Network Computing - Notes - v0.2.0

260236

March 2025

Preface

Every theory section in these notes has been taken from the sources:

- Course slides. [1]

About:

GitHub repository

These notes are an unofficial resource and shouldn't replace the course material or any other book on network computing. It is not made for commercial purposes. I've made the following notes to help me improve my knowledge and maybe it can be helpful for everyone.

As I have highlighted, a student should choose the teacher's material or a book on the topic. These notes can only be a helpful material.

Contents

1 Datacenters

1.1 What is a Datacenter?

A **Datacenter** is a specialized **facility that houses multiple computing resources**, including servers, networking equipment, and storage systems. These **resources are co-located** (placed together in the same physical location) to ensure efficient operations, leverage shared environmental controls (such as cooling and power), and maintain physical security.

So the main characteristics are:

- **Centralized Infrastructure**: Unlike traditional computing models where resources are scattered, datacenters consolidate thousands to millions of machines in a single administrative domain.
- **Full Control over Network and Endpoints**: Datacenters operate under a single administrative entity, allowing customized configurations beyond conventional network standards.
- **Traffic Management**: Unlike the open Internet, datacenter traffic is highly structured, and the organization can define routing, congestion control, and network security policies.

Feature	Datacenter Networks	Traditional Networks
Ownership	Fully controlled by a single organization	Usually spans multiple independent ISPs
Traffic	High-speed internal communication (east-west traffic)	Lower-speed, external client-based traffic (north-south)
Routing	Customizable (non standard protocols)	Uses standard internet protocols (BGP, OSPF, etc.)
Latency	Optimized for ultra-low latency	Variable latency, dependent on ISPs
Redundancy	High redundancy to ensure failover and fault tolerance	Often limited by ISP policies

Table 1: Difference between Datacenters and other networks (e.g., LANs).

Why are datacenters important?

Datacenters are the backbone of modern cloud computing, large-scale data processing, and AI/ML workloads. They provide **high computational power and storage** for various applications, such as:

1. **Web Search & Content Delivery**. For example, when a user searches for "Albert Einstein" on Google, the request is processed in a datacenter where:

   (a) The query is parsed and sent to multiple servers.
   (b) Indexed data is retrieved.
   (c) A ranked list of results is generated and sent back to the user.

2. **Cloud Computing**. Services like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud offer computation, storage, and networking resources on-demand.
   - Infrastructure as a Service (IaaS): Virtual machines, storage, and networking.
   - Platform as a Service (PaaS): Databases, development tools, AI models.
   - Software as a Service (SaaS): Google Drive, Microsoft Office 365.
3. **AI and Big Data Processing**. Large-scale computations like MapReduce and deep learning training rely on distributed datacenter resources.
4. **Enterprise Applications**. Datacenters host internal IT infrastructure for businesses, including databases, ERP systems, and virtual desktops.

Evolution of Datacenters

While the concept of centralized computing dates back to the 1960s, the modern datacenter model emerged with cloud computing in the 2000s. Notable developments include:

- 1970s: IBM mainframes operated in controlled environments similar to early datacenters.
- 1990s: Rise of client-server computing required dedicated server rooms.
- 2000s-Present: Hyperscale datacenters by Google, Microsoft, and Amazon revolutionized networking, storage, and scalability.

What's new in Datacenters?

Datacenters have been around for decades, but modern datacenters have undergone significant changes in scale, architecture, and service models. The primary **factors driving these changes** include:

- ✔ The exponential **growth of internet services** (Google, Facebook, Amazon, etc.).
- ✔ The **shift to cloud computing** and on-demand services.
- ✔ The need for **better network scalability, fault tolerance, and efficiency**.

One of the most striking changes in modern data centers is their massive scale:

- Companies like Google, Microsoft, Amazon, and Facebook operate **datacenters with over a million servers at a single site**.
- **Microsoft alone has more than 100,000 switches and routers** in some of its datacenters.
- **Google processes billions of queries per day**, requiring vast computational resources.
- **Facebook and Instagram serve billions of active users**, with every interaction generating requests to datacenters.

Another major change is the **shift from owning dedicated computing infrastructure** to **renting scalable cloud resources**. Datacenters no longer just host enterprise applications, **they now offer computing, storage, and network infrastructure as a service**. The most common cloud computing models are:

- **Infrastructure as a Service (IaaS)**. User rent virtual machines (VMs), storage, and networking instead of maintaining their own physical servers (e.g., Amazon EC2).
- **Platform as a Service (PaaS)**. Provides a platform with pre-configured environments for software development (databases, frameworks, etc.).
- **Software as a Service (SaaS)**. Full software applications hosted in datacenters and delivered via the internet (e.g., Google Drive).

The move to cloud computing has fundamentally changed datacenters, shifting the focus to resource allocation, security, and performance guarantees. They are also moving from multi-tenancy to single-tenancy:

- **Single-Tenancy**. A client gets **dedicated infrastructure** for their services.
- **Multi-Tenancy**. **Resources are shared among multiple clients while ensuring isolation**.

✖ **Implications**. But this massive scale brings new challenges:

- **Scalability**: The need for **efficient network designs** to handle rapid growth.

  Traditional datacenter topologies, such as three-based architectures, are inefficient at scale. New designs, like **Clos-based networks (Fat Tree)** and **Jellyfish (random graphs)**, are being developed to:

  ✔ Ensure **high bisection bandwidth** (allow any-to-any communication efficiently).

  ✔ Provide **scalable and fault-tolerant networking**.

- **Cost management**: More machines mean **higher power, cooling, and hardware costs**.

  Datacenters are **expensive to build and maintain**, requiring:

  - **Efficient resource utilization** (prevent idle servers from wasting power).
  - **Energy-efficient cooling solutions** (cooling accounts for a *huge* portion of operational costs).
  - **Automation to reduce human intervention** (e.g., AI-based network optimization).

- **Reliability**: Hardware failures become **common at scale**, requiring **automated fault-tolerant solutions**.

  At the scale of modern datacenters, **hardware and software failures are common**. A key principle is: "*In large-scale systems, failures are the norm rather than the exception.*" (Microsoft, ACM SIGCOMM 2015).

  Thus, new **automated failover mechanisms** are required to:

  - Detect failures **quickly**.
  - Redirect traffic **seamlessly**.
  - Ensure **minimal service disruption**.

- **Performance & Isolation Guarantees**: In modern datacenters, **customers expect strict performance guarantees** for applications like: low-latency financial transactions, high-bandwidth video streaming, machine learning model training.

  To meet these demands, datacenters implement:

  ✔ **Performance Guarantees**: Allocating bandwidth and compute power dynamically.

  ✔ **Isolation Guarantees**: Ensuring one user's workload does not interfere with another's.

  But this requires **advanced networking techniques**, such as:

  - **Traffic engineering** to avoid congestion.
  - **Load balancing** to distribute workloads efficiently.
  - **Software-defined networking (SDN)** for centralized control over traffic flows.

Key Takeaways: What is a Datacenter?

- **Datacenters centralize** computing resources for performance, security, and scalability.
- **They differ from traditional networks** by offering more control, lower latency, and higher redundancy.
- **Applications include cloud services, AI, and enterprise computing**.
- **Scalability is a key challenge**, with hyperscale datacenters hosting millions of machines.
- **Efficiency and cost containment are major concerns**, requiring innovative architectures.

1.2 Datacenter Applications

Modern datacenters host a variety of applications that range from web services to large-scale data processing. These **applications can be classified based on their traffic patterns and computational needs**.

Customer-Facing Applications (North-South Traffic)

Customer-facing applications involve direct interaction with users. This type of traffic follows a **North-South communication model**, meaning that **data flows between external users and the datacenter**.

Example 1: North-South Traffic

Examples include:

- **Web Search** (e.g., Google, Bing)
  - A user submits a query (e.g., "Albert Einstein").
  - The request is routed through the datacenter's frontend servers.
  - Backend database and indexing servers fetch relevant results.
  - The response is assembled and sent back to the user.
- **Social Media Platforms** (e.g., Facebook, Instagram, X (ex Twitter))
  - Users interact with content hosted in the datacenter (e.g., loading a feed, liking posts).
  - Each interaction requires queries to databases and caching systems.
  - Content delivery is optimized using load balancers.
- **Cloud Services** (e.g., Google Drive, Dropbox, OneDrive)
  - Users upload, store, and retrieve files.
  - Requests must be efficiently distributed across storage nodes.

Large-Scale Computation (East-West Traffic)

Unlike customer-facing applications, backend computations do not involve direct interaction with external users. Instead, they focus on **processing massive datasets within the datacenter**. This type of traffic is known as **East-West traffic** because it occurs **between servers inside the datacenter** *rather than between the datacenter and the external world.*

Example 2: East-West Traffic

Examples include:

- **Big Data Processing** (e.g., MapReduce, Hadoop, Spark)
  - Large datasets are distributed across multiple servers.
  - Each server processes a portion of the data in parallel.
  - Results are combined to generate insights (e.g., web indexing, analytics).
- **Machine Learning & AI Training** (e.g., Deep Learning Models)
  - AI models are trained on massive datasets using clusters of GPUs/TPUs.
  - The process requires high-bandwidth, low-latency communication.
  - Synchronization between nodes is critical (e.g., gradient updates in distributed training).
- **Distributed Storage & Backup Systems** (e.g., Google File System, Amazon S3)
  - Data is replicated across multiple locations for reliability.
  - Servers frequently exchange data to ensure consistency and fault tolerance.

⚖ Key differences between North-South and East-West traffic

Feature	N-S traffic	E-W traffic
Direction	External users ↔ Datacenter	Within datacenter
Examples	File downloads	AI training
Bandwidth Needs	Moderate	Very High
Latency Sensitivity	High	Critical
Traffic Type	Query-response	Bulk data transfer

Table 2: Differences between North-South and East-West traffic.

In terms of latency sensitivity, North-South traffic is high because user interactions must be fast. On the other hand, East-West traffic is critical because synchronization delays affect computation.

Traffic Patterns and Their Impact on Networking

The way data moves within a datacenter **heavily influences network design**. The main **goal** is to **ensure high bandwidth, low latency, and efficient resource utilization**.

- **Any-to-Any Communication Model**
  - In **large-scale distributed applications**, **any server should be able to communicate with any other server at full bandwidth**.
  - **Network congestion can severely degrade performance**, especially for AI/ML workloads and big data processing.
- **High-Bandwidth Requirements**
  - Applications like **MapReduce** and **deep learning** require **high data transfer rates**.
  - If bandwidth is insufficient, **bottlenecks occur**, leading to delays.
- **Latency is a Critical Factor**
  - **Low-latency networking is essential** for interactive applications and distributed computing.
  - AI training, for example, requires nodes to synchronize frequently; a delay in one node slows down the entire process.
- **Worst-Case (Tail) Latency Matters**
  - It's not enough for **most requests** to be fast; **the slowest request** can delay the entire computation.
  - **Minimizing tail latency** is crucial for efficient AI model training and database queries.

Challenges in Datacenter Traffic Management

The massive scale and complexity of modern datacenters introduce **several networking challenges**, including:

- **Network Congestion and Bottlenecks**. When multiple servers communicate simultaneously, **some network links become overloaded**, leading to congestion.

  For example, if many AI training jobs share the same network path, it can become a bottleneck, slowing down training.

  This can be a **critical issue for applications requiring real-time performance** (e.g., financial transactions, cloud gaming).
- **Load Balancing and Traffic Engineering**. How do we distribute traffic *efficiently* across network links? The solutions are: **Equal-Cost Multi-path Routing** (ECMP, spreads traffic across multiple paths); **Dynamic Traffic Engineering** (adjusts paths in real time based on congestion levels).

- **Avoiding Link Over-Subscription**. If too many servers send data over a single link, the available **bandwidth is divided**, leading to **slow performance**. Modern datacenters aim for **full-bisection bandwidth**, meaning **any server can talk to any other server at full capacity**.
- **Scaling Challenges**. Traditional datacenter network architectures do not scale well beyond a certain point. **New network topologies** (e.g., Fat Tree, Jellyfish) are being adopted to address these limitations.

Key Takeaways: Datacenter Applications

- Datacenters handle **two major types of applications**:
  1. **Customer-facing applications (North-South traffic)** involve external users.
  2. **Large-scale computations (East-West traffic)** occur within the datacenter.
- **Traffic patterns** affect **bandwidth, latency, and congestion control**.
- **Managing congestion and ensuring high bandwidth** is critical for performance.
- **New network topologies and routing techniques** help address scaling challenges.

1.3 Network Architecture

The **primary goal** of a datacenter network is to **interconnect thousands to millions of servers** efficiently. Unlike traditional networks, which focus on wide-area communication, datacenter networks emphasize:

- **High throughput**: Supporting massive data transfers.
- **Low latency**: Ensuring real-time performance for applications.
- **Scalability**: Accommodating rapid growth without performance degradation.
- **Fault tolerance**: Handling hardware failures with minimal disruption.

Datacenter **networks physically and logically connect servers through a multi-tiered architecture**. This hierarchical structure ensures that servers in different racks, pods, or clusters can communicate efficiently.

Traditional Three-Tier Datacenter Network

Most datacenter networks follow a **Three-Tier design**, which is optimized for scalability and efficiency. The three tiers are:

- **Edge Layer (Access Layer)**
  - Located at the **bottom of the hierarchy**, closest to the servers.
  - Consists of **Top-of-Rack (ToR) switches** that connect servers within a rack.
  - ✔ **Purpose**: Aggregates traffic from multiple servers and forwards it to the higher layers.
  - Typically uses **high-speed links (10-100 Gbps per port)** to connect servers.
- **Aggregation Layer (Distribution Layer)**
  - Intermediate layer **between the edge and core layers**.
  - **Connects multiple ToR switches** within a datacenter pod.
  - ✔ **Purpose**: Helps distribute traffic efficiently **without overwhelming core routers**.
  - Implements **load balancing, redundancy, and failover mechanisms**.
- **Core Layer (Backbone Layer)**
  - The **top layer** of the hierarchy.
  - Composed of **high-capacity, high-speed switches and routers**.
  - ✔ **Purpose**: Responsible for:
    - **Routing large volumes of traffic** between different aggregation switches.

    - **Connecting the datacenter to external networks** (e.g., the Internet or private backbones).
  - Core switches often run **at 100 Gbps or higher per port** to support high aggregate bandwidth.

Key **characteristics of the Three-Tier model**:

- **Position**:
  - **Edge Layer**: Closest to servers.
  - **Aggregation Layer**: Intermediate between edge and core.
  - **Core Layer**: Backbone layer.
- **Primary Function**:
  - **Edge Layer**: Connects servers within racks.
  - **Aggregation Layer**: Aggregates ToR traffic.
  - **Core Layer**: Routes traffic between datacenters or externally.
- **Switch Type**:
  - **Edge Layer**: Top-of-Rack (ToR).
  - **Aggregation Layer**: Aggregation switches.
  - **Core Layer**: Core routers.
- **Speed (per port)**:
  - **Edge Layer**: 10-100 Gbps.
  - **Aggregation Layer**: 40-100 Gbps.
  - **Core Layer**: 100 and more Gbps.
- **Fault Tolerance**:
  - **Edge Layer**: Redundant paths to aggregation layer.
  - **Aggregation Layer**: Load balancing across core switches.
  - **Core Layer**: High redundancy & backup links.

⚠ Limitations of the Traditional Three-Based Model

Although widely used, the traditional three-tier model faces **scalability and performance challenges** as datacenters grow.

- **Scalability Issues**. **Traditional networks are hierarchical**, meaning most communication **must pass through the core layer**. As datacenters scale, **core switches become bottlenecks** due to increased traffic.
- **Bandwidth Bottlenecks**. The model assumes that the **most traffic is North-South** (client to server). However, modern workloads involve **high East-West traffic** (server-to-server communication).

  Over-subscription occurs when the network cannot handle full-bisection bandwidth.

- **Over-Subscription Problem**. **Over-Subscription** refers to **the ratio of worst-case achievable bandwidth to total bisection bandwidth**. For example:
  - If 40 servers per rack each have a 10 Gbps link, total demand is 400 Gbps.
  - If the uplink capacity to the aggregation layer is only 80 Gbps, we have a 5:1 over-subscription.
  - This means only 20% of the potential bandwidth is available, causing congestion.

  Over-subscription ratios in large-scale networks can reach 50:1 or even 500:1, **severely limiting performance**.
- **Performance Issues in High-Density Environments**. High latency when traffic must **traverse multiple hops** to reach other racks. **Failures in core routers** can impact a large number of servers. **Inconsistent network performance** due to congestion in aggregation switches.

Modern Datacenter Network Designs

To overcome the **scalability and congestion challenges** of traditional three-based networks, modern datacenters use alternative architectures.

- ✔ **Fat Tree (Clos Network)**. Fat Tree is a **multi-stage switching architecture** designed to:
  - **Ensure full-bisection bandwidth**: Every server can communicate at full capacity.
  - **Provide multiple paths** between any two servers (high redundancy).
  - **Balance traffic dynamically** to avoid congestion.

  It uses K-ary fat tree topology where each pod consists of aggregation and edge switches, and core switches connect multiple pods. The advantages are:
  - **Scalability**: Expands easily by adding more pods.
  - **Fault Tolerance**: Multiple paths prevent failures from disrupting traffic.
  - **Better Load Balancing**: Traffic is evenly distributed.
- ✔ **Jellyfish: Random Graph-Based Topology**. Instead of a strict hierarchical structure, Jellyfish uses a **randomized topology**. The advantages are:
  - **Higher network capacity** with **lower cost**.
  - **More flexible scaling** than Fat Tree.
  - **Better fault tolerance** since the network adapts dynamically.

✔ **BCube: Datacenter Network for Cloud Computing**. Designed for high-performance cloud computing environments. It is optimized for: multi-path communication, resilience against failures and lowe latency compared to hierarchical models.

Key Takeaways: Network Architecture

- Traditional **three-tier datacenter networks** include **Edge, Aggregation, and Core layers**.
- **Core switches bottlenecks** as datacenters scale.
- **Over-subscription limits bandwidth**, causing congestion.
- **Modern topologies like Fat Tree and Jellyfish improve scalability, fault tolerance, and load balancing**.

1.4 High and Full-Bisection Bandwidth

Why is High-Bandwidth important in Datacenters?

Modern datacenters **handle massive amounts of data** due to applications like AI training, cloud services, and big data processing. These workloads *require*:

- **High-bandwidth connections** to support fast data transfers.
- **Low latency** to ensure real-time performance.
- **Scalability** to accommodate increasing workloads.

Unlike traditional networks, where traffic primarily flows between users and servers (North-South), **datacenters experience heavy East-West traffic** (server-to-server communication). This shift **demands high-bandwidth and scalable network designs**.

One step at a time: What a Bisection Bandwidth is and why Full-Bisection Bandwidth is important

Bisection Bandwidth is a key metric that measures the **total bandwidth available between two halves of a network**.

Definition 1: Bisection Bandwidth

If a network is split into two equal halves, the **Bisection Bandwidth** is the **total data transfer rate available between them**.

Definition 2: Full-Bisection Bandwidth

The **Full-Bisection Bandwidth** is when every server can communicate with every other server at **full network speed**.

In other words, bisection bandwidth can be thought of as cutting a data center network in half and measuring the total capacity of the links connecting the two halves. This tells us how much data can flow between the two sections simultaneously.

Example 3: Understand what bisection bandwidth is

Imagine a 1000-server datacenter, where 500 servers are processing data while 500 servers store the results. If the bisection bandwidth is **low**, the **data transfer between processing and storage nodes will be delayed**. This results in slow machine learning model training or delayed database queries.

As we can imagine, the full-bisection bandwidth is a real and critical aspect:

- **Prevents bottlenecks**: Ensures high-throughput communication across racks and clusters.
- **Essential for AI/ML training**: AI models require massive parallel computations with continuous data exchanges.

- **Optimized for cloud computing**: Services like AWS, Google Cloud, and Azure depend on fast, reliable inter-server communication.

⚠ Then try to get high-bandwidth all the time! Yes, but there are some challenges...

Ideally, high-bandwidth should be the ultimate goal, but unfortunately, there are some problems with traditional three-based networks:

✖ **The Problem with Traditional Three-Based Networks**. The standard **three-tier (core-aggregation-edge) topology** struggles to scale due to:

1. **Over-subscription** (definition on page 15): The ratio of available bandwidth to required bandwidth is too high.
2. **Core congestion**: Core routers become bottlenecks as traffic grows.
3. **Single points of failure**: A failure in a core switch can affect a large portion of the datacenter.

✖ **Over-Subscription and Its Impact on Network Performance**. A naive solution would be to use over-subscription to solve these problems, but this limits performance. **Over-Subscription** happens when the **network is provisioned with less bandwidth than needed** to cut costs.

$$\text{Over-subscription} = \frac{\text{Total server bandwidth demand}}{\text{Available bandwidth at aggregation/core layer}}$$

Common over-subscription ratios are:

- 5:1, only 20% of host bandwidth is available.
- 50:1, only 2% of host bandwidth is available.
- 500:1, only 0.5% of host bandwidth is available.

At 500:1 over subscription, congestion becomes severe, **limiting network efficiency**.

✖ **The cost problem: scaling is expensive!**

- Increasing bisection bandwidth requires **more high-performance network hardware**.
- **Scaling traditional networks** (adding more core switches) is extremely costly.
- **Energy consumption rises** with additional hardware.

Thus, **alternative solutions** are needed to achieve high-bandwidth networking **without excessive costs**.

Solutions to Achieve High and Full-Bisection Bandwidth

To overcome these challenges, researchers and engineers have designed **new network architectures**.

- **Fat Tree (Clos Network) - The Scalable Solution**. Unlike traditional three-based designs, Fat Tree provides **multiple paths** for traffic.
  - **Advantages**
    - **Ensure full-bisection bandwidth** by allowing traffic to take alternative routes.
    - **Eliminates single points of failure** using redundant paths.
    - **Load balancing** optimizes network utilization.
- **Jellyfish - A More Flexible Approach**. Uses a **randomized, non-hierarchical** topology instead of a fixed three structure.
  - **Advantages**
    - **Better bandwidth scaling** as new servers are added.
    - **More resilient to failures** (no single critical point of failure).
- **BCube - Optimized for Cloud Services**. Designed for high-performance cloud environments with **massive inter-server communication**.
  - **Advantages**
    - **Fast re-routing** in case of failures.
    - **Low-latency communication for cloud applications**.

Key Takeaways: High and Full-Bisection Bandwidth

- **High-bandwidth networking** is essential for modern datacenters.
- **Full-bisection bandwidth** ensures servers communicate at **full speed**.
- **Over-subscription** creates **bottlenecks**, limiting performance.
- **New network architectures** (Fat Tree, Jellyfish, BCube) solve scalability issues.

1.5 Fat-Tree Network Architecture

A **Fat-Tree** is a **multi-layer, hierarchical network topology** that provides *high scalability*, *full-bisection bandwidth*, and *fault tolerance*. It is a **special type of Clos Network**[1], designed to **overcome bandwidth bottlenecks** in traditional three-based networks.

The key idea is: Instead of a traditional tree where higher levels become bottlenecks, Fat-Tree ensures equal bandwidth at every layer by **increasing the number of links as we move higher in the hierarchy**.

Structure of a K-Ary Fat-Tree

A **K-ary Fat-Tree** consists of **three layers**:

1. **Edge Layer (Top-of-Rack, ToR switches)**:
   - Connects directly to the servers.
   - Each edge switch connects $\frac{k}{2}$ servers and $\frac{k}{2}$ aggregation switches.
2. **Aggregation Layer**
   - Connects multiple edge switches.
   - Ensures **local traffic routing** between racks before sending to the core.
   - Each aggregation switch connects $\frac{k}{2}$ edge switches and $\frac{k}{2}$ core switches.
3. **Core Layer**
   - The backbone of the Fat-Tree, interconnecting multiple aggregation layers.
   - Consists of $\left(\frac{k}{2}\right)^2$ core switches, where each connects to k pods.

Example 4: Fat-Tree with $k = 4$

- Each pod contains:
  - $\left(\frac{4}{2}\right)^2 = 4$ servers.
  - 2 layers of 2 2-port switches (Edge and Aggregation).
- Each Edge Switch connects 2 servers and 2 aggregation switches.
- Each Aggregation Switch connects 2 Edge switches and 2 Core switches.
- The Core Layer consists of $\left(\frac{k}{2}\right)^2 = 4$ core switches.

[1] A **Clos Network** is a type of multistage **switching topology that enables high-bandwidth and fault-tolerant communication by interconnecting multiple small switches instead of relying on a few large ones**. It is commonly used in datacenter networks (e.g., Google Jupiter Fabric) to maximize scalability and minimize congestion.

As a result, multiple paths between servers ensure no single point of failure and full-bisection bandwidth.

Why Use Fat-Tree in Datacenters?

Cost-Effective Scaling

- Can be built using **cheap, commodity switches** instead of expensive core routers.
- All switches operate at **uniform capacity**, simplifying hardware requirements.

Full-Bisection Bandwidth

- Each switch and server has **equal access to bandwidth**, preventing bottlenecks.
- Every packet has **multiple available paths**, ensuring **load balancing**.

High Fault Tolerance

- If one **switch or link fails**, traffic is rerouted through **alternative paths**.
- **No single point of failure**, unlike traditional three-based architectures.

Efficient Load Balancing

- **Multipath Routing** ensures traffic is evenly distributed.
- **No congestion at higher layers**, as each pod has equal bandwidth allocation.

Problems in Fat-Tree Networks

Fat-Tree is a highly scalable and efficient network topology, but **practical challenges exist** when handling real-world workloads.

- **Many flows running simultaneously**. In large datacenters, multiple applications generate concurrent flows. Some flows are **small but latency-sensitive** (mice flows), while others are **large data transfers** (elephant flows). The Fat-Tree must **efficiently balance all these flows** across available paths.
- **Traffic locality is unpredictable**. Some services (e.g., Facebook/Meta workloads) have localized communication within a rack, while others require data exchange across the entire network. Fat-Tree must **dynamically adapt to different workload patterns**.

- **Traffic is bursty**. Some applications **generate sudden traffic spikes**, leading to temporary congestion. This is problematic for routing since **congestion-aware path selection is difficult**.
- **Too Many Paths Between a Source and Destination**. Unlike traditional network that have a single best route, Fat-Tree networks offer multiple equal-cost paths. *Which path should be used?* Random selection might lead to congestion.
- **Random Path Selection Leads to Collisions**. If routing randomly assigns traffic flows, two large elephant flows may end up on the same link. This creates a congestion hotspot, even though other links remain underutilized.
  - **Ideal case**: Traffic should be spread evenly across all available links.
  - **Reality**: Without congestion awareness, routing **cannot react to traffic conditions dynamically**.
- **Short-Lived vs. Long-Lived Flows Create Conflicts**. An **ideal routing scenario** would be to evenly distribute all flows. However, if a short, latency-sensitive flow suddenly appears on a congested link, its performance suffers. The key problem is that **Fat-Tree does not inherently prioritize latency-sensitive flows**.

⚠ TCP Incast: A Major Issue in Fat-Tree Datacenters

Large-scale parallel requests cause network congestion. In fact, some workloads (e.g., distributed storage systems, AI training) involve a **single client requesting data from multiple servers simultaneously**. This means that all servers respond at once, **overwhelming the switch's buffer capacity**. This results in **packet loss and retransmissions**, significantly increasing latency.

Definition 3: TCP Incast

TCP Incast is a **network congestion issue** that occurs in datacenters when multiple servers send data to a single receiver simultaneously, overwhelming the switch's buffer capacity and causing severe packet loss and performance degradation.
In other words, TCP Incast happens when many-to-one communication causes network congestion, leading to packet loss, TCP retransmissions, and increased latency.

But in this scenario, how does **TCP Incast** happen?

1. A client application requests data from multiple storage servers.
2. All storage servers respond **simultaneously**.
3. The switch **cannot handle all packets at once**, causing **buffer overflow**.

4. **Packet loss triggers TCP retransmissions**, further slowing down performance.

This involves several issues:

- Causes **severe latency spikes**, affecting (AI training and large-scale cloud) workloads.
- Traditional TCP **was not designed for this kind of bursty traffic**.
- **Fat-Tree cannot solve this issue alone**, it requires transport-layer optimizations.

Google's Approach to Solving Fat-Tree Challenges

Google faced severe scalability, congestion, and failure recovery challenges in its datacenters. Instead of using a traditional Fat-Tree model, they **developed a Clos-based architecture** known as **Google Jupiter Fabric**. The key challenges that Google is addressing are:

- **Scalability**. Traditional networks could not handle Google's exponential growth. Needed a network that scales gracefully by adding more capacity in stages.
- **Failure Tolerance**. A single failure should not impact traffic significantly. Needed path redundancy to ensure seamless operations.
- **Performance and Cost**. High-performance custom-built switching to support full-bisection bandwidth. Used commodity merchant silicon (off-to-shelf networking chips) instead of proprietary network devices, reducing costs.

The solutions adopted by Google are:

✔ **Clos Topology for Scalability & Fault Tolerance**. Google moved from traditional Fat-Tree to Clos networks to improve scalability.

- **Multiple layers of switches**, with multiple paths between every two endpoints.
- **Graceful fault recovery**: if one switch fails, traffic is rerouted dynamically.
- **Incremental scalability**: new switching stages can be added without network downtime.

A Clos network was chosen because, unlike Fat-Tree, which suffers from static oversubscription, **Clos networks offer more flexible bandwidth allocation**.

Note that Fat-Tree inherits the scalability and fault tolerance of Clos, but its hierarchical and structured nature leads to congestion, routing complexity, and TCP Incast problems. Google recognized that Fat-Tree had structural limitations, so they modified Clos into the Jupiter Fabric.

✔ **Custom Hardware: Merchant Silicon Instead of Proprietary Switches**. Google avoided vendor lock-in by using commodity hardware (merchant silicon). The reasons are:

- **Lower cost** than custom ASIC-based routers.
- **Faster** hardware **upgrade cycles**.
- **More control** over network design and software stack.

✔ **Centralized Control for Routing and Network Management**. In traditional datacenters, routing is distributed, meaning each switch makes independent routing decisions. This approach does not scale well in Clos networks with thousands of switches.

The solution is **precomputed routing decisions**. Instead of switches making their own decisions, Google precomputes traffic flows centrally and pushes them to switches.

Advantages

✔ **Improves traffic engineering**: Load balancing decisions are optimized globally rather than per switch.

✔ **More predictable performance**.

✔ **Less congestion**: Can react dynamically to network failures.

Key Takeaways: Fat-Tree Network Architecture

- **Fat-Tree is a special type of Clos Network** that overcomes bottlenecks in traditional tree networks.
- **K-ary** Fat-Tree has three layers (Edge, Aggregation, Core), **ensuring equal bandwidth for all nodes**.
- **Fat-Tree** provides multiple paths, but **routing is difficult due to unpredictable traffic patterns**.
- **Collisions between large flows create network hotspots**.
- **TCP Incast is a major issue**, where too many responses at once cause packet loss.
- Google's Datacenter Network Strategy:
  - **Moved from Fat-Tree to Clos topology** for better scalability and failure recovery.
  - **Used merchant silicon instead of proprietary hardware** to cut costs and improve flexibility.
  - **Implemented centralized control for routing** to optimize traffic flows.
  - **Designed the Jupiter Fabric** to handle **Google-scale workloads** with incremental scalability.

References

[1] Antichi Gianni. Network computing. Slides from the HPC-E master's degree course on Politecnico di Milano, 2024.

Index