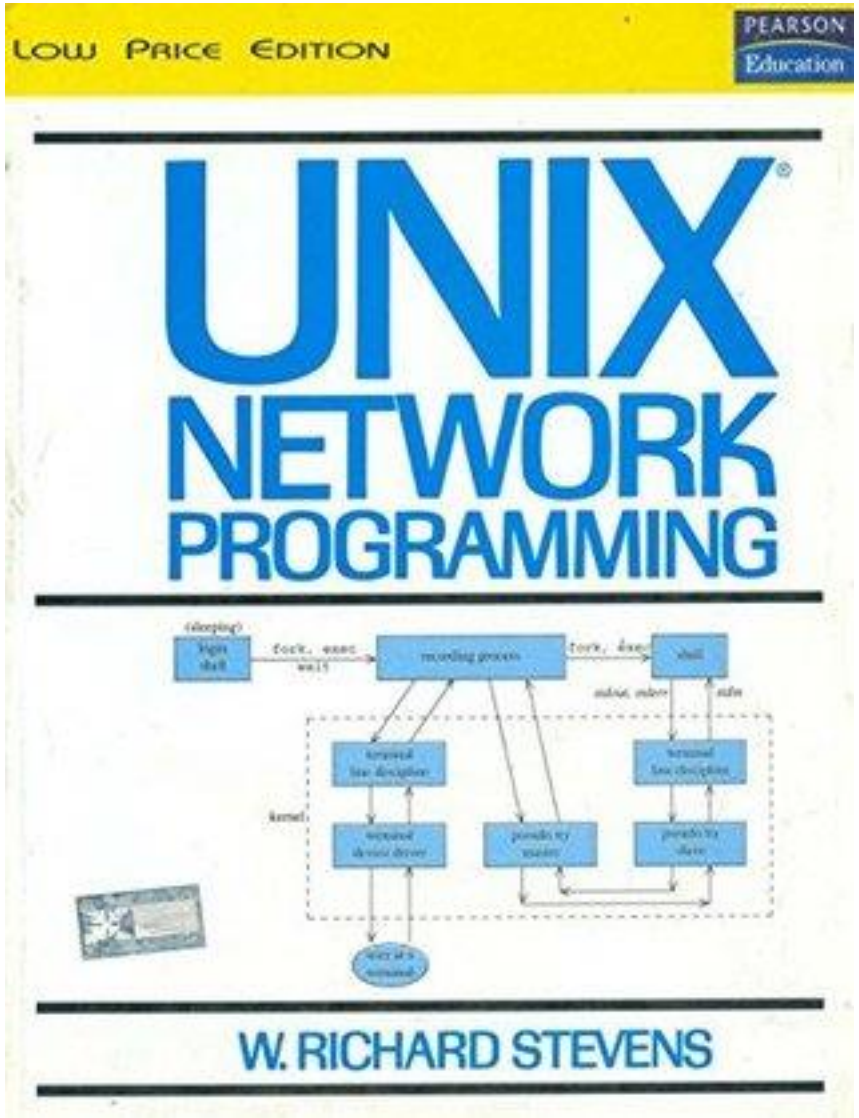
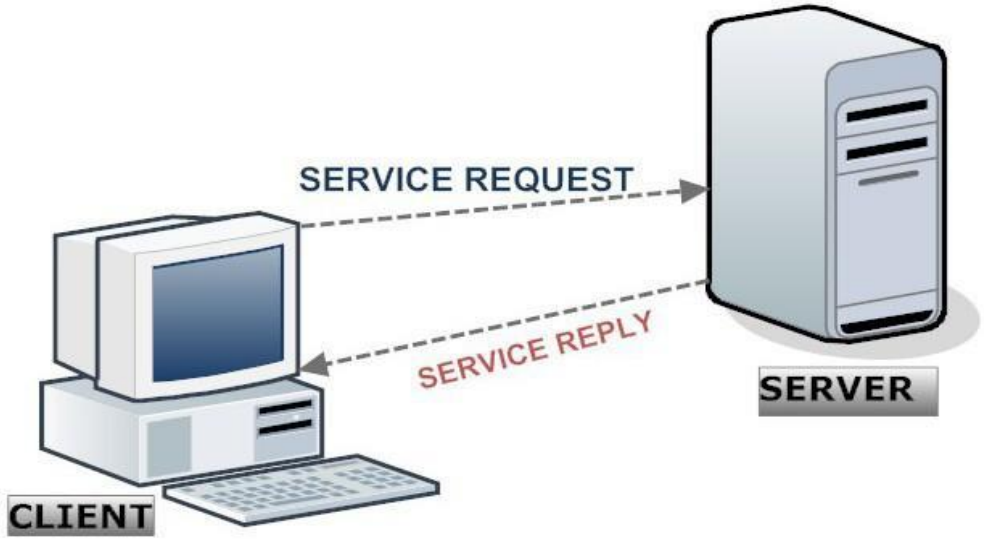
Network Programming



Network programming involves writing programs to communicate with processes either on the same or on other machines on the network using standard Protocols…

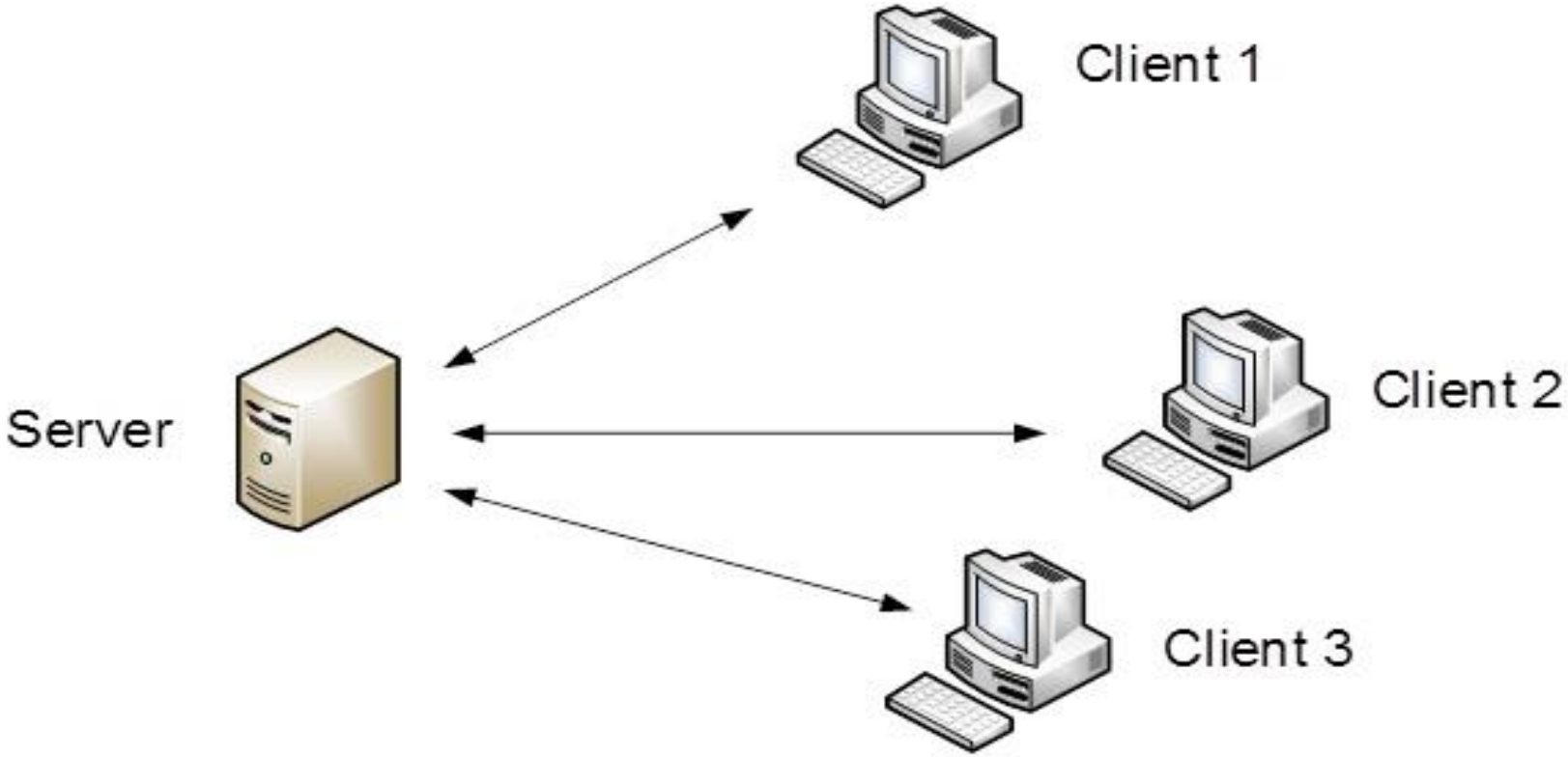
Introduction:

- When writing programs that communicate across a computer network, one must first invent,
- A protocol, an agreement on how those programs will communicate.
- **High-level decision must be made as to which program would initiate the communication first and when responses are expected…**
- Example,
- **WebServer program waits for clients to send request and only after the request is received it responds with a reply…**



Single Server – Serving multiple Clients

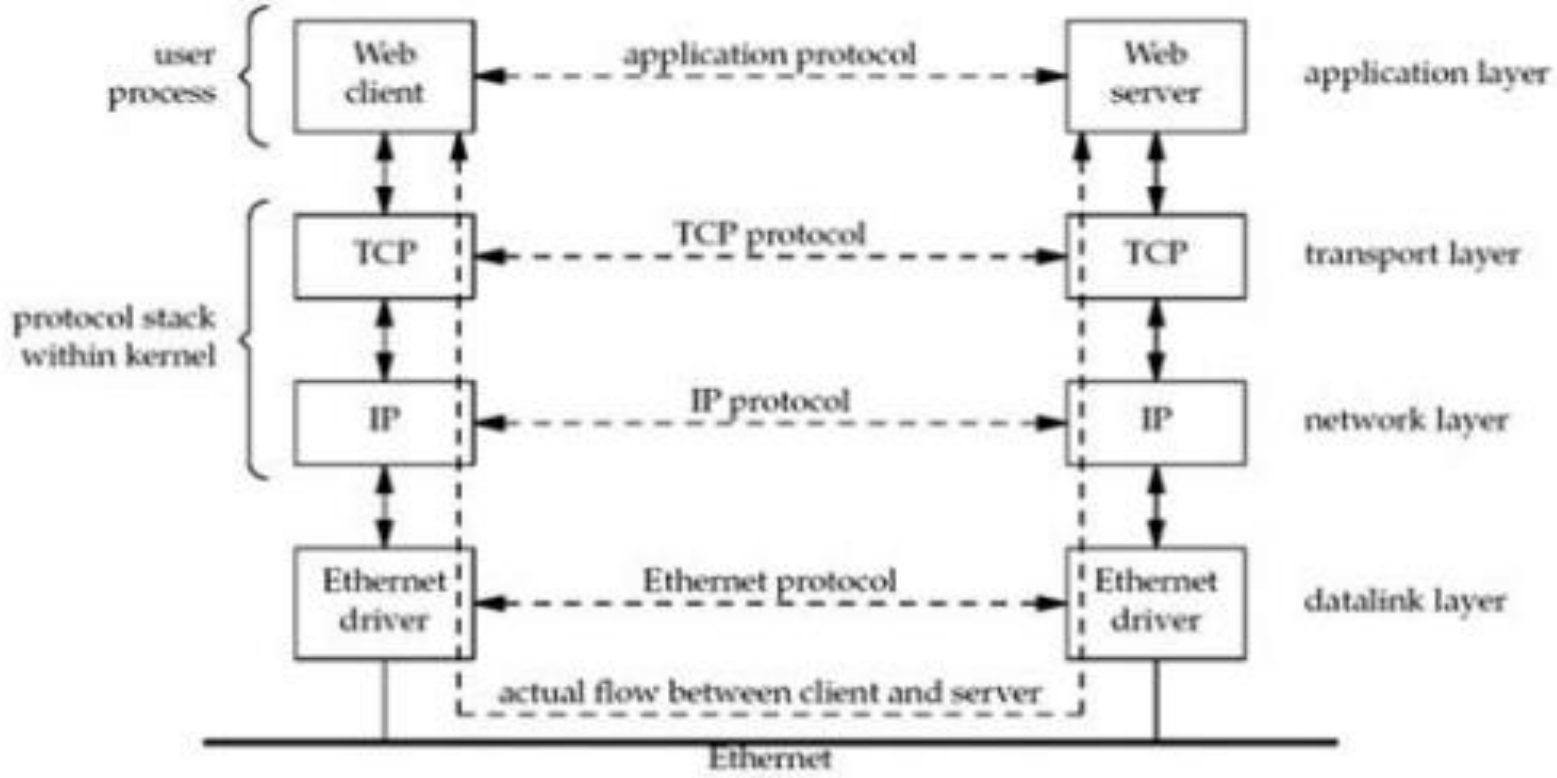
- Clients normally communicate with one server at a time, although using a Web browser , communication might happen with many different Web servers over, in specific time period.
- The client and the server application may be thought of as communicating via a network protocol, but actually, multiple layers of network protocols are typically involved.



Internet protocol suite

- Web clients and servers communicate using the Transmission Control Protocol, or TCP. TCP, in turn, uses the Internet Protocol, or IP, and IP communicates with a datalink layer of some form. Below fig depicts when client and server are on the same Ethernet,

Figure 1.3. Client and server on the same Ethernet communicating using TCP.



- Client and server communicate using an application protocol. Transport layers communicate using TCP.
- Actual flow of information between the client and server goes down the protocol stack on one side, across the network, and up the protocol stack on the other side. Also client and server are typically user processes, while the TCP and IP protocols are normally part of the protocol stack within the kernel.
- Ethernet standards used in Ethernet networks. It uses UTP (Cat3 or higher) cables and Hubs. Hubs use a physical star topology and a logical bus topology.
- **Hubs** repeat and forward signals to all nodes.

Figure 1.4. Client and server on different LANs connected through a WAN.

Routers are the building blocks of WANs. The largest WAN today is the *Internet*. Many companies build their own WANs and these private WANs may or may not be connected to the Internet.

https://www.youtube.com/watch?v=1z0ULvg_pW8

Difference between Hub Switch and Router in tabular form:

HUB	SWITCH	ROUTER
Hub is a broadcast device.	The switch is a multicast device.	The router is a routing device.
Hub works in the physical layer of the OSI model.	The switch works in the data link layer and network layer of the OSI model.	The router works in the network layer of the OSI model.
Hub is used to connect devices to the same network.	The switch is used to connect devices to the network.	The router is used to connect two different networks.
Hub sends data in the form of bits.	The switch sends data in the form of frames.	The router sends data in the form of packets.
Hub works in **half-duplex**.	The switch works in **full-duplex**.	The router works in full-duplex.
Only one device can send data at a time.	Multiple devices can send data at a time.	Multiple devices can send data at a time.
Hub does not store any MAC address of a node in the network.	Switch stores the **IP Address** and **MAC address** of **nodes** used in a network.	Router stores the IP Address and MAC address of nodes used in a network.

Types of protocols

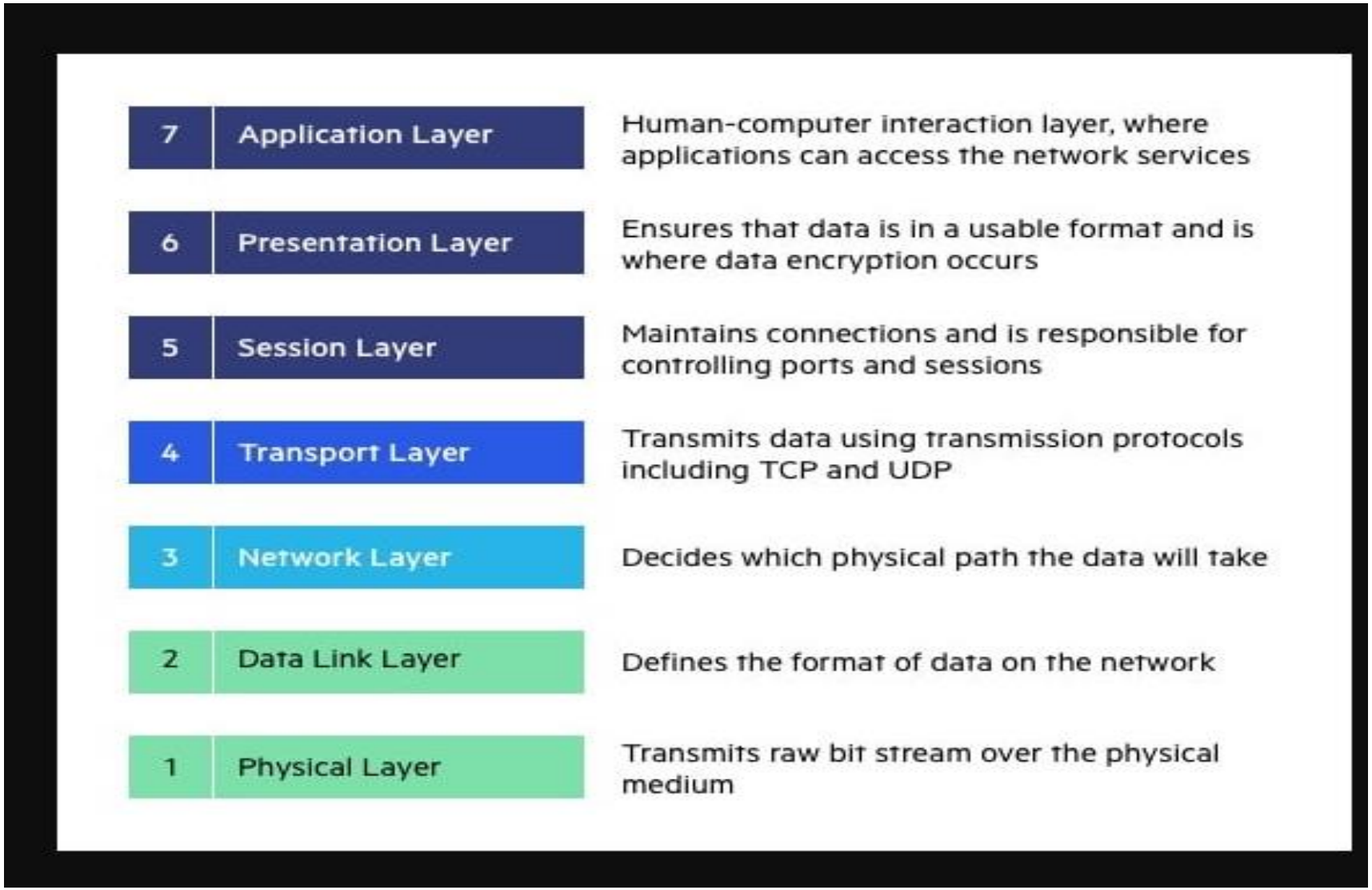
- Protocols can be broadly classified into three major categories-
- **Communication** : These protocols formally set out the rules and formats through which data is transferred. These protocols handle syntax, semantics, error detection, synchronization, and authentication.

  Eg : **HTTP, TCP, UDP, BGP, ARP, IP, DHCP,**
- **Management** : These protocols assist in describing the procedures and policies that are used in monitoring, maintaining, and managing the computer network. These protocols also help in communicating these requirements across the network to ensure stable communication.

  Eg: **FTP, Telnet**
- **Security:** These protocols secure the data in passage over a network. These protocols also determine how the network secures data from any unauthorized attempts to extract or review data.

  Eg: **SSL, HTTPS, TSL**

Functions of OSI Layers



Types of Network Protocols with Functions

TCP and UDP protocols

- Both protocols allow network applications to exchange data between nodes. The main difference between both is that TCP is a connection-oriented protocol while UDP is a connectionless protocol.
- When the TCP protocol is used, a special connection is opened up between two network devices, and the channel remains open to transmit data until it is closed. On the other hand, a UDP transmission does not make a proper connection and merely broadcasts its data to the specified network address without any verification of receipt.

IP protocol

- Internet Protocol is **connectionless** and **unreliable** protocol. It ensures no guarantee of successfully transmission of data.
- In order to make it reliable, it must be paired with reliable protocol such as TCP at the transport layer.
- Internet protocol transmits the data in form of a datagram.

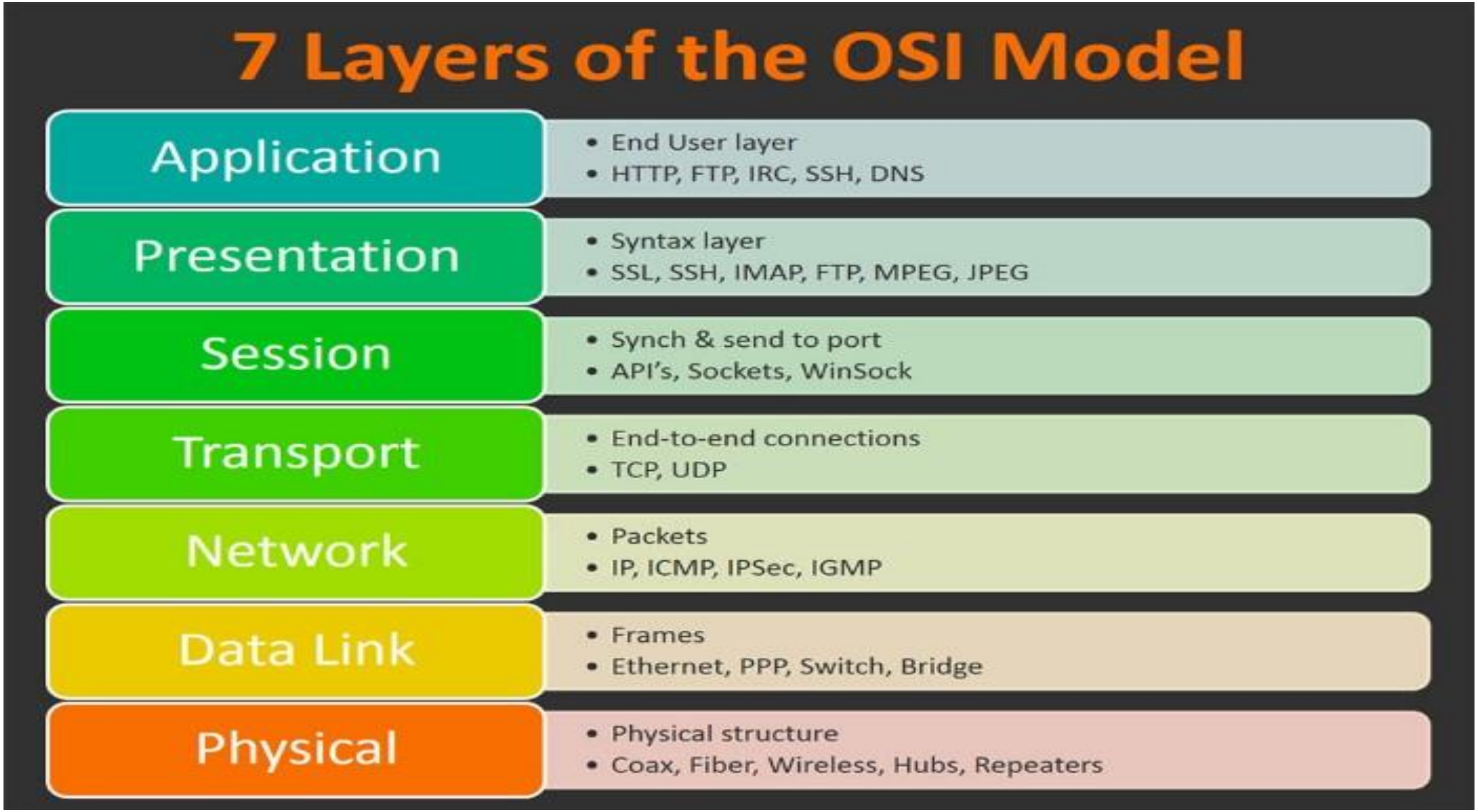
Secure SHell (SSH)

- It is a cryptography-based network protocol for operating network services securely and reliably over an unsecured network. Some particular applications include remote command-line, remote command execution, login, but any network service can be made secure with the help of SSH.

Continued…

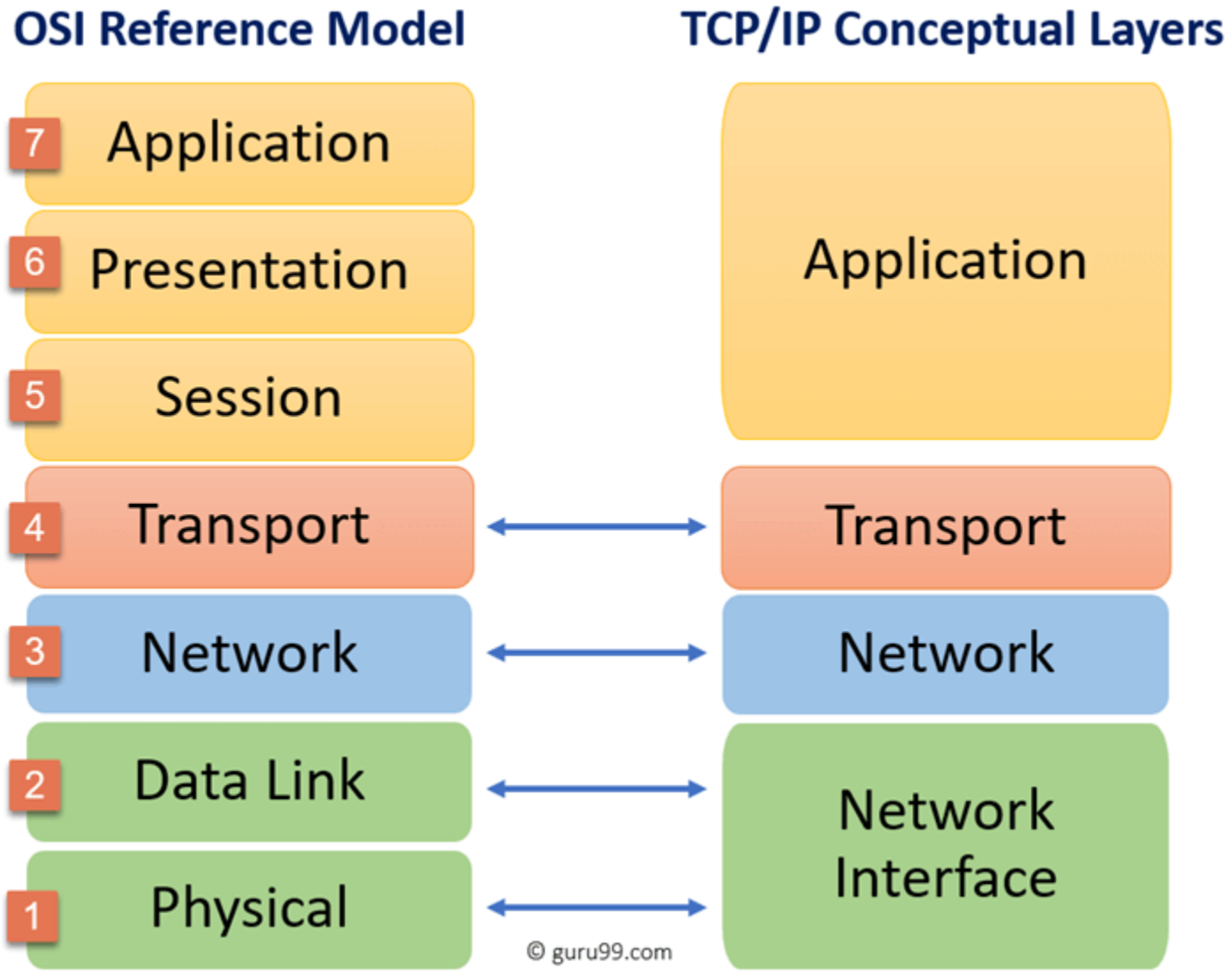
- **Internet Relay Chat (IRC) Protocol**
- uses a client-server model to provide a chatroom.
- A single IRC server is set up, and users connect to the server via IRC clients. The protocol allows users to set usernames on the server and engage in private chats or group chats via different IRC channels.
- **Internet Message Access Protocol (IMAP)**
- It is an Internet email protocol that stores emails on the mail server but allows the end-user to retrieve, see, and manipulate the messages as they were stored locally on the user's devices.
- **Domain Name Service Protocol(DNS)**
- DNS stands for Domain Name Service. This service allows us to access a node by its name. By default, nodes use IP addresses to identify each other on the network. DNS service allows us to map a name to an IP address. When we access a node by its name, the DNS service translates the name into the IP address. Let's take an example.

Protocols associated with all the Layers of OSI - Model



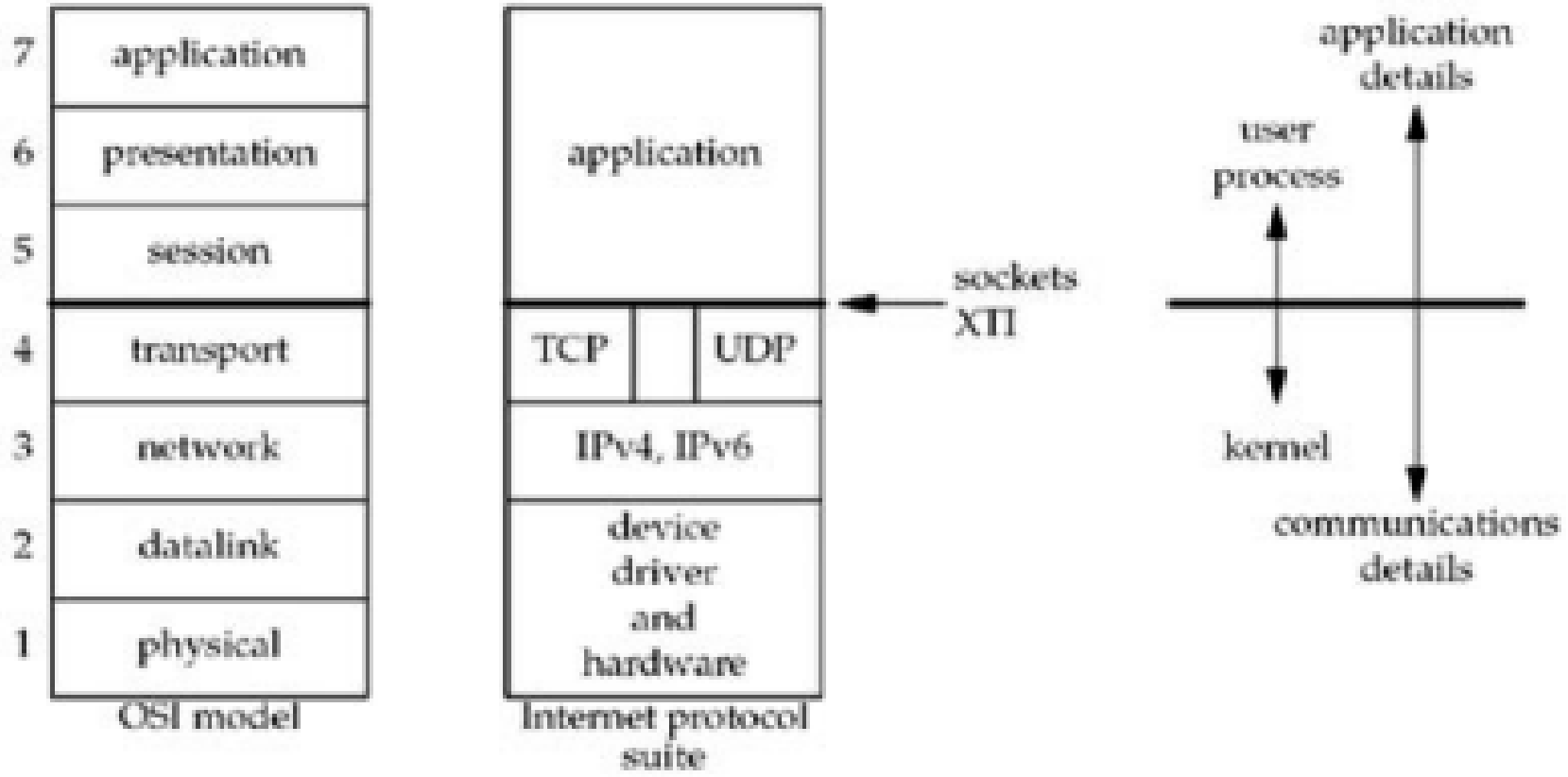
- The **TCP/IP model** is a concise version of the OSI model. It contains four layers, unlike seven layers in the OSI model. The layers are:
- Process/Application Layer
- Host-to-Host/Transport Layer
- Internet Layer
- Network Access/Link Layer

TCP/IP Model


OSI Reference Model
TCP/IP Conceptual Layers
7 Application
6 Presentation
5 Session
4 Transport
3 Network
2 Data Link
1 Physical
Application
Transport
Network
Network Interface
© guru99.com

Need of Socket Programming: The sockets programming interfaces are the interfaces from the upper three layers (the "application") into the transport layer. There are 3 reasons why sockets provide interface from the upper three layers of the OSI model into the Transport Layer.

Figure 1.14. Layers in OSI model and Internet protocol suite.



BSD

BSD

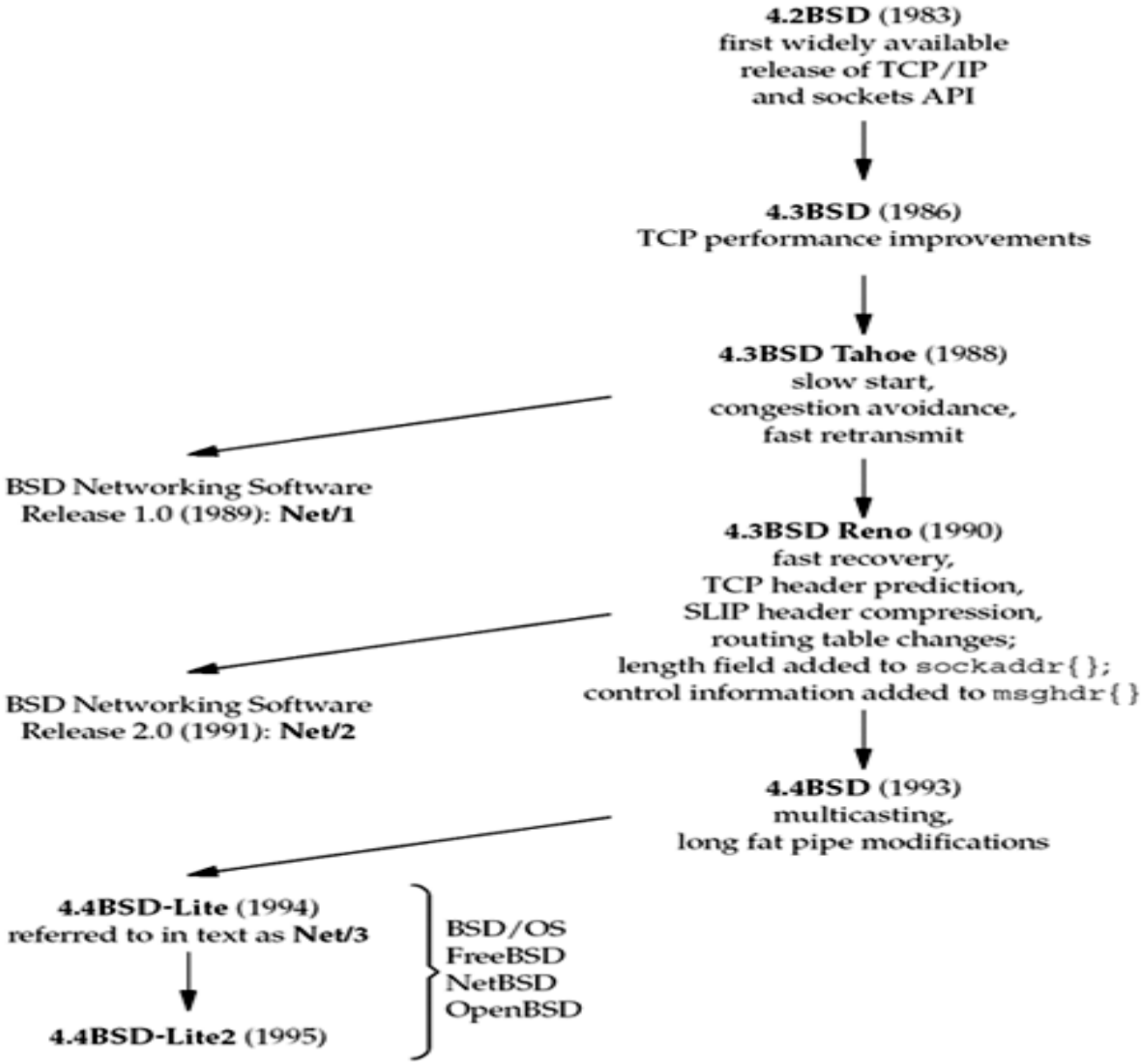
Berkeley Software Distribution (BSD, sometimes called Berkeley Unix) is the UNIX operating system derivative developed and distributed by the Computer Systems Research Group (CSRG) of the University of California, Berkeley, from 1977 to 1995.

BSD

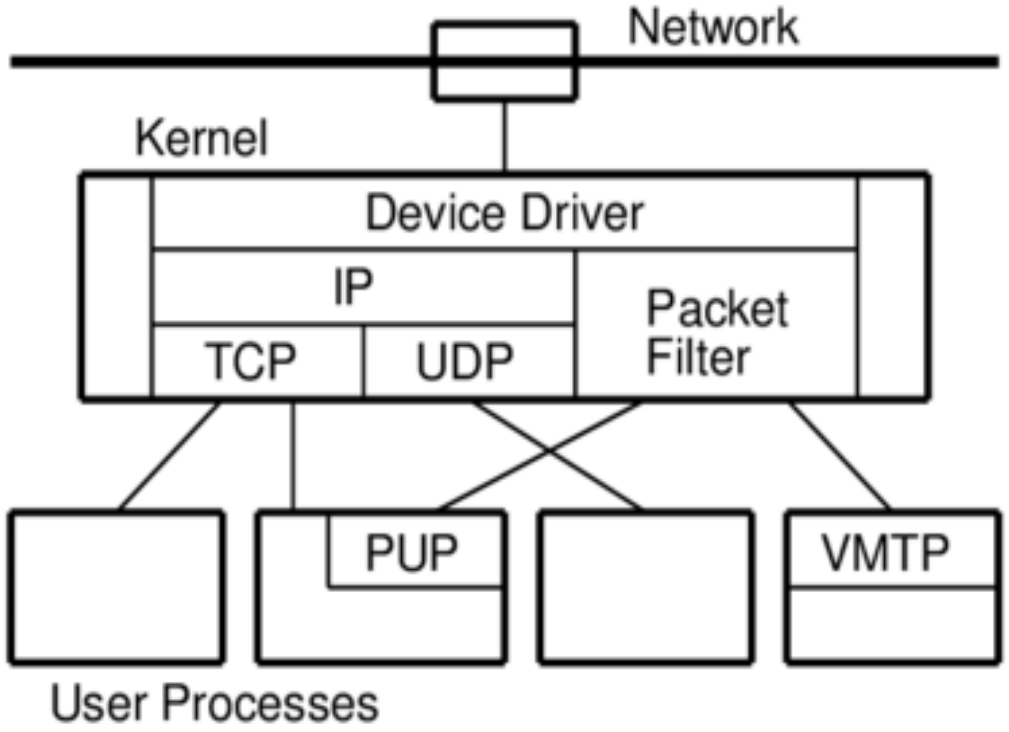
The History of BSD

- 1977 - Bill Joy puts together 1st Berkeley Software Distribution (Version 1)
- mid-1978 - 2BSD released with improved Pascal, termcap, vi (about 75 shipped)
- 1978 - Berkeley obtains a VAX-11/780
- A copy of AT&T 32/V UNIX is installed - does not take advantage of virtual memory

BSD History


4.2BSD (1983)
first widely available
release of TCP/IP
and sockets API
4.3BSD (1986)
TCP performance improvements
4.3BSD Tahoe (1988)
slow start,
congestion avoidance,
fast retransmit
BSD Networking Software
Release 1.0 (1989): Net/1
4.3BSD Reno (1990)
fast recovery,
TCP header prediction,
SLIP header compression,
routing table changes;
length field added to sockaddr{};
control information added to msghdr{}
BSD Networking Software
Release 2.0 (1991): Net/2
4.4BSD (1993)
multicasting,
long fat pipe modifications
4.4BSD-Lite (1994)
referred to in text as Net/3
4.4BSD-Lite2 (1995)
BSD/OS
FreeBSD
NetBSD
OpenBSD

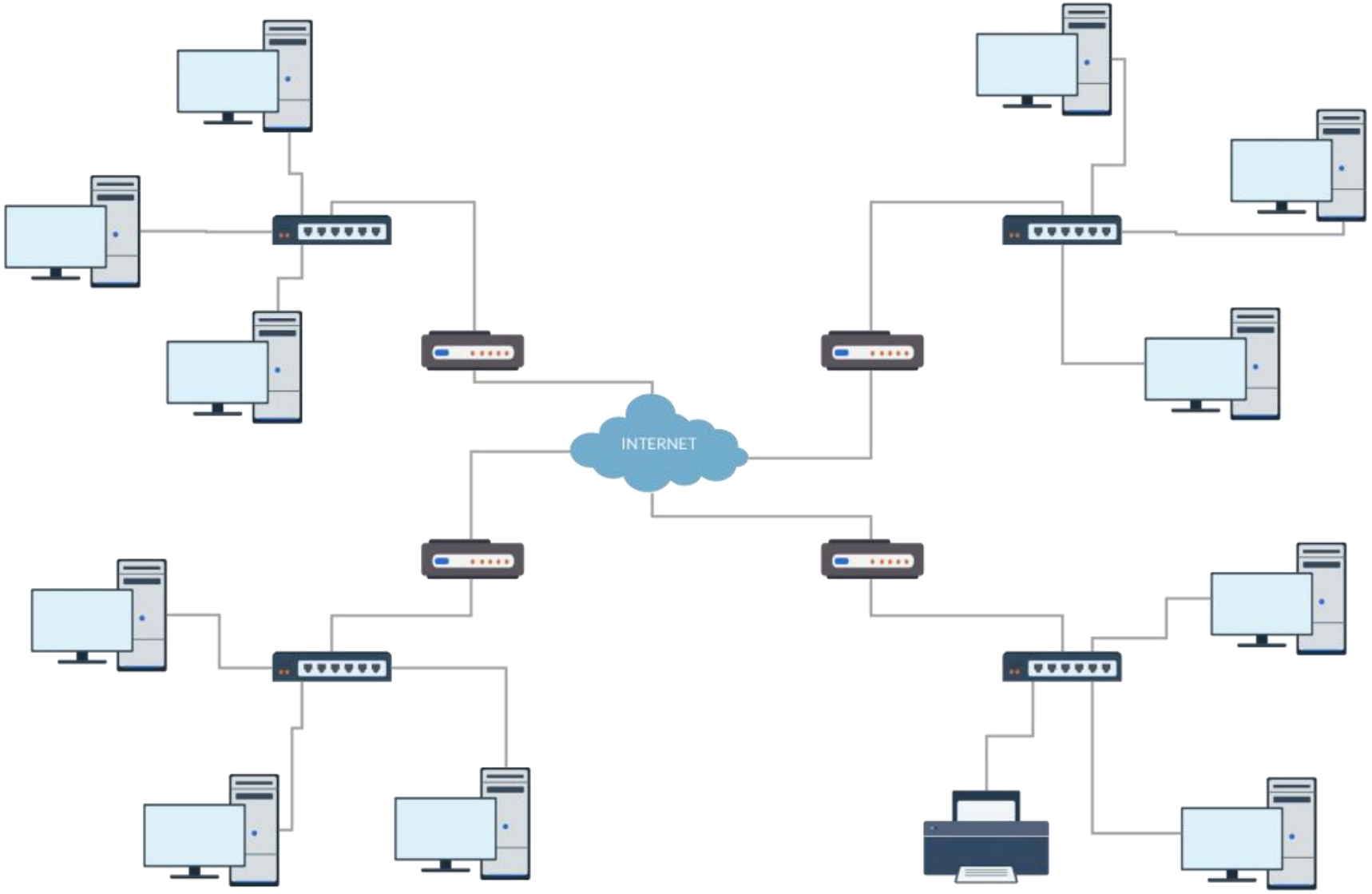
BSD Network



Potentially Unwanted Applications

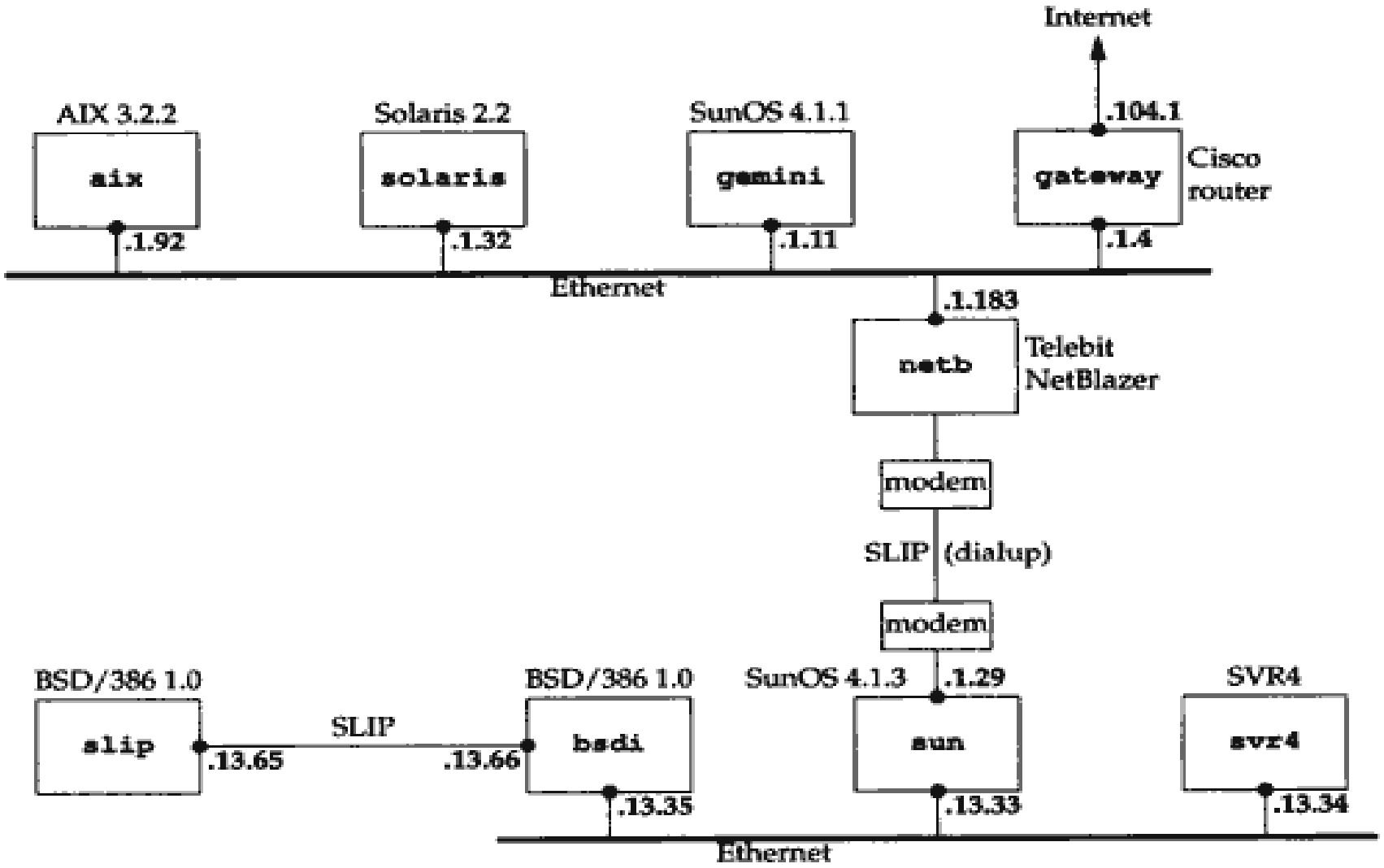
Versatile Message Transaction Protocol

Test Network and Hosts

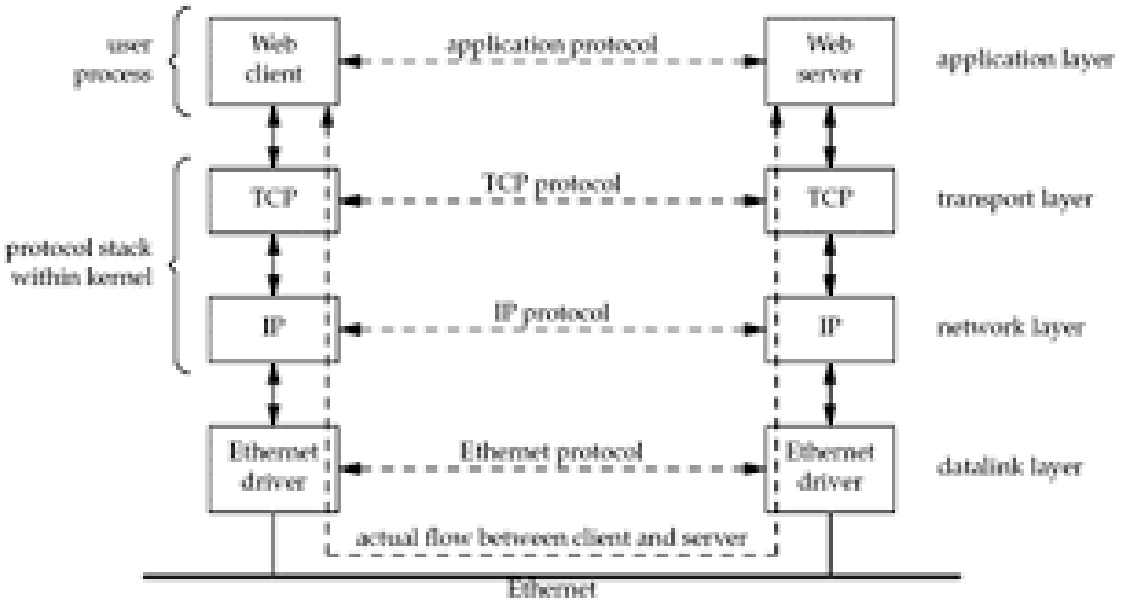


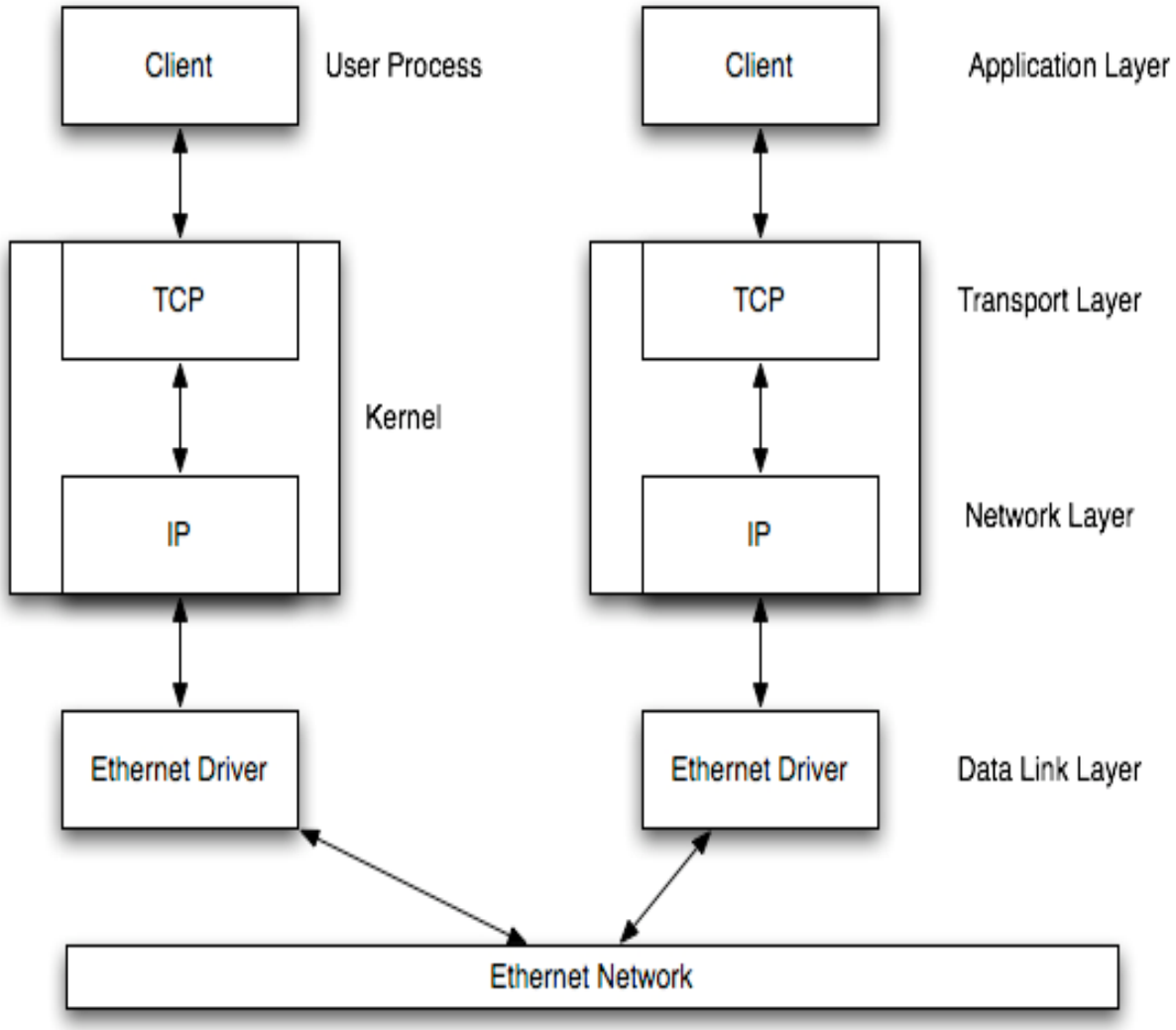
- For each host, the OS and the type of hardware (since some of the operating systems run on more than one type of hardware) is shown. The name within each box is the hostname that appears in the text.
- Machines are largely spread out across the Internet and virtual private networks (VPNs) or secure shell (SSH) connections provide connectivity between these machines regardless of where they live physically.
- Although there are no current Unix standards with regard to network configuration and administration, two basic commands are provided by most Unix systems and can be used to discover some details of a network netstat and ifconfig.

Test Network and Hosts

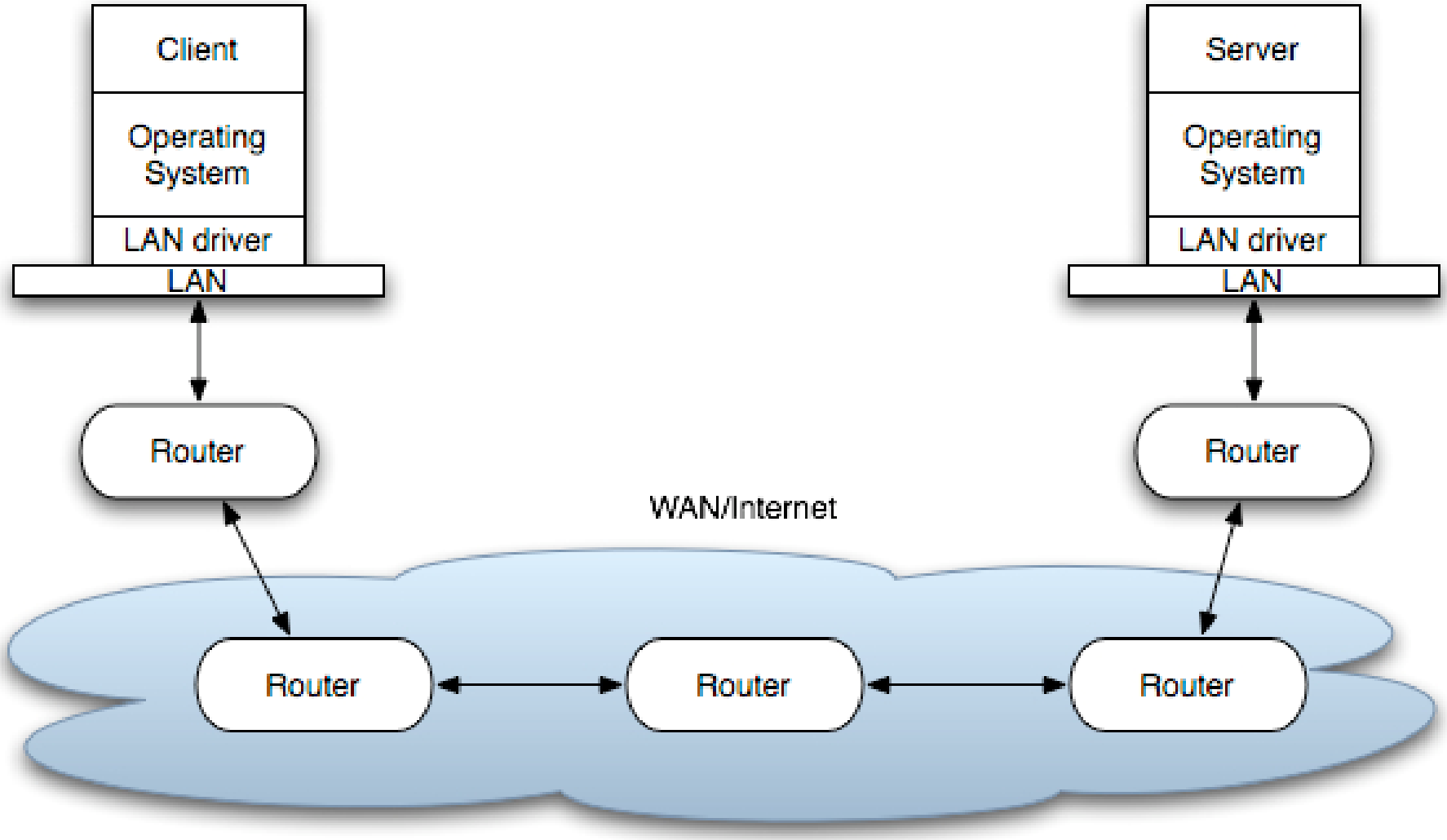


Communication over LAN

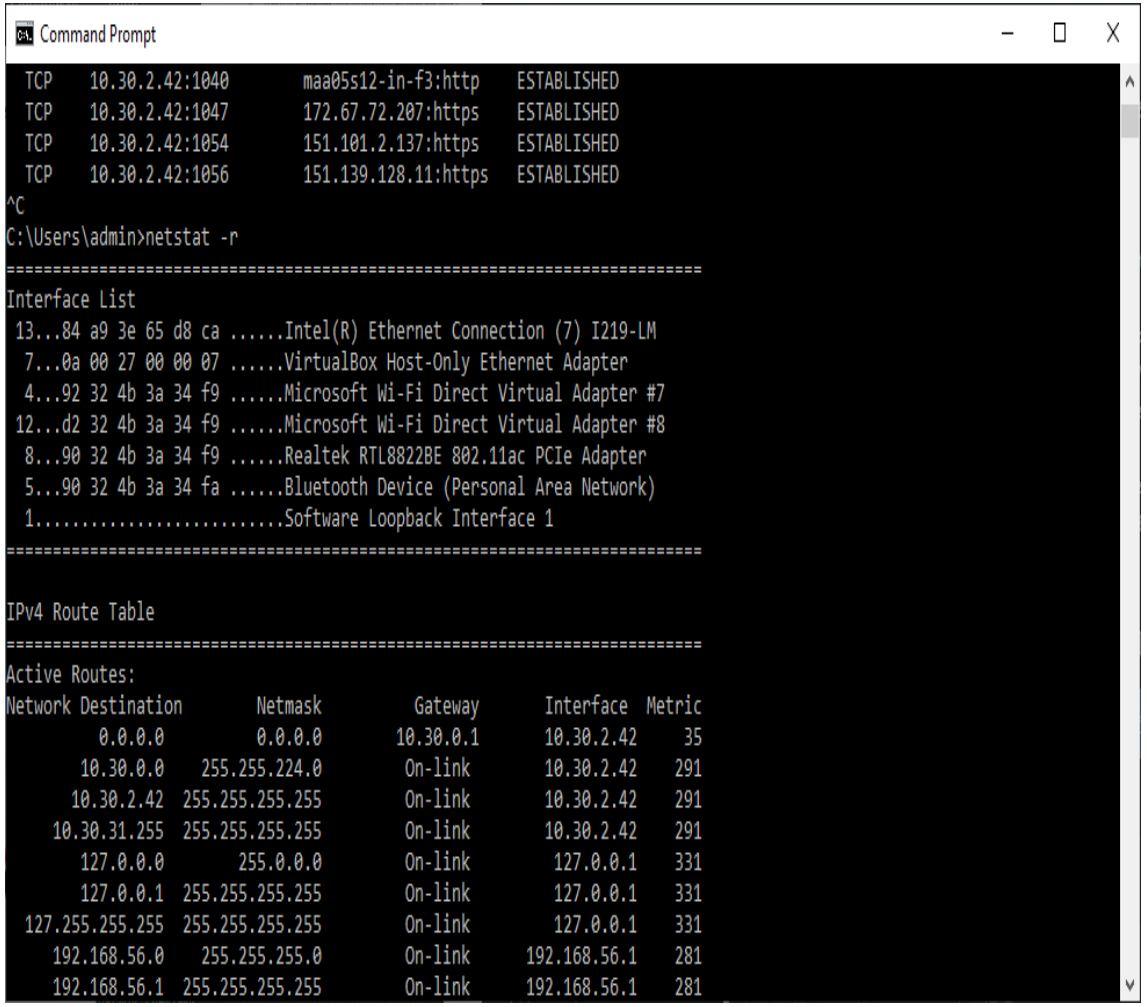




Communication over WAN



Discovering Network Topology – netstat –ni and netstat -r



Options	Meaning
-a	All (TCP, UDP, SCTP, ICMP)
-n	Numeric addresses
-b	Display executables
-o	Process id
-f	Fully Qualified domain name
-p proto	Specific protocols
-r	Routing table
-s	Protocol statistics
-t	Current connection network status

- The netstat command, meaning **network statistics**, is a Command Prompt command used to display very detailed information about how your computer is communicating with other computers or network devices.
- # netstat –a : Shows both listening and non-listening sockets.
- # netstat –l : List only listening ports.
- # netstat –lt : List only listening TCP ports.
- # netstat –c : Print the netstat information continuously every few seconds.

Unix Standard - POSIX

THE POSIX STANDARDS

Posix.1 : IEEE 1003.1-1990 adapted by ISO as ISO/IEC 9945:1:1990 standard
***gives standard for base operating system API**

Posix.1b : IEEE 1003.4-1993
*** gives standard APIs for real time operating system interface including interprocess communication**

Posix.1c : Threads and Extensions

Table 1. List of POSIX Base Standards

POSIX.1	System Interface (basic reference standard)[a,b]
POSIX.2	Shell and Utilities[a]
POSIX.3	Methods for Testing Conformance to POSIX[a]
POSIX.4	Real-time Extensions
POSIX.4a	Threads Extensions
POSIX.4b	Additional Real-time Extensions
POSIX.6	Security Extensions
POSIX.7	System Administration
POSIX.8	Transparent File Access
POSIX.12	Protocol Independent Network Interfaces
POSIX.15	Batch Queuing Extensions
POSIX.17	Directory Services

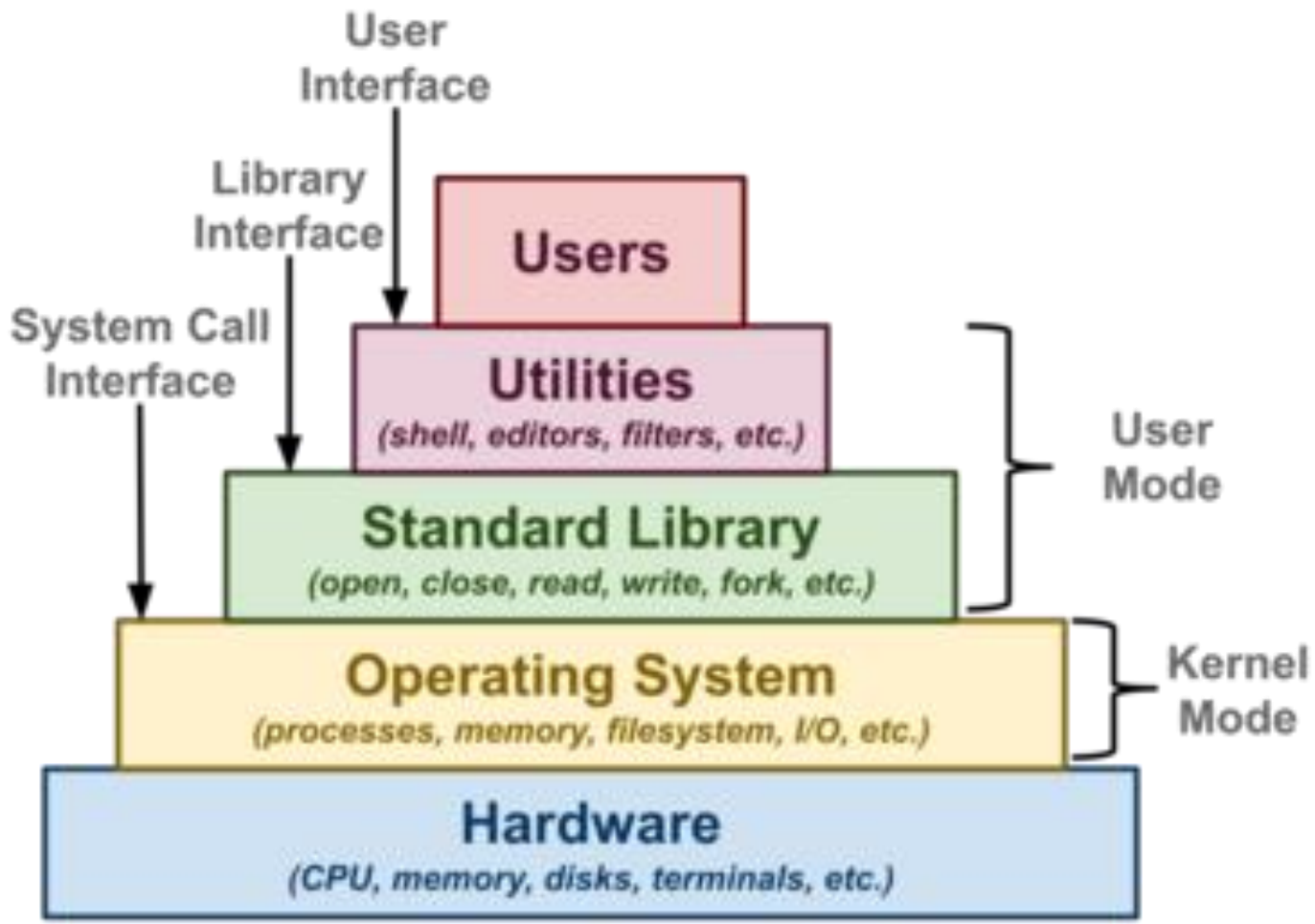
[a] Approved IEEE standards
[b] Approved ISO/IEC standard

Unix Standard - POSIX

What does POSIX mean?

- **P**ortable **O**perating **S**ystem **I**nterface for Unix" is the name of a family of related standards specified by the IEEE to define the application programming interface (API), along with shell and utilities interfaces, for software compatible with variants of the UNIX operating system, although the standard can apply to any operating system.

Unix APIs

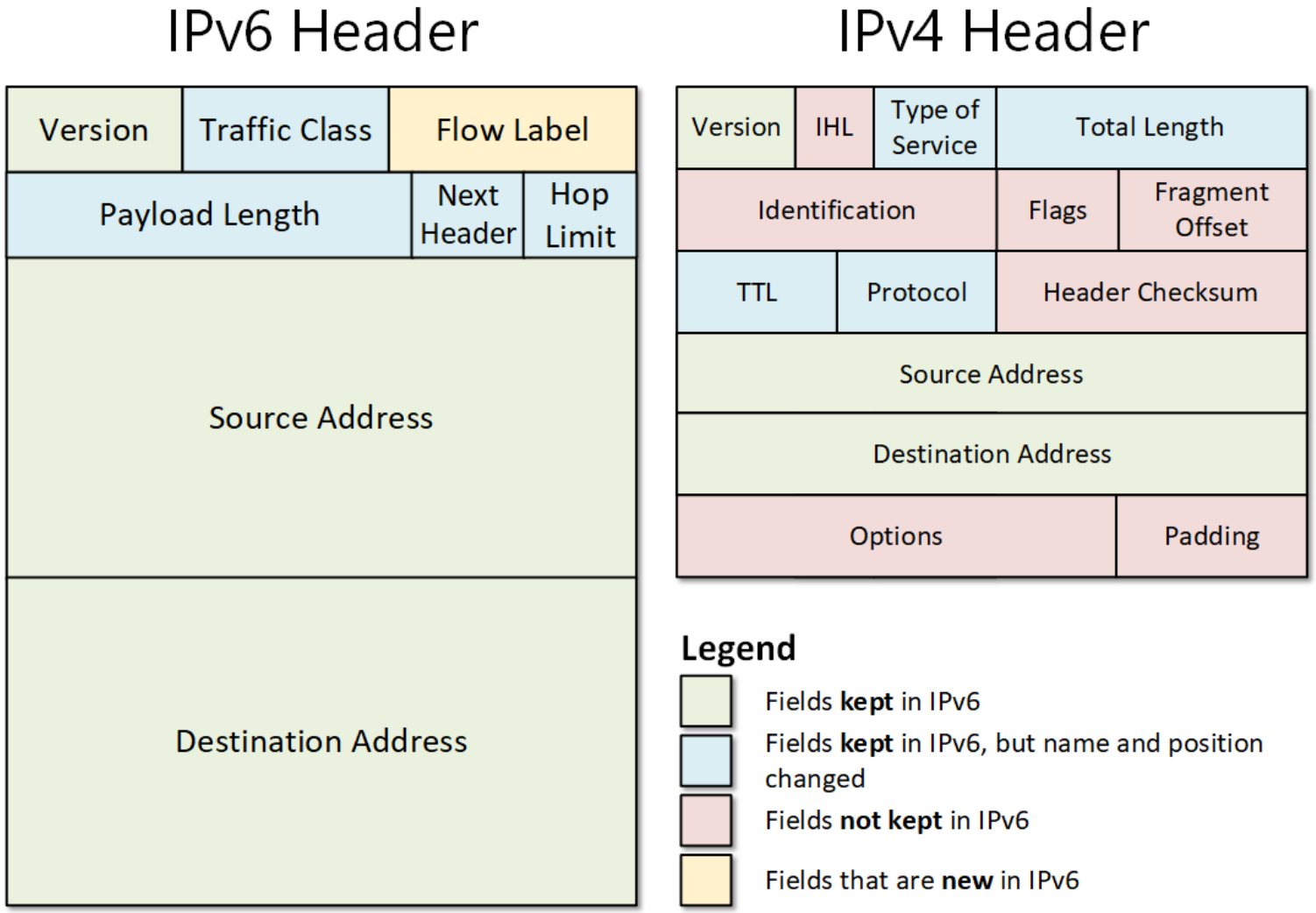

User Interface
Library Interface
System Call Interface
Users
Utilities
(shell, editors, filters, etc.)
Standard Library
(open, close, read, write, fork, etc.)
Operating System
(processes, memory, filesystem, I/O, etc.)
Hardware
(CPU, memory, disks, terminals, etc.)
User Mode
Kernel Mode

64 Bit Architectures : The common programming model for existing 32-bit Unix systems is called the **ILP32** model, that denotes,
integers (I), long integers (L), and pointers (P) occupy 32 bits.

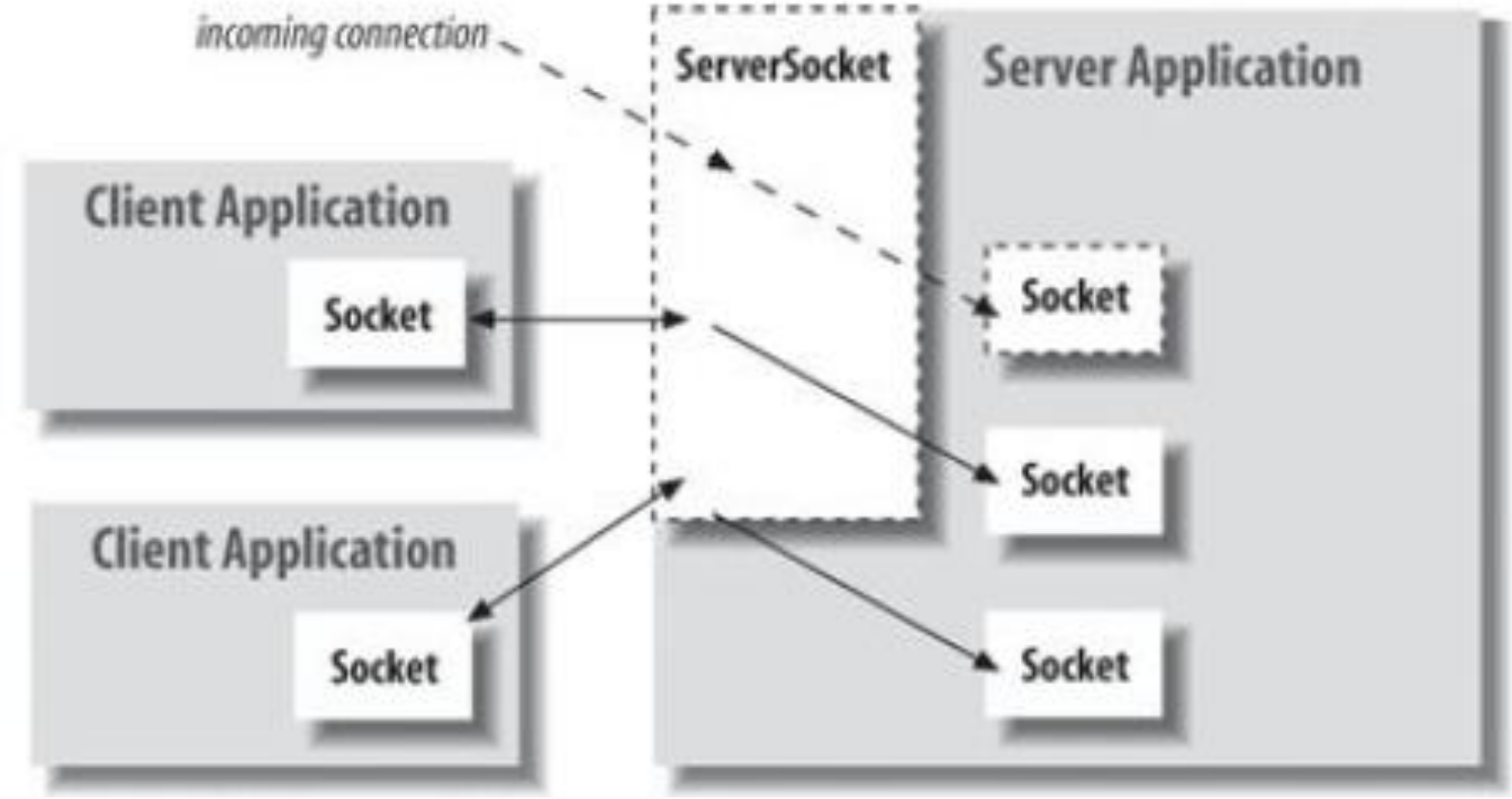
The model for 64-bit Unix systems is called the **LP64** model, that denotes,
long integers (L) and pointers (P) require 64 bits.
Figure below compares these two models.

Datatype	ILP32 model	LP64 model
char	8	8
short	16	16
int	32	32
long	32	64
pointer	32	64

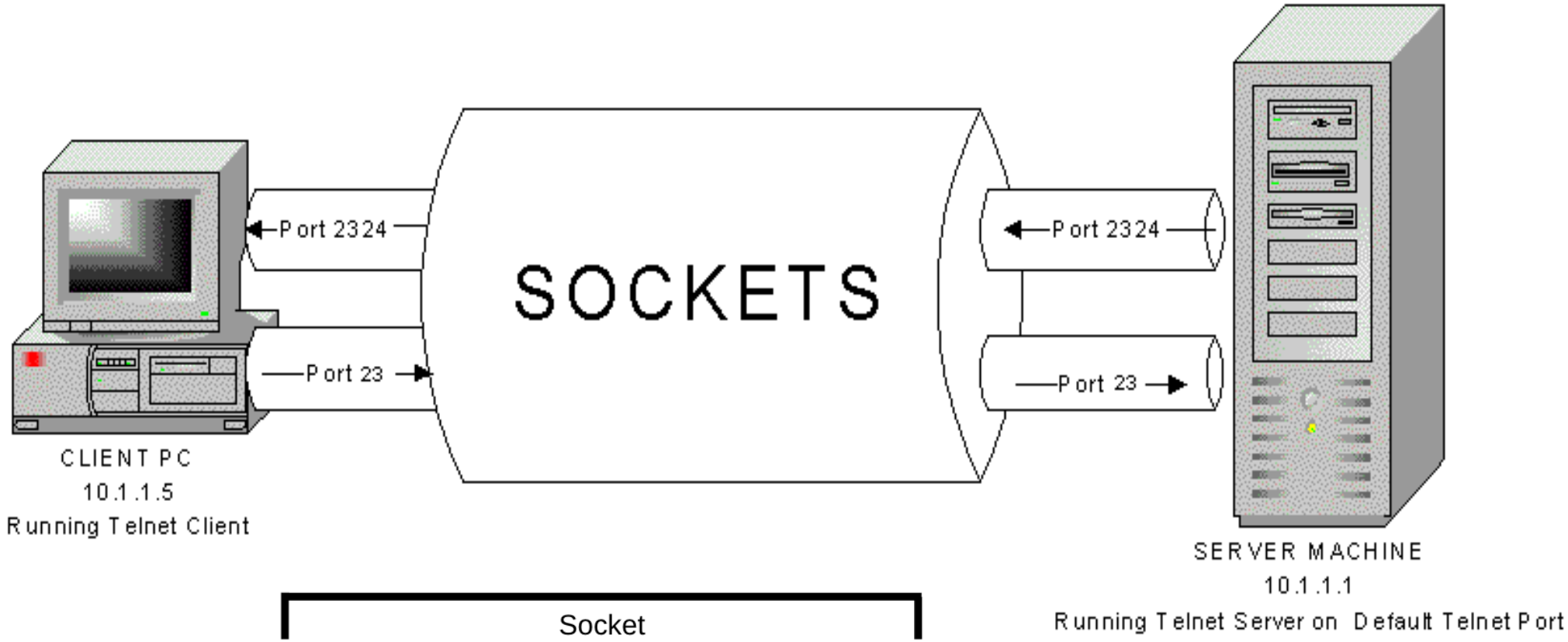
64 Bit Architectures

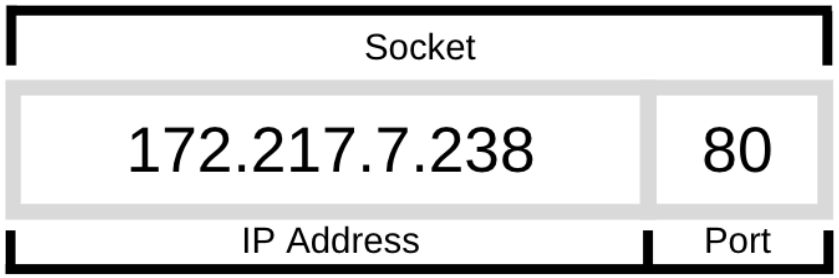


Sockets : An end point for communication between processes across the network



Sockets : An end point for communication between processes across the network

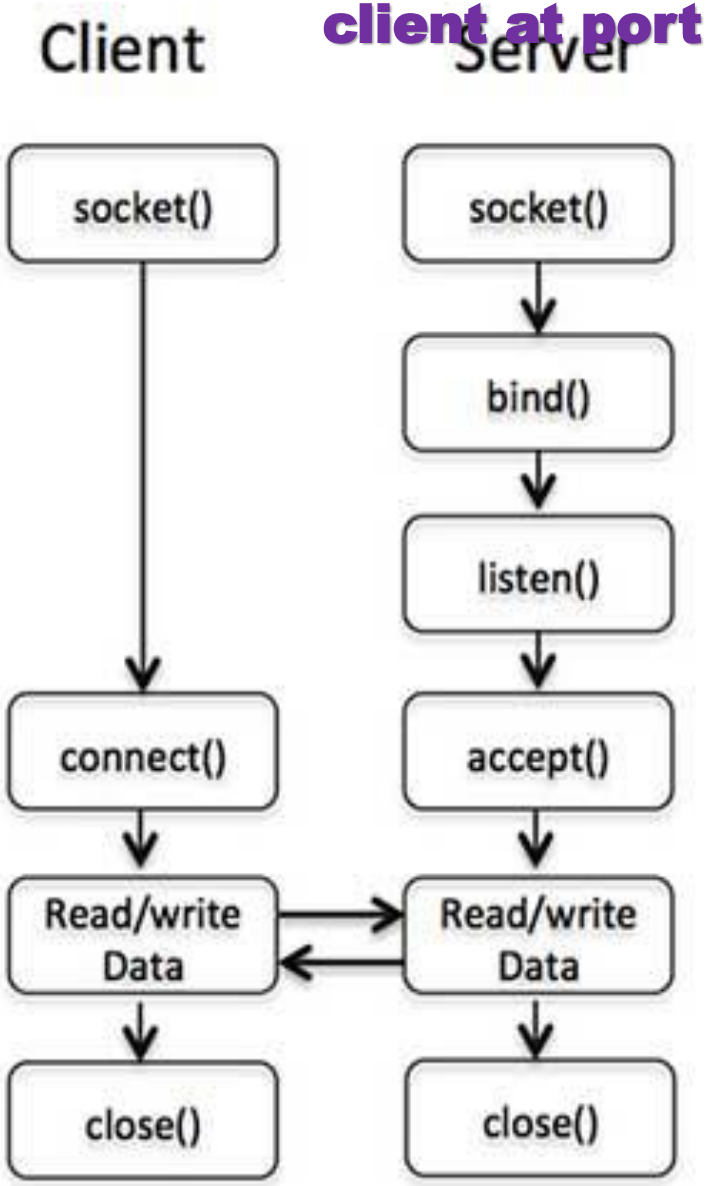


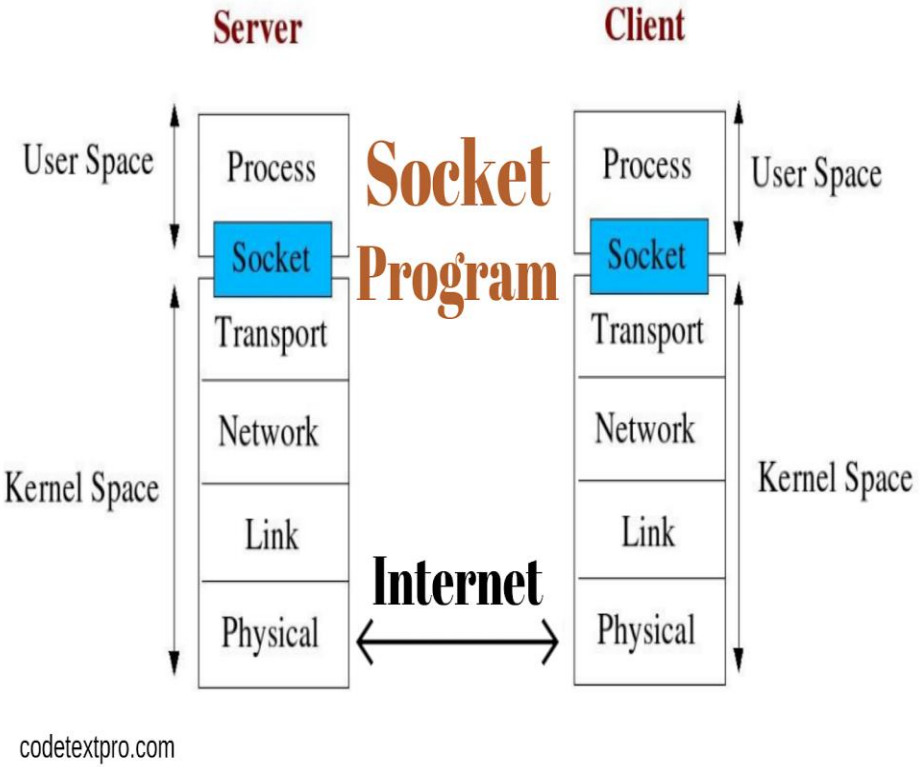


Example :

Day Time Client....

Connection between Day Time Server and client at port no. 13



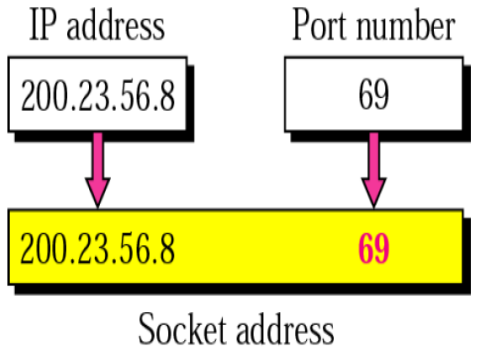


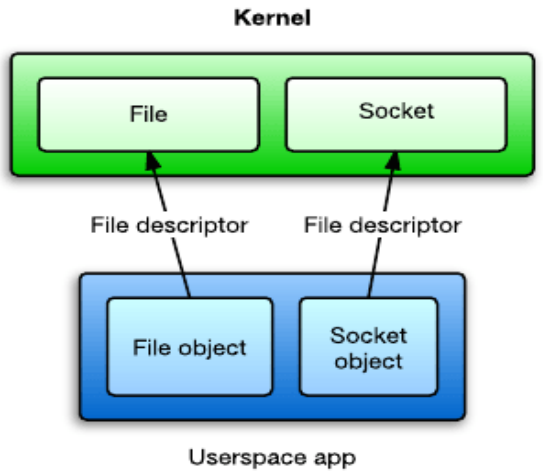
Source Code of Day Time Client

```
#include <unp.h>  // includes system headers & constants

int main(int argc, char **argv)
{
   int sockfd, n = 0;
   char recvline[MAXLINE + 1];
   struct sockaddr_in servaddr;
            if(argc != 2)
              err_quit("usage: a.out <Ipaddress>");
            if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
                        err_sys( "Socket Error");
   }

   bzero(&servaddr, sizeof(servaddr));
   servaddr.sin_family = AF_INET;
   servaddr.sin_port = htons(13); //Daytime server
    if( inet_pton(AF_INET,argv[1],&servAddress.sin_addr <= 0)
    err_quit(" inet_pton error for %s", argv[1]);
   if (connect(sockfd, (SA *) &servaddr, sizeof(servaddr)) < 0) {
                        err_sys( "Connect Error");
   }
```





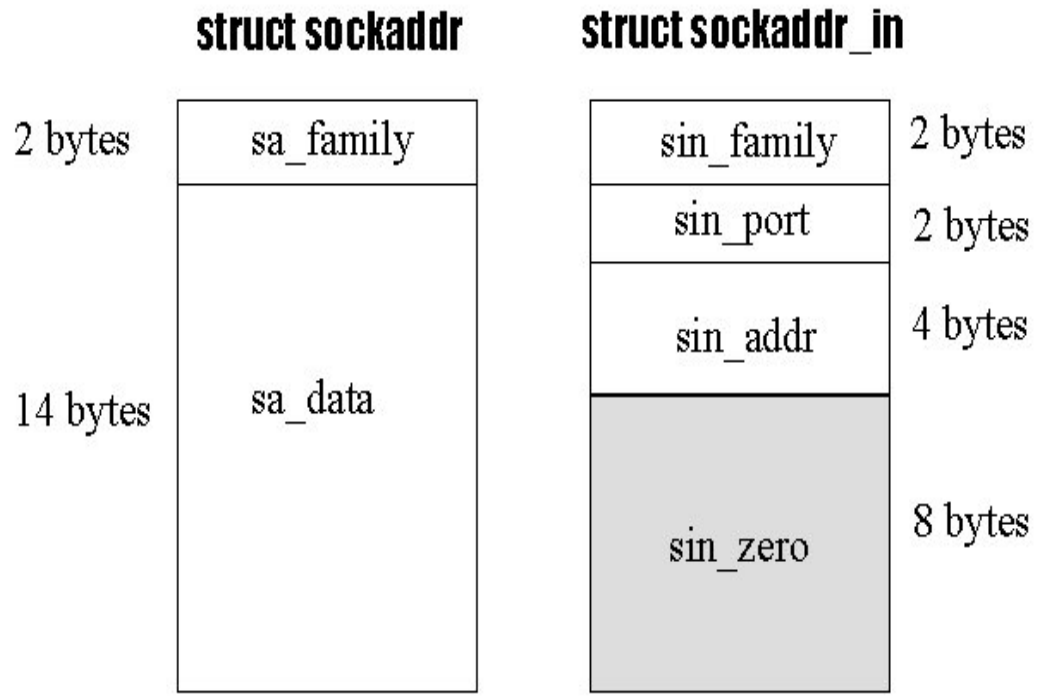
AF_APPLETALK	Apple Computer Inc. Appletalk network
AF_INET	Internet domain
AF_PUP	Xerox Corporation PUP internet
AF_UNIX	Unix file system

127.0.0.1 -- > b'\x7f\x00\x00\x01'

```
while ((n = read(sockfd, recvline, MAXLINE)) > 0) {
                recvline[n] = 0;        //null terminate
                 if(fputs(recvline,stdout) == EOF)
                 err_sys("fputs error");
                 }
               if((n<0)
                  err_sys("read error");
                exit(0);
}
```

Input 206.168.112.96
Output Mon May 26 20:58:40 2003

sockaddr vs. sockaddr_in



A pointer to a **struct sockaddr_in** can be cast to a pointer to a **struct sockaddr** and vice-versa.

DayTime Server…

```
#include "unp.h“
#include <time.h>
int main(int argc, char **argv)
{
int listenfd, connfd;
struct sockaddr_in servaddr;
char buff[MAXLINE];
time_t ticks;
listenfd = Socket(AF_INET, SOCK_STREAM, 0);
bzeros(&servaddr, sizeof(servaddr));
Serveraddr.sin_family=AF_INET;
Serveraddr.sin_addr.s_addr=hton1(INADDR_ANY);   //IP address of client on any interface.
Serveraddr.sin_port=htons(13);
Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
Listen(listenfd, LISTENQ); // convert socket to listening socket.
// LISTENQ- A queue of No of client connections
// 3 steps to prepare listenfd are, socket, bind & listen.
```

```
for (;;) {

connfd = Accept(listenfd, (struct SA *) NULL, NULL);
//connected descriptor
ticks = time(NULL);

snprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));   // carriage return & line feed are appended

write(connfd, buff, strlen(buff));

close(connfd);

}

}
```

Listen on the Port for connections

Accept connection request from Client

Write to socket… { Serve the Client }

Source Code of Day Time Client – IPV6

```
}
int main()
{
int s;
struct sockaddr_in6 addr;

s = socket(AF_INET6, SOCK_STREAM, 0);
addr.sin6_family = AF_INET6;
addr.sin6_port = htons(5000);
inet_pton(AF_INET6, "::1", &addr.sin6_addr);
connect(s, (struct sockaddr *)&addr, sizeof(addr));


while ((n = read(sockfd, recvline, 1000)) > 0) {
                    recvline[n] = 0;
                    fputs(recvline, stdout);
                }



close(sockfd);
return 0;
}
```

Error Handling and Wrapper functions

In any real-world program, it is essential to check every function call for an error return. we check for errors from socket, inet_pton, connect, read, and fputs, and when one occurs, we call our own functions, err_quit and err_sys, to print an error message and terminate the program.

Most of the time, this is what we want to do. Occasionally, we want to do something other than terminate when one of these functions returns an error .

```
if ( (connfd = accept (listenfd, (SA *) &cliaddr, &clilen)) < 0) {
    if (errno == EINTR)
        continue;           /* back to for() */
    else
        err_sys("accept error");
}
```

Continued…

- Programs can be shortened by defining a wrapper function that performs the actual function call, tests the return value, and terminates on an error. The convention we use is to capitalize the name of the function, as shown below.

  sockfd = Socket(AF_INET, SOCK_STREAM, 0);

Many system calls will report the EINTR error code if a signal occurred while the system call was in progress. No error actually occurred, it's just reported that way because the system isn't able to resume the system call automatically. This coding pattern simply retries the system call when this happens, to ignore the interrupt.

wrapper function for the socket function

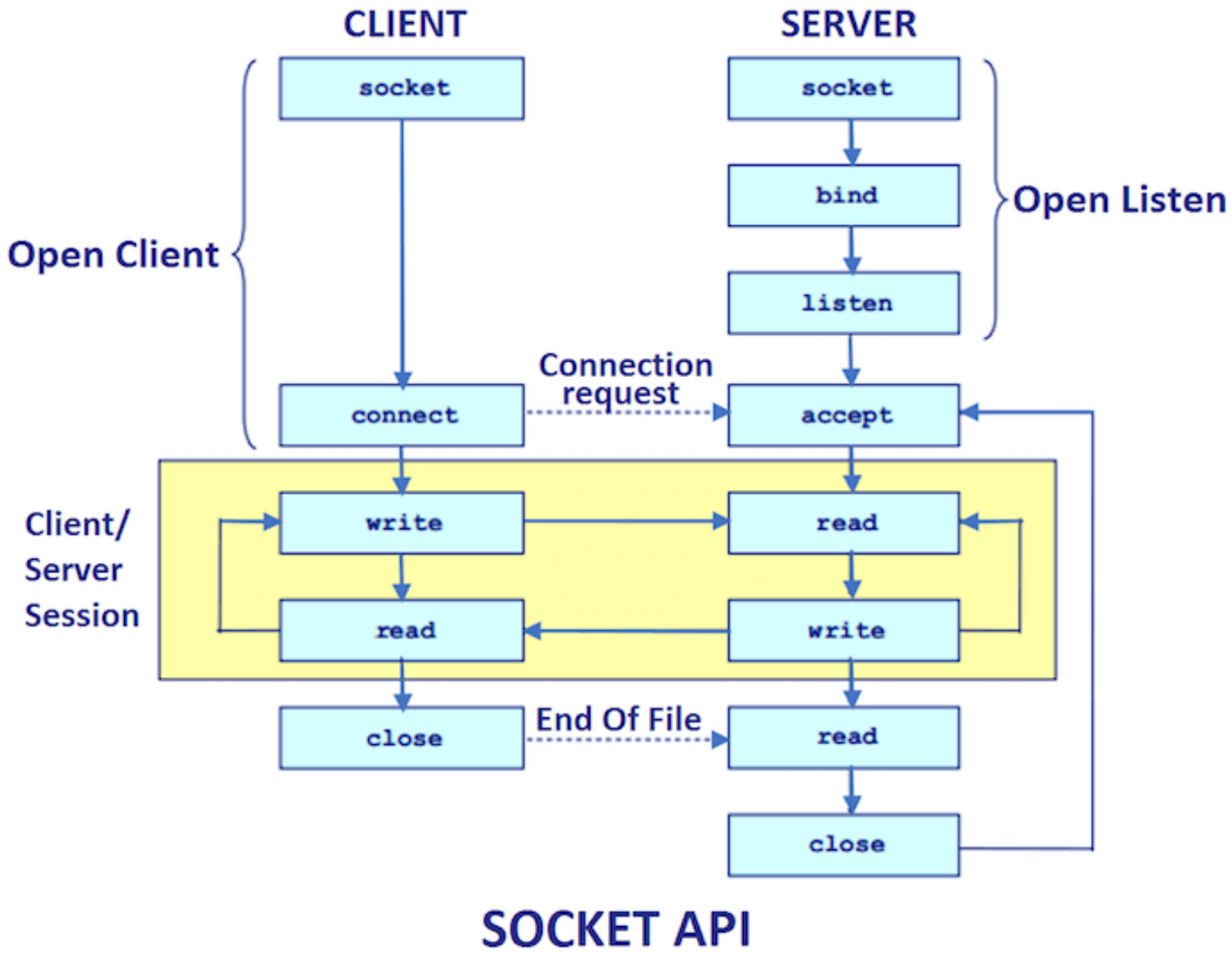
```
int Socket(int family, int type, int protocol)
{           int n;
            if((n=socket(family, type, protocol)) < 0)
            err_sys(“socket error);
             return(n);
}
```

Wrapper function for pthread_mutex_lock.

```
72 void
73 Pthread_mutex_lock(pthread_mutex_t *mptr)
74 {
75     int     n;

76     if ( (n = pthread_mutex_lock(mptr)) == 0)
77         return;
78     errno = n;
79     err_sys("pthread_mutex_lock error");
80 }
```

A ***mu****tual* ***ex****clusion object (mutex)* is a program object that allows multiple program threads to share the same resource, such as file access, but not simultaneously. When a program is started, a mutex is created with a unique name. After this stage, any thread that needs the resource must lock the mutex from other threads while it is using the resource. The mutex is set to unlock when the data is no longer needed or the routine is finished.


CLIENT
SERVER
socket
socket
bind
listen
Open Client
Open Listen
Connection request
connect
accept
Client/ Server Session
write
read
read
write
close
End Of File
read
close

SOCKET API

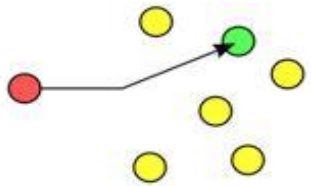
Transport Layer: TCP, UDP and SCTP

- SCTP(**Stream Control Transmission Protocol (SCTP))** is a reliable, message-oriented transport layer protocol. These transport protocols use the network-layer protocol IP, either IPv4 or IPv6. It maintains the message boundaries and detects the lost data, duplicate data as well as out-of-order data.
- SCTP provides the Congestion control as well as Flow control.
- UDP is a simple, unreliable datagram protocol.
- TCP is a sophisticated, reliable byte stream protocol.
- SCTP is similar to TCP as a reliable transport protocol, but it also provides message boundaries, transport-level support for multihoming, and a way to minimize head-of-line blocking.

TCP AND UDP

TCP

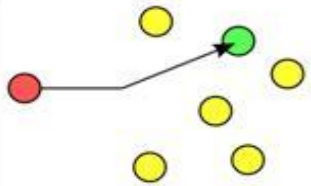
- Slower but reliable transfers
- Typical applications:
  - Email
  - Web browsing



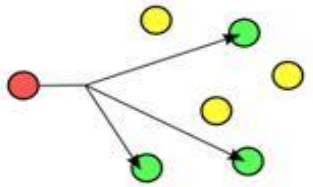
unicast

UDP

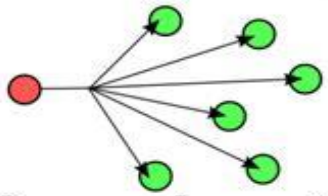
- Fast but non-guaranteed transfers ("best effort")
- Typical applications:
  - VoIP
  - Music streaming



unicast

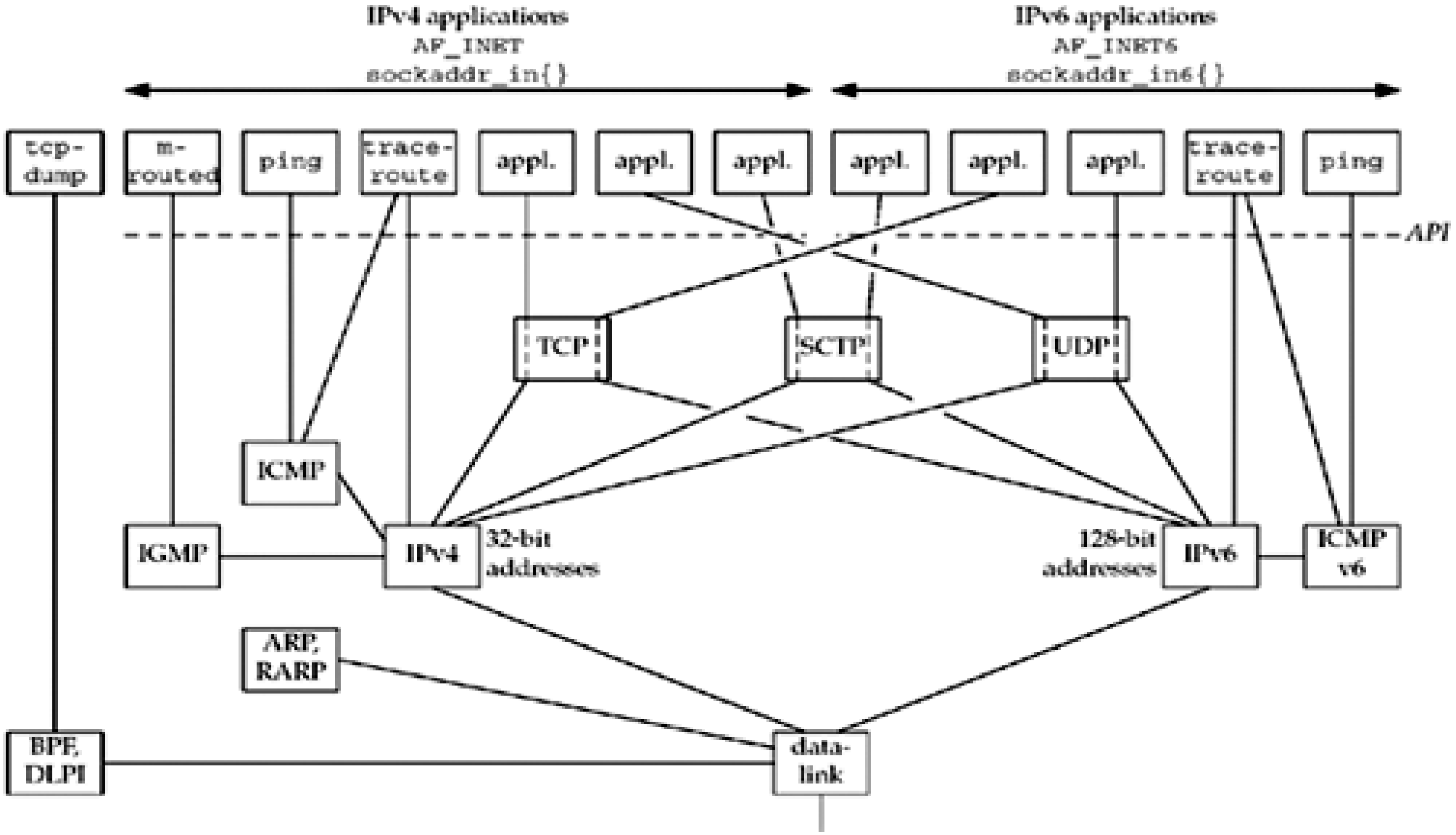


multicast

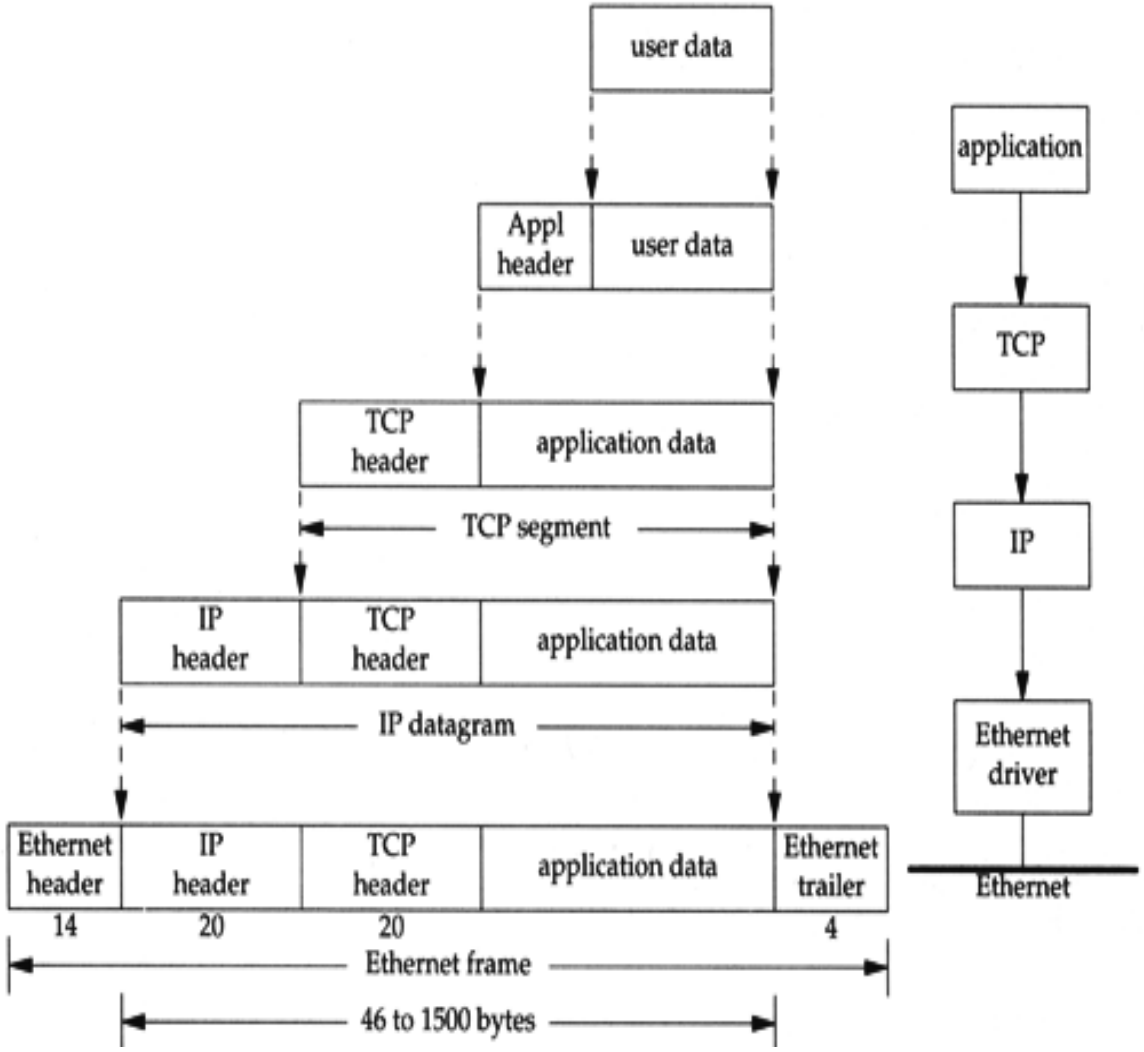


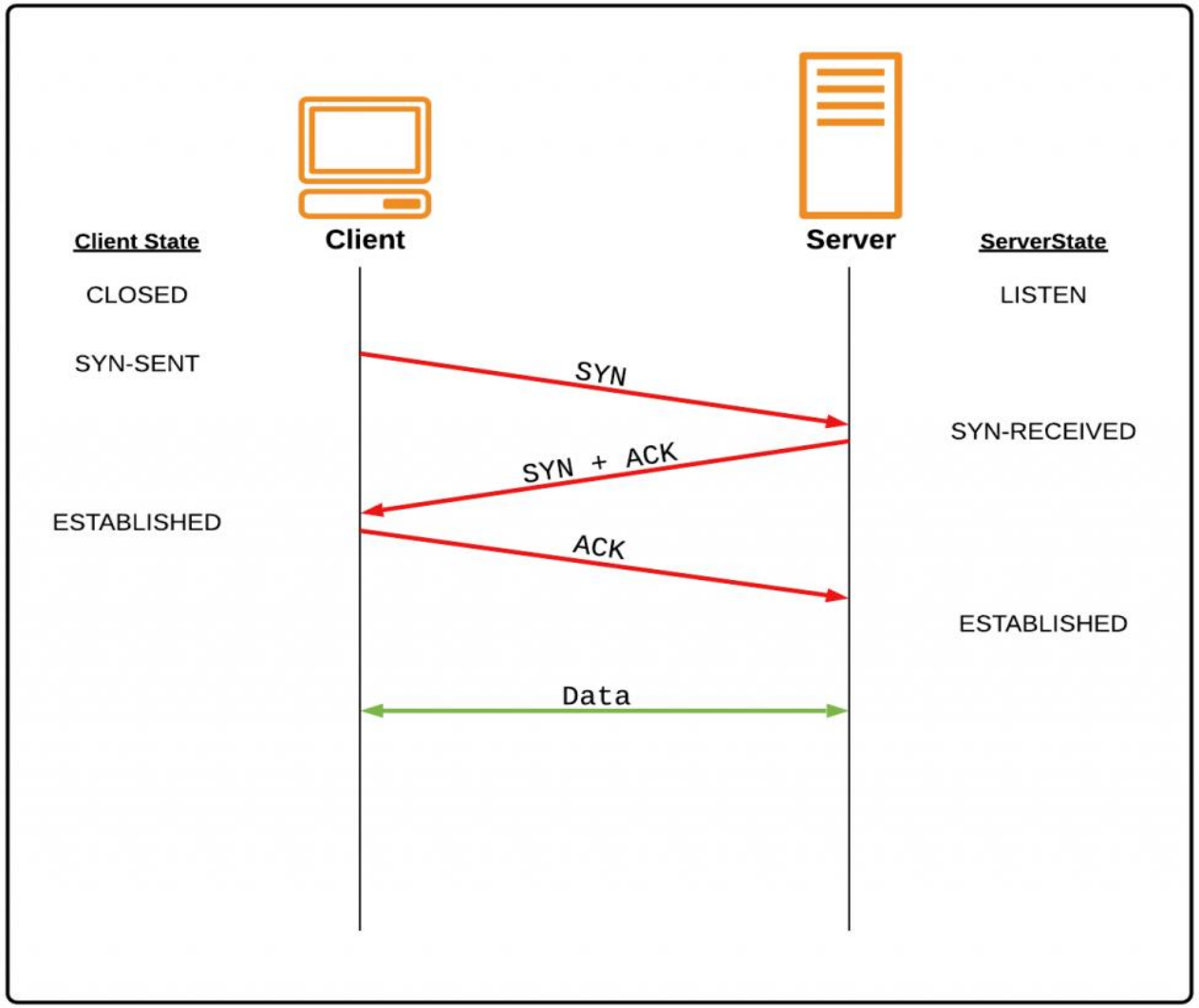
broadcast

TCP/IP – The Big-picture

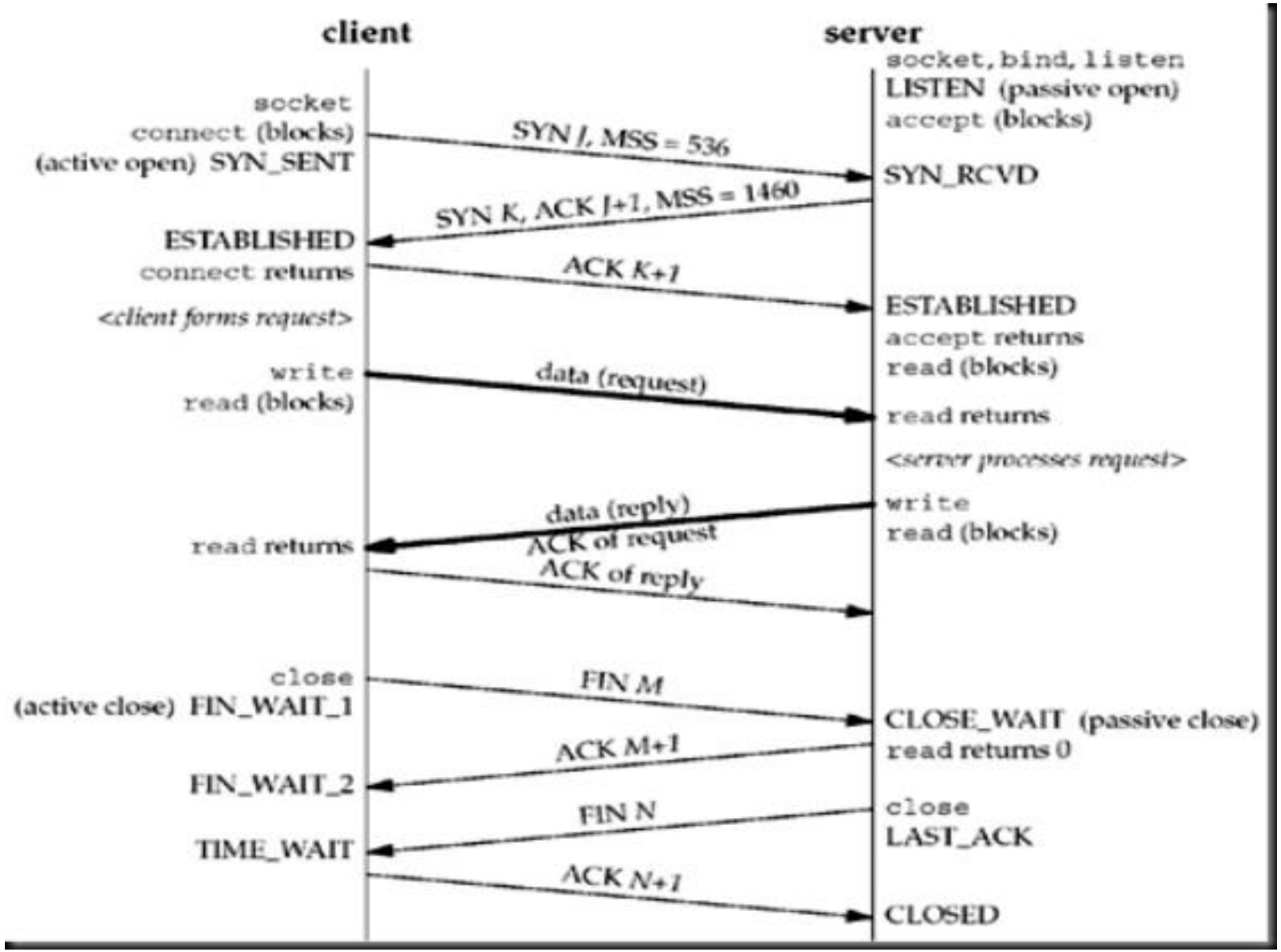


TCP - Connection

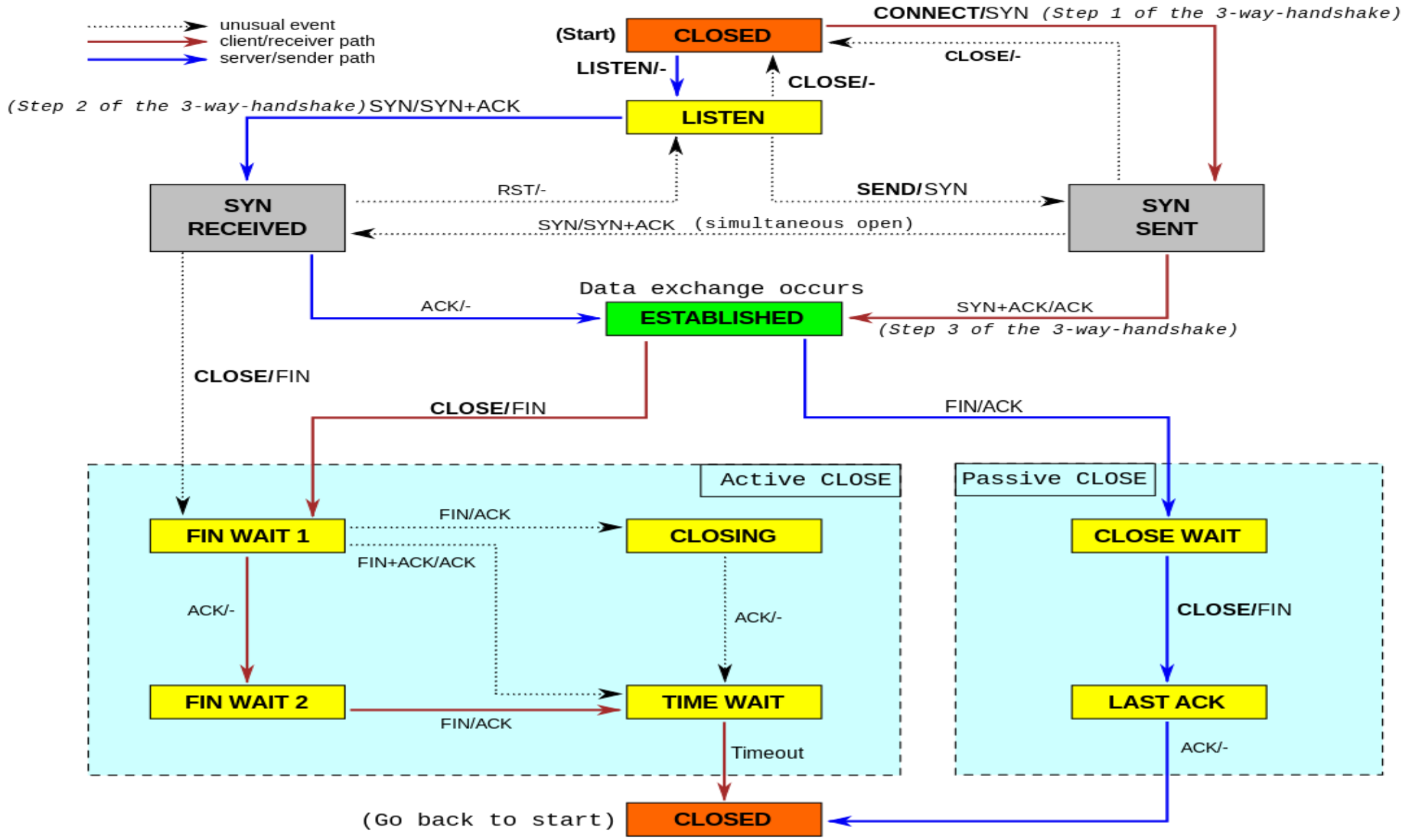




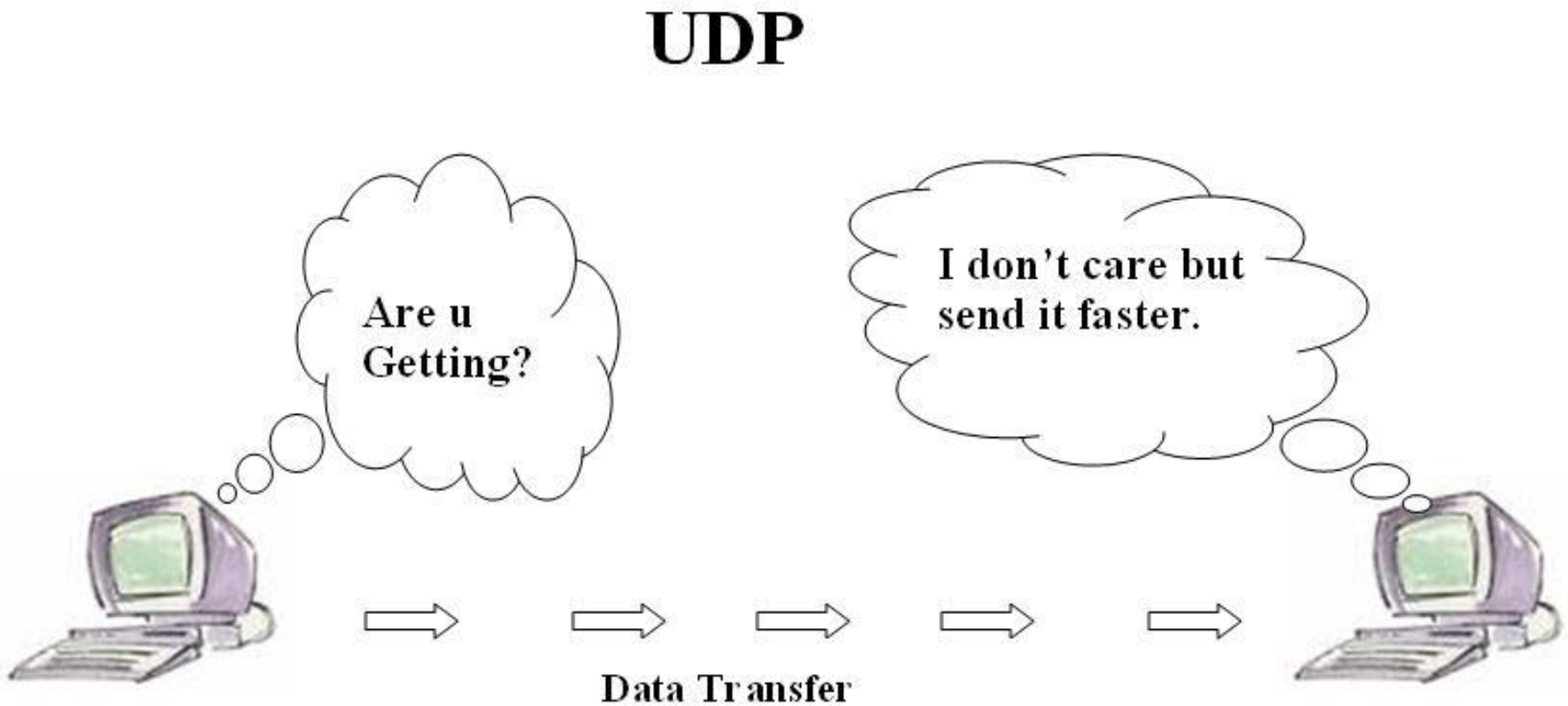
TCP – Connection : Packet Exchange



TCP-Connection state diagram



UDP


UDP
Are u Getting?
I don't care but send it faster.
Data Transfer

TCP Use cases

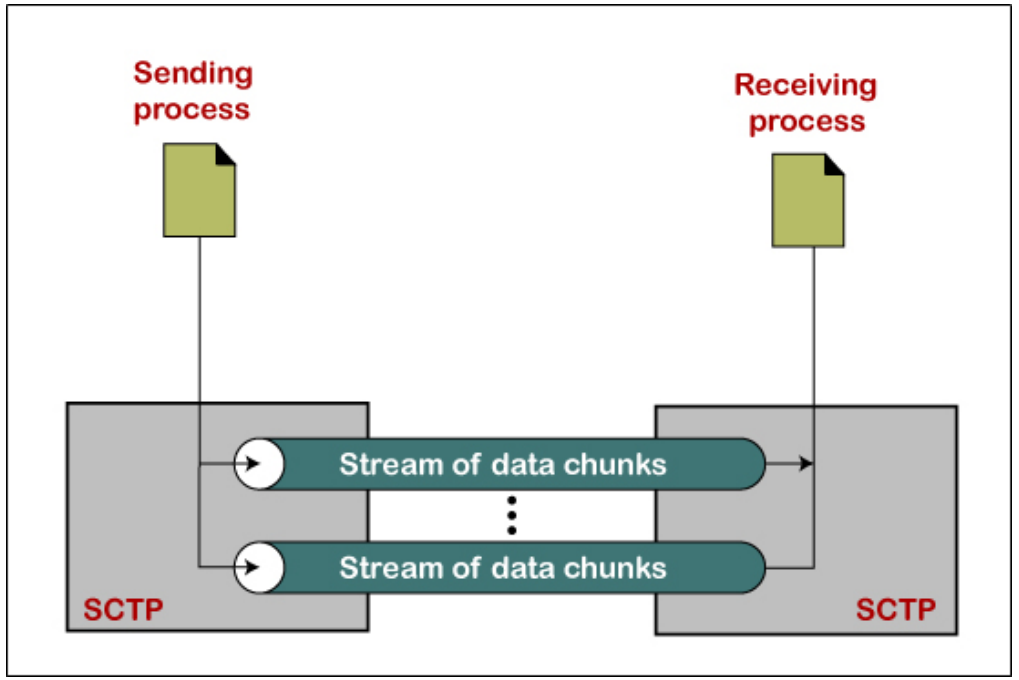
Application	Description
DHCP	Dynamic Host Configuration Protocol assigns IP addresses
DNS	Domain Name System translates website names to IP addresses
HTTP	Hypertext Transfer Protocol used to transfer web pages
NBNS	NetBIOS Name Service translates local host names to IP addresses
SMTP	Simple Mail Transfer Protocol sends email messages
SNMP	Simple Network Management Protocol manages network devices
SNTP	Simple Network Time Protocol provides time of day
Telnet	Bi-directional text communication via a terminal application
TFTP	Trivial File Transfer Protocol used to transfer small amounts of data

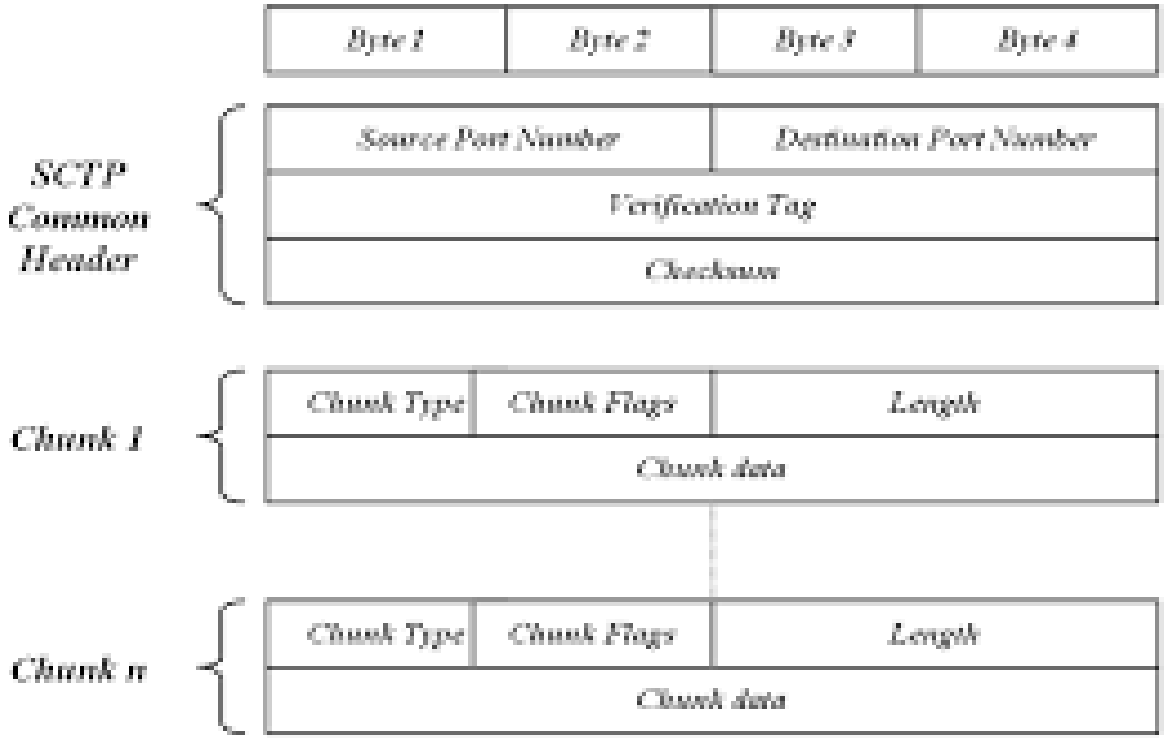
UDP Use cases

UDP Applications

- Used for applications that can tolerate small amount of packet loss:
  - Multimedia applications,
  - Internet telephony,
  - real-time-video conferencing
  - Domain Name System messages
  - Audio
  - Routing Protocols

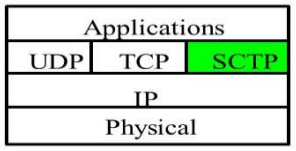
SCTP



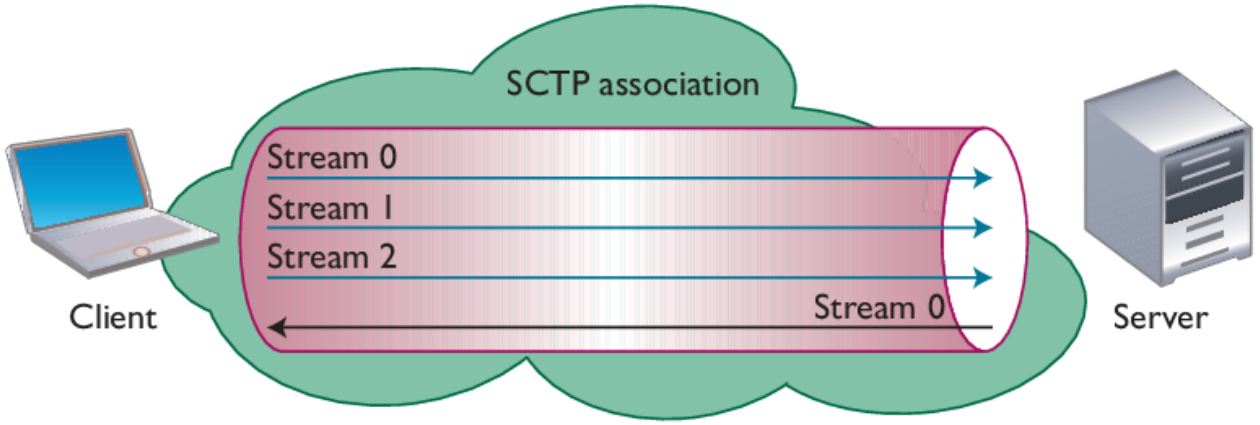


What is SCTP?

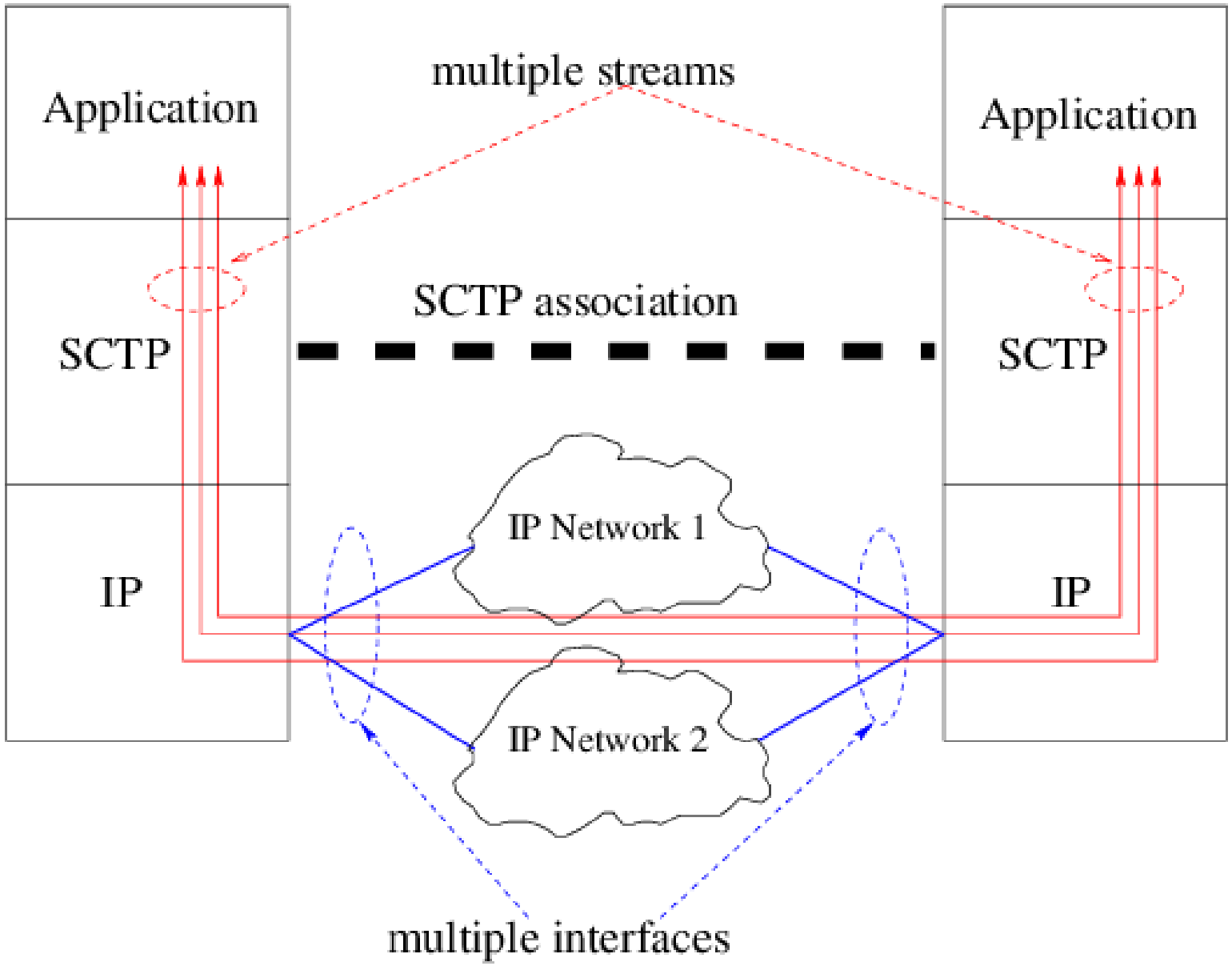
- SCTP is Stream Control Transmission Protocol, a Transport layer protocol.
- SCTP is reliable data transfer protocol which operates over the Network layer protocol like IP.



http://rivus.sourceforge.net/



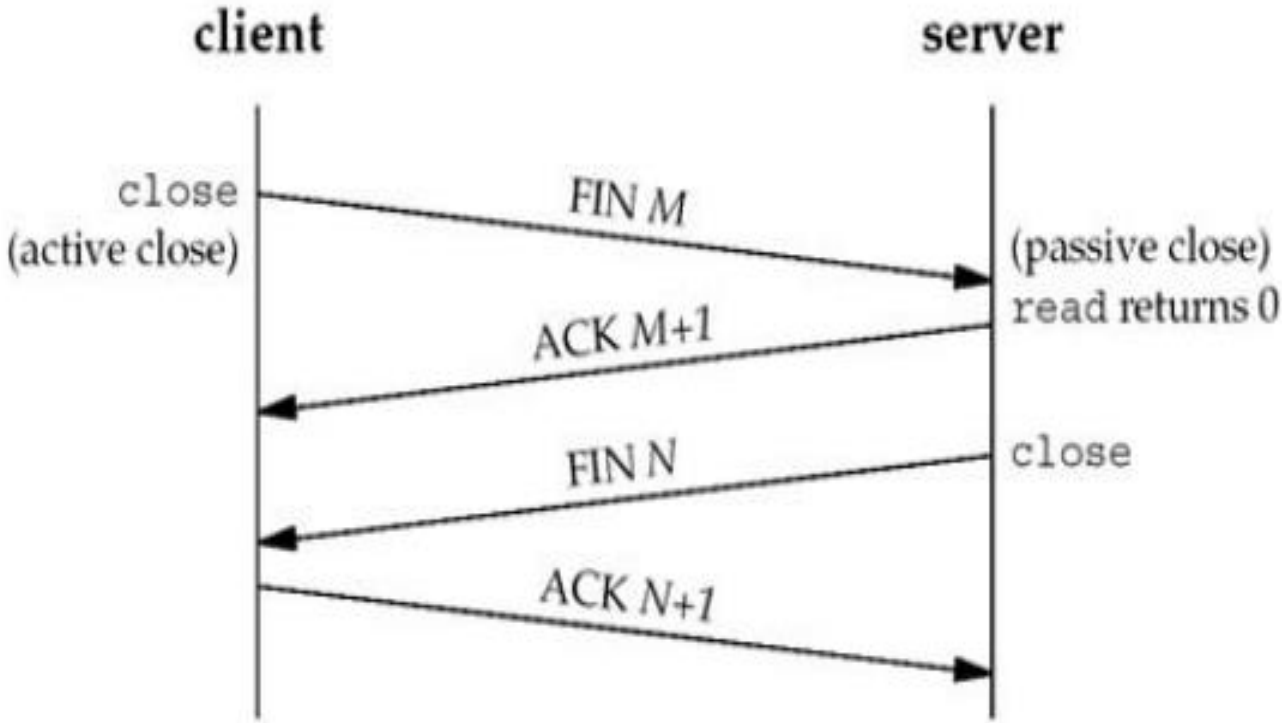
SCTP


Application
multiple streams
Application
SCTP association
SCTP
SCTP
IP Network 1
IP
IP
IP Network 2
multiple interfaces

TCP Connection Termination

Packets exchanged when a TCP connection is closed – Takes 4 segments to terminate the connection.

1. Calls ***close*** first. (Active close)
2. Passive close at the other end.
3. Application will not receive any Data after this. Closes socket and sends a FIN.
4. Acknowledges the FIN.



TCP – Options

Each SYN can contain TCP options. Commonly used options include the following:

MSS option: With this option, the TCP sending the SYN announces its maximum segment size, the maximum amount of data that it is willing to accept in each TCP segment, on this connection. The sending TCP uses the receiver's MSS value as the maximum size of a segment that it sends.

Window scale option: The maximum window that either TCP can advertise to the other TCP is 65,535, because the corresponding field in the TCP header occupies 16 bits. But, high-speed connections, common in today's Internet (45 Mbits/sec and faster, as described in RFC 1323 [Jacobson, Braden, and Borman 1992]), or long delay paths (satellite links) require a larger window to obtain the maximum throughput possible. This newer option specifies that the advertised window in the TCP header must be scaled (left shifted) by 0–14 bits, providing a maximum window of almost one gigabyte (65,535 x 214). Both end-systems must support this option for the window scale to be used on a connection.

Timestamp option: This option is needed for high-speed connections to prevent possible data corruption caused by old, delayed, or duplicated segments. Since it is a newer option, it is negotiated similarly to the window scale option. As network programmers there is nothing we need to worry about with this option.

Comparison

	TCP	UDP	SCTP
Reliability	trustworthy	Unreliable	Trustworthy
Connection type	Connection-oriented	Connectionless	Connection-oriented
Transmission type	Byte-oriented	News-oriented	News-oriented
Transfer sequence	Strictly ordered	Disordered	Partially ordered
Overload control	Yes	No	Yes
Error tolerance	No	No	Yes