CS3205: Introduction to Computer Networks

Socket Programming in C

Background Details for Assignment 1

February 7, 2021

Introduction

- In computer networks, end systems are interconnected using packet switches and communication links
- End systems are used to run applications
- Communication links (wired and wireless) and packet switches (routers and switches) are used to forward packets
- Information is divided into packets that are transported from source to destination using protocols
- Protocol is a set of rules and procedures for communicating data packets in the network (e.g., HTTP and TCP)

Network overview

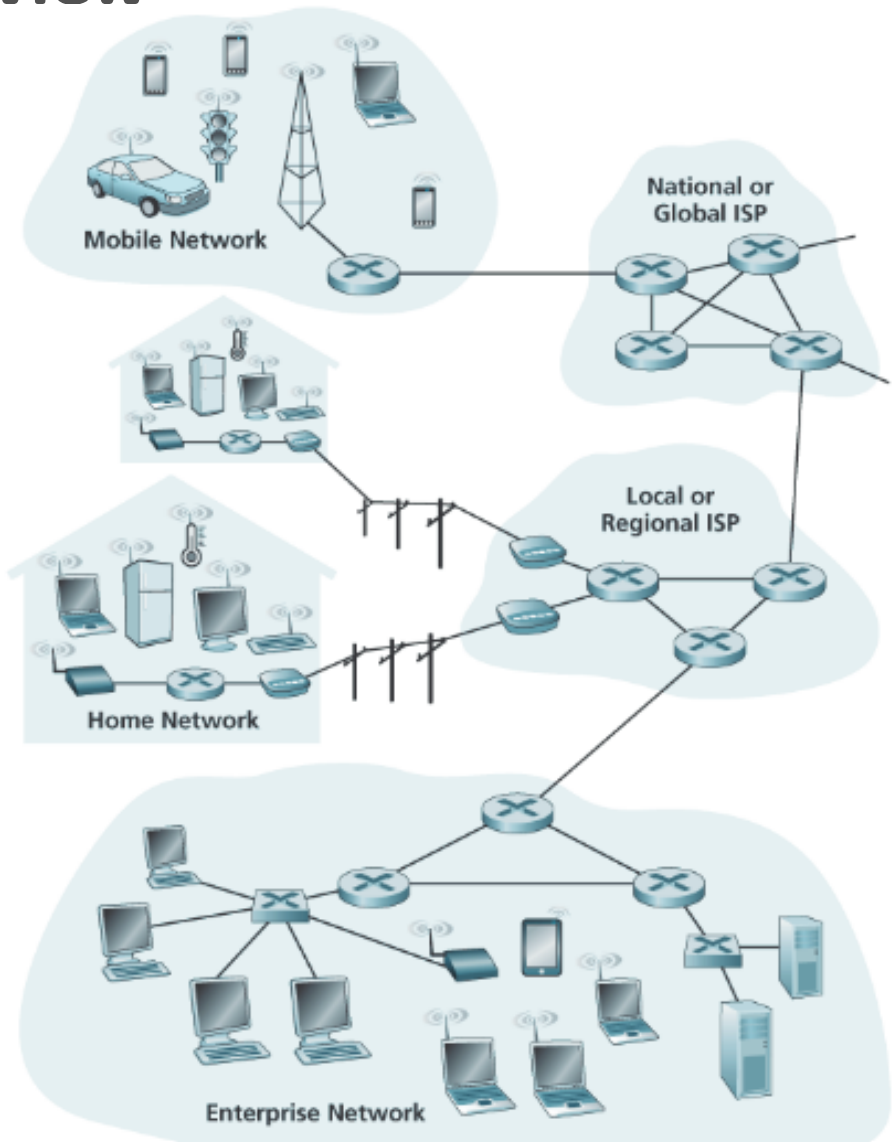

Figure: An example of a connected network[1]

[1] J. F. Kurose and K. W. Ross "Computer Networking: A Top Down Approach", 7th Edition, Pearson, 2017.

Transportation network

Figure: Transportation network[2]

[2] Internet.

Network protocols

- Packets are handled and delivered independently
- Different protocols are used to accomplish different communication tasks
- A set of protocols used to accomplish a task are considered as protocol families (e.g., TCP/IP)
- Many protocols are standardized by IETF and defined as RFCs

Layered architecture

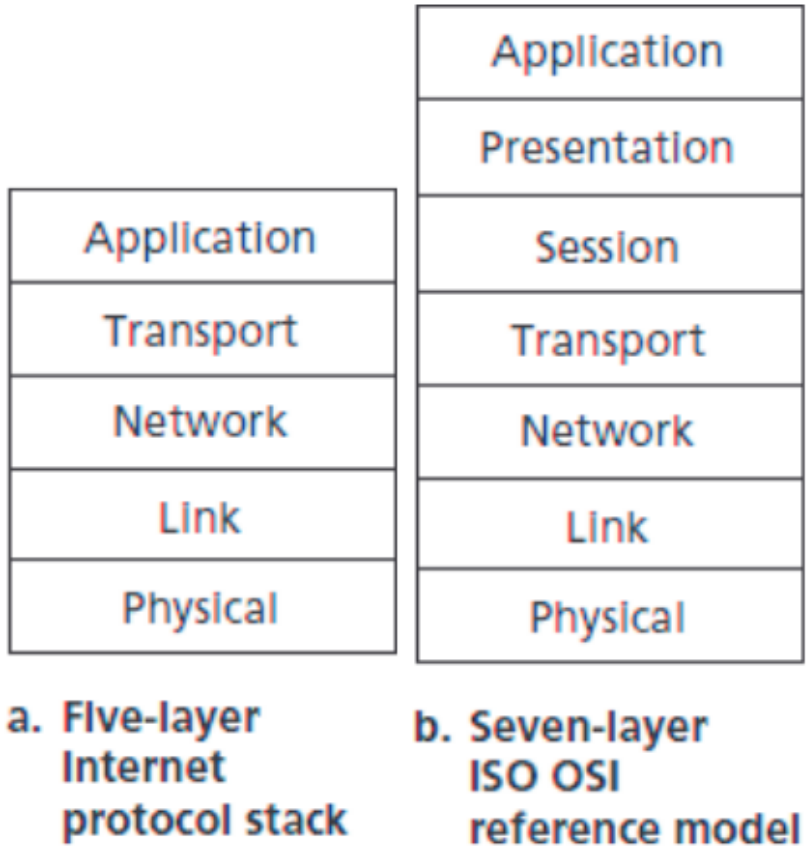

Figure: a) The Internet protocol stack and b) OSI reference model[3]

[3] J. F. Kurose and K. W. Ross "Computer Networking: A Top Down Approach", 7th Edition, Pearson, 2017.

TCP/IP network topology

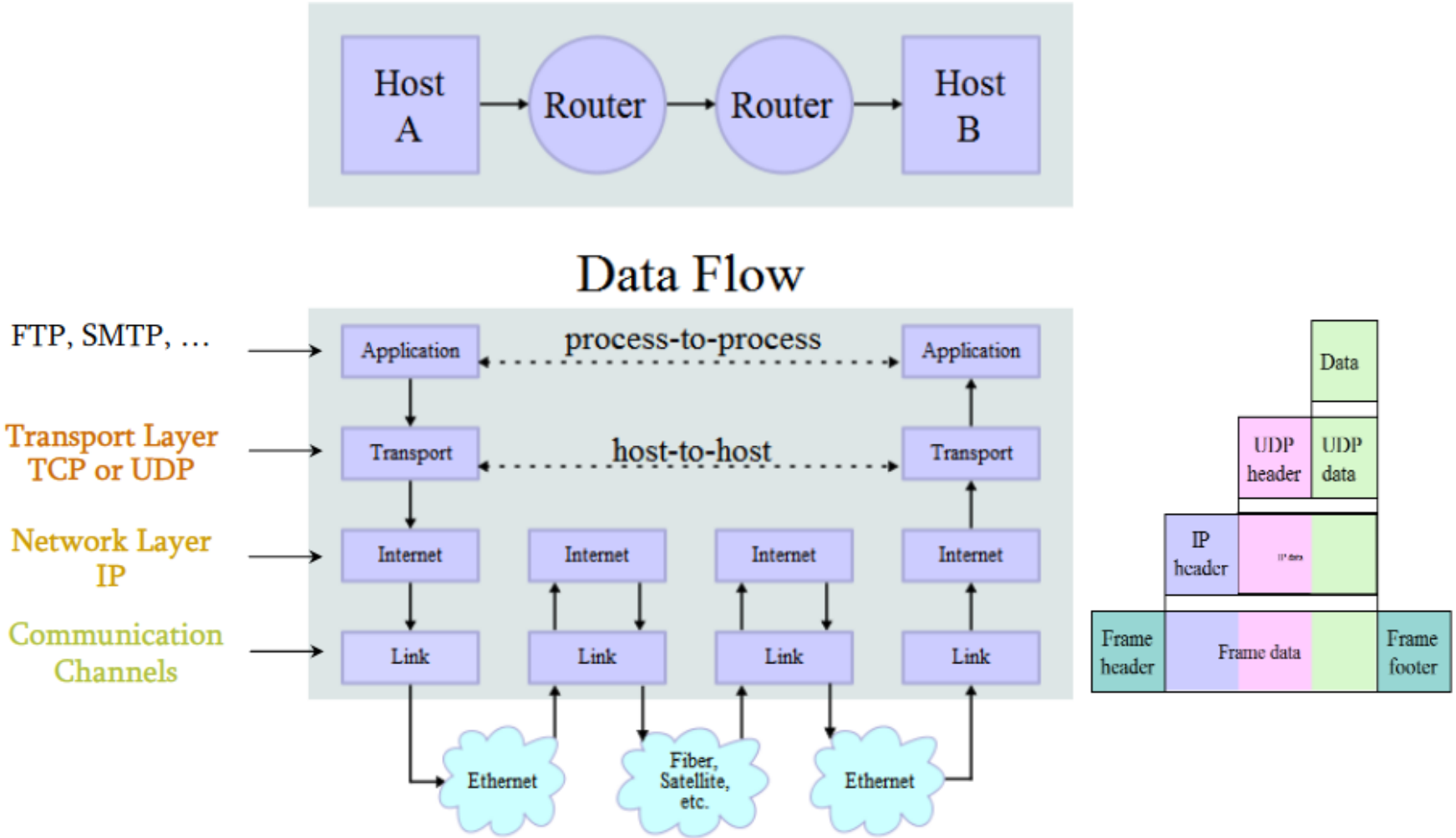

Figure: Internet protocol stack data flow[4]

[4] Internet

Internet protocol stack and service models

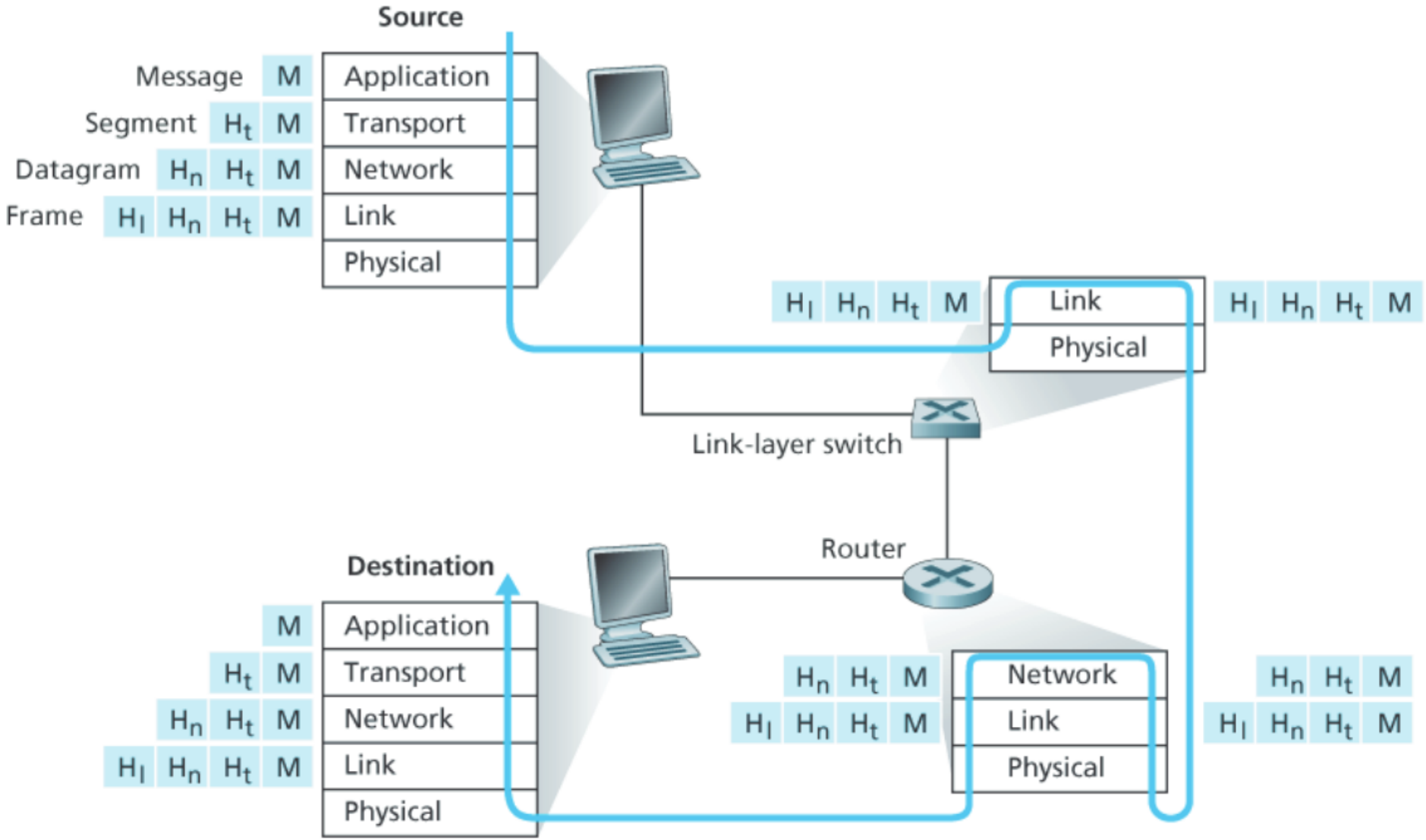

Figure: Association of the Internet protocol layer with entities[5]

[5] J. F. Kurose and K. W. Ross "Computer Networking: A Top Down Approach", 7th Edition, Pearson, 2017.

Transport layer protocol

- Transmission Control Protocol (TCP)
  - ◇ It provides connection-oriented service and bidirectional
  - ◇ It guarantees the delivery of application layer messages
  - ◇ It provides congestion control and flow control mechanisms
  - ◇ It's reliable (i.e., in order, all arrive, and no duplicates)
- User Datagram Protocol (UDP)
  - ◇ Unidirectional and unreliable
  - ◇ No acknowledgements
  - ◇ No retransmissions
  - ◇ Out of order and duplicates possible
  - ◇ Connectionless
- Both TCP and UDP use port numbers to provide application-specific services

Application architecture and processes

- Application architecture types: i) client-server and ii) peer-to-peer
- Process can be considered as a program that is running within a host
- Interprocess vs. processes running on different hosts
- A network application consists of pair of processes (client and server)
- A process sends or receives messages in the network through a software interface called a socket

TCP/IP network sockets

- Socket is one end point of a two way communication link
- Socket is the interface between the application layer and the transport layer within a host
- Socket is uniquely identified by its socket address to communicate with other hosts
- Socket address consists of i) transport protocol (TCP or UDP), ii) IP address, and iii) port number
- Socket uses FIFO method to transfer packets
- Socket types:
  - stream sockets (TCP) - reliable and connection oriented
  - datagram sockets (UDP) - connectionless and provides best-effort datagram service

Socket communication over the Internet

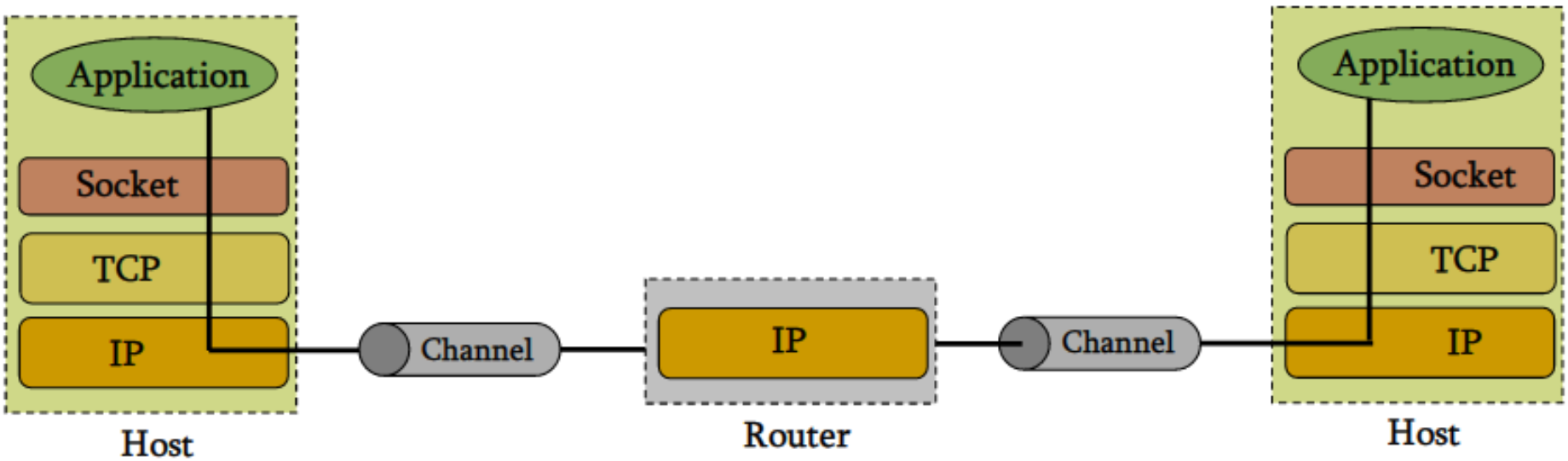

Figure: Socket communication between two hosts[6]

[6] Internet.

Sockets and ports

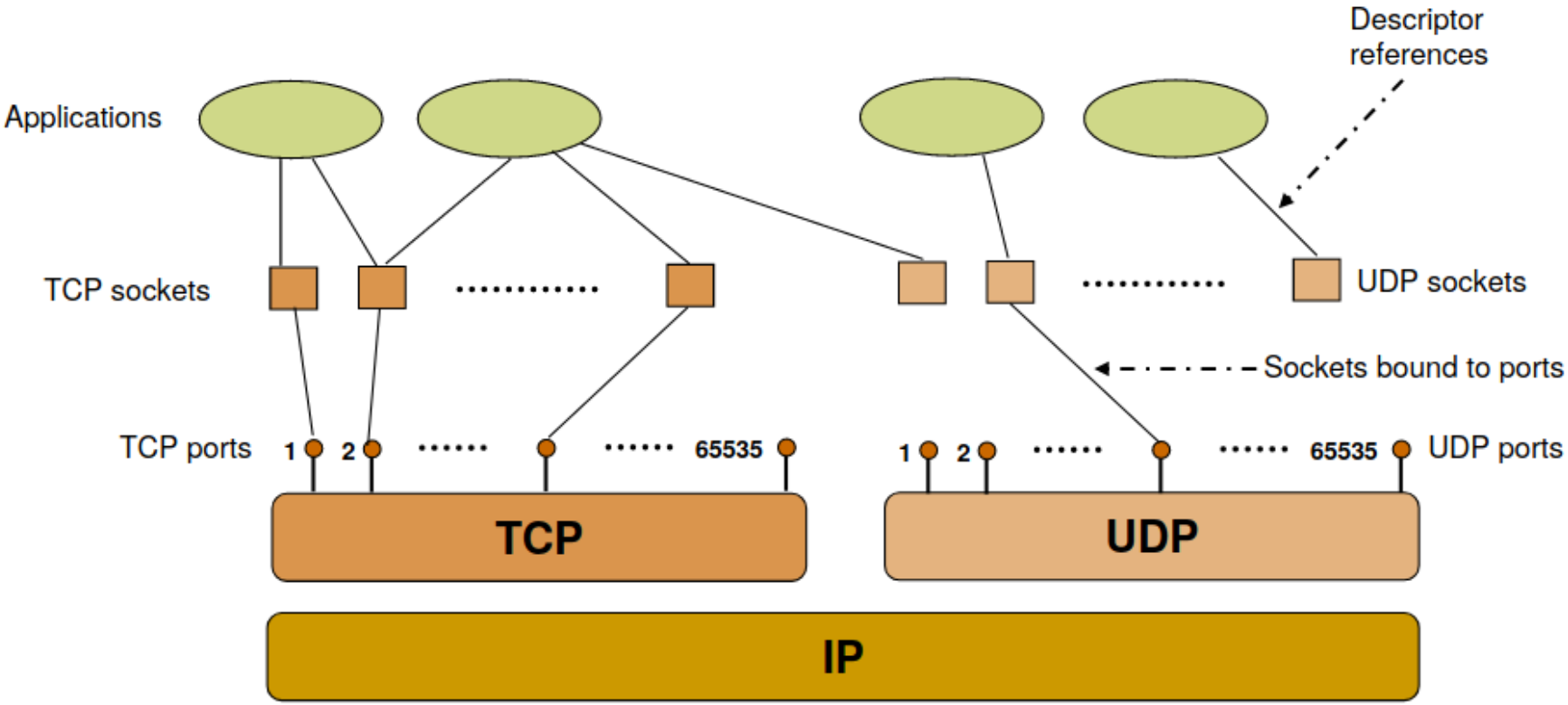

Figure: Applications, sockets, and ports[7]

[7] Internet.

Server and host communication

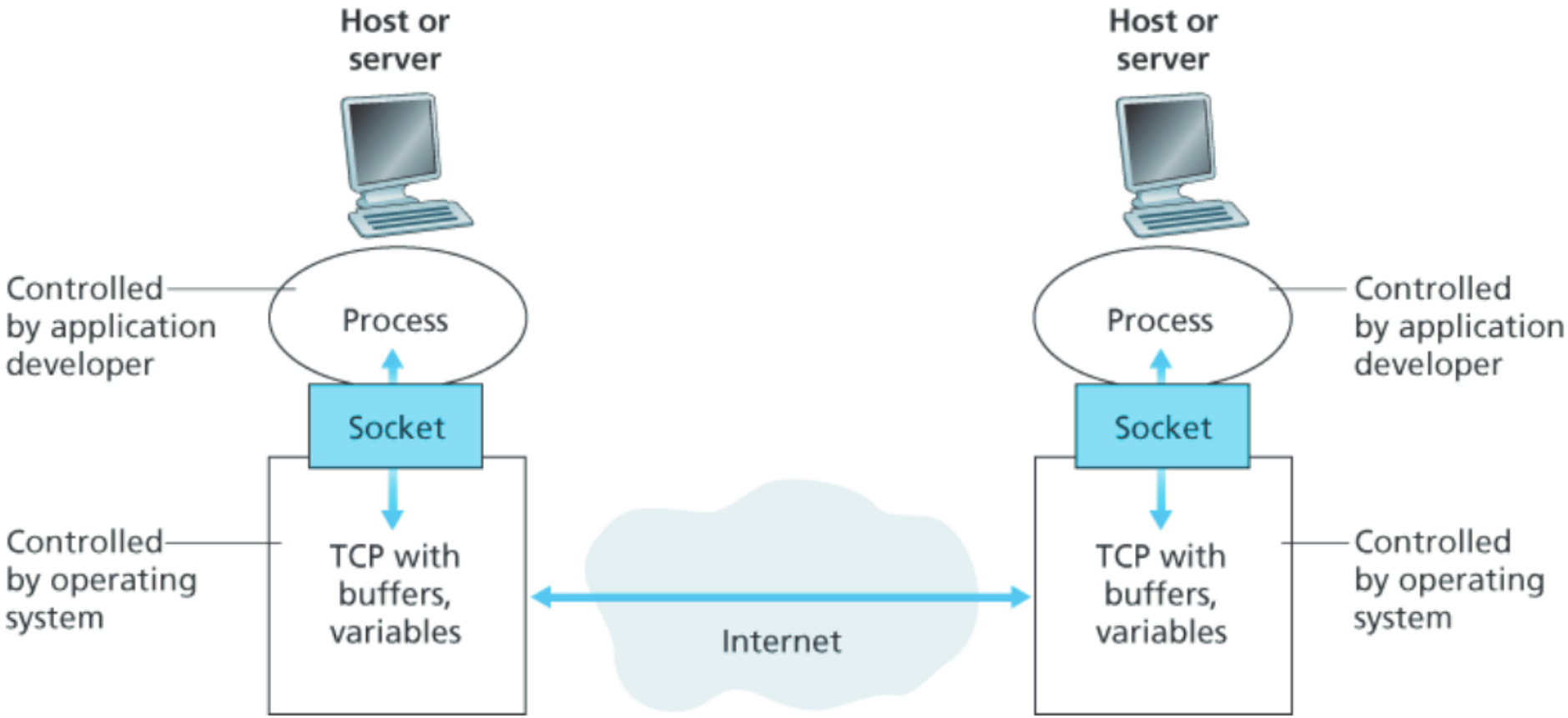

Figure: Application processes, sockets, and transport protocol[8]

[8] J. F. Kurose and K. W. Ross "Computer Networking: A Top Down Approach", 7th Edition, Pearson, 2017.

Server vs. client sockets

- Server socket
  - ◇ passive socket
  - ◇ attaches to a port address, and waits for clients requests and then responds

- Client socket
  - ◇ active socket
  - ◇ must know the server socket address for initiating the communication

Client-server communication using sockets

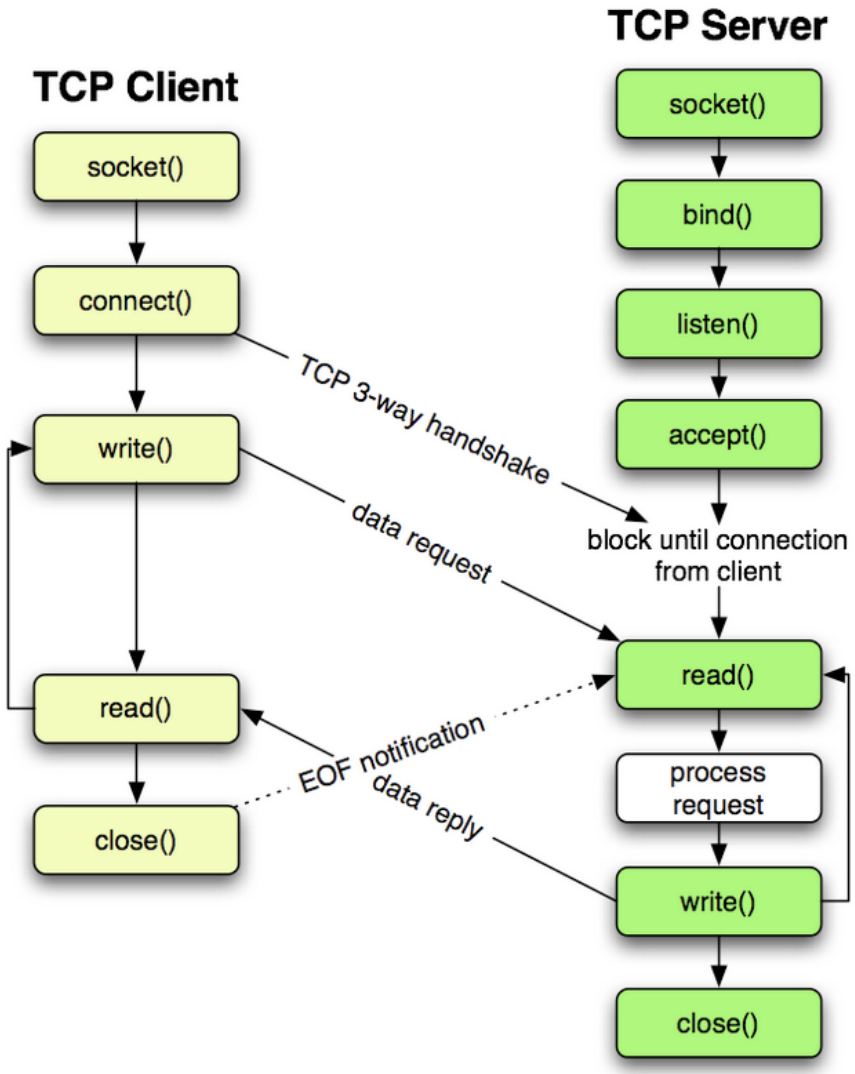

Figure: Client-server communication[9]

[9] Internet.

Socket function calls and meaning

	Primitive	Meaning
	SOCKET	Create a new communication endpoint
	BIND	Associate a local address (port) with a socket
Only needed for Streams	LISTEN	Announce willingness to accept connections
Only needed for Streams	ACCEPT	Passively establish an incoming connection
Only needed for Streams	CONNECT	Actively attempt to establish a connection
To/From forms for Datagrams	SEND(TO)	Send some data over the socket
To/From forms for Datagrams	RECEIVE(FROM)	Receive some data over the socket
	CLOSE	Release the socket

Figure: Socket function calls[10]

[10] Internet.

Create socket in C: socket()

- int sockid = socket(family, type, protocol);
  - *sockid* denotes a socket descriptor
  - *family* denotes a communication domain that is AF_INET for IPv4 and AF_INET6 for IPv6 (AF stands for address family)
  - *type* denotes communication type that is SOCK_STREAM for TCP and SOCK_DGRAM for UDP
  - *protocol* specifies protocol that is IPPROTO_TCP and IPPROTO_UDP (0 is set to default protocol)
- upon failures return -1

Assign address to socket: bind()

- Associates and reserves a port for socket
- int status = bind(sockid, & addrport, size);
- *addrport* denotes the IP address and port of the server
- *size* denotes the size of the *addrport* structure
- *status* upon failure -1 returned

Listen for connections: listen()

- int status = listen(sockid, queueLimit);
- *queueLimit* denotes the no of participants that can wait for a connection
- *status* 0 if listening and -1 for error

Establish connection: connect()

- int status = connect(sockid, & foreignAddr, addrlen);
- *foreignAddr* denotes address of the passive participant
- *addrlen* denotes sizeof(foreignAddr)
- *status* denotes 0 for successful and -1 for error

Incoming connection: accept()

- int s = accept(sockid, & clientAddr, & addressLen);
- *clientAddr* denotes address of the active participant
- *addrLen* denotes sizeof(clientAddr)

Exchanging data with stream socket

- int count = send(sockid, msg, msgLen, flags);
  - *msg* denotes message to be transmitted, const void[]
  - *msgLen* denotes length of message to transmit
  - *flags* denotes integer, usually just 0
  - *count* denotes no of bytes to be transmitted

- int count = recv(sockid, recvBuf, bufLen, flags);
  - *recvBuf* denotes that stores received bytes, void[]
  - *bufLen* denotes bytes received
  - *flags* denotes integer, usually just 0
  - *count* denotes no of bytes received (-1 if error)

Socket close in C: close()

- status = close(sockid);
- *status* 0 if successful, -1 if error

Server program in C

```c
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#define MAX 80
#define PORT 8080
#define SA struct sockaddr
```

Figure: Server program headerfiles[11]

[11] Internet.

Server program in C

```c
int main()
{
    int sockfd, connfd, len;
    struct sockaddr_in servaddr, cli;

    // socket create and verification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    // Binding newly created socket to given IP and verification
    if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
        printf("socket bind failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully binded..\n");

    // Now server is ready to listen and verification
    if ((listen(sockfd, 5)) != 0) {
        printf("Listen failed...\n");
        exit(0);
    }
    else
        printf("Server listening..\n");
    len = sizeof(cli);

    // Accept the data packet from client and verification
    connfd = accept(sockfd, (SA*)&cli, &len);
    if (connfd < 0) {
        printf("server acccept failed...\n");
        exit(0);
    }
    else
        printf("server acccept the client...\n");

    // Function for chatting between client and server
    func(connfd);

    // After chatting close the socket
    close(sockfd);
}
```

Figure: Server main program[12]

[12] Internet.

Server program in C

```c
// Function designed for chat between client and server.
void func(int sockfd)
{
    char buff[MAX];
    int n;
    // infinite loop for chat
    for (;;) {
        bzero(buff, MAX);

        // read the message from client and copy it in buffer
        read(sockfd, buff, sizeof(buff));
        // print buffer which contains the client contents
        printf("From client: %s\t To client : ", buff);
        bzero(buff, MAX);
        n = 0;
        // copy server message in the buffer
        while ((buff[n++] = getchar()) != '\n')
            ;

        // and send that buffer to client
        write(sockfd, buff, sizeof(buff));

        // if msg contains "Exit" then server exit and chat ended.
        if (strncmp("exit", buff, 4) == 0) {
            printf("Server Exit...\n");
            break;
        }
    }
}
```

Figure: Server receive and send program[13]

[13] Internet.

Client program in C

```c
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#define MAX 80
#define PORT 8080
#define SA struct sockaddr
```

Figure: Client program headerfiles[14]

[14] Internet.

Client program in C

```c
int main()
{
    int sockfd, connfd;
    struct sockaddr_in servaddr, cli;

    // socket create and varification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    servaddr.sin_port = htons(PORT);

    // connect the client socket to server socket
    if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr)) != 0) {
        printf("connection with the server failed...\n");
        exit(0);
    }
    else
        printf("connected to the server..\n");

    // function for chat
    func(sockfd);

    // close the socket
    close(sockfd);
}
```

Figure: Client main program[15]

[15] Internet.

Client program in C

```c
void func(int sockfd)
{
    char buff[MAX];
    int n;
    for (;;) {
        bzero(buff, sizeof(buff));
        printf("Enter the string : ");
        n = 0;
        while ((buff[n++] = getchar()) != '\n')
            ;
        write(sockfd, buff, sizeof(buff));
        bzero(buff, sizeof(buff));
        read(sockfd, buff, sizeof(buff));
        printf("From Server : %s", buff);
        if ((strncmp(buff, "exit", 4)) == 0) {
            printf("Client Exit...\n");
            break;
        }
    }
}
```

Figure: Client send and receive program[16]

[16] Internet.

Output

Server side:

```
Socket successfully created..
Socket successfully binded..
Server listening..
server acccept the client...
From client: hi
     To client : hello
From client: exit
     To client : exit
Server Exit...
```

Client side:

```
Socket successfully created..
connected to the server..
Enter the string : hi
From Server : hello
Enter the string : exit
From Server : exit
Client Exit...
```

Figure: Output of server and client programs[17]

[17] Internet.

References for more details

- J. F. Kurose and K. W. Ross "Computer Networking: A Top Down Approach", 7th Edition, Pearson, 2017.

- Introduction to Socket Programming in C using TCP/IP

  https://www.csd.uoc.gr/ hy556/material/tutorials/cs556-3rd-tutorial.pdf

- TCP client-server implementation in C

  https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/

Questions and Answers