**PRACTICAL NUMBER 1**

**NETWORKING DEVICES**

**1. Router**

* **Function:** Connects different networks together; routes data from one network to another.
* **Key Use:** Connects home/office networks to the internet.
* **Smart Feature:** Assigns IP addresses, provides firewall and security.

**2. Switch**

* **Function:** Connects multiple devices (computers, printers, etc.) within a **LAN**.
* **Key Use:** Forwards data to the specific device it's meant for using MAC addresses.
* **Smarter than:** A hub (because it sends data only to the intended device).

**3. Hub *(Legacy device, less used now)***

* **Function:** Connects multiple Ethernet devices.
* **Key Use:** Broadcasts incoming data to **all ports** (not efficient).
* **Less intelligent:** Doesn’t filter data or know destination devices

**4. Modem *(Modulator-Demodulator)***

* **Function:** Converts digital data from a computer to analog for transmission (and vice versa).
* **Key Use:** Connects your home network to your ISP (Internet Service Provider).
* **Types:** DSL, Cable, Fiber modems

**5. Access Point (AP)**

* **Function:** Extends a wired network by adding **Wi-Fi** capability.
* **Key Use:** Used in wireless LANs (WLANs).
* **Often built-in:** Many routers include built-in access points.

**6. Network Interface Card (NIC)**

* **Function:** Allows a computer to connect to a network.
* **Key Use:** Every device on a network needs one.
* **Types:** Wired (Ethernet NIC), Wireless (Wi-Fi NIC)

**7. Bridge**

* **Function:** Connects two LANs using the **same protocol**.
* **Key Use:** Divides network into segments to reduce traffic.
* **Smarter than hub:** Filters traffic based on MAC addresses**🔹 8. Gateway**
* **Function:** Acts as a **translator** between different network protocols.
* **Key Use:** Connects different systems (e.g., enterprise network to the internet).

**9. Repeater**

* **Function:** Boosts or regenerates signals in a network.
* **Key Use:** Extends the range of a network by amplifying the signal.
* **Common in:** Large buildings or areas with weak signals.

**10. Firewall *(Hardware or Software)***

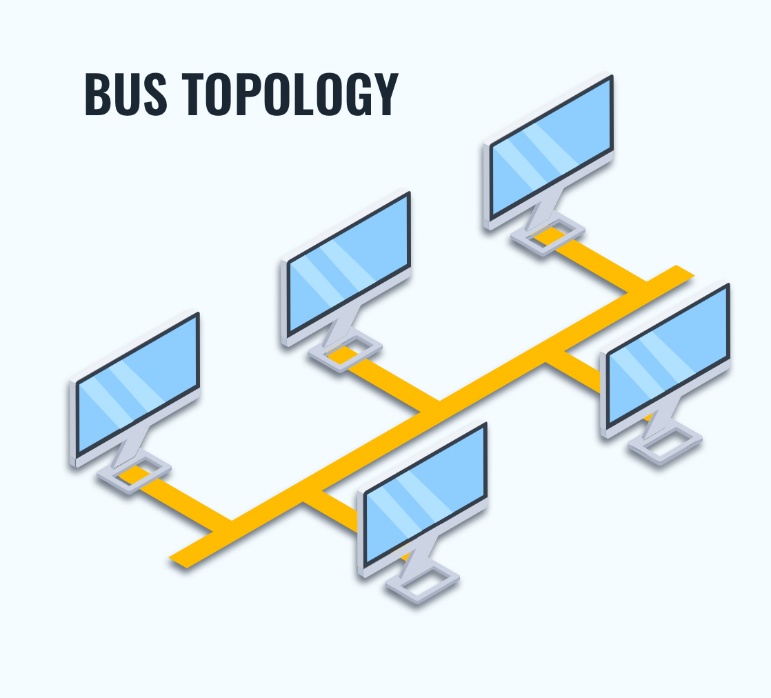
* **Function:** Controls incoming and outgoing network traffic based on security rules.
* **Key Use:** Protects against unauthorized access.

### ****11. Brouter (Bridge + Router)****

A **Brouter** is a hybrid networking device that combines the functions of both a **bridge** and a **router**.

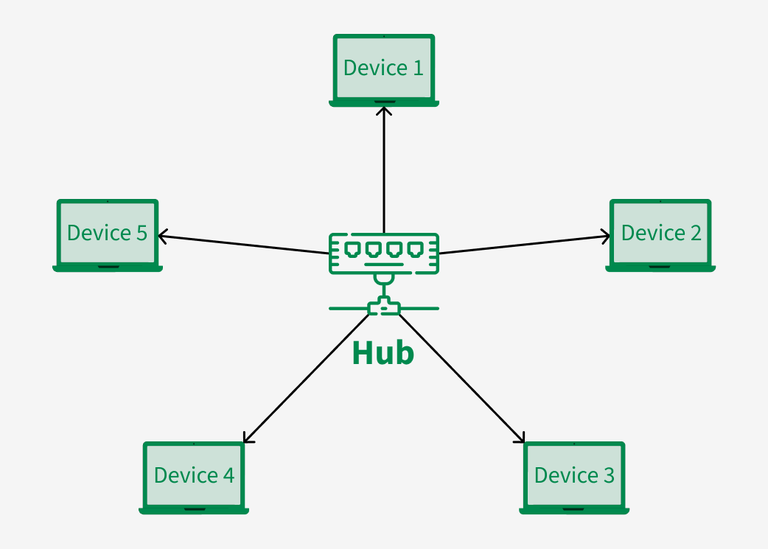
**TOPOLOGIES**

**1. Bus Topology**

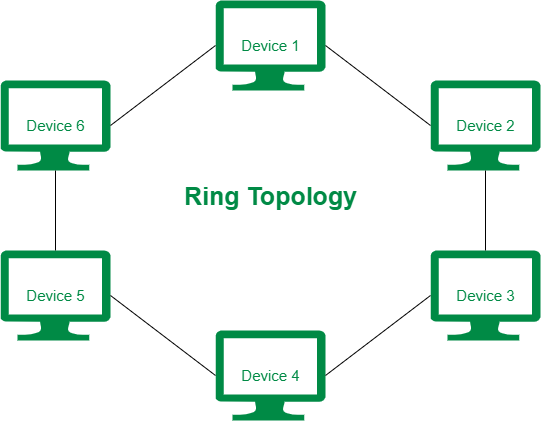
* **Structure:** All devices are connected to a single central cable (the bus).
* **Data Flow:** Data travels in both directions along the cable.
* **Advantages:**
  + Easy and inexpensive to implement.
  + Requires less cable than star topology.
* **Disadvantages:**
  + If the main cable fails, the entire network goes down.
  + Difficult to troubleshoot.
  + Limited cable length and number of nodes.
  + 

**2. Star Topology**

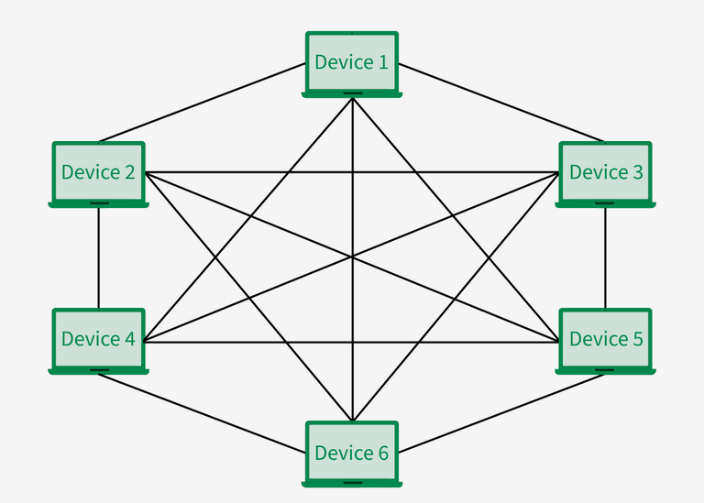
* **Structure:** All devices are connected to a central device (hub/switch).
* **Data Flow:** Devices communicate through the central device.
* **Advantages:**
  + Easy to install and manage.
  + Failure of one node doesn’t affect the rest.
* **Disadvantages:**
  + Central device is a single point of failure.
  + Uses more cable than bus topology.



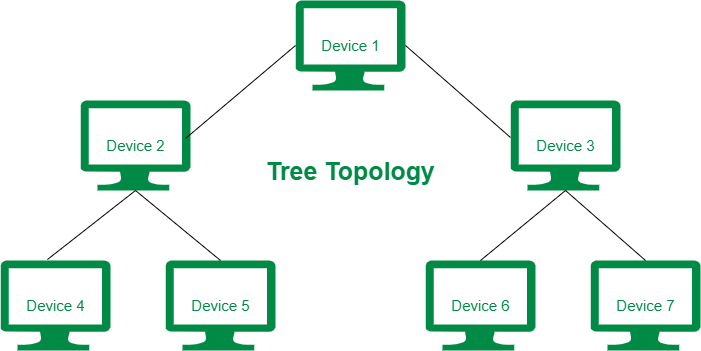
**3. Ring Topology**

* **Structure:** Each device is connected to two others, forming a circular path.
* **Data Flow:** Data travels in one direction (or both in a dual ring).
* **Advantages:**
  + Data flows in an orderly manner.
* **Disadvantages:**
  + A failure in any cable/device breaks the loop.
  + Troubleshooting is hard.
  + Adding/removing devices disrupts the network. 

**4. Mesh Topology**

* **Structure:** Every device is connected to every other device.
* **Types:**
  + **Full Mesh:** All nodes are interconnected.
  + **Partial Mesh:** Some nodes are interconnected.
* **Advantages:**
  + Very reliable and redundant.
  + Failure of one link doesn’t affect communication.
* **Disadvantages:**
  + Expensive and complex.
  + Requires lots of cabling and ports.
  + 

**5. Tree Topology *(Hierarchical)***

* **Structure:** A combination of star topologies connected to a central bus or main cable.
* **Advantages:**
  + Scalable and structured.
  + Easy to expand.
* **Disadvantages:**
  + Failure in the backbone affects the entire network.
  + More cabling.
  + 

**6. Hybrid Topology**

* **Structure:** Combination of two or more different topologies.
* **Example:** Star-Bus, Star-Ring.
* **Advantages:**
  + Flexible and scalable.
* **Disadvantages:**
  + Complex design and maintenance.
  + Costly.

BASIC NETWORK TROUBLESHOOTING COMMANDS

### 1. ping

* **Purpose:** Tests connectivity to another device or website.
* **Usage:**
* ping google.com
* ping 192.168.1.1
* **What it shows:** Whether the target is reachable, response time, and packet loss.

### 2. ipconfig (Windows only)

* **Purpose:** Displays IP address, subnet mask, default gateway, and more.
* **Usage:**
* ipconfig
* ipconfig /all
* ipconfig /release
* ipconfig /renew
* ipconfig /flushdns
* **Key Uses:**
  + /release and /renew help refresh IP address.
  + /flushdns clears the DNS cache.

### 3. ifconfig (Linux/macOS; deprecated in Linux but still used)

* **Purpose:** Shows and configures network interfaces.
* **Usage:**
* ifconfig

Use ip a instead on modern Linux systems.

### 4. ip (Linux only, replacement for ifconfig)

* **Usage:**
* ip a # Show IP addresses
* ip r # Show routing table
* ip link # Show interfaces

### 5. tracert (Windows) / traceroute (Linux/macOS)

* **Purpose:** Shows the path (hops) data takes to reach a destination.
* **Usage:**
* tracert google.com # Windows
* traceroute google.com # Linux/macOS

### 6. nslookup

* **Purpose:** Queries DNS to find the IP address of a domain name.
* **Usage:**
* nslookup google.com

### 7. route

* **Purpose:** Displays and manipulates the routing table.
* **Usage:**
* route print # Windows
* route -n # Linux

### 10. telnet

* **Purpose:** Tests connectivity to a specific port on a host.
* **Usage:**
* telnet google.com 80

**PRACTICAL 2&3**

**SOCKET PROGRAMMING FOR TRANSPORT LAYER PACKETS (TCP)**

**CLIENT CODE**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 10126

#define BUFFER\_SIZE 1024

int main() {

int sock;

struct sockaddr\_in serv\_addr;

char buffer[BUFFER\_SIZE];

FILE \*fp;

char filename[256] = "CN143prac2.txt"; // Change this to your file name

// Create socket

sock = socket(AF\_INET, SOCK\_STREAM, 0);

if (sock < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

// Convert IPv4 address from text to binary

if (inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr) <= 0) {

perror("Invalid address");

exit(EXIT\_FAILURE);

}

// Connect to server

if (connect(sock, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr)) < 0) {

perror("Connection failed");

exit(EXIT\_FAILURE);

}

// Send filename

send(sock, filename, strlen(filename), 0);

printf("Sending file: %s\n", filename);

// Open file to send

fp = fopen(filename, "rb");

if (!fp) {

perror("File open failed");

close(sock);

exit(EXIT\_FAILURE);

}

// Send file content

ssize\_t bytes\_read;

while ((bytes\_read = fread(buffer, 1, BUFFER\_SIZE, fp)) > 0) {

send(sock, buffer, bytes\_read, 0);

}

printf("File sent successfully.\n");

fclose(fp);

close(sock);

return 0;

}

SERVER

// server.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 10126

#define BUFFER\_SIZE 1024

int main() {

int server\_fd, client\_sock;

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t addr\_len;

char buffer[BUFFER\_SIZE];

FILE \*fp;

// Create socket

server\_fd = socket(AF\_INET, SOCK\_STREAM, 0);

if (server\_fd < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Configure server address

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

// Bind socket

if (bind(server\_fd, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) < 0) {

perror("Bind failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Listen for connections

if (listen(server\_fd, 5) < 0) {

perror("Listen failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

printf("Server listening on port %d...\n", PORT);

addr\_len = sizeof(client\_addr);

client\_sock = accept(server\_fd, (struct sockaddr\*)&client\_addr, &addr\_len);

if (client\_sock < 0) {

perror("Accept failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Receive filename

char filename[256];

int bytes\_received = recv(client\_sock, filename, sizeof(filename) - 1, 0);

if (bytes\_received < 0) {

perror("Filename receive failed");

close(client\_sock);

exit(EXIT\_FAILURE);

}

filename[bytes\_received] = '\0'; // Null-terminate the filename

printf("Receiving file: %s\n", filename);

// Open file to write received content

fp = fopen(filename, "wb");

if (!fp) {

perror("File open failed");

close(client\_sock);

exit(EXIT\_FAILURE);

}

// Receive file content

while ((bytes\_received = recv(client\_sock, buffer, BUFFER\_SIZE, 0)) > 0) {

fwrite(buffer, 1, bytes\_received, fp);

}

printf("File received successfully.\n");

fclose(fp);

close(client\_sock);

close(server\_fd);

return 0;

}

**OUTPUT**

**CLIENT**

Sending file: CN126prac2.txt

File sent successfully.

SERVER

Server listening on port 10126...

Receiving file: CN126prac2.txt

File received successfully.

**PRACTICAL 4&5**

**SOCKET PROGRAMMING FOR TRANSPORT LAYER PACKETS (UDP)**

// udp\_server.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 10126

#define BUFFER\_SIZE 1024

int main() {

int sockfd;

char buffer[BUFFER\_SIZE];

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t addr\_len = sizeof(client\_addr);

FILE \*fp;

// Create UDP socket

sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

if (sockfd < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Server address configuration

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

// Bind socket

if (bind(sockfd, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) < 0) {

perror("Bind failed");

close(sockfd);

exit(EXIT\_FAILURE);

}

printf("UDP Server listening on port %d...\n", PORT);

// Receive filename

char filename[256];

int n = recvfrom(sockfd, filename, sizeof(filename), 0,

(struct sockaddr\*)&client\_addr, &addr\_len);

filename[n] = '\0';

printf("Receiving file: %s\n", filename);

fp = fopen(filename, "wb");

if (!fp) {

perror("File open failed");

close(sockfd);

exit(EXIT\_FAILURE);

}

// Receive file data

while (1) {

n = recvfrom(sockfd, buffer, BUFFER\_SIZE, 0,

(struct sockaddr\*)&client\_addr, &addr\_len);

if (n == 0 || (n == 4 && strncmp(buffer, "EOF", 3) == 0)) {

break; // End of file

}

fwrite(buffer, 1, n, fp);

}

printf("File received successfully.\n");

fclose(fp);

close(sockfd);

return 0;

}

CLIENT

// udp\_client.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define SERVER\_IP "127.0.0.1"

#define PORT 10126

#define BUFFER\_SIZE 1024

int main() {

int sockfd;

struct sockaddr\_in server\_addr;

FILE \*fp;

char buffer[BUFFER\_SIZE];

char filename[] = "CN143prac2.txt"; // Change if needed

// Create UDP socket

sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

if (sockfd < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Server address setup

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

inet\_pton(AF\_INET, SERVER\_IP, &server\_addr.sin\_addr);

// Send filename first

sendto(sockfd, filename, strlen(filename), 0,

(struct sockaddr\*)&server\_addr, sizeof(server\_addr));

printf("Sending file: %s\n", filename);

// Open file

fp = fopen(filename, "rb");

if (!fp) {

perror("File open failed");

close(sockfd);

exit(EXIT\_FAILURE);

}

// Send file contents

size\_t bytes\_read;

while ((bytes\_read = fread(buffer, 1, BUFFER\_SIZE, fp)) > 0) {

sendto(sockfd, buffer, bytes\_read, 0,

(struct sockaddr\*)&server\_addr, sizeof(server\_addr));

}

// Send EOF signal

sendto(sockfd, "EOF", 4, 0, (struct sockaddr\*)&server\_addr, sizeof(server\_addr));

printf("File sent successfully.\n");

fclose(fp);

close(sockfd);

return 0;

}

**OUTPUT**

CLIENT

Sending file: CN143prac2.txt

Line 1: Hello from the client!

Line 2: This is UDP file transfer.

Line 3: Using sockets in C.

File sent successfully.

SERVER

UDP Server listening on port 10126...

Receiving file: CN126prac3.txt

Line 1: Hello from the client!

Line 2: This is UDP file transfer.

Line 3: Using sockets in C.

File 'CN126prac3.txt' received successfully.