Data and Computer Communications

Tenth Edition

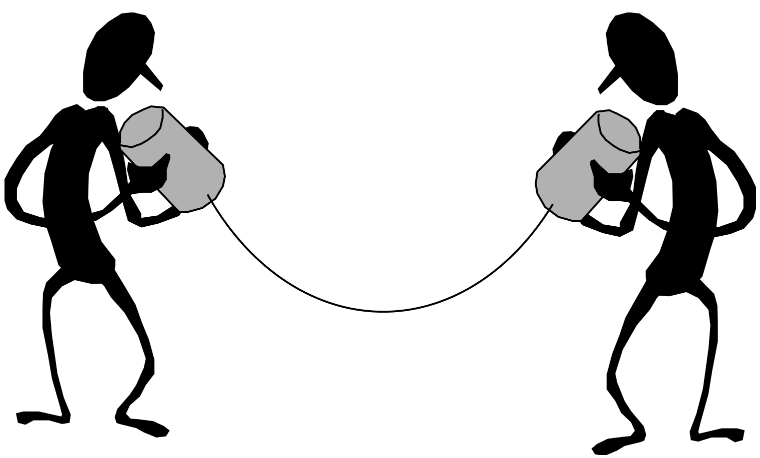
by William Stallings

CHAPTER 2

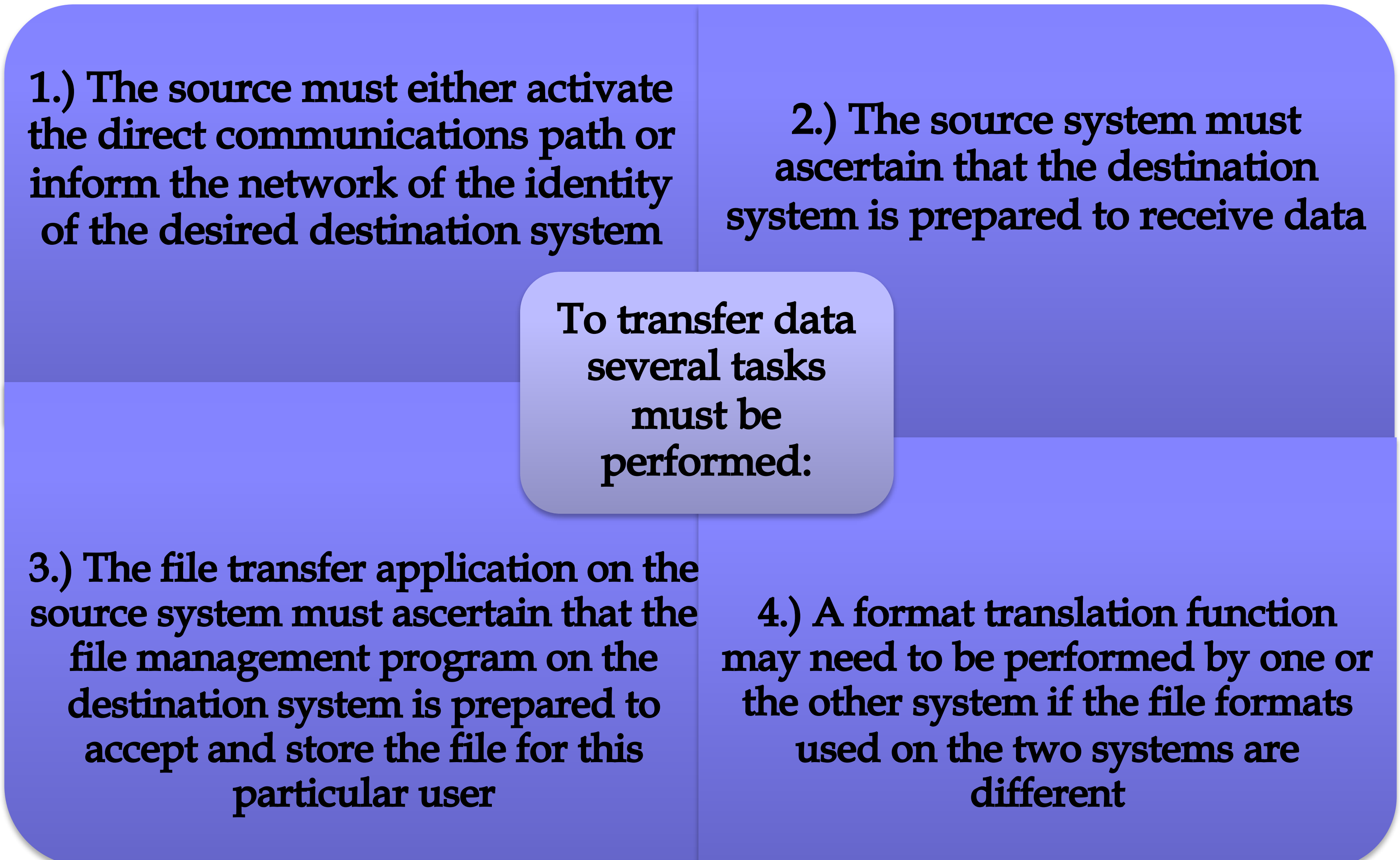
Protocol Architecture, TCP/IP, and Inte rnet-Based Applications

To destroy communication completely, there must be no r
ules in common between transmitter and receiver—neithe
r of alphabet nor of syntax.

—*On Human Communication*,
Colin Cherry



The Need for a Protocol Architecture



Functions of Protocol Architecture

- Breaks logic into subtask modules which are implemen ted separately
- Modules are arranged in a vertical stack
  - Each layer in the stack performs a subset of function
  - 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 pe
er layers to communicate

The key features of a protocol are:

Syntax	• Format of data blocks
Semantics	• Control information for coordination and error handling
Timing	• Speed matching and sequencing

A Simple Protocol Architecture

Agents involved:

- Applications
- Computers
- Networks

Examples of applications include file transfer and electronic mail

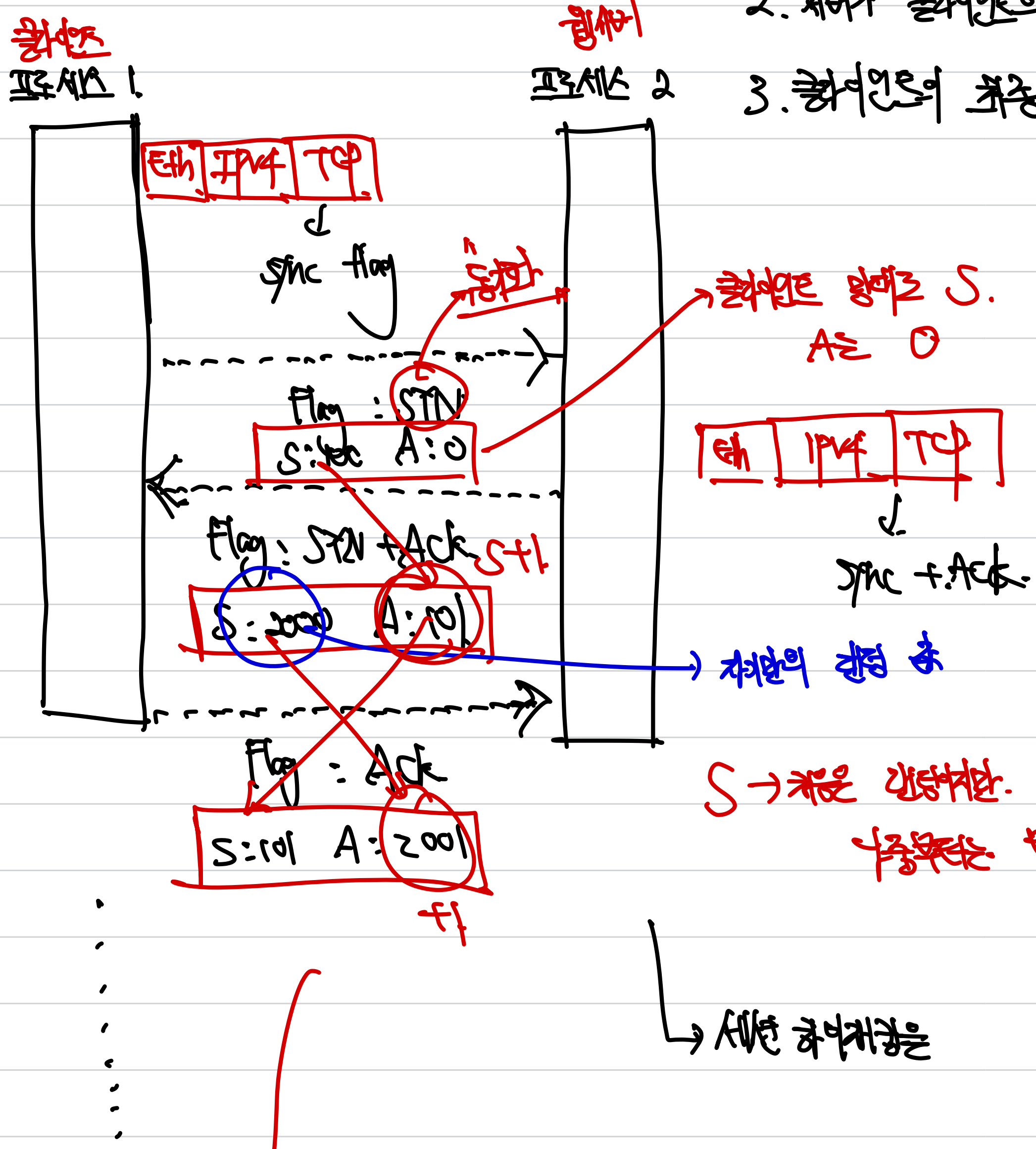
These execute on computers that support multiple simultaneous applications

TCP를 이용한 통신과정

1. 클라이언트가 서버에게 요청 패킷을 보냄
2. 서버가 클라이언트의 요청을 받아들이는 패킷
3. 클라이언트의 최종 수락 패킷.

↓

"3 Way Handshake.

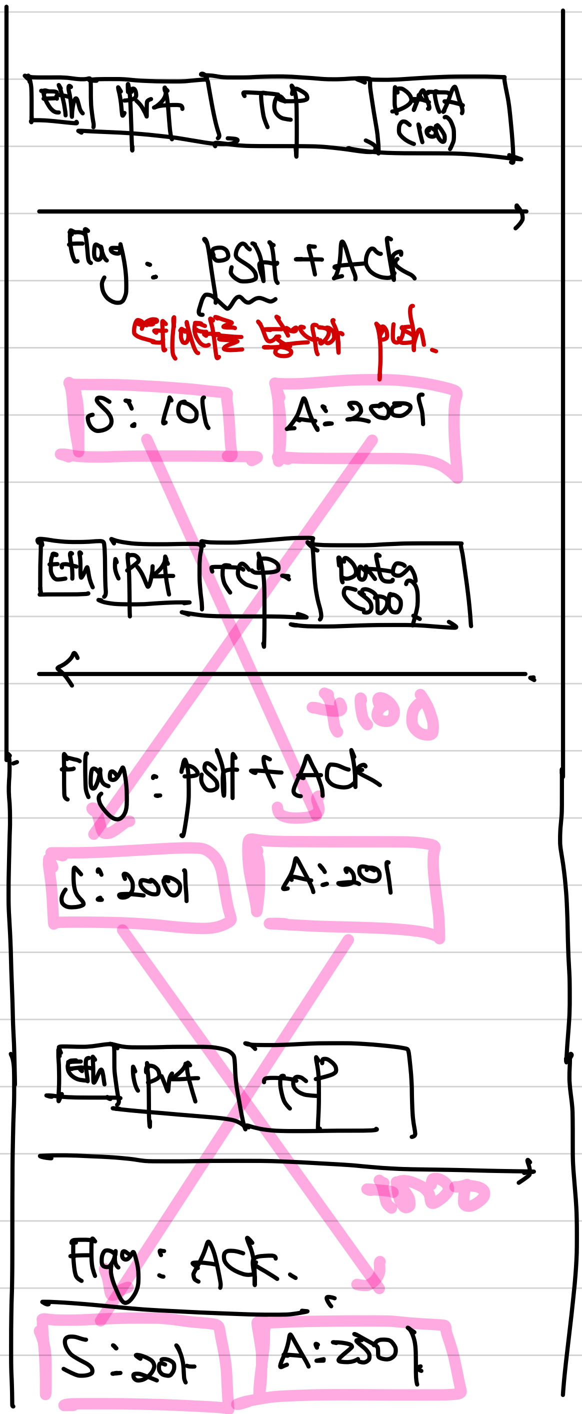


연결이 수립후 데이터를 주고 받음

(규칙)

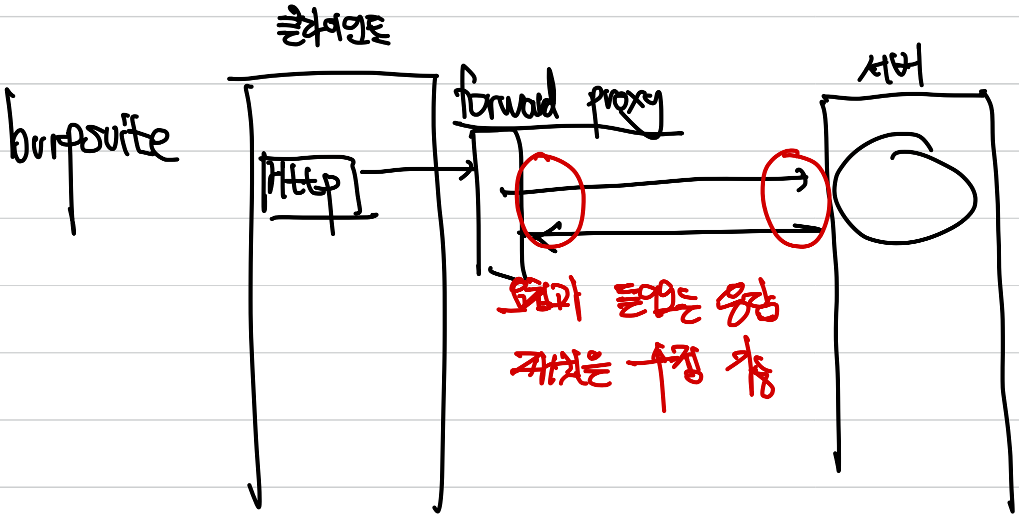
1. 보낸 쪽에서 또 보낼 때는 SEQ 번호와 Ack 번호 그대로
2. 받는 쪽에서 SEQ 번호는 받은 Ack 번호가 된다
3. 받는 쪽에서 Ack 번호는 받은 SEQ 번호 + 데이터의 크기

클라이언트 웹서버 TCP 상태 천이도 (연결상태)



→ HTTP

→ HTTP 1.0의 특징.



계층별 장비

1계층

① 리피터 (MAC, IP 보르고 전기적 신호를 증폭).

② 허브 (전기적 신호를 증폭 (but 물리적으로만 연결) LAN 전송거리를 연장 → 여러 대의 장비를 LAN에 접속할 수 있도록 한다) → broadcast (flooding을 한다)

MAC 주소를 모른다 ↔ 스위치, 라우터

broadcast → 의도적

flooding → 의도적 X

허브에 연결된 장비들은 하나의 Collision domain 안에 있다.

하나의 → 1층에 장치

↳ 스위치의 핵심 기술 (CSMA/CD).

2계층

③ 브릿지 → (허브와는 다르게 MAC 주소를 보고 중)

소프트 웨어

하드웨어.

④ 스위치 → (→ 각각의 포트가 Collision domain에 있다)

3계층

⑤ L3 스위치, 라우터 (

Routing이 핵심 + ACL

→ 전 세계가 연결.

↳ 멀티캐스트, 브로드 캐스트, 목적지를 모르는 유니캐스트는 수신한 경우 수신포트를 제외한 모든 포트로 flooding (X)

↳ 이런 Frame을 모두 차단

통신 케이블

100BASE-T

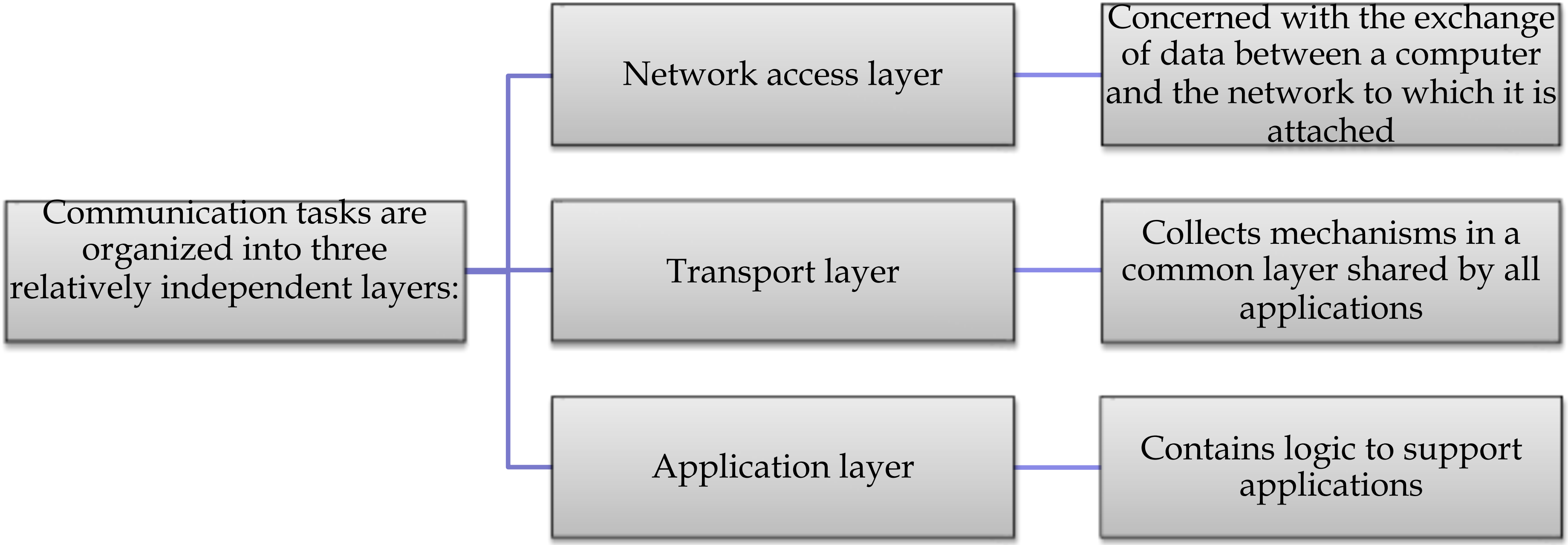
100 → LAN Speed (Mbps)

T → Twisted: (숫자면 최대 통신 거리)

BASE → Baseband용 cable (digital)

BROAD → broadband용 cable (analog)

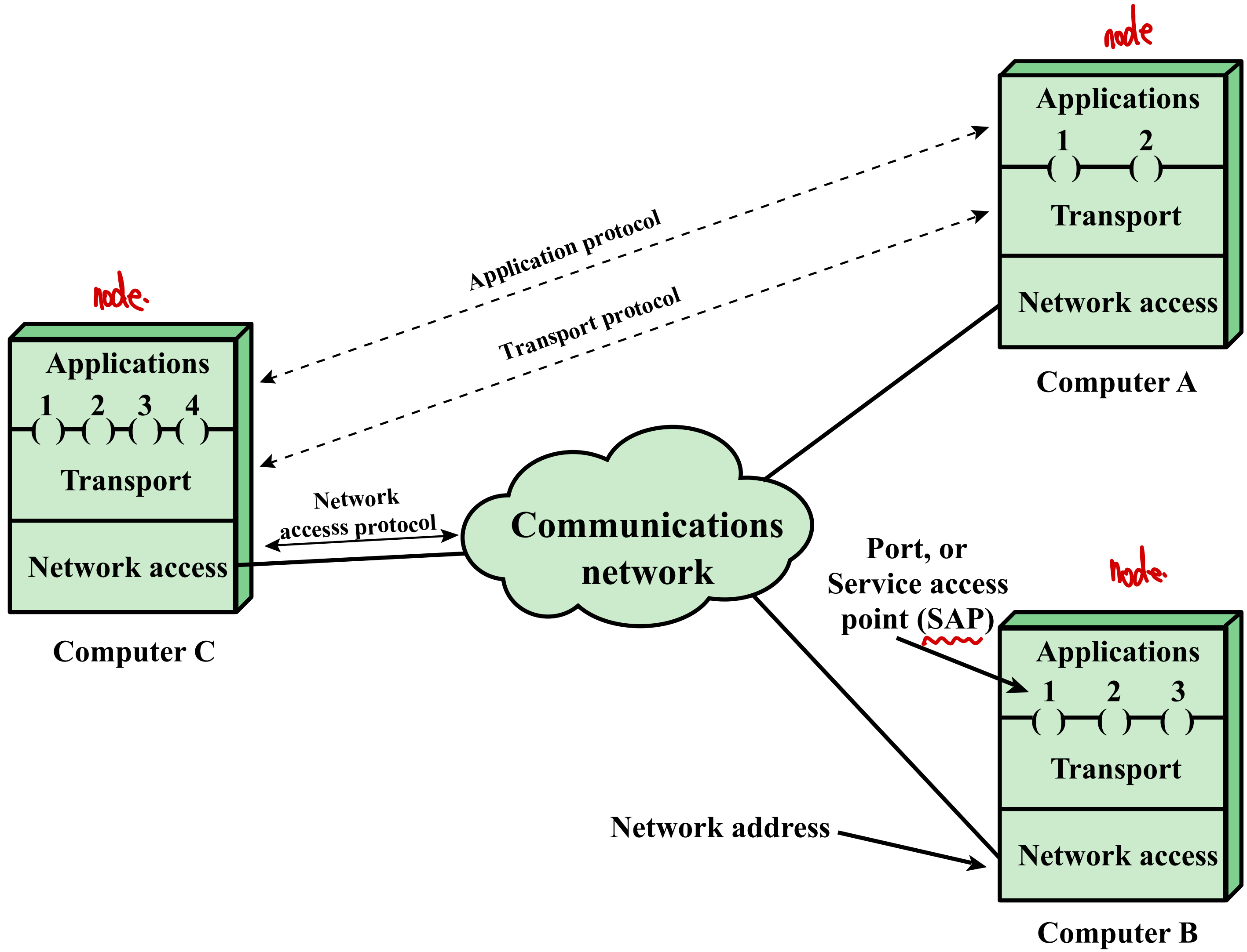
Communication Layers



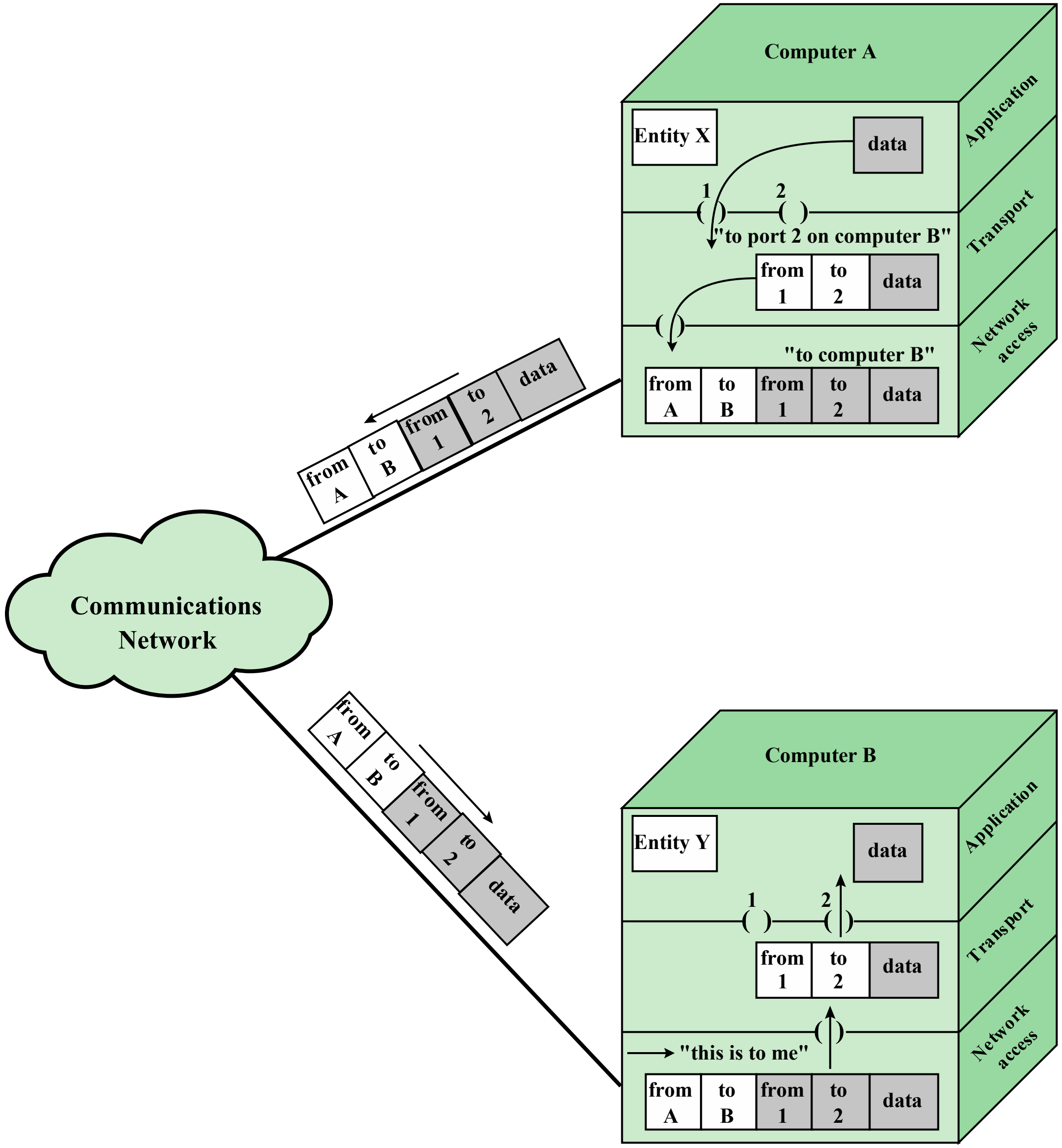
7계층 ("프로토콜 HTTP")

→ 소켓 프로그래밍

→ 프로토콜을 "만들 수 있음"



Protocol Architectures and Networks



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

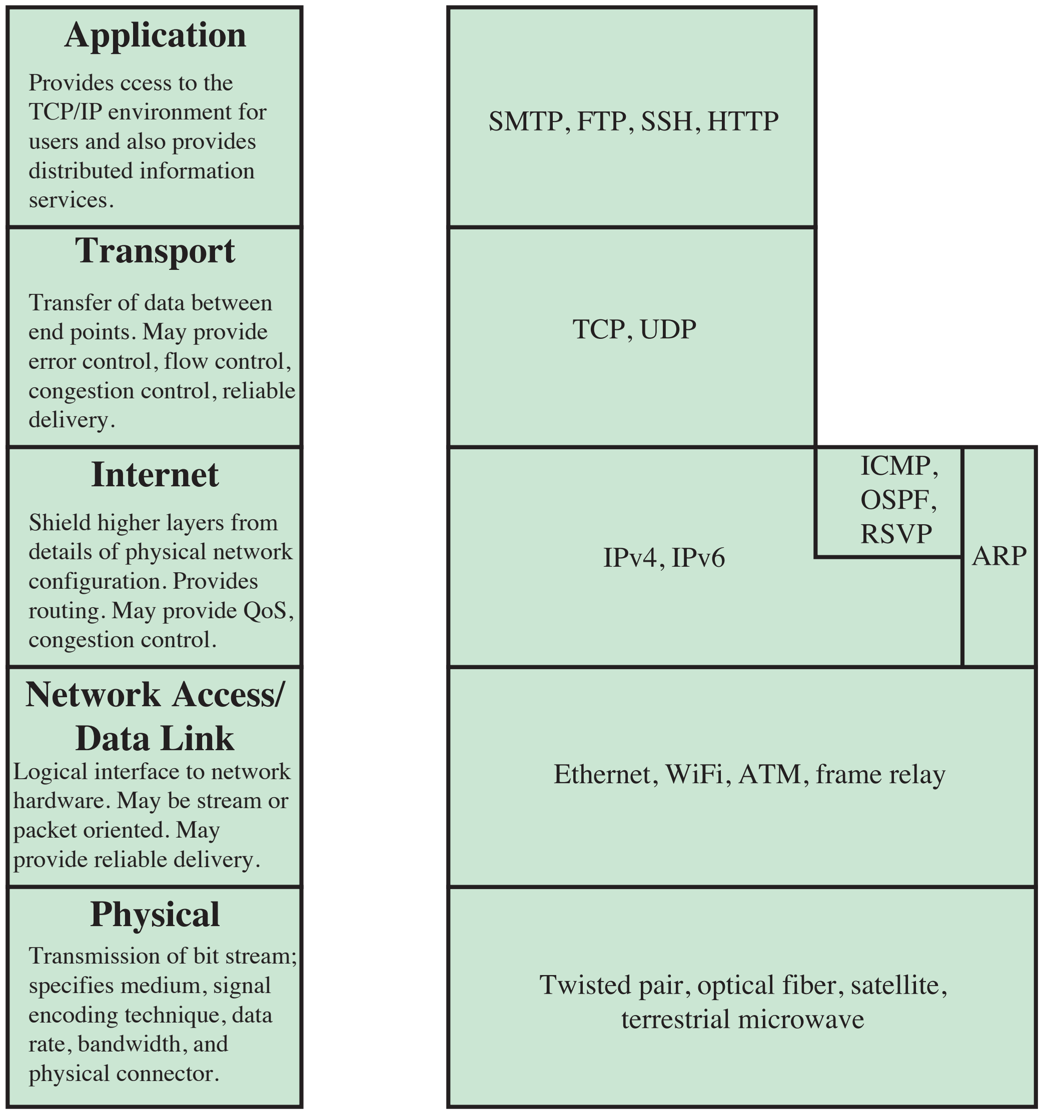
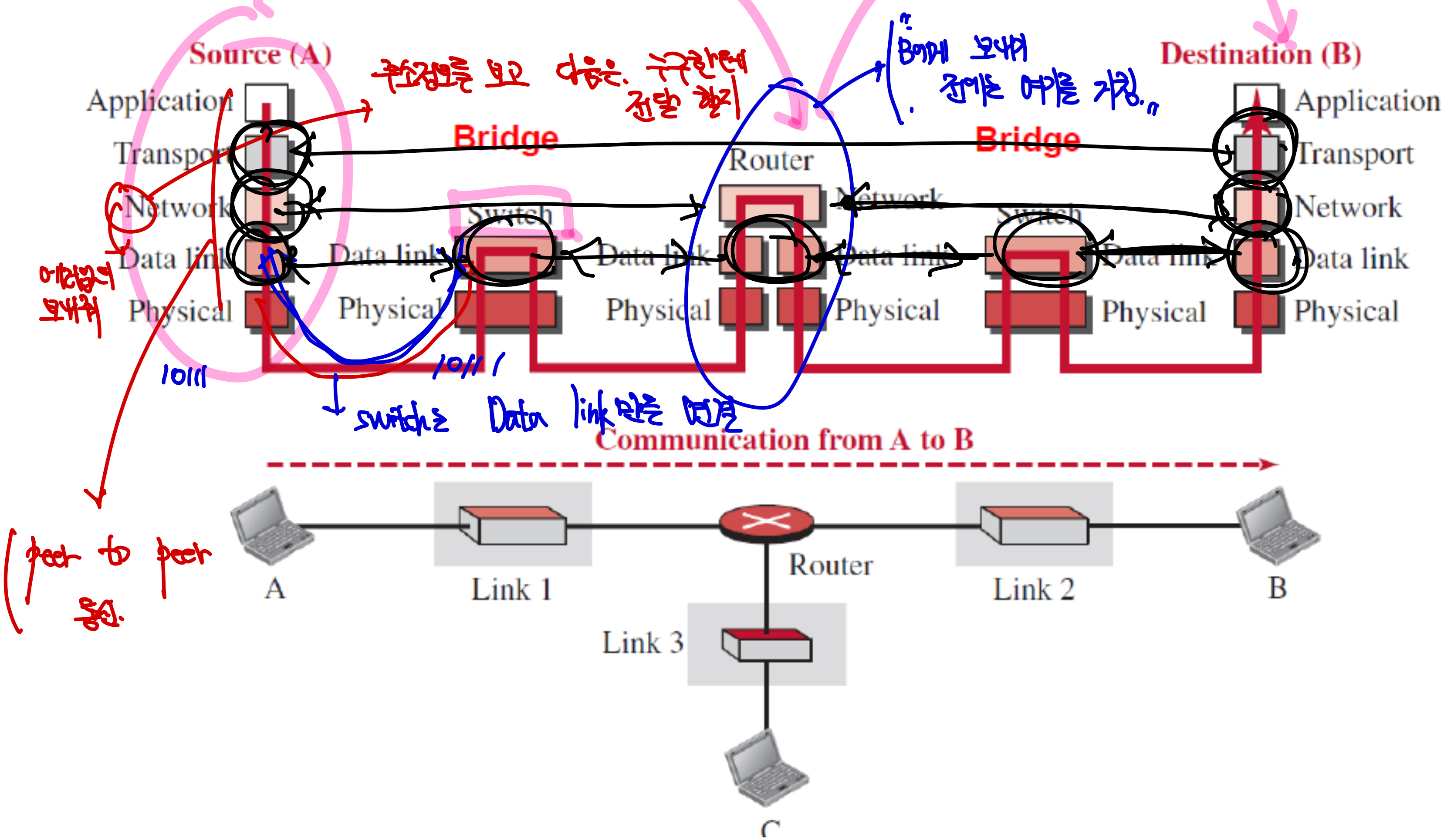


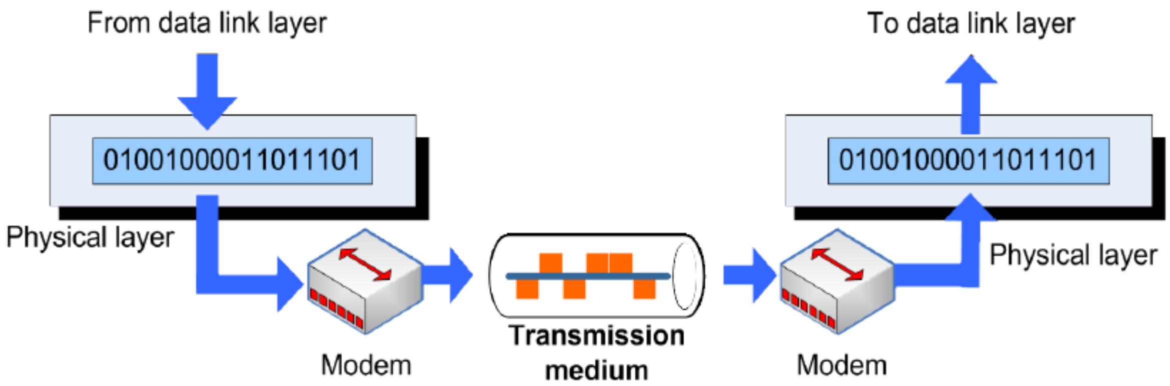
Figure 2.3 The TCP/IP Layers and Example Protocols

Internet Communications



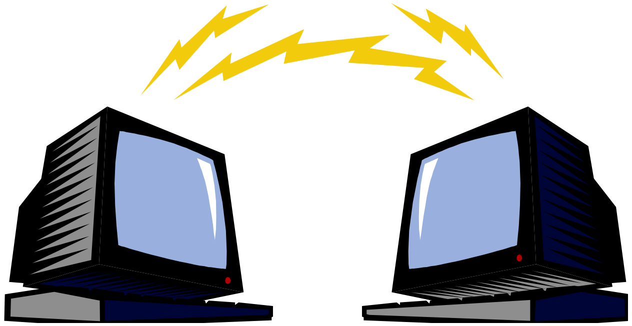
Physical Layer

- Covers the physical interface between computer and network
- Concerned with issues like:
  - Characteristics of transmission medium
  - Nature of the signals
  - Data rates
- Protocols : V.24, RS-232C, RS-449, CDMA PHY, Coax, Fiber, Satellite

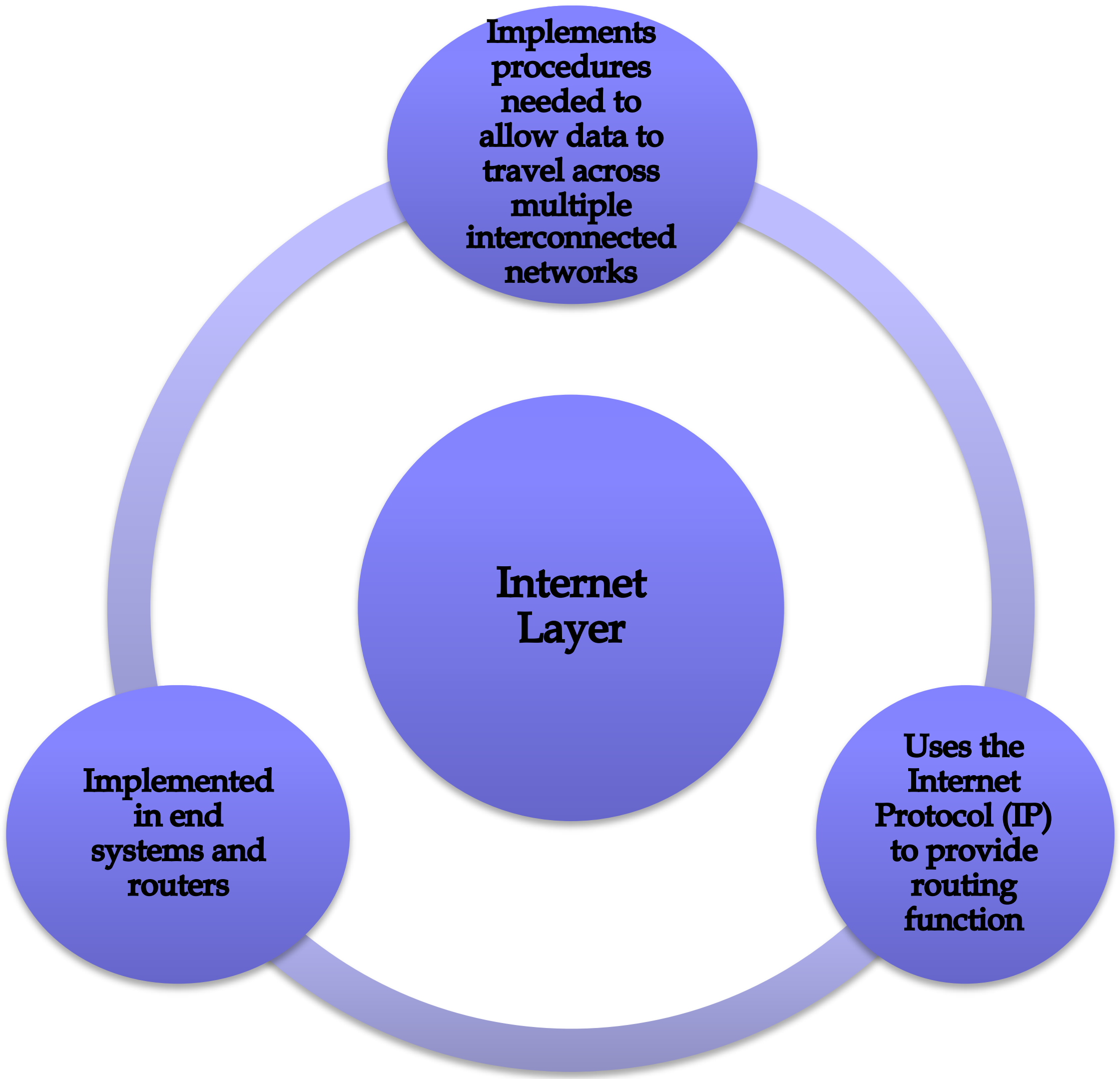


Network Access/Data Link Layer

- 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 sy stems attached to the same network
- Protocols : HDLC, SDLC, LAPB, ATM…



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 u ser applications
- A separate module is needed for each different typ e of application that is peculiar to that application

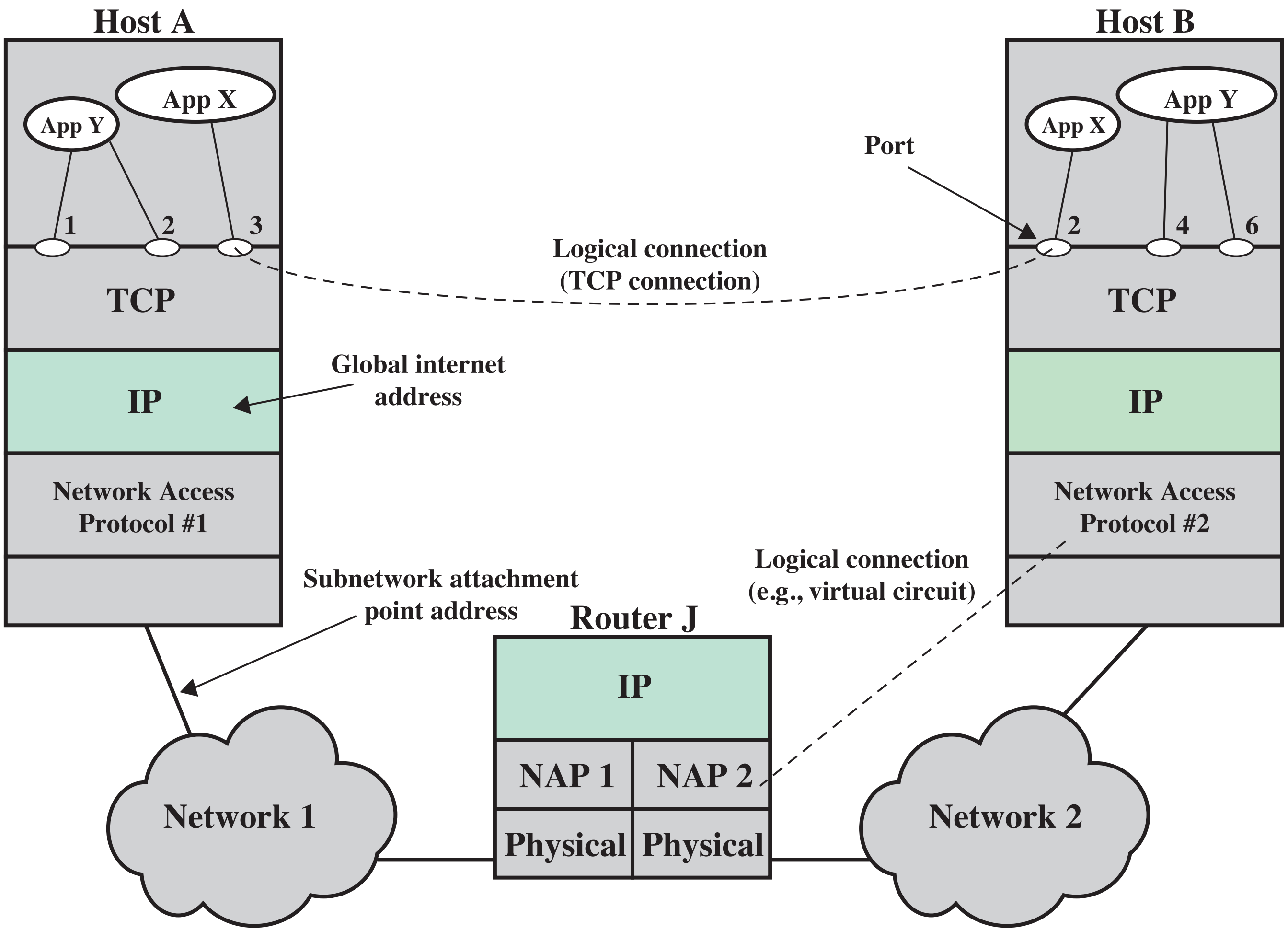


Figure 2.4 TCP/IP Concepts

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

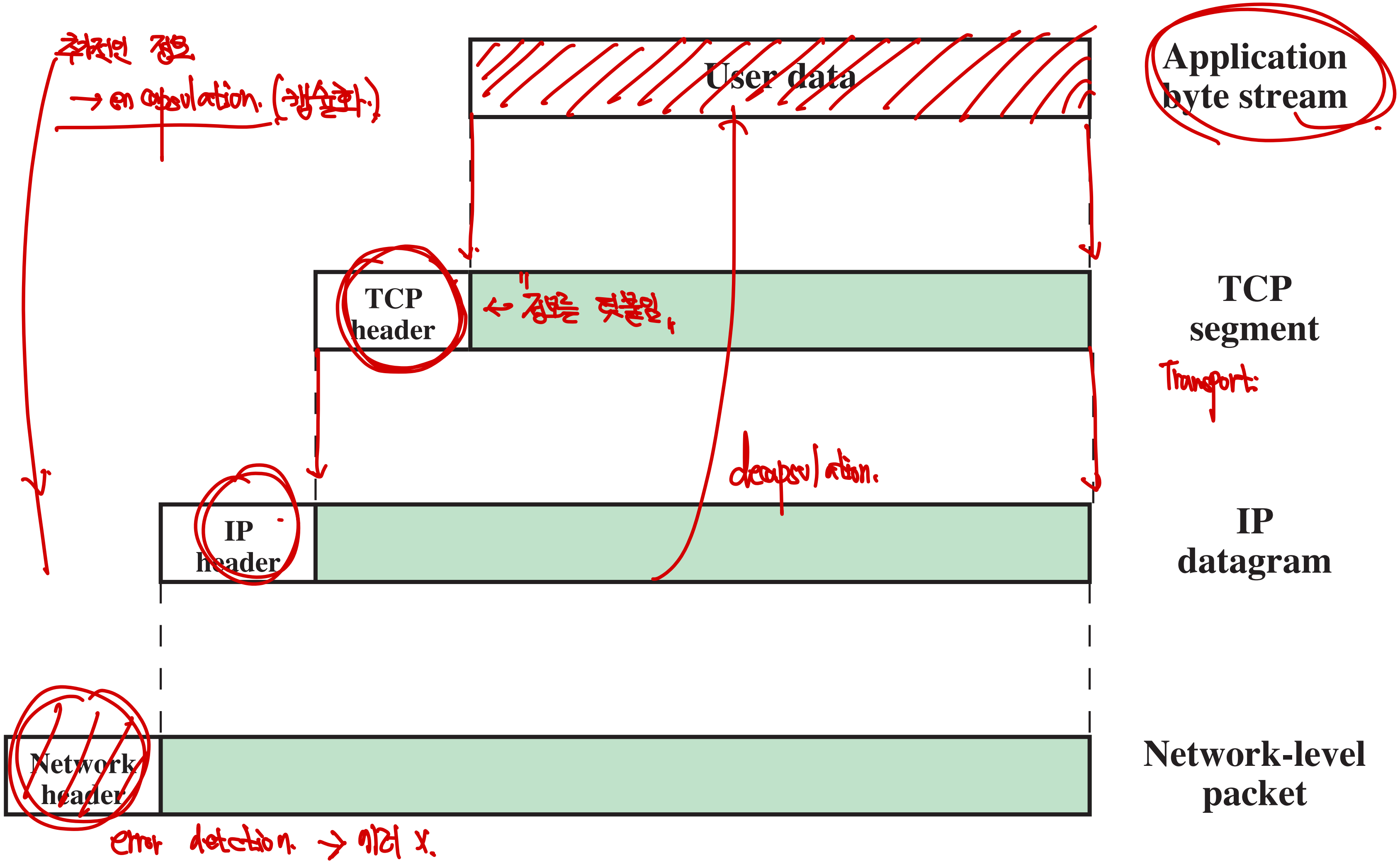
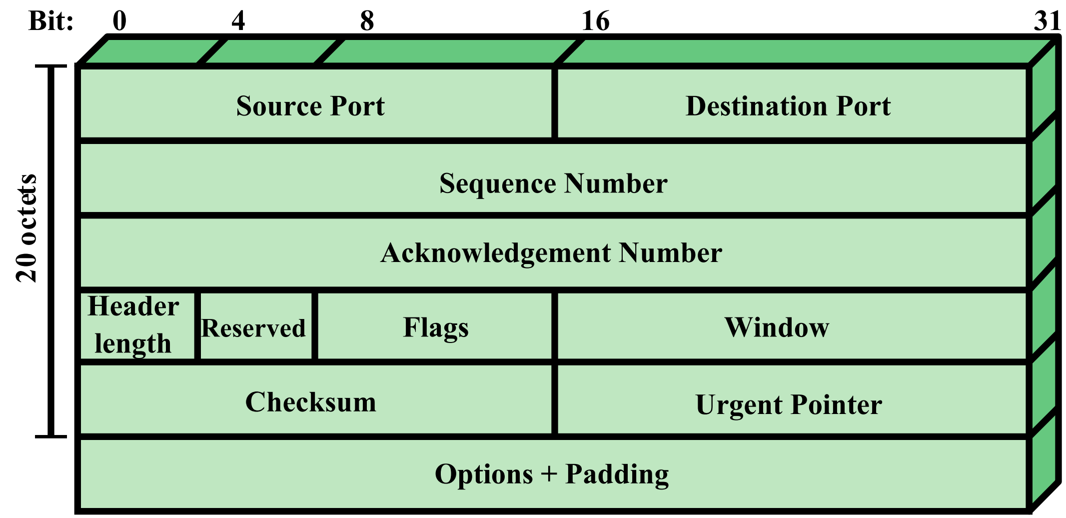


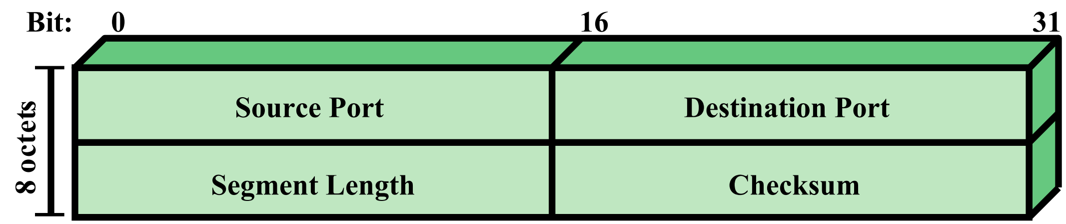
Figure 2.5 Protocol Data Units (PDUs) in the TCP/IP Architecture

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

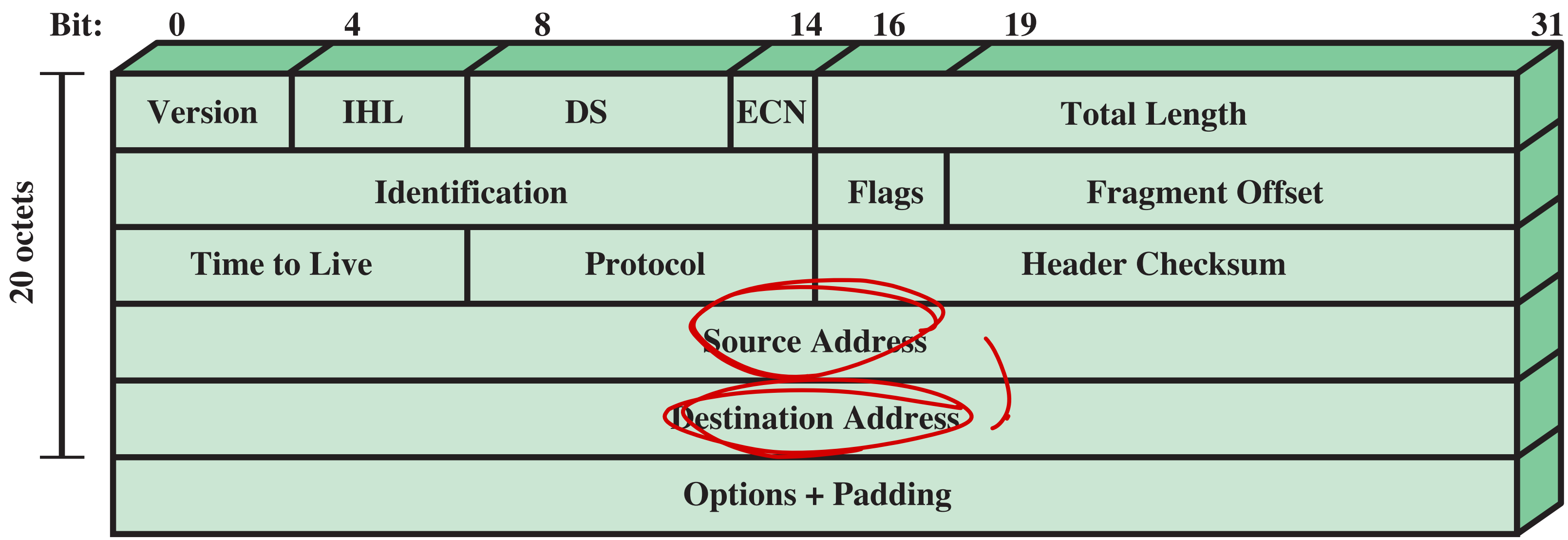


(b) UDP Header

TCP and UDP Headers

User Datagram Protocol (UDP)

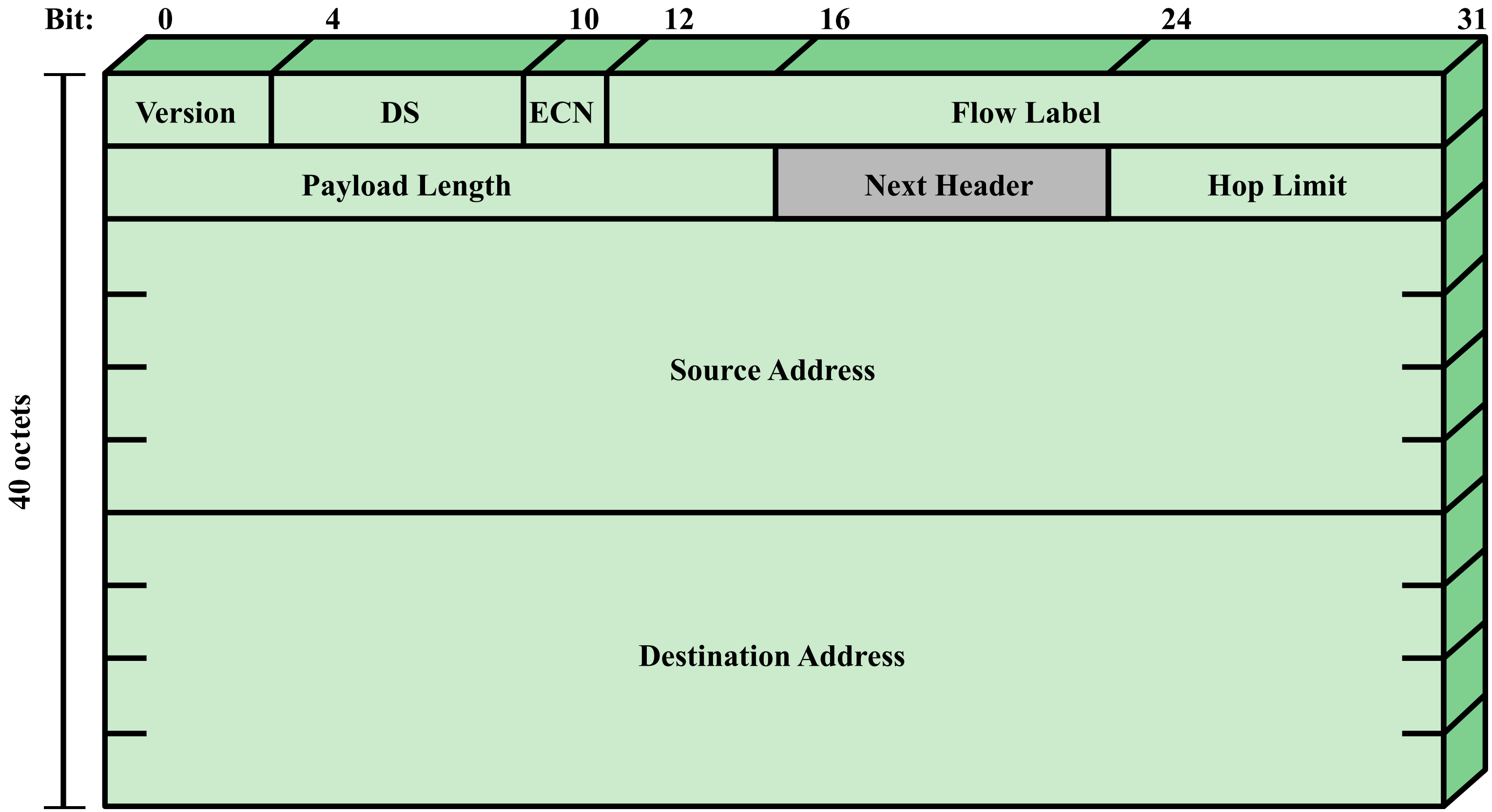
- Alternative to TCP
- Does not guarantee delivery, preservation of sequen ce, or protection against duplication
- Enables a procedure to send messages to other proce dures with a minimum of protocol mechanism
- Adds port addressing capability to IP
- Used with Simple Network Management Protocol (S NMP)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



(a) IPv4 Header

network 계층

(a) IPv4 Header

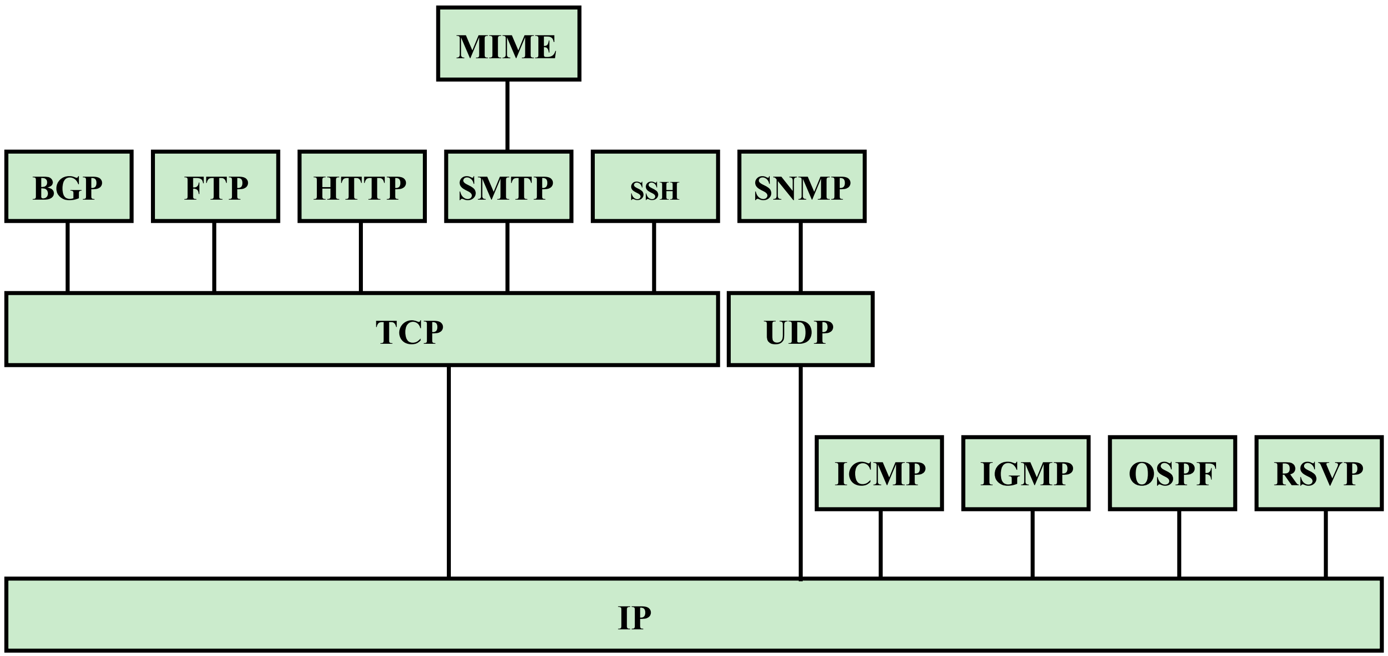


IPv6 Header

DS = Differentiated services field
ECN = Explicit congestion notification field

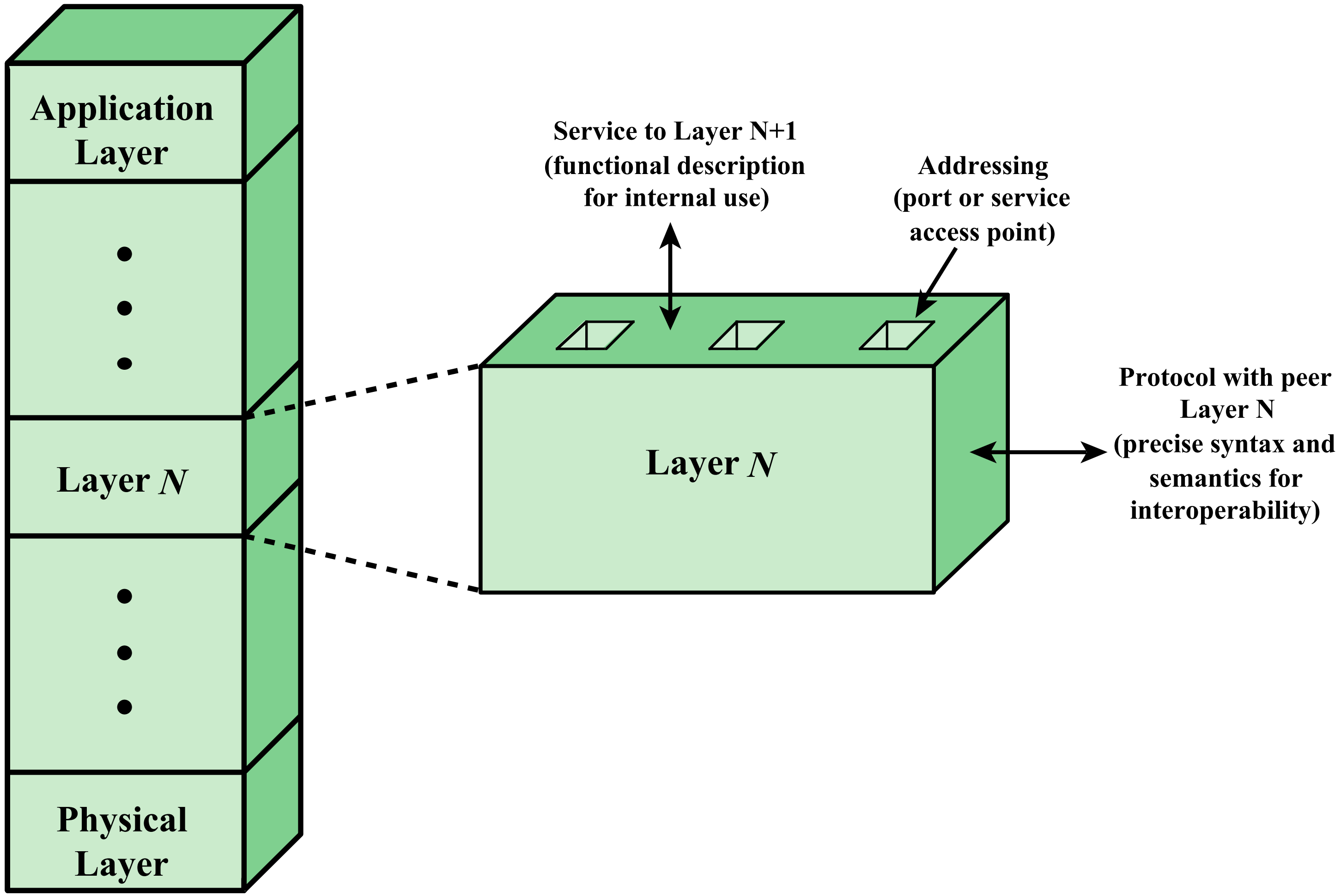
Note: The 8-bit DS/ECN fields were formerly known as the Type of Service field in the IPv4 header and the Traffic Class field in the IPv6 header.

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**

Some Protocols in the TCP/IP Protocol Suite



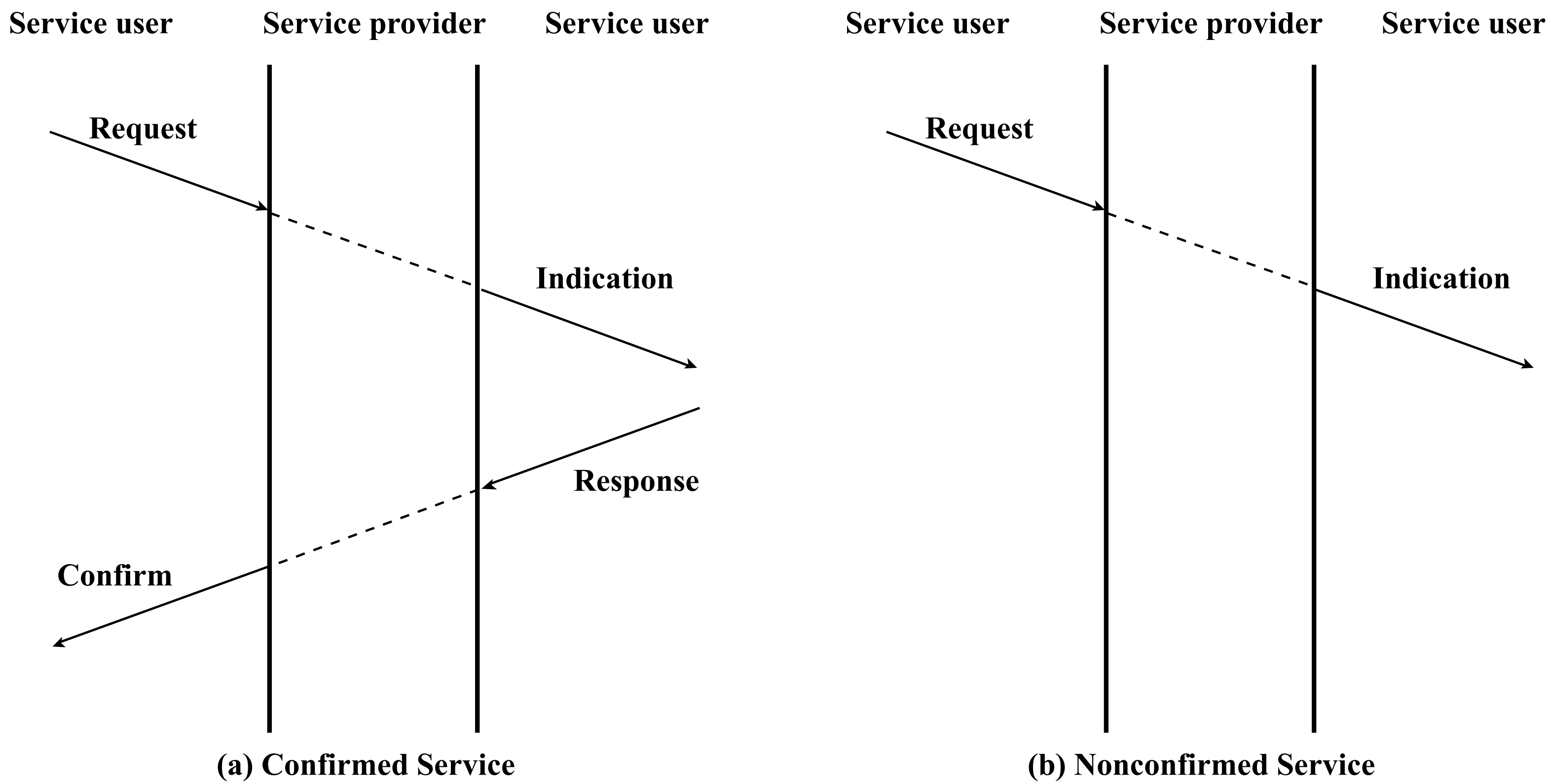
A Protocol Architecture as a Framework for Standardization

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



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

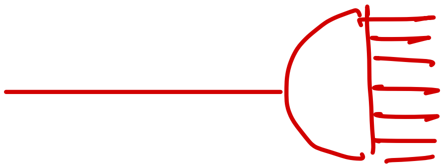
Secure Shell (SSH)

- Provides a secure remote logon capability

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

Socket ()



bind (). ip
port:

The Socket

- Formed by the concatenation of a port value and an IP a ddress
  - 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 f or 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

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 2.4

Core Socket Functions

(Table can be found on page 78 in textbook)

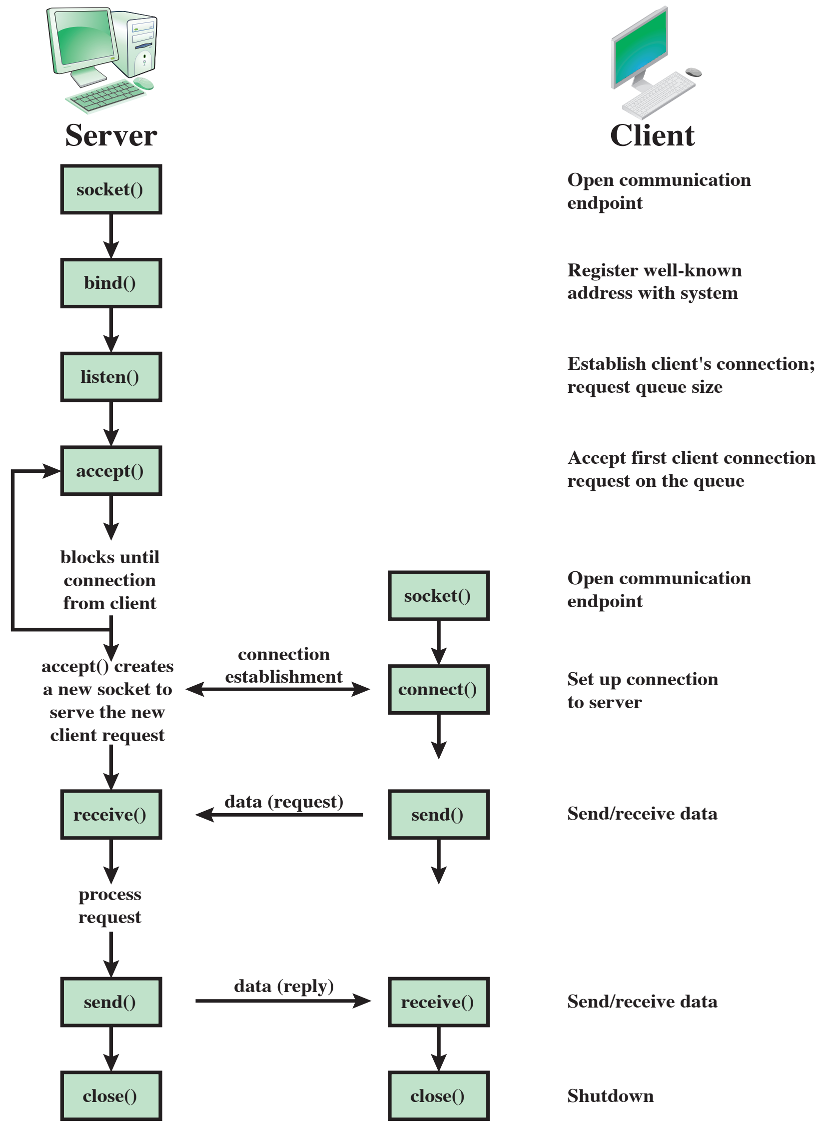


Figure 2.12 Socket System Calls for Connection-Oriented Protocol

```
#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 81 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 82 in textbook)

Summary

- The need for a protocol arc hitecture
- Simple protocol architectu re
- TCP/IP protocol architect ure
  - TCP/IP layers
  - Operation of TCP and IP
  - TCP and UDP
  - IP and IPv6
  - Protocol interfaces
- Standardization within a p rotocol architecture
  - Standards and protocol layer s
  - Service primitives and para meters
- Traditional internet-ba sed applications
- Multimedia
  - Media types
  - Multimedia application s
  - Multimedia technologie s
- Sockets programming
  - The socket
  - Sockets interface calls