The City College of New York

# CSC 36000:
# Modern Distributed Computing *with AI Agents*

By Saptarashmi Bandyopadhyay
Email: sbandyopadhyay@ccny.cuny.edu
Assistant Professor of Computer Science
City College of New York and Graduate Center at City University of New York

# Today's Lecture

Networking for Distributed Systems

Distributed Networked Communication: Persistence and Synchronicity

Network Communication in High-Performance Distributed Computing

Multi-Agent Systems in NextGen Networks

# Networking for Distributed Systems

# Introduction to Networking

A distributed computing system necessarily involves connecting separate computers in some way.

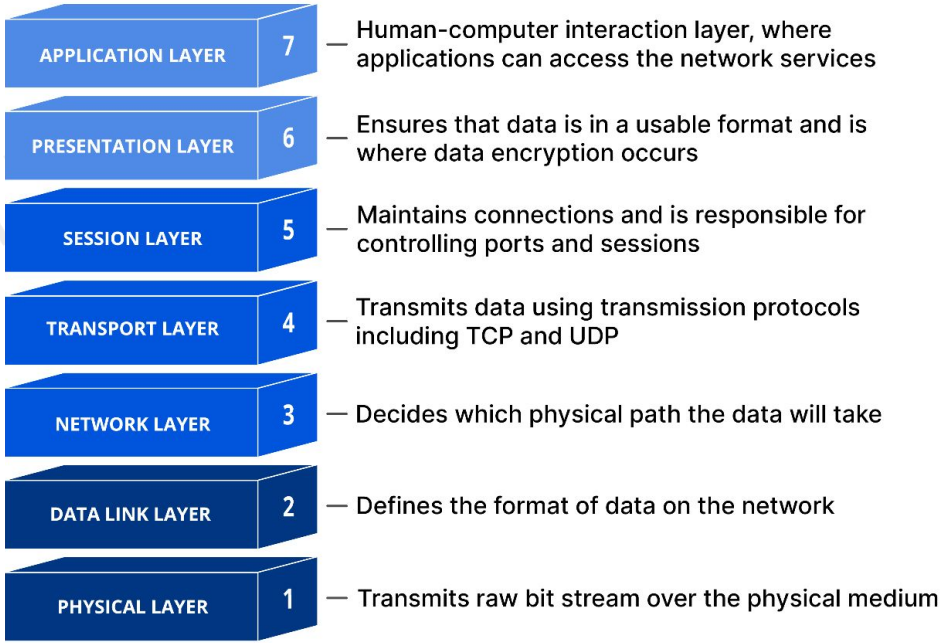
Often we have left out the particulars of how this is accomplished to focus on higher-level ideas related to practical applications, but it's also important to know the more concrete details of how these computers are connected!

This is where *networking* comes into play.

# Networking Stack (Open Systems Integration)

Networks as we know them are split into seven different layers, ranging from the *physical layer* (e.g. ethernet cables) to the *application* layer (e.g. HTTP)

For distributed systems, we'll pay special attention to the upper half from layers 4 to 7.



# OSI for Distributed Systems

**Lower Layers (Physical, Data Link, Network):** Often treated as a "black box" providing unreliable datagrams (IP) or reliable streams (TCP).

**Transport Layer:**

- Standard TCP/UDP is often too primitive or high-overhead for distributed apps.

**Middleware Layer:** Sits logically between Transport and Application.

- Protocols: Application-independent protocols for:
  - Authentication
  - Distributed Locking.
  - High-level communication

# Distributed Networked Communication: Persistence and Synchronicity

# Distributed Communication Revisited from Past Classes

We saw before that what communication we use often depends on how certain we want to be that the message was delivered (*persistence*) and the need for all nodes to be on the same page (*synchronicity*)

**1. Persistent Communication:**

- Message is stored by the middleware (e.g., mail server, message queue) until delivered.
- Sending application (like email service) does not need to be active for the message to be delivered to the receiver at any time.
- Example: Email

# Distributed Communication Revisited from Past Classes

We saw before that what communication we use often depends on how certain we want to be that the message was delivered (*persistence*) and the need for all nodes to be on the same page (*synchronicity*)

**2. Transient Communication:**

- Message is stored only as long as sending/receiving applications are executing.
- If the receiver is down, the message is discarded.
- Example: Transport-level sockets (TCP/UDP)

# Distributed Communication Revisited from Past Classes

We saw before that what communication we use often depends on how certain we want to be that the message was delivered (*persistence*) and the need for all nodes to be on the same page (*synchronicity*)

**3. Asynchronous vs. Synchronous:**

- Asynchronous: Sender continues immediately after submission.
- Synchronous: Sender blocks until request is accepted, delivered, or processed.

# Which Communication Style to Use?

Distributed Computing prioritizes *Low Latency* and *High Throughput*.

In a Distributed System, you'd probably want to prioritize **Transient Communication:**

- Computing tasks are typically active simultaneously.
- Storing messages (Persistence) is unnecessary overhead; if a node crashes, the computation often restarts.

# Which Communication Style to Use?

Distributed Computing prioritizes *Low Latency* and *High Throughput*.

In a Distributed System, you'd probably want to prioritize **Transient Communication**:

- Computing tasks are typically active simultaneously.
- Storing messages (Persistence) is unnecessary overhead; if a node crashes, the computation often restarts.

… As well as **Asynchronous Communication**:
Crucial for parallelism. A compute node must not block waiting for data delivery if it has other work to do.

# Network Communication in High-Performance Distributed Computing

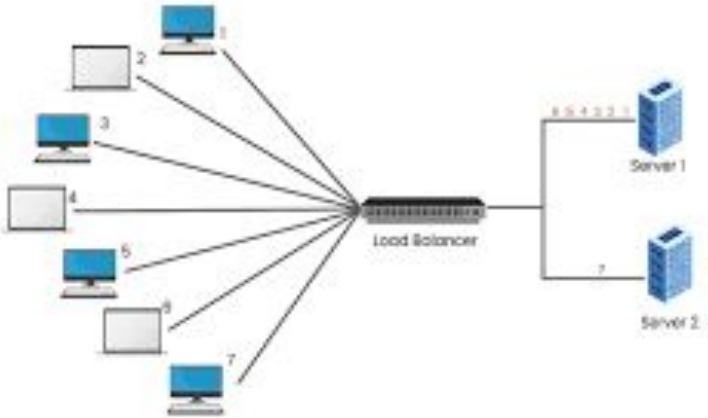
# Cluster Computing

**Definition:** A collection of similar computing nodes, closely connected by a high-speed LAN.

**Characteristics:**

- Homogeneous: Same OS, same hardware.
- Single Managing Node: A master node controls allocation and job queues.
- Example: SLURM Clusters.

This is the de-facto setup for GPU resources.



# High-Performance Distributed Networking

The standard for communication in GPU clusters is the Message-Passing Interface (MPI), a high-level method compared to lower-level methods like Sockets.

**Why not Sockets?**

- Sockets (TCP/IP) are too low-level and often lack the speed required for parallel computing.
- Sockets do not support complex topologies or group communication easily.

**MPI Design Goals:**

- Hardware Independence: Works on both shared-memory machines and networks.
- Topology Support: Processes can be organized into groups and topologies (e.g., Ring All-Reduce)
- Identifier: Processes are identified by (groupID, processID), not IP address.

# MPI Operations

MPI implements several different communication operations to streamline networking in distributed computing clusters. Here are a few:

**Transient Asynchronous (Non-blocking):**

- `MPI_bsend`: Append message to local send buffer and return immediately (local copy).
- `MPI_isend`: Pass a pointer to the message. The system transmits when ready (zero-copy, highest speed).

**Transient Synchronous (Blocking):**

- `MPI_sendrecv`: Send a request and block until a reply is received.
- `MPI_ssend`: Block until the receiver explicitly accepts the message.

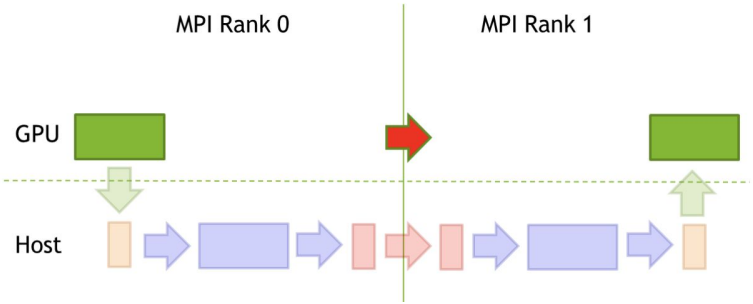
# Example: CUDA-Aware MPI

MPIs enable accelerated communication among distributed hardware accelerators like GPUs/TPUs.

In NVIDIA GPUs, CUDA-Aware MPIs often use GPUDirect RDMA to optimize CUDA performance.

If GPUDirect RDMA is available the buffer can be directly moved to the network without touching the host memory at all.

The data is directly moved from the buffer in the device memory of MPI Rank 0 to the device memory of MPI Rank 1.




RDMA (Remote direct memory access)

# Multi-Agent Systems in NextGen Networks

# Why use Multi-Agent AI in Distributed Networks?

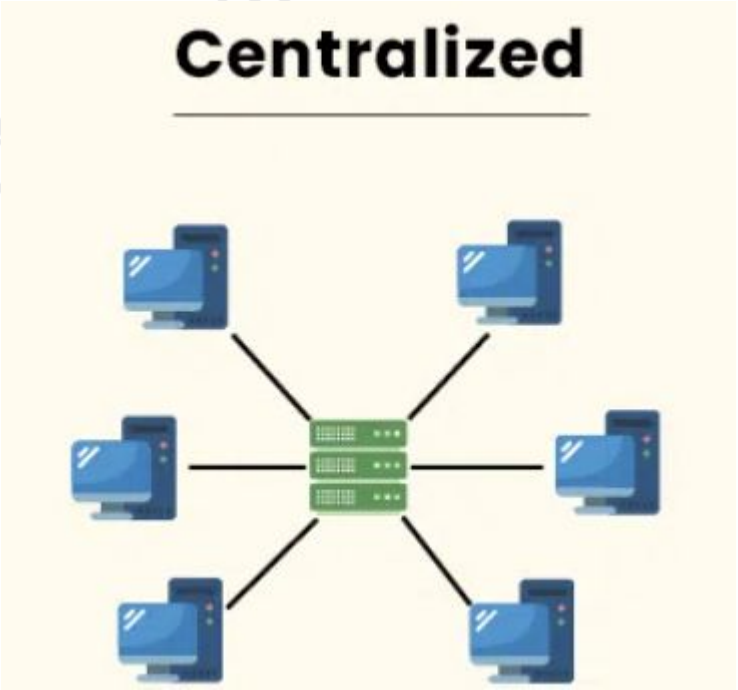
A very useful property of Multi-Agent systems is they allow agents to intelligently learn and share experiences in real-time as opposed to learning the same thing independently.

This is applied in many areas related to Distributed Systems as a whole:

- Concurrency
- Handling independent failures
- Overcoming the lack of a global clock

# Centralized Networks

Centralized networks often come with a single point of failure: If the central server goes down, no one can access the information

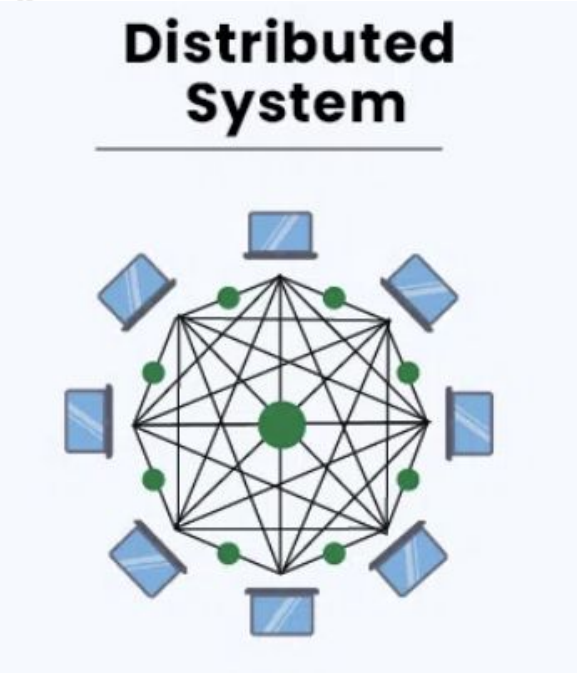


# Distributed Networks

In a Distributed Network, however, there are many nodes who can supply the information.

This is a critical capability, especially for upcoming 6G and 7G networks which are going to have more capacity for devices than ever before.

The amount of traffic could very easily overwhelm a purely centralized network!



# 6G / 7G to Enable Multi-Agent Systems

One important guarantee of 6G / 7G specifications is *ultra-low latency*.

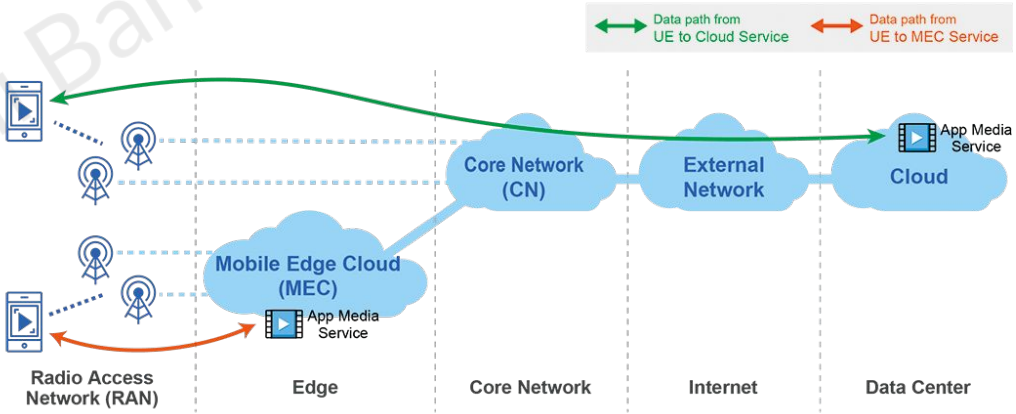
Formally, latency is the time it takes for a device to send a data packet to another device and receive a response.

A primary use for centralized networks was often their low latency, but as latency gets lower and lower, more decentralized multi-agent edge computing systems (e.g. Self-Driving Cars) become practical in the real world.

# Edge Computing as a Distributed System

An emerging architecture for Distributed Networks is *Multi-Access Edge Computing (MEC)* : a paradigm where computation is offloaded from an abstract “cloud” to physical devices users possess (e.g. phones, cars, etc.)

This allows for data to be immediately processed and analyzed without needing to go through so many nodes.



# Example: Multi-Agent Distributed Preventative Maintenance

| Agent Role | Core Responsibility | Distributed Computing Link |
|---|---|---|
| Failure Agent | Real-time anomaly detection, Root Cause Analysis, Incident Reporting | Local execution of AI/ML models directly on Edge infrastructure, processing high-velocity sensor data |
| Work Order Agent | Drafts maintenance orders, estimates resources required | Integration with distributed work order web services |
| Planning Agent | Schedules optimal, non-disruptive maintenance slots based on constraints | Utilizes distributed optimization algorithms for constraint satisfaction scheduling |

# Questions?