



# C PROGRAMMING SOCKET COMMUNICATION

2024

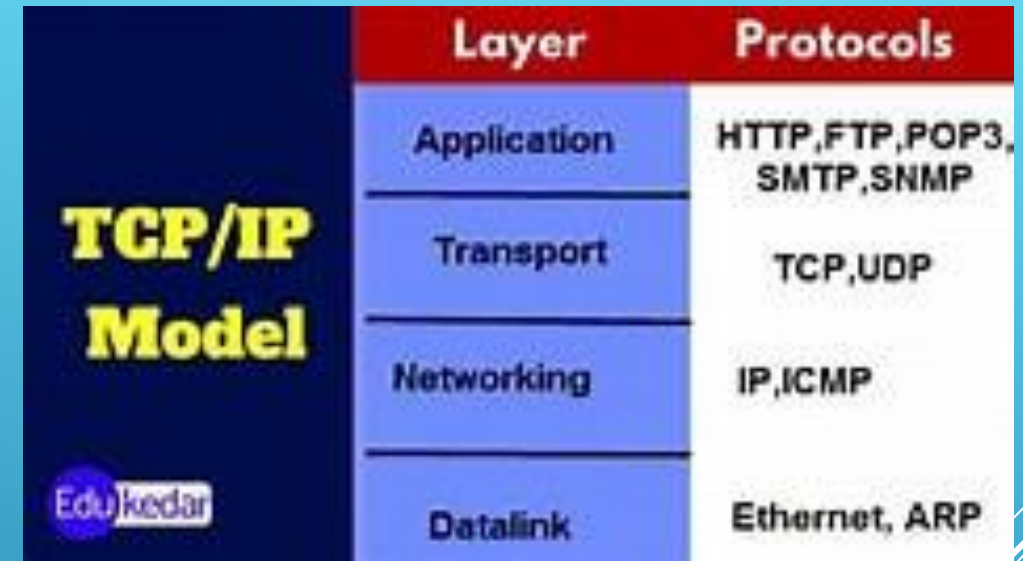
EUGENE KO

**Application Layer:** This layer is responsible for providing network services directly to end-users or applications. It includes protocols such as HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol), and DNS (Domain Name System).

**Transport Layer:** The transport layer is responsible for end-to-end communication between the source and destination hosts. It ensures reliable and orderly delivery of data by handling issues such as flow control, error correction, and congestion control. The most common protocols at this layer are TCP (Transmission Control Protocol) and UDP (User Datagram Protocol).

**Internet Layer:** This layer is responsible for routing packets across different networks to reach their destination. It deals with logical addressing (such as IP addresses) and packet forwarding. The main protocol at this layer is the Internet Protocol (IP).

**Link Layer:** Also known as the Network Interface Layer or Network Access Layer, this layer deals with the physical transmission of data over the network medium. It includes protocols and standards for connecting devices within the same local network, such as Ethernet, Wi-Fi, and PPP (Point-to-Point Protocol).

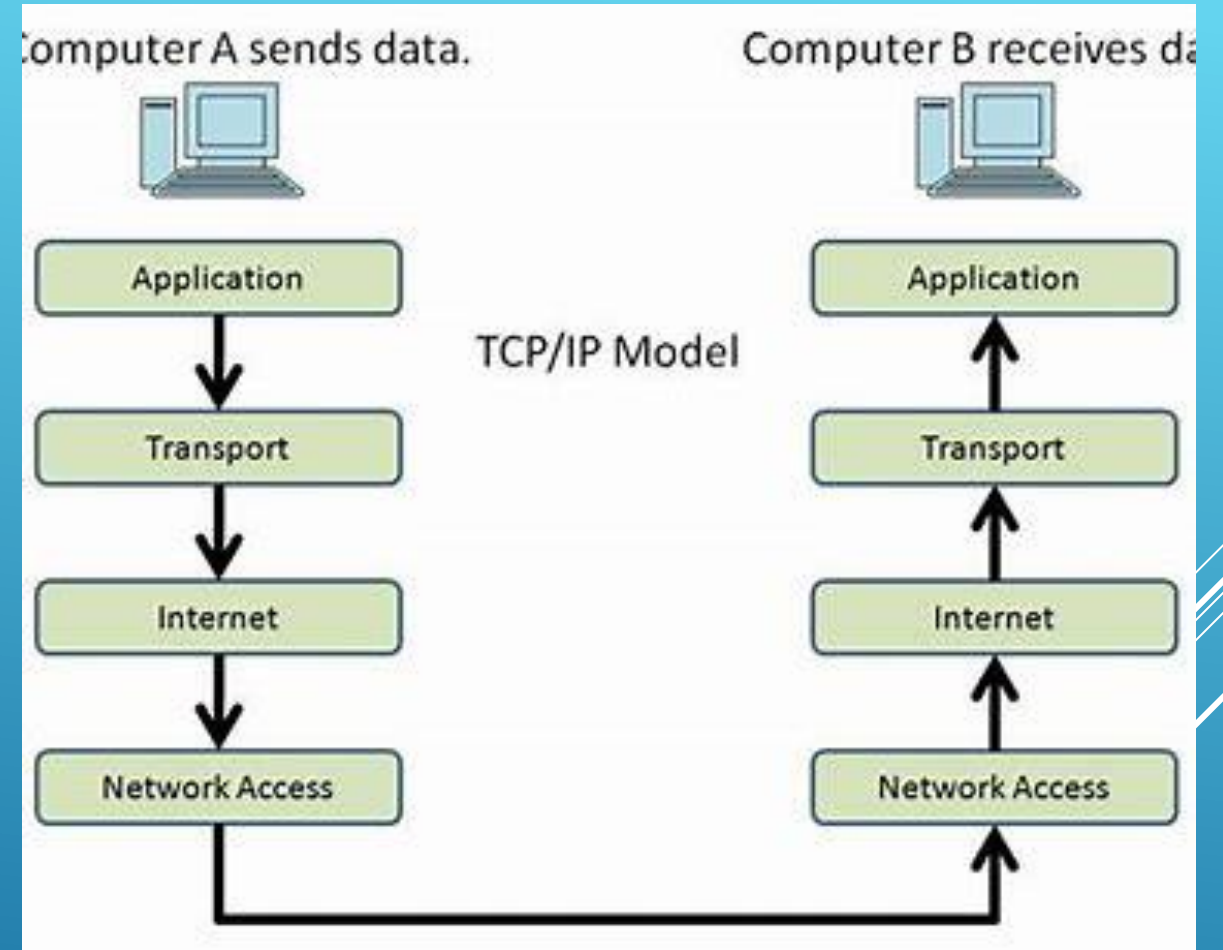


The diagram illustrates the TCP/IP Model, showing four layers and their associated protocols. The layers are listed in a column on the left, and the protocols are listed in a column on the right. The layers are Application, Transport, Networking, and Datalink. The protocols are HTTP, FTP, POP3, SMTP, SNMP for Application; TCP, UDP for Transport; IP, ICMP for Networking; and Ethernet, ARP for Datalink. The diagram is titled 'TCP/IP Model' and includes a logo for 'Edu kedar'.

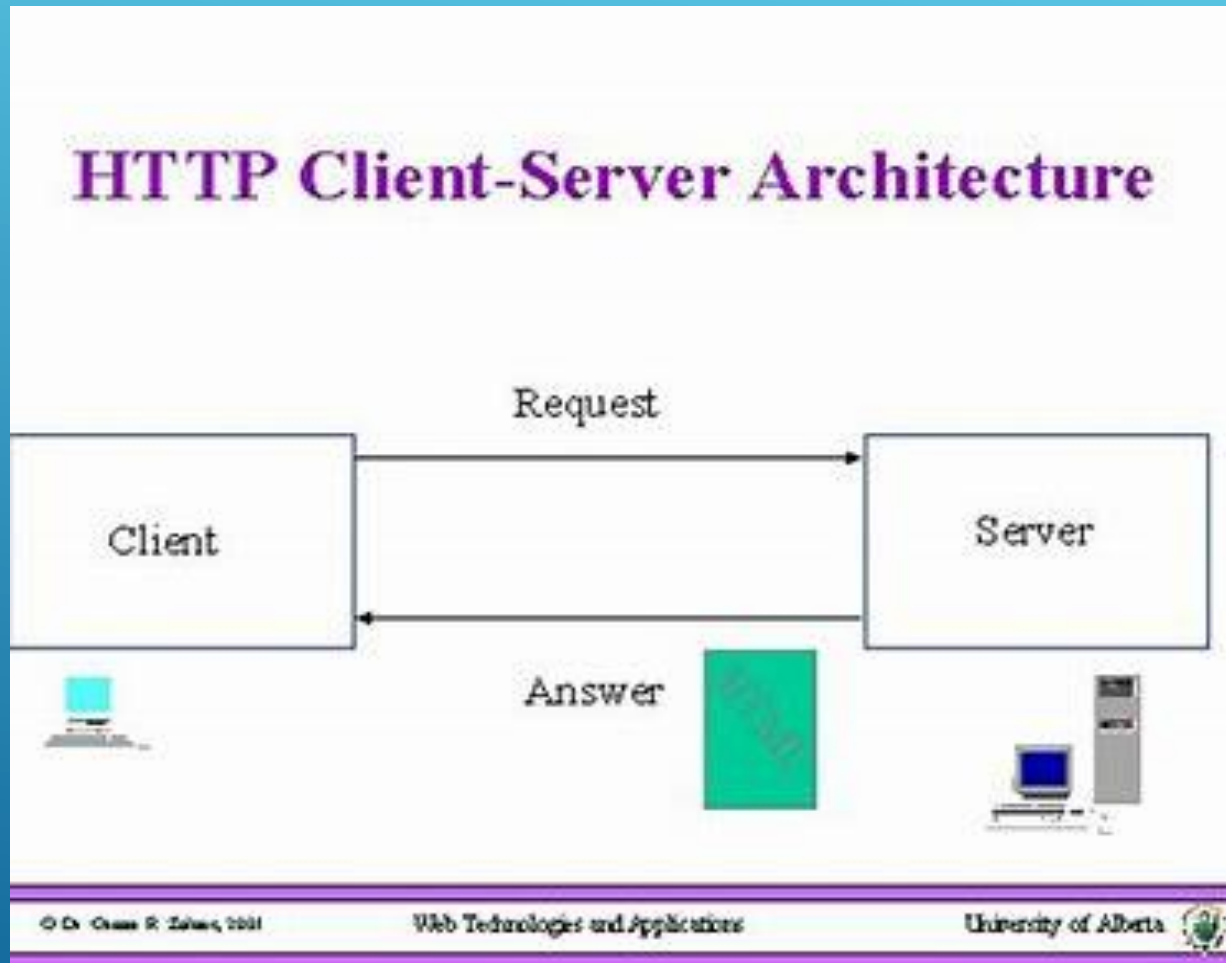
| Layer       | Protocols                   |
|-------------|-----------------------------|
| Application | HTTP, FTP, POP3, SMTP, SNMP |
| Transport   | TCP, UDP                    |
| Networking  | IP, ICMP                    |
| Datalink    | Ethernet, ARP               |

# ENCAPSULATION

encapsulation refers to the process of **adding protocol headers** (and possibly trailers) to data packets as they move down the protocol stack. Each layer of the TCP/IP model adds its own header to the data received from the layer above before passing it down to the next layer. **When data is transmitted across a network, each layer of the TCP/IP model encapsulates the data with its own header, creating a packet.**



# CLIENT SERVER MODEL



# CLIENT PROGRAM

```
#include <stdio.h>
#include <winsock2.h>
#pragma comment(lib, "ws2_32.lib")

#define PORT 8080
#define MAX_BUFFER_SIZE 1024

int main() {
    WSADATA wsa;
    SOCKET client_socket;
    struct sockaddr_in server;
    char buffer[MAX_BUFFER_SIZE];

    // Initialize Winsock
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
    {
        printf("WSAStartup failed.\n");
        return 1;
    }

    // Create socket
    if ((client_socket = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
    {
        printf("Socket creation failed.\n");
        WSACleanup();
        return 1;
    }

    // Prepare the sockaddr_in structure
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_port = htons(PORT);

    // Connect to server
    if (connect(client_socket, (struct sockaddr*)&server, sizeof(server)) < 0)
    {
        printf("Connection failed.\n");
        closesocket(client_socket);
        WSACleanup();
        return 1;
    }

    // Actual communication
    char message[] = "Hello from client";
    send(client_socket, message, strlen(message), 0);
    printf("Message sent to server: %s\n", message);

    int bytes_received = recv(client_socket, buffer, MAX_BUFFER_SIZE, 0);
    printf("Message from server: %s\n", buffer);

    closesocket(client_socket);
    WSACleanup();
    return 0;
}
```

# SERVER PROGRAM

```
#include <stdio.h>
#include <winsock2.h>

#define PORT 8080
#define MAX_BUFFER_SIZE 1024

int main() {
    WSADATA wsa;
    SOCKET server_socket, client_socket;
    struct sockaddr_in server, client;
    int client_len = sizeof(client);
    char buffer[MAX_BUFFER_SIZE];

    // Initialize Winsock
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
    {
        printf("WSAStartup failed.\n");
        return 1;
    }

    // Create socket
    if ((server_socket = socket(AF_INET,
    SOCK_STREAM, 0)) == INVALID_SOCKET) {
        printf("Socket creation failed.\n");
        return 1;
    }

    // Prepare the sockaddr_in structure
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(PORT);
```

```
    // Bind
    if (bind(server_socket, (struct
    sockaddr*)&server, sizeof(server)) == SOCKET_ERROR)
    {
        printf("Bind failed.\n");
        closesocket(server_socket);
        WSACleanup();
        return 1;
    }

    // Listen
    listen(server_socket, 3);

    // Accept an incoming connection
    if ((client_socket = accept(server_socket,
    (struct sockaddr*)&client, &client_len)) ==
    INVALID_SOCKET)
    {
        printf("Accept failed.\n");
        closesocket(server_socket);
        WSACleanup();
        return 1;
    }
```

```
    // Actual communication


    int bytes_received = recv(client_socket,
    buffer, MAX_BUFFER_SIZE, 0);

    send(client_socket, buffer, bytes_received, 0);

    printf("Message from client: %s\n", buffer);

    closesocket(server_socket);
    closesocket(client_socket);
    WSACleanup();
    return 0;
}
```

# WSAStartup()

- ▶ initializing the Winsock library
  - ▶ Winsock (Windows Sockets)
    - ▶ a programming interface and the supporting library
    - ▶ handling communication between a client and server applications over a network in the Windows operating system.
- 
- A series of white diagonal lines of varying lengths and thicknesses are positioned in the bottom right corner of the slide, creating a modern, abstract graphic element.



# SOCK\_STREAM

- ▶ specify the type of communication protocol
- ▶ Stream-oriented communication: This type of communication establishes a virtual connection between two endpoints, providing a continuous and reliable data flow. It ensures that data is delivered in the same order it was sent and without duplication or loss













