A Scalable, Commodity Data Center Network Architecture

Mohammad Al-Fares, Alexander Loukissas, Amin Vahdat

SIGCOMM 2008

Outline

- Background
- Fat tree based solution
- Implementation and evaluation
- Review

Data centers

- The various online services we use daily, such as Google Search, YouTube videos, Taobao shopping, WeChat chats, and so on, all rely on vast numbers of computers to process information and store data.
- All these computers, along with the networking devices connecting them, as well as supporting infrastructure like power supply and cooling systems, are concentrated in massive facilities known as **data centers**.
- Three core functions of a DC: storing data, running applications, and processing requests.

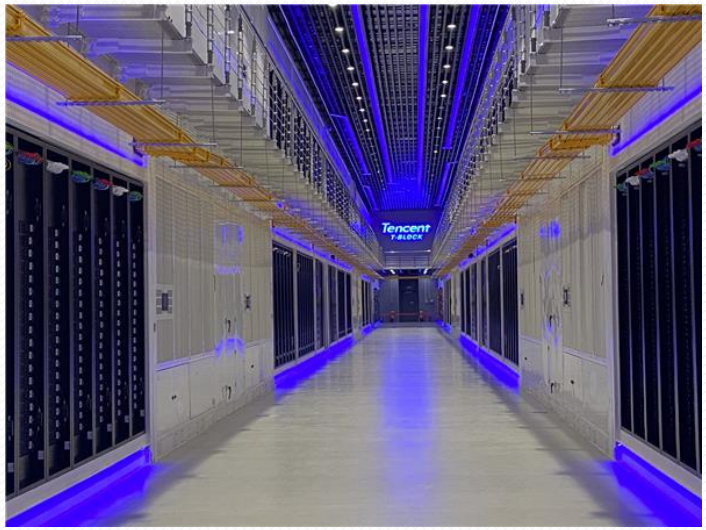
Data centers

- Clusters of thousands of computers
- Where the Internet lives

Google datacenter

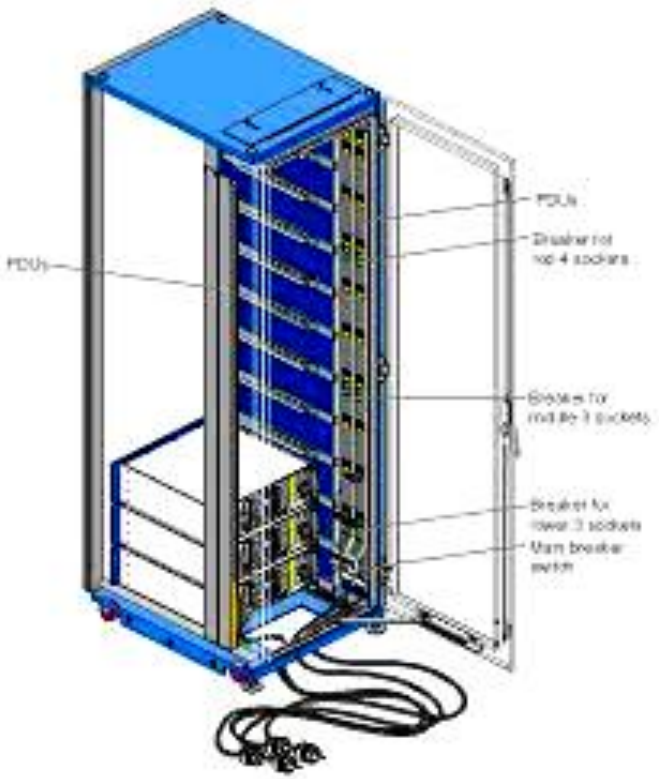
Microsoft underwater datacenter

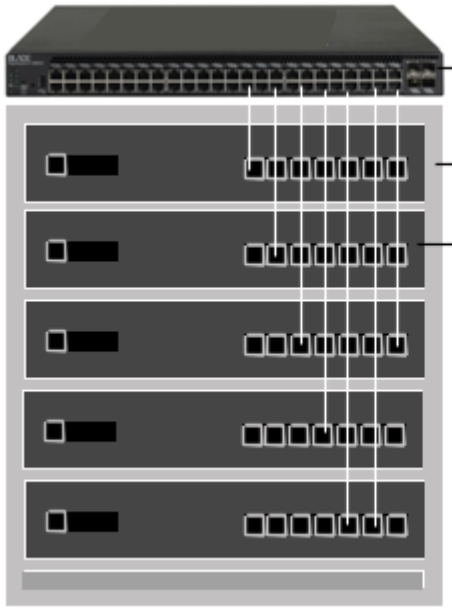
阿里巴巴千岛湖数据中心



腾讯贵安七星数据中心

Data center racks





Fundamental Concepts

- **Switch:** In the context of data centers, the term "switch" typically refers to a device that integrates both Layer 2 switching and Layer 3 routing functionalities. This is because it must handle both intra-LAN server communication and cross-subnet server communication.
- **Aggregate Bandwidth:** The sum of the bandwidth capacities of all connections within a network.

Fundamental Concepts

- **Bisection Bandwidth:** Imagine splitting the entire network into two equal parts. The sum of the bandwidth of all links connecting these two halves is the bisection bandwidth, namely the cut in graph theory. This metric evaluates the overall communication capacity of a network.
- **Commodity Hardware:** Refers to widely available, relatively low-cost, and functionally standardized general-purpose hardware devices, as opposed to specialized, expensive high-end custom equipment.

Fundamental Concepts

- Three-tier Hierarchical Model
  - Edge (Access Layer): The outermost layer of the network where end devices (e.g., servers, computers) directly connect.
  - Aggregation (Distribution Layer): The intermediary layer that connects the access and core layers, aggregates traffic, and enforces network policies (e.g., routing, security policies).
  - Core (Core Layer): The backbone of the network, whose sole purpose is to perform extremely high-speed data forwarding between different aggregation layers.

DC Communications

- Modern data centers are incredibly large in scale, often containing tens or even hundreds of thousands of servers.
- These servers need to constantly communicate with each other—for example, a complex computational task may require multiple servers to collaborate, or the data requested by a user might be distributed across multiple servers.
- This necessitates high-speed, efficient, and reliable network interconnections within the data center.

DC Communications

- Plenty of M2M communications, the principle bottleneck in large-scale clusters is often inter-node communication bandwidth.
  - MapReduce: must perform significant data shuffling to transport the output of its map phase before proceeding with its reduce phase.
  - Web search engine: often requires parallel communication with every node in the cluster hosting the inverted index to return the most relevant results
- Managed by one single authority

Two approaches for DC network

- Approach 1: Specialized hardware and communication protocols
  - For example: InfiniBand, Myrinet
  - Do not leverage commodity parts, expensive
  - Not compatible with TCP/IP applications
- Approach 2: Leverages commodity Ethernet switches and routers to interconnect cluster machines.
  - Unmodified applications, OS, and hardware
  - Unfortunately, aggregate cluster bandwidth scales poorly with cluster size, and achieving the highest levels of bandwidth incurs non-linear cost increases with cluster size.

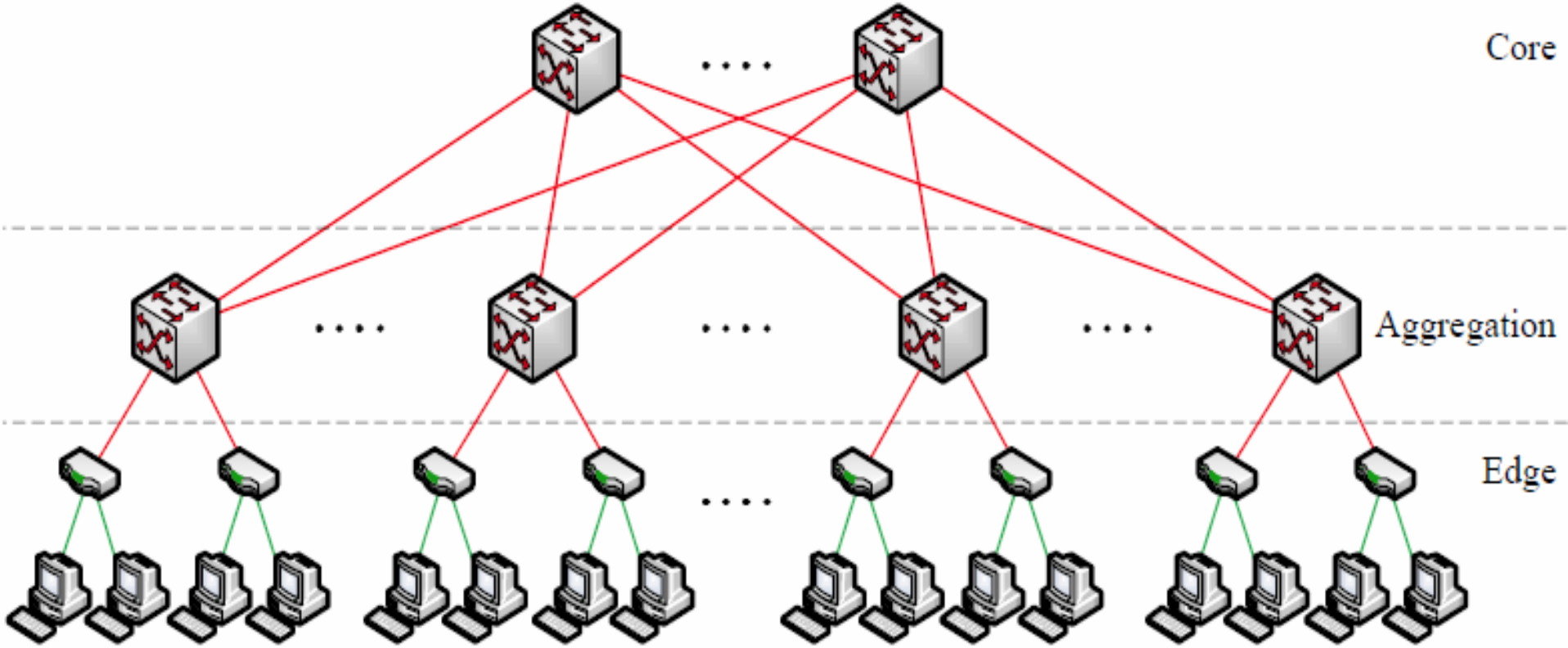
Two approaches for DC network

- For compatibility and cost reasons, most cluster communication systems follow the second approach.
- However, communication bandwidth in large clusters may become oversubscribed by a significant factor depending on the communication patterns.
- How to address these bottlenecks with non-commodity solutions, e.g., large 10Gbps switches and routers？

Desired Properties for a DC Network Architecture

- Scalable interconnection bandwidth: an arbitrary host can communicate with any other host at the full bandwidth of its local network interface.
- Economies of scale: make cheap off-the-shelf Ethernet switches the basis for large scale data center networks.
- Backward compatibility: the entire system should be backward compatible with hosts running Ethernet and IP.

Current Data Center Network Topologies



Current Data Center Network Topologies

- Three tiers: core, aggregation, edge (ToR switch)
- Two types of switches:
  - 48-port GigE switch, with four 10 GigE uplinks, used at the edge of the tree
  - 128-port 10 GigE switch for higher levels of a communication hierarchy
  - Both types of switches allow all directly connected hosts to communicate with one another at the full speed of their network interface.

Problems of the Topology

- Oversubscription: the ratio of the total bisection bandwidth of a particular communication topology to the worst-case achievable aggregate bandwidth among the end hosts.
  - Ideal: 1:1, all hosts may potentially communicate with arbitrary other hosts at the full bandwidth of their network interface
  - Typical designs are oversubscribed by a factor of 2.5:1 to 8:1

Problems of the Topology

- Multi-path Routing: Delivering full bandwidth between arbitrary hosts in larger clusters requires a “multi-rooted” tree with multiple core switches
  - ECMP performs static load-splitting among flows, not condisering flow bandwidth in making allocation decisions, often leading to over-subscription
  - Limit the multiplicity of paths to 8–16

Problems of the Topology

- Cost: 维持一定的oversubscription，cost会随规模急剧增加。

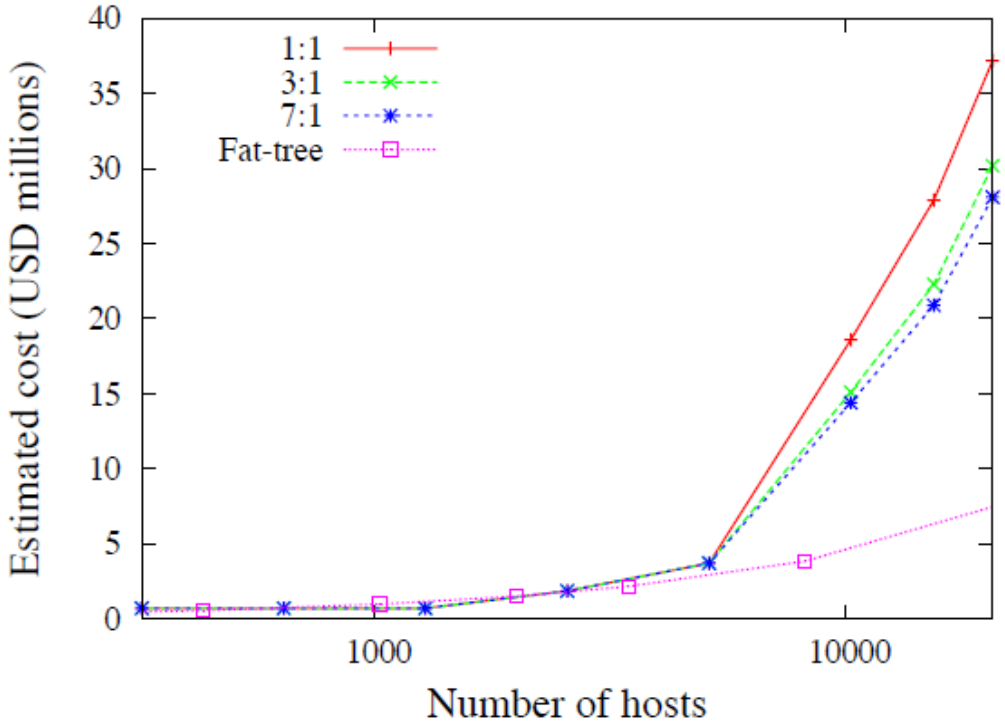


Figure 2: Current cost estimate vs. maximum possible number of hosts for different oversubscription ratios.

Problems of the Topology

- Cost:
  - Using the largest 10 GigE and GigE switches to build a datacenter with 1:1 oversubscription
  - A cluster can be up to 27,648 hosts

	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

Table 1: The maximum possible cluster size with an oversubscription ratio of 1:1 for different years.

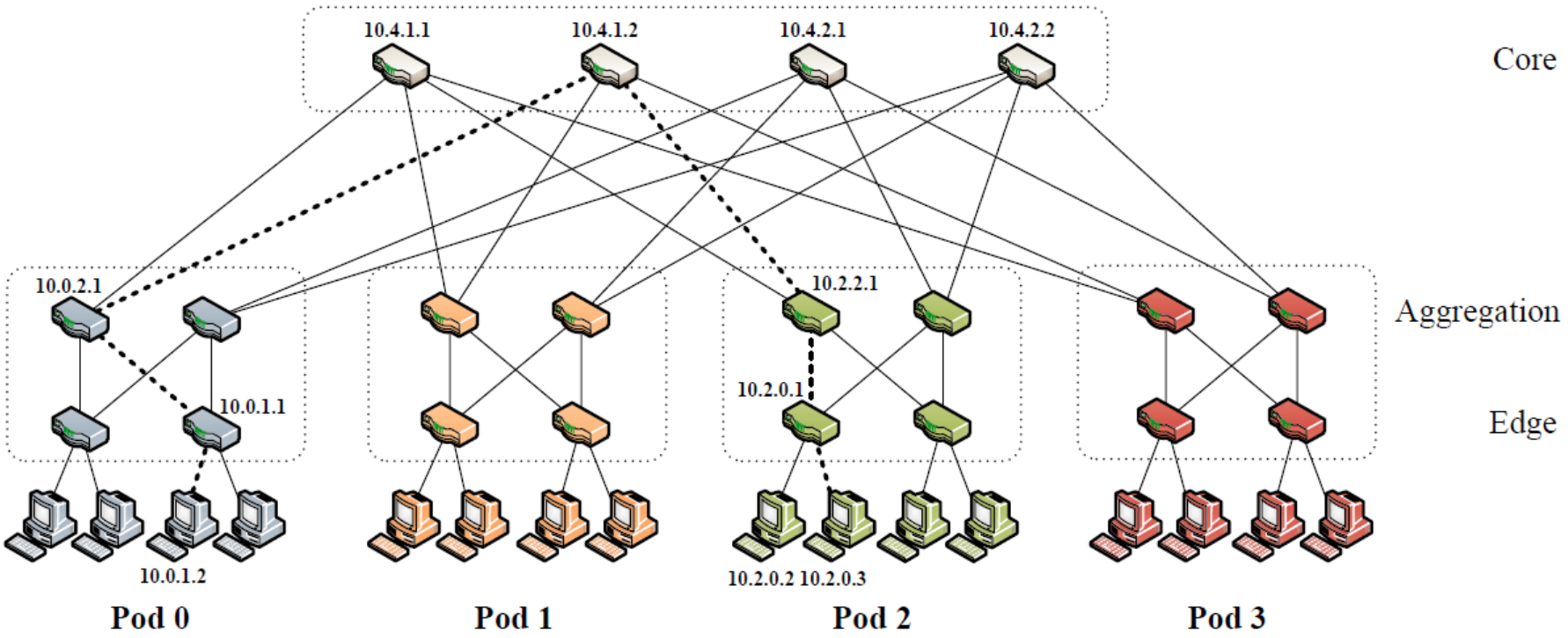
Problems of the Topology

- Cost:
  - Traditional high-end switch architectures are constrained by port density: in 2008, 10 GigE switches were just becoming commercialized, with per-port costs about 5 times higher than Gigabit switches, though this gap was steadily narrowing.
  - the topology had better require no high-speed uplinks and offers superior scalability

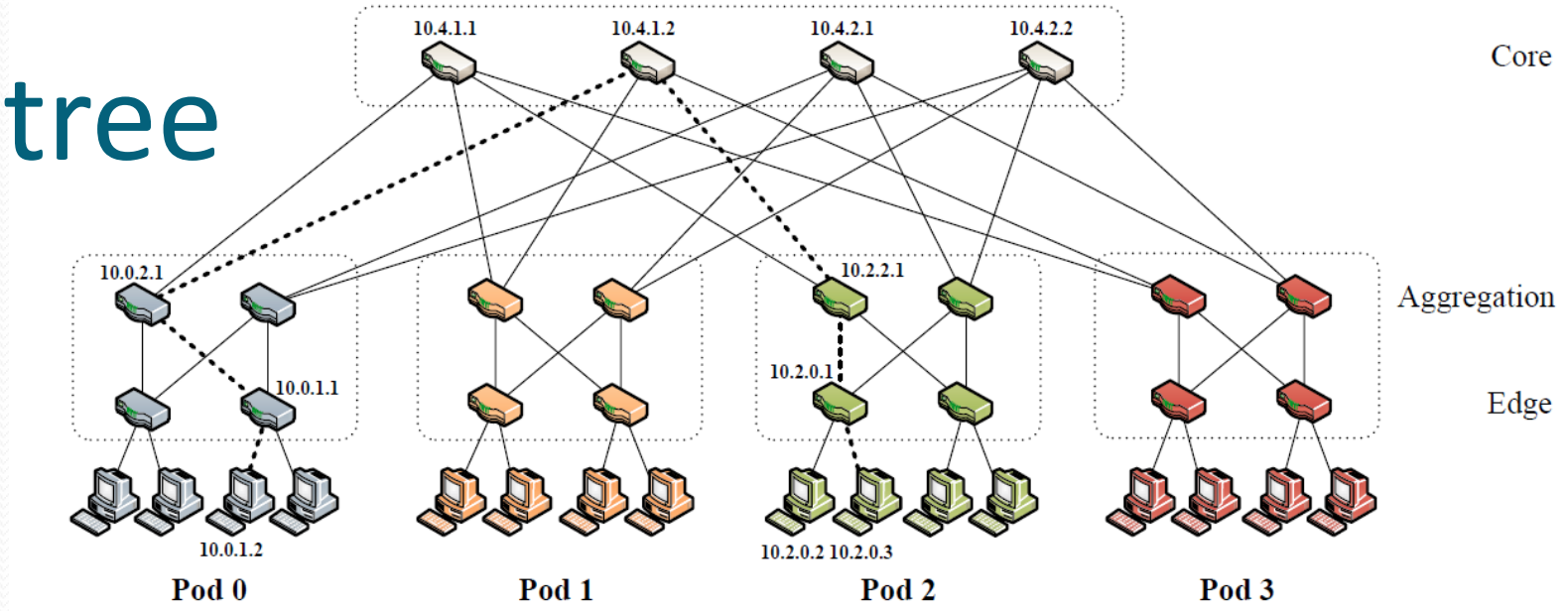
Outline

- Background
- Fat tree based solution
- Implementation and evaluation
- Review

Fat-tree

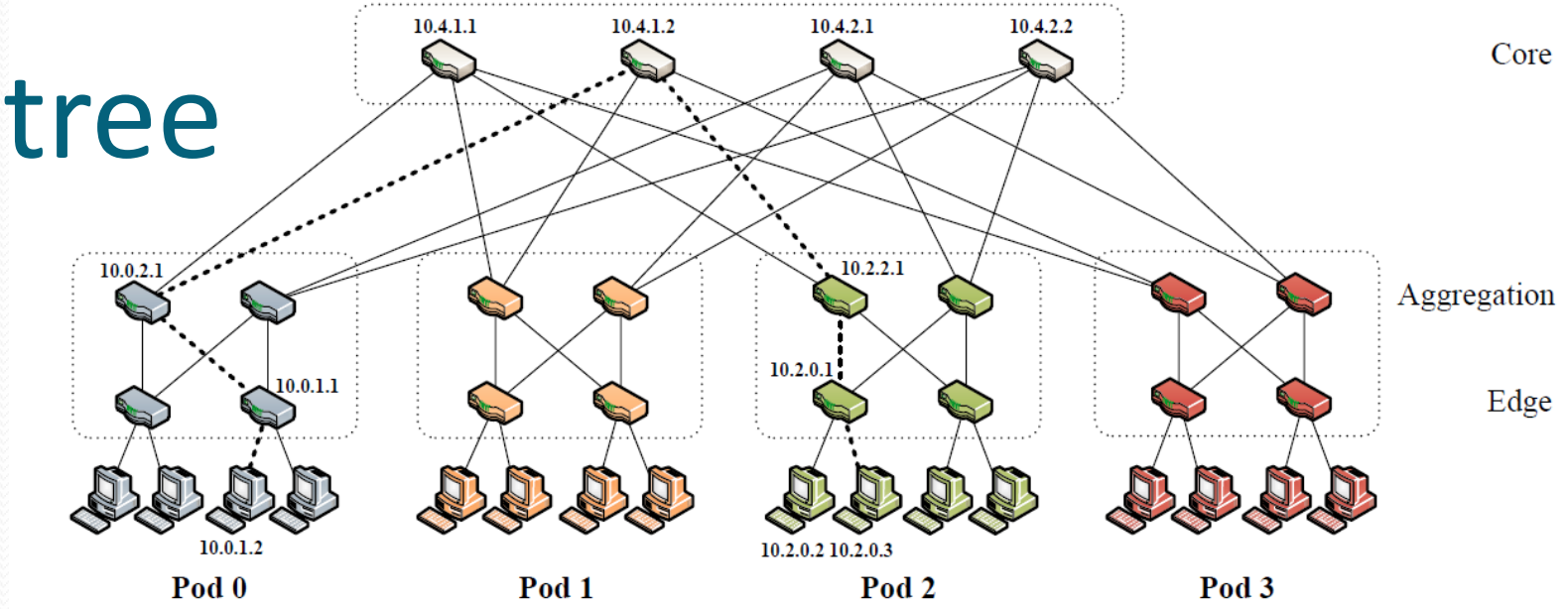

10.4.1.1
10.4.1.2
10.4.2.1
10.4.2.2
Core
10.0.2.1
10.0.1.1
10.0.1.2
10.2.2.1
10.2.0.1
10.2.0.2 10.2.0.3
Aggregation
Edge
Pod 0
Pod 1
Pod 2
Pod 3

Fat-tree



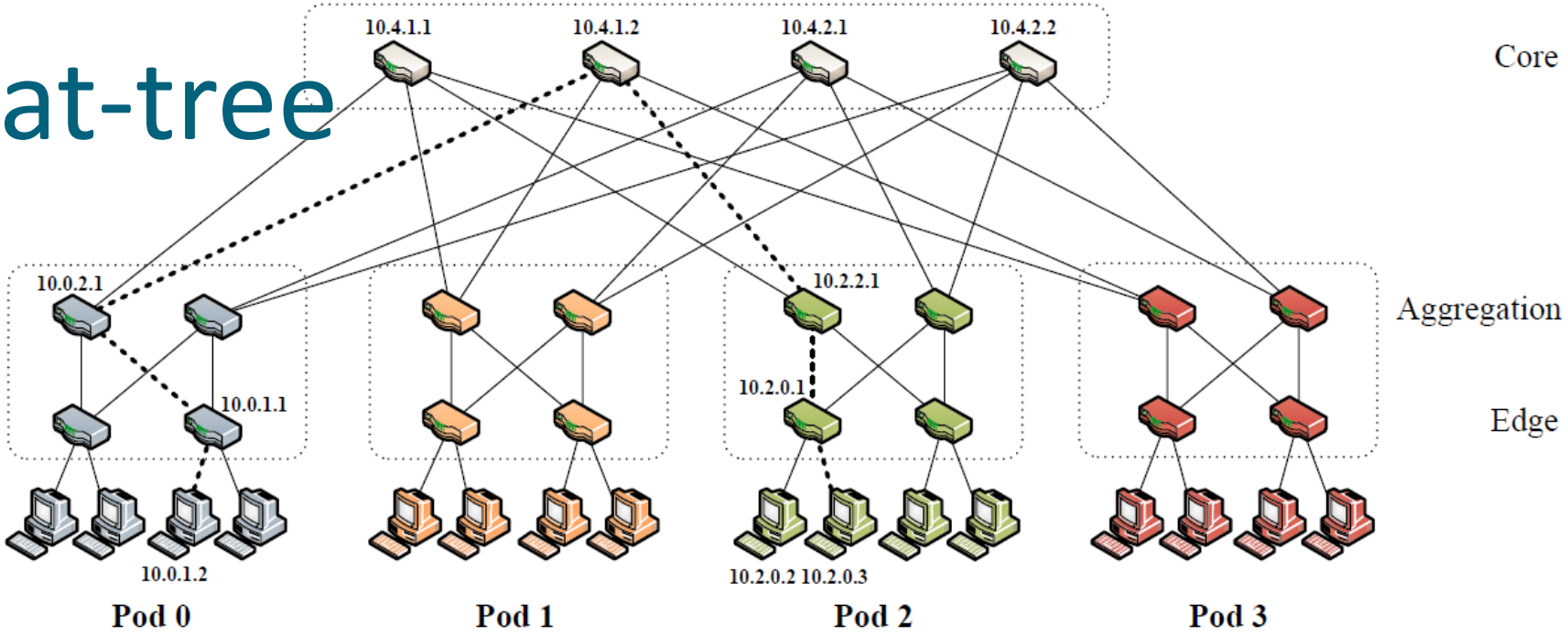
- *k* is the number of the ports of a switch.
- *k* pods, each containing two layers of *k*/2 switches.
- Each *k*-port switch in the lower layer is directly connected to *k*/2 hosts.
  - The total number of hosts is $k^3/4$. Why?

Fat-tree



- Each of the remaining $k/2$ ports is connected to $k/2$ of the k ports in the aggregation layer.
- The number of core switches?
  - Hint: considering the remaining number of ports in the aggregation layer

Fat-tree



- $(k/2)^2$ k-port core switches. Each has one port connected to each of k pods.
- The i^{th} port of any core switch is connected to pod i
- $(k/2)^2$ shortest-paths between any two hosts

Fat-tree

- Focus on designs up to $k = 48$.
- Use identical 48-port GigE switches.
- The network supports 27,648 hosts, made up of 1,152 subnets with 24 hosts each. There are 576 equal-cost paths between any given pair of hosts in different pods.
- The cost of deploying such a network architecture would be $8.64M, compared to $37M for the traditional techniques.

Architecture Design

- Motivation
  - There are $(k/2)^2$ shortest-paths between any two hosts on different pods, but only one is chosen.
    - Each path has 5 hops
  - Protocols like OSPF selects path based on hop counts.
    - it is possible for a small subset of core switches, perhaps only one, to be chosen as the intermediate links between pods.
  - Need a simple, fine-grained method of traffic diffusion.

Addressing

- All IP addresses in the network within the private 10.0.0.0/8 block.
- The pod switches are given addresses of the form 10.*pod.switch*.1,
  - *pod* denotes the pod number (in [0, *k*−1]),
  - *switch* denotes the position of that switch in the pod (in [0, *k*−1], starting from left to right, bottom to top).
- Give core switches addresses of the form 10.*k*.*j*.*i*,
  - *j* and *i* denote that switch's coordinates in the $(k/2)^2$ core switch grid (each in [1, (*k*/2)], starting from top-left).

Addressing

- The hosts are given addresses of the form 10.pod.switch.ID,
  - ID is the host's position in that subnet (in [2, k/2+1], starting from left to right)
  - Therefore, each lower-level switch is responsible for a /24 subnet of k/2 hosts (for k < 256)

Two-level Routing Table

- Each entry in the main routing table will potentially have an additional pointer to a small secondary table of (*suffix*, *port*) entries.
  - A first-level prefix is terminating if it does not contain any second level suffixes,
  - A secondary table may be pointed to by more than one first-level 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

Two-level Routing Table

- Entries in the primary table are left-handed (i.e., */m* prefix masks of the form 1^m0^{32-m}), entries in the secondary tables are right-handed (i.e. */m* suffix masks of the form $0^{32-m}1^m$).
- If the longest-matching prefix search yields a non-terminating prefix, then the longest-matching suffix in the secondary table is found and used.

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

Two-level Routing Table

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

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.

- The routing table of any pod switch will contain no more than $k/2$ prefixes and $k/2$ suffixes.

Two-Level Lookup Implementation

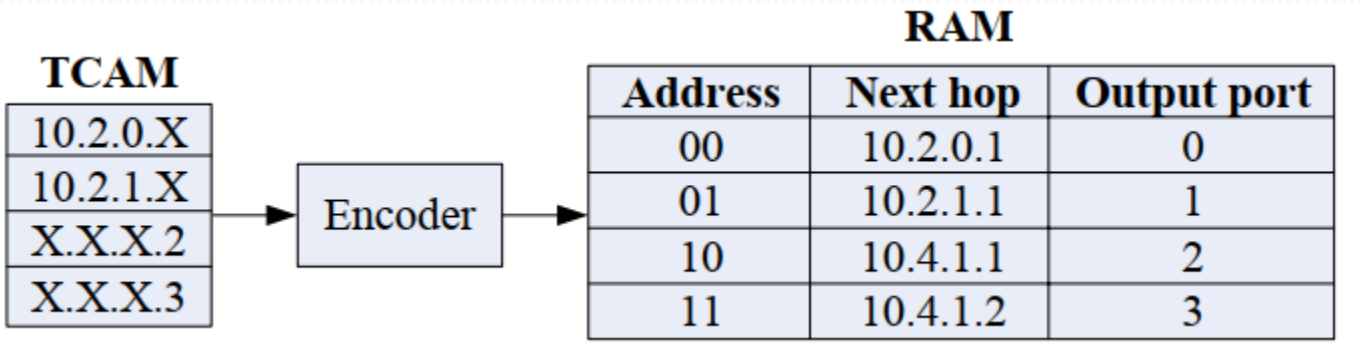
- The two-level lookup can be implemented in hardware using Content-Addressable Memory (CAM)
- A CAM can perform parallel searches among all its entries in a single clock cycle.
- A TCAM can store don't care bits in addition to matching 0's and 1's in particular positions, making it suitable for storing variable length prefixes

Two-Level Lookup Implementation

- Hardware Structure: use a TCAM that stores both prefixes and suffixes. This TCAM directly indexes a RAM which contains the corresponding next-hop IP address and output port.
- Storage and Priority Encoding: Left-handed (prefix) entries are stored at lower numerical addresses, while right-handed (suffix) entries are stored at higher addresses.
- The CAM output always select the entry with the smallest matching address, ensuring a left-handed entry is chosen when both a prefix and a suffix match.

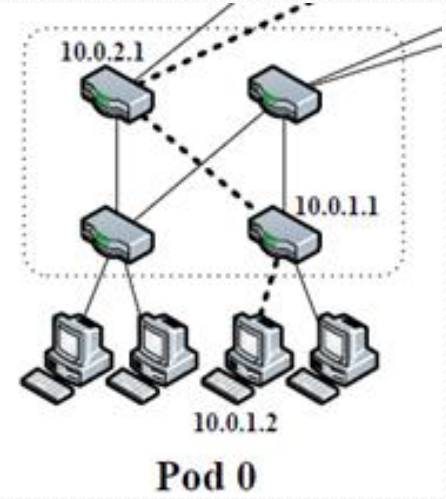
Two-Level Lookup Implementation

- Routing Semantics and Example: This design ensures correct longest-prefix-match semantics.
- For example, using the routing table in Figure below, a packet with destination IP address 10.2.0.3 matches the left-handed entry 10.2.0.X and the right-handed entry X.X.X.3. The packet is correctly forwarded on port 0. However, a packet with destination IP address 10.3.1.2 matches only the right-handed entry X.X.X.2 and is forwarded on port 2.



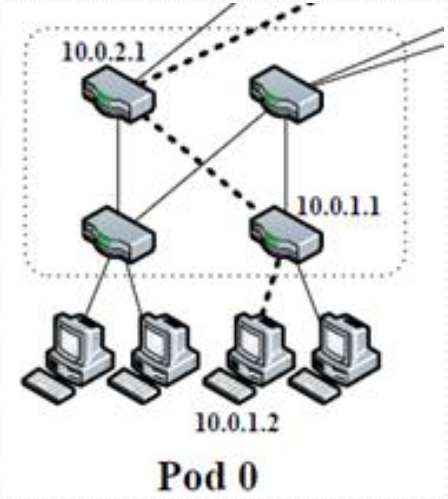
Routing Algorithm

- Pod switches
  - If a host sends a packet to another host in the same pod but on a different subnet, then all upper-level switches in that pod will have a terminating prefix pointing to the destination subnet's switch.



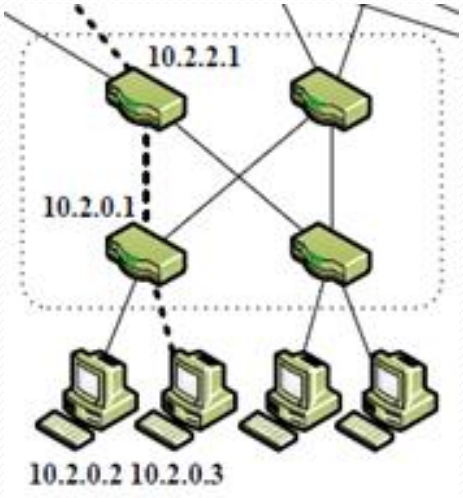
Routing Algorithm

- Pod switches
  - For all other outgoing inter-pod traffic, the pod switches have a default /0 prefix with a secondary table matching host IDs. Employ the host IDs as a source of deterministic entropy; they will cause traffic to be evenly spread upward among the outgoing links to the core switches



Routing Algorithm

- Aggregation switches
  - Once a packet reaches its destination pod, the receiving upper-level pod switch will also include a (10.*pod*.*switch*.0/24, *port*) prefix to direct that packet to its destination subnet switch, where it is finally switched to its destination host.



```
1  foreach pod x in [0, k − 1] do
2      foreach switch z in [(k/2), k − 1] do
3          foreach subnet i in [0, (k/2) − 1] do
4              addPrefix(10.x.z.1, 10.x.i.0/24, i);
5          end
6          addPrefix(10.x.z.1, 0.0.0.0/0, 0);
7          foreach host ID i in [2, (k/2) + 1] do
8              addSuffix(10.x.z.1, 0.0.0.i/8,
               (i − 2 + z)mod(k/2) + (k/2));
9          end
10     end
11 end
```

Generating upper aggregation switch routing table;

For lower switches, z in [0, (*k*/2)-1], and omit line 3-5.

Routing Algorithm

- Core switches
  - Once a packet reaches a core switch, there is exactly one link to its destination pod, and that switch will include a terminating /16 prefix for the pod of that packet (10.*pod*.0.0/16, *port*).

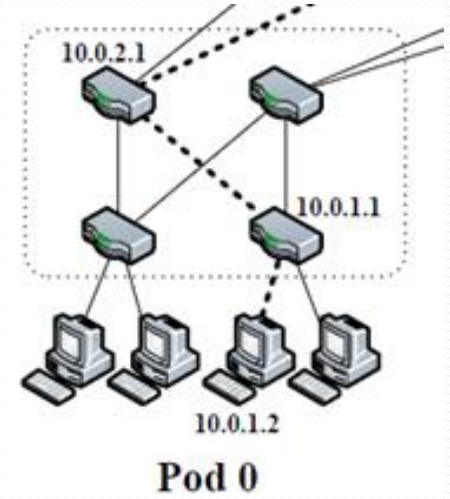
```
1 foreach j in [1, (k/2)] do
2     foreach i in [1, (k/2)] do
3         foreach destination pod x in [0, (k/2) − 1] do
4             addPrefix(10.k.j.i,10.x.0.0/16, x);
5         end
6     end
7 end
```

Algorithm 2: Generating core switch routing tables.

An Example

- Source: 10.0.1.2; destination: 10.2.0.3
  - At the gateway switch (10.0.1.1), matches with the /0 first-level prefix, then matches with the 0.0.0.3/8 secondary-level suffix, then forward to port 2, and routed to the pod switch 10.0.2.1.
    - (i=3, z=1)

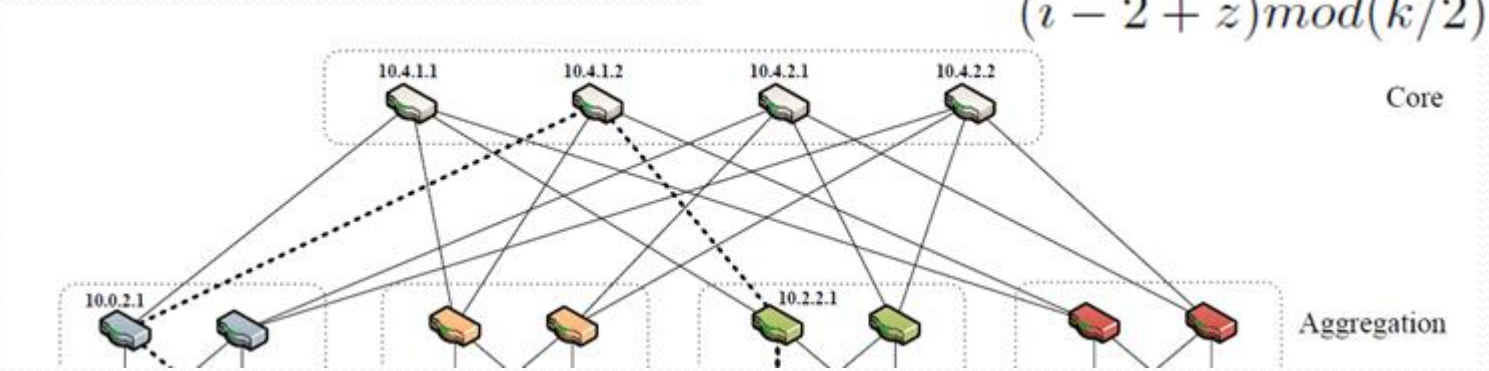
```
6  addPrefix(10.x.z.1, 0.0.0.0/0, 0);
7  foreach host ID i in [2, (k/2) + 1] do
8      addSuffix(10.x.z.1, 0.0.0.i/8,
       (i − 2 + z)mod(k/2) + (k/2));
```



An Example

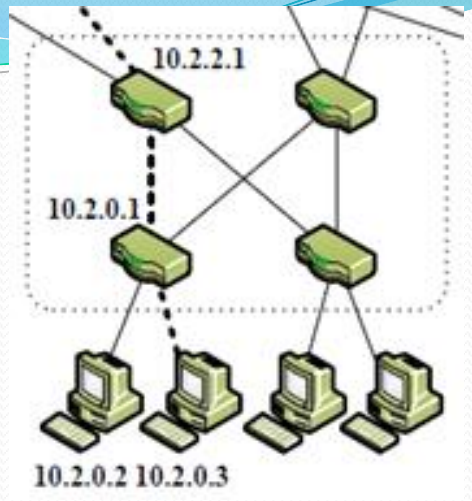
- At the gateway switch (10.0.2.1), matches with the /0 first-level prefix, then matches with the 0.0.0.3/8 secondary-level suffix, then forward to port 3, and routed to the core switch 10.4.1.1.
  - (i=3, z=2)

```
6  addPrefix(10.x.z.1, 0.0.0.0/0, 0);
7  foreach host ID i in [2, (k/2) + 1] do
8      addSuffix(10.x.z.1, 0.0.0.i/8,
       (i − 2 + z)mod(k/2) + (k/2));
```



An Example

3 **foreach** *destination pod* x *in* $[0, (k/2) - 1]$ **do**
4 addPrefix($10.k.j.i$,$10.x.0.0/16$, x);



- At 10.4.1.1, matches a terminating 10.2.0.0/16 prefix, which points to pod 2 on port 2, and switch 10.2.2.1.
- At 10.2.2.1, matches a terminating prefix 10.2.0.0/24, which points to the switch responsible for that subnet, 10.2.0.1 on port 0.

- How about the destination becomes 10.2.0.2?
- Centralized algorithm, not a distributed one.
  - Why feasible?

An Example

- For the destination 10.2.0.2：
  - At gateway switch (10.0.1.1), matches port 3, next hop (10.0.3.1).
  - At gateway switch (10.0.3.1), matches port 3, next hop (10.4.2.2).
  - The following routing matches terminating first-level prefixes.

Flow Classification

- The two-level routing technique is static, but traffics are not evenly distributed among the hosts.
- Edge switch
  1. Recognize subsequent packets of the same flow, and forward them on the same outgoing port. (packets of same <src IP, dst IP, src port, dst port, proto> belong to a same flow)
  2. Periodically reassign a minimal number of flow output ports to minimize any disparity between the aggregate flow capacity of different ports.

Flow Scheduling

- Traffics are dominated by few large long-lived flows
- Edge switch
  - Additionally detect any outgoing flow whose size grows above a predefined threshold, and periodically send notifications to a central scheduler specifying the source and destination for all active large flows.

Flow Scheduling

- Central Scheduler
  - Maintains boolean state for all links in the network their availability to carry large flows.
  - When the scheduler receives a notification of a new flow, it linearly searches through the core switches to find one whose corresponding path components do not include a reserved link.
  - Upon finding such a path, the scheduler marks those links as reserved, and notifies the relevant lower- and upper-layer switches in the source pod with the correct outgoing port that corresponds to that flow's chosen path.

Fault-Tolerance

- Lower- to Upper-layer Switches
  - impacts outgoing inter/intra-pod traffic, intra-pod traffic using the upper-layer switch as an intermediary, and incoming inter-pod traffic
  - The response involves local cost adjustment, broadcast of failure tags within the pod to lower-layer switches, and propagation of the failure status to core switches and subsequently to upper-layer switches in other pods to avoid the affected path.

Fault-Tolerance

- Upper-layer to Core Switches
  - affects outgoing and incoming inter-pod traffic
  - The response involves local rerouting and a core switch broadcasting a failure tag to all its connected upper-layer switches in other pods, instructing them to avoid it for traffic destined to the affected pod.

Power and Heat Issues

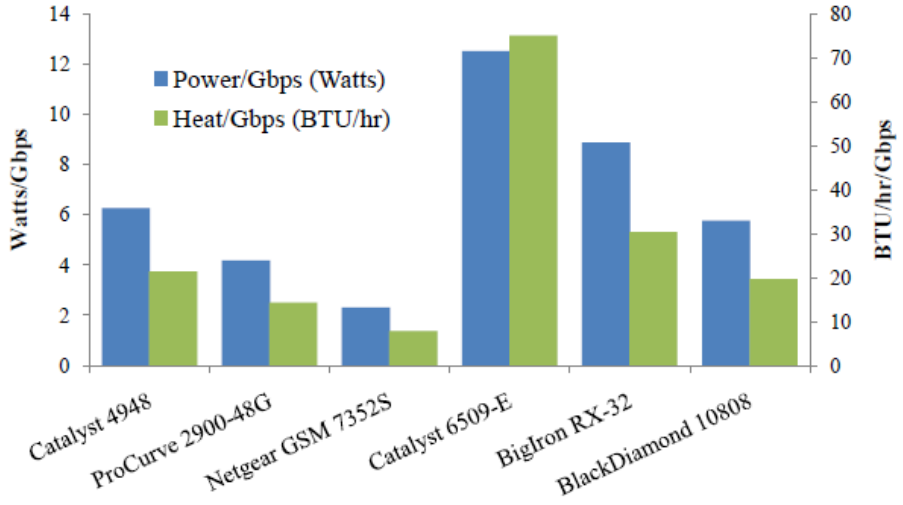


Figure 6: Comparison of power and heat dissipation.

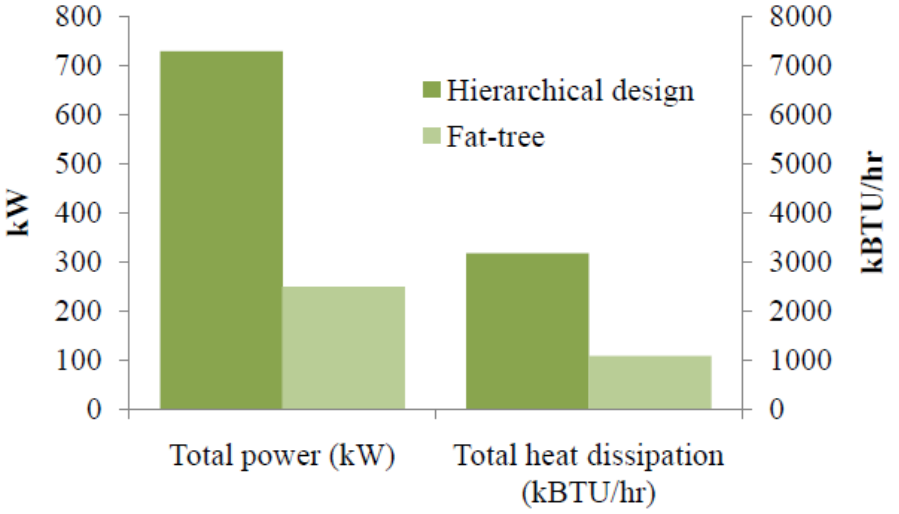


Figure 7: Comparison of total power consumption and heat dissipation.

- 不同Switch的能耗效率，后三个是10GigE的switch
- Employs more individual switches, is superior to those incurred by current data center designs, with 56.6% less power consumption and 56.5% less heat dissipation.

Outline

- Background
- Fat tree based solution
- Implementation and evaluation
- Review

Implementation

- Implement router prototype with Click.
  - Support the two-level routing table
  - 4-ports

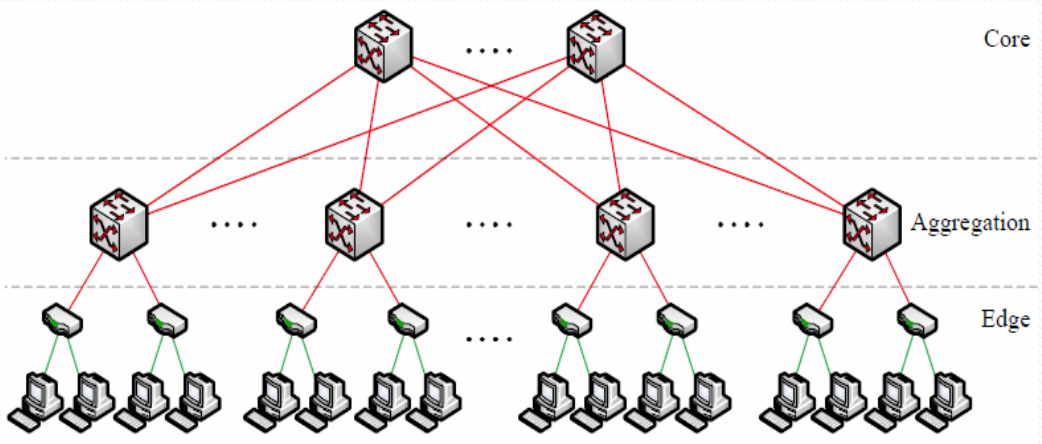
- The Click Modular Router Project
  - A software architecture for building flexible and configurable routers
  - https://github.com/kohler/click

Experiment Description

- Implement a 4-port fat-tree (*k*=4): there are 16 hosts, four pods (each with four switches), and four core switches.
  - Multiplex these 36 elements onto ten physical machines, interconnected by a 48-port ProCurve 2900 switch with 1 Gigabit Ethernet links.
  - Each pod of switches is hosted on one machine; each pod's hosts are hosted on one machine; and the two remaining machines run two core switches each.

Experiment Description

- Hierarchical tree network,
  - four machines running four hosts each, and four machines each running pod switches with one additional uplink.
  - The four pod switches are connected to a 4-port core switch running on a dedicated machine



Benchmark Suite

- Random: A host sends to any other host in the network with uniform probability.
- Stride(*i*): A host with index *x* will send to the host with index (*x*+*i*) mod 16.
- Staggered Prob (*SubnetP,PodP*): Where a host will send to another host in its subnet with probability *SubnetP*, and to its pod with probability *PodP*, and to anyone else with probability 1−*SubnetP*−*PodP*.

Benchmark Suite

- Inter-pod Incoming: Multiple pods send to different hosts in the same pod, and all happen to choose the same core switch. That core switch's link to the destination pod will be oversubscribed. The worst-case local oversubscription ratio for this case is (k−1):1.

Benchmark Suite

- Same-ID Outgoing: Hosts in the same subnet send to different hosts elsewhere in the network such that the destination hosts have the same host ID byte. Static routing techniques force them to take the same outgoing upward port. The worst-case ratio for this case is (k/2):1.

Results

Test	Tree	Two-Level Table	Flow Classification	Flow Scheduling
Random	53.4%	75.0%	76.3%	93.5%
Stride (1)	100.0%	100.0%	100.0%	100.0%
Stride (2)	78.1%	100.0%	100.0%	99.5%
Stride (4)	27.9%	100.0%	100.0%	100.0%
Stride (8)	28.0%	100.0%	100.0%	99.9%
Staggered Prob (1.0, 0.0)	100.0%	100.0%	100.0%	100.0%
Staggered Prob (0.5, 0.3)	83.6%	82.0%	86.2%	93.4%
Staggered Prob (0.2, 0.3)	64.9%	75.6%	80.2%	88.5%
Worst cases:				
Inter-pod Incoming	28.0%	50.6%	75.1%	99.9%
Same-ID Outgoing	27.8%	38.5%	75.4%	87.4%

Aggregate Bandwidth of the network, as a percentage of ideal bisection bandwidth. The ideal bisection bandwidth for the fat-tree network is 1.536Gbps

Review

- What is the datacenter network? What is the desired property of the datacenter network?
- What is the traditional three-tier topology for the datacenter, its limitations?
- How Fat-tree differs from the traditional design? In
  - Topology
  - Addressing
  - Routing algorithm