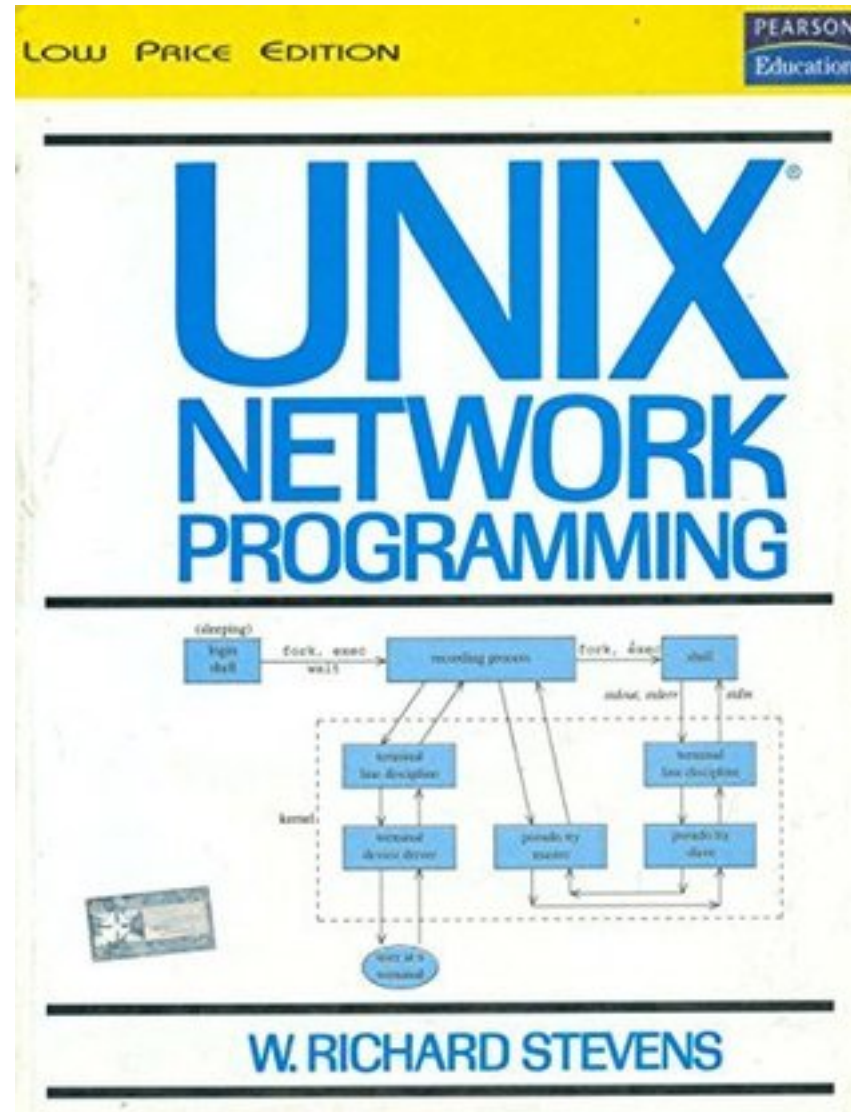


# Network Programming



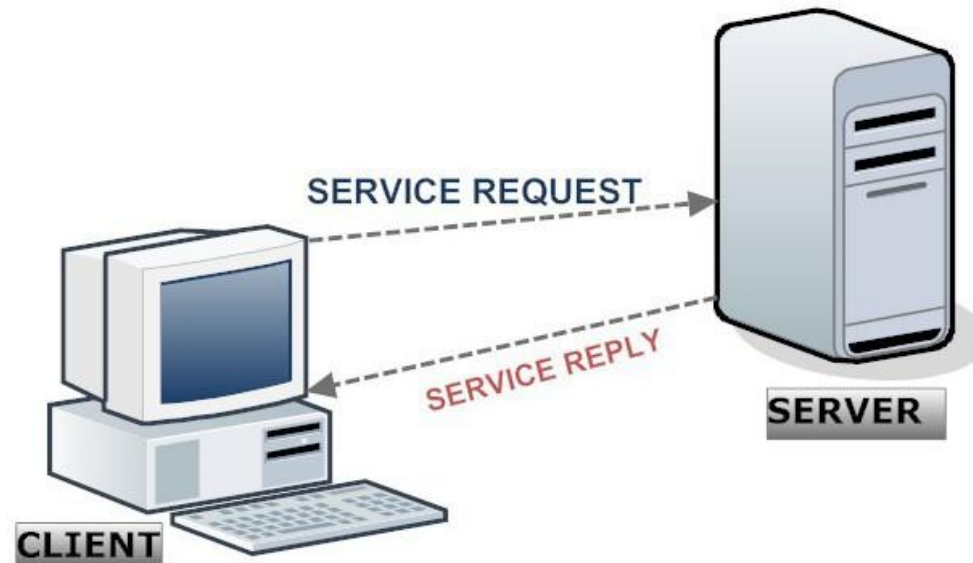
# UNIT-1

- **Introduction:** Introduction, Client/server communication, OSI Model, BSD Networking history, Test Networks and Hosts, Unix Standards, 64-bit architectures.
- **Transport Layer:** TCP, UDP and SCTP, TCP Connection Establishment and Termination.
- **Self learning topics:** TCP/IP Protocols in nut shell.

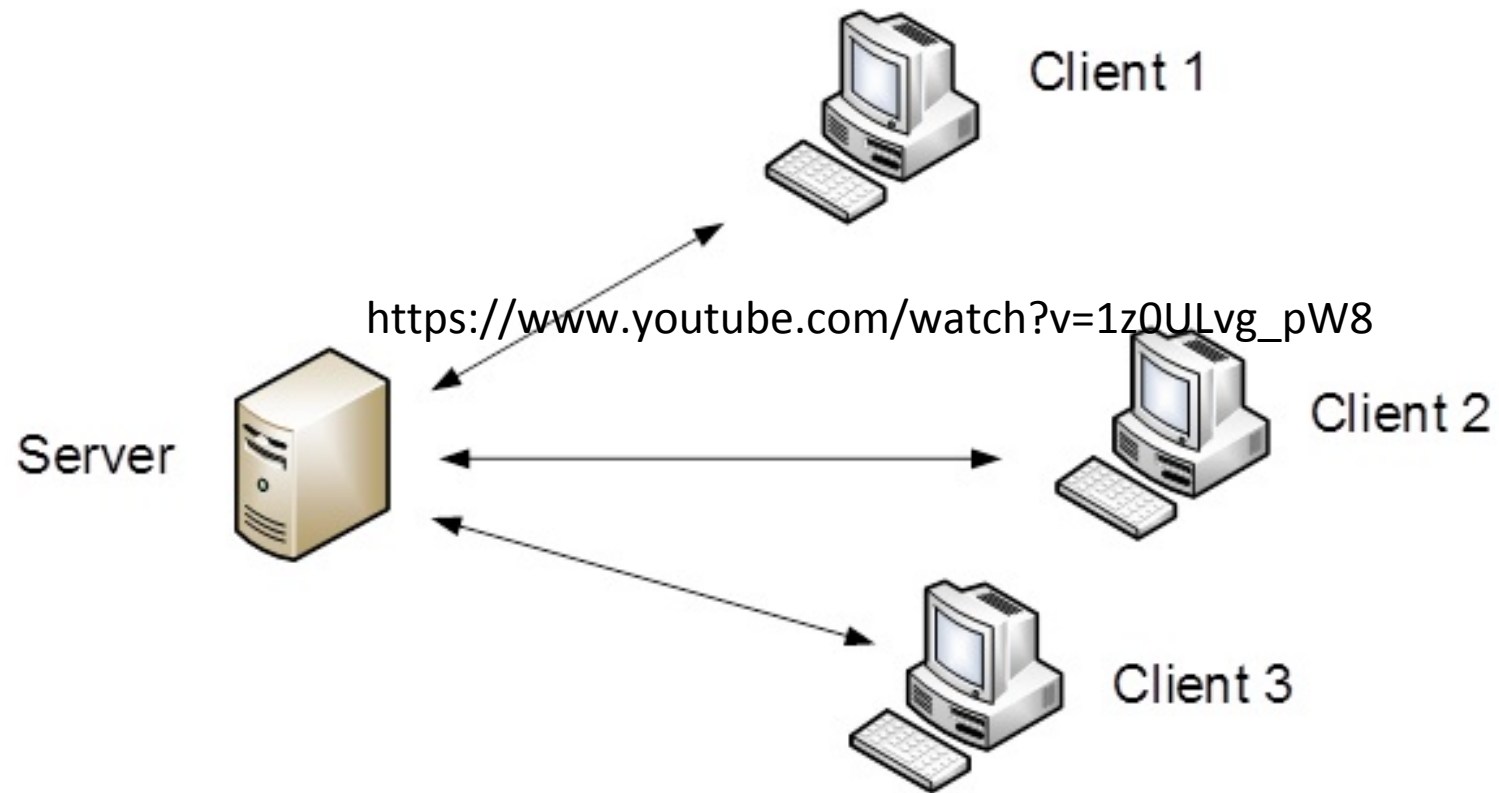
**Network programming** involves writing programs **to**  
**communicate with processes** either on the **same or on**  
**other machines** on the network using **standard Protocols...**

High-level decision must be made as to **which program would initiate** the **communication first** and **when responses** are expected...

**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



# OSI - Model

## 7 Layers of the OSI Model

### Application

- End User layer
- HTTP, FTP, IRC, SSH, DNS

### Presentation

- Syntax layer
- SSL, SSH, IMAP, FTP, MPEG, JPEG

### Session

- Synch & send to port
- API's, Sockets, WinSock

### Transport

- End-to-end connections
- TCP, UDP

### Network

- Packets
- IP, ICMP, IPSec, IGMP

### Data Link

- Frames
- Ethernet, PPP, Switch, Bridge

### Physical

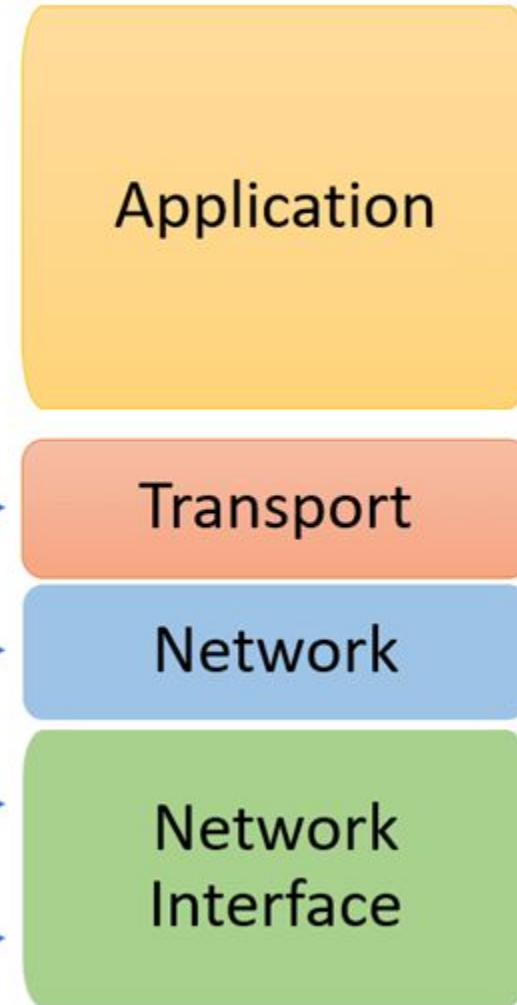
- Physical structure
- Coax, Fiber, Wireless, Hubs, Repeaters

# TCP/IP Model

## OSI Reference Model



## TCP/IP Conceptual Layers





**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.



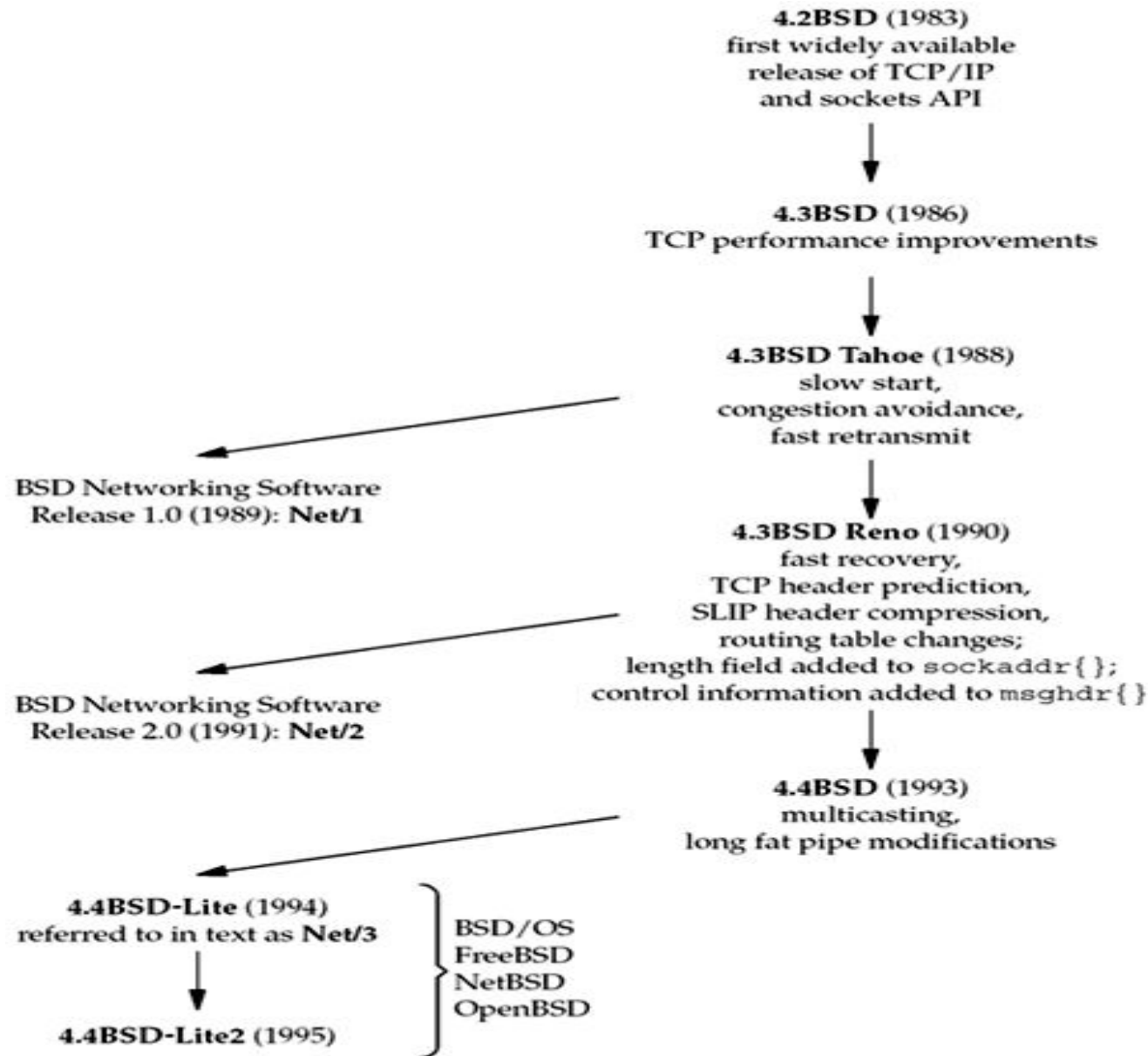
## BSD



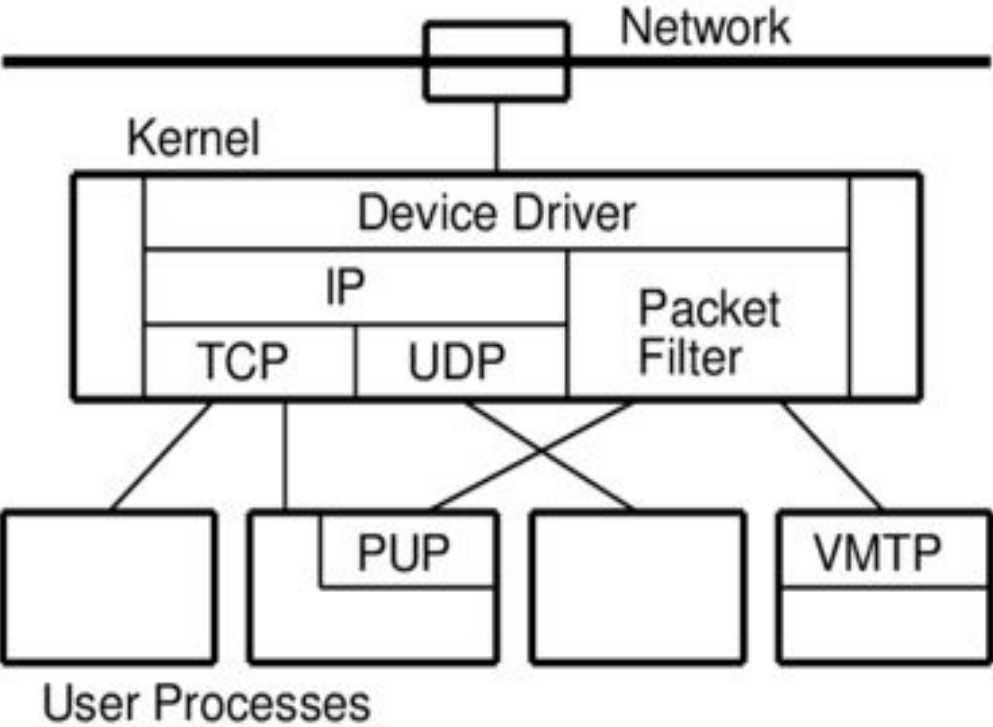
# The History of BSD

- ◆ 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



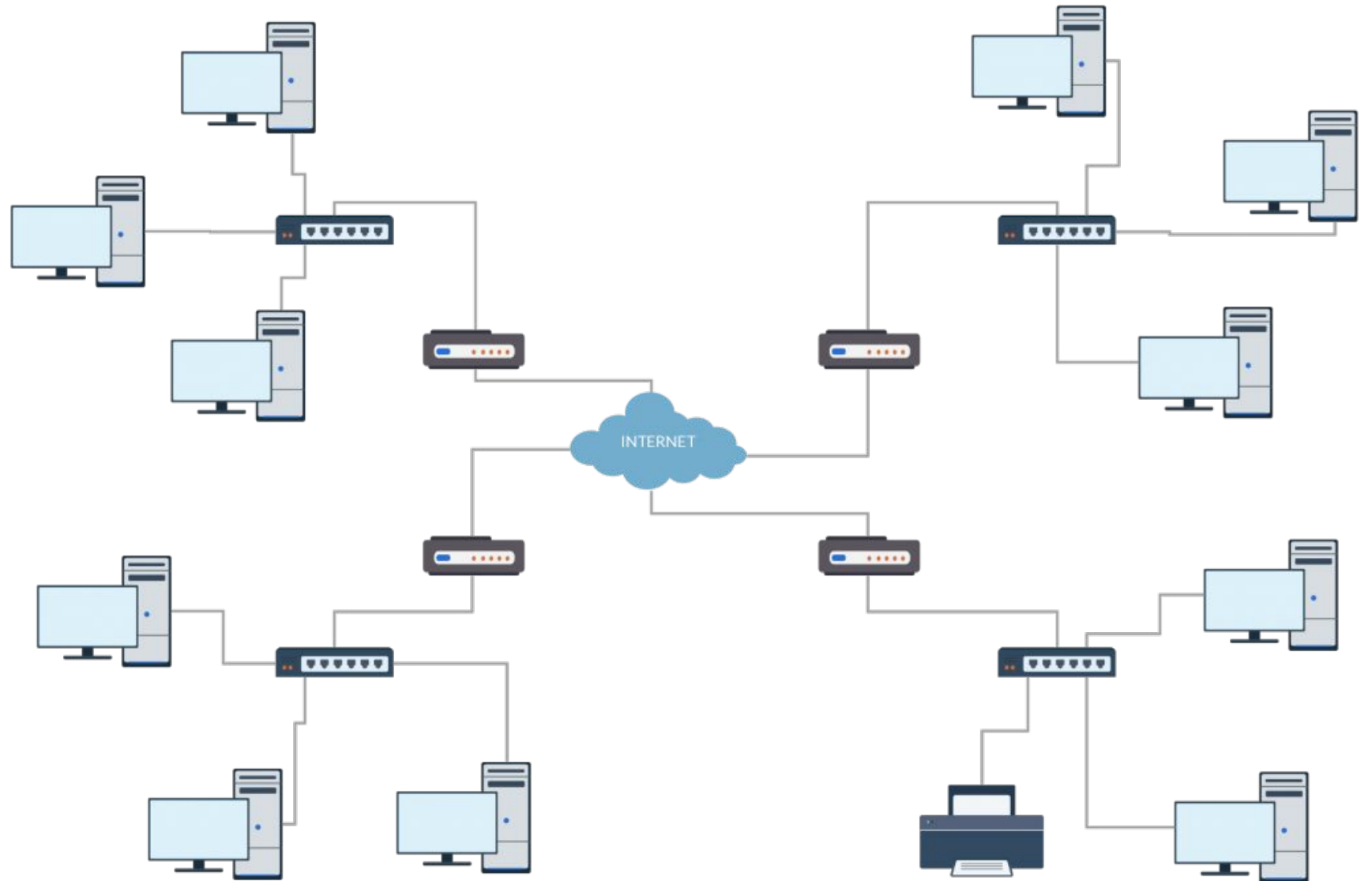
# BSD Network



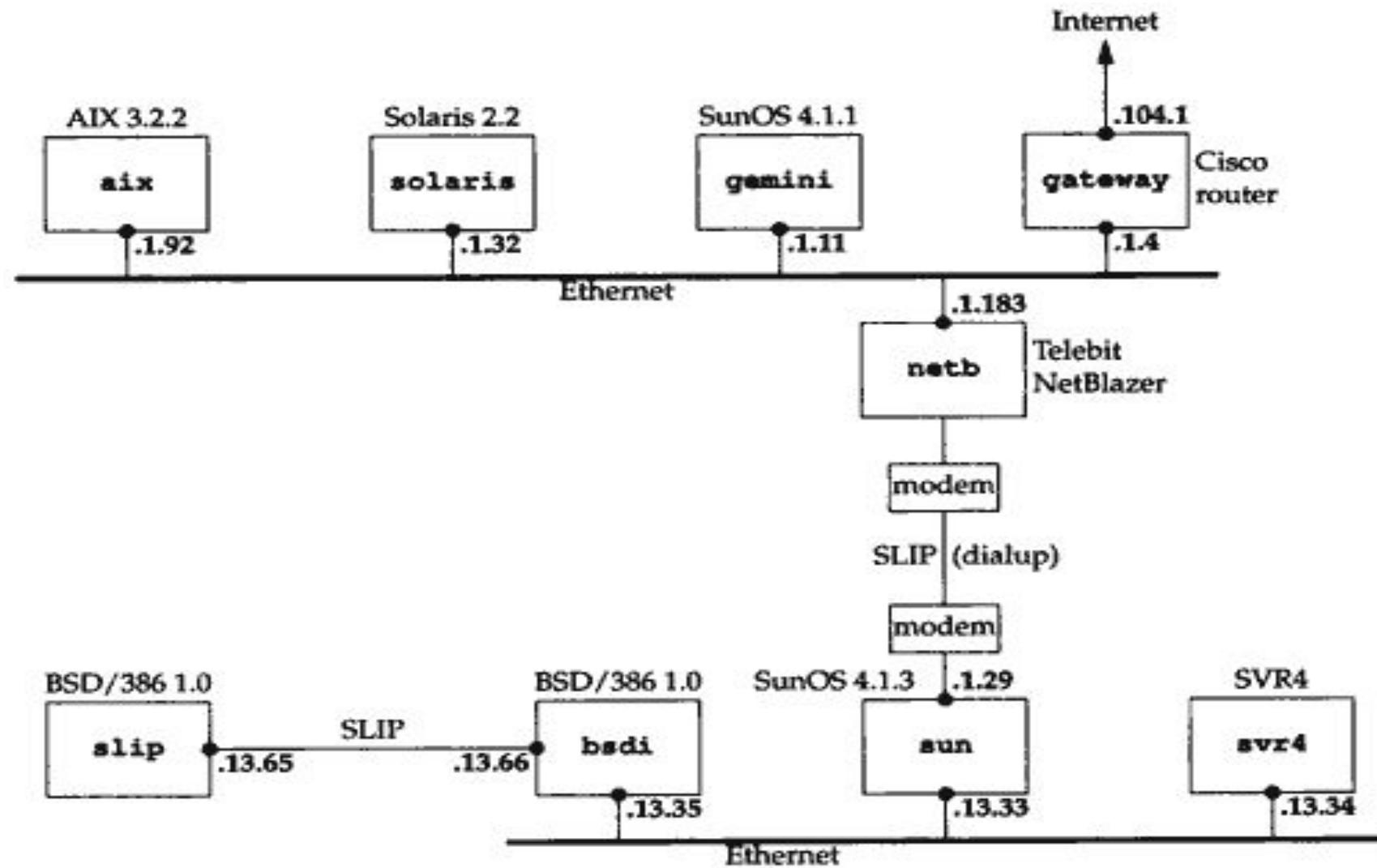
**Potentially Unwanted Applications**

**Versatile Message Transaction Protocol**

# Test Network and Hosts



# Test Network and Hosts



# Communication over LAN

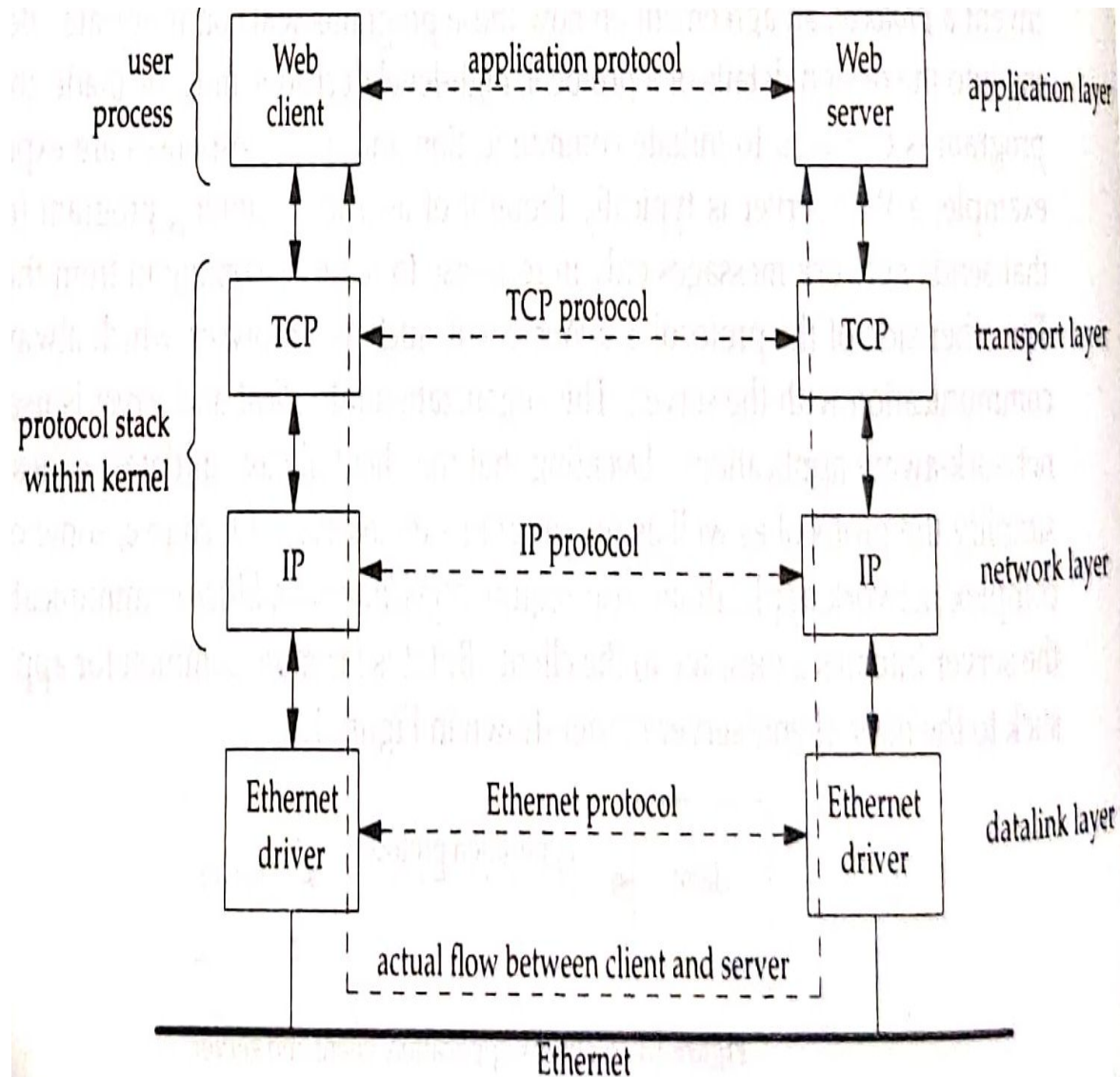
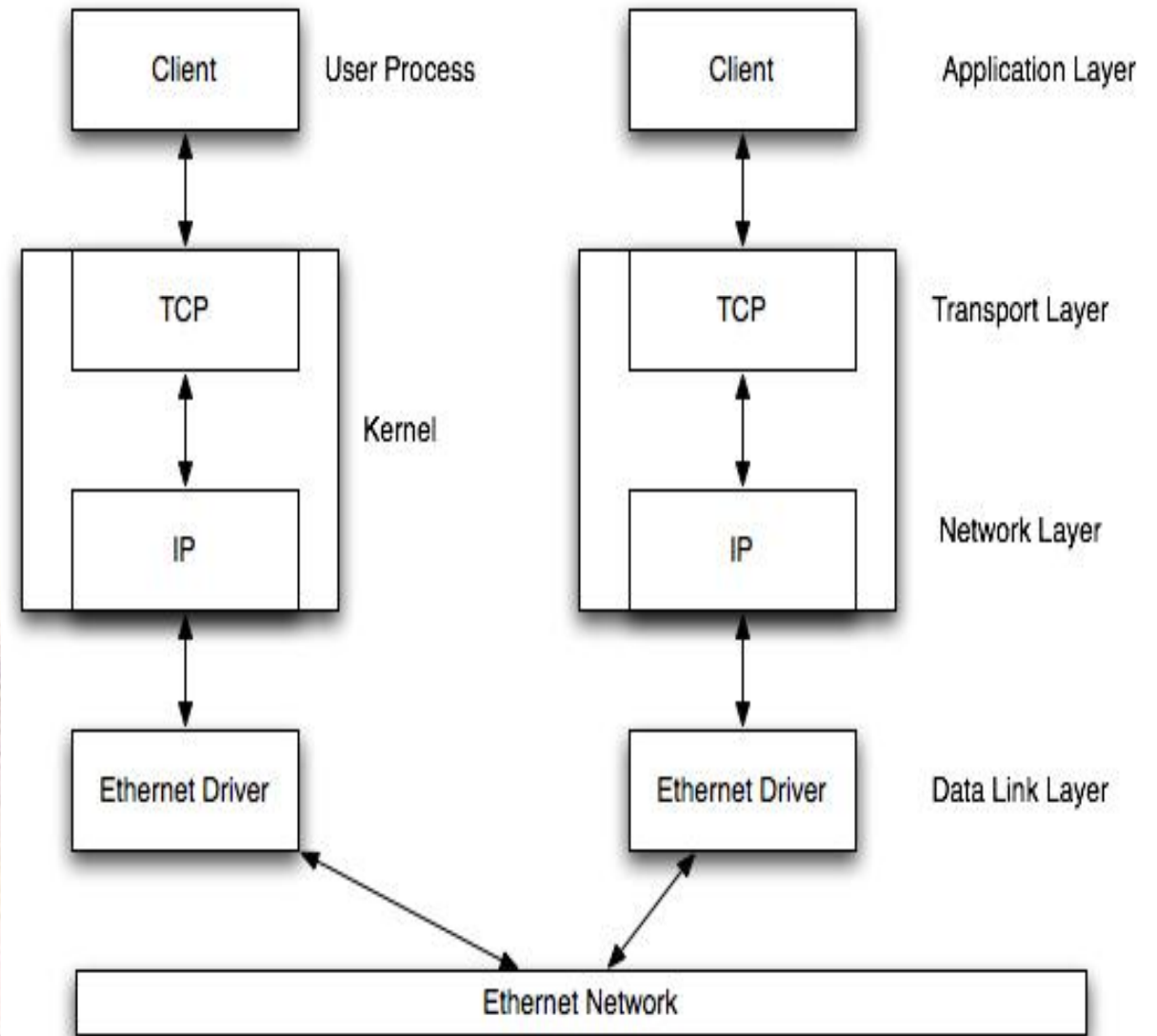
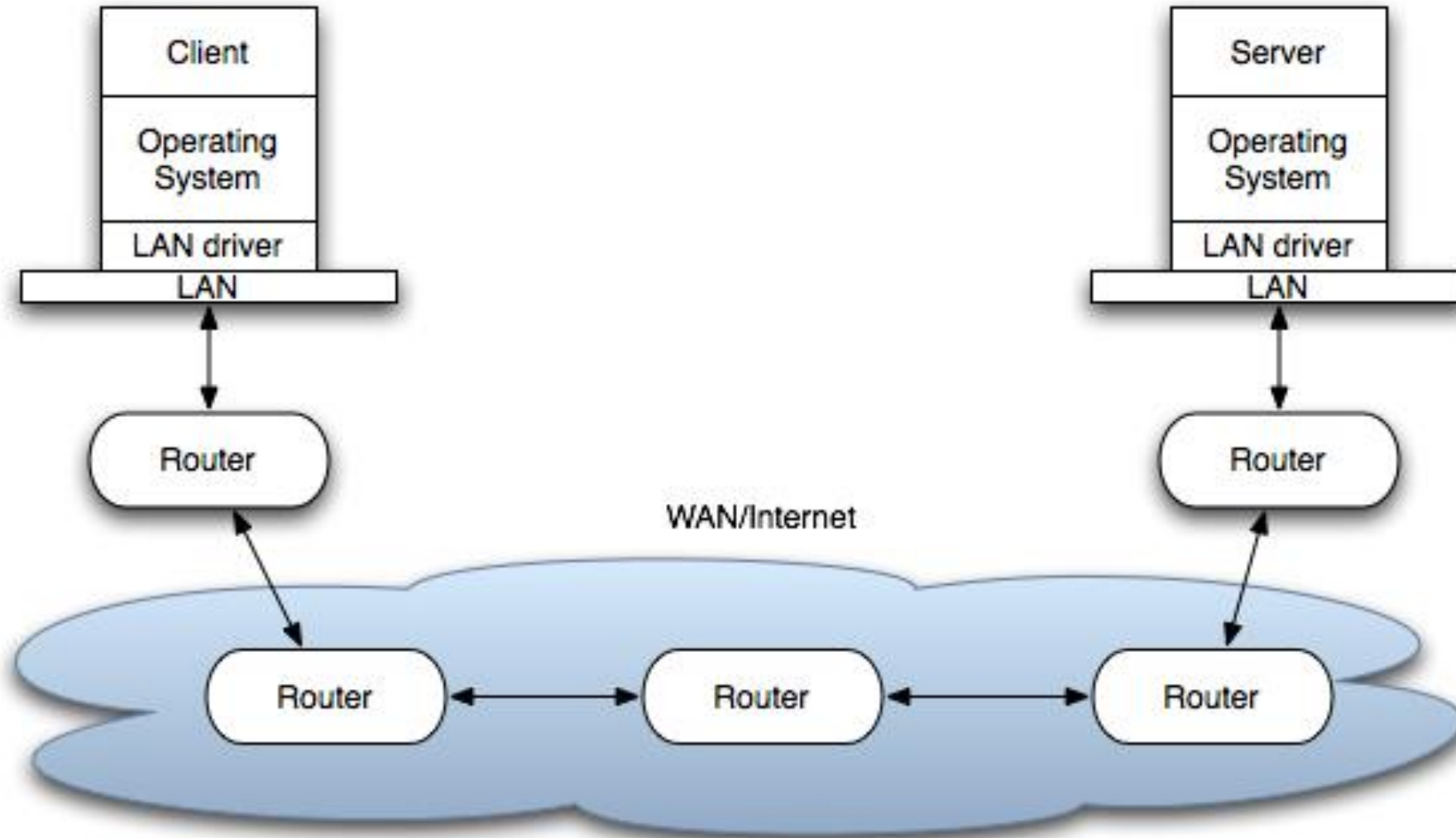


Figure 1.3 Client and server on the same Ethernet communicating using TCP.





# Communication over WAN





# Discovering Network Topology – netstat –ni and netstat -r

```
Command Prompt
TCP 10.30.2.42:1040 maa05s12-in-f3:http ESTABLISHED
TCP 10.30.2.42:1047 172.67.72.207:https ESTABLISHED
TCP 10.30.2.42:1054 151.101.2.137:https ESTABLISHED
TCP 10.30.2.42:1056 151.139.128.11:https ESTABLISHED
^C
C:\Users\admin>netstat -r
=====
Interface List
13...84 a9 3e 65 d8 ca .....Intel(R) Ethernet Connection (7) I219-LM
7...0a 00 27 00 00 07 .....VirtualBox Host-Only Ethernet Adapter
4...92 32 4b 3a 34 f9 .....Microsoft Wi-Fi Direct Virtual Adapter #7
12...d2 32 4b 3a 34 f9 .....Microsoft Wi-Fi Direct Virtual Adapter #8
8...90 32 4b 3a 34 f9 .....Realtek RTL8822BE 802.11ac PCIe Adapter
5...90 32 4b 3a 34 fa .....Bluetooth Device (Personal Area Network)
1.....Software Loopback Interface 1
=====
IPv4 Route Table
=====
Active Routes:
Network Destination Netmask Gateway Interface Metric
0.0.0.0 0.0.0.0 10.30.0.1 10.30.2.42 35
10.30.0.0 255.255.224.0 On-link 10.30.2.42 291
10.30.2.42 255.255.255.255 On-link 10.30.2.42 291
10.30.31.255 255.255.255.255 On-link 10.30.2.42 291
127.0.0.0 255.0.0.0 On-link 127.0.0.1 331
127.0.0.1 255.255.255.255 On-link 127.0.0.1 331
127.255.255.255 255.255.255.255 On-link 127.0.0.1 331
192.168.56.0 255.255.255.0 On-link 192.168.56.1 281
192.168.56.1 255.255.255.255 On-link 192.168.56.1 281
```

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

# 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

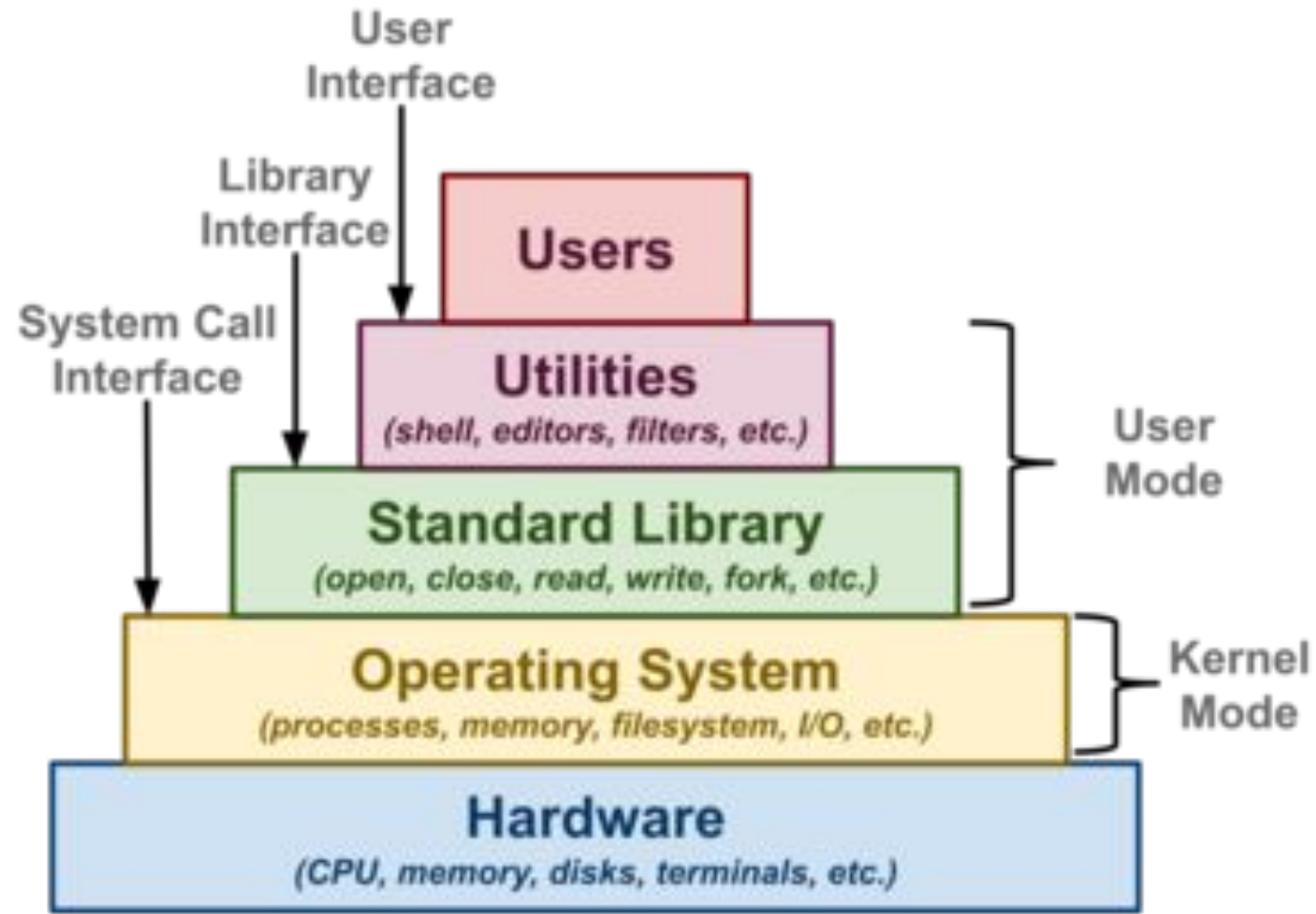
POSIX.1	System Interface (basic reference standard) <sup>a,b</sup>
POSIX.2	Shell and Utilities <sup>a</sup>
POSIX.3	Methods for Testing Conformance to POSIX <sup>a</sup>
POSIX.4	Real-time Extensions
POSIX.4a	Threads Extensions
POSIX.4b	Additional Real-time Extensions
POSIX.6	Security Extensions
POSIX.7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

## Unix Standard - POSIX

### What does POSIX mean?

- **P**ortable **O**perating **S**ystem **I**nterface for **U**nix" 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



## 64 Bit Architectures

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

# 64 Bit Architectures

## IPv6 Header

Version	Traffic Class	Flow Label	
Payload Length		Next Header	Hop Limit
Source Address			
Destination Address			

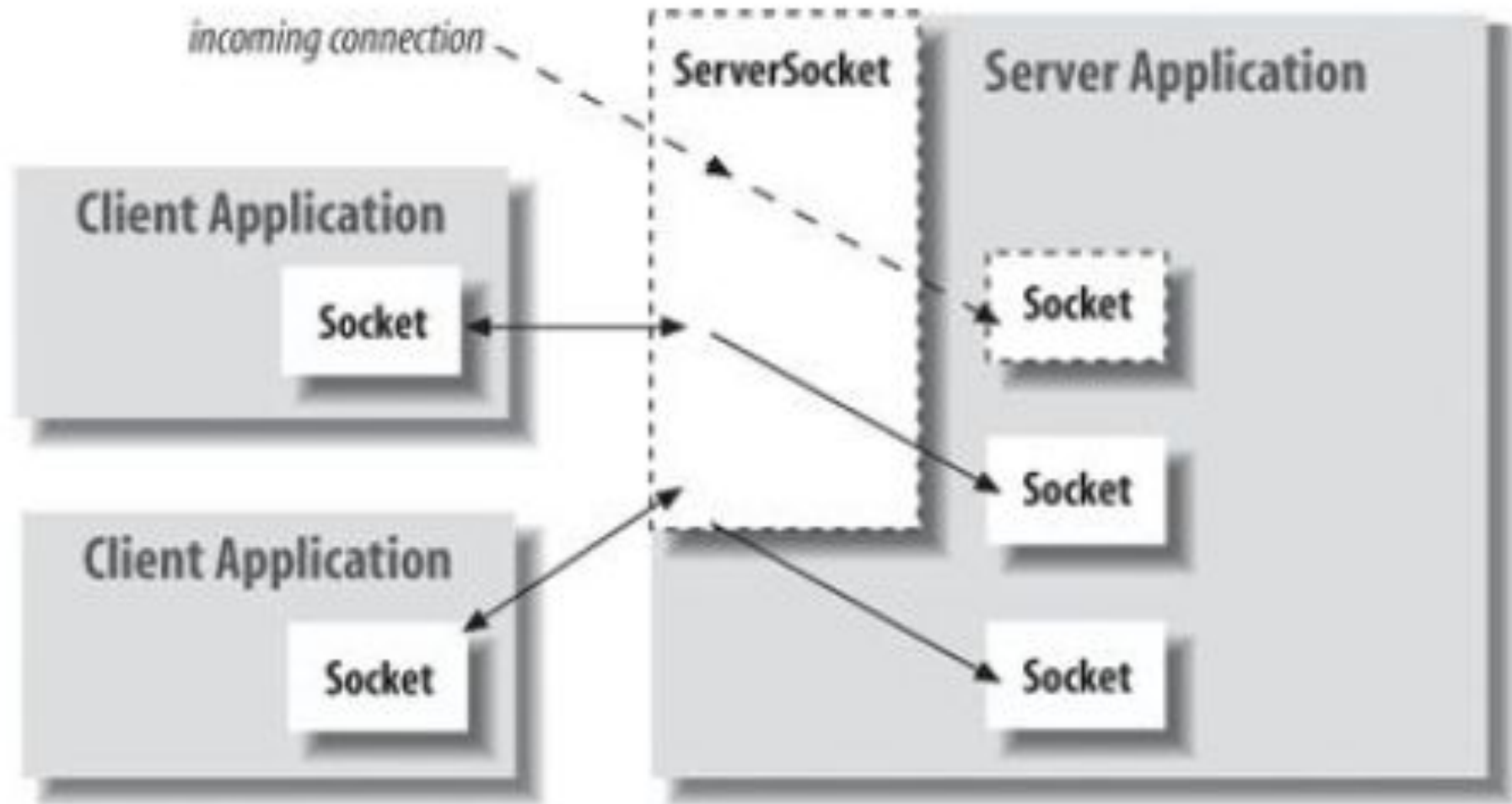
## IPv4 Header

Version	IHL	Type of Service	Total Length	
Identification			Flags	Fragment Offset
TTL	Protocol		Header Checksum	
Source Address				
Destination Address				
Options			Padding	

### Legend

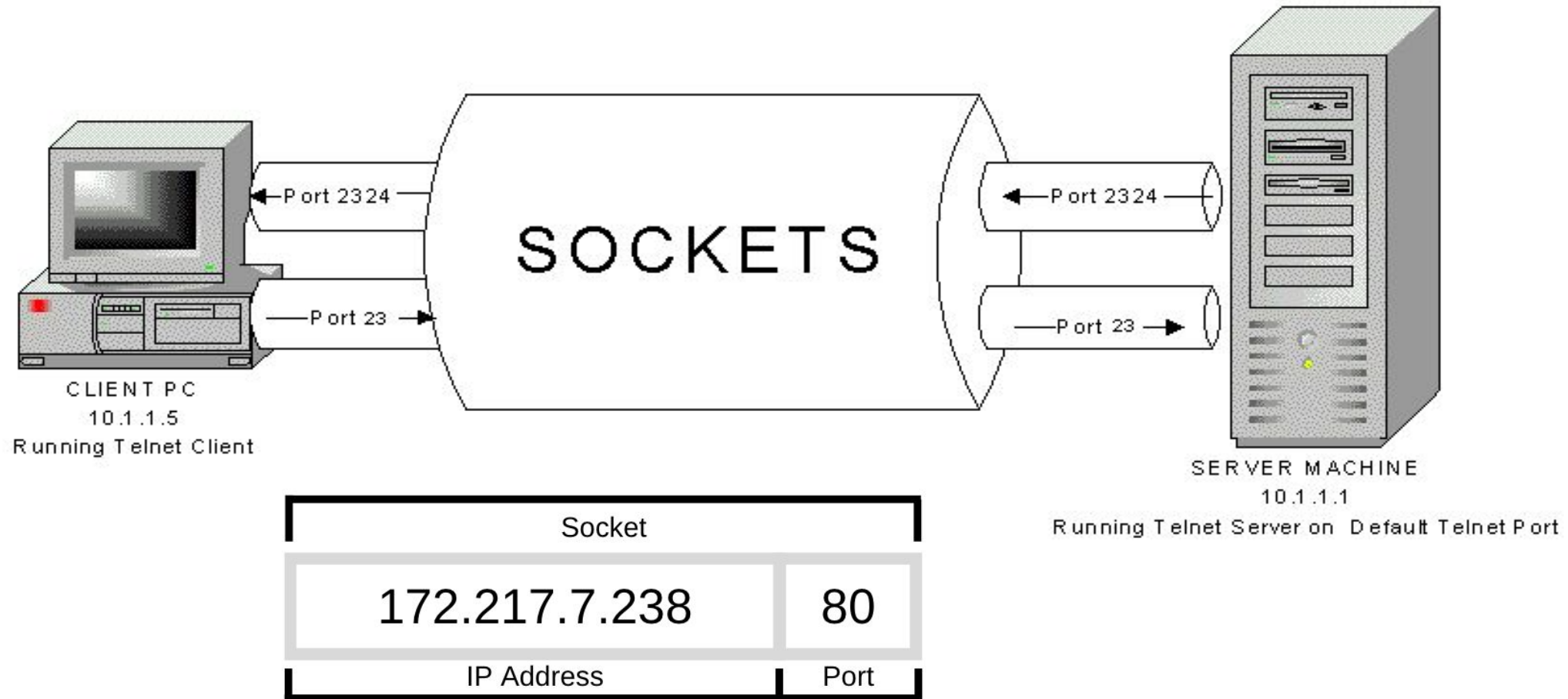
	Fields <b>kept</b> in IPv6
	Fields <b>kept</b> in IPv6, but name and position changed
	Fields <b>not kept</b> in IPv6
	Fields that are <b>new</b> in IPv6

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

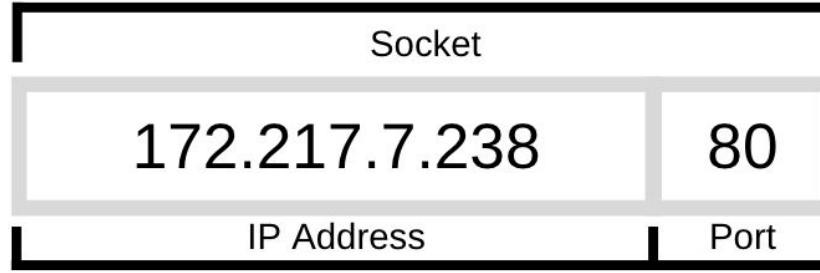




# Sockets : An end point for communication between processes across the network



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



```
struct sockaddr_in s_addr, c_addr;
```

```
...
```

```
struct sockaddr_in
```

```
{
```

```
    short sin_family;
```

AF\_INET

```
    u_short sin_port;
```

```
    struct in_addr sin_addr;
```

```
    char sin_zero[8];
```

Not use

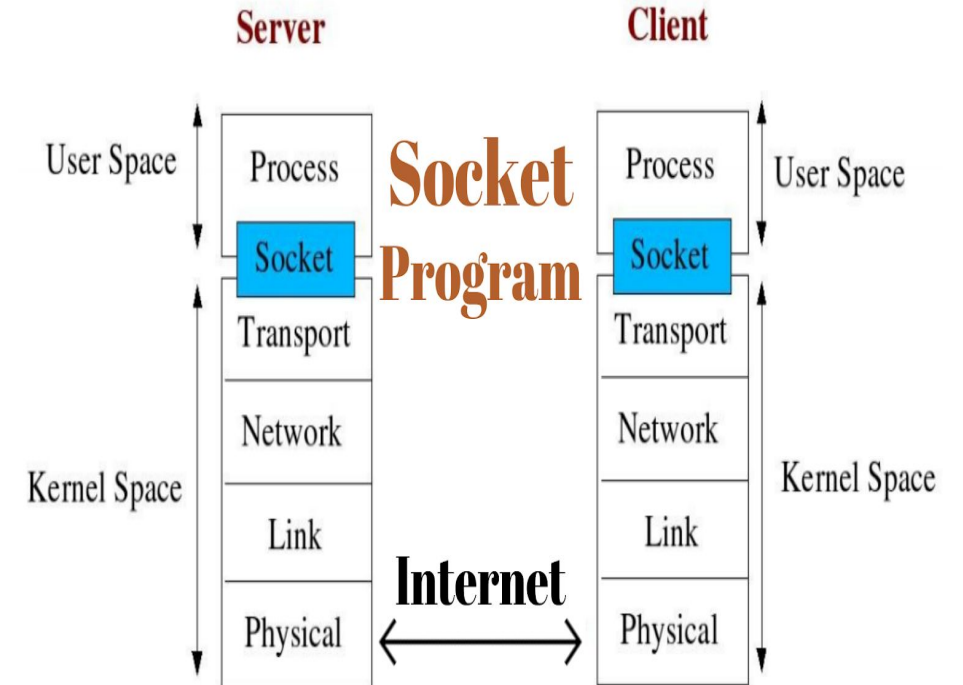
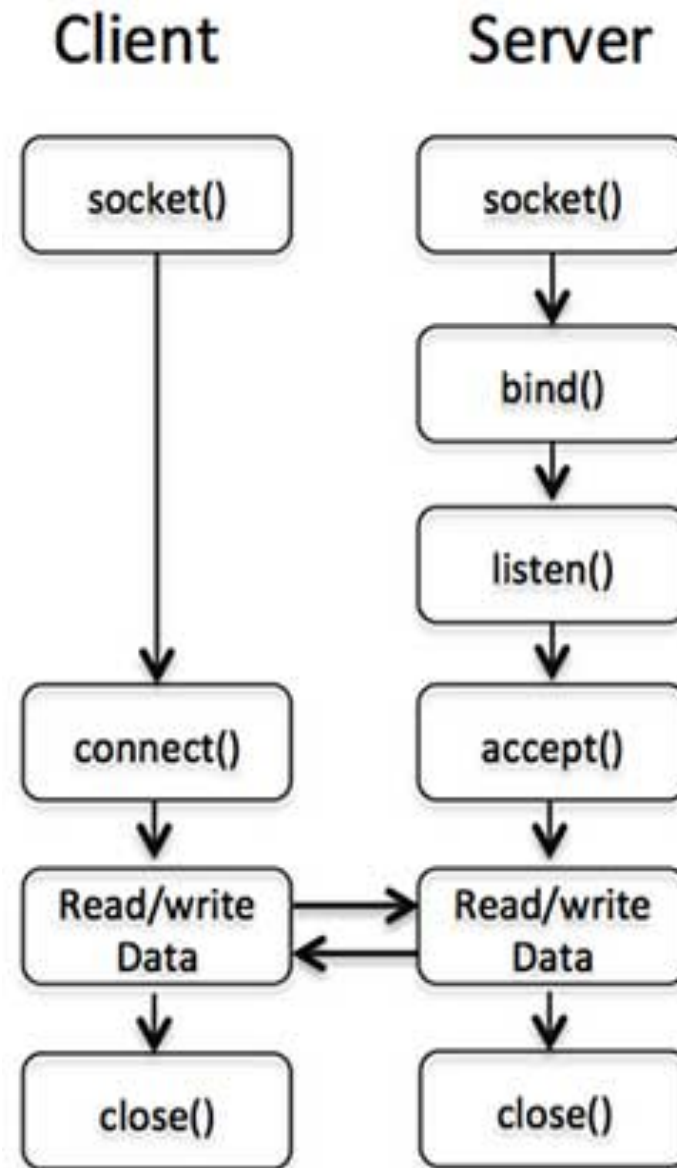
```
}
```

```
struct in_addr { unsigned  
long s_addr;  
};
```

Example ::

Day Time Client....

## Day Time Server at port no. 13



# Source Code of Day Time Client

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
```

```
int main(int argc, char **argv)
{
```

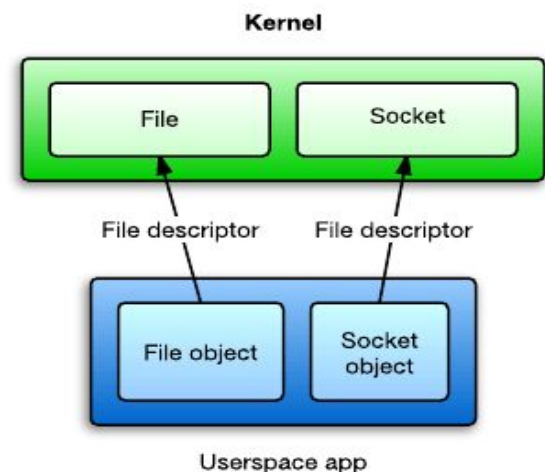
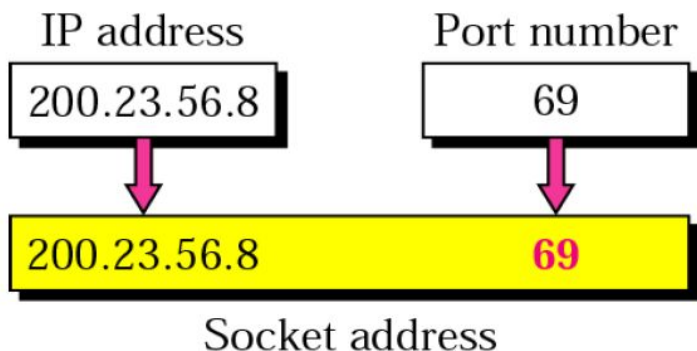
```
    int sockfd, n = 0;
    char recvline[1000 + 1];
    struct sockaddr_in servaddr;
```

```
    int port = 13;
```

```
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        err(1, "Socket Error");
    }
```

```
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(port);
```

```
    inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
    if (connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0) {
        err(1, "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'

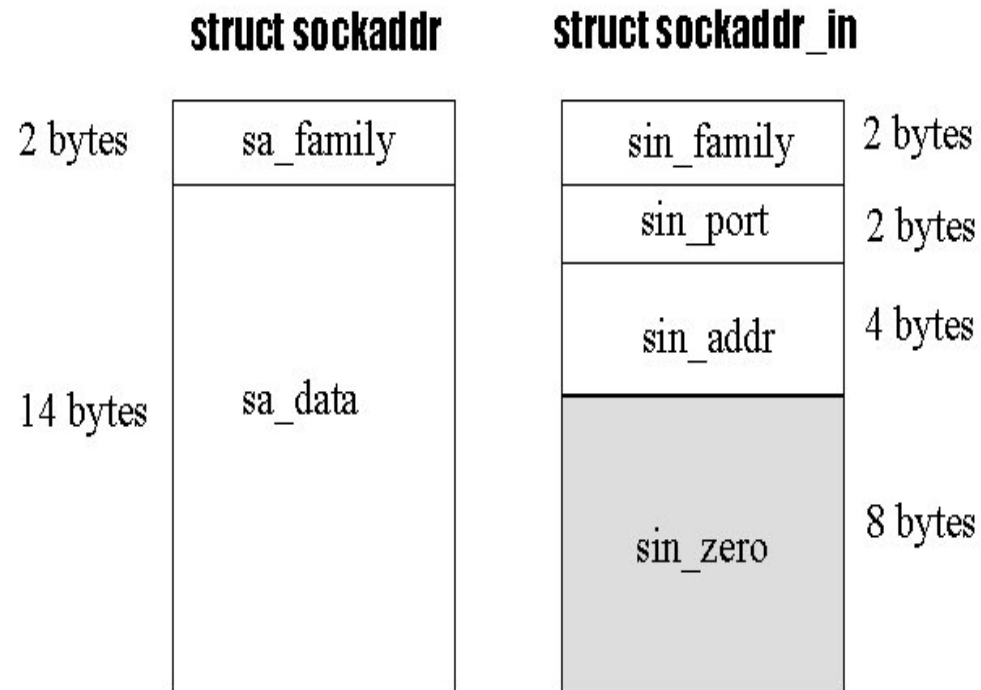
```

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

return 0;
}

```

## sockaddr vs. sockaddr\_in



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

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

```
struct sockaddr_in s_addr, c_addr;
```

```
...
```

```
struct sockaddr_in
```

```
{
```

```
    short sin_family;
```

```
    u_short sin_port;
```

```
    struct in_addr sin_addr;
```

```
    char sin_zero[8];
```

```
}
```

AF\_INET

Not use

```
struct in_addr {  
  
    unsigned long s_addr;  
  
};
```

# DayTime Server...

```
#include <sys/socket.h>
```

```
#include <sys/types.h>
```

```
#include <netinet/in.h>
```

```
#include <netdb.h>
```

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

```
int main(int argc, char **argv)
```

```
{
```

```
int listenfd, connfd;
```

```
int port = atoi(argv[1]);
```



```
struct sockaddr_in servaddr;
```

```
char buff[1000];
```

```
time_t ticks;
```

```
listenfd = socket(AF_INET, SOCK_STREAM, 0);
```



**CREATE A SOCKET**

```
bzero(&servaddr, sizeof(servaddr));
```


```
servaddr.sin_family = AF_INET;
```

```
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
servaddr.sin_port = htons(port);
```

**Initialize Socket Address**

```
bind(listenfd, (struct sockaddr *) &servaddr, sizeof(servaddr));
```



**Bind the SOCKET**

```
listen(listenfd, 8);
```



**Listen on the Port for connections**

```
for (;;) {
```

```
connfd = accept(listenfd, (struct sockaddr *) NULL, NULL);
```

**Accept connection request from Client**

```
ticks = time(NULL);
```

```
snprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));
```

**Write to socket... { Serve the Client }**

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

```
close(connfd);
```

```
}
```

```
}
```

## 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;
}
```

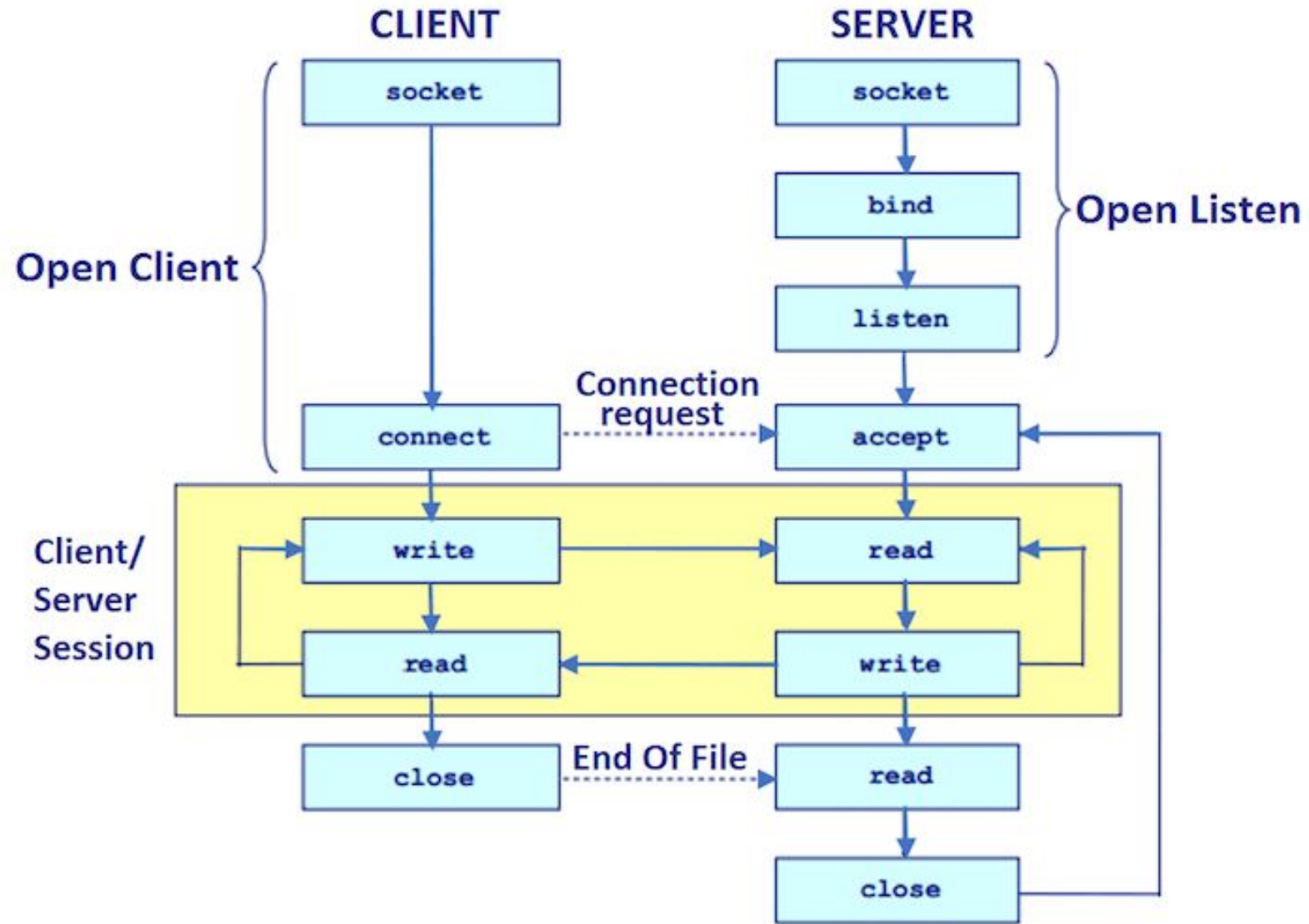
# Error Handling and Wrapper functions

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.

We find that 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

```
int Socket(int family, int type, int protocol)

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

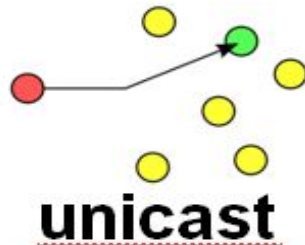


## SOCKET API

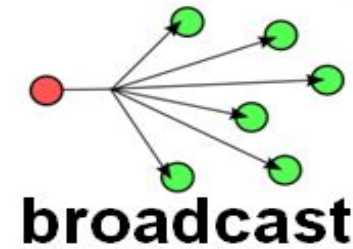
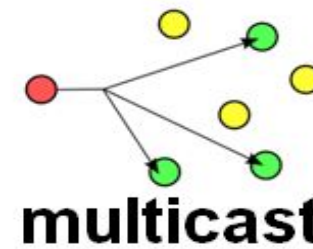
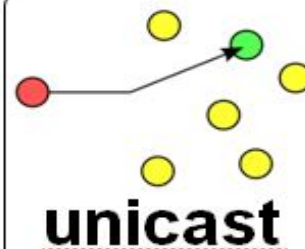
# TCP AND UDP



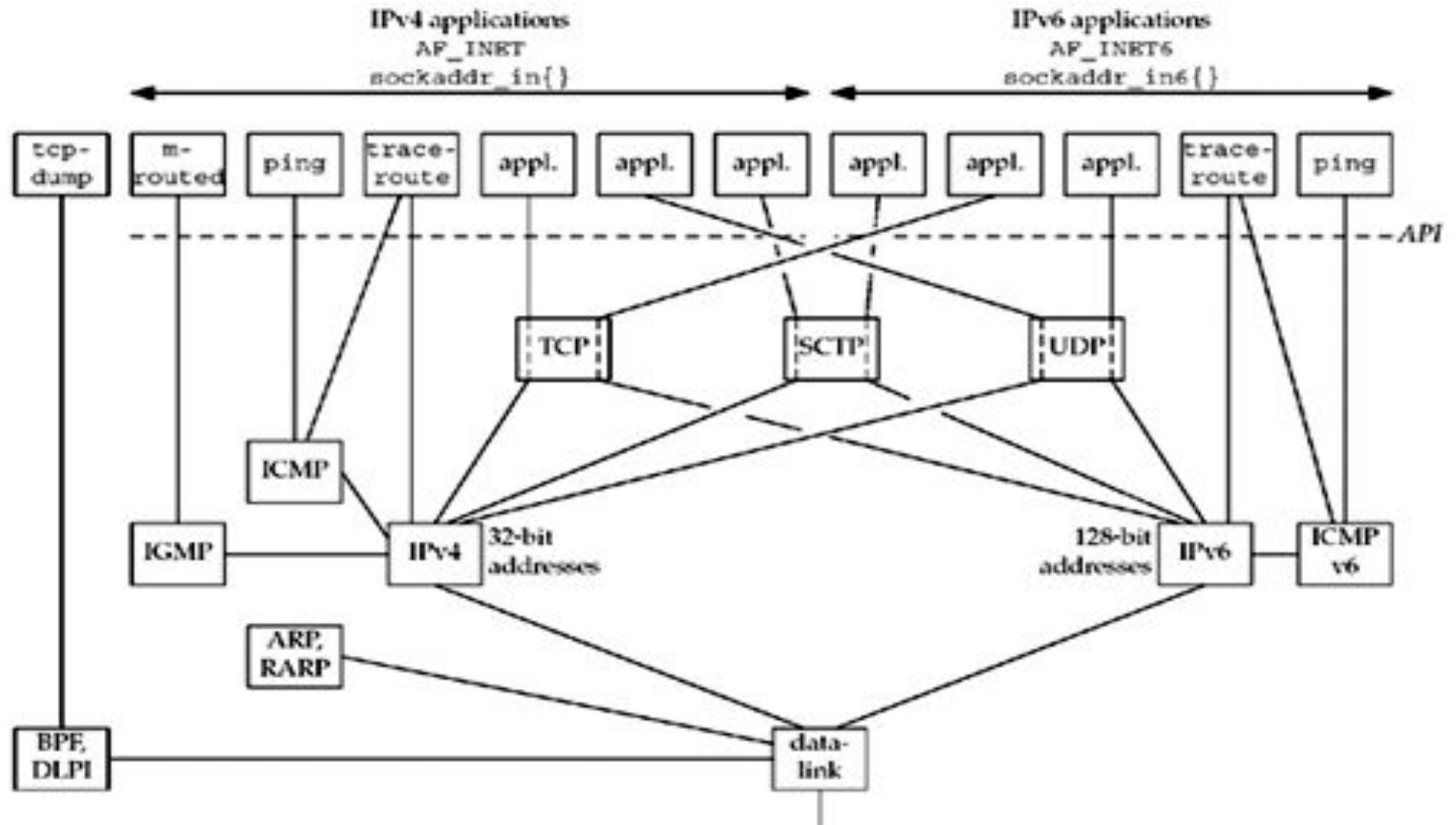
- **Slower but reliable transfers**
- **Typical applications:**
  - Email
  - Web browsing



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



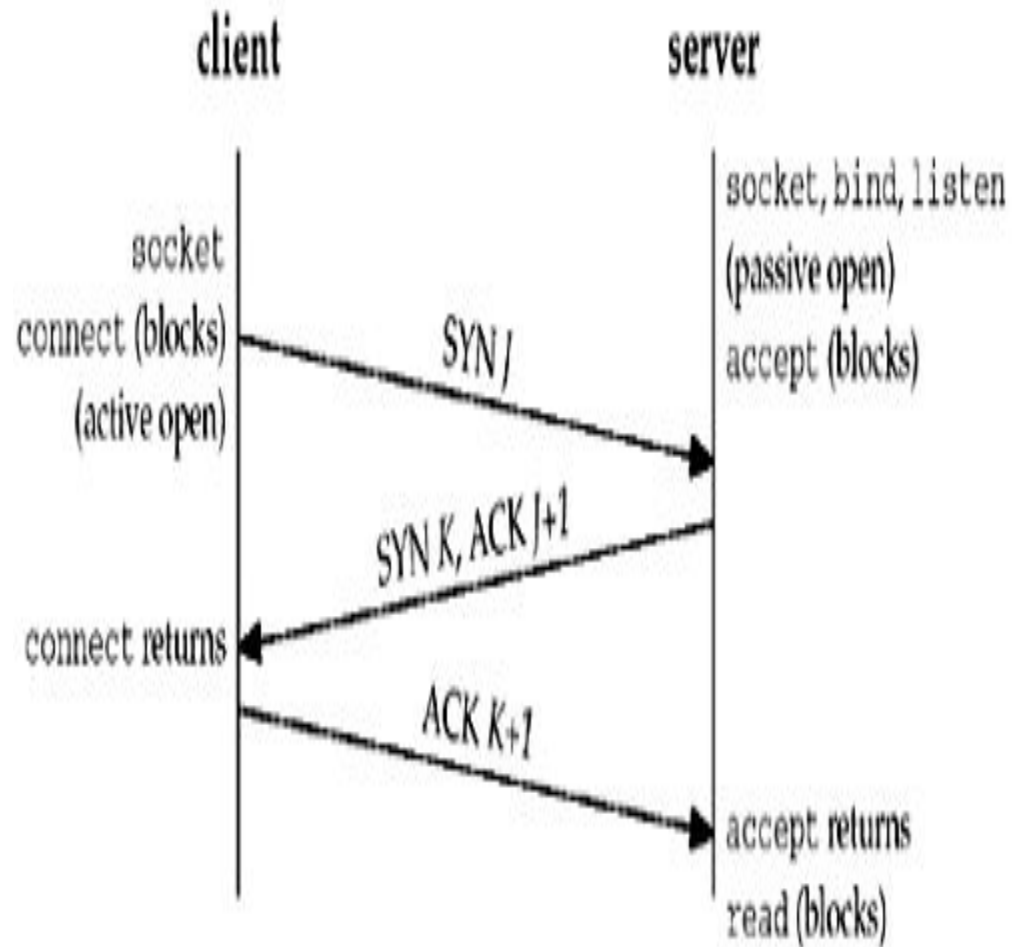
# TCP/IP – The Big-picture





# TCP Connection Establishment and Termination

## Three-Way Handshake



- The following scenario occurs when a TCP connection is established:

1. The server must be prepared to accept an incoming connection. This is normally done by calling `socket`, `bind`, and `listen` and is called a *passive open*.
2. The client issues an *active open* by calling `connect`. This causes the client TCP to send a "synchronize" (SYN) segment, which tells the server the client's initial sequence number for the data that the client will send on the connection. Normally, there is no data sent with the SYN; it just contains an IP header, a TCP header, and possible TCP options
3. The server must acknowledge (ACK) the client's SYN and the server must also send its own SYN containing the initial sequence number for the data that the server will send on the connection. The server sends its SYN and the ACK of the client's SYN in a single segment.
4. The client must acknowledge the server's SYN.

- The minimum number of packets required for this exchange is three; hence, this is called TCP's *three-way handshake*.

- **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*
- **Window scale** option. The maximum window that either TCP can advertise to the other TCP is **65,535**
- **Timestamp option**. This option is needed for **high-speed connections** to prevent possible data corruption caused by old, delayed, or duplicated segments..

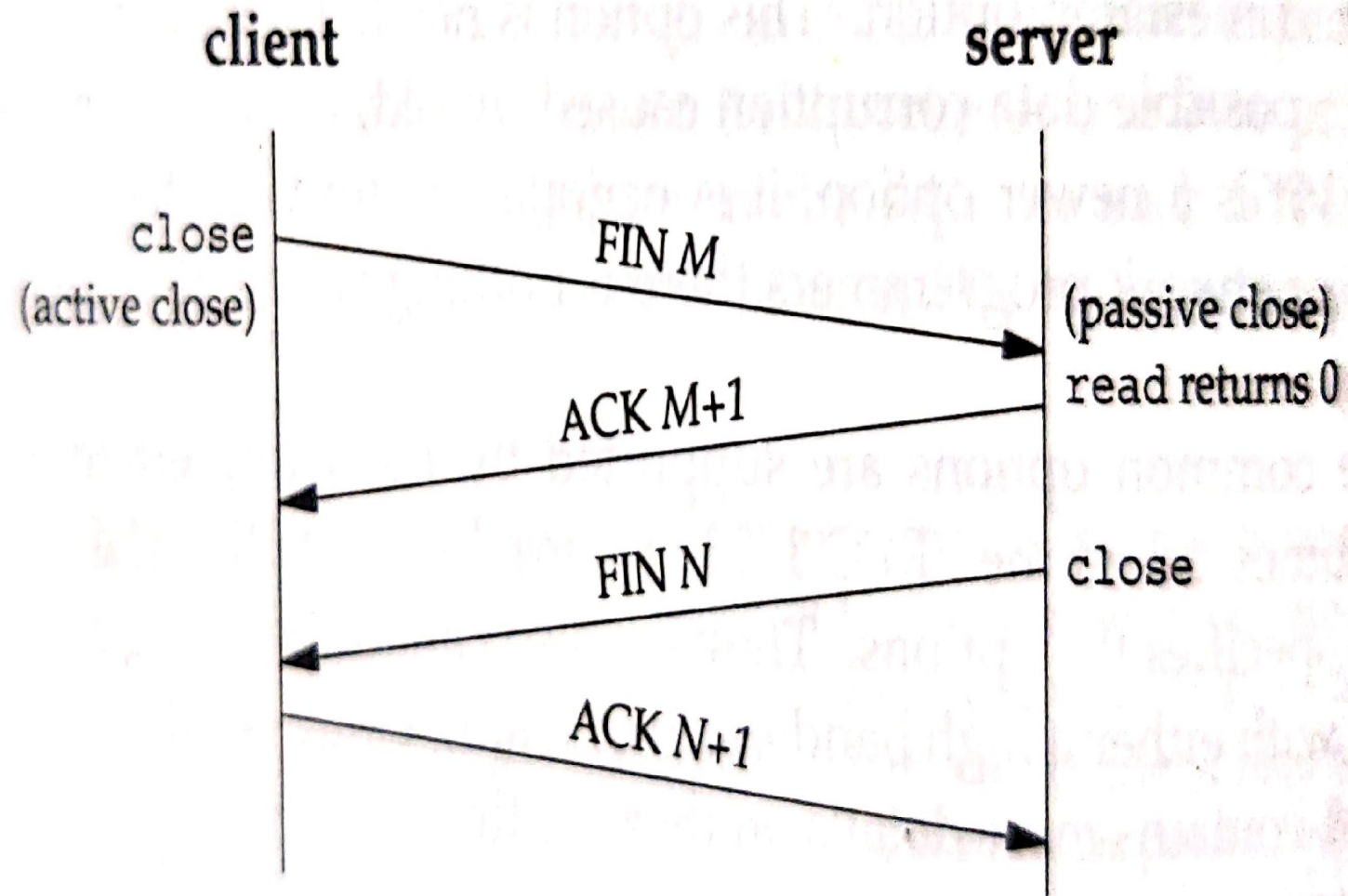


Figure 2.3 Packets exchanged when a TCP connection is closed.

# TCP Connection Termination

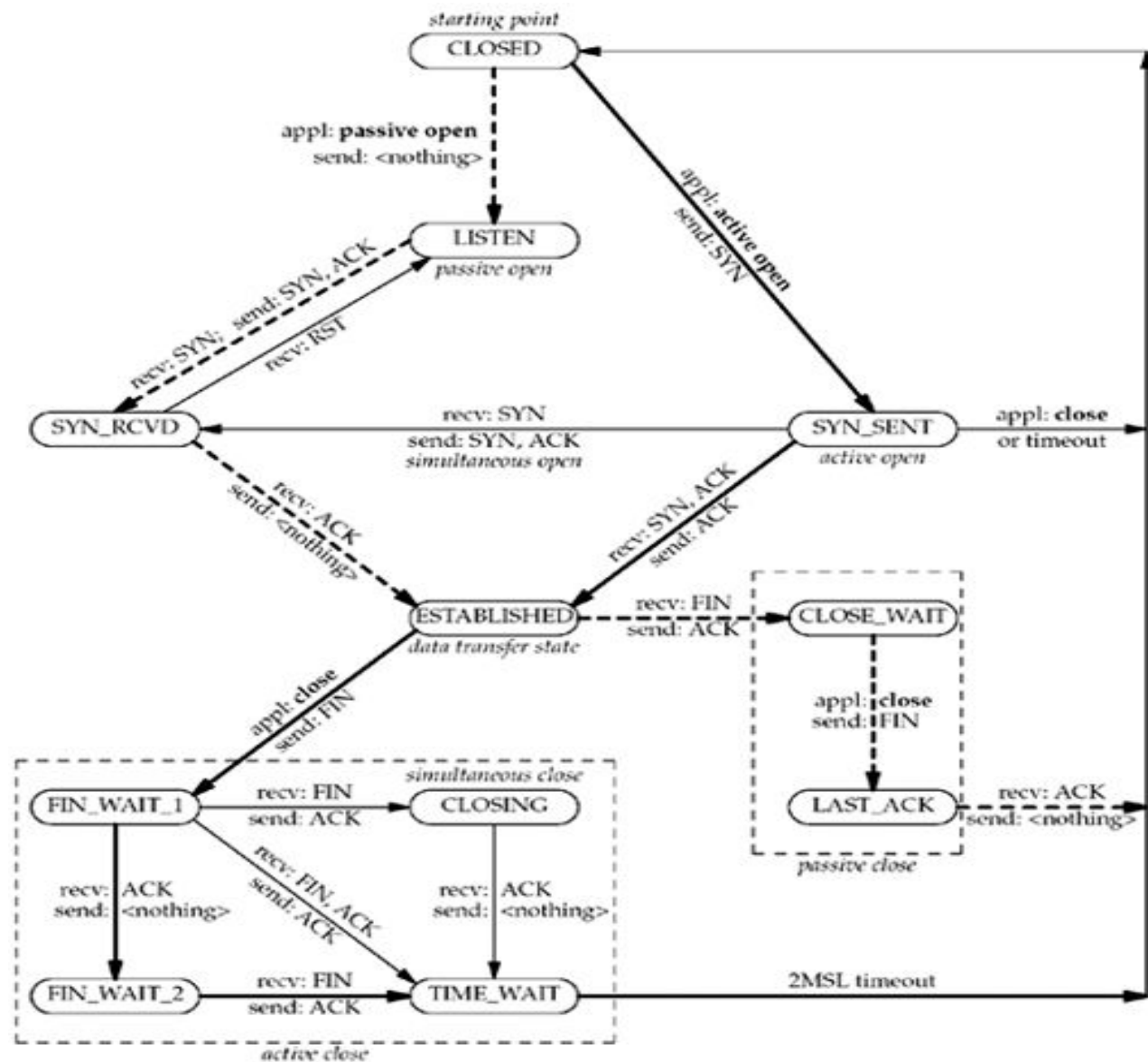
- While it takes three segments to establish a connection, it takes four to terminate a connection.
1. One application calls *close* first, and we say that this end performs the *active close*. This end's TCP sends a *FIN* segment, which means it is finished sending data.
  2. The other end that receives the *FIN* performs the *passive close*. The received *FIN* is acknowledged by TCP. The receipt of the *FIN* is also passed to the application as an end-of-file (after any data that may have already been queued for the application to receive), since the receipt of the *FIN* means the application will not receive any additional data on the connection.
  3. Sometime later, the application that received the end-of-file will close its socket. This causes its TCP to send a *FIN*.
  4. The TCP on the system that receives this final *FIN* (the end that did the active close) acknowledges the *FIN*.

# TCP State Transition Diagram

- The operation of TCP with regard to connection establishment and connection termination can be specified with a *state transition diagram*.
- There are 11 different states defined for a connection and the rules of TCP dictate the transitions from one state to another, based on the current state and the segment received in that state.
- For example, if an application performs an active open in the CLOSED state, TCP sends a SYN and the new state is SYN\_SENT.
- If TCP next receives a SYN with an ACK, it sends an ACK and the new state is ESTABLISHED. This final state is where most data transfer occurs.

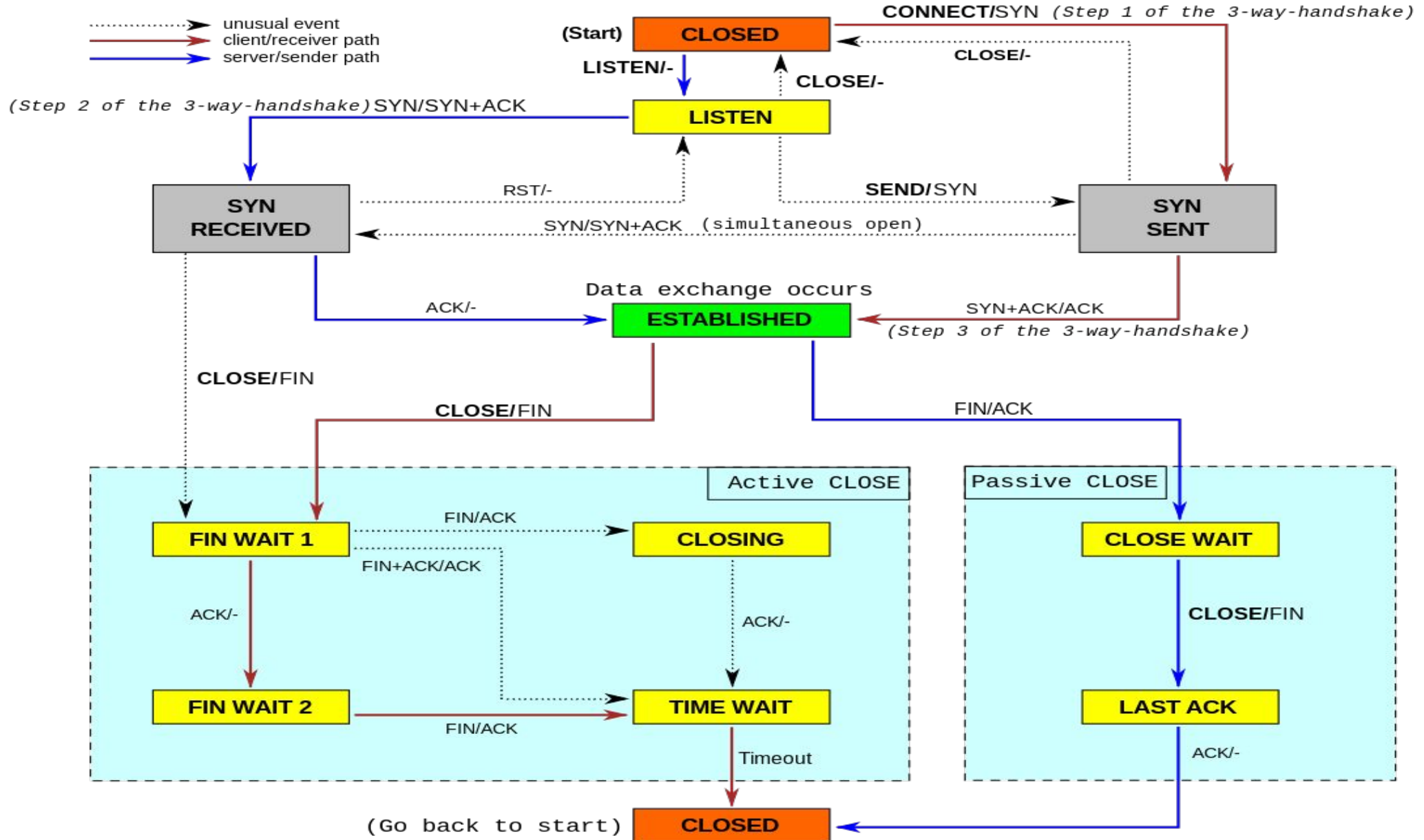
- The two arrows leading from the ESTABLISHED state deal with the termination of a connection.
- If an application calls close before receiving a FIN (an active close), the transition is to the FIN\_WAIT\_1 state.
- But if an application receives a FIN while in the ESTABLISHED state (a passive close), the transition is to the CLOSE\_WAIT state.



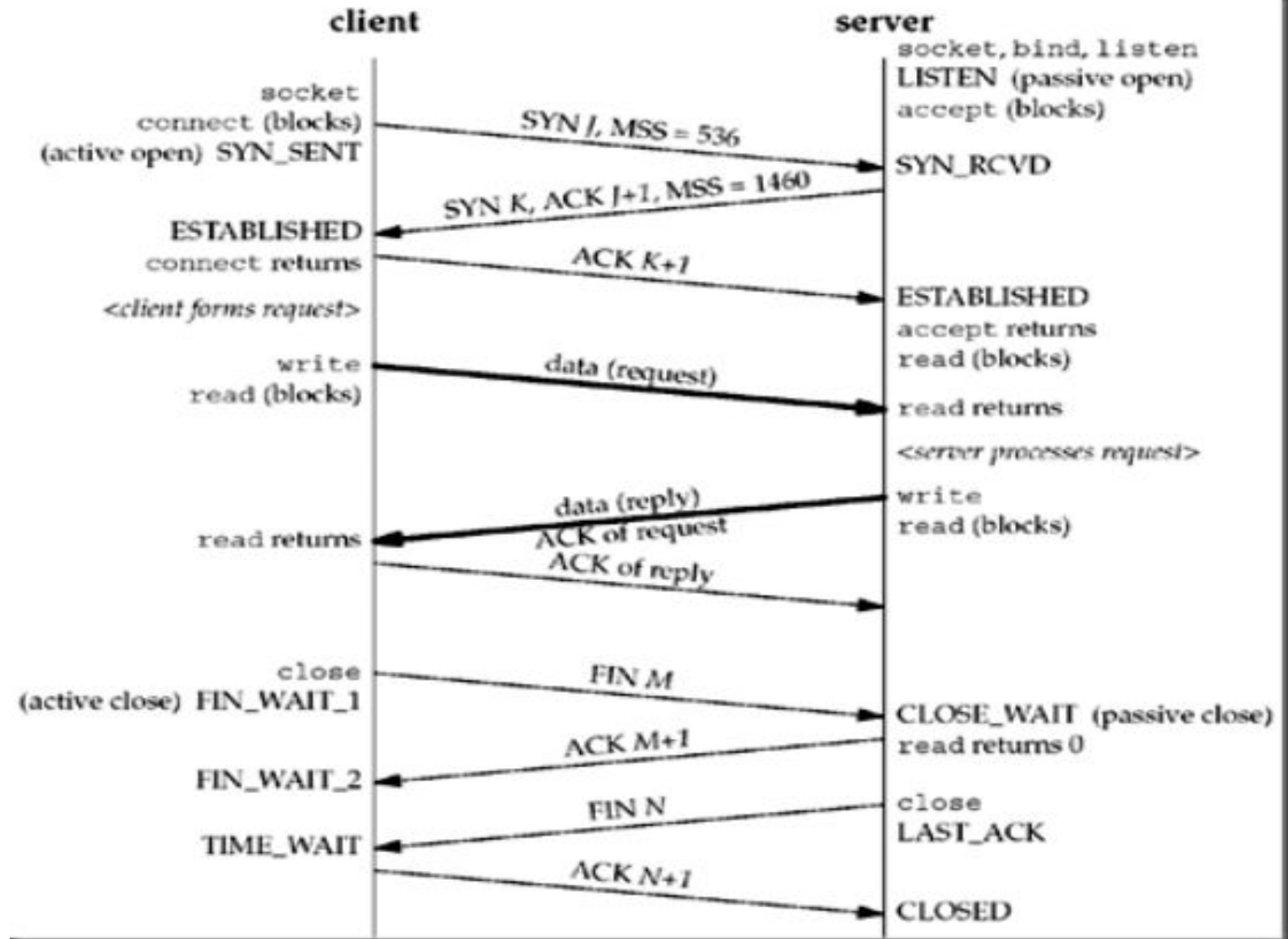


—————> indicate normal transitions for client  
 - - - - -> indicate normal transitions for server  
 appl: indicate state transitions taken when application issues operation  
 recv: indicate state transitions taken when segment received  
 send: indicate what is sent for this transition

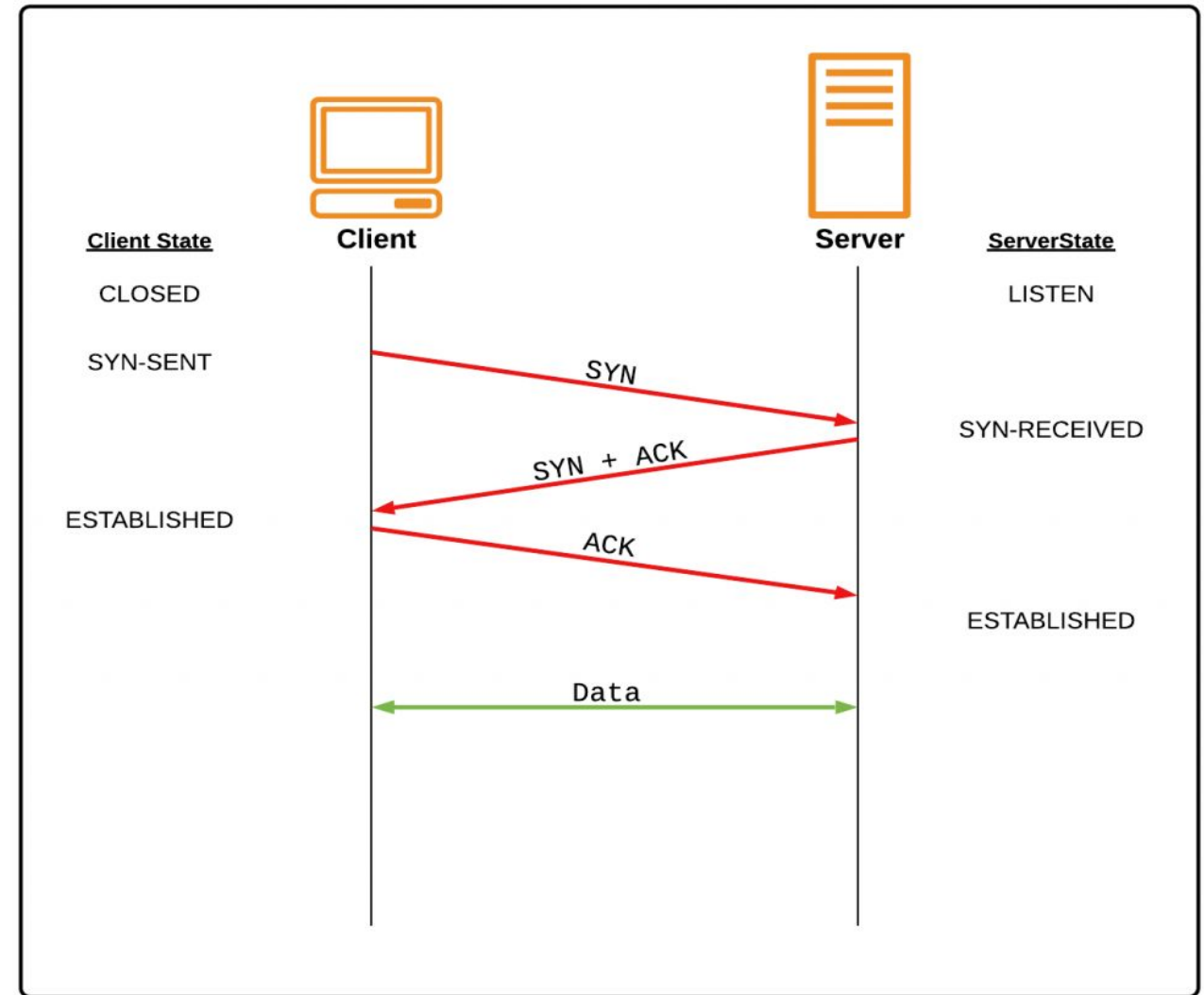
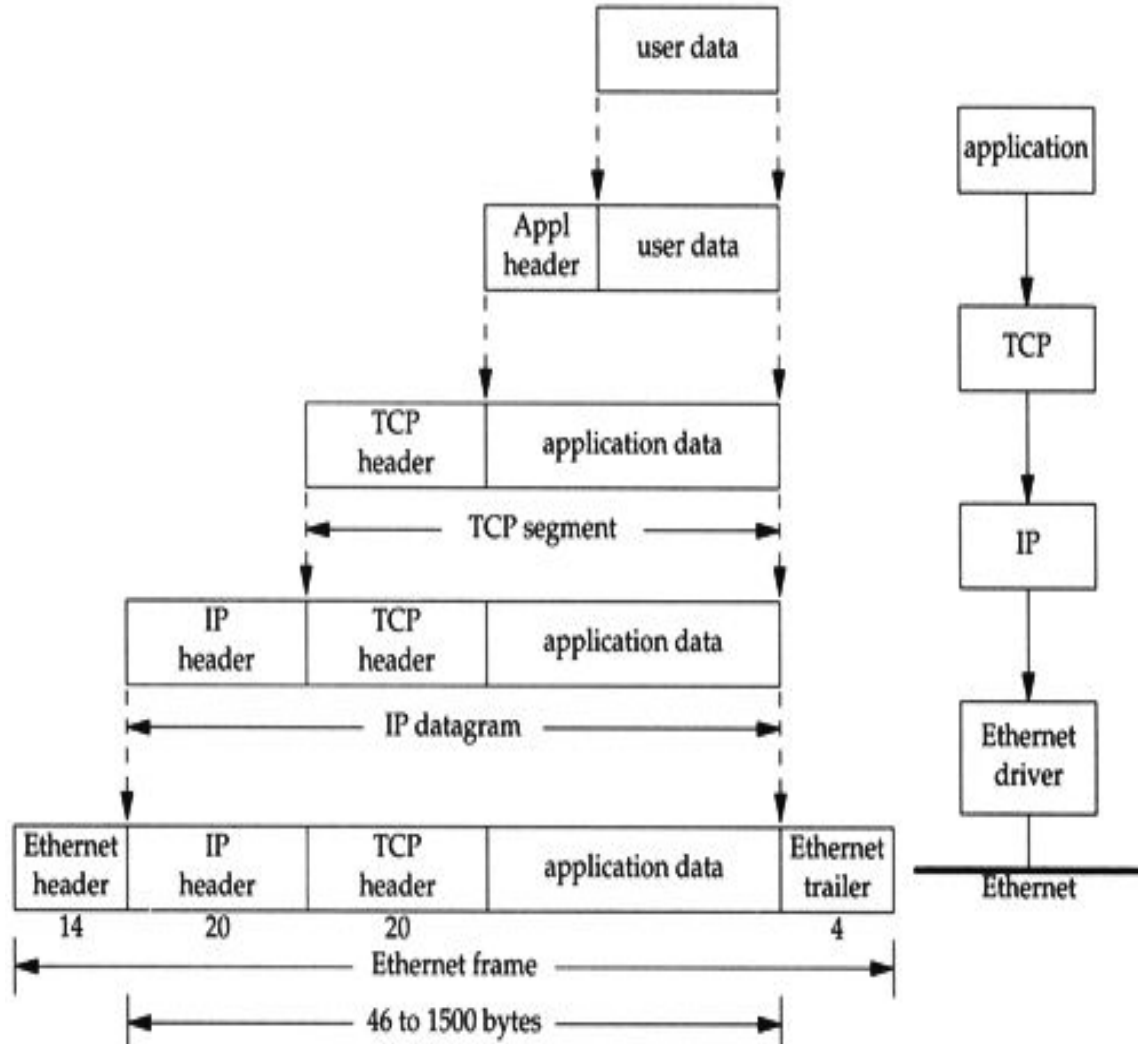
## TCP-Connection state diagram



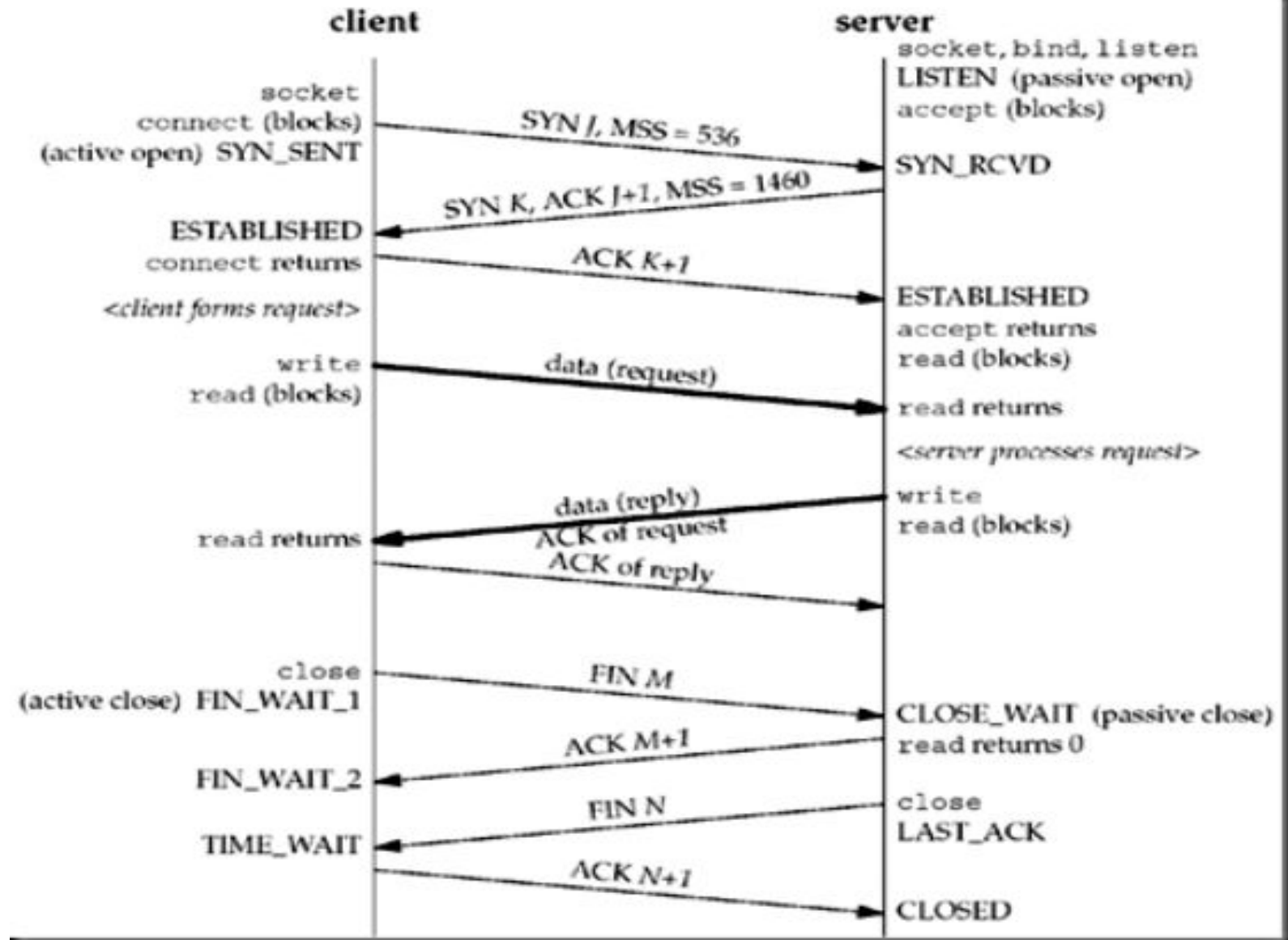
# TCP – Connection : Packet Exchange



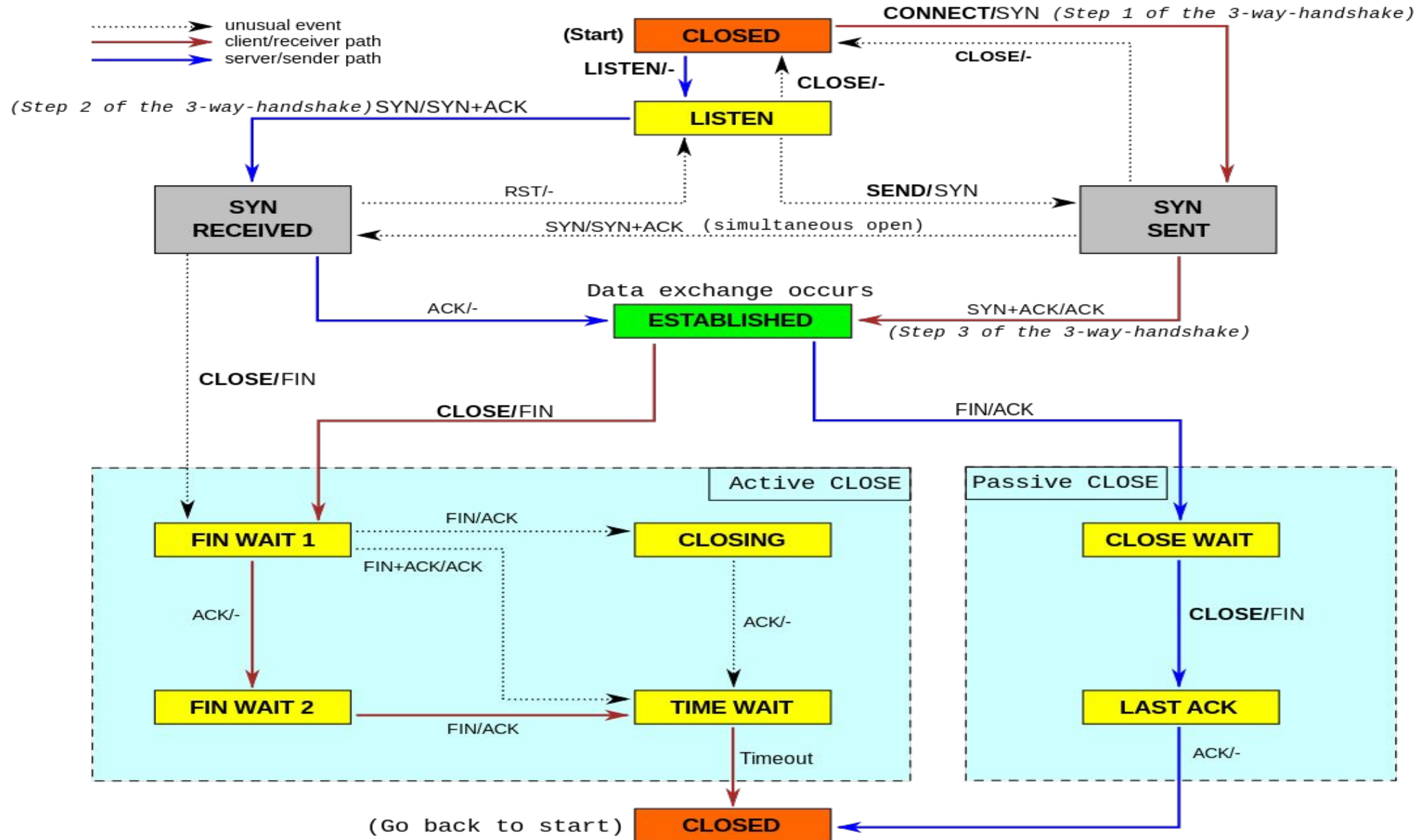
# TCP - Connection



# TCP – Connection : Packet Exchange



# TCP-Connection state diagram



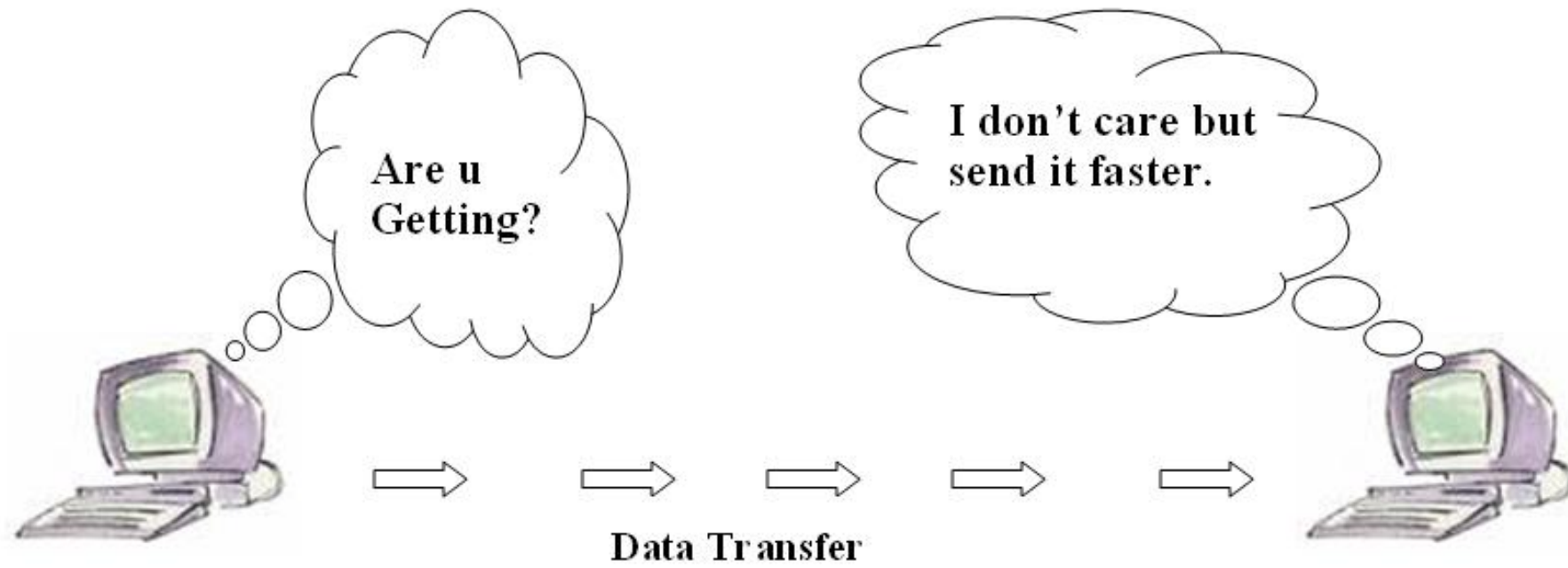


## TCP Use cases

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

# UDP

## UDP



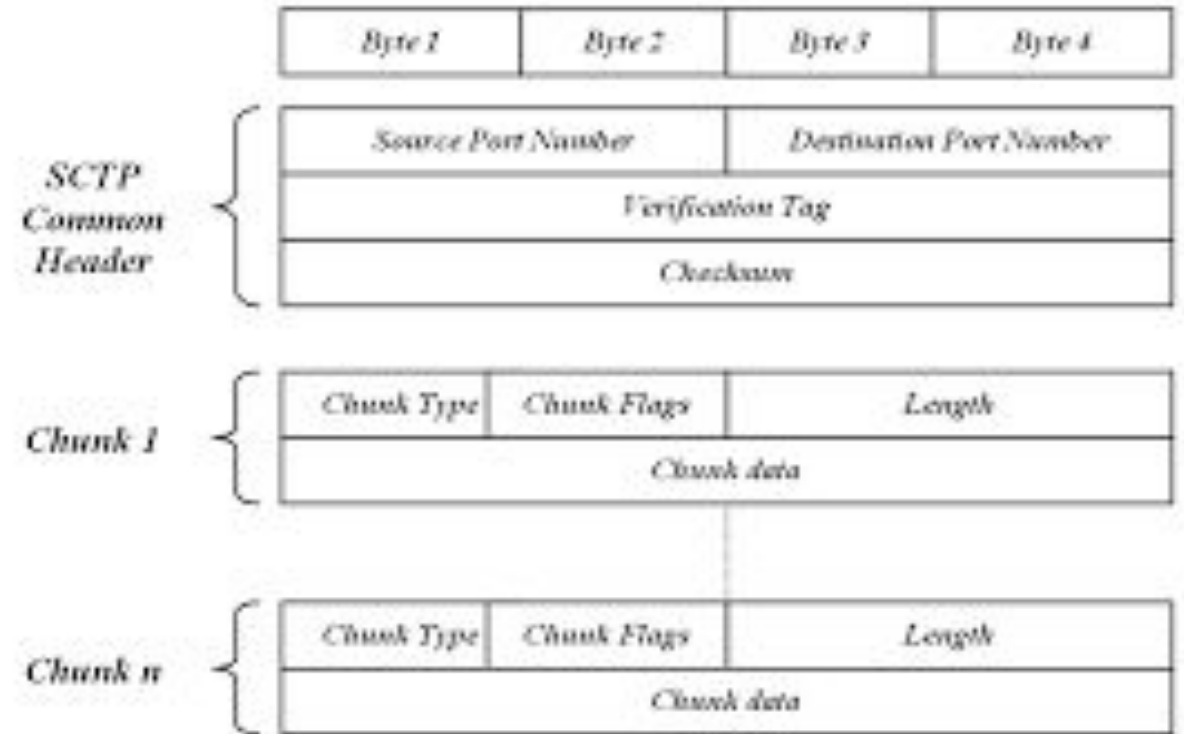
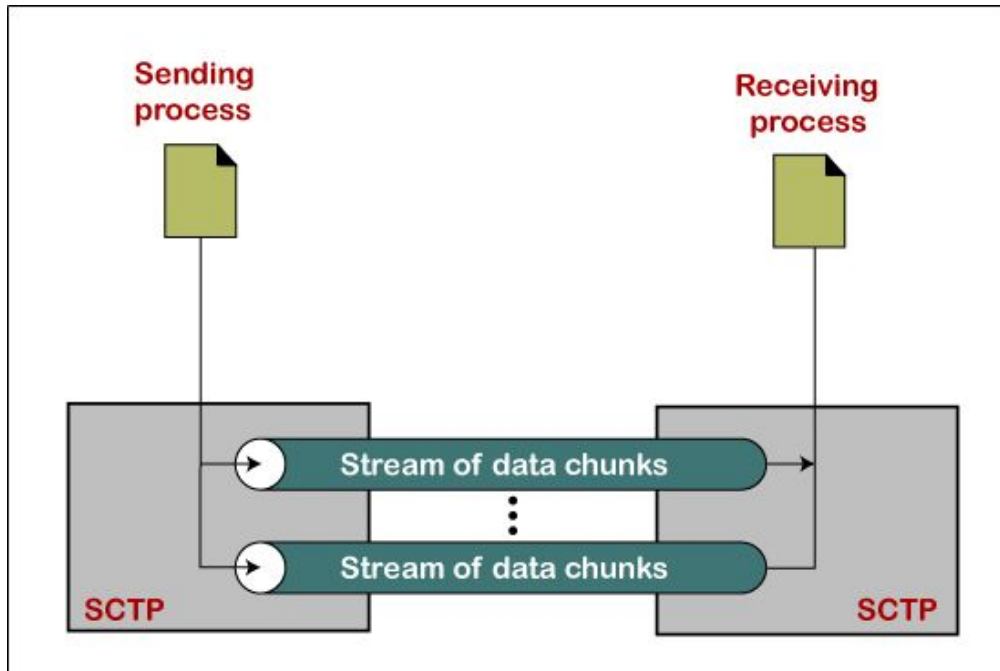


# UDP Use cases

## UDP Applications

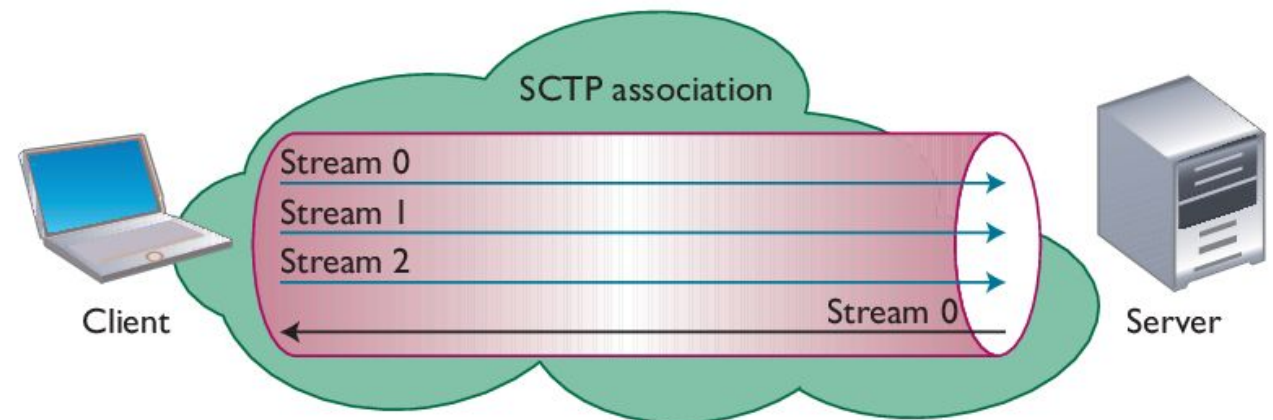
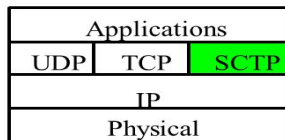
- Used for applications that can tolerate small amount of packet loss:
  - Multimedia applications,
  - Internet telephony,
  - real-time-video conferencing
  - Domain Name System messages
  - Audio
  - Routing Protocols

# SCTP

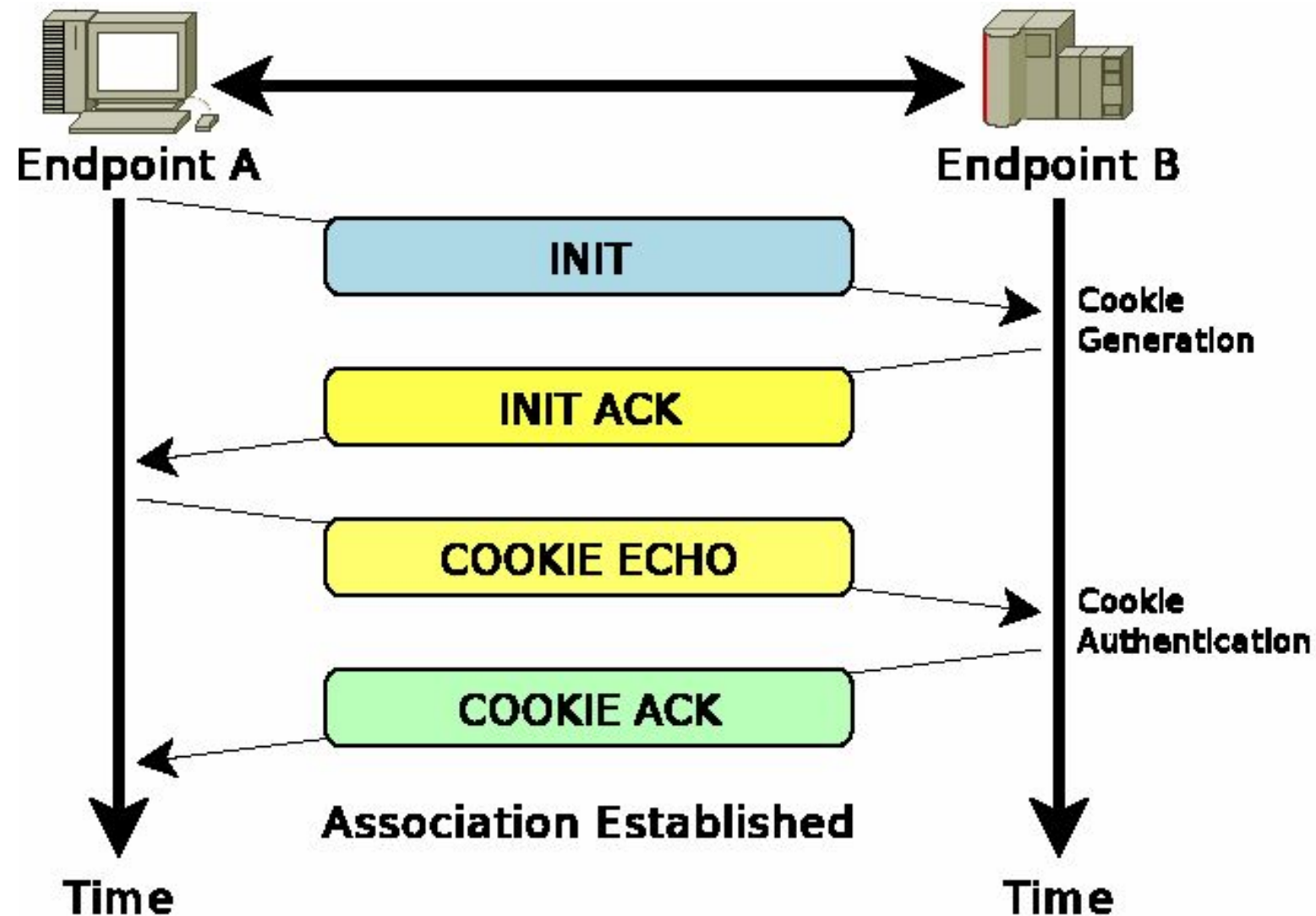


## What is SCTP?

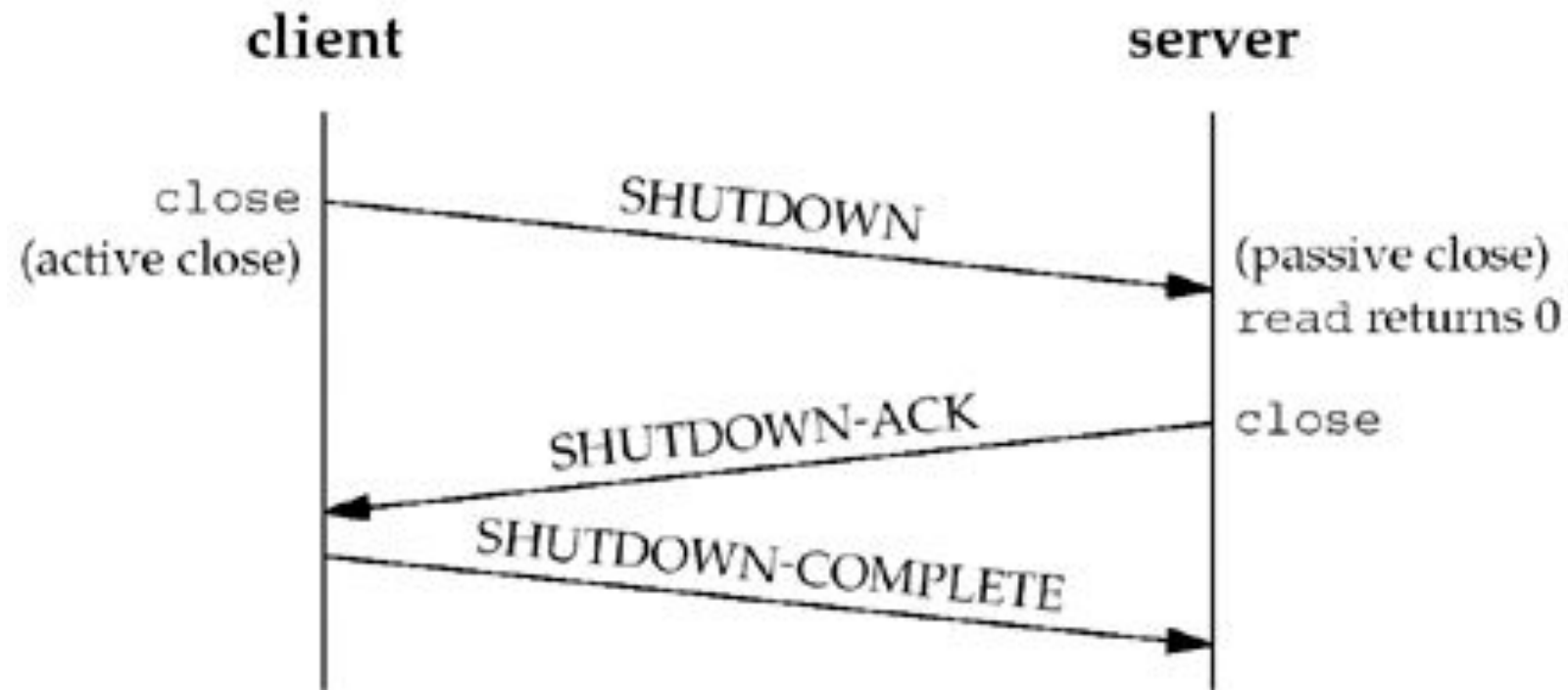
- SCTP is Stream Control Transmission Protocol, a Transport layer protocol.
- SCTP is reliable data transfer protocol which operates over the Network layer protocol like IP.



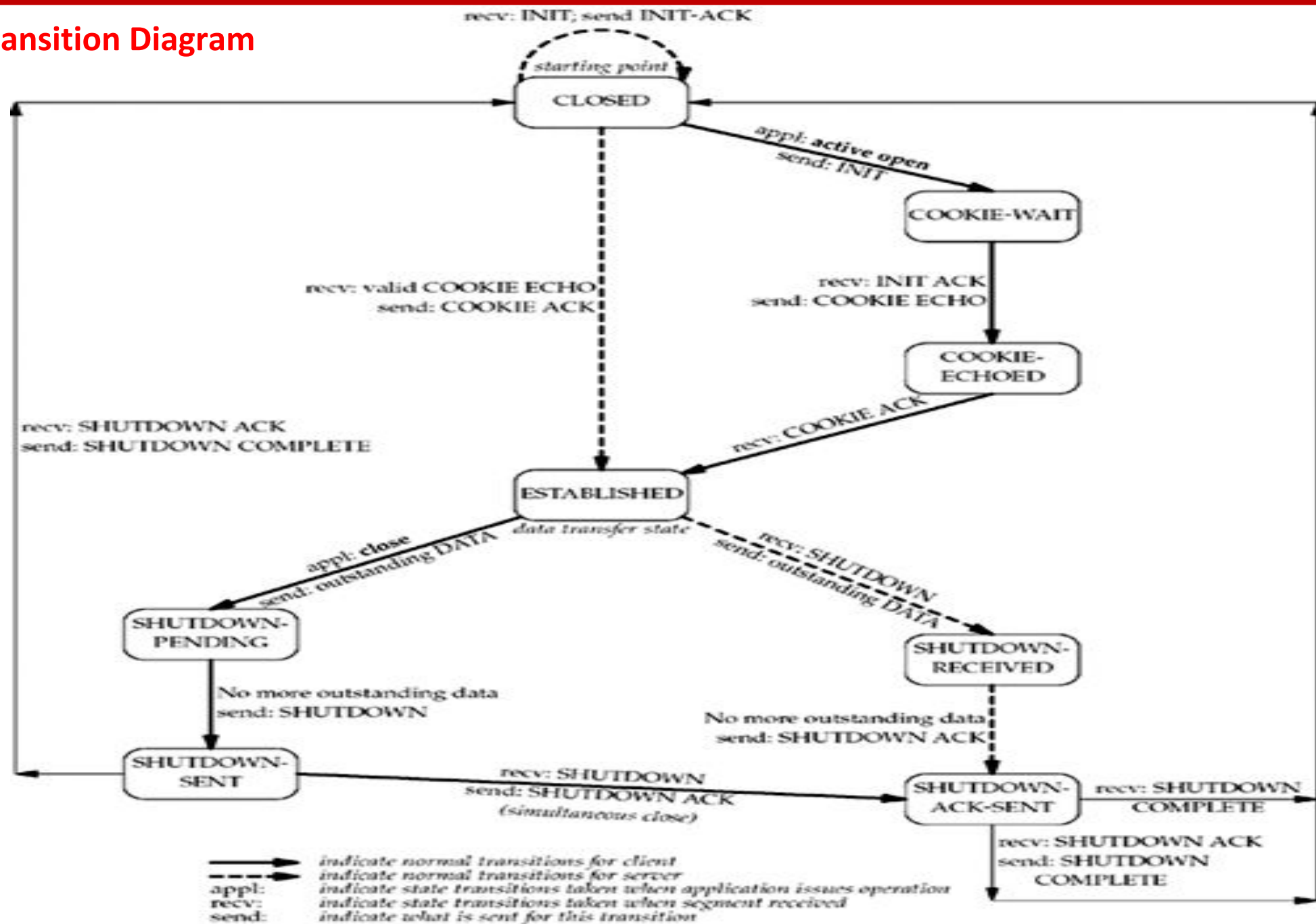
# SCTP – 4 way Handshake



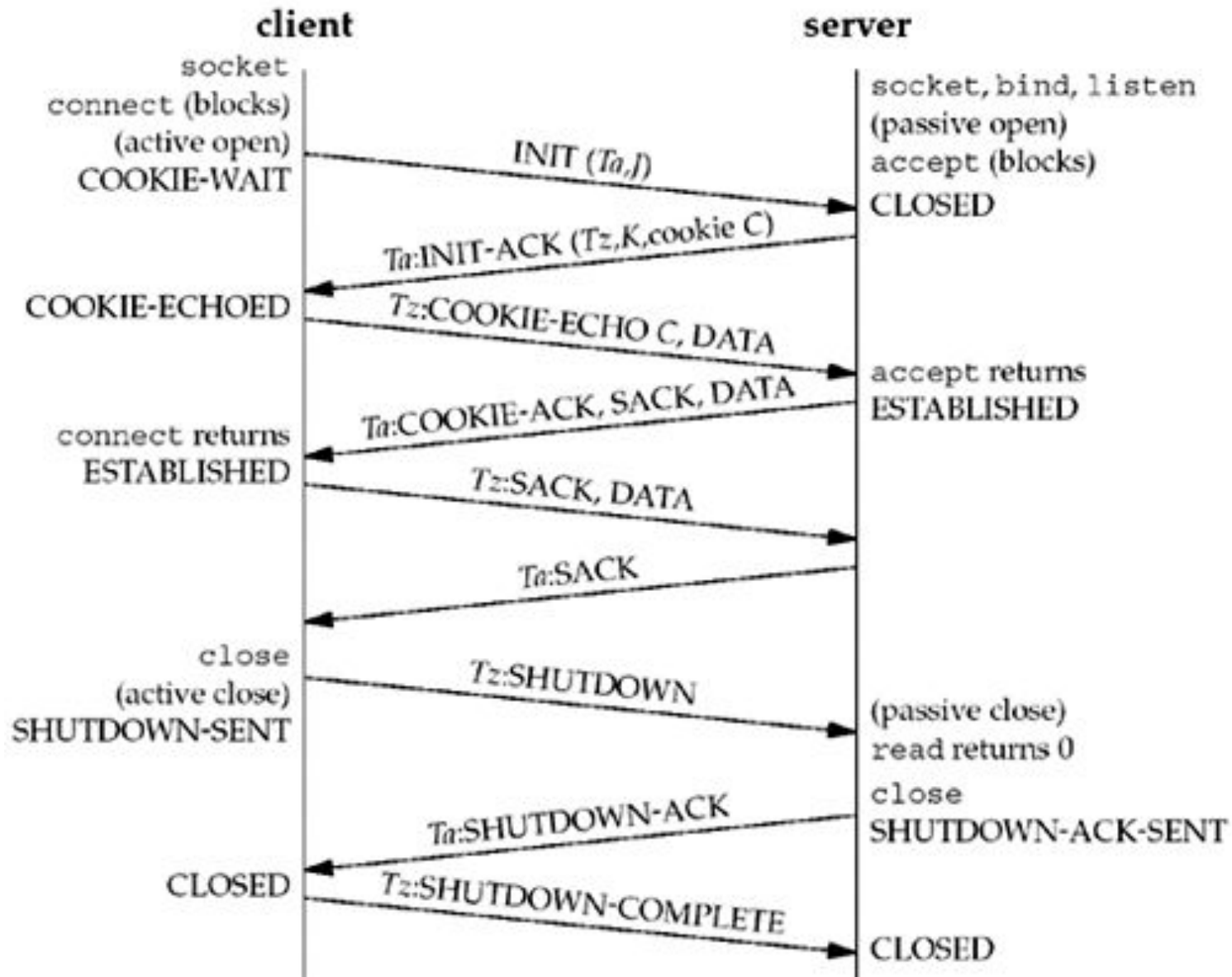
## SCTP – Closing



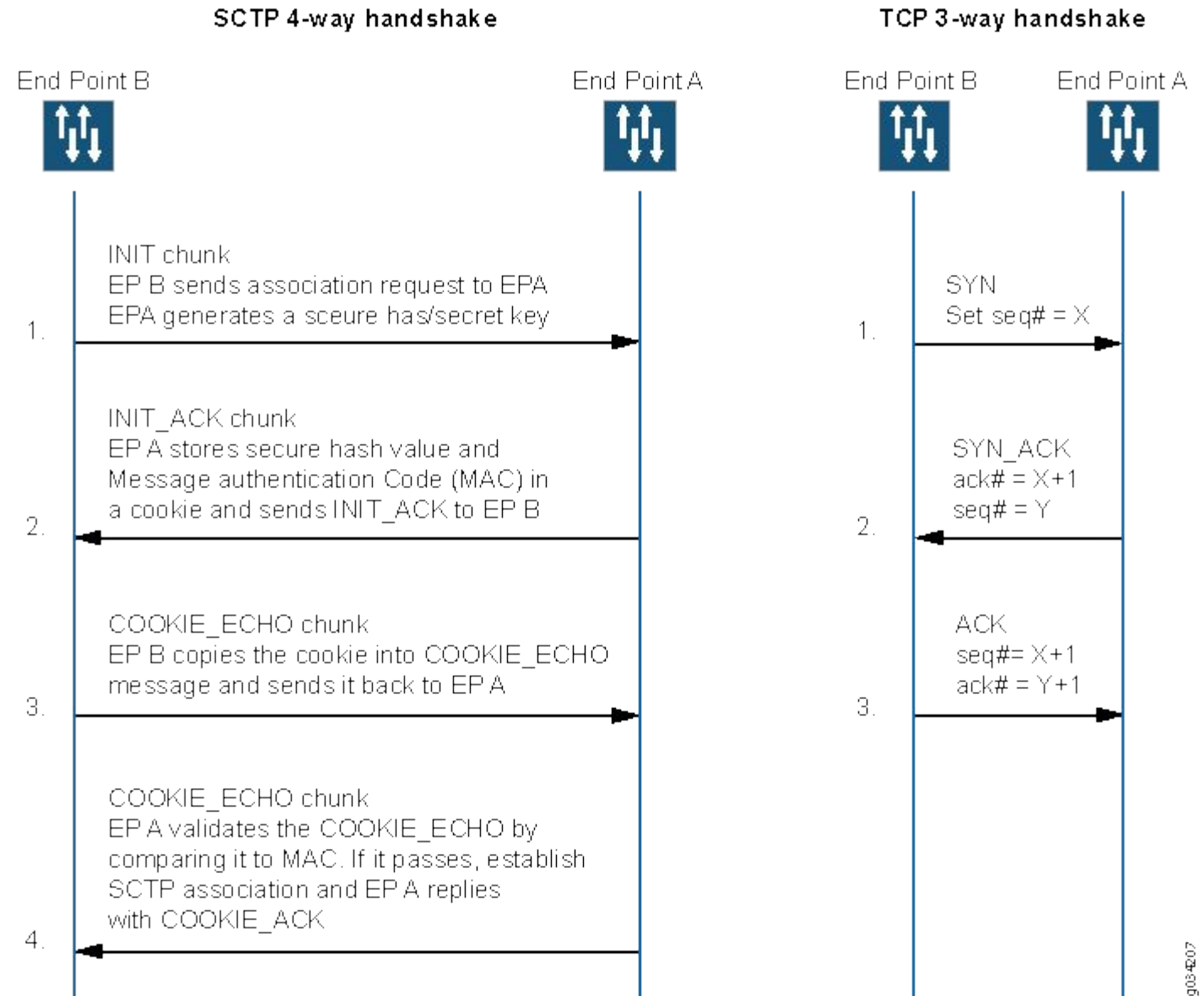
## SCTP – State Transition Diagram



## SCTP – Packet Exchange



# SCTP – 4 way handshake





# 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



# Review Questions

1. What is network programming? With neat diagram ,Explain the Client and Server Communication over LAN and WAN
2. Write a program to implement TCP daytime client
3. Write a program to implement TCP daytime client for IPV6
4. Explain the Error Handling using Wrapper functions
5. Write a program to implement TCP daytime Server
6. Explain the layers in the OSI model and Internet Protocol suite
7. Write a brief note on BSD Networking History
8. Write a short note on various UNIX standards
9. With the neat diagram give the overview of TCP/IP Protocol
10. Write a short note on i) TCP ii) UDP iii) SCTP protocols

11. Explain in detail, TCP Connection Establishment and Termination
12. Explain TCP state Transition Diagram
13. With neat diagram, explain Packet Exchange for TCP connection
14. Explain Sockaddr\_in structure and its parts.
15. Give the comparison on TCP, UDP and SCTP protocols