Principles of Network Applications

Creating a Network Application

Applications are built at the **end systems** (hosts) rather than the core of the network.

Key Idea

Applications are built by writing programs that:

- Run on different end systems.
- Communicate over the network by sending **messages**.
- Example: A web server communicates with a web browser.

Note: Core devices (routers, switches) do not run user applications. They only forward data.

Analogy

Think of the Internet like a postal system:

- The **applications** are like people writing and reading letters.
- The **network core** (routers, switches) is like the postal workers and sorting centers.
- Writers (applications) don't need to know the inner workings of the post office — only how to send/receive letters.

Benefit: Because development is only on the end systems, new applications can be created and deployed rapidly.

Architectural Paradigms

There are two dominant models of building network applications.

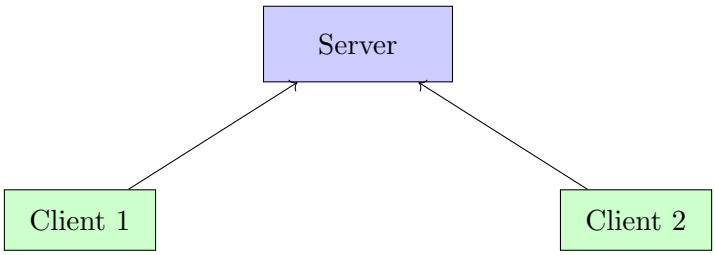
Client-Server Paradigm

- **Server:**
  - Always-on host.
  - Has a permanent IP address.

  - Usually located in data centers for scalability.

- **Clients:**
  - Contact and communicate with the server.
  - May be intermittently connected.
  - Often have dynamic IP addresses.
  - Do not directly communicate with each other.

- **Examples:** HTTP (web), IMAP (email), FTP (file transfer).



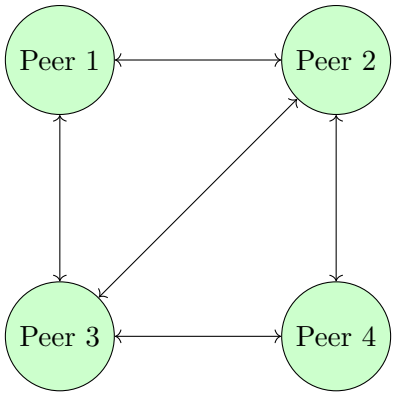
Analogy

Imagine a **restaurant**:

- The **server** is like the chef in the kitchen, always available.
- The **clients** are customers who come and go.
- Customers don't cook for each other — they always request food from the chef.

Peer-to-Peer (P2P) Architecture

- No always-on server.
- End systems (**peers**) directly communicate.
- Peers both **request and provide services**.
- **Self-scalability:** new peers bring more resources but also demand.
- Peers may be intermittently connected with changing IP addresses.
- More complex to manage.
- **Example:** BitTorrent.



Analogy

Think of P2P like a **potluck dinner**:

- Everyone brings food (resources) and shares.
- You are both a **consumer** and a **provider**.
- Unlike a restaurant (client-server), there is no single chef serving everyone.

Processes Communicating

- A **process** is a program running within a host.
- Within the same host: processes use inter-process communication (defined by the OS).
- Across different hosts: processes exchange **messages**.
- **Client process:** initiates communication.
- **Server process:** waits to be contacted.
- Even in P2P applications, a process can act as both client and server.

Analogy

Think of two people talking on the phone:

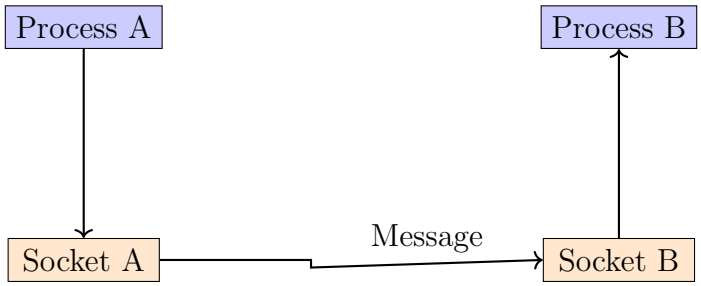
- One **initiates the call** (client).
- The other **answers** (server).
- Once connected, both can send and receive messages.

Sockets

Sockets Analogy

A socket is like a **door** between the process and the network:

- The sending process pushes a message out through its socket.
- The transport layer delivers the message to the receiving socket.
- Always two sockets are involved: sender's and receiver's.



Real-Life Analogy

A socket is like a **house's mailbox**:

- To send a letter, you drop it into your mailbox (sending socket).
- The postman (network) delivers it to the recipient's mailbox (receiving socket).
- The recipient retrieves it and hands it to the right person (process).

Addressing Processes

- To receive messages, a process must have an **identifier**.
- A host is identified by a unique **IP address**.
- But an IP address alone is not enough — multiple processes may run on the same host.
- Identifier = (IP address + port number).
- **Common port numbers:**
  - HTTP server: 80
  - Mail server (SMTP): 25
- Example: Sending an HTTP request to `gaia.cs.umass.edu`

  - IP address: 128.119.245.12
  - Port: 80

Analogy

Think of an office building:

- The **IP address** is like the building's street address.
- The **port number** is like the office room number.
- Without both, you can't deliver the message to the right person.

Application-Layer Protocols

Definition

An application-layer protocol specifies:

- Types of messages (request, response).
- Message syntax: fields and their structure.
- Message semantics: meaning of fields.
- Rules for sending and responding to messages.

Types of Protocols

- **Open protocols:**
  - Defined in RFCs.
  - Publicly available, ensure interoperability.
  - Examples: HTTP, SMTP.
- **Proprietary protocols:**
  - Owned by companies.
  - Not publicly documented.
  - Example: Zoom.

Analogy

Protocols are like **languages**:

- Two people can only talk effectively if they speak the same language.
- Similarly, two applications can only communicate if they follow the same protocol.