A Scalable, Commodity Data Center Network Architecture

By John Drab

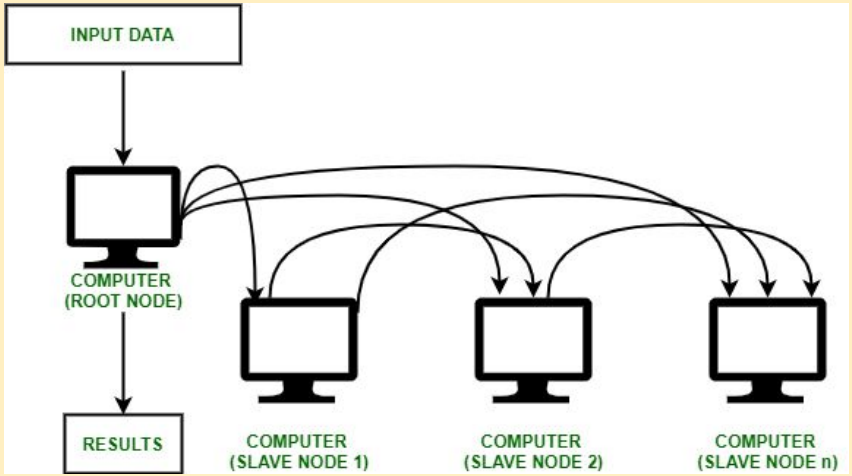
10/02/2025

UCDAVIS

Background

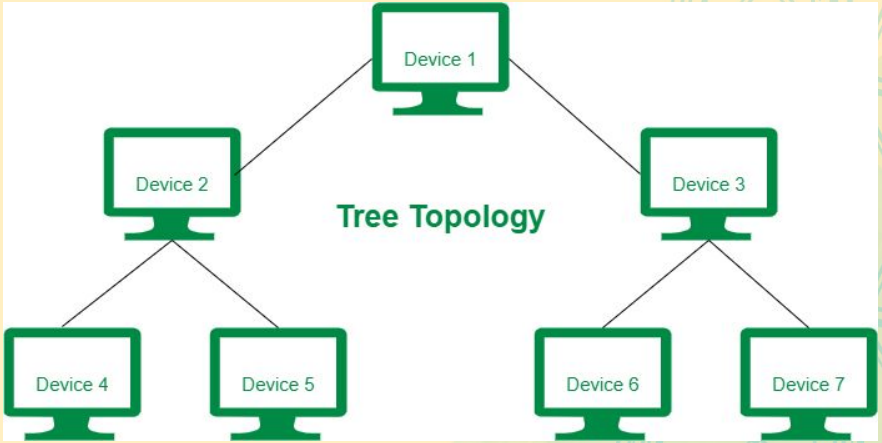
Data Center Cluster Computing

- Rapid scaling of data center compute with commodity PCs



Existing Network Topology

- Three-tier network architecture
- Expensive specialized switches at top



Oversubscription

Three-tiered tree topologies (Core -> Aggregate -> Edge -> Host)

- Hosts connect directly to edges (Top-of-Rack, ToR switch)
- ToR uplinks to larger switches (Aggregation Layer)
- Aggregate switches link to very high-capacity routers/switches at ‘top’ (Core)

Oversubscription

- Cost of larger capacity switches does not scale linearly (Very expensive!)
- Example: Base host to ToR connection (e.g., 1Gbps) x 40 hosts (= 40 Gbps)
- But ToR uplink bandwidth only 10 Gbps
- Leads to oversubscription ratio of 4:1

Three-Tier Topology Example

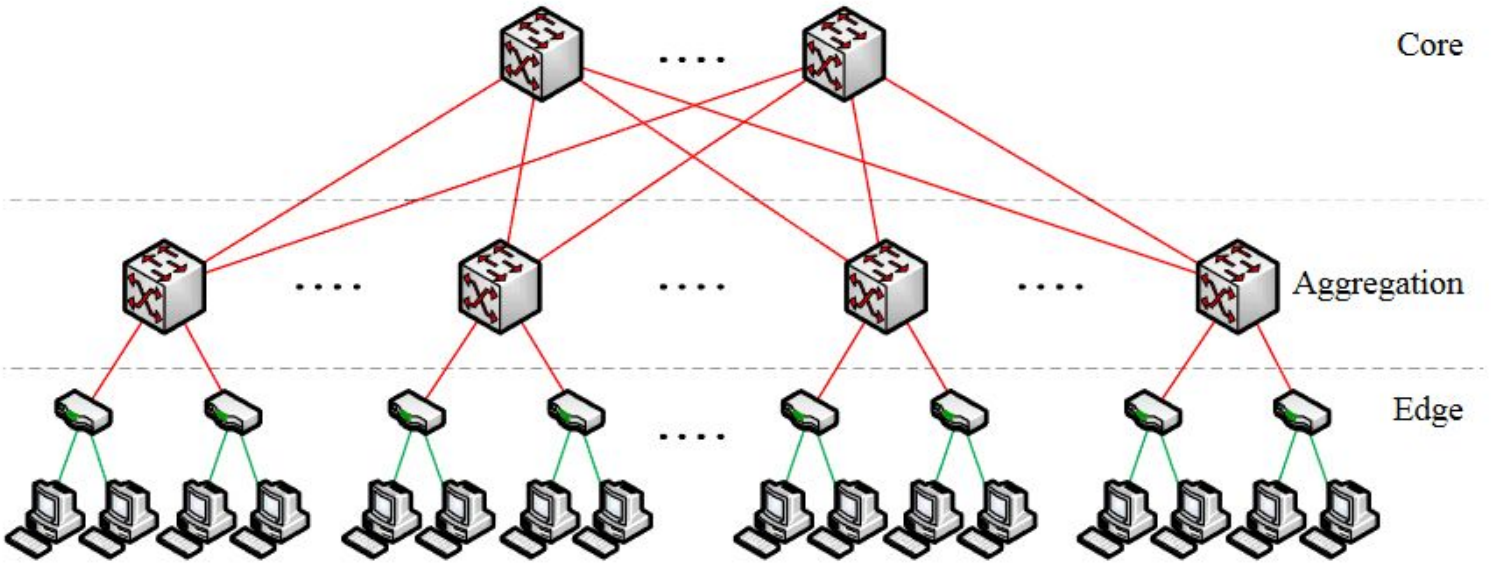


Figure 1: Common data center interconnect topology. Host to switch links are GigE and links between switches are 10 GigE.

Primary Motivation

Bandwidth and Scalability

- Current network topologies for data centers are cost-ineffective at supporting all available bandwidth at edges
- Cluster computing bottlenecked by inter-node communication

Economies of Scale

- Leverage *commodity switches* which provide cost and compatibility benefits compared to specialized hardware
- Existing tree topologies struggle with networks being oversubscribed
- Non-uniform bandwidth results in congestion at root nodes

Commodity Hardware

VS

Speciality Hardware

Bandwidth Bottlenecks

Multi-Path Routing

- Single-root tree: Each edge switch only has one uplink to the core. All traffic must share that single link, which becomes a bottleneck.
- In a *multi-root* tree: Each edge switch has multiple uplinks to different core switches, creating several equal-cost uplinks.

ECMP Load Balancing

- **ECMP (Equal-Cost Multi-Path):** Traffic spread using a hash of each flow's IP addresses, ports, etc. to pick one uplink. All packets in that flow stick to the same path
- **Static:** ECMP doesn't look at how big flows are. Two huge flows might land on the same link, overloading it, while other links stay empty.
- **Limited paths:** Switches usually only support 8–16 paths, even if the network has more available.

Fat-Tree Topology

Scalable: k-ary fat-tree supports ($k^3/4$) servers.

Full bisection bandwidth: Any server can communicate with any other at line rate.

Commodity switches: Uses multiple cheap switches instead of expensive routers.

Multipath routing: Equal-cost paths spread traffic, improving throughput & fault tolerance.

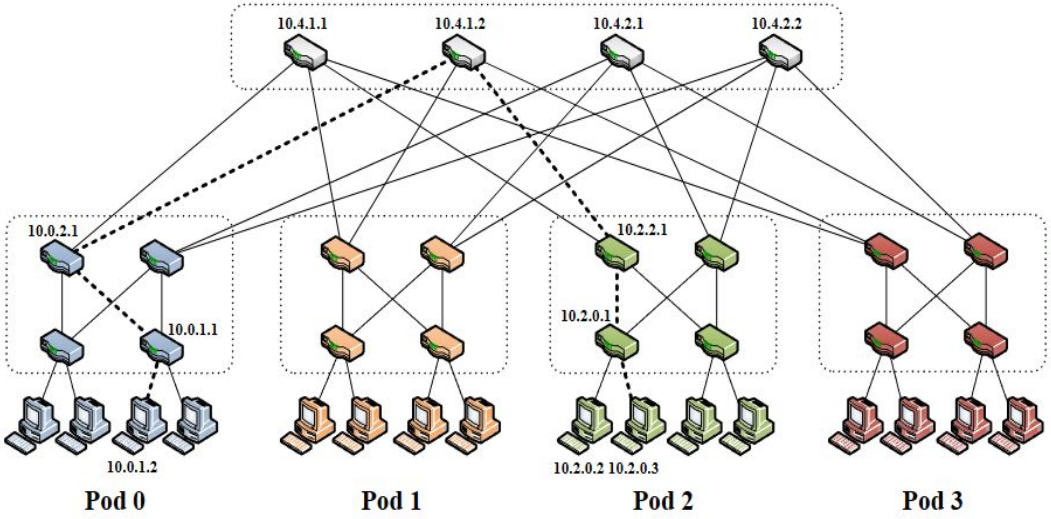


Figure 3: Simple fat-tree topology. Using the two-level routing tables described in Section 3.3, packets from source 10.0.1.2 to destination 10.2.0.3 would take the dashed path.

Fat-Tree Architecture

- Goal of optimizing **bisection bandwidth** in the network, taking advantage of Fat-tree's large fanout
- Fat-tree layout not optimized for traditional routing (i.e OSPF, OSPF-ECMP)
- **Need to create a simple, fine-grained traffic diffusion method:**
  - Utilize two-level routing tables to distribute traffic based on low-order bits of destination IP
  - IP address follows quad-dotted form:
    - **Pods**: 10.pod.switch.1 | **Core switches**: 10.k.j.i | **Hosts**: 10.pod.switch.ID

Bisection Bandwidth

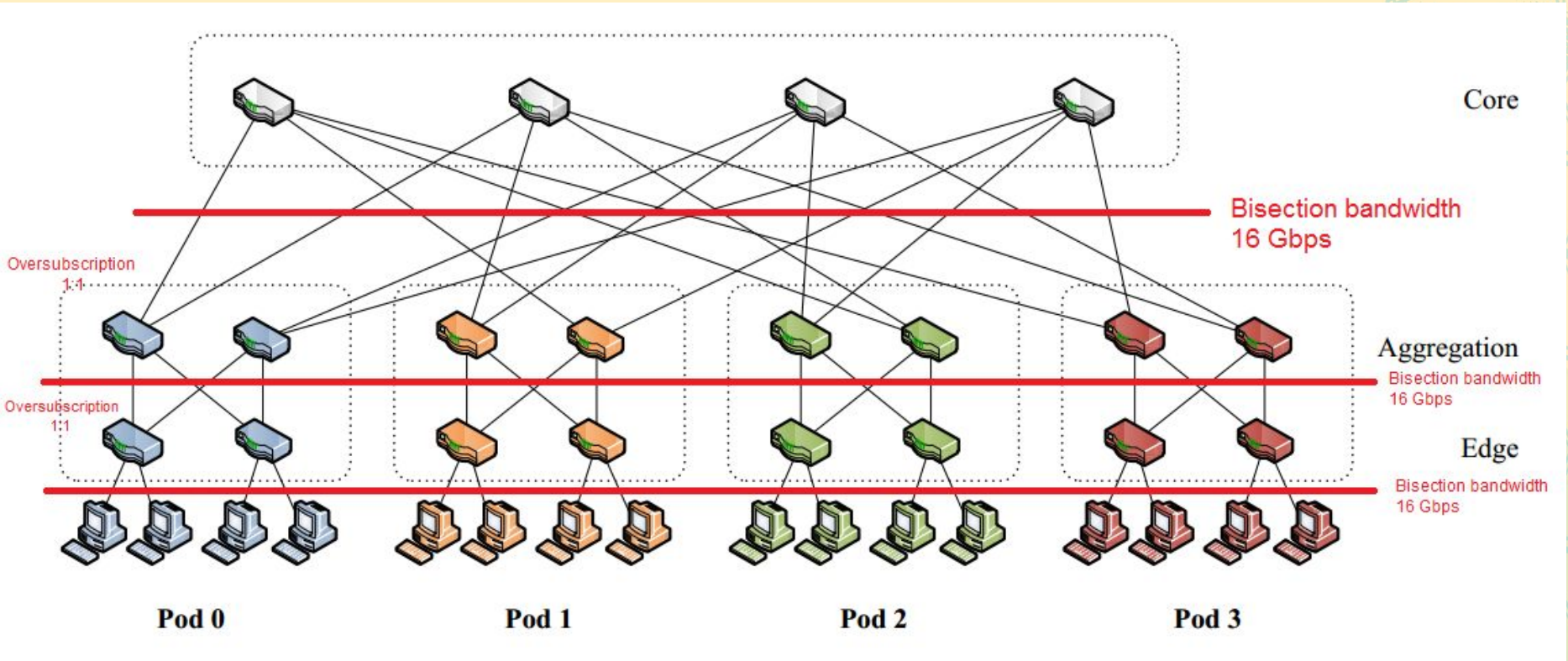


Image source

Routing Table

- Two-level lookup design ensures even traffic distribution across multiple equal-cost paths
- Structure of two-level lookup:
  - Primary table: matches prefixes (left-handed, e.g., 10.2.0.X)
  - Secondary table: matches suffixes (right-handed, e.g., X.X.X.3), each prefix may point to a suffix table for more fine-grained routing.
- **Crucially allows for efficient routing with commodity switch table size limitations**

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.

Two-level TCAM Lookup Engine

- Hardware based Ternary Content Addressable Memory (TCAM):
  - Matches variable-length prefixes/suffixes in 1 clock cycle and supporters ‘don’t care’ (X) bits -useful for IP routing

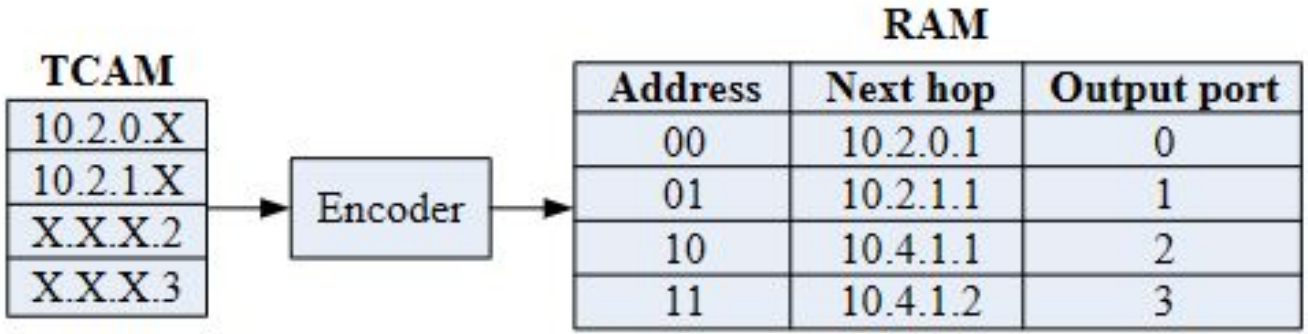


Figure 5: TCAM two-level routing table implementation.

Routing Algorithm

- Intra-pod traffic: Upper and lower pod switches hold terminating prefixes to subnets in the pod
  - Uses a default /0 prefix with secondary suffix-based lookup.
  - Host IDs (last byte of IP) are used to deterministically spread traffic to core switches.
  - Prevents packet reordering by maintaining consistent paths for flows.
- Each core switch holds terminating /16 prefixes (e.g., 10.2.0.0/16) for destination pods
  - Once in the destination pod, upper switches forward based on a /24 subnet prefix.
- Central controller computes and installs small static routing tables of size ≤ k first-level prefixes and ≤ k/2 second-level suffixes

Flow Control

- Flow = sequence of packets sharing key header fields (e.g., src/dst IP, transport port).
  - Important to optimize to avoid local congestion and balance load across upward ports in pod switches and preserve packet ordering while improving throughput.
- Dynamic Port Reassignment:
  - Recognize and maintain consistent path for each flow (prevents reordering).
  - Periodically reassign ports to minimize flow imbalance across available paths
  - Can revert to standard two-level routing if switch state is lost: stateless fallback

Fault-Tolerance/Power & Heat

- Nature of Fat-tree enables fault resilience with multiple redundant paths
- Switches use BFD sessions to detect neighbor failures.
- Upon link or switch failure, broadcast tags inform peers to avoid the affected path:
  - Lower→Upper failure: reroute outgoing flows locally, notify peers in the pod and core switches.
  - Upper→Core failure: reroute inter-pod flows locally; core notifies all connected upper-layer switches.
- Fat-tree architecture uses many low-power switches, results in significant power and heat reductions (~56.55% total power and heat)

Implementation

NetFPGA Prototype

- NetFPGA board running an open-source IPv4 router with TCAM lookup
- Added two-level routing lookup table into the TCAM pipeline
- Results: no measurable increase in latency, showed that this process could be done with existing commodities with no downside (not just theoretical but can be implemented today)

Click-Based Prototype

- The authors built every switch (core, aggregation, edge) as a user-level Click Router on Linux VMs, where Click is designed as a graph of processing modules to perform tasks like router lookup
- Connected the VMs via a real 48-port GigE switch (HP ProCurve), but capped each virtual link to 96 Mbps so that packet-processing CPU cost, not line-rate, dominates.

Evaluation and Results

- Paper evaluated bisection bandwidths using 4-port Fat-tree, 16 hosts, 20 switches.
    - Tested Two-Level Table, Flow Classification, and Flow Scheduling methods

Flow Scheduler with the Fat-tree architecture yielded near-optimal bandwidths, scales well, and is efficient.

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%

Packaging

- **Fat-tree topology requires large number of connections between many switches**
  - 10 GigE can provide some cable reduction but at minimal benefit
- **Devised Pod-based packaging to minimize external cabling and maintain scalability**
  - Combine first two levels of switches into a rack (pod switch)
  - 1152 external ports divided and connected to host and core layer switches (576 each)
  - 576 core switches distributed evenly (12 per pod across 48 pods). Cables between pods/core grouped in sets of 12, enabling organized bundling and simplified routing.
  - Racks are organized around pod switch

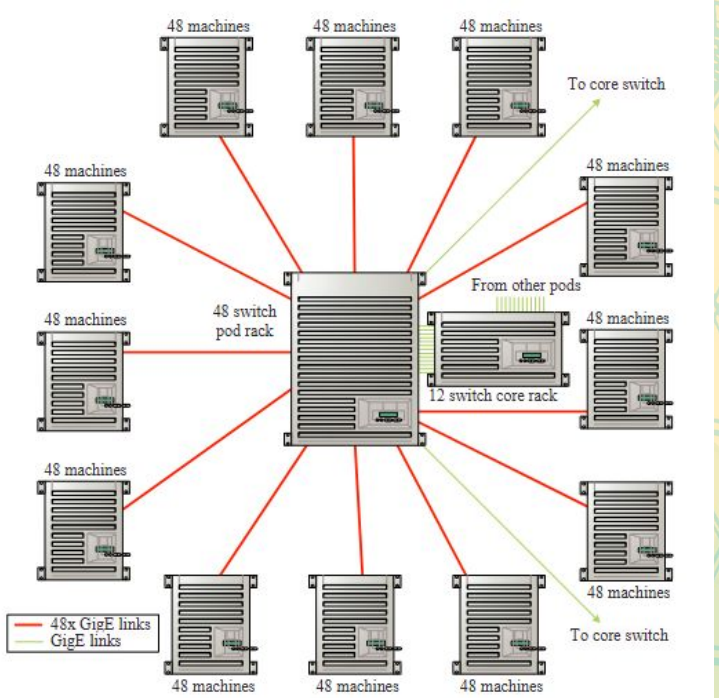


Figure 8: Proposed packaging solution. The only external cables are between the pods and the core nodes.

Thank You!

Discussion

- What kind of advantages could high-bandwidth networking offer in failure recovery or replication-heavy workloads (i.e file system data management)?
- With larger bandwidth data can be moved faster which might reduce the need for complex data locality strategies, how might this improve performance or free up space for larger data applications?