

Analysis of Data Center Network Topologies: Path Diversity, Bisection Bandwidth, and the Fat Tree Visualizer

Au Ze Hong, Maxwell
Cloud Computing — Assignment 1 (Question 5)

February 2026

1 Introduction and Literature Survey

Modern data centers rely on structured network topologies to deliver high bisection bandwidth and multipath redundancy. The dominant paradigm is the *Clos network* [2], which arranges switches in multiple stages so that any input can reach any output through several independent paths. Al-Fares et al. [3] showed that a *k-ary fat tree*—a folded Clos built from commodity switches—delivers full bisection bandwidth cost-effectively.

Three hyperscalers have published their production designs: **Google Jupiter** [4, 5] uses aggregation blocks interconnected via Optical Circuit Switches (OCS); **Amazon** [6] uses a leaf-spine (2-tier Clos) with full bipartite mesh; **Meta** [7, 8] uses a 3-level Clos with spine, fabric, and ToR layers in pods.

We fork the open-source *Fat Tree Visualizer* [1] and extend it with three production topology modules, a BFS-based path analysis engine, an exact bisection bandwidth algorithm, and a fault tolerance analyser.

2 Solution Methodology

2.1 Path Counting via BFS

We count shortest paths using BFS: from source s , maintain $\text{dist}(v)$ and $\text{count}(v)$ for each node. When neighbour u is first discovered, $\text{count}(u) \leftarrow \text{count}(v)$; when reached again at equal distance, $\text{count}(u) += \text{count}(v)$. Complexity: $O(|V| + |E|)$.

For paths through switch w , the *product principle* applies: if $\text{dist}(s, w) + \text{dist}(w, t) = \text{dist}(s, t)$, then paths through $w = \text{count}(s \rightarrow w) \times \text{count}(w \rightarrow t)$.

2.2 Bisection Bandwidth

For fat trees: $B = k^{d-1} \times k = k^d$ (each of k^{d-1} root switches has k links per side). For other topologies: we enumerate all balanced host-group partitions via bitmask (feasible for ≤ 20 groups), greedily assign switches to minimise crossing links, and return the minimum.

2.3 Fault Tolerance Analysis

We remove the most critical switch (a root/spine switch) and re-run BFS on the reduced graph. We measure: (i) host pairs that lose all shortest paths, (ii) average path-count reduction, and (iii) oversubscription ratio (downlinks/uplinks at the edge layer).

3 Answers for the Given Diagram

The assignment diagram is a $k=2, d=3$ fat tree with 4 root switches (S1–S4), 4 level-1 switches, 4 edge switches, and 8 hosts (M1–M8, with M1–M6 labelled).

3.1 Part (a): Server \rightarrow Root Switch

Each server has exactly 1 shortest path to each individual root switch (length 3), since the fat tree’s wiring deterministically maps each upward port to a specific root. There are $k^{d-1} = 4$ root switches, so 4 total root-reachable paths:

$$\text{Paths per root} = 1, \quad \text{Total root switches} = k^{d-1} = 4$$

Verified: M1 \rightarrow S1 = 1 path, M1 \rightarrow S2 = 1 path, \dots , M1 \rightarrow S4 = 1 path (each length 3).

3.2 Part (b): Host-to-Host Paths

3.3 Part (c): Bisection Bandwidth

$$B = k^d = 2^3 = 8 \text{ link-units}$$

The minimum cut is at the root level: 4 root switches \times 2 links each crossing the bisection.

Table 1: Path analysis for the given $k=2$, $d=3$ fat tree diagram.

Pair	Paths	Len.	Explanation
M1 \rightarrow M3	2	4	Different edge switches, same pod. Up through 1 of 2 level-1 switches, down through the destination edge switch: 2 paths.
M1 \rightarrow M5	4	6	Different pods. Must traverse root: up 3 levels, down 3 levels. At the root level $k=2$ choices, giving $2^2=4$ paths.
M1 \rightarrow M5 via S2	1	6	S2 is one of 4 root switches. Exactly 1 of the 4 paths passes through S2. By product principle: $\text{count}(M1 \rightarrow S2) \times \text{count}(S2 \rightarrow M5) = 1 \times 1 = 1$.

4 Extension to Other Topologies (Part d)

We repeat parts (a)–(c) for three production-inspired topologies and a scaled fat tree ($k=4$).

Table 2: Topology parameters for the extended analysis.

	Fat Tree ($k=4$)	Jupiter	Amazon	Meta
Hosts / Switches / Links	64 / 48 / 192	32 / 28 / 96	16 / 12 / 48	32 / 28 / 96
Root switches	16 (level-0)	4 (OCS)	4 (spine)	4 (spine)
Oversubscription	1:1	1:1	0.50:1	1:1

4.1 Fat Tree ($k=4$, $d=3$)

(a) 1 path to each root switch (length 3); 16 root switches total. (b) M1 \rightarrow M3 (same edge switch): 1 path (len 2). M1 \rightarrow M5 (same pod, different edge): 4 paths (len 4). M1 \rightarrow M5 through S2: 0—S2 is a root switch, but same-pod traffic does not traverse the root level. (c) Bisection BW = $k^d = 4^3 = 64$ link-units.

4.2 Google Jupiter

4 aggregation blocks, 4 OCS, 2 agg + 4 ToR per block, 2 hosts/ToR (32 hosts).

(a) M1 \rightarrow S1 (OCS): 2 paths (length 3)—M1’s ToR connects to 2 agg switches, each connects to the OCS. (b) M1 \rightarrow M3 (same block, different ToR): 2 paths (len 4, via 2 agg switches). M1 \rightarrow M5 (same block, same ToR): 2 paths (len 4). M1 \rightarrow M9 (inter-block): 16 paths (len 6)— $2 \text{ agg}_{\text{src}} \times 4 \text{ OCS} \times 2 \text{ agg}_{\text{dst}} = 16$. Through S2: 4 paths. (c) Bisection BW = 16 link-units (exact balanced-partition enumeration; no closed-form—topology is not a regular fat tree).

4.3 Amazon Leaf-Spine

4 spine, 8 leaf switches, full bipartite mesh, 2 hosts/leaf (16 hosts).

(a) M1 \rightarrow S1 (spine): 1 path (length 2)—each host has a single parent leaf. (b) M1 \rightarrow M3 (different leaf): 4 paths (len 4, via 4 spines). M1 \rightarrow M2 (same leaf): 1 path (len 2). M1 \rightarrow M5 through S2: 1 path—each spine carries exactly 1/4 of traffic. (c) Bisection BW = 8 link-units (exact balanced-partition enumeration).

4.4 Meta 3-Level Clos

4 spine, 4 pods (2 fabric + 4 ToR each), 2 hosts/ToR (32 hosts).

(a) M1 \rightarrow S1 (spine): 2 paths (length 3)—2 fabric switches per pod. (b) M1 \rightarrow M3 (same pod, different ToR): 2 paths (len 4, via 2 fabric). M1 \rightarrow M9 (inter-pod): 16 paths (len 6)— $2 \text{ fabric}_{\text{src}} \times 4 \text{ spine} \times 2 \text{ fabric}_{\text{dst}} = 16$. Through S2: 4 paths. M1 \rightarrow M5 (same pod) through S2: 0 paths—intra-pod traffic stays local. (c) Bisection BW = 16 link-units (exact enumeration).

4.5 Cross-Topology Summary (Parts a–c)

5 Experiments Beyond the Assignment

We go beyond parts (a)–(c) with three experiments that reveal insights not obtainable from lecture material alone. All numbers are generated by our analysis engine (viewable live in the browser).

5.1 Experiment 1: Multi-Failure Cascade

We progressively remove 1, 2, \dots , N root/spine switches and measure path survival (fraction of original shortest-path count that remains) across all host pairs.

Insight: All topologies degrade linearly, but at different rates. The fat tree loses only $\sim 5\%$ per root removed (16 roots total), retaining 79.5% at 4 failures—the same point where production topologies collapse. Jupiter and

Table 3: Summary of parts (a)–(c) across all topologies (BFS-computed).

Metric	Fat Tree ($k=2$)	Fat Tree ($k=4$)	Jupiter	Amazon	Meta
(a) Paths to root	4×1	16×1	4×2	4×1	4×2
(b) Cross-group paths	4	4^\dagger	16	4	16
(b) Paths via 1 root	1	0^\dagger	4	1	4
(c) Bisection BW	8	64	16	8	16
Oversubscription	1:1	1:1	1:1	0.50:1	1:1
Traffic locality	No	No	Yes	No	Yes

[†]M1→M5 in fat tree $k=4$ are in the same pod, so traffic stays within the pod (4 paths, len 4) and does not traverse root switches.

Table 4: Path survival (%) under progressive root-switch failures.

Roots removed	Fat Tree ($k=4$) (of 16)	Jupiter (of 4)	Amazon (of 4)	Meta (of 4)
1	94.9	79.5	76.7	79.5
2	89.8	59.1	53.3	59.1
3	84.7	38.6	30.0	38.6
4	79.5	18.2	6.7	18.2

Meta drop to 18.2% at total root loss; same-pod pairs survive but inter-pod pairs disconnect (54/66). Amazon degrades fastest (to 6.7%) because its flat 2-tier design has no locality: nearly all cross-leaf pairs disconnect (112/120) when all spines fail.

5.2 Experiment 2: Network Diameter and Path Diversity

Table 5: All-pairs shortest-path statistics (computed by BFS over all host pairs).

	Fat Tree ($k=4$)	Jupiter	Amazon	Meta
Diameter (max distance)	6	6	4	6
Min host distance	2	2	2	2
Avg host distance	5.43	5.48	3.87	5.48
Avg paths per pair	13.00	12.81	3.80	12.81

Insight: Amazon’s 2-tier design achieves the lowest diameter (4 vs. 6) and average distance (3.87 vs. ~ 5.4), at the cost of 0.50:1 oversubscription. Jupiter and Meta have identical statistics (avgDist = 5.48, avgPaths = 12.81) despite different architectures—their 3-level structure produces equivalent path-length distributions. The fat tree closely matches Jupiter/Meta in path diversity (13.00 vs. 12.81) while Amazon has only 3.80 paths/pair, quantifying the redundancy vs. simplicity trade-off.

5.3 Experiment 3: Cost Efficiency

We define two efficiency metrics: *avg paths per switch* (path diversity per unit of switch hardware) and *bisection BW per link* (bandwidth per unit of cabling).

Insight: Jupiter and Meta achieve the highest paths-per-switch (0.457), indicating efficient path diversity per unit of switch hardware. The fat tree’s BW-per-link (0.333) is $2\times$ that of the production topologies (0.167), but it requires 48 switches vs. 28—a 71% hardware premium for that bandwidth advantage. Amazon has the fewest switches (12) but the lowest paths-per-switch (0.317) and BW-per-link (0.167), reflecting its lean 2-tier design’s limited path diversity.

6 Use of AI and Vibe Coding

This project used **Claude Code** (Anthropic’s Claude Opus 4.6) as an AI copilot:

1. **Correctness audit.** Prompt: “assess the correctness of the analysis for all supported architectures.” The AI found three bugs: (i) hardcoded host count; (ii) bisection heuristic; (iii) missing inter-pod analysis.
2. **Bug fixing & feature development.** Targeted fixes plus new analysis modules (multi-failure cascade, all-pairs statistics, cost efficiency).
3. **Prompt engineering lesson.** Specific, scoped prompts outperformed vague ones for end-to-end code tracing.

7 Discussion and Conclusion

Key findings. (1) For the given $k=2$, $d=3$ diagram: 1 path per root switch (4 roots), 2 M1→M3 paths, 4 M1→M5 paths (1 via S2), bisection BW of 8. (2) Multi-failure cascade shows the fat tree retains 79.5% survival

Table 6: Cost-efficiency metrics.

	Fat Tree ($k=4$)	Jupiter	Amazon	Meta
Switches	48	28	12	28
Links	192	96	48	96
Paths/switch	0.271	0.457	0.317	0.457
BW/link	0.333	0.167	0.167	0.167

at 4 root failures, while Jupiter/Meta drop to 18.2% and Amazon to 6.7%—locality protects intra-pod pairs but inter-group traffic disconnects. (3) Amazon’s 2-tier design has the lowest diameter (4) and avg distance (3.87) but only 3.80 paths/pair and 0.50:1 oversubscription. (4) Jupiter and Meta produce identical graph-theoretic metrics (avgDist=5.48, avgPaths=12.81) despite different architectures—the choice between them is operational (OCS reconfigurability vs. pod isolation), not topological.

Limitations. All links are treated as equal bandwidth (1 link-unit). In practice, spine links run at higher speeds and Jupiter’s OCS dynamically reconfigures bandwidth. Our exact bisection algorithm is exponential in host groups; for >20 groups the code falls back to a spatial heuristic.

The source code is in the appendix and runs live in the browser. The original visualizer is at <https://github.com/h8liu/ftree-vis>.

References

- [1] H. Liu, “Fat Tree Visualizer,” GitHub. <https://github.com/h8liu/ftree-vis>
- [2] C. Clos, “A study of non-blocking switching networks,” *Bell Syst. Tech. J.*, 32(2), 1953.
- [3] M. Al-Fares et al., “A scalable, commodity data center network architecture,” *ACM SIGCOMM*, 2008.
- [4] A. Singh et al., “Jupiter rising,” *ACM SIGCOMM*, 2015. <https://dl.acm.org/doi/10.1145/2785956.2787508>
- [5] S. Poutievski et al., “Jupiter evolving,” *ACM SIGCOMM*, 2022.
- [6] JR Rivers, S. Callaghan, “Dive deep on AWS networking,” AWS re:Invent 2022 (NET402).
- [7] A. Andreyev, “Introducing data center fabric,” Meta Eng. Blog, Nov. 2014.
- [8] A. Andreyev et al., “Reinventing Facebook’s DC network with F16,” Meta Eng. Blog, Mar. 2019.

A Key Source Code

A.1 BFS Shortest Path Counting (analysis.js)

```

1 function countShortestPaths(adj, src, dst) {
2   var dist = {}, count = {};
3   var queue = [src];
4   dist[src] = 0; count[src] = 1;
5   while (queue.length > 0) {
6     var node = queue.shift();
7     var neighbors = adj[node] || [];
8     for (var i = 0; i < neighbors.length; i++) {
9       var nb = neighbors[i];
10      if (dist[nb] === undefined) {
11        dist[nb] = dist[node] + 1;
12        count[nb] = count[node];
13        queue.push(nb);
14      } else if (dist[nb] === dist[node] + 1) {
15        count[nb] += count[node];
16      }
17    }
18  }
19  return { count: count[dst] || 0, dist: dist[dst] || -1 };
20 }

```

A.2 Exact Bisection Bandwidth (analysis.js)

```

1 // Enumerate balanced host-group partitions, assign switches greedily
2 for (var mask = 1; mask < (1 << nGroups) - 1; mask++) {
3   var leftCount = 0;
4   for (var g = 0; g < nGroups; g++)
5     if (mask & (1 << g)) leftCount += groupSizes[g];
6   if (leftCount !== halfHosts) continue;
7   // ... assign switches to minimize crossing links ...
8   // ... track minimum cut across all partitions ...
9 }

```

A.3 Fault Tolerance Analysis (analysis.js)

```
1 function analyzeFaultTolerance(graph, adj) {
2   var roots = findRootSwitches(graph);
3   var testSwitch = roots[0];
4   // Build adjacency without this switch
5   var adjReduced = {};
6   for (var id in adj) {
7     if (id === testSwitch.id) continue;
8     adjReduced[id] = adj[id].filter(function(n) {
9       return n !== testSwitch.id; });
10  }
11  // Compare path counts before/after removal
12  for (var i = 0; i < hosts.length; i++)
13    for (var j = i+1; j < hosts.length; j++) {
14      var orig = countShortestPaths(adj, h1, h2);
15      var reduced = countShortestPaths(adjReduced, h1, h2);
16      // ... measure disconnections and path reduction ...
17    }
18 }
```