Data Communication Interfaces and TCP/IP

References:

- Behrouz A. Forouzan, “TCP/IP Protocol Suite”, 2nd ed., McGraw-Hill, 2003, ISBN 0-07-246060-1.
- Numerous white papers that are available on the World Wide Web.

Data Communication Protocols

- Data communication takes place according to strictly defined ***constraints*** and ***rules***, such as:
  - the services that can be requested and provided
  - the legal formats for messages that are exchanged
  - the rules for starting and terminating a session
  - the rules for exchanging control and status information
  - the rules for exchanging data messages
  - the rules for dealing with error conditions
  - etc.
- These constraints and rules are together called a ***protocol***.
- In a ***layered communication interface***, the interactions at each layer are governed by separate protocols.

Five Elements of a Communication Protocol

“1. The ***service*** to be provided by the protocol.

2. The ***assumptions*** about the environment in which the protocol is executed.

3. The ***vocabulary*** of messages used to implement the protocol.

4. The ***encoding*** (i.e., the format) of each message in the vocabulary.

5. The ***procedure rules*** guarding the consistency of message exchanges.”

Source: G. R. Holzmann, “Design and Validation of Computer Protocols”, Prentice Hall, 1991, ISBN 0-13-539925-4.

Ex: Session and Application Layer Protocols

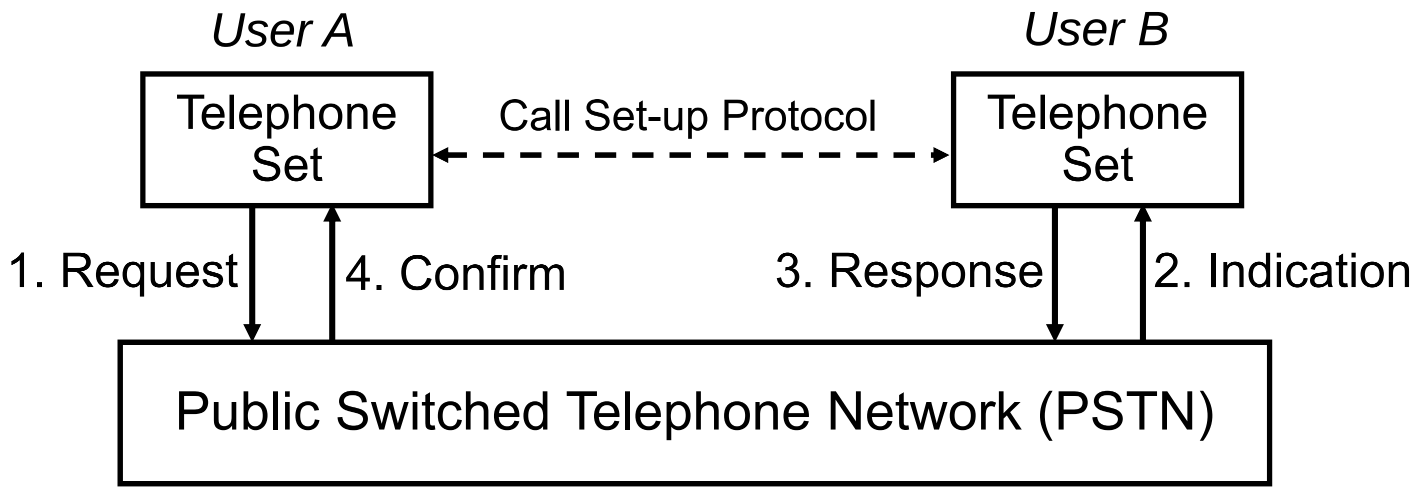
- A ***session layer*** protocol is responsible for:
  - establishing a new connection
  - choosing full or half-duplex communication mode
  - synchronizing the communication at a high level
  - possibly recovering from lost connections
  - terminating an existing connection

- An ***application layer*** protocol governs how an application (computer or human) uses an existing reliable communication connection.

- Protocol interactions between two peers at the same layer on different nodes can be viewed as having four steps: (1) ***request***, (2) ***indication***, (3) ***response***, and (4) ***confirm***.

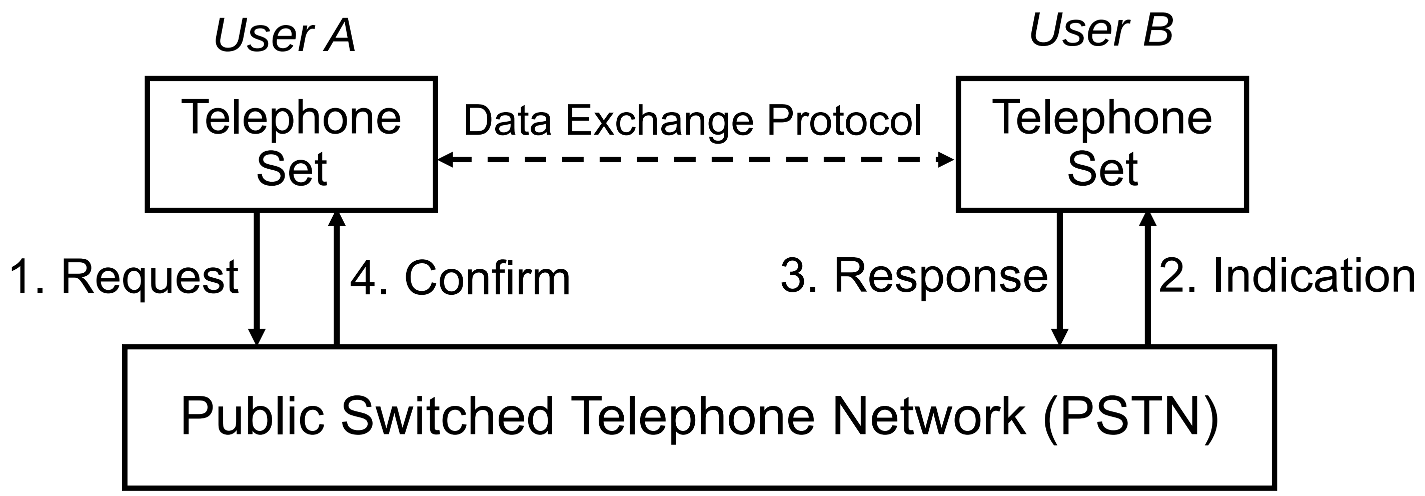
- Many of the basic protocol concepts can be illustrated using a human-to-human telephone call using old-style telephones with handsets.

Setting Up a Telephone Call (Session Layer)



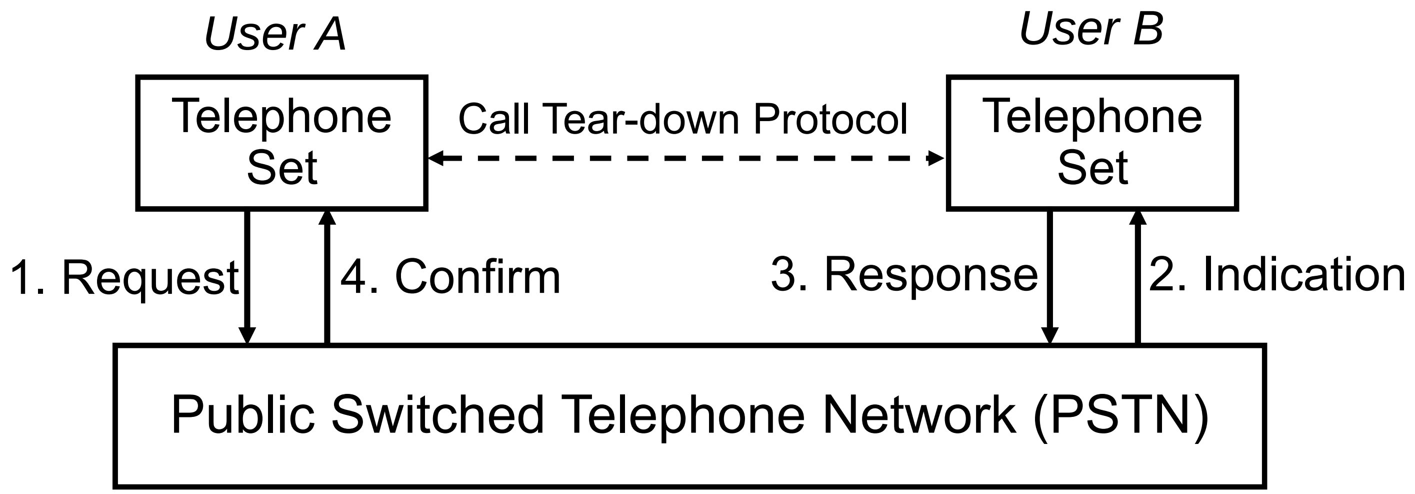
Request:	Local telephone taken off hook. Hear dial tone. Enter telephone number. Hear locally-generated ringing sound signal.
Indication:	Remote telephone receives the electrical ringing signal. Remote telephone emits audible ringing sound.
Response:	Remote telephone taken off hook. Ringing stops.
Confirm:	Local telephone no longer receives a ringing sound signal. Instead, hears silence then remote party saying “Hello!”

Telephone Conversation (Application Layer)



Request:	User A speaks a sentence into their telephone set.
Indication:	User B hears user A’s sentence being spoken.
Response:	B's response might be "Yes". Or B's response to A's sentence might be implied by the content of sentences spoken next by user B to user A.
Confirm:	User A hears user B’s "Yes" or a new sentence from B.

Ending a Telephone Call (Session Layer)

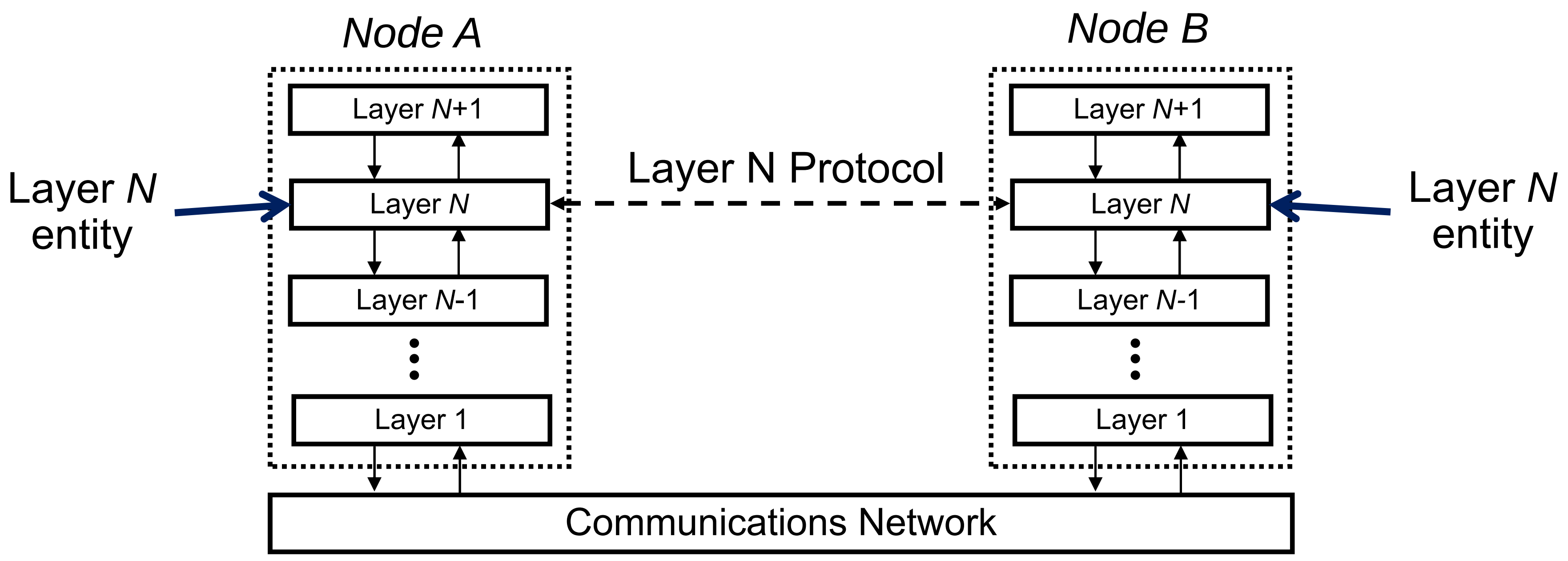


Request:	User A says “Good bye B!”.
Indication:	User B hears User A say “Good bye B!”.
Response:	User B says “Good bye A!”.
Confirm:	User A hears User B say “Good bye A!”.
Request:	User A and B both hang up. The telephone network takes down the connection that carried the call. If one user does not hang up, they hear silence at first. Eventually a loud off-hook alarm signal will be heard.

Layered Data Communication Interfaces

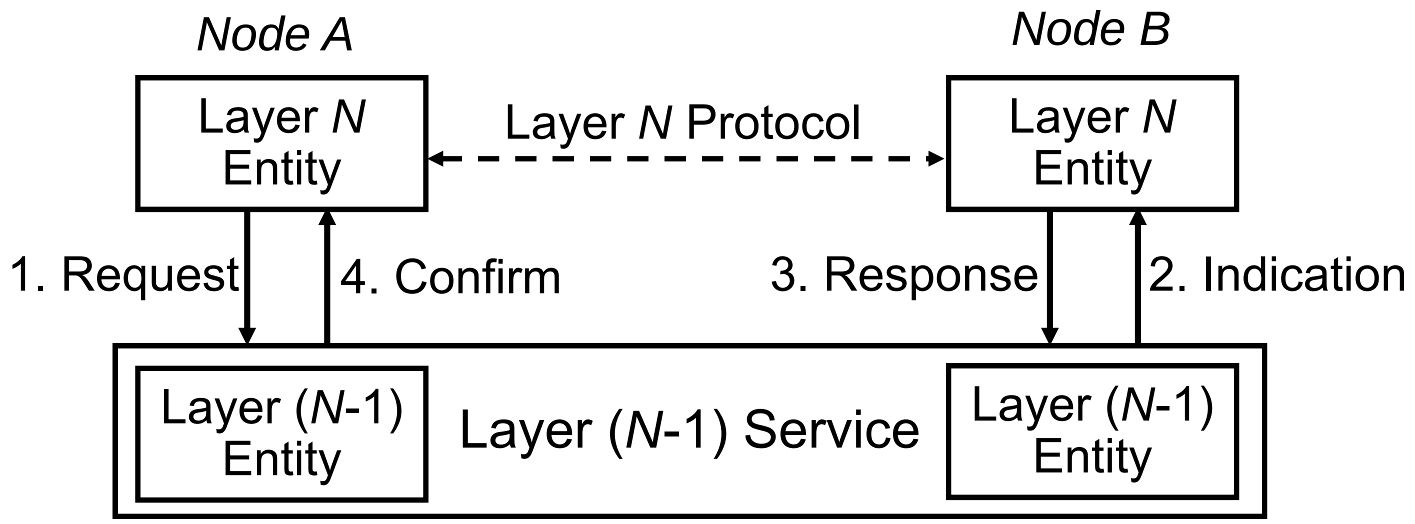
- Layering is used to manage the complexity of data communication interfaces (as well as computer systems).
- Layering provides many advantages in communications:
  - ***Divide-and-conquer*.** Each layer by itself is simpler to specify, design, and verify.
  - One layer can be modified or reworked more easily with minimal impact on the other layers.
  - Application software and the operating system can interact with the communications interface at one or more possible layers (although it is usually best to access the interface at the highest possible layer).

Layered Protocol Stacks



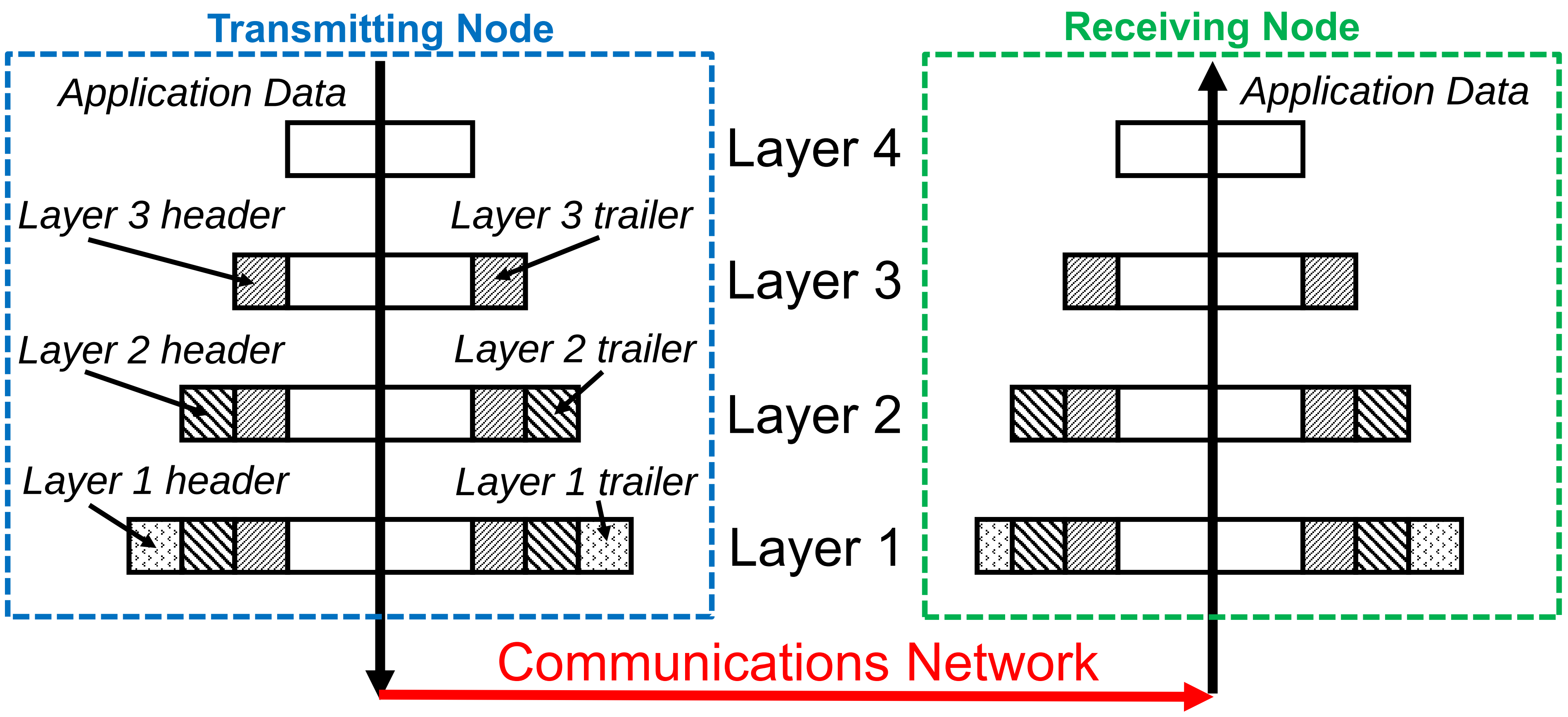
- Layer *N* provides “services” to layer *N*+1. Typical services include: (1) ***set up*** a connection; (2) ***transfer*** data; (3) ***take down*** a connection.
- Layer *N* uses the services of layer *N*-1.
- The ***layer N entity*** in Node A interacts indirectly with its ***peer*** (i.e., the layer *N* entity in Node B) according to a precisely defined set of rules called the ***layer N protocol***.

Layer N Protocol Primitives



Request:	Node A layer-*N* entity requests service from Node A layer-(*N*-1) entity, which provides layer-(*N*-1) services.
Indication:	Node B layer-(*N*-1) entity signals to Node B layer-*N* entity that its layer-*N* peer in sys. A is requesting a service.
Response:	Node B layer-*N* entity signals to Node B layer-(*N*-1) entity that it has provided the layer-*N* service for its peer in A.
Confirm:	Node A layer-(*N*-1) entity signals to Node A layer-*N* entity that the requested layer-*N* service has been done.

Protocol Overhead in Data Packets



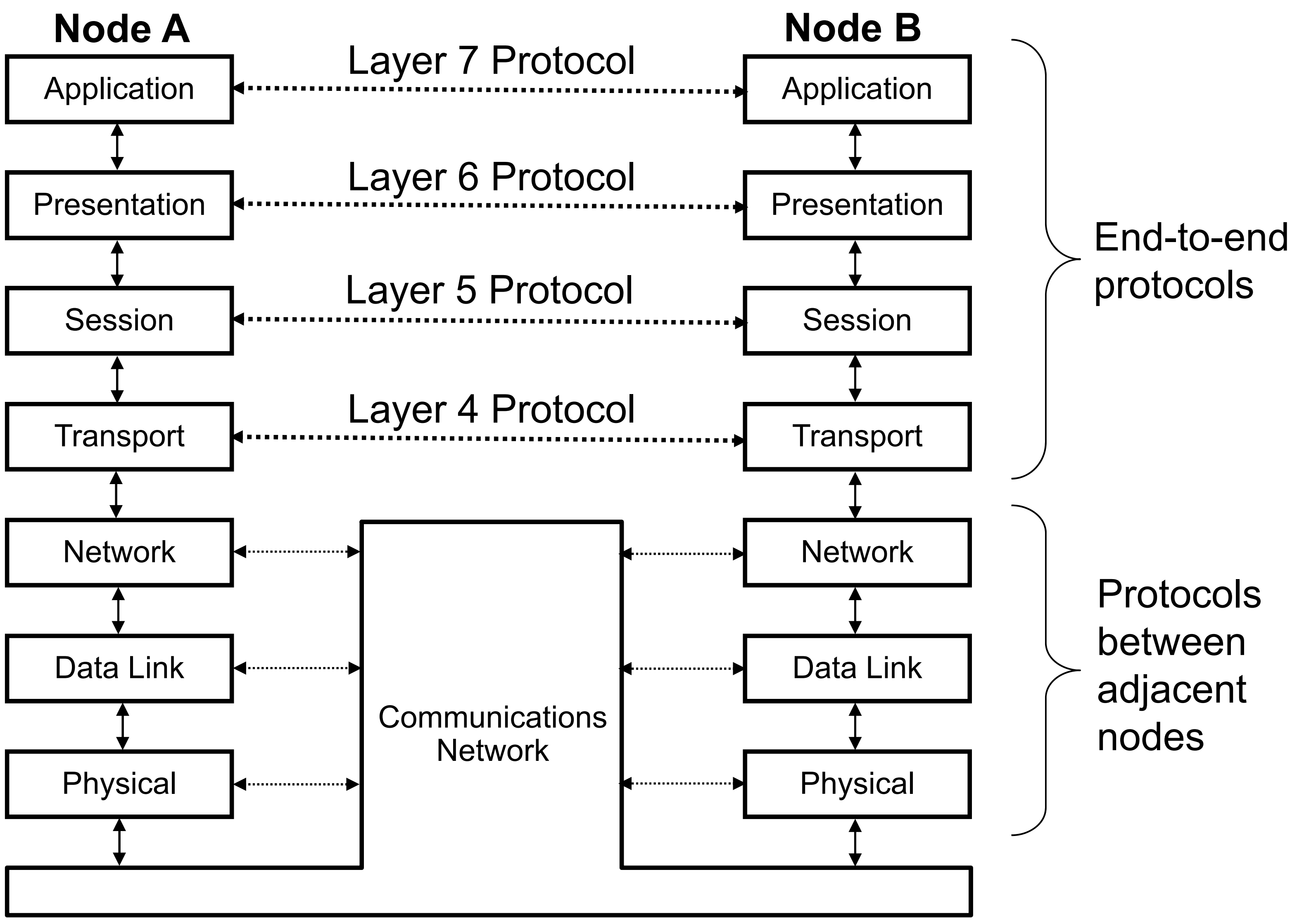
- A header and/or trailer is *added* to the data packet at each layer *going down the protocol stack* in the *transmitting node*.
- The headers and/or trailers are *processed* and *removed* when *going up the protocol stack* in the *receiving node*.

The ISO OSI-RM 7-layer Protocol Stack (1980)

ISO	=	International Standards Organization
OSI	=	Open Systems Interconnection
RM	=	Reference Model

Layer	**Name**	**Typical Services**
7.	*Application*	Generation, processing & consumption of information.
6.	*Presentation*	Data format conversion, compression, encryption.
5.	*Session*	Opening and closing of communication channels.
4.	*Transport*	Control of end-to-end flow of data byte streams.
3.	*Network*	Addressing and routing of packets through network.
2.	*Data Link*	Tx. of data frames from one node to the next node.
1.	*Physical*	Transmission of bits over the communication medium.

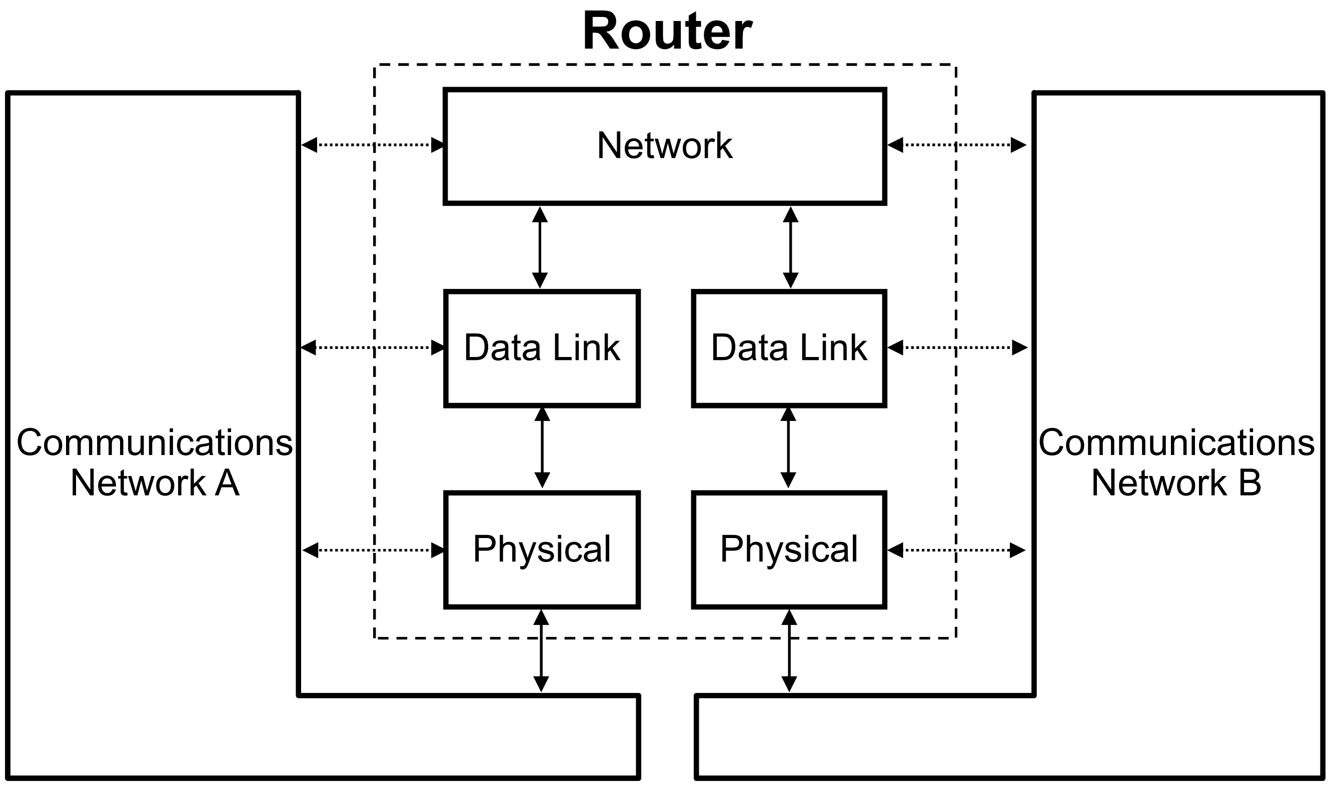
OSI Protocol Interfaces



Routers in the OSI Model

--- A *router* is used to interface two networks that operate using different layer 3 protocols and/or that use different layer 3 packet formats.
e.g. FDDI campus backbone to/from the ETLC building

--- Typical router functions include: (1) forwarding the packet to the next node on its route to the destination; (2) filtering data packets according to destination address; and (3) any required data packet reformatting.

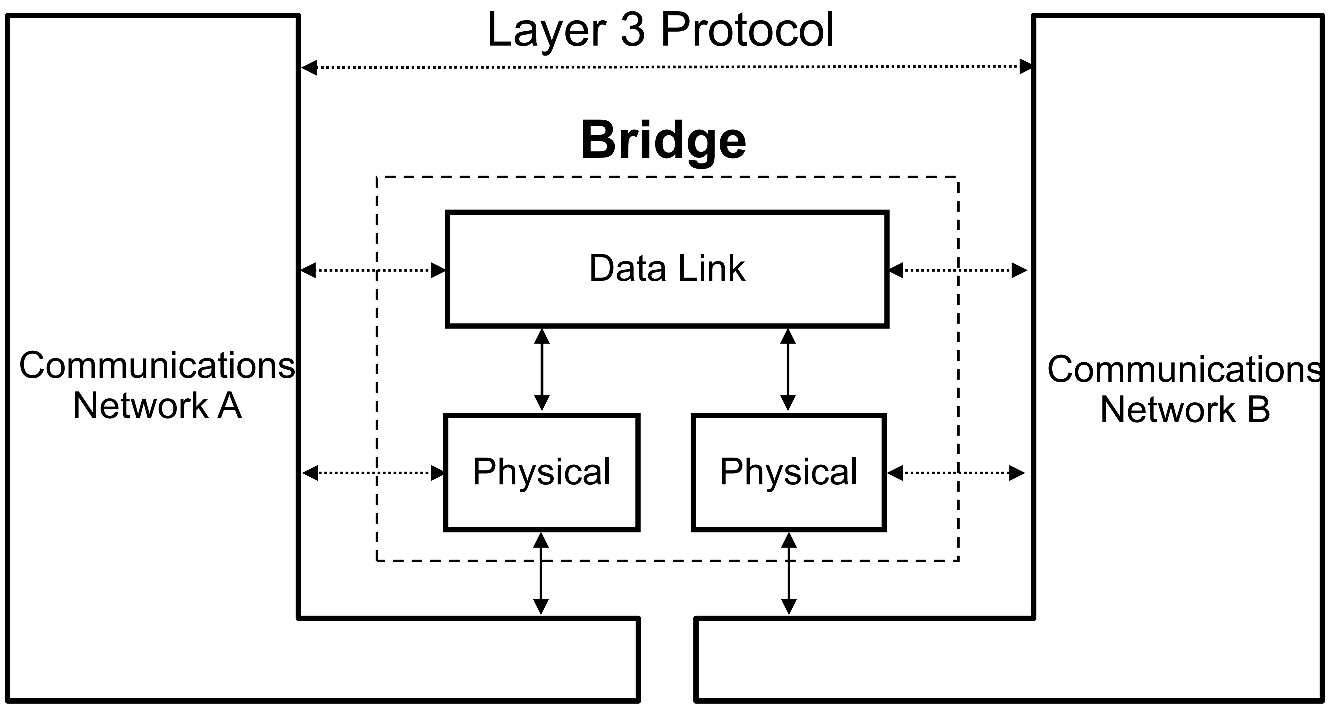


Bridges in the OSI Model

--- A *bridge* is usually used to interface two networks that operate using the same layer 3 protocol.

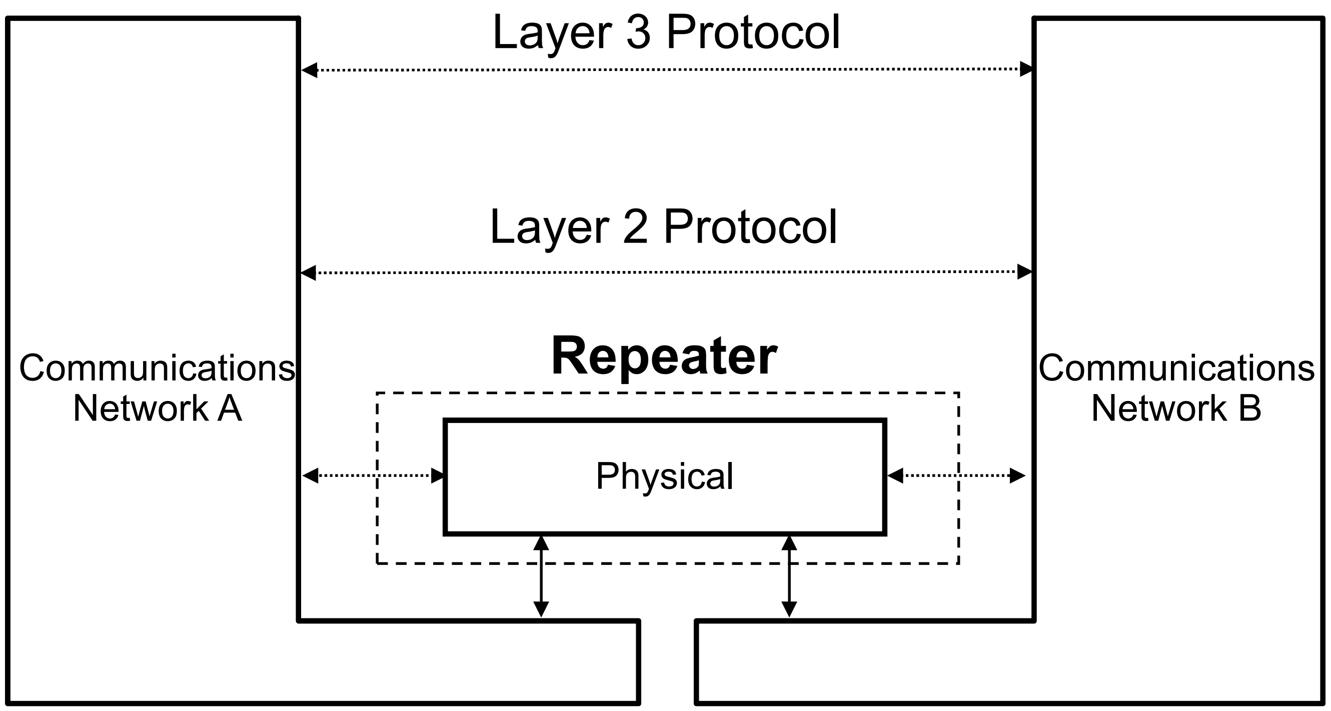
e.g. ETLC 2nd floor Ethernet to/from 5th floor Ethernet

--- A bridge *filters data packets* according to the destination address. The filtering action is useful for preventing traffic in network A from being sent unnecessarily to network B, and vice versa.

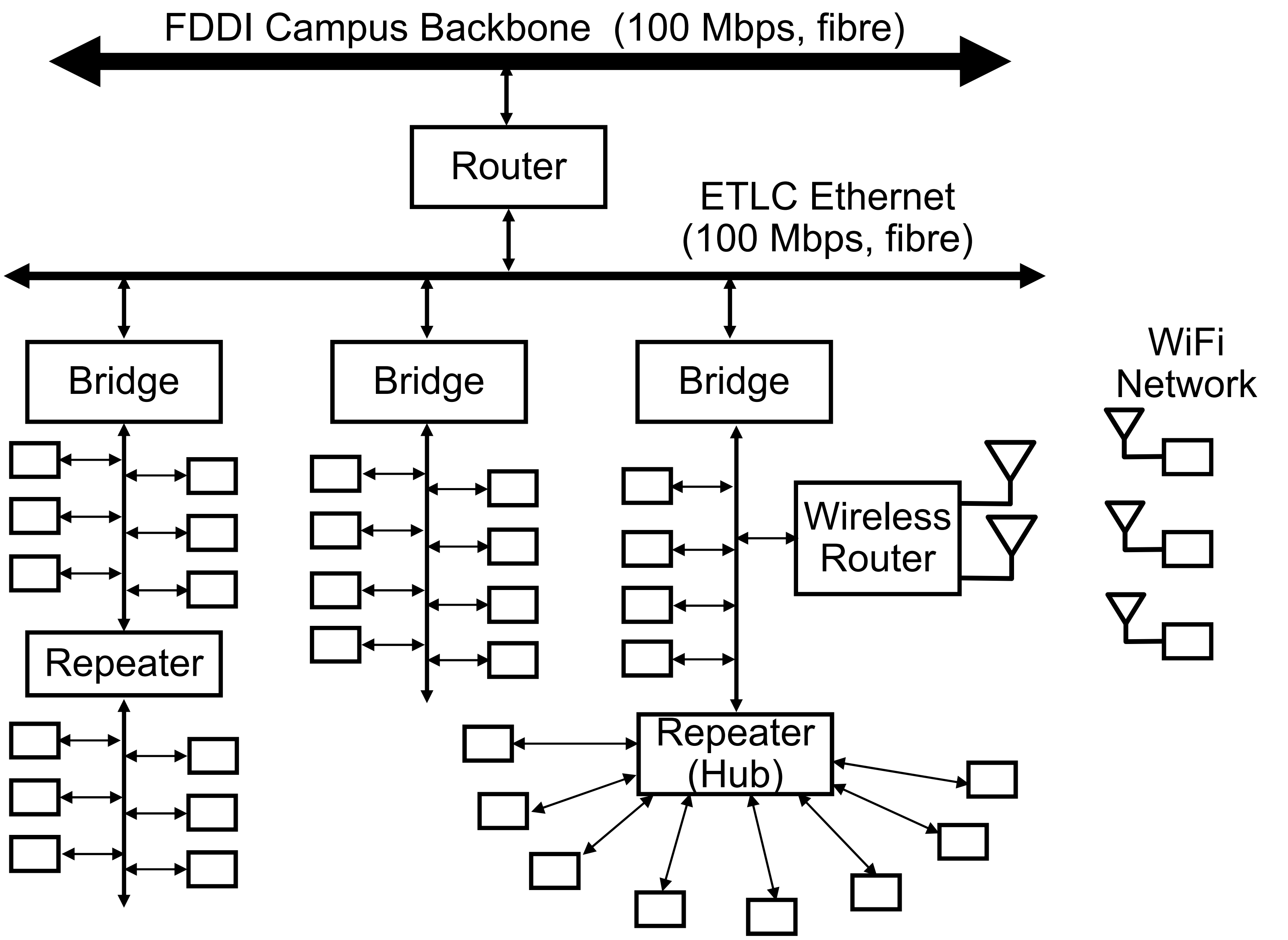


Repeaters & Hubs in the OSI Model

- A *repeater* is used to interface two networks that operate using the same layer 2 and layer 3 protocols.
- A repeater is a “dumb” device that *transfers bits back and forth* between the two networks. A repeater filters out noise and amplifies the signal.
- A *hub* is a repeater that creates a star topology (shown on next slide).



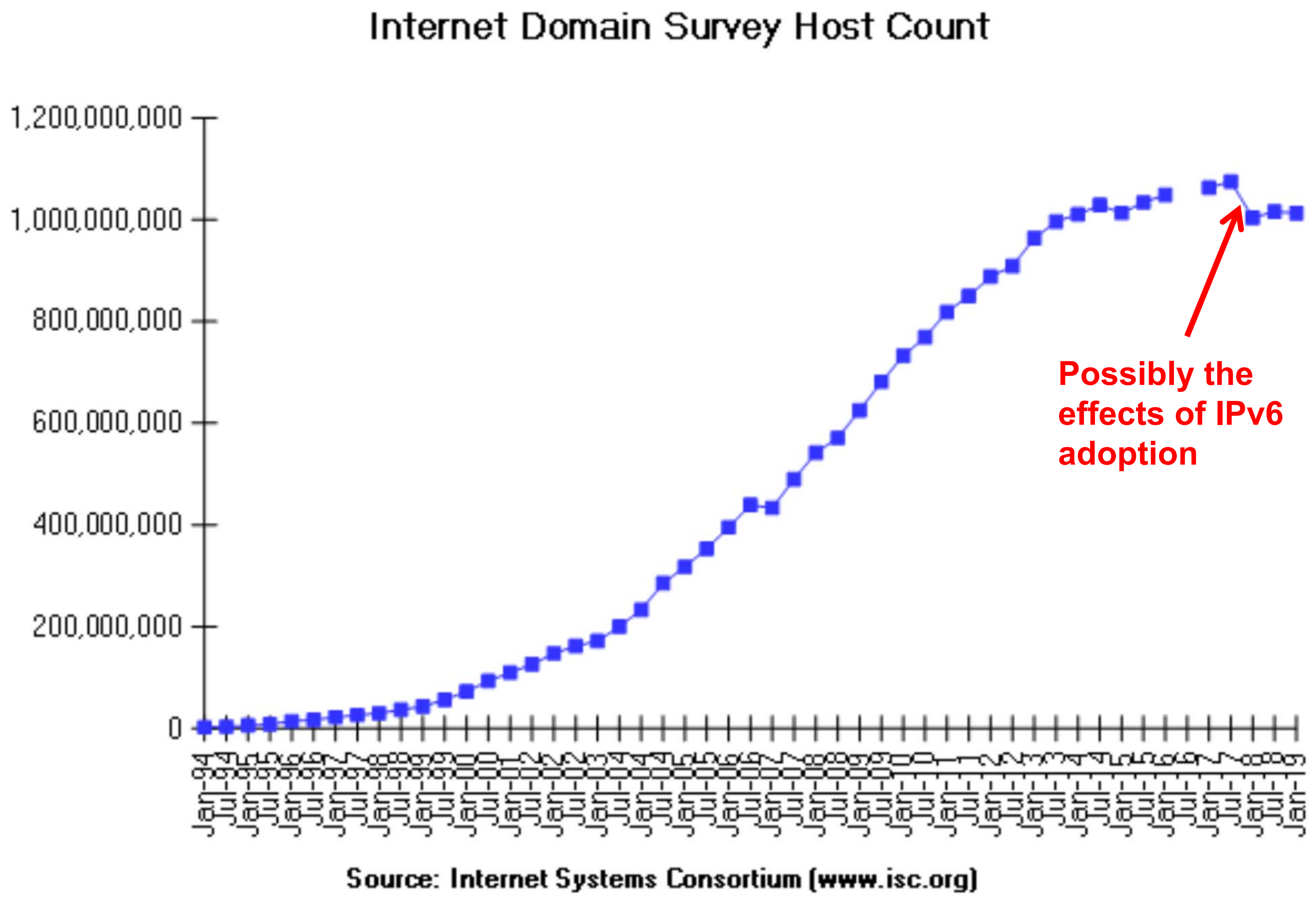
Data Communication Interfaces and TCP/IP



History of Large-Scale Data Networks

1963:	ReserVec airline reservation system (Ferranti Canada) enters service with Trans-Canada Air Lines (now called Air Canada).
1964:	SABRE airline reservation system in service with American Airlines.
1962:	RAND Corporation study recommends deployment of a packet switched data network capable of surviving a nuclear attack.
1969:	U.S. Dept. of Defense Advanced Research Projects Agency (ARPA) sponsors the construction of a large, general-purpose, packet switching data network. ARPANET starts with 4 nodes: UCLA, SRI (Stanford), UC Santa Barbara, & the University of Utah.
1972:	First electronic mail program created by Ray Tomlinson.
1974:	Vincent Cerf and Bob Kahn coin the term Internet in a paper on TCP.
1985:	The ARPANET connects roughly 1200 nodes.
1987:	The civilian Internet connects roughly 25,000 nodes.
1990:	Early version of World-Wide Web developed at CERN in Geneva.
1993:	Marc Andreessen develops Mosaic, the first WWW browser.
2017:	The Internet connects over 1,100,000,000 nodes as of July 2017.

Internet Domain Survey IPv4 Host Count (Jan 2019)



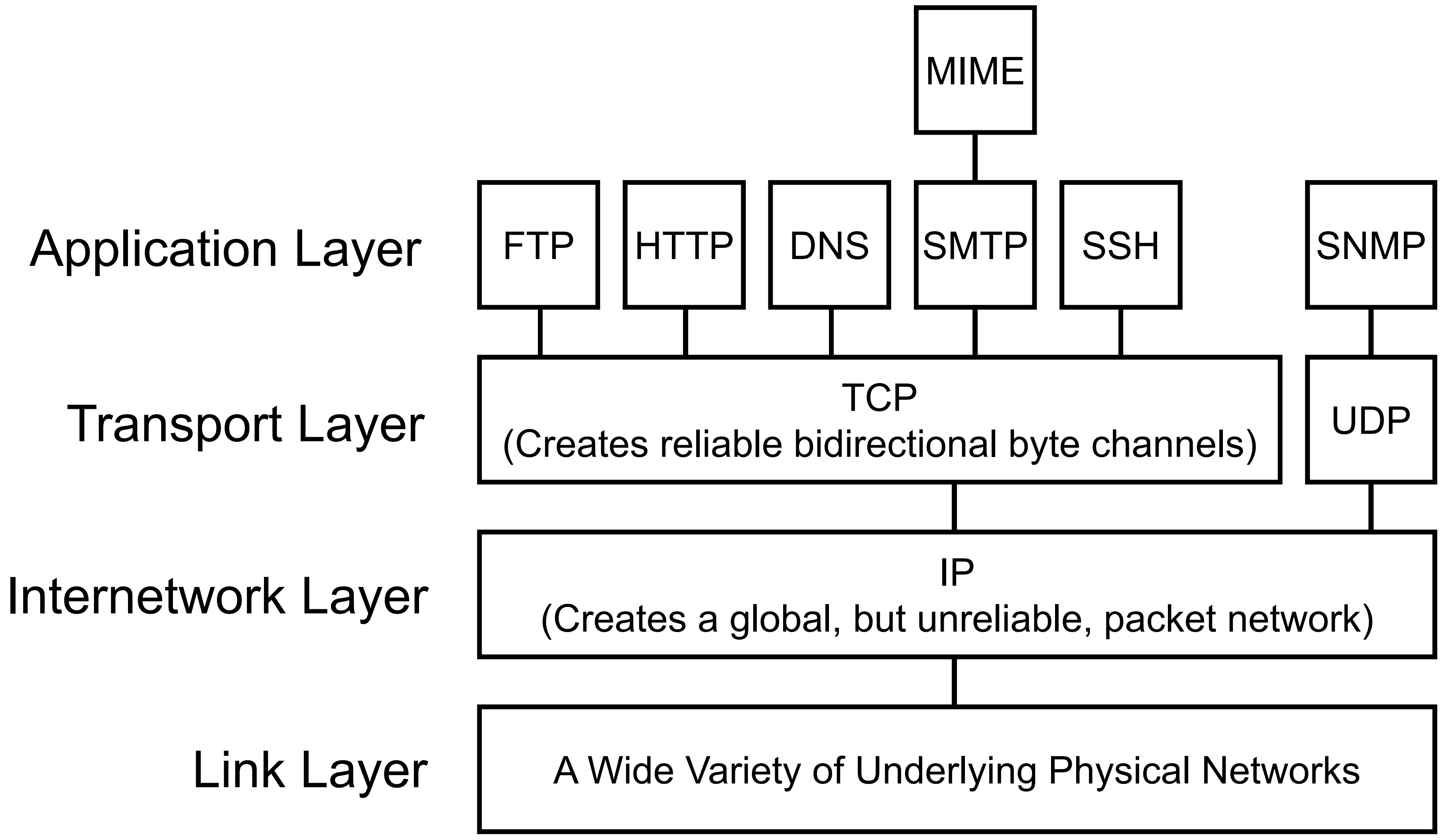
Evolution of TCP/IP

- ***Network Control Protocol*** (NCP): The protocol used in the ARPANET from 1970 until the early 1980s.
- ***Transmission Control Protocol*** (TCP): A successor to NCP first described in 1973 by Vincent Cerf and Bob Kahn.
- TCP/IP: In the late 1970s, TCP was split up into two distinct protocols: A new Transmission Control Protocol (TCP) and the ***Internetwork Protocol*** (IP).
- In 1981, Univ. Calif. Berkeley modified the UNIX operating system to include an implementation of TCP/IP.
- In 1983, TCP/IP became the exclusive protocol on the ARPANET.

The Structure of TCP/IP

- TCP/IP evolved before the 7-layer ISO Reference Model was available, so the layers in TCP/IP do not correspond exactly to the seven standard layers.
- To *maximize flexibility and portability*, the physical and data link layers are left to the underlying network.
- **IP** corresponds closely to the ISO network layer. It provides a ***connectionless datagram service*** between two addressed nodes, with no guarantees on delivery or on the specific route that was taken through the network.
- **TCP** corresponds to the ISO transport layer. It builds on the services of IP to provide a ***reliable, connection-oriented two-way stream of bytes***. The byte order is preserved.
- The functions of the ISO presentation and session layers are provided by the application protocol and TCP.

TCP/IP and Some Related Protocols


MIME
Application Layer
FTP
HTTP
DNS
SMTP
SSH
SNMP
Transport Layer
TCP
(Creates reliable bidirectional byte channels)
UDP
Internetwork Layer
IP
(Creates a global, but unreliable, packet network)
Link Layer
A Wide Variety of Underlying Physical Networks

TCP/IP-Related Protocols and Applications

FTP	=	File Transfer Protocol
HTTP	=	Hypertext Transfer Protocol
DNS	=	Domain Name Service
SMTP	=	Simple Mail Transfer Protocol
MIME	=	Multi-purpose Internet Mail Extension
SSH	=	Secure Shell
UDP	=	User Datagram Protocol
SNMP	=	Simple Network Management Protocol

The details of these, and the many other TCP/IP-related protocols and applications, are beyond the scope of this course. For much more detail see “TCP/IP Protocol Suite” by Behrouz Forouzan (McGraw Hill, 2003).

Services Provided by IP

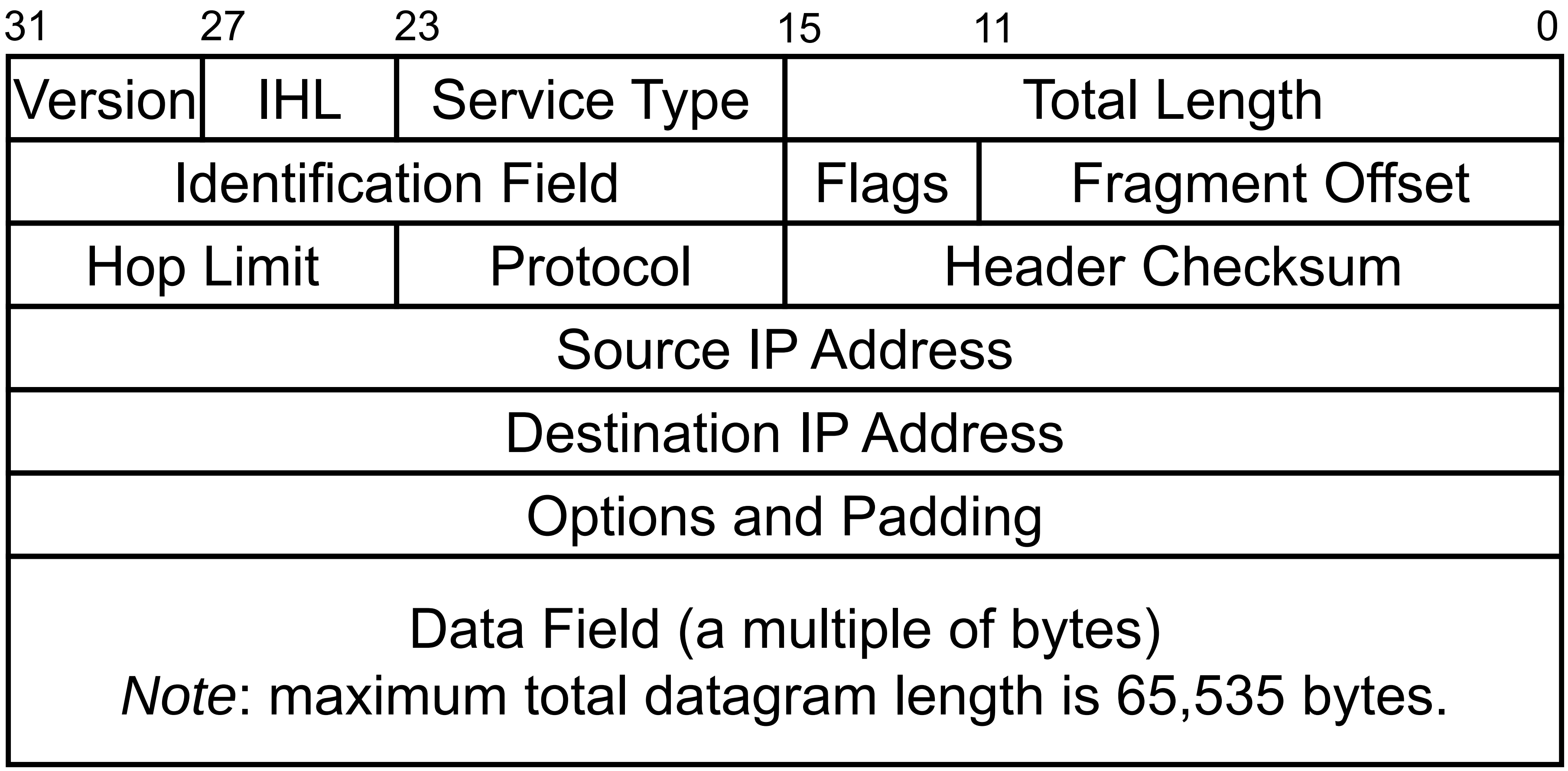
- IP provides two services to the overlying layers:
  1. **Send** a given data packet to a node with a given address.
  2. **Receive** a data packet that was sent to the present node.

  The data packet plus IP header is called a “datagram”.

- IP provides an *unreliable*, *connectionless* delivery service:
  - No warning is given of failure to deliver a datagram.
  - No guarantee that a series of datagrams sent to the same destination will take the same route. The order of such a sequence of datagrams might not be preserved.
  - IP can break up and send a packet as multiple datagrams.

- IP prepends its own header to each datagram to provide information that is required for IP datagram processing at the intervening (tandem) nodes and at the destination node.

IP Version 4 Datagram Format (IPv4)



IHL = Internet Header Length (measured in 32-bit words).
Padding ensures that the header is a multiple of words long.

IP Version 4 Address Formats (IPv4)

Class A: Few networks, each with many hosts

0	Network	Host (24 bits)

Class B: Medium number of hosts and networks

1 0	Network (14 bits)	Host (16 bits)

Class C: Many networks, each with few hosts

1 1 0	Network (21 bits)	Host (8 bits)

Class D: Intended for multicasting to all hosts in defined groups

1 1 1 0	Host Group Identifier (28 bits)

Class E: Reserved addresses

1 1 1 1	Address (28 bits)

More on IPv4 Addresses

- In order to participate in send and receive IP datagrams with a TCP/IP-based Internet, a node needs to have:
  1. A unique IP address e.g. 0x8D0E4818 = 141.14.72.24
  2. A subnet mask e.g. 0xFFFFC000 = 255.255.192.0
  3. The IP address of a "gateway" router
  4. The IP address of a "name server"
- A ***static IP address*** is associated "permanently" with the hardware of the node (e.g., with the 6-byte *Ethernet physical address* of the node).
- A ***dynamic IP address*** is assigned to a node from a pool of temporary addresses for a limited period of time, usually referred to as a *lease*. When the lease expires, the node must request a dynamic IP address once again. In this way, dynamic IP addresses are returned & reused.
- The ***subnet mask*** is a 32-bit mask which, when bitwise ANDed with an IP address, yields the ***base address*** for the block of IP addresses corresponding to the local subnetwork.

  e.g. 141.14.72.24 AND 255.255.192.0 => 141.14.64.0

  0x8D0E4818 AND 0xFFFFC000 => 0x8D0E4000

IP Version 6

- The rapid growth of the Internet, and the rather rigid way that 32-bit IPv4 addresses have been allocated, has caused some high-growth parts of the world (e.g., Asia) to run out of addresses. IPv4 needs to be replaced with a better protocol.
- IP Version 6 (IPv6), first defined in 1996, uses ***128-bit addresses.*** IPv6 is gradually replacing IPv4 (still common).
- IPv6 has new options and it has been designed to be more easily expanded with new functionalities.
- IPv6 has a new mechanism (the ***flow label***) for handling special types of traffic, like real-time audio and video.
- IPv6 has support for ***encryption*** and ***authentication*** options to provide confidentiality and better packet integrity.

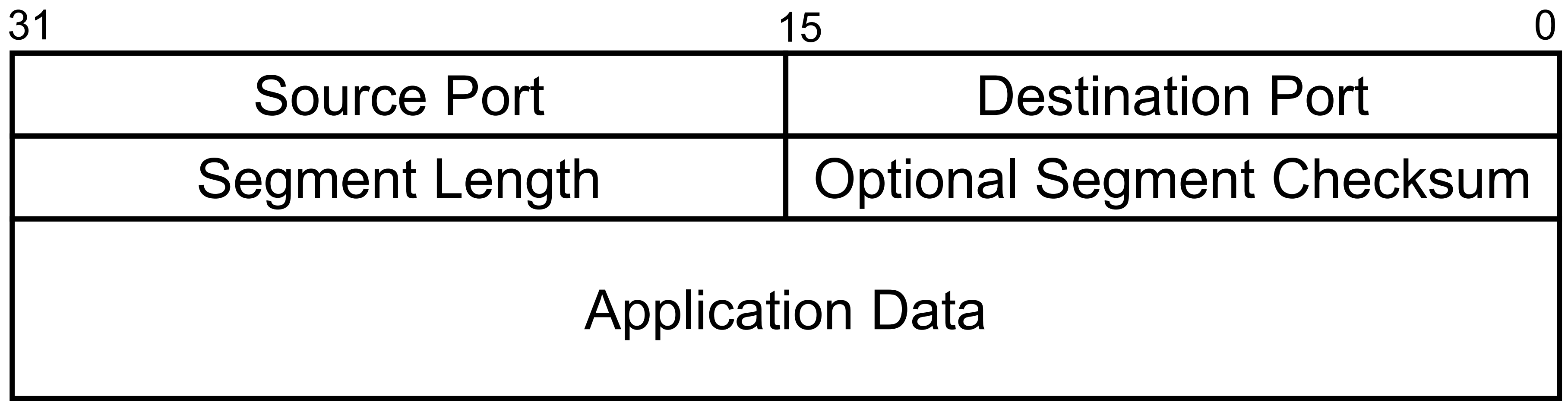
Behrouz A. Forouzan, “TCP/IP Protocol Suite”, 2nd ed., McGraw-Hill, 2003.

Transport Layer Protocols: TCP and UDP

- **User Datagram Protocol (UDP)** provides a ***connectionless*** transport service on top of IP:
  - No guarantee of delivery or of delivery order of the data packets, which in UDP and TCP are called “segments”.
  - IP address is augmented with source and destination port numbers (16 bits each).
  - UDP provides a simpler “lightweight” alternative to TCP which is suitable for “streaming multimedia” services.

- **Transmission Control Protocol (TCP)** provides a reliable, ***connection-oriented*** transport service on top of IP:
  - Like UDP, port numbers are provided.
  - A connection must be set up to carry a communication.
  - A feature-rich set of service primitives is provided to the overlying application layer.

UDP Segment Format



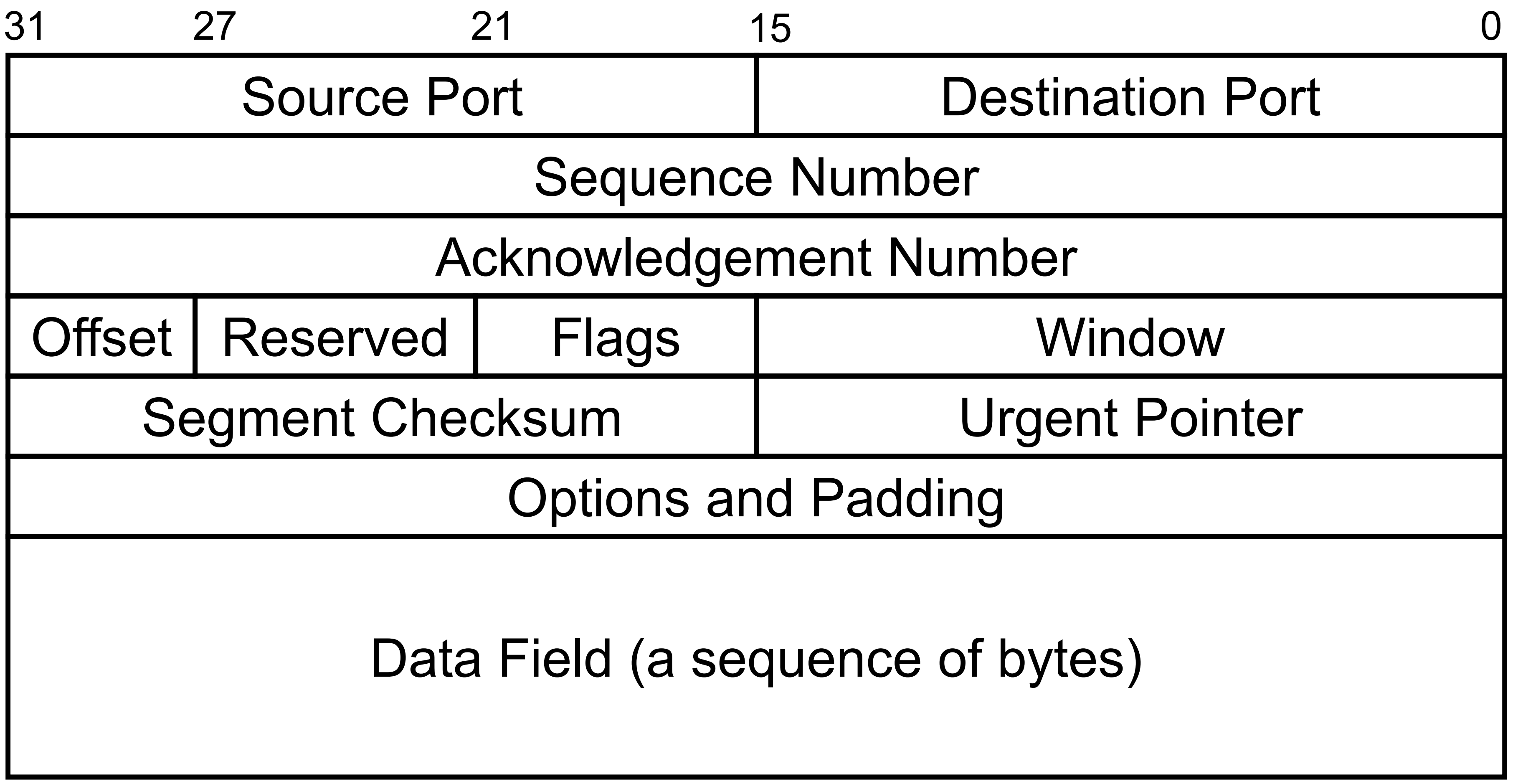
- The ***port numbers*** can be used to identify applications or high-level services on the source and destination nodes.

- If the ***checksum*** is included, then the checksum field is nonzero. If a recomputed checksum does not match the checksum that was sent in the segment, then the entire segment is discarded and no further action is taken.

Some Standard Port Numbers in UDP & TCP

Port	Prot.	Service	Port	Prot.	Service
7	TCP	Echo	70	TCP	Gopher
13	both	Daytime	79	TCP	Finger
19	both	Character gen.	80	TCP	HTTP
20	TCP	FTP data	109	TCP	POP-2
22	TCP	Secure shell	110	TCP	POP-3
23	TCP	Telnet	111	both	RPC
25	TCP	SMTP	161	UDP	SNMP
37	both	Time	162	UDP	SNMP-TRAP
67	UDP	BOOTP-server	179	TCP	BGP
68	UDP	BOOTP-client	520	UDP	RIP
69	UDP	TFTP	>2000		user-defined

TCP Segment Format



TCP Header Fields

- The ***sequence number*** is a number that corresponds to the first data byte. All data bytes have sequence numbers.
- The ***acknowledgement number*** identifies the sequence number of the next data byte that is expected to be received. Contains valid number only used if the ACK bit is set.
- The ***data offset*** gives the number of 32-bit words in the header.
- Six flags are used for various purposes:

URG:	Urgent pointer field enabled (MSB)
ACK:	Acknowledgement field enabled
PSH:	Push function. Force transmission of segment.
RST:	Reset the connection.
SYN:	Synchronize the sequence numbers.
FIN:	No more data will be sent from sender. (LSB)

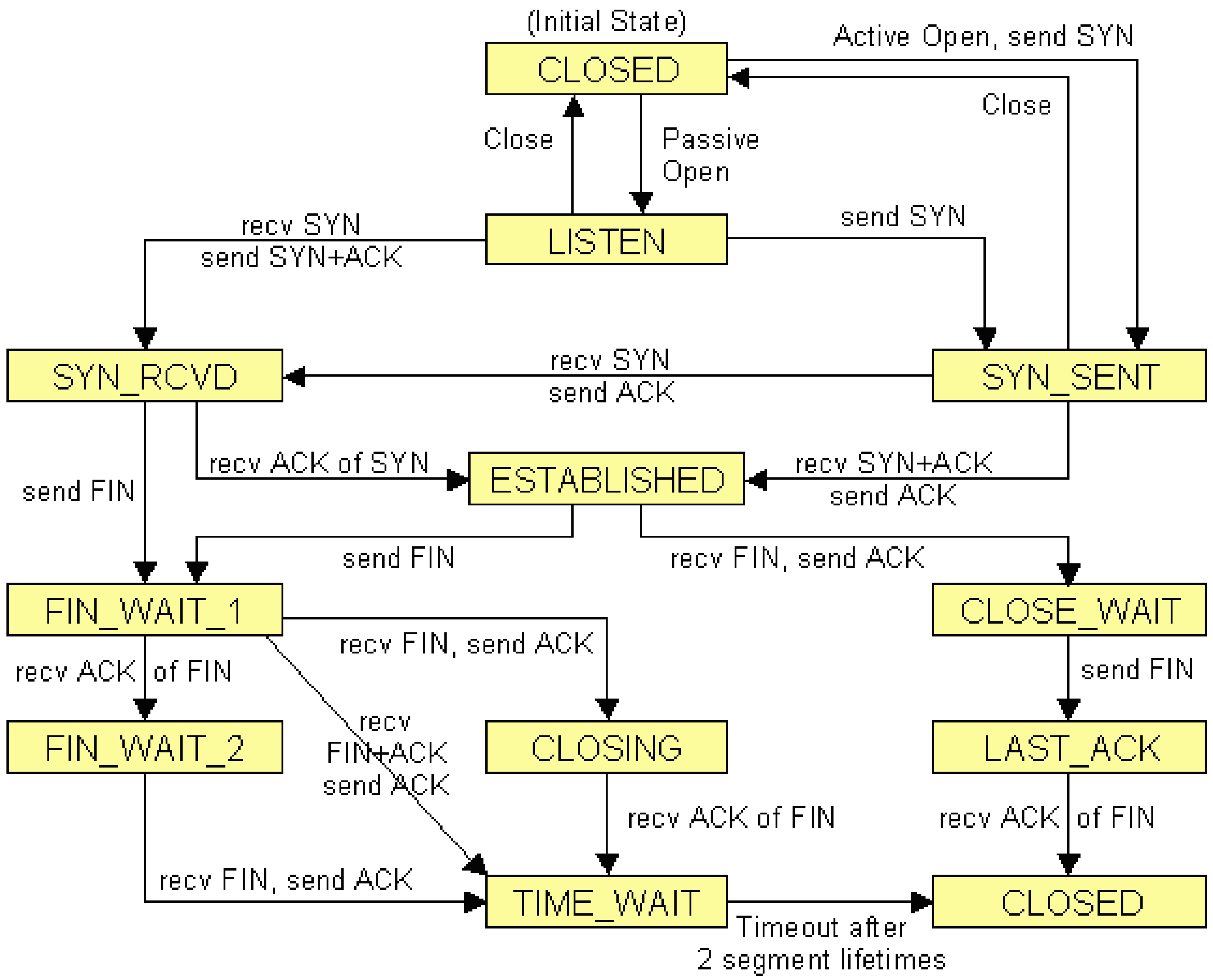
TCP Header Fields (cont'd)

- The ***window*** indicates the maximum number of data bytes that the other TCP entity must be prepared to accept without an acknowledgement. The buffer size in the receiver must be taken into account when negotiating the window.
- The ***checksum*** is computed over the TCP segment and allows the receiving TCP entity to detect transmission errors.
- The ***urgent pointer*** gives the sequence number of the first nonurgent data byte. The urgent bytes are located at the start of the payload for fast access. Only used if URG = 1.
- Only one ***option*** is currently defined in TCP. It specifies the ***maximum segment size***, in bytes, that will be accepted. This size is the largest TCP payload allowed one segment. The value is determined when the connection is established.

The TCP Finite State Machine

- An 11-state ***finite state machine*** (FSM) is used to structure the status and input/output behaviour of the TCP entity.
- The TCP FSM controls: (a) ***connection establishment***, (b) ***data transfer***, and (c) ***connection termination***.
- At any one time within a host, a TCP connection exists in exactly one state in the TCP FSM. The TCP states at the two ends of the same TCP connection can be different.
- ***Transitions are made from state to state*** within the TCP FSM as a result of ***events***, such as receiving commands from the host's software, timer timeouts, and receiving TCP segments from the other end of the connection.
- In addition to causing state transitions, ***events can also cause TCP segments to be transmitted*** to the other end.

The State Diagram of the TCP FSM



Aside: Client-Server Communication

- Many communications between processes on networked nodes can be structured as client-server interactions.

- A ***server process*** resides on a networked node and waits for other processes (clients) to make connections to it and to request services from it.
  Ex: Unix systems will have server processes for each of the standard port numbers to handle incoming connection requests to those port numbers.

- A ***client process*** resides on a networked node and can make connections and send service requests to remote server processes.

- TCP/IP connections are commonly used to support client-server communication over the Internet.

The 11 TCP States

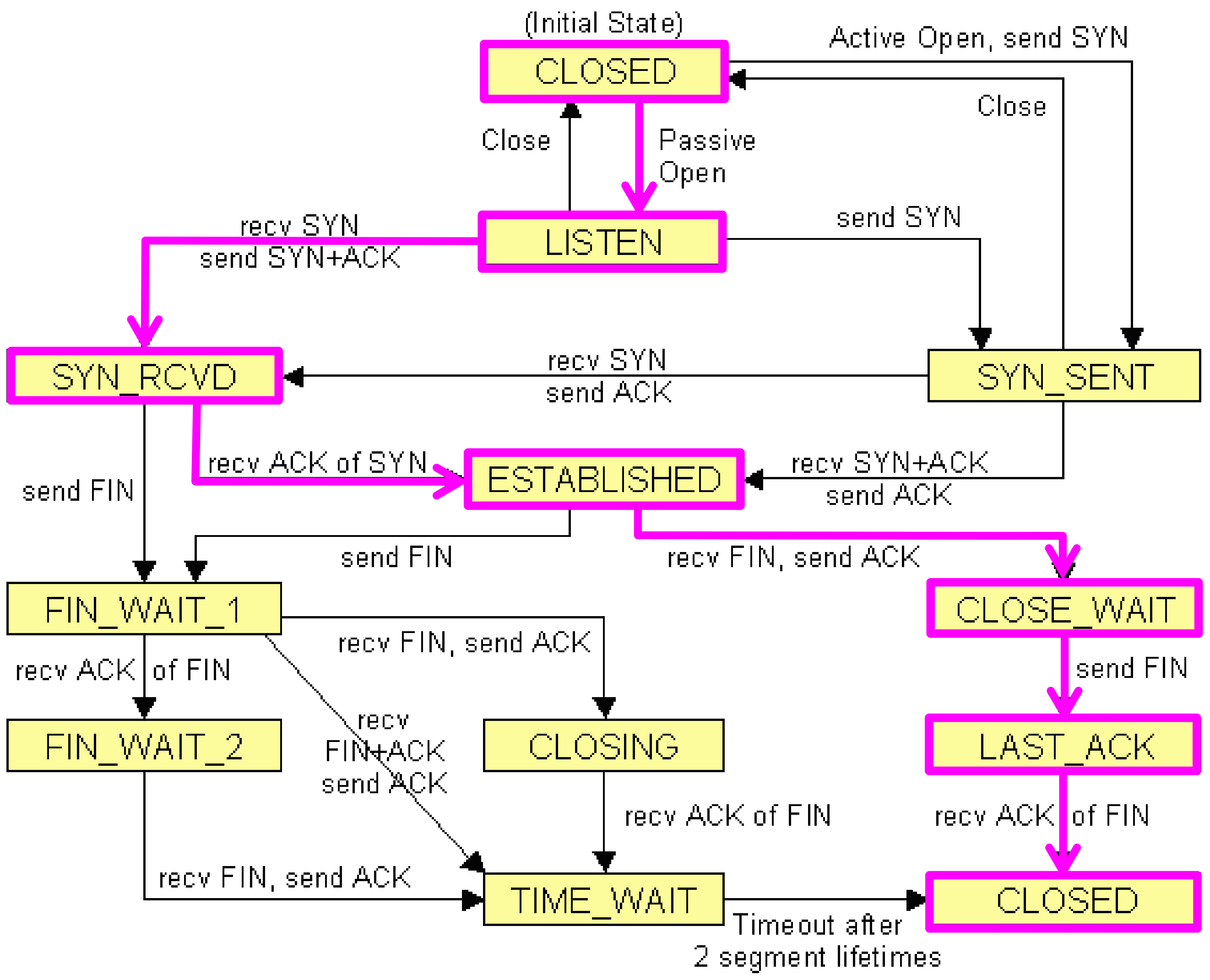
CLOSED	There is no connection.
LISTEN	The server is waiting for a call from a client.
SYN-SENT	Connection request was sent; waiting for ack.
SYN-RCVD	A connection request was received.
ESTABLISHED	A connection is established.
FIN-WAIT-1	The host requests to close the connection.
FIN-WAIT-2	The other side accepts the conn. closing.
CLOSING	Both ends have decided to close the conn.
TIME-WAIT	Waiting for retransmitted segments to die.
CLOSE-WAIT	Server is waiting for the application to close.
LAST-ACK	Server is waiting for the last ack.

Ref: B. A. Forouzan, "TCP/IP Protocol Suite", 2nd ed., McGraw-Hill, 2003., p. 329.

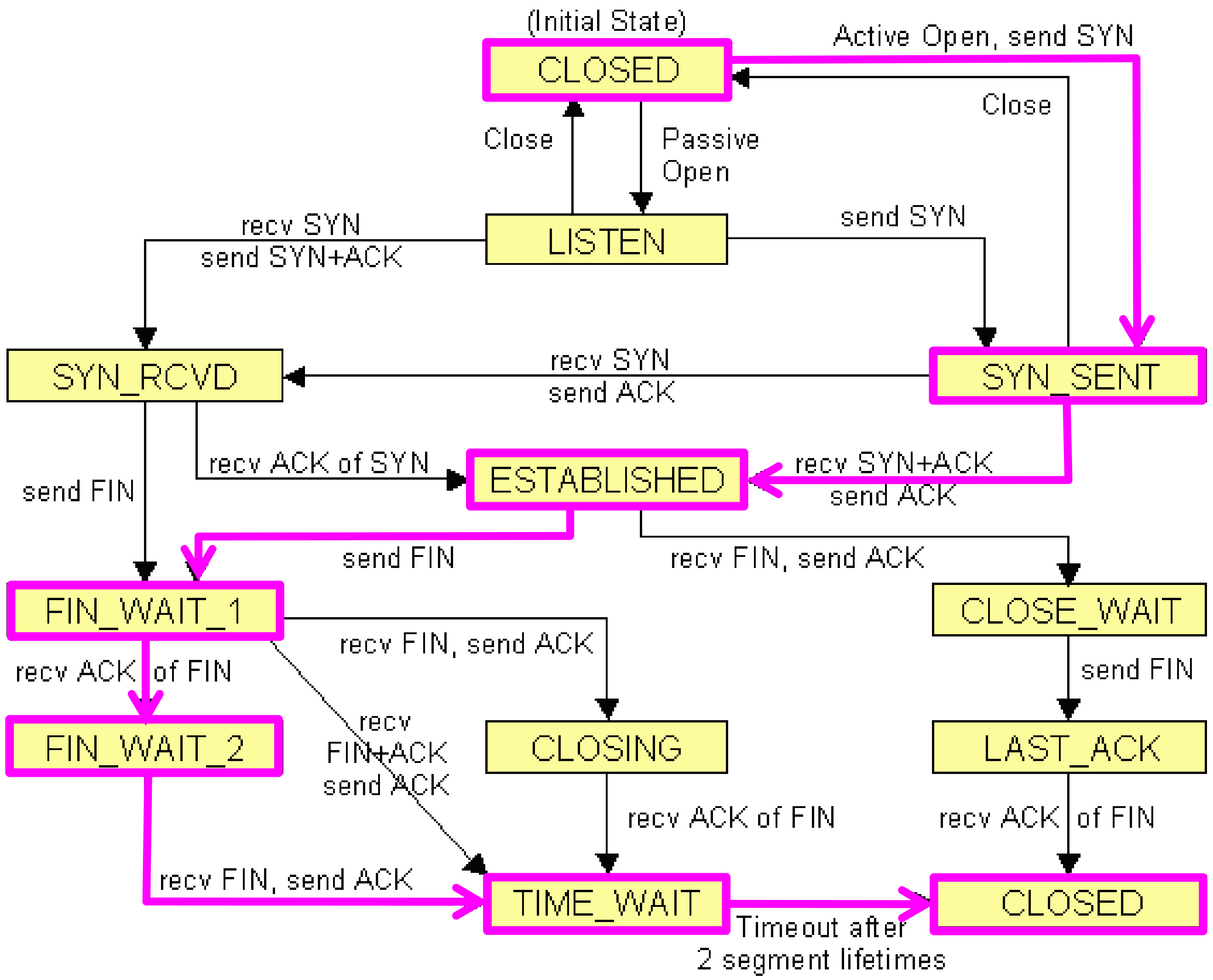
TCP Events Communicated on Segments

- A SYN segment sent by a client has the SYN bit set to 1. The *source and destination port numbers* are specified. The *first sequence number from the client* side is specified so that the server will be synchronized.
- A SYN+ACK segment sent by a server has the SYN and ACK bits set to 1. The *acknowledgement number* is the client sequence number plus 1. The segment also specifies the *first sequence number from the server*, so the client will be synchronized. The *client window size* is specified.
- An ACK segment is sent by the client with the ACK bit set to 1. The *acknowledgement number* is the server sequence number plus 1. The client specifies the *server window size*.
- A FIN segment has the FIN bit set to 1. It is sent by a host to *finish* (take down, terminate) the current connection.
- A RST segment has the RST bit set to 1. It is used by a TCP host to warn the other side that the connection is being *reset*.

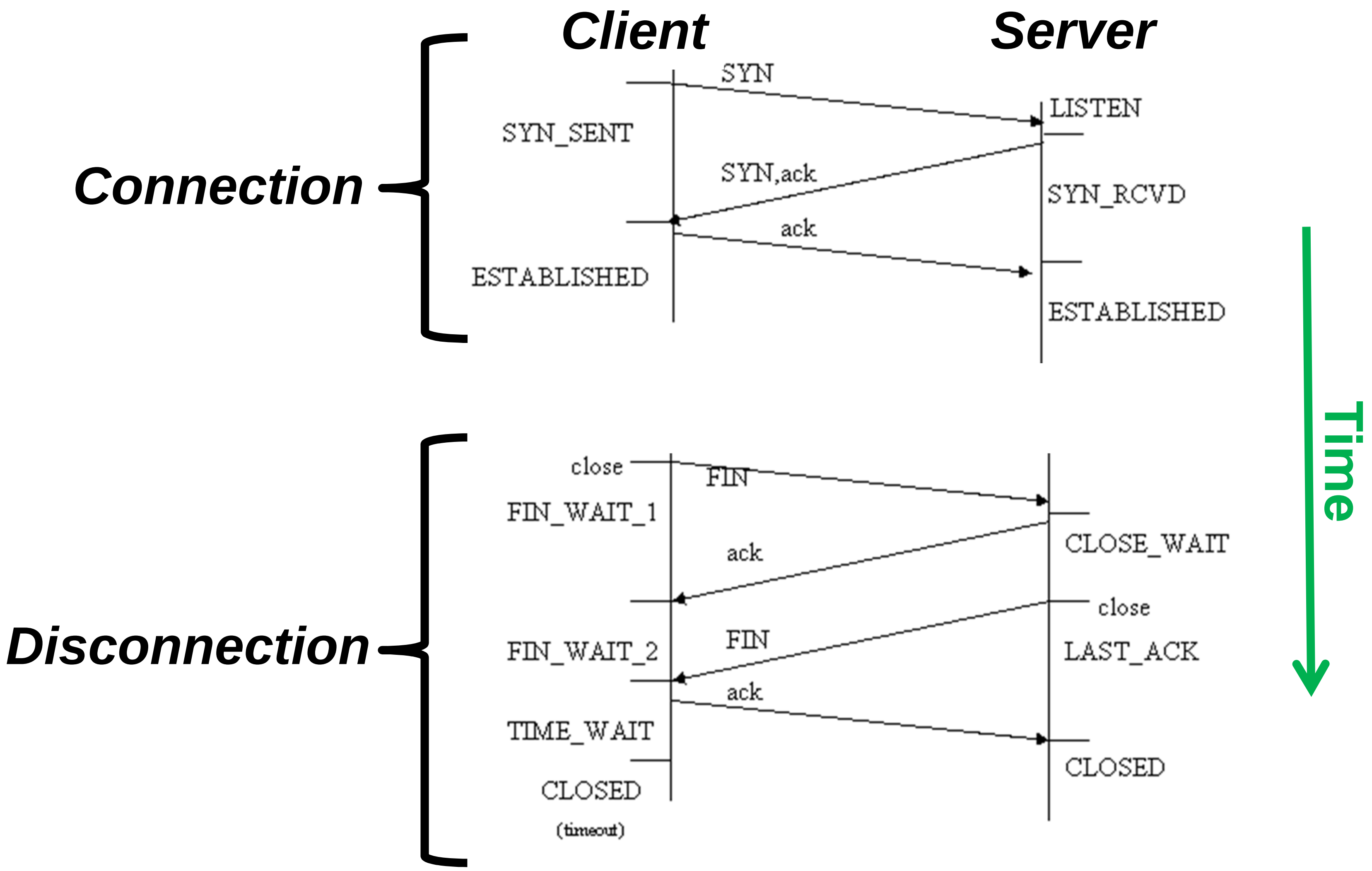
Normal TCP State Sequence in Servers



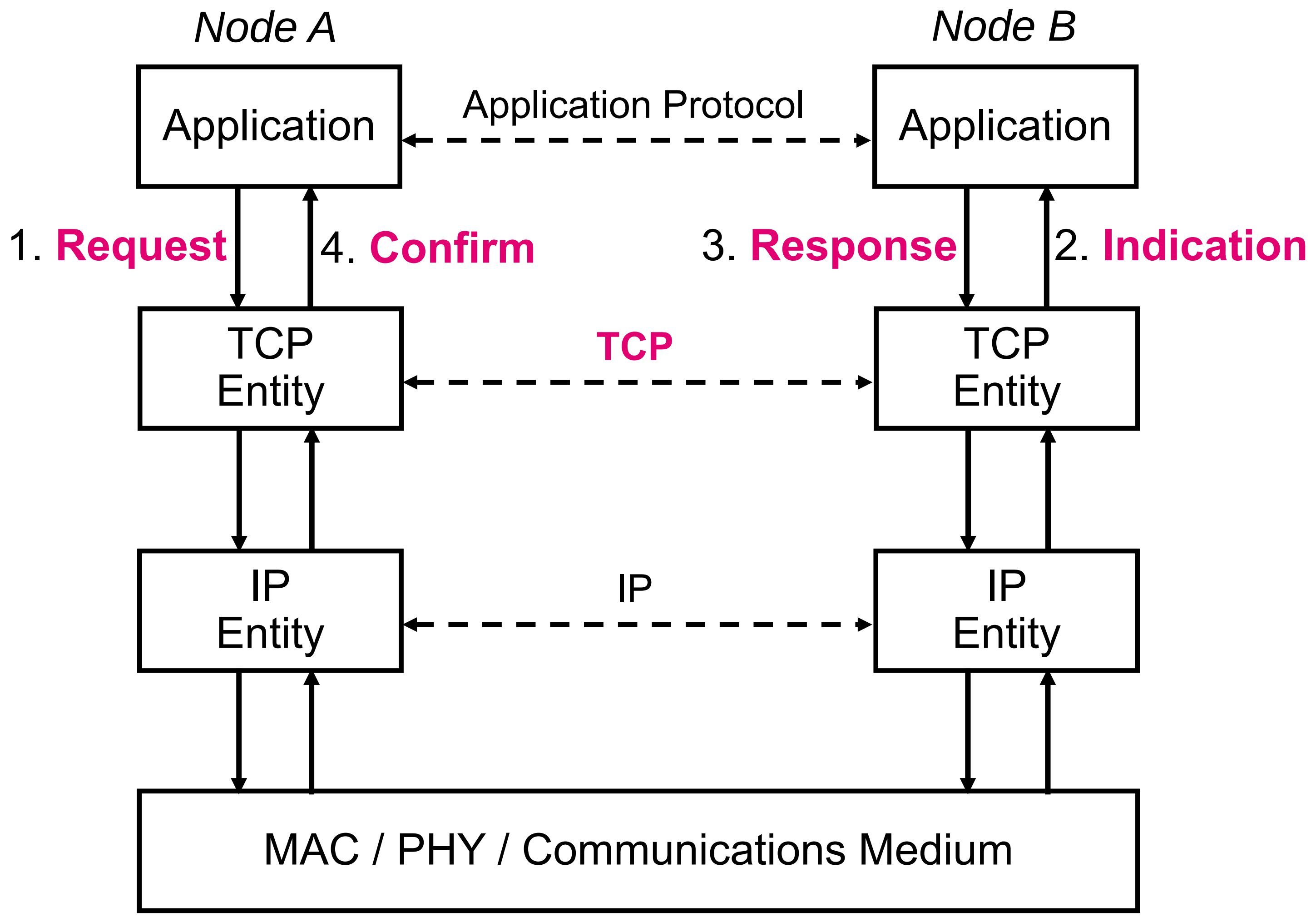
Normal TCP State Sequence in Clients



TCP Connection and Disconnection



Accessing the Services of TCP



TCP Service Request & Response Primitives

1. Listen for connection attempt at specified security and precedence from *any* remote destination.
2. Listen for connection attempt at specified security and precedence from a *specified* remote destination.
3. Request an active connection at a specified security and precedence to a specified destination.
4. Request a connection at a particular security and precedence to a specified destination and transmit data with the request.
5. Transfer data across the named connection.
6. Issue incremental allocation for receive data to TCP.
7. Close the connection gracefully.
8. Close the connection abruptly.
9. Query the connection status.

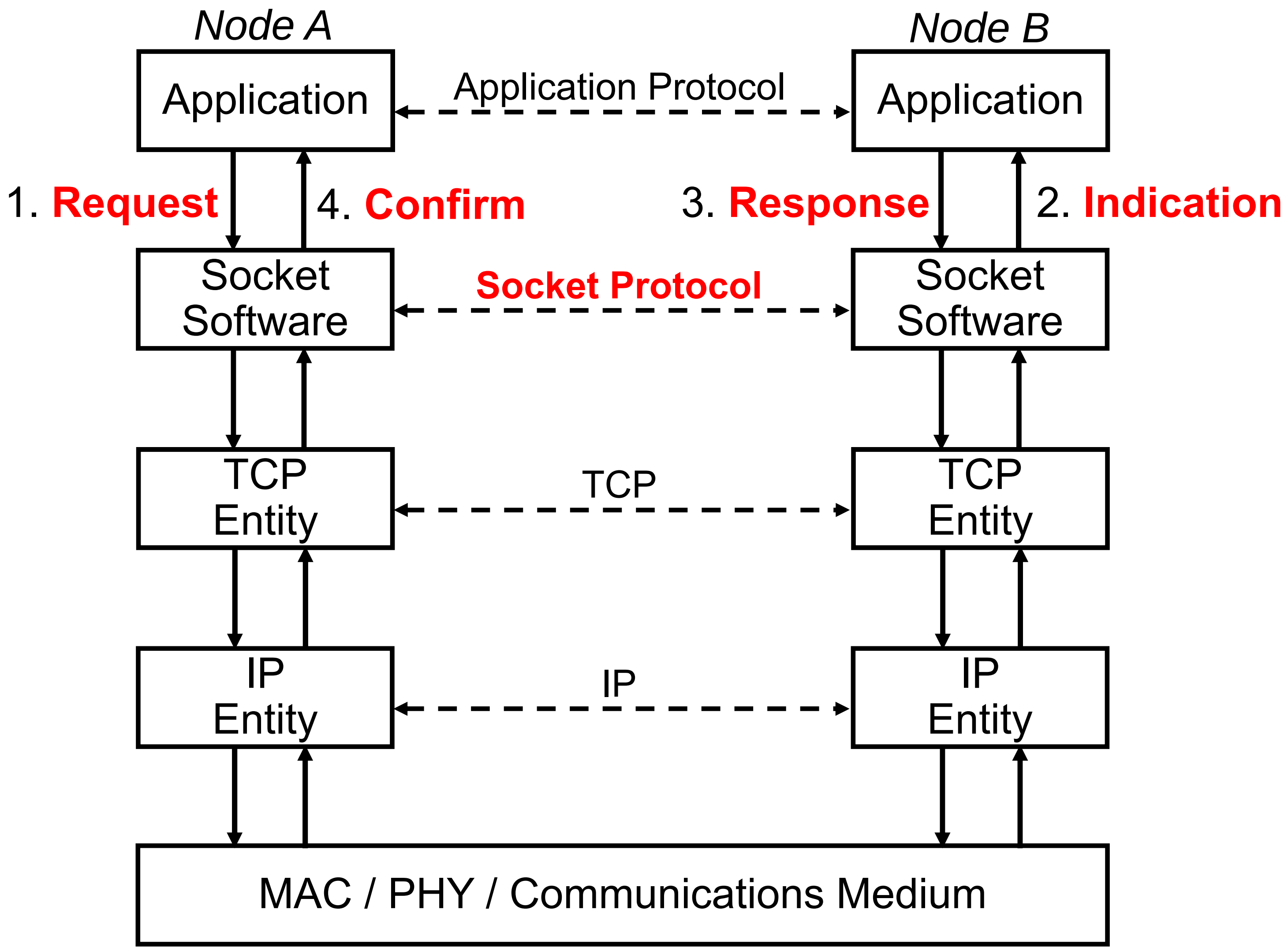
TCP Service Confirm & Indication Primitives

1. Informs TCP user of the connection name that has been assigned to the pending connection that was just requested.
2. Report failure to open an active connection.
3. Report success at obtaining an active connection.
4. Report arrival of data. Data is included.
5. Reports that the remote (peer) TCP user has requested to close the connection gracefully. All remote data has now been sent.
6. Reports that the connection has been terminated. The reason for the termination is included.
7. Reports the current status of the connection.
8. Reports a service request error or an internal error.

Accessing the TCP/IP Services in Unix

- The services of TCP and IP can be accessed directly by implementation-specific function calls or system calls.
- However, because of the complexity of TCP/IP, it is common to provide a simplified ***Application Program Interface*** (API).
- In Unix (including Linux) systems, the API uses software abstractions called "***sockets***", which can be used like files.
- A socket is really a data structure that ties together a TCP connection with a process or task.
- Sockets must exist and be properly initialized at both ends of a TCP connection for communication to proceed.

Introducing a Socket Layer into the Stack



Client-Server Communication with Sockets

- A ***server process*** gets ready for incoming connection requests by creating and initializing a socket using the API.
  1. socket() - create a new socket on local node
  2. bind() - bind a socket to a local port number
  3. listen() - listen for connection requests to socket
  4. accept() - block the process until connection occurs

- A ***client process*** initiates a communication channel by:
  1. socket() - create a new socket on local node
  2. connect() - connect the socket to specified server

- After the connection has been set up, both processes can exchange data by using the following API functions:
  1. send() - send data bytes to the remote process
  2. receive() - receive data bytes from remote process

Example of Client-Server Code on Unix

- Here the server creates and initializes a socket, then waits for a connection request from a client.

- When a connection establishment request is received, the server will: (1) output the next received message to its output stream; (2) send an acknowledgement back to the client; and then (3) terminate itself.

- The client (1) creates and initializes a socket; (2) requests a connection to the server; (3) prompts the user for a test message to send; (4) sends that message to the server; (5) waits for an acknowledgement message from the server; (6) outputs the acknowledgement; and then (7) terminates itself.

Example Server Code

```
/* A simple server in the internet domain using TCP
   The port number is passed as an argument */
#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_addr.s_addr = INADDR_ANY;
     serv_addr.sin_port = htons(portno);
     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;
}
```

Source: Sockets Tutorial, www.cs.rpi.edu/courses/sysprog/socket/sock.html

Example Client Code

```
#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,&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;
}
```

Output from Client-Server Example

```
nyquist >
nyquist > server 2000
```

```
itanium >
itanium > client nyquist 2000
Please enter the message:
```

Output from Client-Server Example

```
nyquist >
nyquist > server 2000
```

```
itanium >
itanium > client nyquist 2000
Please enter the message: How’s the weather in NY?
itanium>
```

Output from Client-Server Example

```
nyquist >
nyquist > server 2000
Here is the message: How’s the weather in NY?
nyquist >
```

```
itanium >
itanium > client nyquist 2000
Please enter the message: How’s the weather in NY?
itanium >
```

Output from Client-Server Example

```
nyquist >
nyquist > server 2000
Here is the message: How’s the weather in NY?
nyquist >
```

```
itanium >
itanium > client nyquist 2000
Please enter the message: How’s the weather in NY?
itanium > I got your message
itanium >
```

BOOTP and DHCP

- ***Bootstrap Protocol*** (BOOTP) is a protocol that is used by a BOOTP server to provide a newly connecting client with (1) its IP address, (2) the subnet mask, (3) the IP address of the gateway, and (4) the IP address of the name server.
- The BOOTP server is accessed using UDP to port 67.
- A static IP address is retrieved from a fixed table using the physical address of the client.
- ***Dynamic Host Configuration Protocol*** (DHCP) is an extension to BOOTP that serves out both static and dynamic IP addresses.
- The DHCP server first attempts to find a static IP address for the physical address. If one is not found, then a dynamic IP address is assigned for a negotiable lease time to the client from a pool of available unused dynamic IP addresses.

File Transfer Protocol (FTP)

- The ***File Transfer Protocol*** (FTP) uses the services of TCP to transfer human-readable text (ASCII or EBCDIC-encoded) or binary files between a client and a server.
- FTP servers listen for control connections to Port 21. Such connections can then allow files to be transferred over separate data connections with the server's Port 20.
- The client-side control and data ports do not have to be 20 or 21, respectively.
- FTP provides a relatively large number, currently 51, of different commands. There are 45 three-digit decimal return codes that are used to respond to FTP commands.
- The major strength of FTP is that it is a widely supported file transfer protocol on the Internet; however, it provides no encryption or protection against snooping.

Frequently Used FTP Commands

USER – enter username for authentication purposes

PASS – enter password for authentication purposes

PORT – specify the client port for the data connection

CWD – change the working directory on the server

MKD – make a directory on the server

RMD – remove a directory from the server

TYPE – set transfer mode to either ASCII or binary

GET / RETR – retrieve (download) a remote file

PUT / STOR – store (upload) a file to the remote server

DELE – delete a file from the server

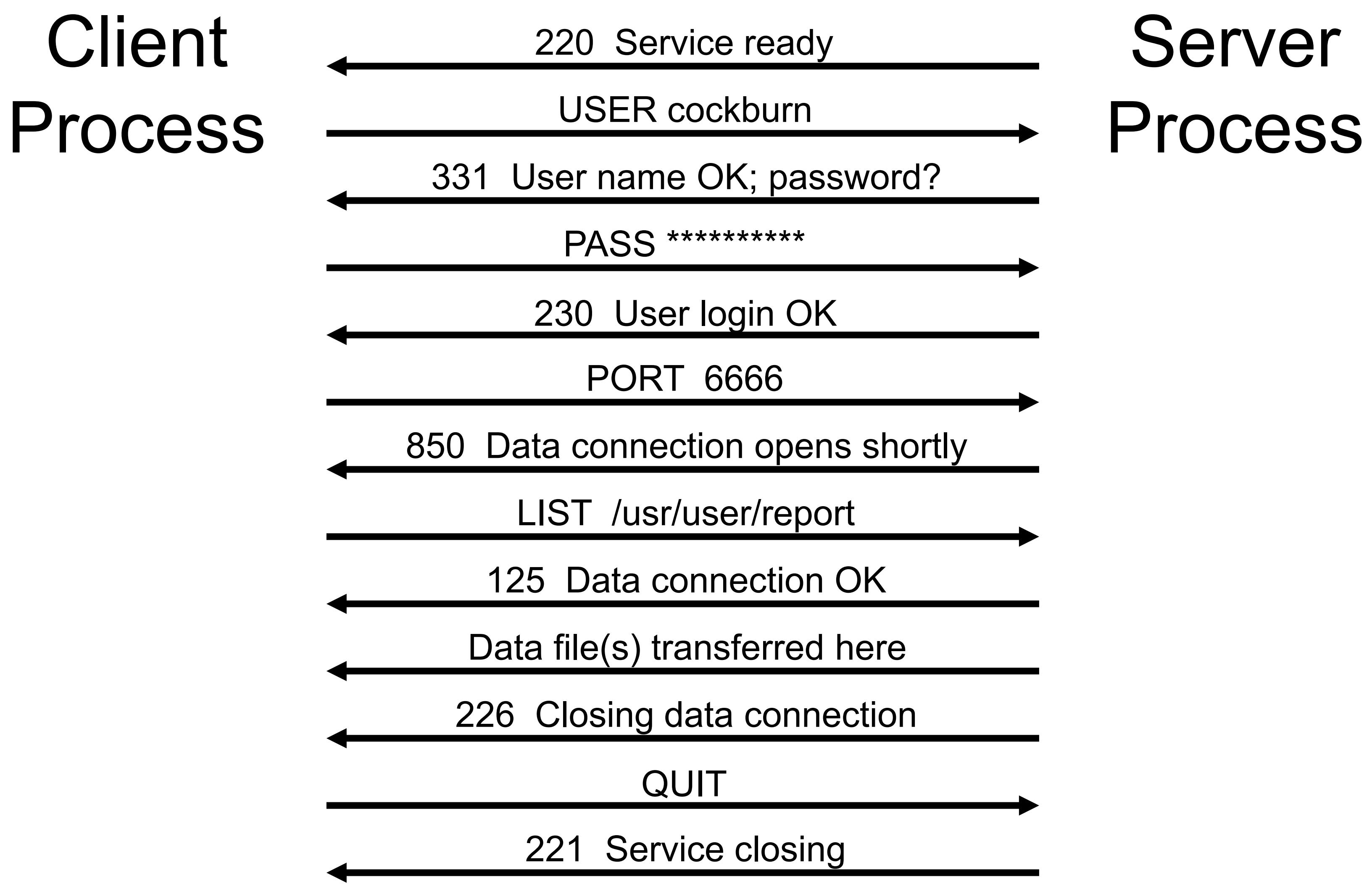
LIST – return information on a specified file or directory

QUIT – terminate the FTP session

Some Common FTP Return Codes

150	File status is OK; data connection will be open shortly
200	Command OK
220	Service ready
221	Service closing
225	Data connection open
226	Closing data connection
230	User login OK
331	User name OK; password is needed
425	Cannot open data connection
426	Connection closed; transfer aborted
550	Action is not done; file unavailable
553	Requested action not taken; file name not allowed

Example: A Brief FTP Session



TELNET

- It is often convenient to be able to log in to a remote server to access software applications as easily as if they were located physically at the site of the server.
- The Internet application TELNET allows one to create a “virtual terminal” to a remote server using a single TCP/IP connection.
- Port 23 is reserved for TELNET on the server side. The client can choose to use another port number.
- TELNET uses an intermediate Network Virtual Terminal (NVT) character set to safely translate characters between the character set of the client and the character set of the server.
- NVT is an extension of ASCII. All 128 7-bit ASCII codes are included, with a fixed “0” bit added as an eighth most significant bit (MSB).
- NVT control codes have been defined, with a fixed “1” as the MSB.
- A special “Interpret as control” code (0xFF) must be sent before each control code to avoid confusion between data and control codes. Binary data 0xFF is sent as one 0xFF followed by a second 0xFF.

Hypertext Transfer Protocol (HTTP)

- The ***Hypertext Transfer Protocol (HTTP)*** is an application layer protocol that is widely used on the TCP/IP-based Internet.
- HTTP is used to exchange information (text, hypertext, images, sounds, video) over the ***World-Wide Web*** (WWW). "Hypertext" is text that is provided with clickable links that simplify the selection and retrieval of related information. A webpage is a document that can contain hypertext.
- The ***Hypertext Markup Language (HTML)*** is widely used to encode the appearance of webpages. HTML is used in the body of an HTTP packet.
- HTTP is a simple request-response protocol:
  - the client sends a request message to the server
  - the server sends a response message to the client
  - a TCP/IP connection is created automatically
- TCP/IP Port 80 on the server is reserved for HTTP.
- At the client machine, a user interacts with HTTP using a browser application (e.g. Internet Explorer, Firefox, Safari, etc.). At the server, HTTP requests are interpreted by a "Webserver" application.

HTTP Request Messages

- Request messages have four parts:
  - Request line = Request type + URL + HTTP version
  - One or more HTTP header lines
  - A blank separator line
  - An optional body (one or more lines)

- A ***Uniform Resource Locator*** (URL) is an Internet file address. The older expression Universal Resource Locator is a synonym.

  The URL format is used by several TCP/IP applications.

  URL = Method **://** HostAddress **:** Port **/** Path
  Method = HTTP, FTP, TELNET, Gopher, News, etc.
  HostAddress = IP address or symbolic address of the host
  Port = TCP port number at the destination (e.g. 80 for HTTP)
  Path = Pathname on the destination host to the desired file

HTTP Request Line

- Request Line = Request type + URL + HTTP version

  Ex: GET http://www.cern.org:80/usr/bin/image1 HTTP/1.1

  Ex: GET http://www.cern.org/usr/bin/image1 HTTP/1.1

  Ex: GET /usr/bin/image1 HTTP/1.1

  Ex: GET http://www.ece.ualberta.ca/~cmpe401/   HTTP/1.1

- Some of the other possible request types/methods in HTTP:

POST	Send input data to server.
PUT	Send a new file to server.
OPTIONS	Obtain a list of the options of the destination node.
DELETE	Remove the specified file from the server.
TRACE	Perform msg. loopback test to the destination node.

HTTP Header Lines

- HTTP request and response messages can contain header lines.
- Header lines communicate additional information, such as formatting, data handling capabilities, software version numbers, date + time, etc.
- Header line format = Name : (space) header-value
- ***General headers*** (used in both request and response messages)
  Ex: Date, MIME-version, Cache-control, Connection, Upgrade
- ***Request headers*** (used only in request messages)
  Ex: Accept, Accept-charset, Accept-encoding, Accept-language
  Authorization, If-modified-since, User-agent, etc.
- ***Response headers*** (used only in response messages)
  Ex: Accept-range, Age, Public, Retry-after, Server
- ***Body Headers*** (used in messages that contain bodies)
  Ex: Content-encoding, Content-length, Expires, Last-modified, etc.

HTTP Response Messages

- Response messages have four parts:
  - Status line
  - One or more HTTP header lines
  - A blank separator line
  - An optional body (one or more lines)
- Status line = HTTP version + status code + status phrase
- Some typical status codes and phrases (and meanings):

200	OK	(The request was successful.)
302	Moved permanently	(The requested URL has been removed.)
400	Bad request	(There is a syntax error in the request.)
403	Forbidden	(The requested service is denied.)
404	Not found	(The file was not found.)
503	Service unavailable	(The service is temporarily unavailable.)

HTTP Example #1

Request message sent from client to server:

```
GET /usr/bin/image1 HTTP/1.1
Accept: image/gif
Accept: image/jpeg
```

Response message sent from server to client:

```
HTTP/1.1  200  OK
Date: Mon, 21-Oct-04 13:37:14 GMT
Server: Nyquist
MIME-version: 1.0
Content-length: 1024

(Body of the document)
```

HTTP Example #2

Request message sent from client to server:

```
HEAD /usr/homepage.html HTTP/1.1
Accept: */*
```

Response message sent from server to client:

```
HTTP/1.1  200  OK
Date: Tue, 09-Oct-04 15:12:10 GMT
Server: Nyquist
MIME-version: 1.0
Content-type: text/html
Content-length: 2048
```

HTTP Example #3

Request message sent from client to server:

```
POST /cgi-bin/doc.pl HTTP/1.1
Accept: image/gif
Accept: image/jpeg
Content-length: 50

(Input information)
```

Response message sent from server to client:

```
HTTP/1.1  200  OK
Date: Fri, 22-Oct-04 18:32:14 GMT
Server: Nyquist
MIME-version: 1.0
Content-length: 1000

(Body of the document)
```

Dynamic Webpages

- A webpage whose content does not change often can be encoded as an occasionally updated ***static HTML*** file.
- However, there are many situations where it is desirable for the server and/or the client to have the ability to change the content and/or the appearance of a webpage.
- At the server, an HTML file can be modified (e.g., ASCII edits) at the time that it is sent back in response to a client request. However, this is a rather inflexible approach.
- ***Server-side scripting languages*** (e.g., ASP, JSP, PHP, ColdFusion, Perl, etc.) are now commonly used to provide dynamic webpages on the server.
- ***Client-side scripting languages*** (e.g., JavaScript, Flash) are used to provide dynamic effects in webpages that are displayed by the browser at the client.