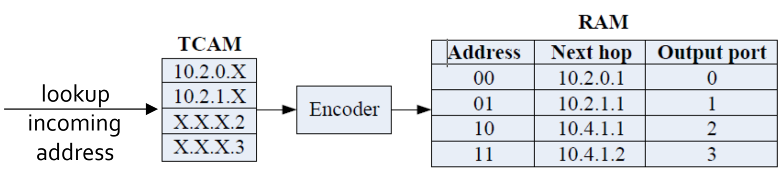
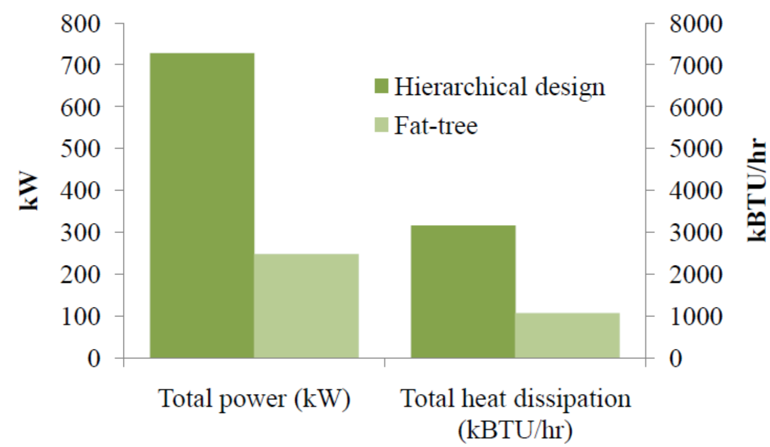


Figure 5: TCAM two-level routing table implementation of switch 10.2.2.1 in the example network

[Beyer]

Topology Power/Heat Dissipation

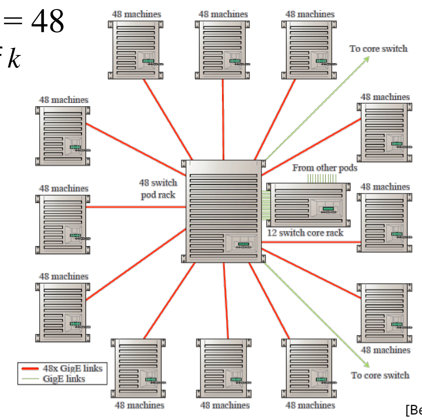


Packaging Problem

Fat-tree has significant cabling overhead
- 1 GigE switches used to reduce cost
- lack of 10 GigE ports leads to more cabling

A packaging solution for $k = 48$
- generalizes to other values of k

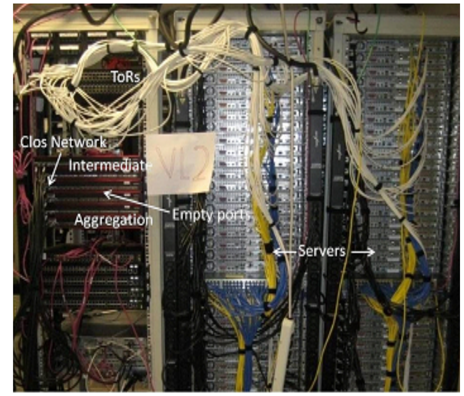
Cabling in general can be a problem in data center networks . . . .



[Beyer]

Other DC Network Topologies

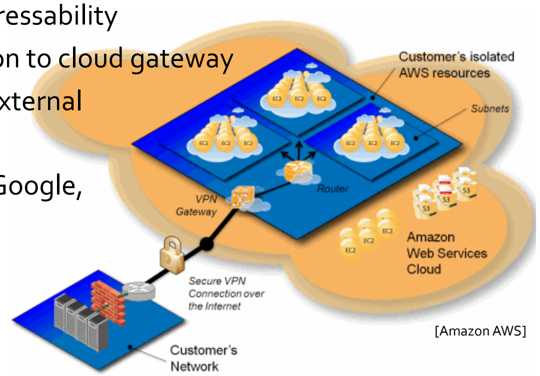
VL2:
- also based on Clos network
- but has a more flexible addressing scheme
- runs link-state routing
- does network load balancing



Other topologies have the hosts themselves also serve as routers

Network Security Evolved

Virtual private clouds
- internal VLANs within cloud
- virtual network functions (VNFs): virtual gateways, virtual firewalls: middleboxes implemented in software
- remove external addressability
- MPLS VPN connection to cloud gateway
- but doesn't protect external facing assets
- providers: Amazon, Google, Microsoft, etc.



[Amazon AWS]

Information Leakage

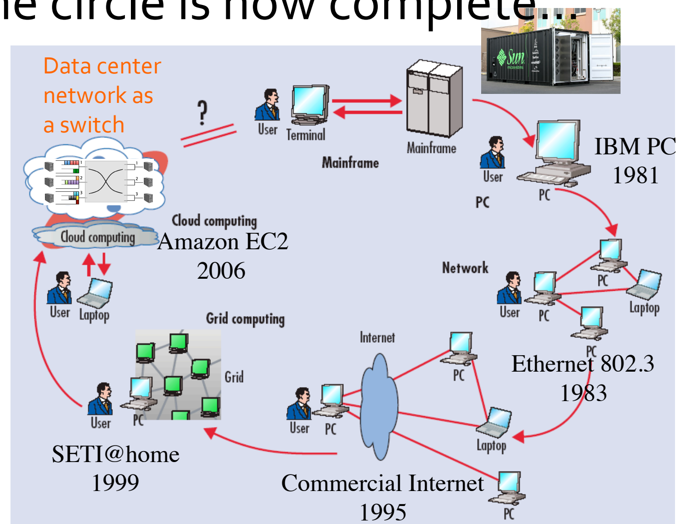
Is your target in a cloud?
- traceroute
- network triangulation

Every VM gets its private/public IP
Are you on the same machine as target?
- IP addresses
- latency checks
- side channels (cache interference)

Can you get on the same machine?
- pigeon-hole principle (n items, m containers, $n > m \Rightarrow$ some containers must be shared)
- placement locality

[Joshi&Lagar-Cavilla]

The circle is now complete…



Source: Voas and Zhang, "Cloud Computing: New Wine or Just a New Bottle?"
IT Professional, 11(2):15–17, March 2009

[Joshi&Lagar-Cavilla]