Fat-Tree Architecture

Fat-Tree Networks Topology

To overcome the bottlenecks of the canonical tree topology, modern data centers use a smarter design. Instead of using a few, very powerful, and expensive switches at the core, this approach uses a large number of identical, commodity switches to build a rich, multi-path network fabric. The most common and important of these designs is the **Fat-Tree**.

The Clos Network Heritage

The DCN Fat-Tree is a modern application of a much older concept from telephony called a **Clos Network**.

A Clos network is a multi-stage switching architecture designed to provide high connectivity in a scalable way.

The Fat-Tree topology used in data centers is technically a **folded-Clos** network (also called a **leaf-spine** architecture).

- It takes a 3-stage feed-forward network and "folds" it in the middle, creating bidirectional paths between the edge and the core.

Fat-Tree Construction

A Fat-Tree network is built in a structured, pod-based manner using a single type of building block: a commodity **n-port switch**.

The construction proceeds as follows:

1. **Pods:** The network is organized into `n` **pods**.
    - A pod is the fundamental building block of the network. Each pod contains two layers of `n/2` switches each:
        - **Aggregation Switches:** `n/2` switches in the upper layer of the pod.
        - **Edge Switches:** `n/2` switches in the lower layer of the pod.
2. **Switch Connectivity within a Pod:**
    - Each of the `n/2` **Edge switches** uses half its ports (`n/2`) to connect down to servers and the other half (`n/2`) to connect up to *every one* of the `n/2` aggregation switches within that same pod.
    - Each of the `n/2` **Aggregation switches** uses half its ports (`n/2`) to connect down to *every one* of the `n/2` edge switches in the pod. The other half of its ports (`n/2`) are used to connect up to the core switches.
3. **The Core Layer:**
    - There is a core layer consisting of $(n/2)^2$ n-port switches.
    - Each core switch has `n` ports, and each port is connected to a different pod.
        - This ensures a direct link from every core switch to every pod in the network.

Fat-Tree Properties

This structured design results in the following key properties:

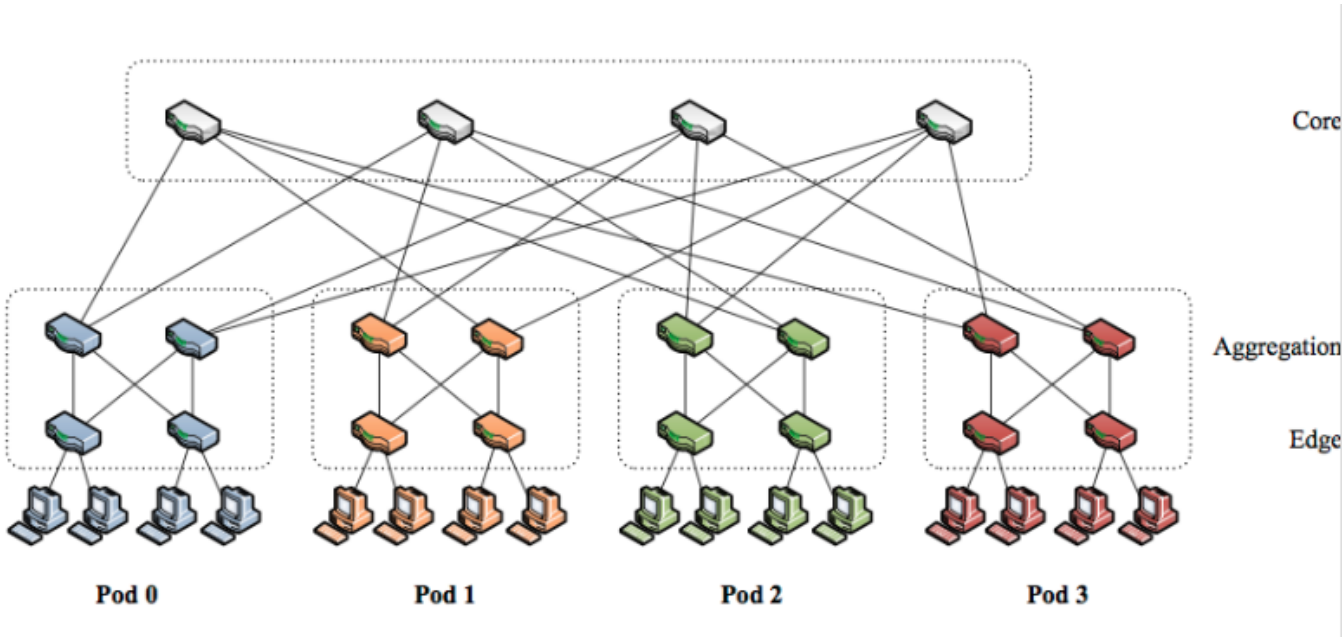
- **Total Servers:** $\frac{n^3}{4}$
- **Total Switches:** $\frac{5n^2}{4}$

Key Advantage: This topology provides a **large number of parallel, equal-cost paths** between any two servers, even if they are in different pods. This massive path diversity eliminates the bottlenecks of the traditional tree design and is ideal for handling heavy East-West traffic.

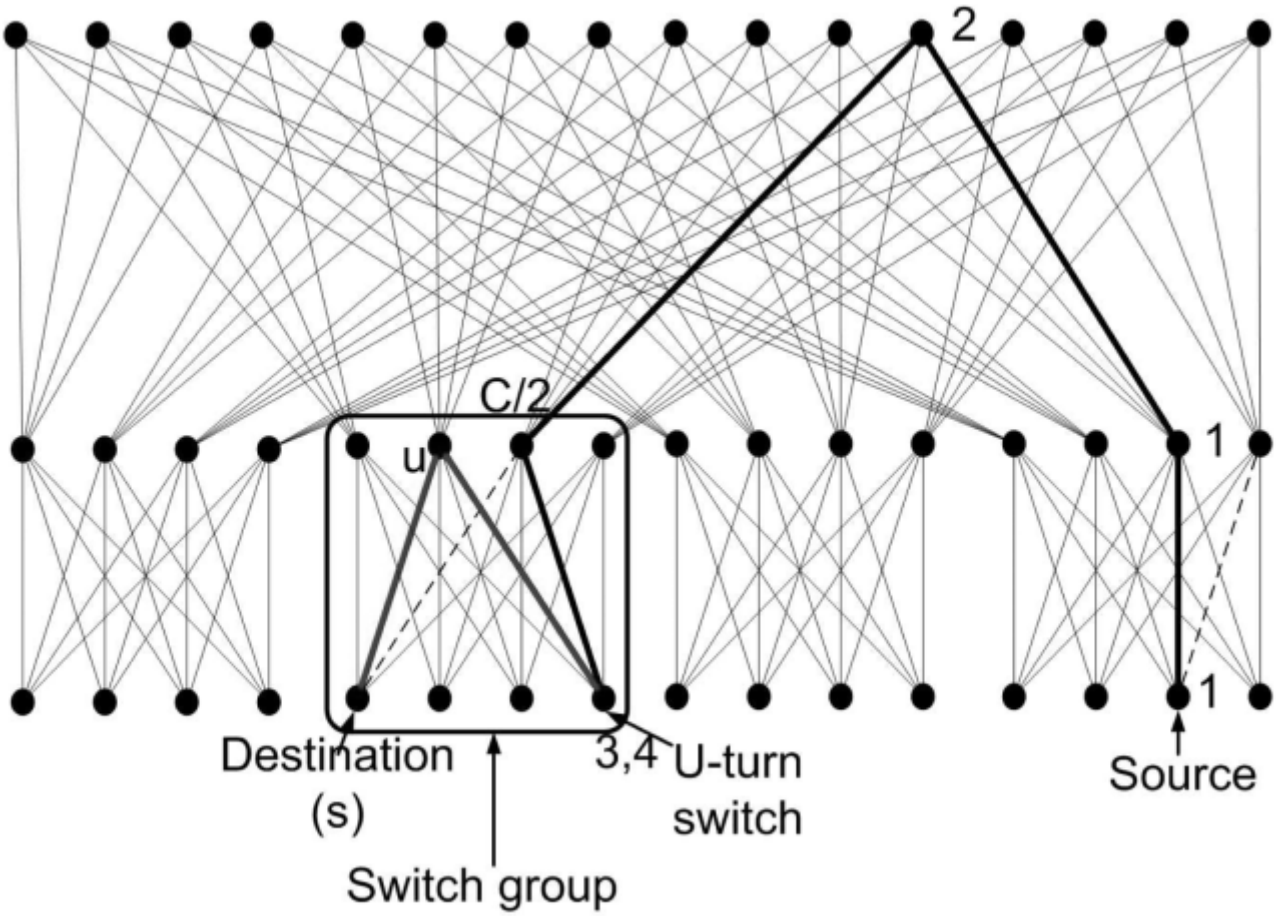
Example: A Fat-Tree with n=4 Switches

Let's use 4-port switches (`n=4`) as an example:

- **Pods:** There are `n=4` pods.
- **Switches per Pod:** Each pod has `n/2=2` edge switches and `n/2=2` aggregation switches.
- **Servers per Edge Switch:** Each edge switch connects down to `n/2=2` servers.
- **Core Switches:** There are `(n/2)² = 2² = 4` core switches.
- **Total Servers:** `4³/4 = 16` servers.
- **Total Switches:** `5 * 4² / 4 = 20` switches.



Example: A Fat-Tree with n=8 Switches



Routing in Fat-Tree Networks (skip to slides 92-93)

The structured design of the Fat-Tree topology allows for deterministic and efficient routing strategies. The **primary goal** is to **leverage the rich path diversity** to evenly **distribute traffic and avoid bottlenecks**.

Path Characteristics

The routing path depends on whether the traffic is staying within a pod or traveling between pods.

- **Intra-Pod Traffic:** If a server communicates with another server in the same pod, the traffic path is simple.
    - The packet travels from the source server up to its Edge switch, then to a common Aggregation switch within the pod, and back down to the destination Edge switch and server. The Core layer is not used.
- **Inter-Pod Traffic:** This is the more complex case. For any two servers located in different pods:
    - All traffic must traverse the **Core layer**.
    - There are exactly $(n/2)^2$ equal-cost shortest paths between them (one for each switch of the core layer).
    - Each of these paths has a fixed length of **6 hops** (in a standard 3-layer Fat-Tree):

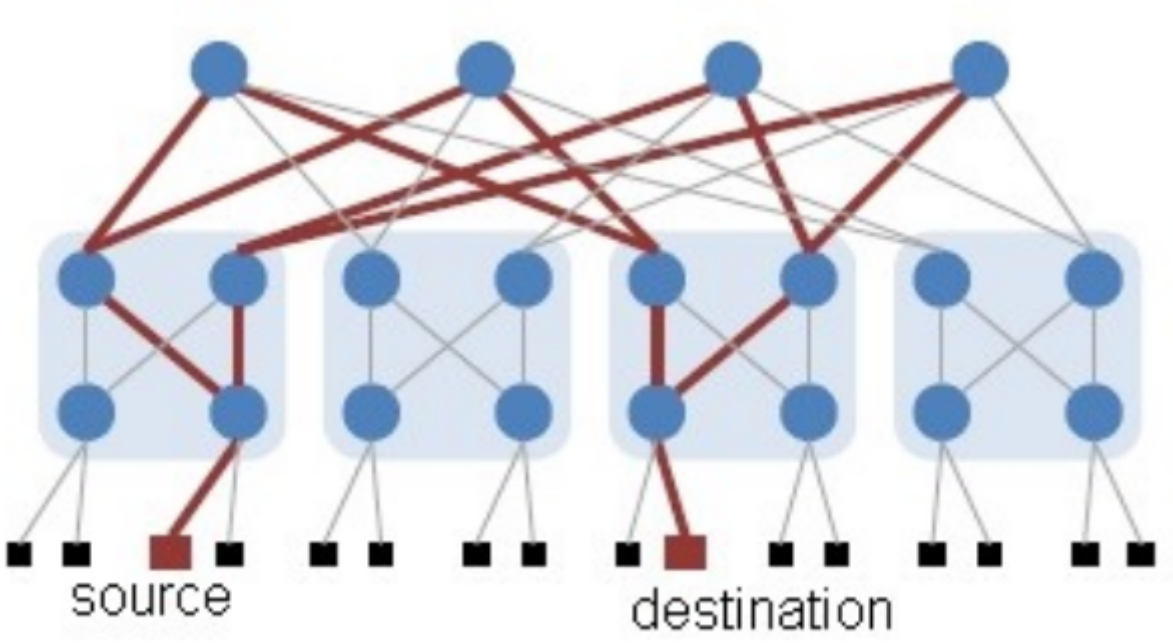
      Upstream Path (Source to Core)

        - **Hop 1:** Source Server → Edge Switch

- **Hop 2:** Edge Switch → Aggregation Switch
- **Hop 3:** Aggregation Switch → **Core Switch (Turning Point)**

Downstream Path (Core to Destination)

- **Hop 4: Core Switch** → Aggregation Switch
- **Hop 5:** Aggregation Switch → Edge Switch
- **Hop 6:** Edge Switch → Destination Server



The Routing Goal and a Proposed Mechanism

The main objective of the routing algorithm is to **distribute traffic evenly across all available core switches**.

- This prevents any single core switch from becoming a **bottleneck** and ensures that the network's **full bisection bandwidth** can be utilized.

While ECMP is a general way to achieve this, one specific mechanism proposed for Fat-Trees uses custom **two-level routing tables** in the switches:

- **Primary Table:** Works like a standard routing table, matching the **prefix** of a destination address to determine if the traffic is local to the pod or needs to go to the core.
- **Secondary Table:** For inter-pod traffic, the primary table points to a secondary table.
  - This table uses the **suffix** of the destination address to deterministically select one of the $n/2$ uplinks to the core.
  - By using different suffixes to select different core switches, traffic is effectively spread across the entire core layer.

Fat Tree Case Study: The Facebook "4-Post" Data Center Network

This is a practical example of a large-scale data center network design. It shows how the general concepts of layered topologies are implemented with specific hardware choices, oversubscription ratios, and redundancy mechanisms.

The architecture is built around three main types of switches:

- **Rack Switch (RSW):** This is the Top-of-Rack (ToR) switch that connects directly to up to 48 servers in a rack via 10G links.
- **Cluster Switch (CSW):** This is an aggregation-layer switch. A group of RSWs are connected to a group of CSWs to form a "cluster".
- **Fatcat Switch (FC):** This is the core-layer switch that interconnects all the different clusters. The design uses four of these core switches, which is where the "4-post" name comes from.

Connectivity and Oversubscription

The links between these layers are designed with specific oversubscription ratios to balance cost and performance:

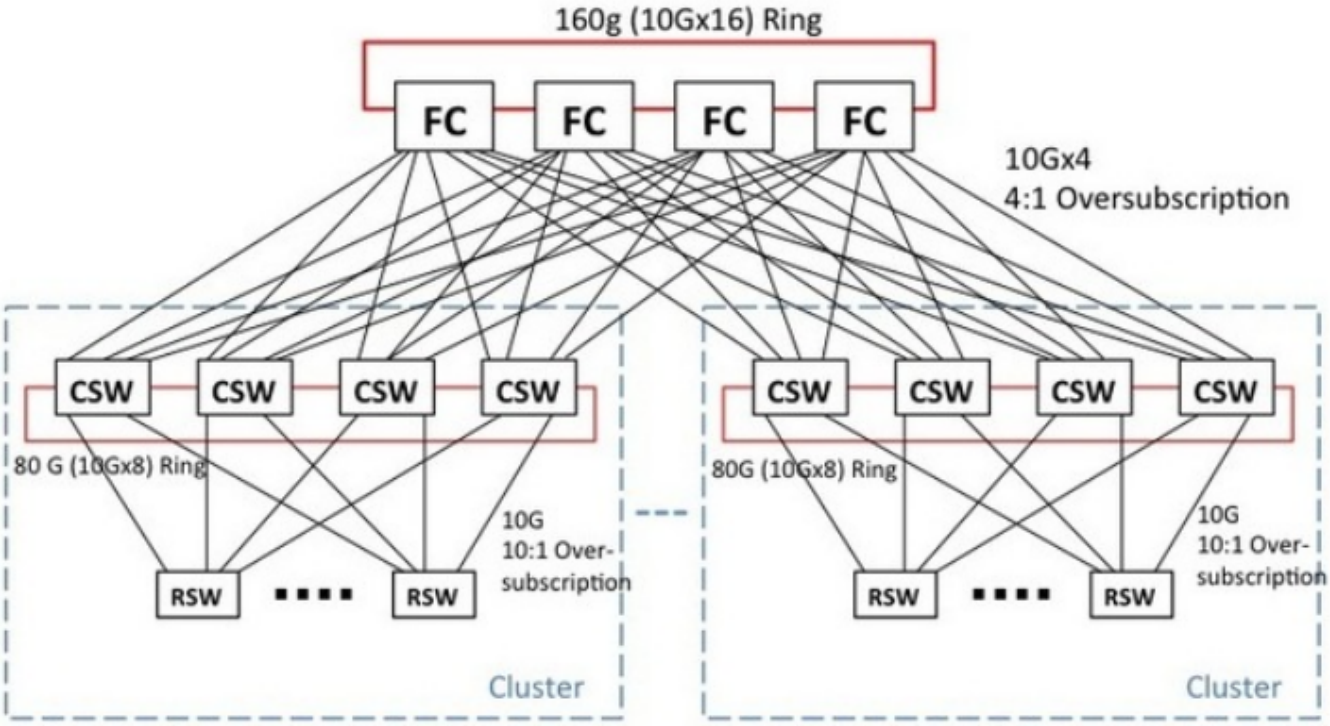
- **From Rack to Cluster (RSW → CSW):** Each RSW has 4 to 8 10G uplinks to the CSWs in its cluster. This results in a **10:1 oversubscription ratio**, meaning the total bandwidth of the servers is 10 times greater than the uplink capacity of their rack.
- **From Cluster to Core (CSW → FC):** Each CSW has four 40G uplinks, with one link going to each of the four core Fatcat switches. This layer has a lower **4:1 oversubscription ratio**.

Redundancy and Protection

To ensure high availability, the design uses "protection rings" for redundancy at the higher layers:

- **Cluster Ring:** Within a single cluster, the CSW switches are connected to each other in a high-capacity ring (e.g., 10G x 8).
- **Core Ring:** The four core Fatcat (FC) switches are also connected to each other in a larger, higher-capacity ring (e.g., 10G x 16).

This ring structure at both the aggregation (CSW) and core (FC) layers provides alternative paths for traffic if a switch or link fails.


160g (10Gx16) Ring
FC
FC
FC
FC
10Gx4
4:1 Oversubscription
CSW
CSW
CSW
CSW
80 G (10Gx8) Ring
10G
10:1 Over-
subscription
RSW
RSW
Cluster
CSW
CSW
CSW
CSW
80G (10Gx8) Ring
10G
10:1 Over-
subscription
RSW
RSW
Cluster