Data and Computer Communications

Tenth Edition

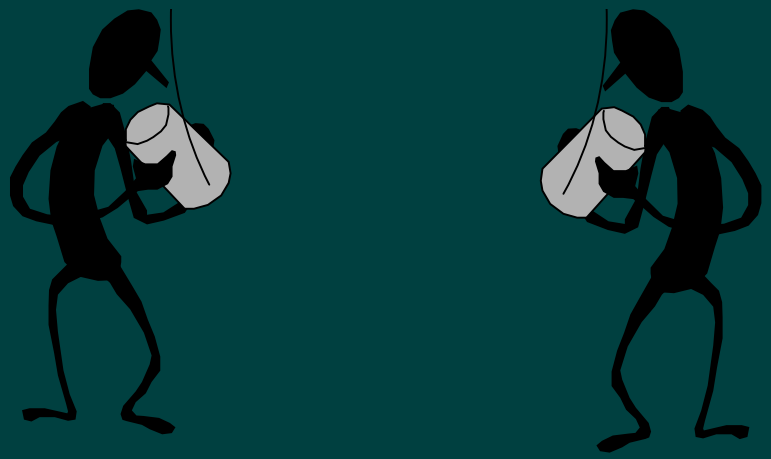
by William Stallings

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

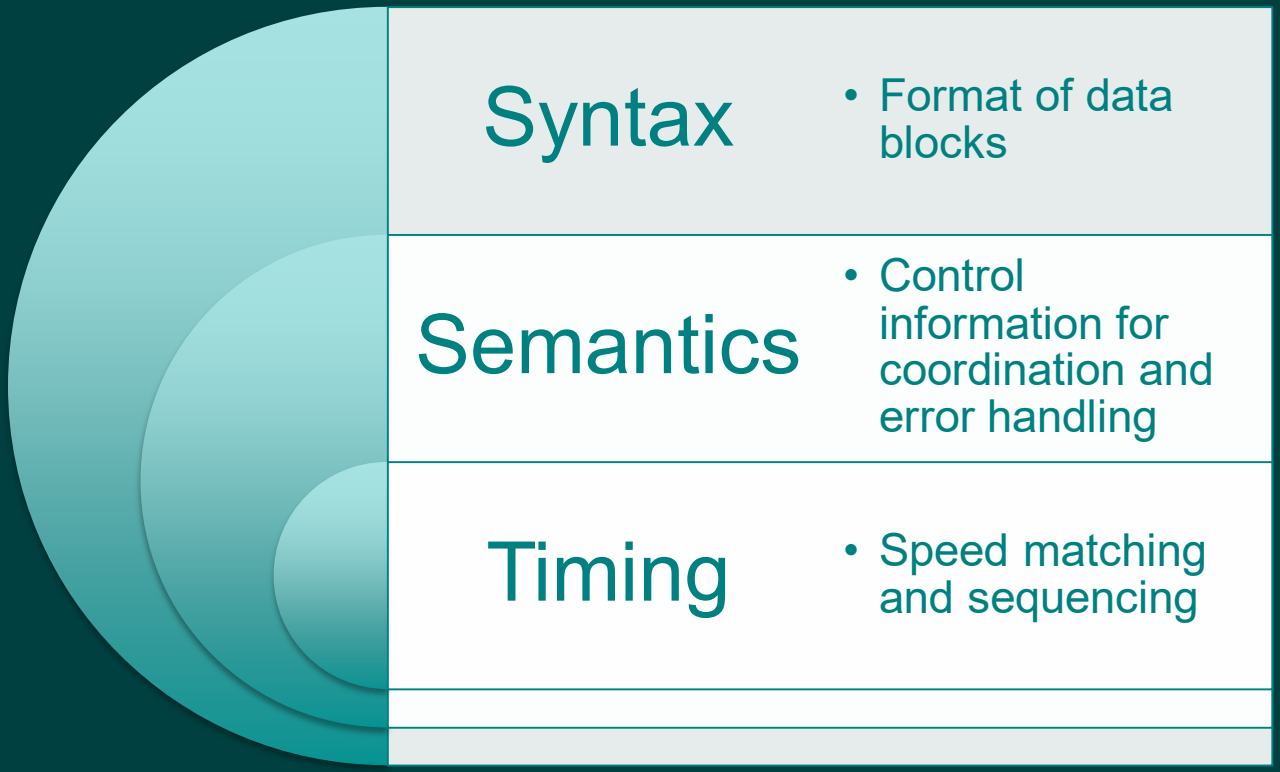
Functions of Protocol Architecture

- 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:



A Simple Protocol Architecture

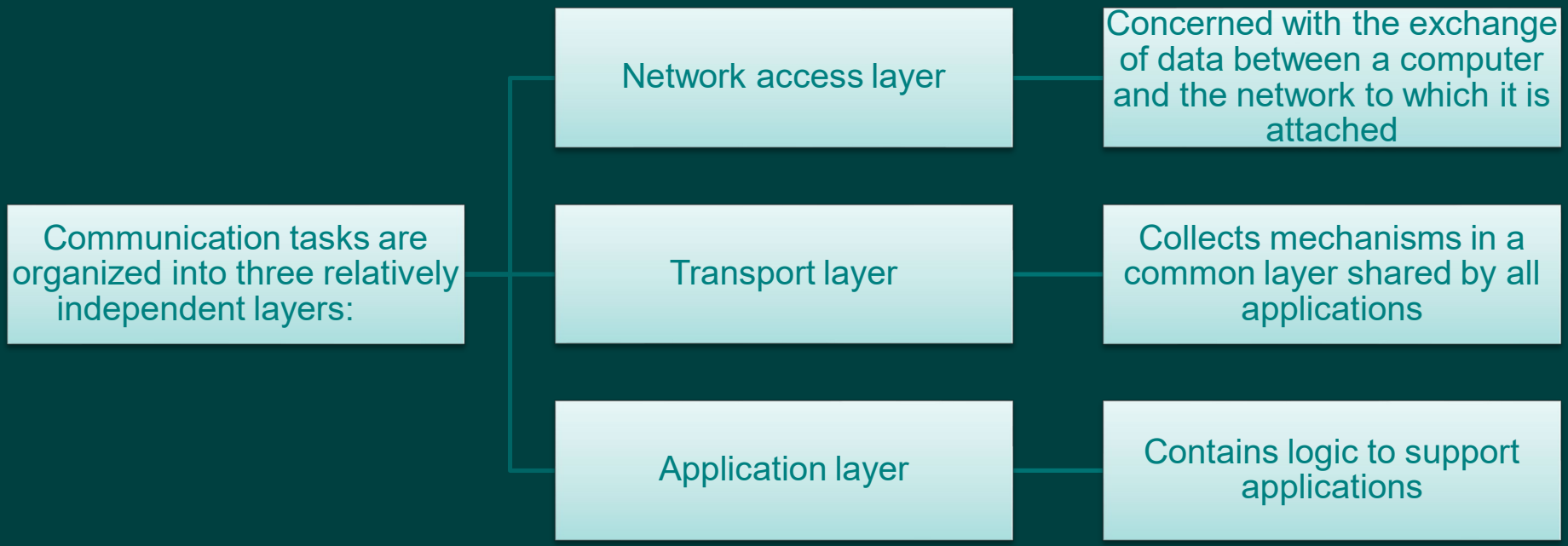
Agents involved:

- Applications
- Computers
- Networks

Examples of applications include file transfer and electronic mail

These execute on computers that support multiple simultaneous applications

Communication Layers



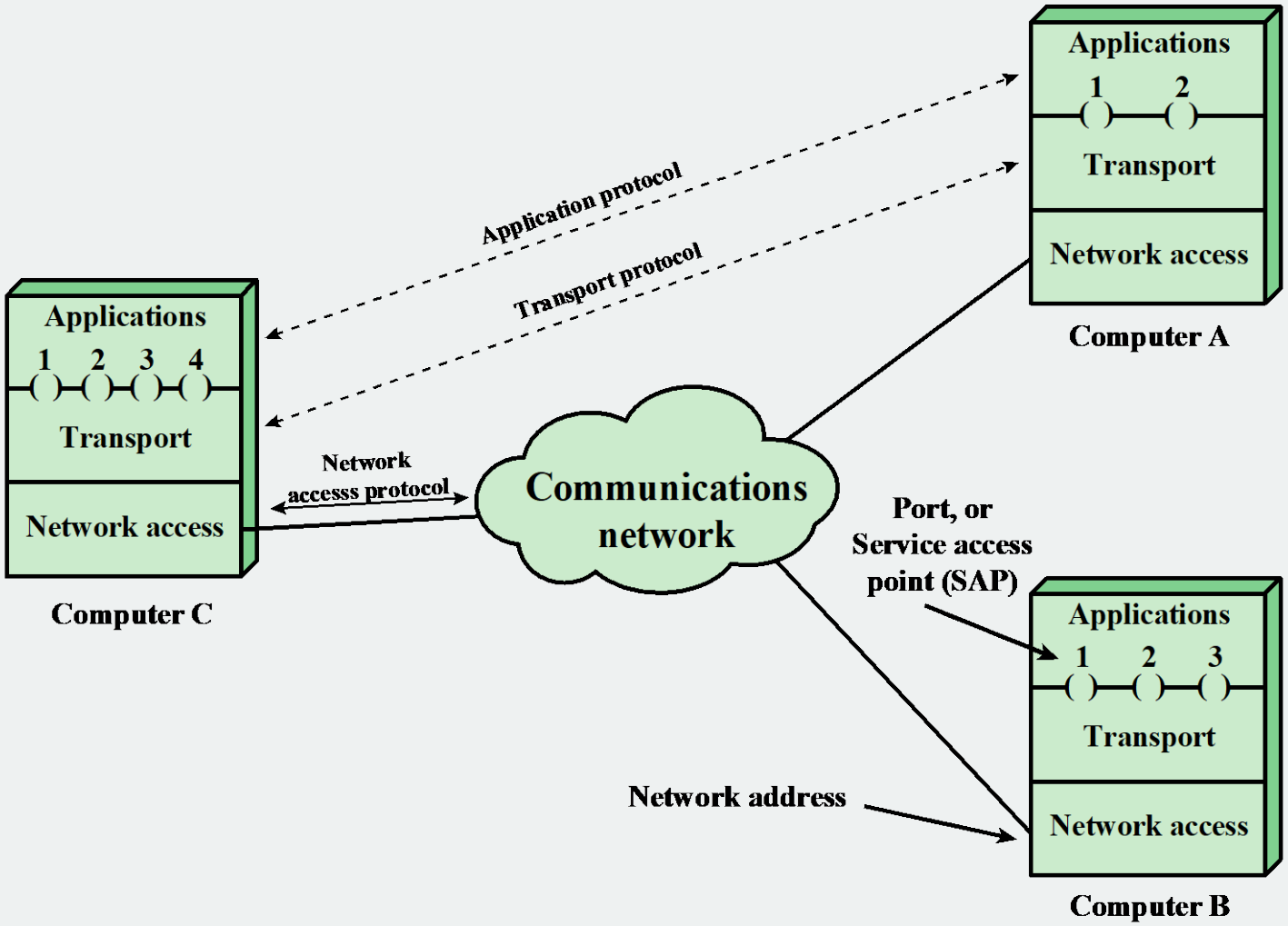


Figure 2.1 Protocol Architectures and Networks

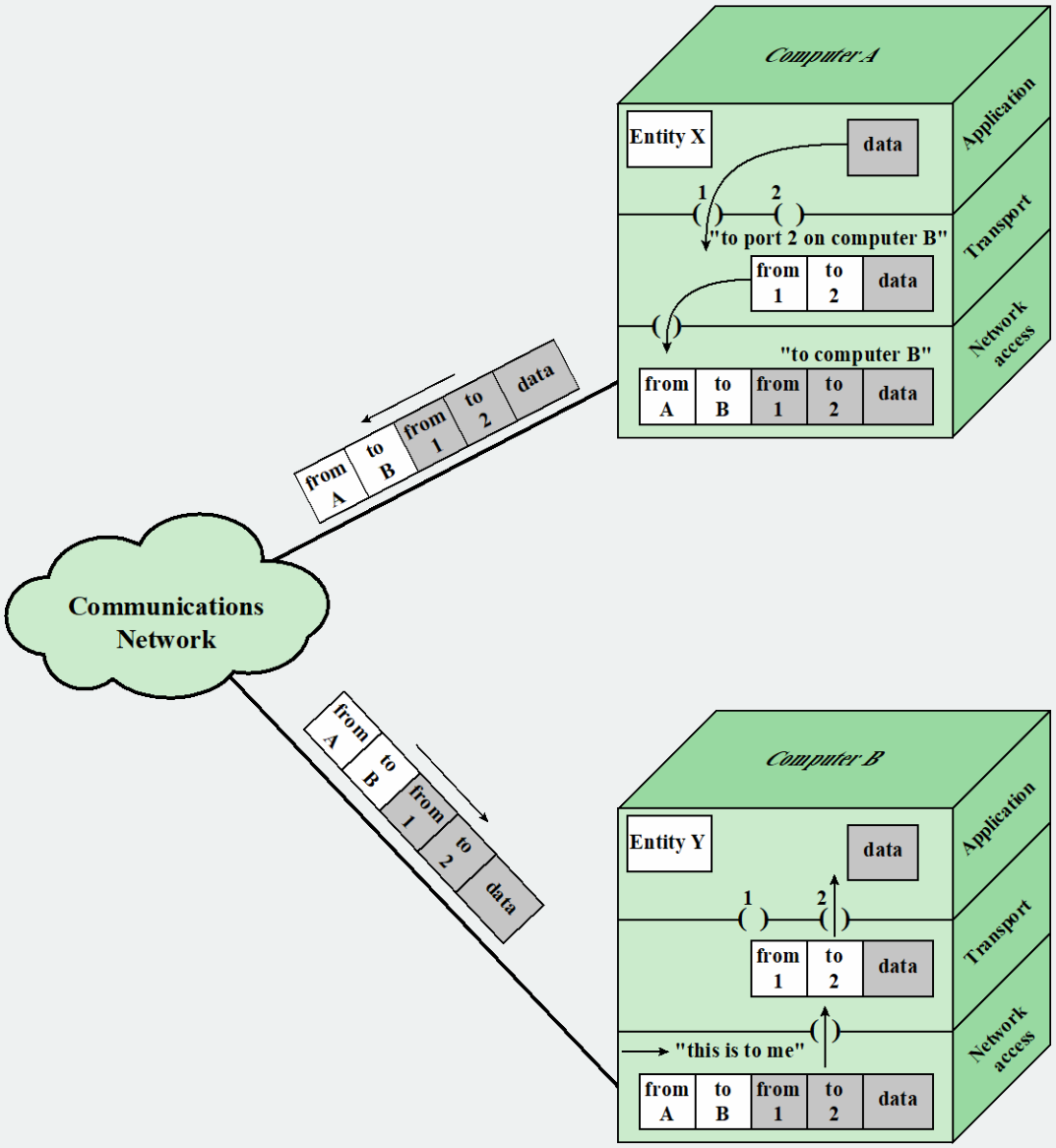
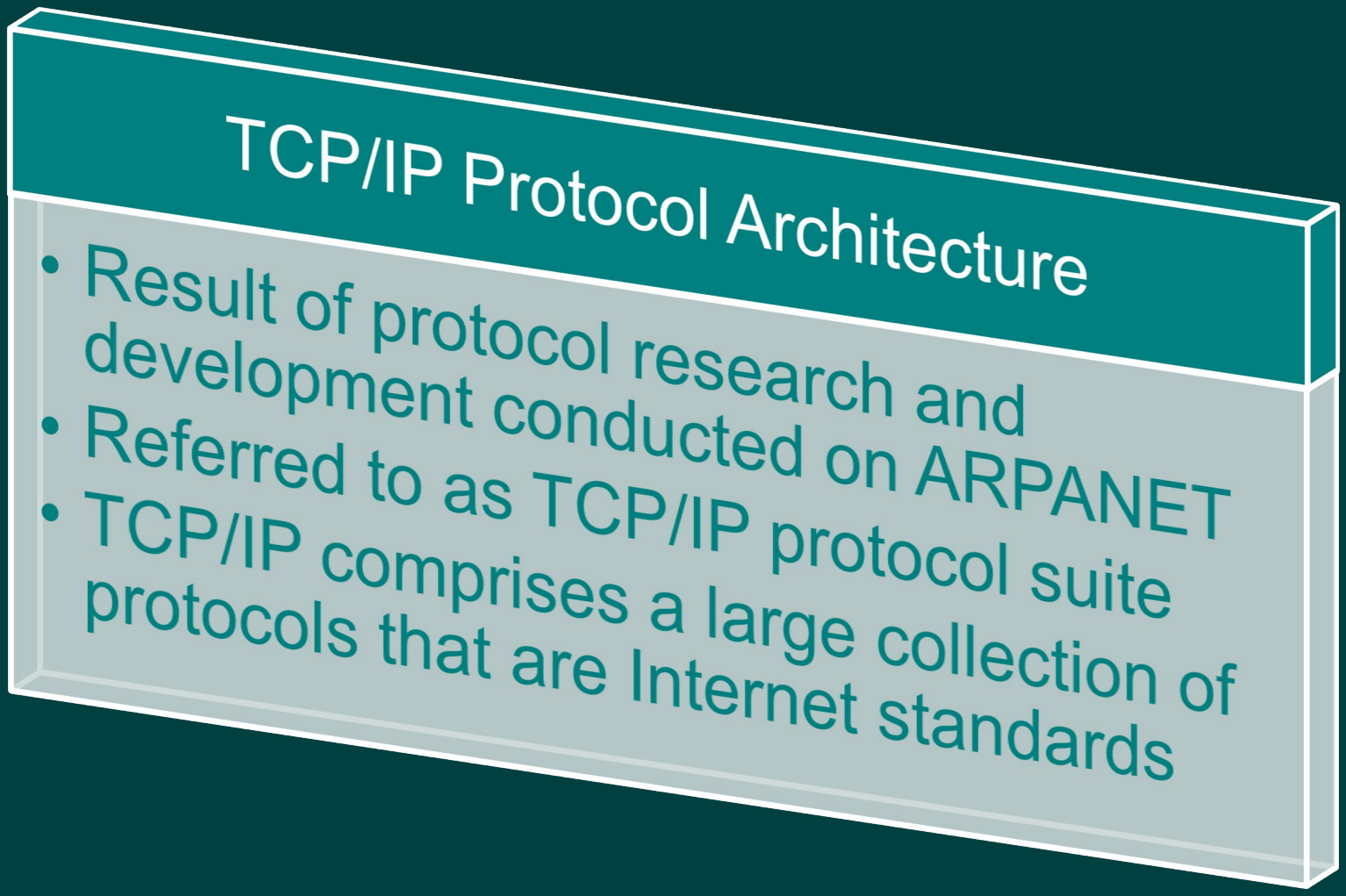


Figure 2.2 Protocols in a Simplified Architecture

TCP/IP Protocol Architecture



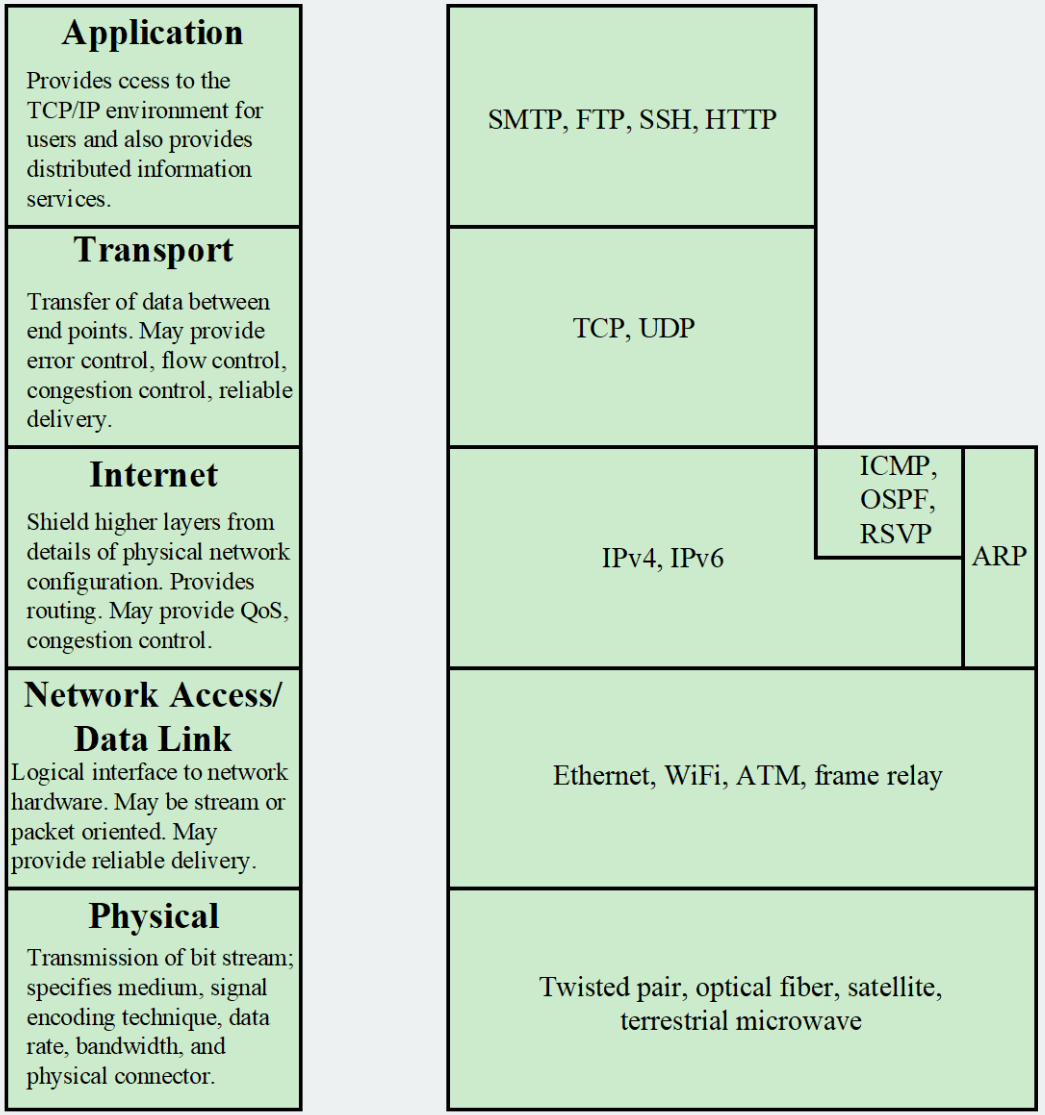


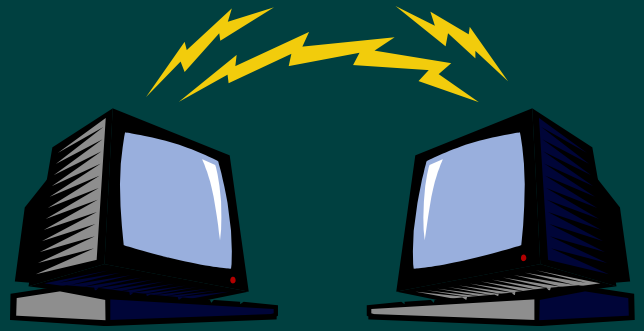
Figure 2.3 The TCP/IP Layers and Example Protocols

Physical Layer

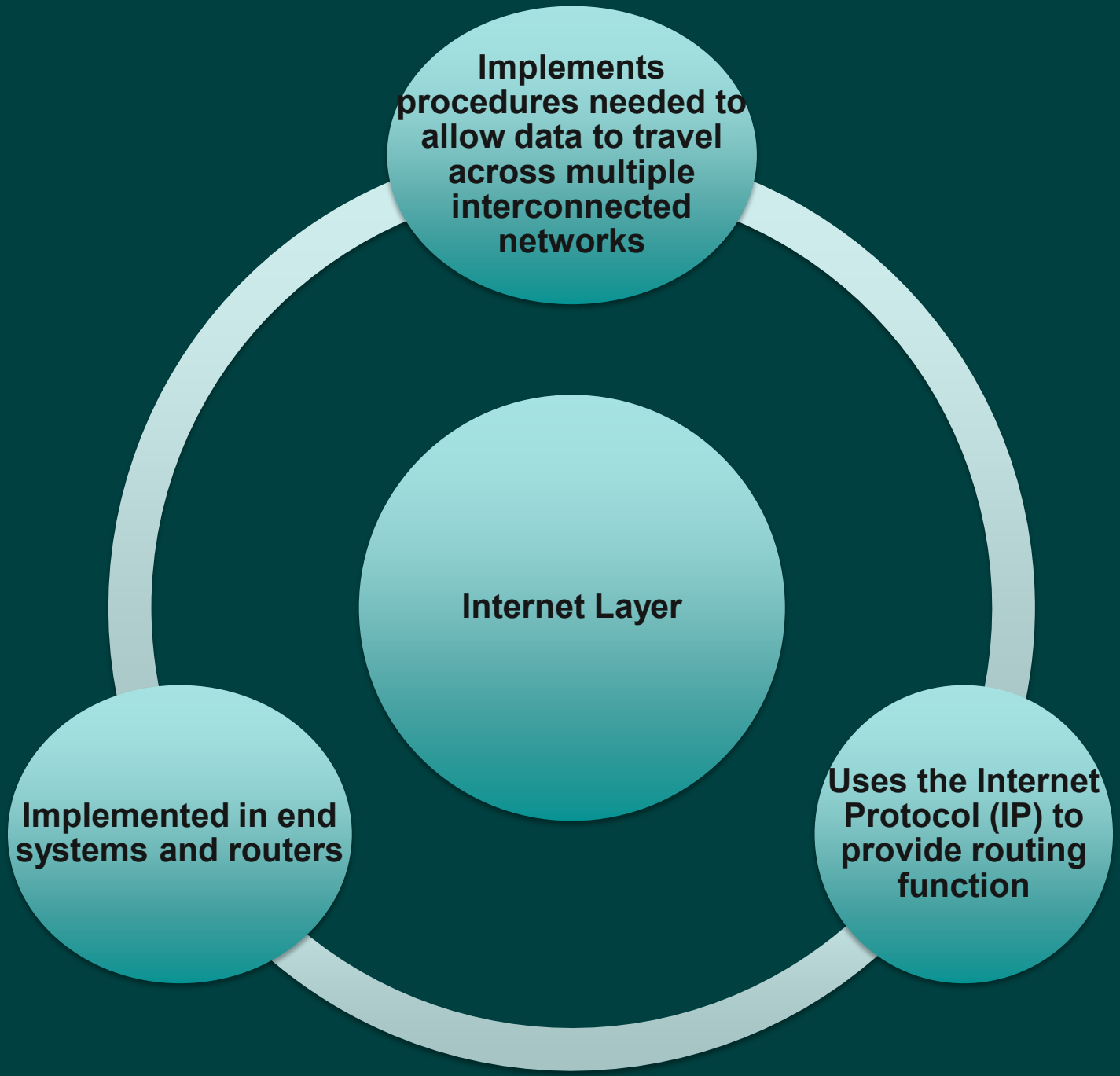
- Covers the physical interface between computer and network
- Concerned with issues like:
  - Characteristics of transmission medium
  - Nature of the signals
  - Data rates

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

Transmission Control Protocol

TCP

- Most commonly used protocol to provide this functionality

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

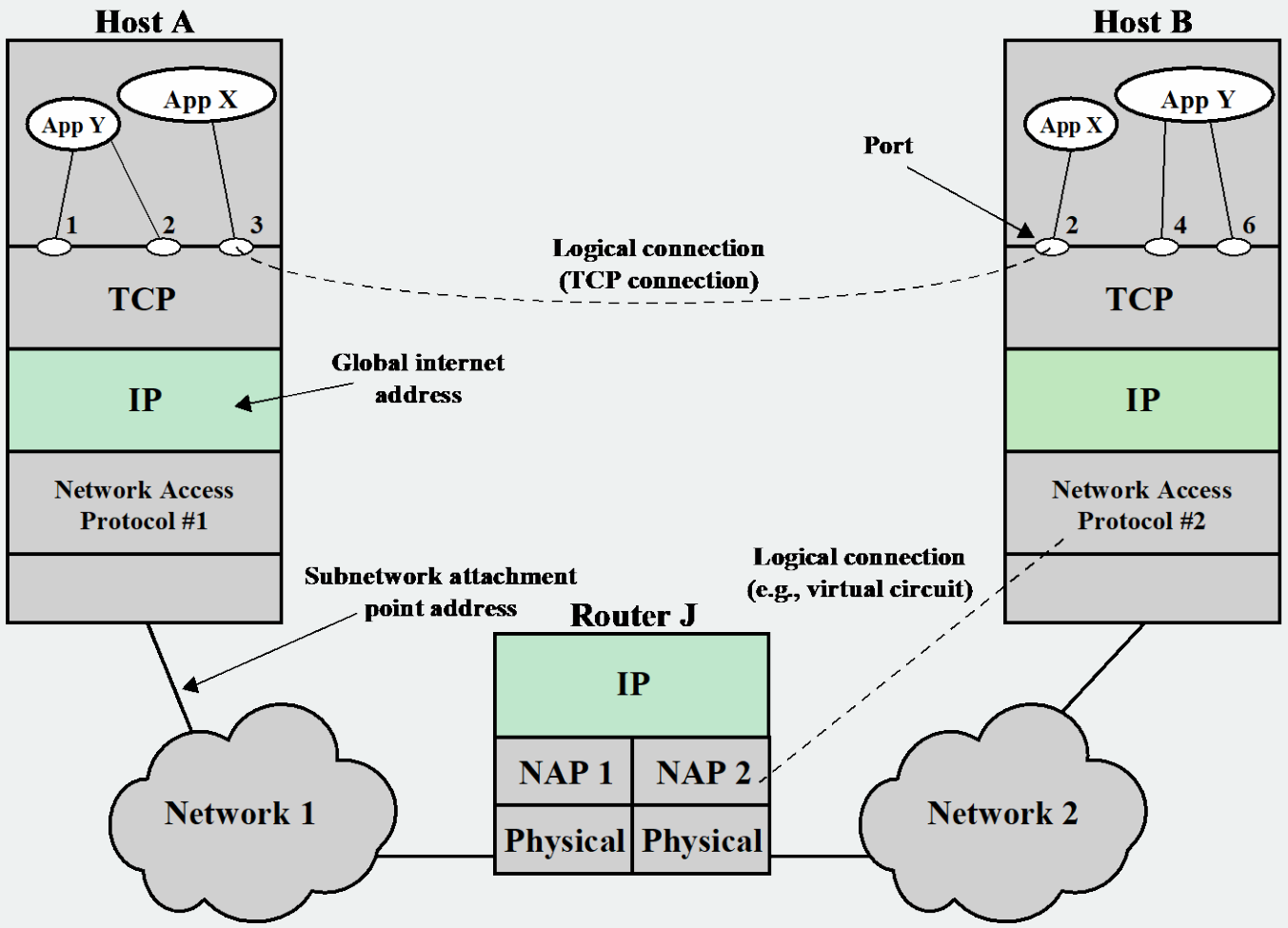
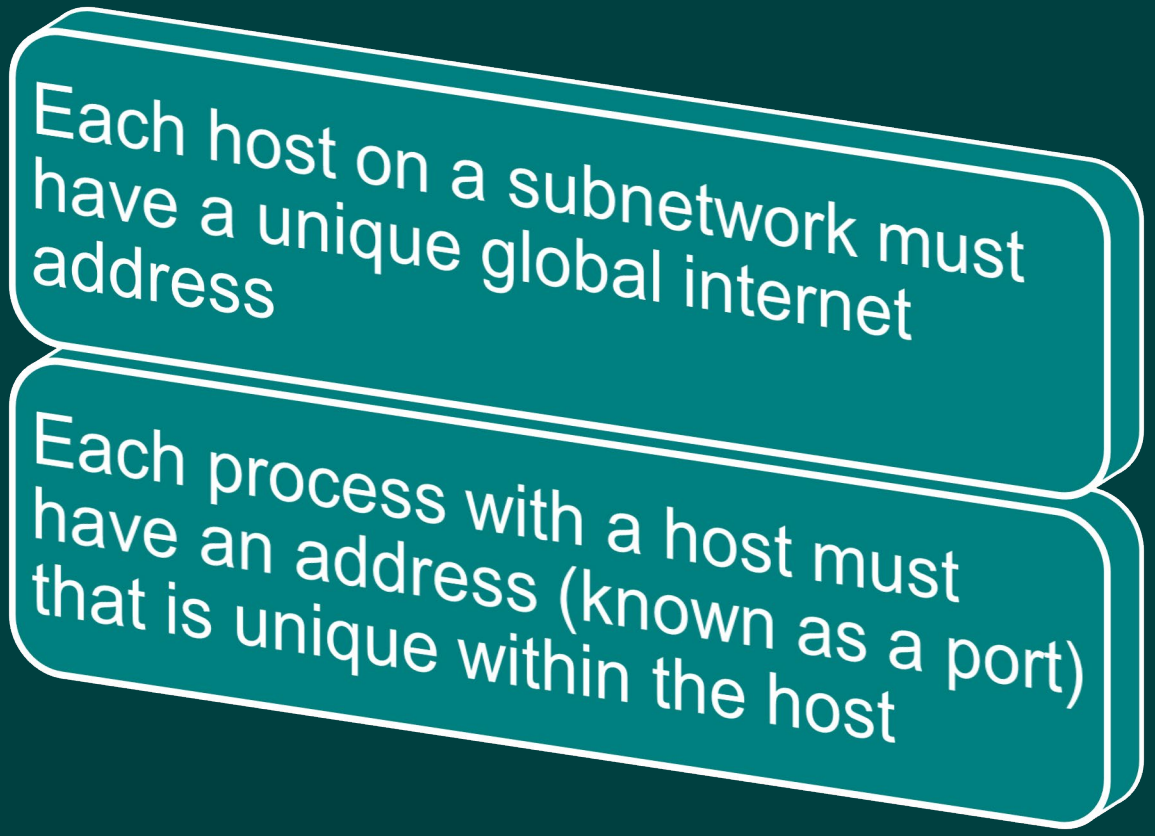


Figure 2.4 TCP/IP Concepts

TCP/IP Address Requirements

Two levels of addressing are needed:



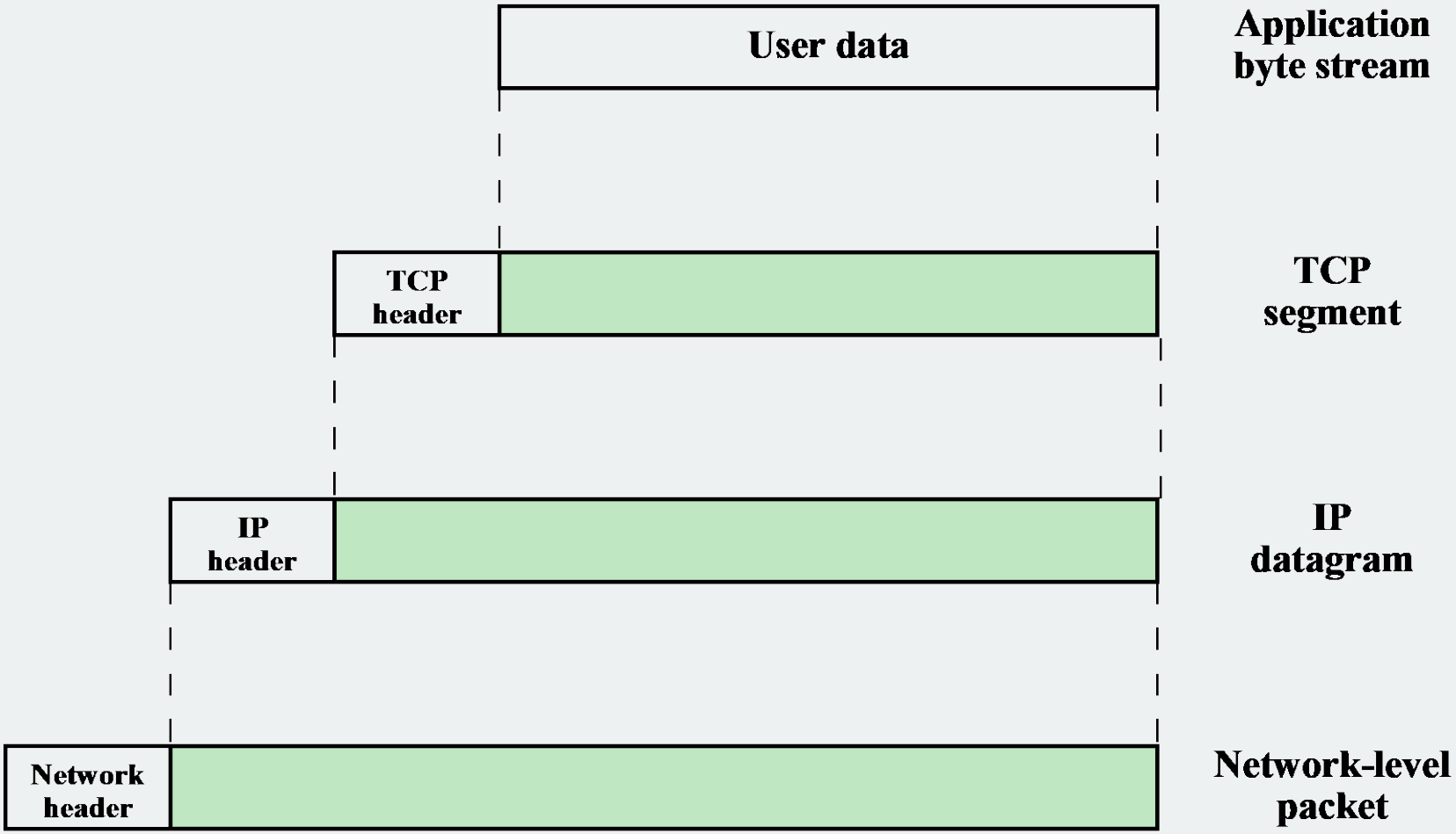
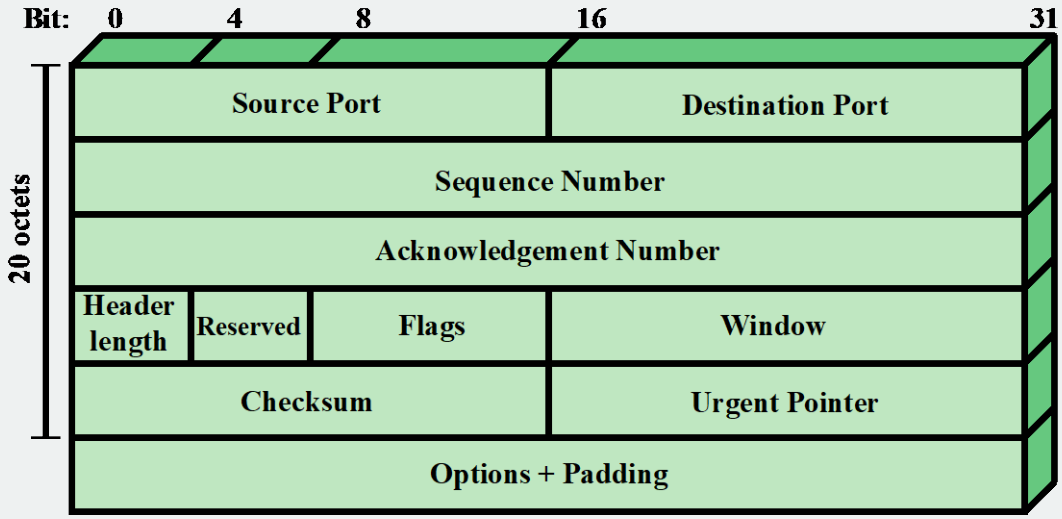


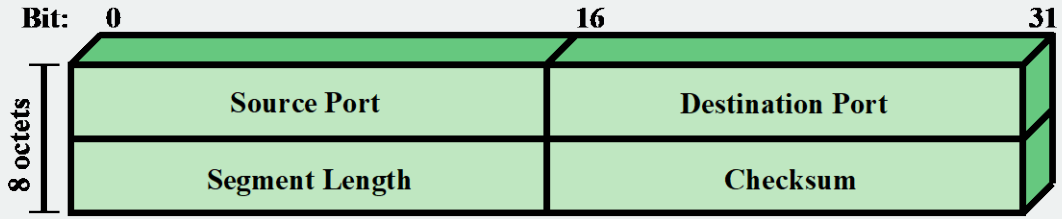
Figure 2.5 Protocol Data Units (PDUs) in the TCP/IP Architecture

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

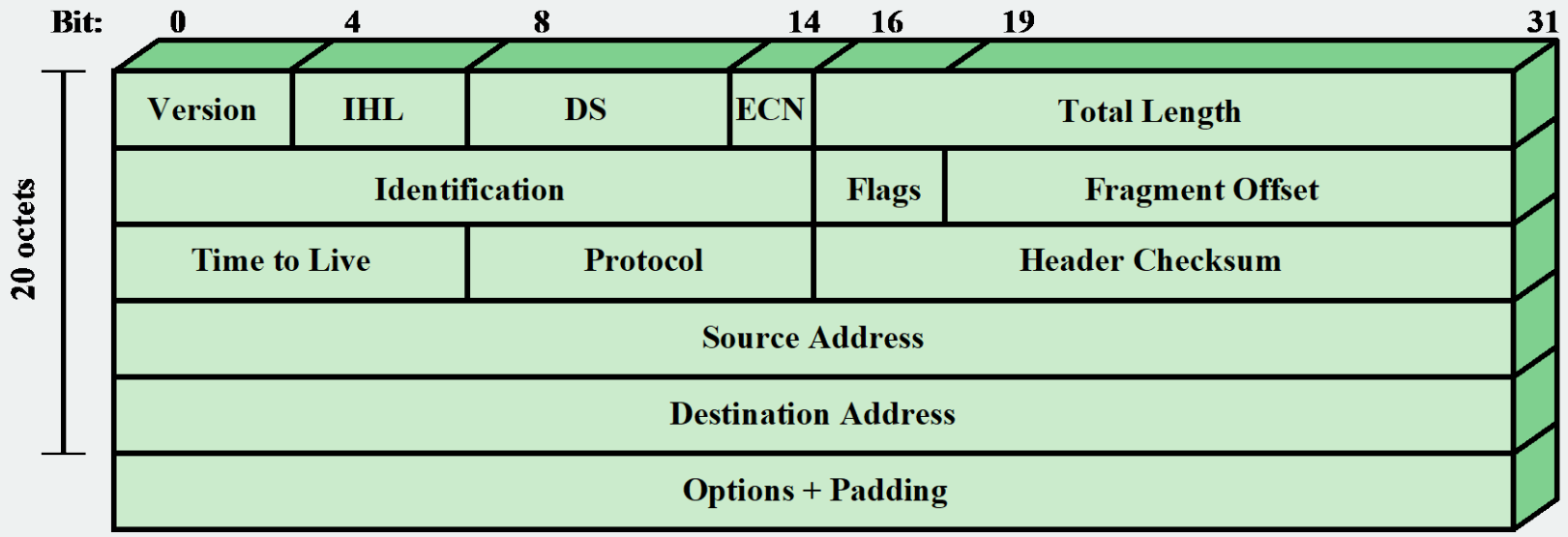


(b) UDP Header

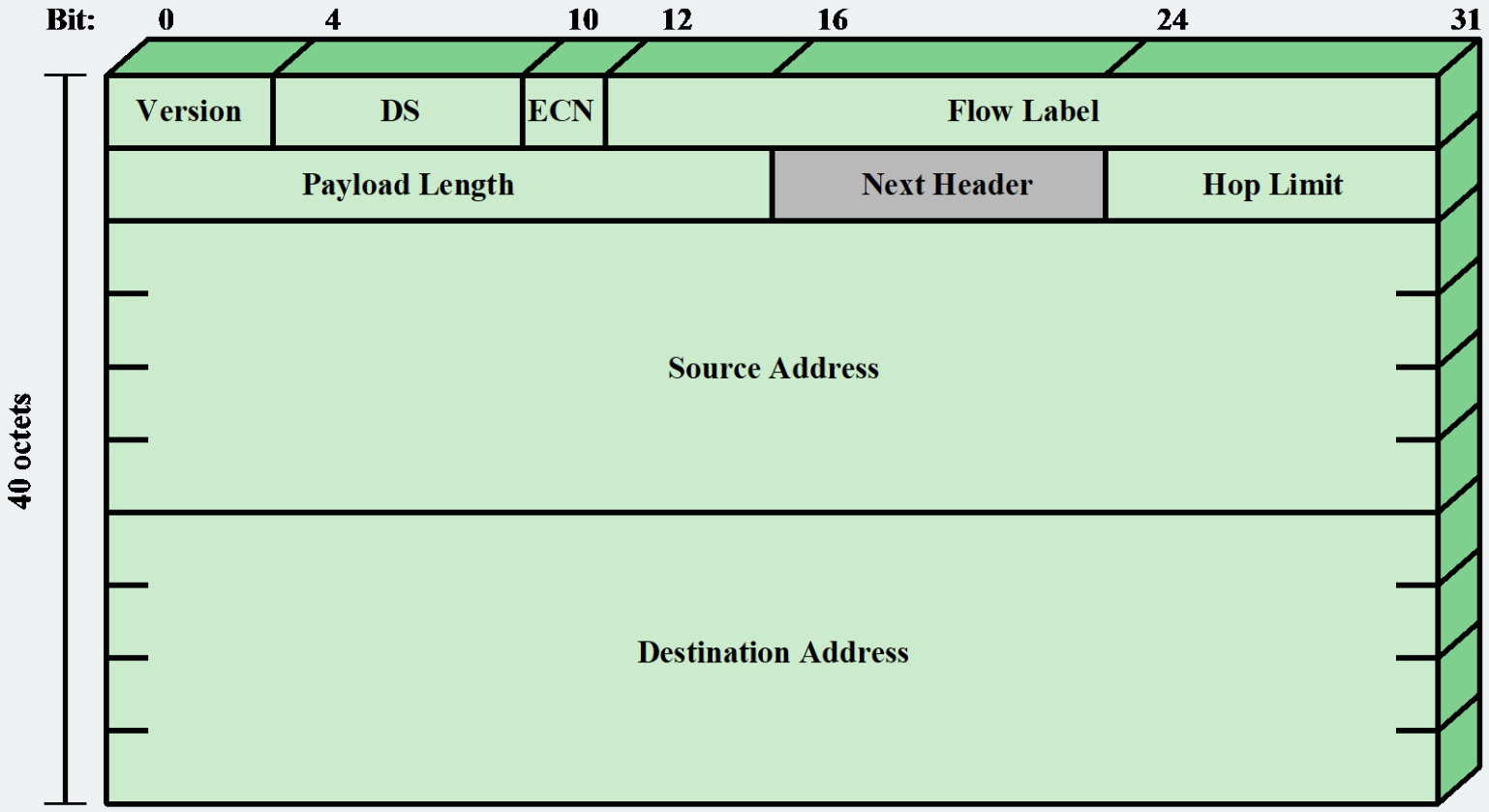
Figure 2.6 TCP and UDP Headers

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

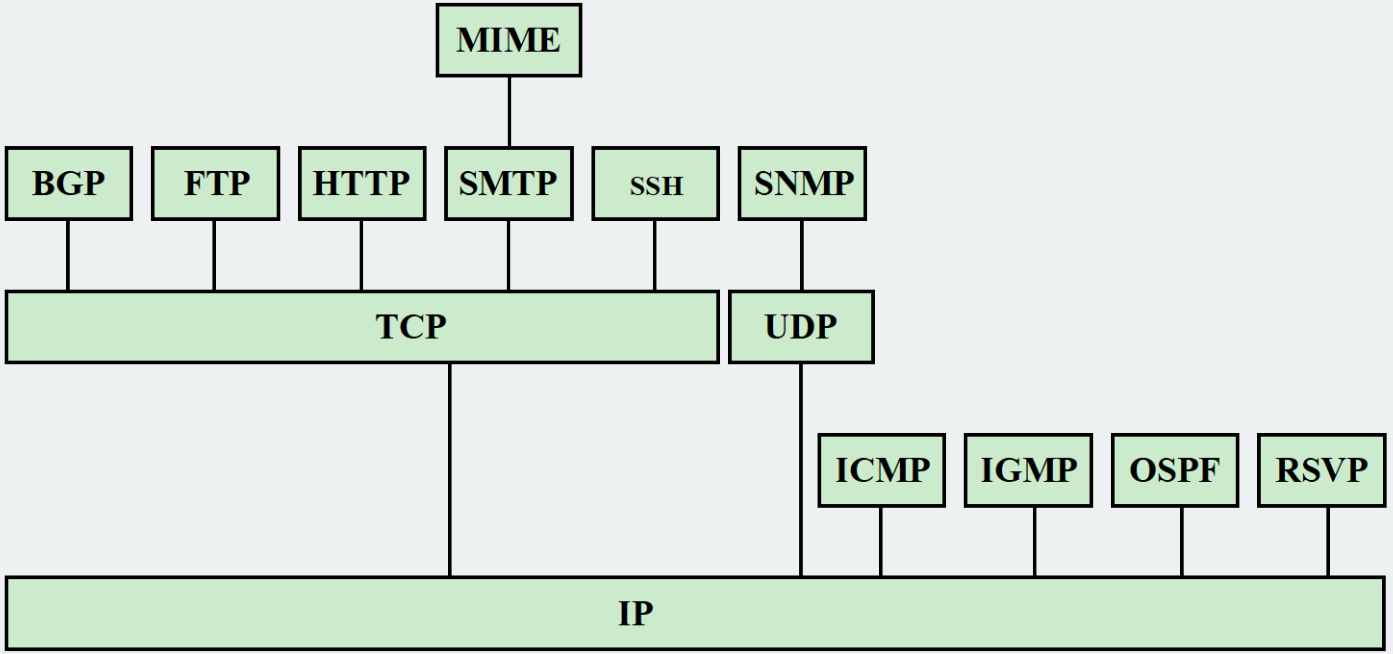


(b) IPv6 Header

DS = Differentiated services field
ECN = Explicit congestion notification field

Note: The 8-bit DS/ECN fields were formerly known as the Type of Service field in the IPv4 header and the Traffic Class field in the IPv6 header.

Figure 2.7 IP Headers



BGP	=	Border Gateway Protocol	OSPF	=	Open Shortest Path First
FTP	=	File Transfer Protocol	RSVP	=	Resource ReSerVation Protocol
HTTP	=	Hypertext Transfer Protocol	SMTP	=	Simple Mail Transfer Protocol
ICMP	=	Internet Control Message Protocol	SNMP	=	Simple Network Management Protocol
IGMP	=	Internet Group Management Protocol	SSH	=	Secure Shell
IP	=	Internet Protocol	TCP	=	Transmission Control Protocol
MIME	=	Multipurpose Internet Mail Extension	UDP	=	User Datagram Protocol

Figure 2.8 Some Protocols in the TCP/IP Protocol Suite

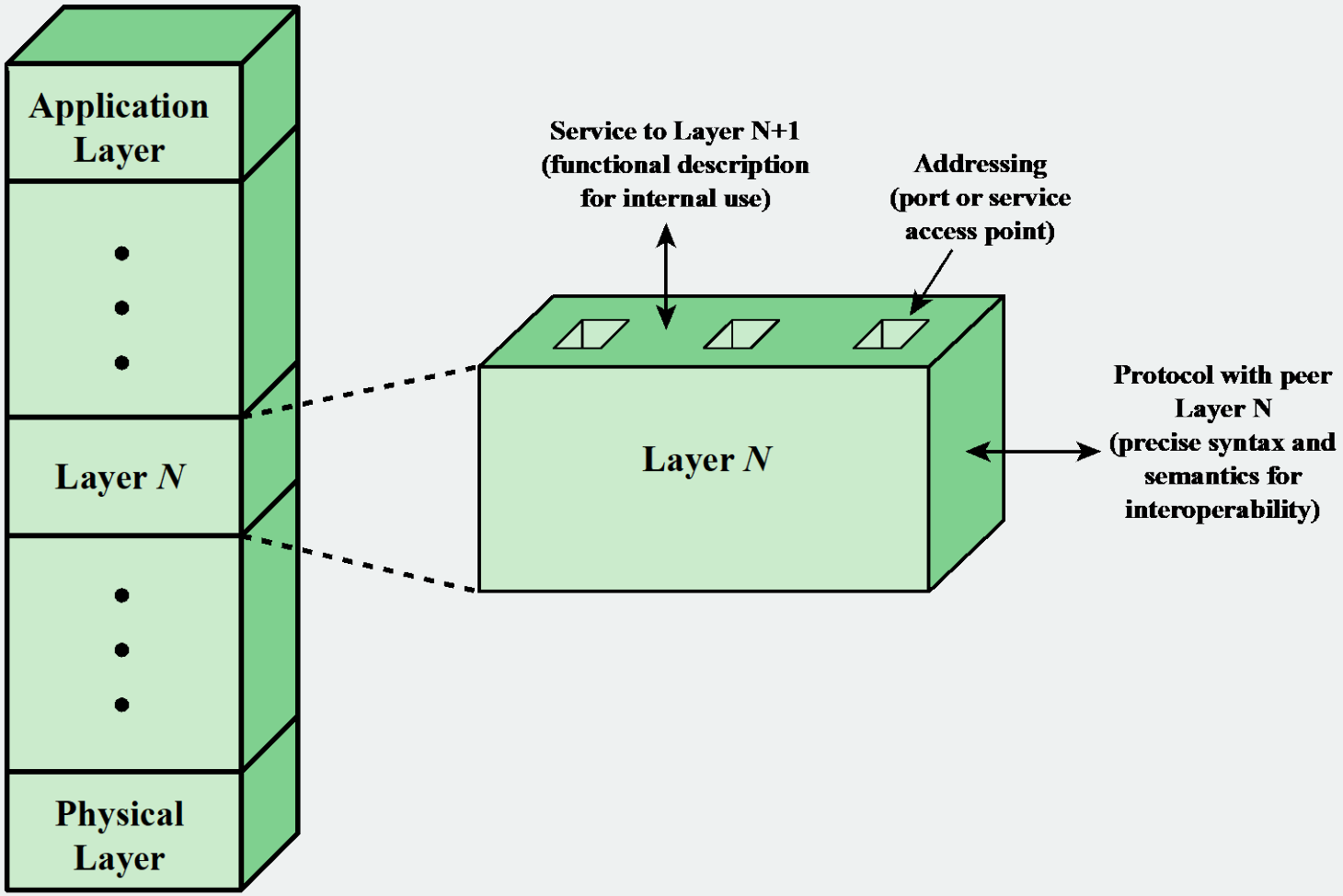


Figure 2.9 A Protocol Architecture as a Framework for Standardization

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

Table 2.1
Service Primitive Types

REQUEST	A primitive issued by a service user to invoke some service and to pass the parameters needed to specify fully the requested service
INDICATION	A primitive issued by a service provider either to 1. indicate that a procedure has been invoked by the peer service user on the connection and to provide the associated parameters, or 2. notify the service user of a provider-initiated action
RESPONSE	A primitive issued by a service user to acknowledge or complete some procedure previously invoked by an indication to that user
CONFIRM	A primitive issued by a service provider to acknowledge or complete some procedure previously invoked by a request by the service user

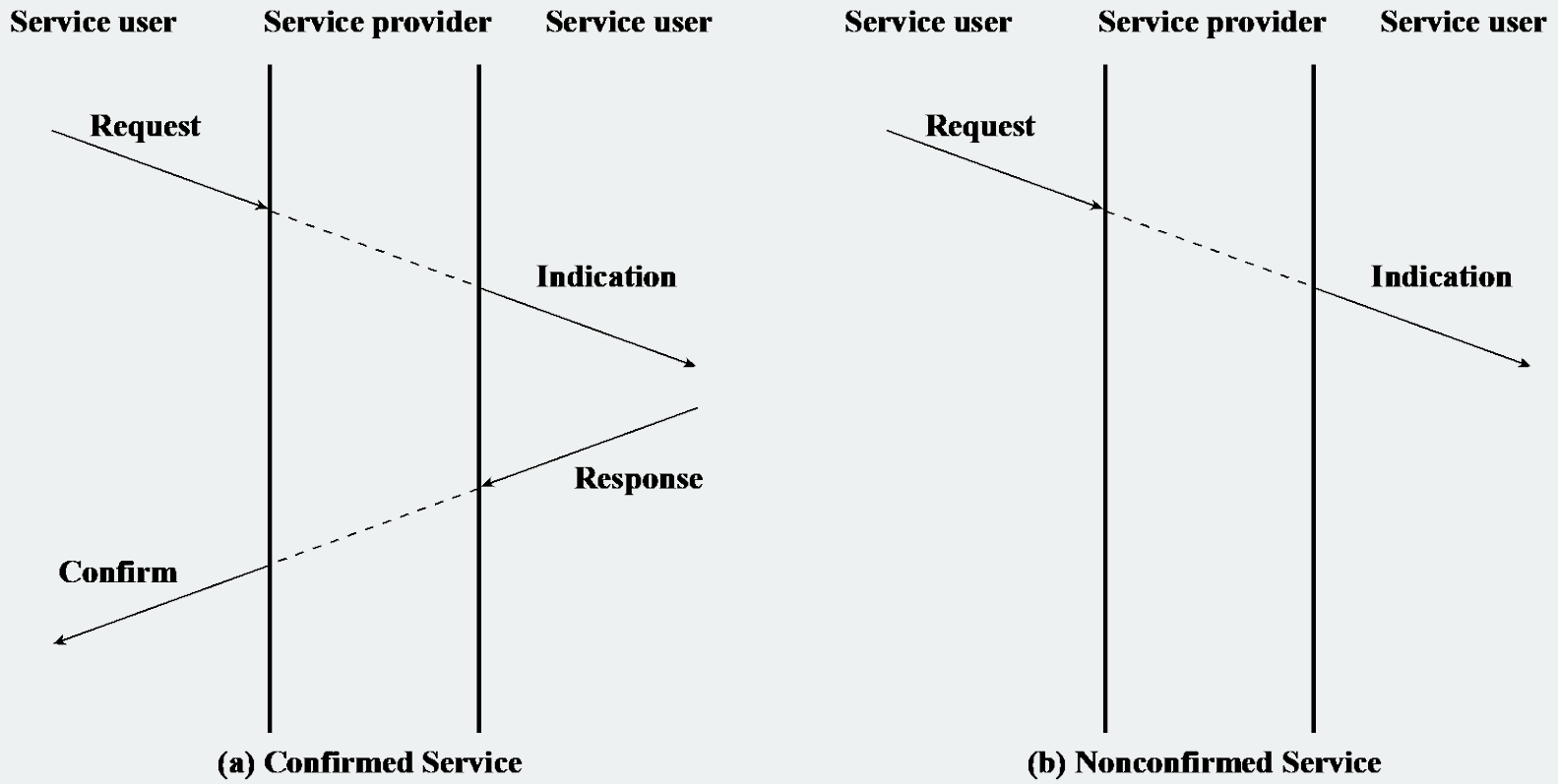


Figure 2.10 Time Sequence Diagrams for Service Primitives

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

Table 2.2
Multimedia Terminology

Media

Refers to the form of information and includes text, still images, audio, and video.

Multimedia

Human-computer interaction involving text, graphics, voice and video. Multimedia also refers to storage devices that are used to store multimedia content.

Streaming media

Refers to multimedia files, such as video clips and audio, that begin playing immediately or within seconds after it is received by a computer from the Internet or Web. Thus, the media content is consumed as it is delivered from the server rather than waiting until an entire file is downloaded.

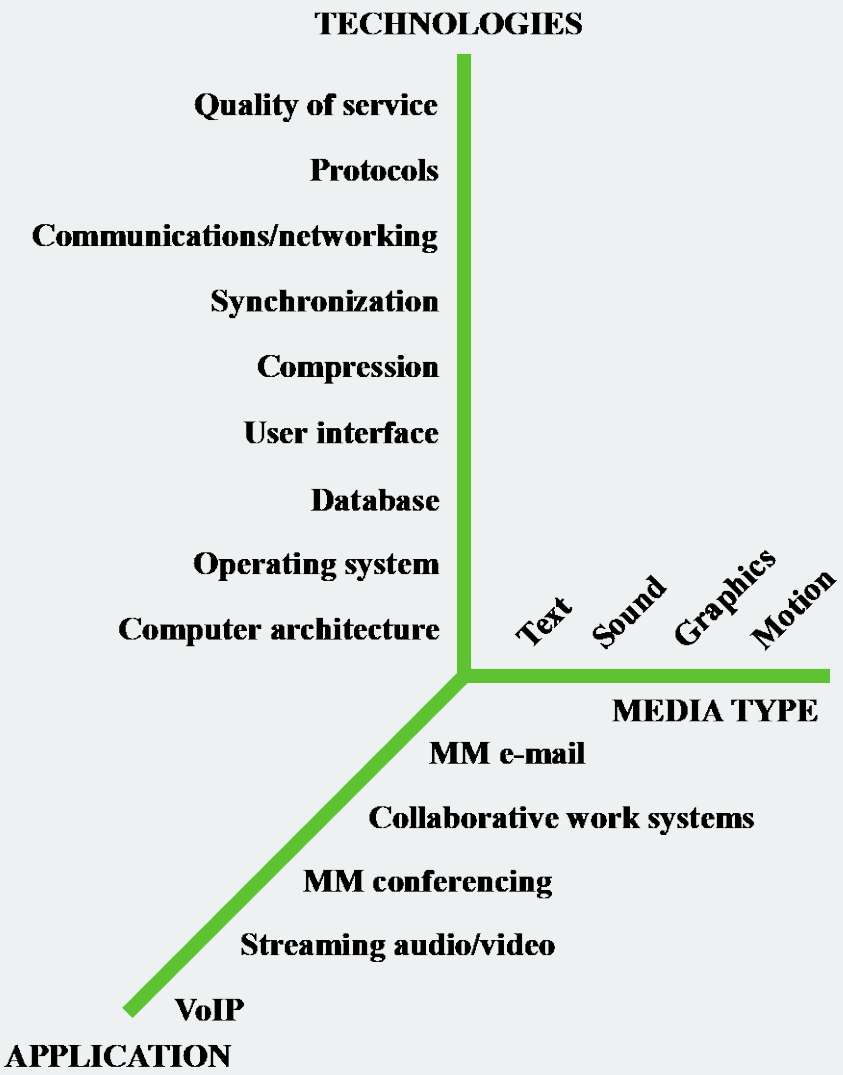
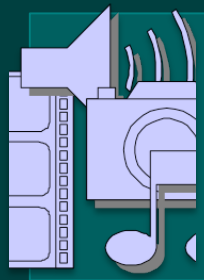


Figure 2.11 A Multimedia Taxonomy

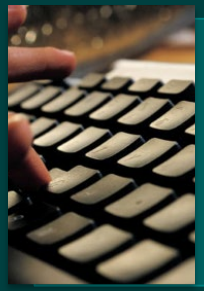
Media Types



audio generally encompasses sounds that are produced by the human speech mechanism

image supports the communication of individual pictures, charts, or drawings

video service carries sequences of pictures in time



text is information that can be entered via a keyboard and is directly readable and printable

Table 2.3

Domains of Multimedia Systems and Example Applications

Domain	Example Application
Information management	Hypermedia, multimedia-capable databases, content-based retrieval
Entertainment	Computer games, digital video, audio (MP3)
Telecommunication	Videoconferencing, shared workspaces, virtual communities
Information publishing/delivery	Online training, electronic books, streaming media

Multimedia Applications

Information systems

- Information kiosks, electronic books that include audio and video, and multimedia expert systems

Communication systems

- Support collaborative work, such as videoconferencing

Entertainment systems

- Computer and network games and other forms of audiovisual entertainment

Business systems

- Business-oriented multimedia presentations, video brochures, and online shopping

Educational systems

- Electronic books with a multimedia component, simulation and modeling applets, and other teaching support systems

Multimedia Technologies

- Some technologies that are relevant to the support of multimedia applications are:

Compression	Communications/ networking	Protocols	Quality of service (QoS)
JPG for still images	Refers to the transmission and networking technologies that can support high-volume multimedia traffic	RTP	Can deal with priority, delay constraints, delay variability constraints, and other similar requirements
MPG for video		SIP	

Sockets Programming

- Concept was developed in the 1980s in the UNIX environment as the Berkeley Sockets Interface
  - De facto standard application programming interface (API)
  - Basis for Window Sockets (WinSock)
- Enables communication between a client and server process
- May be connection oriented or connectionless

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	**domain** Protocol family of the socket to be created (AF_UNIX, AF_INET, AF_INET6) **type** Type of socket to be opened (stream, datagram, raw) **protocol** Protocol to be used on socket (UDP, TCP, ICMP)
`bind( )`	Bind a socket to a port address	**sockfd** Socket to be bound to the port address **localaddress** Socket address to which the socket is bound **addresslength** Length of the socket address structure
`listen( )`	Listen on a socket for inbound connections	**sockfd** Socket on which the application is to listen **queuesize** Number of inbound requests that can be queued at any time
`accept( )`	Accept an inbound connection	**sockfd** Socket on which the connection is to be accepted **remoteaddress** Remote socket address from which the connection was initiated **addresslength** Length of the socket address structure
`connect( )`	Connect outbound to a server	**sockfd** Socket on which the connection is to be opened **remoteaddress** Remote socket address to which the connection is to be opened **addresslength** 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)	**sockfd** Socket across which the data will be sent or read
		data Data to be sent, or buffer into which the read data will be placed
		datalength Length of the data to be written, or amount of data to be read

`sendto( )` `recvfrom( )`	Send and receive data on a datagram socket	**sockfd** Socket across which the data will be sent or read **data** Data to be sent, or buffer into which the read data will be placed **datalength** Length of the data to be written, or amount of data to be read
`close( )`	Close a socket	**sockfd** Socket which is to be closed

(Table can be found on page 54 in textbook)

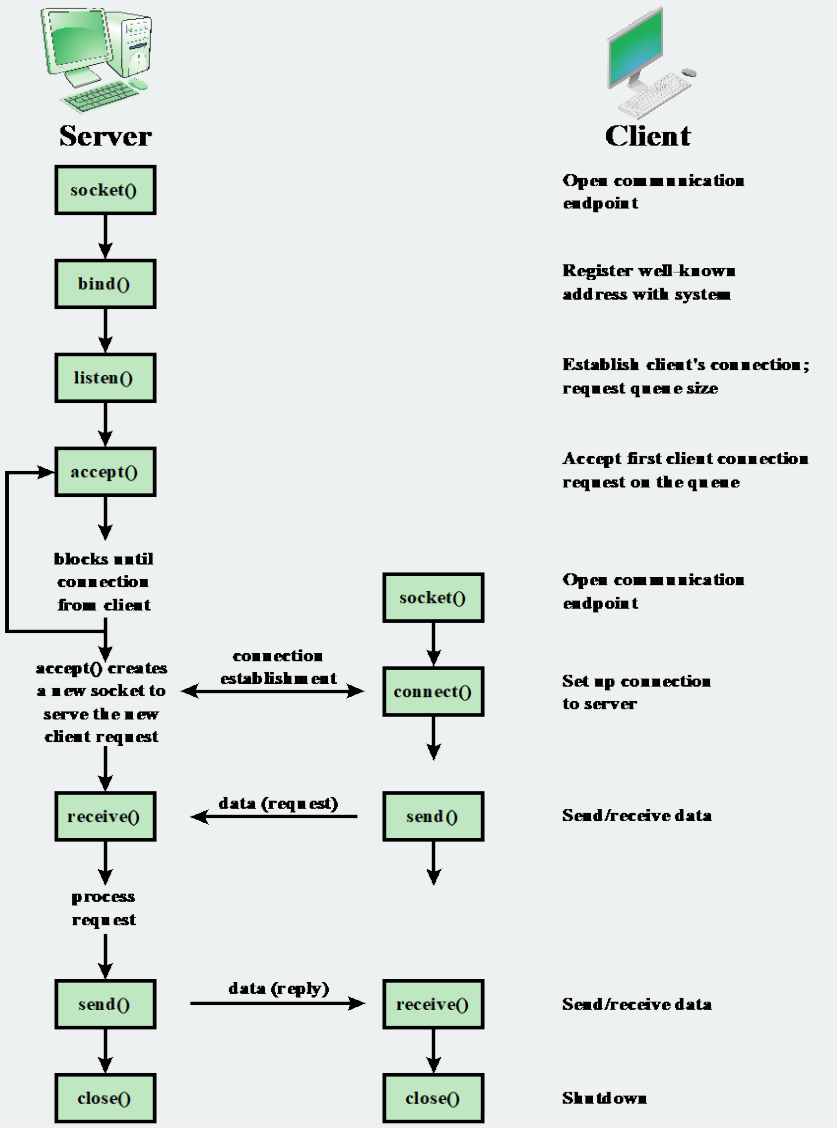


Figure 2.12 Socket System Calls for Connection-Oriented Protocol

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

void error(char *msg)
{
    perror(msg);
    exit(1);
}

int main(int argc, char *argv[])
{
     int sockfd, newsockfd, portno, clilen;
     char buffer[256];
     struct sockaddr_in serv_addr, cli_addr;
     int n;
     if (argc < 2) {
         fprintf(stderr,"ERROR, no port provided\n");
         exit(1);
     }
     sockfd = socket(AF_INET, SOCK_STREAM, 0);
     if (sockfd < 0)
        error("ERROR opening socket");
     bzero((char *) &serv_addr, sizeof(serv_addr));
     portno = atoi(argv[1]);
     serv_addr.sin_family = AF_INET;
     serv_addr.sin_port = htons(portno);
     serv_addr.sin_addr.s_addr = INADDR_ANY;
     if (bind(sockfd, (struct sockaddr *) &serv_addr,
              sizeof(serv_addr)) < 0)
              error("ERROR on binding");
     listen(sockfd,5);
     clilen = sizeof(cli_addr);
     newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
     if (newsockfd < 0)
          error("ERROR on accept");
     bzero(buffer,256);
     n = read(newsockfd,buffer,255);
     if (n < 0) error("ERROR reading from socket");
     printf("Here is the message: %s\n",buffer);
     n = write(newsockfd,"I got your message",18);
     if (n < 0) error("ERROR writing to socket");
     return 0;
}
```

Figure 2.13 Sockets Server

(Figure 2.13 can be found on page 57 in textbook)

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

void error(char *msg)
{
    perror(msg);
    exit(0);
}

int main(int argc, char *argv[])
{
   int sockfd, portno, n;
   struct sockaddr_in serv_addr;
   struct hostent *server;
   char buffer[256];
   if (argc < 3) {
      fprintf(stderr,"usage %s hostname port\n", argv[0]);
      exit(0);
   }
   portno = atoi(argv[2]);
   sockfd = socket(AF_INET, SOCK_STREAM, 0);
   if (sockfd < 0)
       error("ERROR opening socket");
   server = gethostbyname(argv[1]);
   if (server == NULL) {
       fprintf(stderr,"ERROR, no such host\n");
       exit(0);
   }
   bzero((char *) &serv_addr, sizeof(serv_addr));
   serv_addr.sin_family = AF_INET;
   bcopy((char *)server->h_addr,
        (char *)&serv_addr.sin_addr.s_addr,
        server->h_length);
   serv_addr.sin_port = htons(portno);
   if (connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr)) < 0)
       error("ERROR connecting");
   printf("Please enter the message: ");
   bzero(buffer,256);
   fgets(buffer,255,stdin);
   n = write(sockfd,buffer,strlen(buffer));
   if (n < 0)
        error("ERROR writing to socket");
   bzero(buffer,256);
   n = read(sockfd,buffer,255);
   if (n < 0)
        error("ERROR reading from socket");
   printf("%s\n",buffer);
   return 0;
}
```

Figure 2.14 Sockets Client

(Figure 2.14 can be found on page 58 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