

# Data and Computer Communications

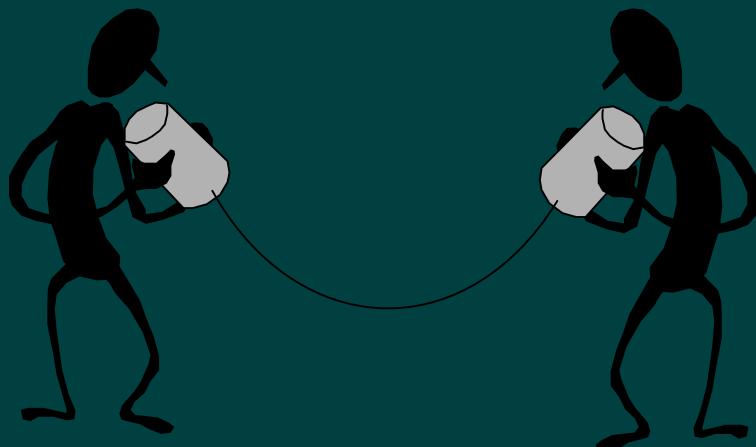
Tenth Edition  
by William Stallings

# CHAPTER 2

**Protocol Architecture, TCP/IP, and  
Internet-Based Applications**

*To destroy communication completely, there must be no rules in common between transmitter and receiver—neither of alphabet nor of syntax.*

*—On Human Communication,*  
Colin Cherry



# The Need for a Protocol Architecture

1.) The source must either activate the direct communications path or inform the network of the identity of the desired destination system

2.) The source system must ascertain that the destination system is prepared to receive data

To transfer data several tasks must be performed:

3.) The file transfer application on the source system must ascertain that the file management program on the destination system is prepared to accept and store the file for this particular user

4.) A format translation function may need to be performed by one or the other system if the file formats used on the two systems are different

# Functions of Protocol Architecture

- Breaks logic into subtask modules which are implemented separately
- Modules are arranged in a vertical stack
  - Each layer in the stack performs a subset of functions
  - Relies on next lower layer for primitive functions
  - Provides services to the next higher layer
  - Changes in one layer should not require changes in other layers

# Key Features of a Protocol

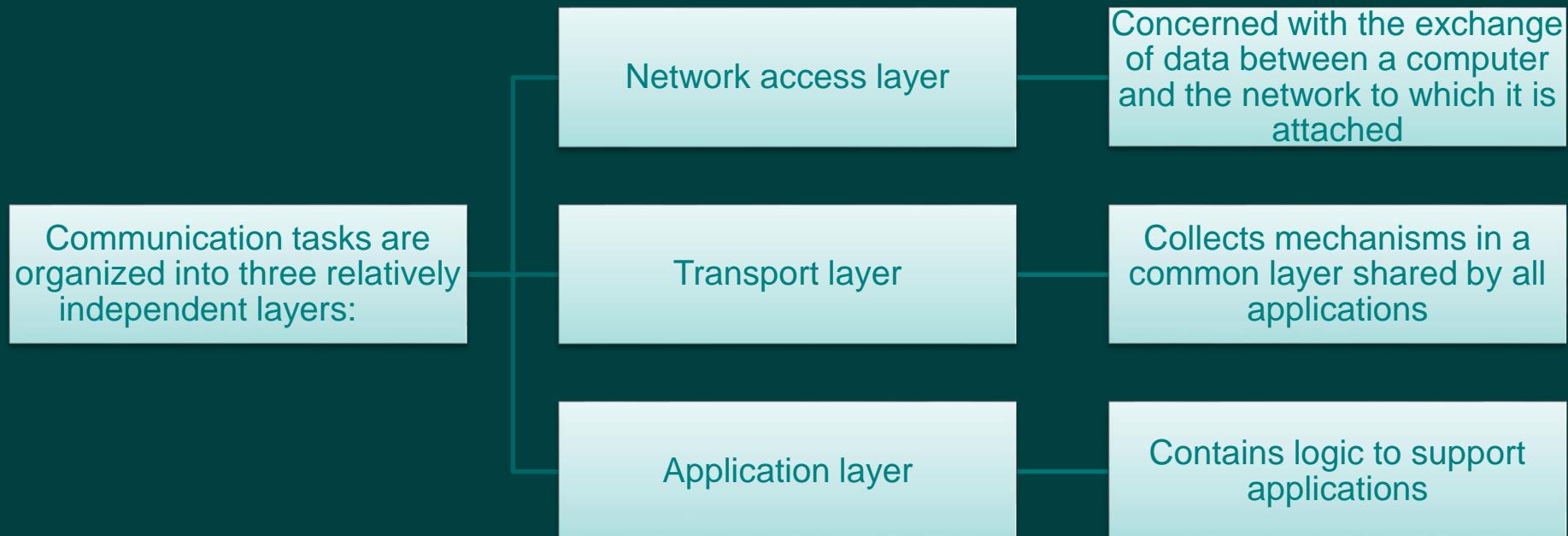
A protocol is a set of rules or conventions that allow peer layers to communicate

The key features of a protocol are:



Syntax	<ul style="list-style-type: none"><li>Format of data blocks</li></ul>
Semantics	<ul style="list-style-type: none"><li>Control information for coordination and error handling</li></ul>
Timing	<ul style="list-style-type: none"><li>Speed matching and sequencing</li></ul>

# Communication Layers



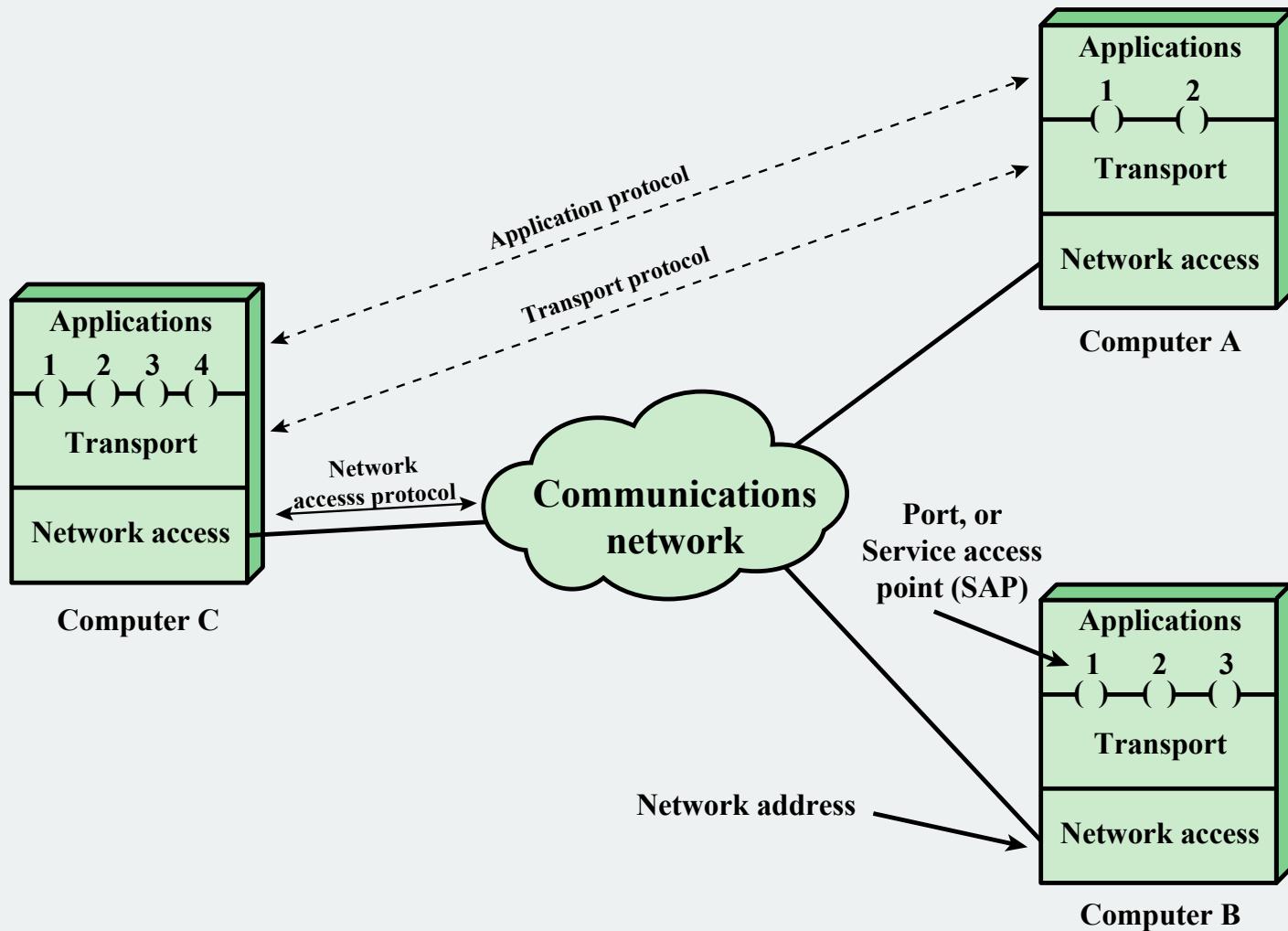


Figure 2.1 Protocol Architecture and Networks

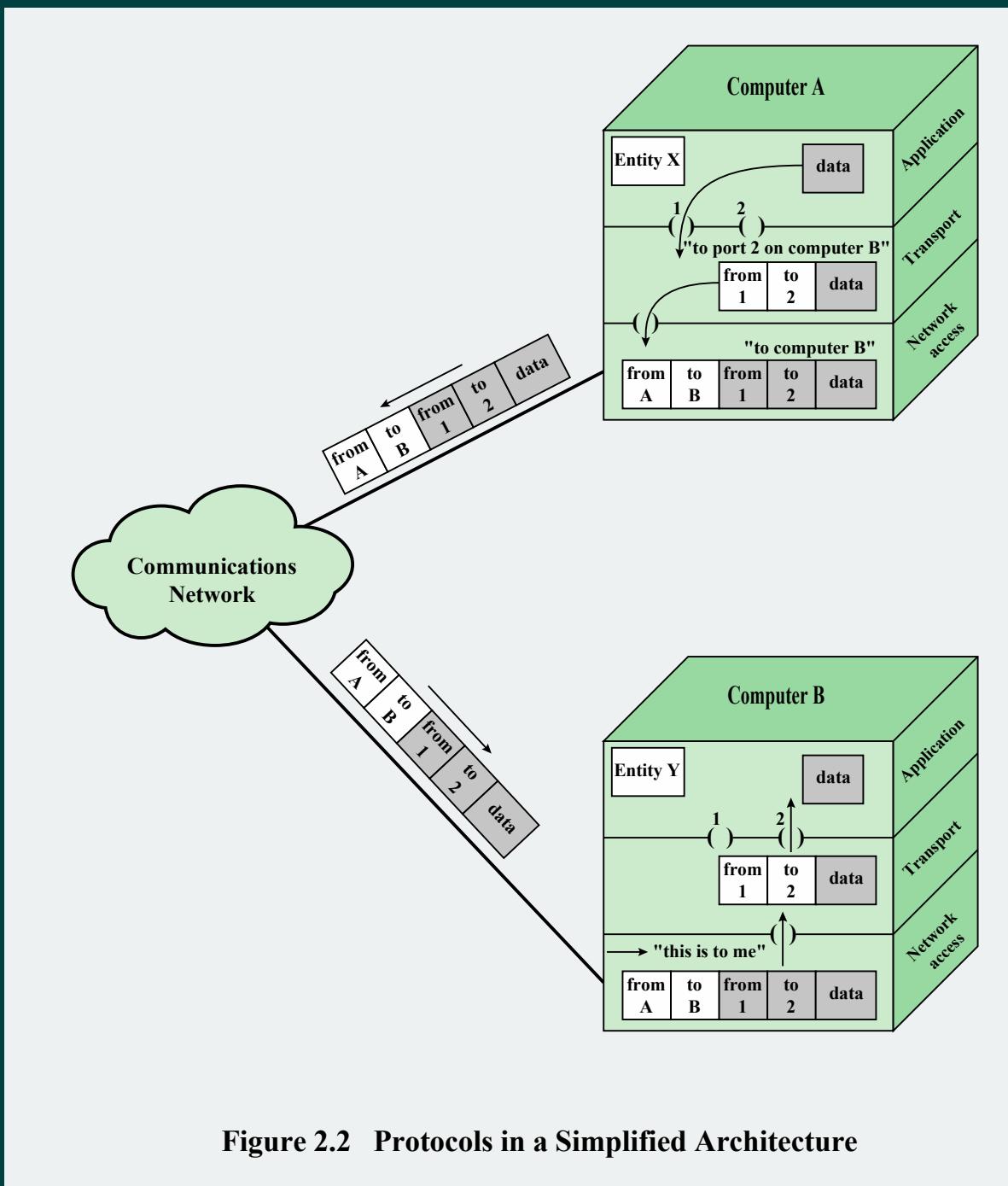
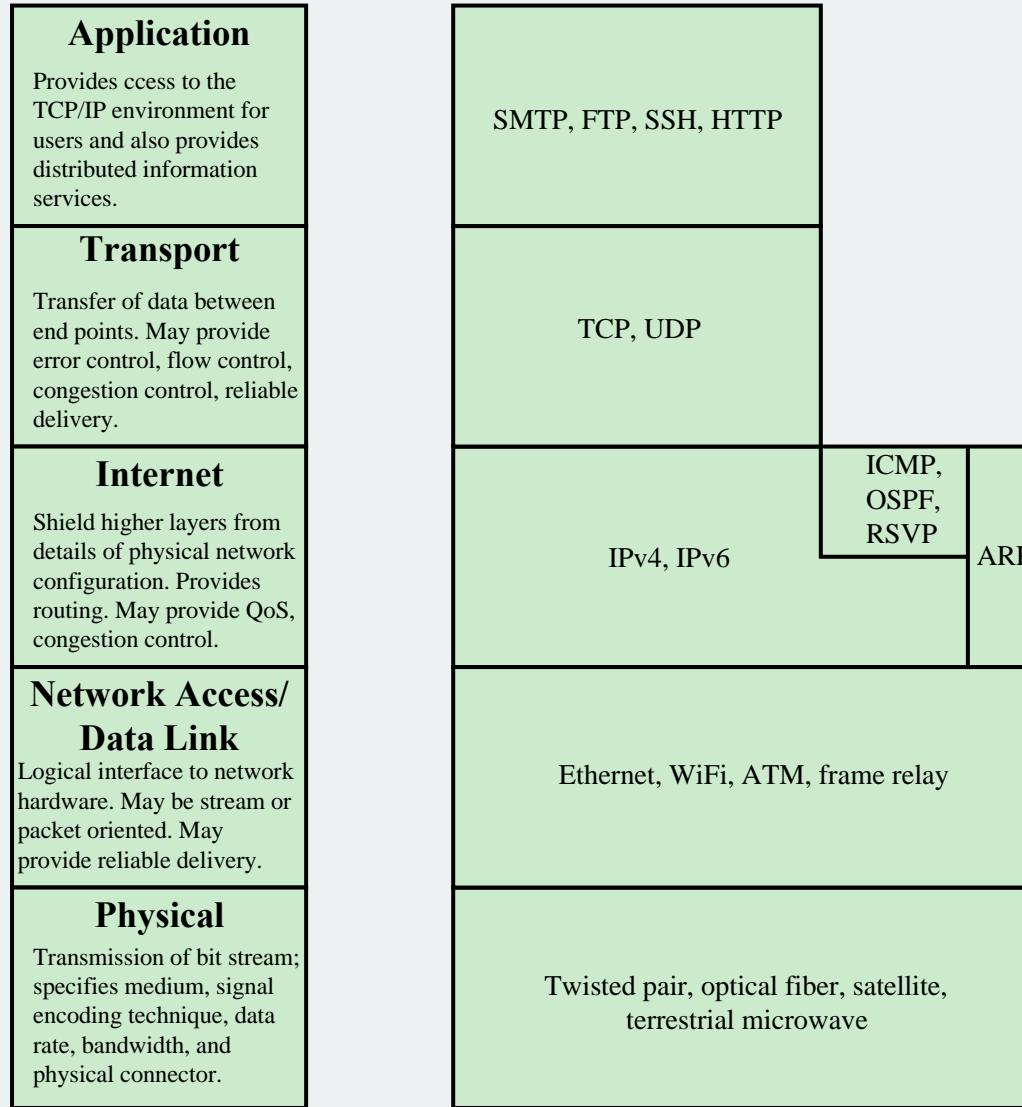


Figure 2.2 Protocols in a Simplified Architecture

# TCP/IP Protocol Architecture

## TCP/IP Protocol Architecture

- Result of protocol research and development conducted on ARPANET
- Referred to as TCP/IP protocol suite
- TCP/IP comprises a large collection of protocols that are Internet standards



**Figure 2.3 The TCP/IP Layers and Example Protocols**

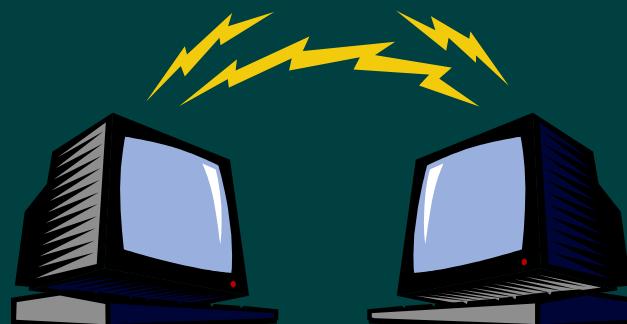
# Physical Layer

- Covers the physical interface between computer and network
- Concerned with issues like:
  - Characteristics of transmission medium
  - Nature of the signals
  - Data rates

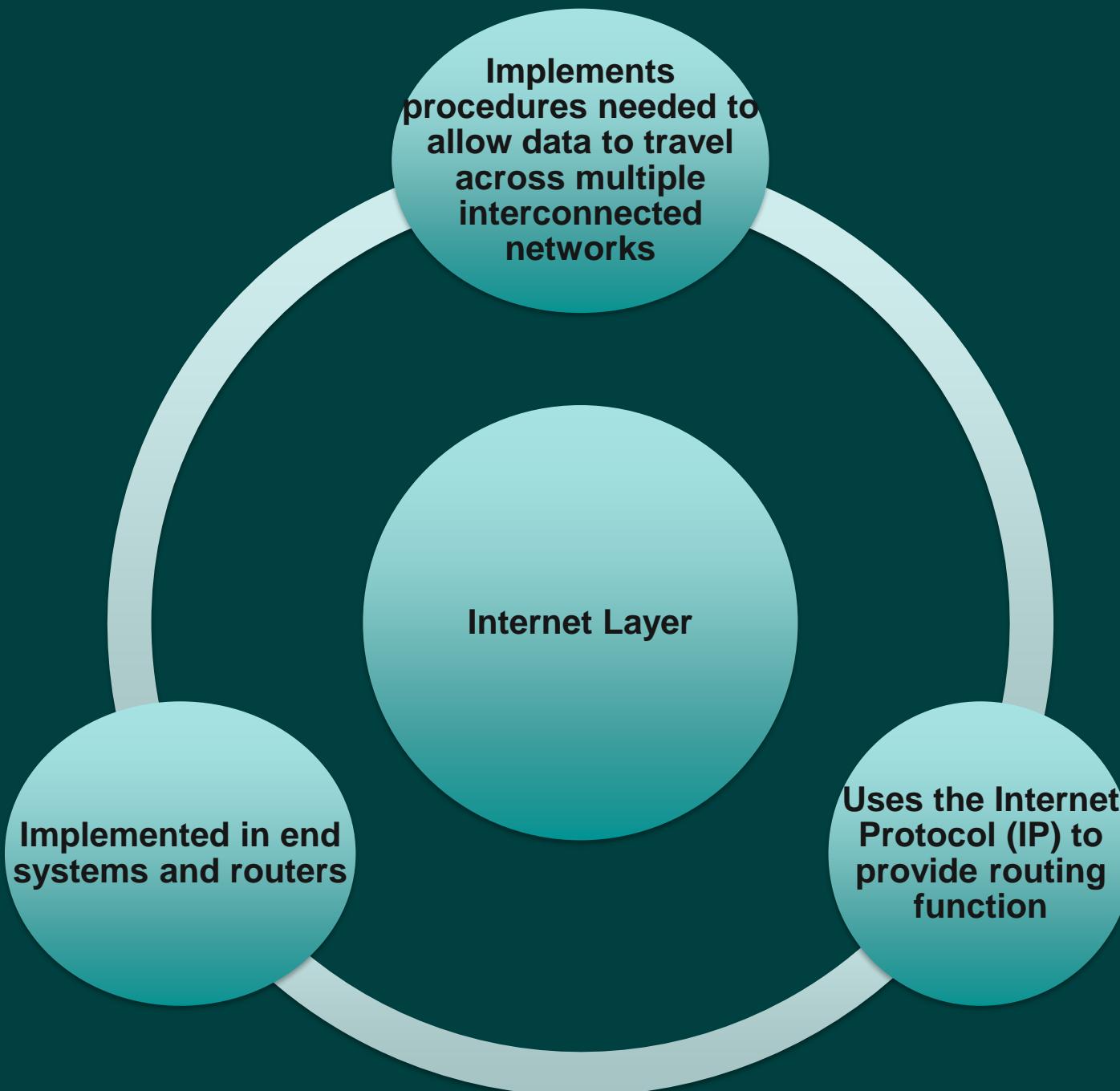


# Network Access/Data Link Layer

- Covers the exchange of data between an end system and the network that it is attached to
- Concerned with:
  - Access to and routing data across a network for two end systems attached to the same network



# Internet Layer



# Host-to-Host (Transport) Layer

- May provide reliable end-to-end service or merely an end-to-end delivery service without reliability mechanisms

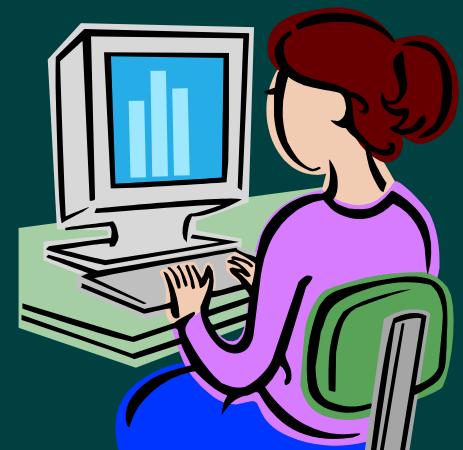
TCP

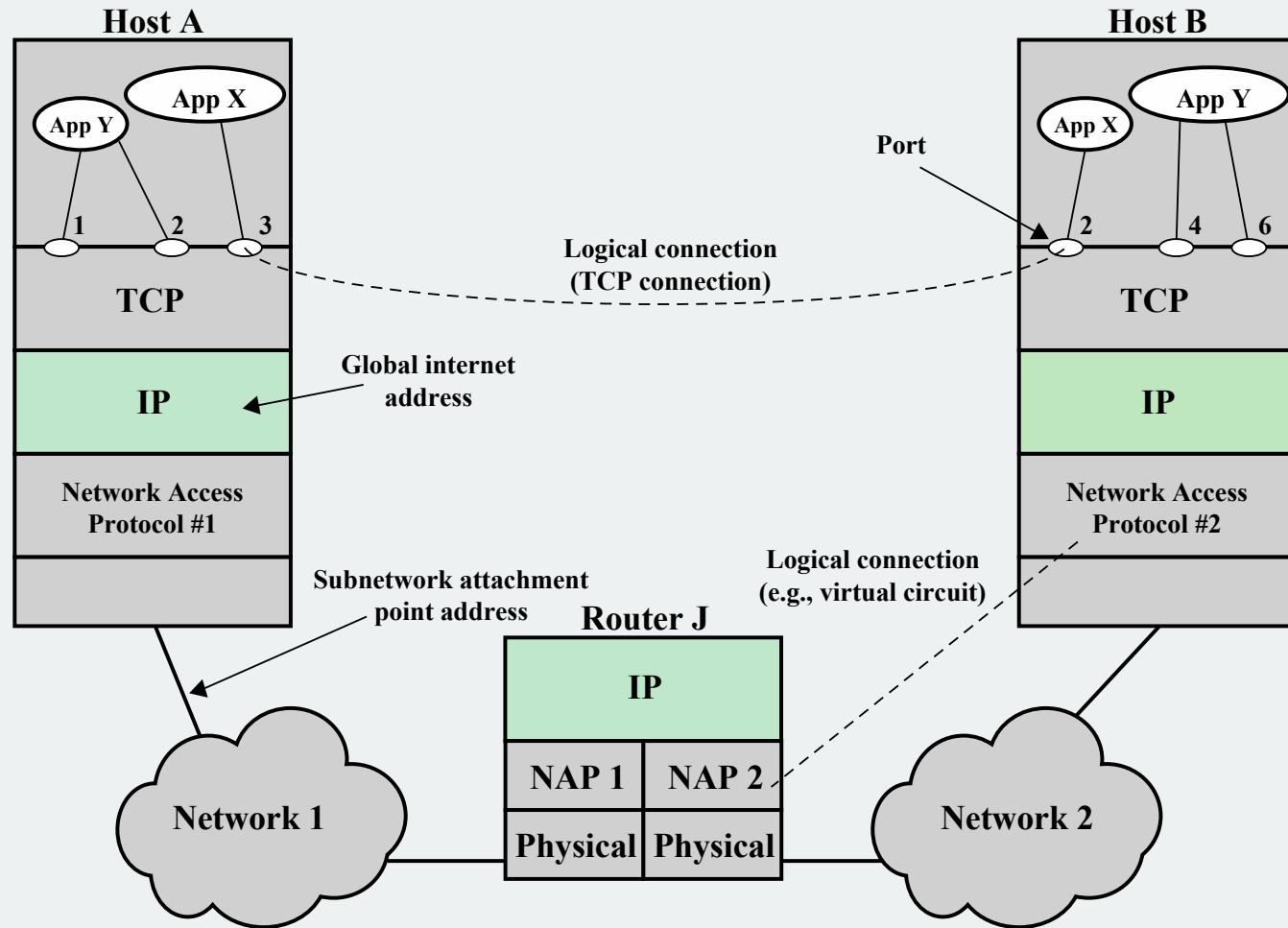
- Most commonly used protocol to provide this functionality

Transmission  
Control Protocol

# Application Layer

- Contains the logic needed to support the various user applications
- A separate module is needed for each different type of application that is peculiar to that application





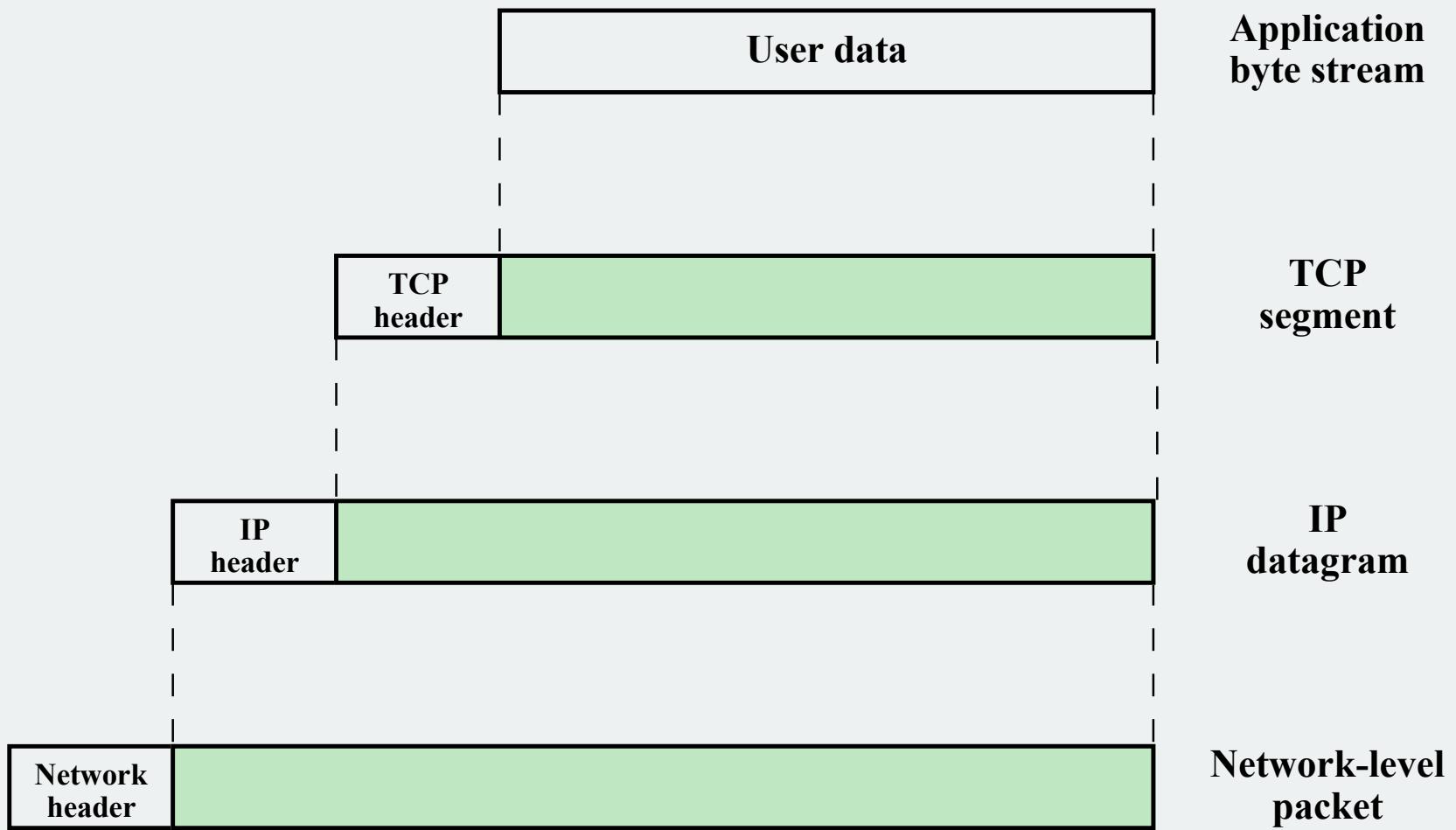
**Figure 2.4 TCP/IP Concepts**

# TCP/IP Address Requirements

Two levels of addressing are needed:

Each host on a subnetwork must have a unique global internet address

Each process with a host must have an address (known as a port) that is unique within the host

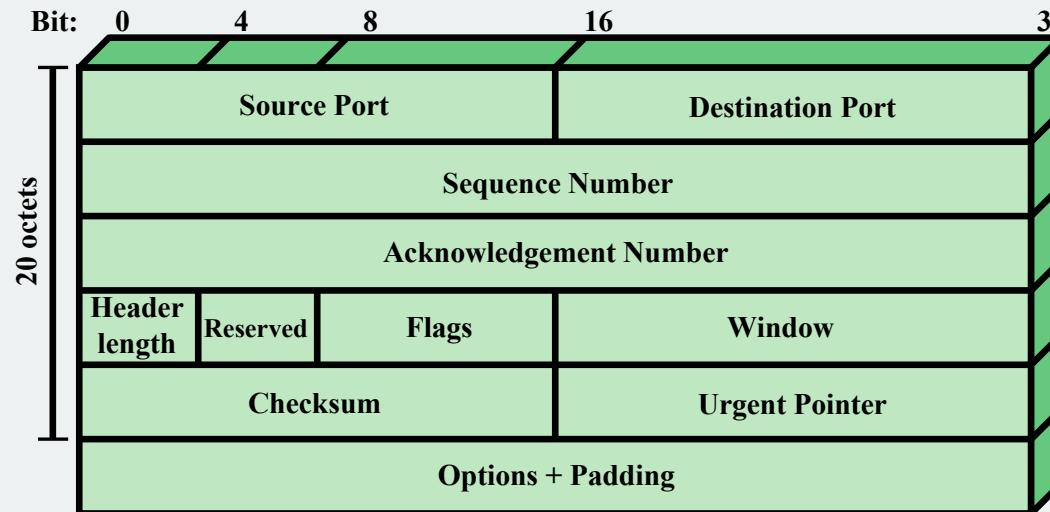


**Figure 2.5 Protocol Data Units (PDUs) in the TCP/IP Architecture**

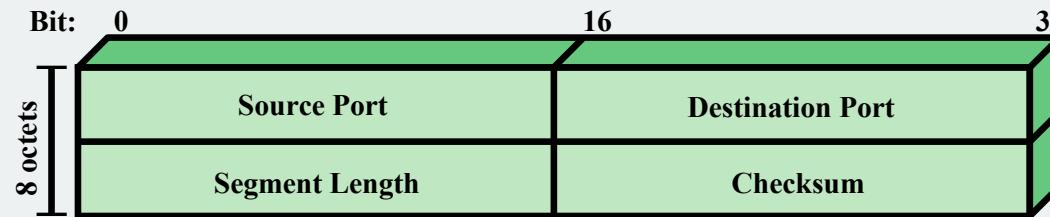
# Transmission Control Protocol (TCP)

- TCP is the transport layer protocol for most applications
- TCP provides a reliable connection for transfer of data between applications
- A TCP segment is the basic protocol unit
- TCP tracks segments between entities for duration of each connection





(a) TCP Header

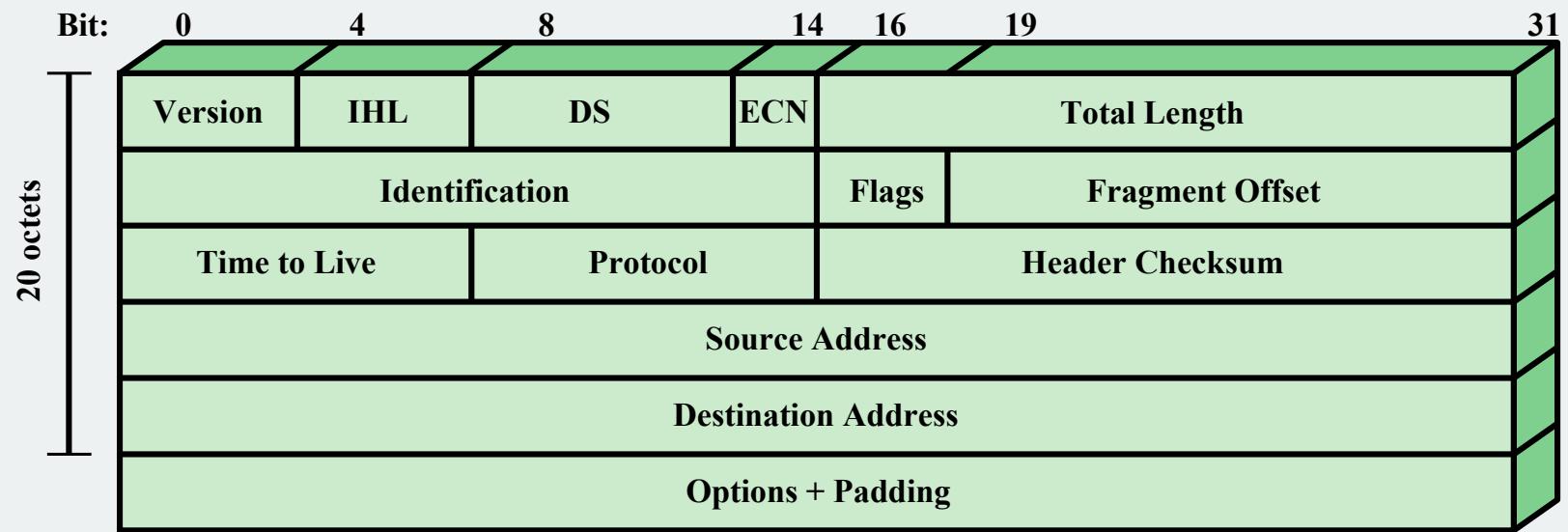


(b) UDP Header

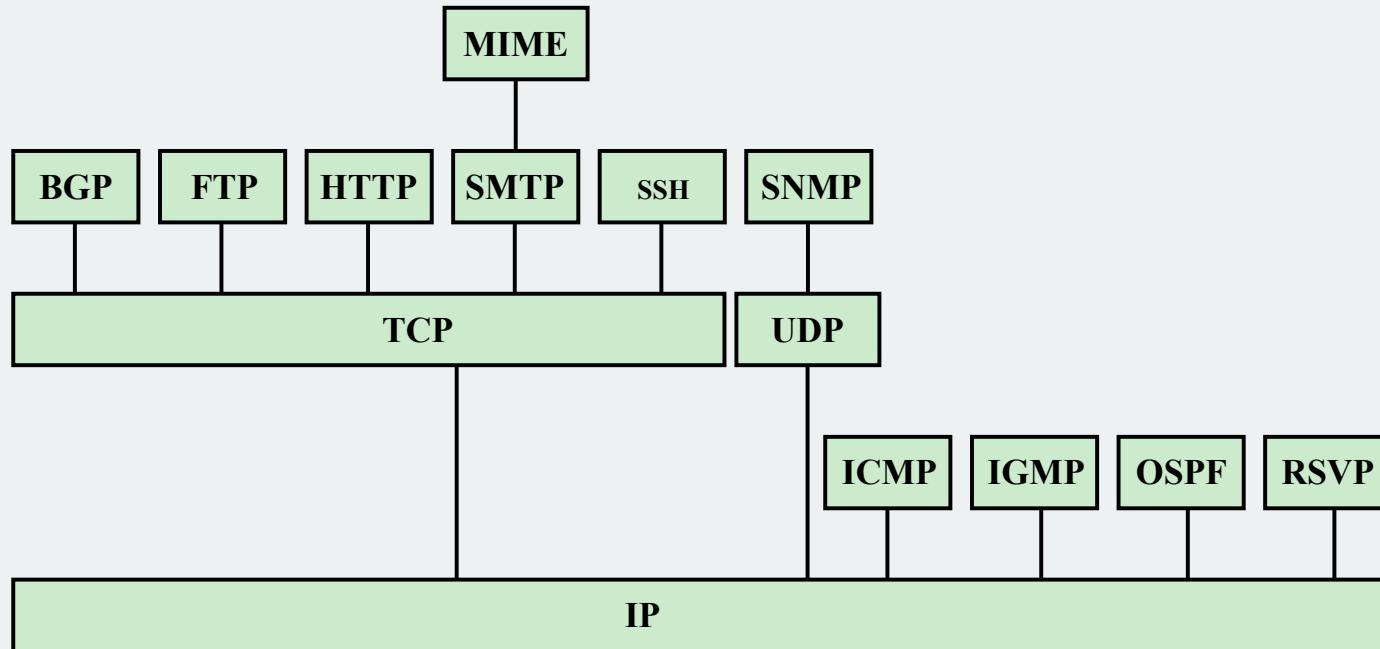
Figure 2.6 TCP and UDP Headers

# User Datagram Protocol (UDP)

- Alternative to TCP
- Does not guarantee delivery, preservation of sequence, or protection against duplication
- Enables a procedure to send messages to other procedures with a minimum of protocol mechanism
- Adds port addressing capability to IP
- Used with Simple Network Management Protocol (SNMP)
- Includes a checksum to verify that no error occurs in the data



(a) IPv4 Header



**BGP** = Border Gateway Protocol  
**FTP** = File Transfer Protocol  
**HTTP** = Hypertext Transfer Protocol  
**ICMP** = Internet Control Message Protocol  
**IGMP** = Internet Group Management Protocol  
**IP** = Internet Protocol  
**MIME** = Multipurpose Internet Mail Extension

**OSPF** = Open Shortest Path First  
**RSVP** = Resource ReSerVation Protocol  
**SMTP** = Simple Mail Transfer Protocol  
**SNMP** = Simple Network Management Protocol  
**SSH** = Secure Shell  
**TCP** = Transmission Control Protocol  
**UDP** = User Datagram Protocol

**Figure 2.8 Some Protocols in the TCP/IP Protocol Suite**

# Service Primitives and Parameters

- Services between adjacent layers
- Expressed as:
  - **Primitives**
    - Specify the function to be performed
  - **Parameters**
    - Used to pass data and control information

# Traditional Internet-Based Applications

- Three common applications that have been standardized to operate on top of TCP are:

## Simple Mail Transfer Protocol (SMTP)

- Provides a mechanism for transferring messages among separate hosts

## File Transfer Protocol (FTP)

- Used to send files from one system to another under user command
- Both text and binary files are accommodated

## Secure Shell (SSH)

- Provides a secure remote logon capability

# The Socket

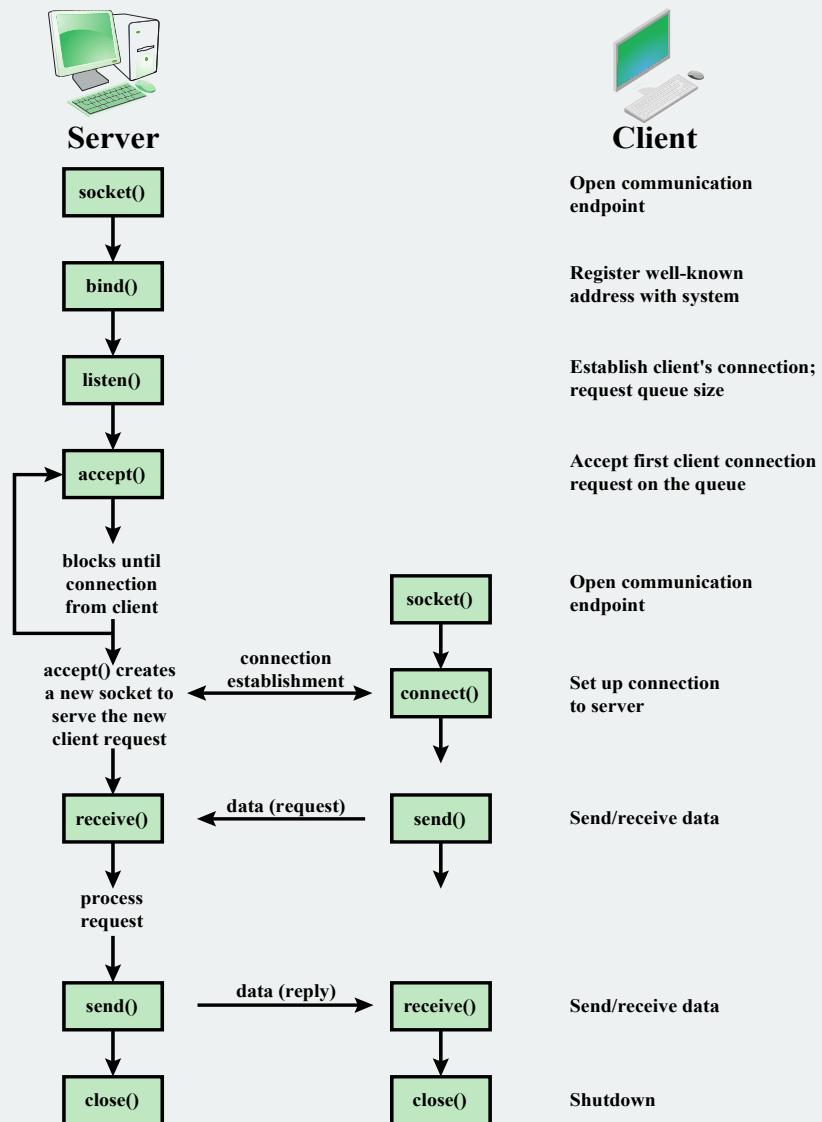
- Formed by the concatenation of a port value and an IP address
  - Unique throughout the Internet
- Used to define an API
  - Generic communication interface for writing programs that use TCP or UDP
- Stream sockets
  - All blocks of data sent between a pair of sockets are guaranteed for delivery and arrive in the order that they were sent
- Datagram sockets
  - Delivery is not guaranteed, nor is order necessarily preserved
- Raw sockets
  - Allow direct access to lower-layer protocols

# Table 2.4

## Core Socket Functions

Format	Function	Parameters	
socket( )	Initialize a socket	<b>domain</b>	Protocol family of the socket to be created (AF_UNIX, AF_INET, AF_INET6)
		<b>type</b>	Type of socket to be opened (stream, datagram, raw)
		<b>protocol</b>	Protocol to be used on socket (UDP, TCP, ICMP)
bind( )	Bind a socket to a port address	<b>sockfd</b>	Socket to be bound to the port address
		<b>localaddress</b>	Socket address to which the socket is bound
		<b>addresslength</b>	Length of the socket address structure
listen( )	Listen on a socket for inbound connections	<b>sockfd</b>	Socket on which the application is to listen
		<b>queuesize</b>	Number of inbound requests that can be queued at any time
accept( )	Accept an inbound connection	<b>sockfd</b>	Socket on which the connection is to be accepted
		<b>remoteaddress</b>	Remote socket address from which the connection was initiated
		<b>addresslength</b>	Length of the socket address structure
connect( )	Connect outbound to a server	<b>sockfd</b>	Socket on which the connection is to be opened
		<b>remoteaddress</b>	Remote socket address to which the connection is to be opened
		<b>addresslength</b>	Length of the socket address structure
send( ) recv( )  read( ) write( )	Send and receive data on a stream socket (either send/recv or read/write can be used)	<b>sockfd</b>	Socket across which the data will be sent or read
		<b>data</b>	Data to be sent, or buffer into which the read data will be placed
		<b>datalength</b>	Length of the data to be written, or amount of data to be read
sendto( ) recvfrom( )	Send and receive data on a datagram socket	<b>sockfd</b>	Socket across which the data will be sent or read
		<b>data</b>	Data to be sent, or buffer into which the read data will be placed
		<b>datalength</b>	Length of the data to be written, or amount of data to be read
close( )	Close a socket	<b>sockfd</b>	Socket which is to be closed

(Table can be found  
on page 78 in  
textbook)



**Figure 2.12** Socket System Calls for Connection-Oriented Protocol

```

1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <sys/socket.h>
4 #include <netinet/in.h>

5 void error(char *msg)
6 {
7     perror(msg);
8     exit(1);
9 }

10 int main(int argc, char *argv[])
11 {
12     int sockfd, newsockfd, portno, clilen;
13     char buffer[256];
14     struct sockaddr_in serv_addr, cli_addr;
15     int n;
16     if (argc < 2) {
17         fprintf(stderr,"ERROR, no port provided\n");
18         exit(1);
19     }
20     sockfd = socket(AF_INET, SOCK_STREAM, 0);
21     if (sockfd < 0)
22         error("ERROR opening socket");
23     bzero((char *) &serv_addr, sizeof(serv_addr));
24     portno = atoi(argv[1]);
25     serv_addr.sin_family = AF_INET;
26     serv_addr.sin_port = htons(portno);
27     serv_addr.sin_addr.s_addr = INADDR_ANY;
28     if (bind(sockfd, (struct sockaddr *) &serv_addr,
29             sizeof(serv_addr)) < 0)
30         error("ERROR on binding");
31     listen(sockfd,5);
32     clilen = sizeof(cli_addr);
33     newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
34     if (newsockfd < 0)
35         error("ERROR on accept");
36     bzero(buffer,256);
37     n = read(newsockfd,buffer,255);
38     if (n < 0) error("ERROR reading from socket");
39     printf("Here is the message: %s\n",buffer);
40     n = write(newsockfd,"I got your message",18);
41     if (n < 0) error("ERROR writing to socket");
42     return 0;
43 }

```

**Figure 2.13 Sockets Server**

(Figure 2.13 can be found on page 81 in textbook)

```

1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <sys/socket.h>
4 #include <netinet/in.h>
5 #include <netdb.h>

6 void error(char *msg)
7 {
8     perror(msg);
9     exit(0);
10 }

11 int main(int argc, char *argv[])
12 {
13     int sockfd, portno, n;
14     struct sockaddr_in serv_addr;
15     struct hostent *server;
16     char buffer[256];
17     if (argc < 3) {
18         fprintf(stderr,"usage %s hostname port\n", argv[0]);
19         exit(0);
20     }
21     portno = atoi(argv[2]);
22     sockfd = socket(AF_INET, SOCK_STREAM, 0);
23     if (sockfd < 0)
24         error("ERROR opening socket");
25     server = gethostbyname(argv[1]);
26     if (server == NULL) {
27         fprintf(stderr,"ERROR, no such host\n");
28         exit(0);
29     }
30     bzero((char *) &serv_addr, sizeof(serv_addr));
31     serv_addr.sin_family = AF_INET;
32     bcopy((char *)server->h_addr,
33           (char *)&serv_addr.sin_addr.s_addr,
34           server->h_length);
35     serv_addr.sin_port = htons(portno);
36     if (connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr)) < 0)
37         error("ERROR connecting");
38     printf("Please enter the message: ");
39     bzero(buffer,256);
40     fgets(buffer,255,stdin);
41     n = write(sockfd,buffer,strlen(buffer));
42     if (n < 0)
43         error("ERROR writing to socket");
44     bzero(buffer,256);
45     n = read(sockfd,buffer,255);
46     if (n < 0)
47         error("ERROR reading from socket");
48     printf("%s\n",buffer);
49     return 0;
50 }

```

**Figure 2.14 Sockets Client**

(Figure 2.14 can be found on page 82 in textbook)



# Summary

- The need for a protocol architecture
- Simple protocol architecture
- TCP/IP protocol architecture
  - TCP/IP layers
  - Operation of TCP and IP
  - TCP and UDP
  - IP and IPv6
  - Protocol interfaces
- Standardization within a protocol architecture
  - Standards and protocol layers
  - Service primitives and parameters
- Traditional internet-based applications
- Multimedia
  - Media types
  - Multimedia applications
  - Multimedia technologies
- Sockets programming
  - The socket
  - Sockets interface calls