Unit V – Transport Layer

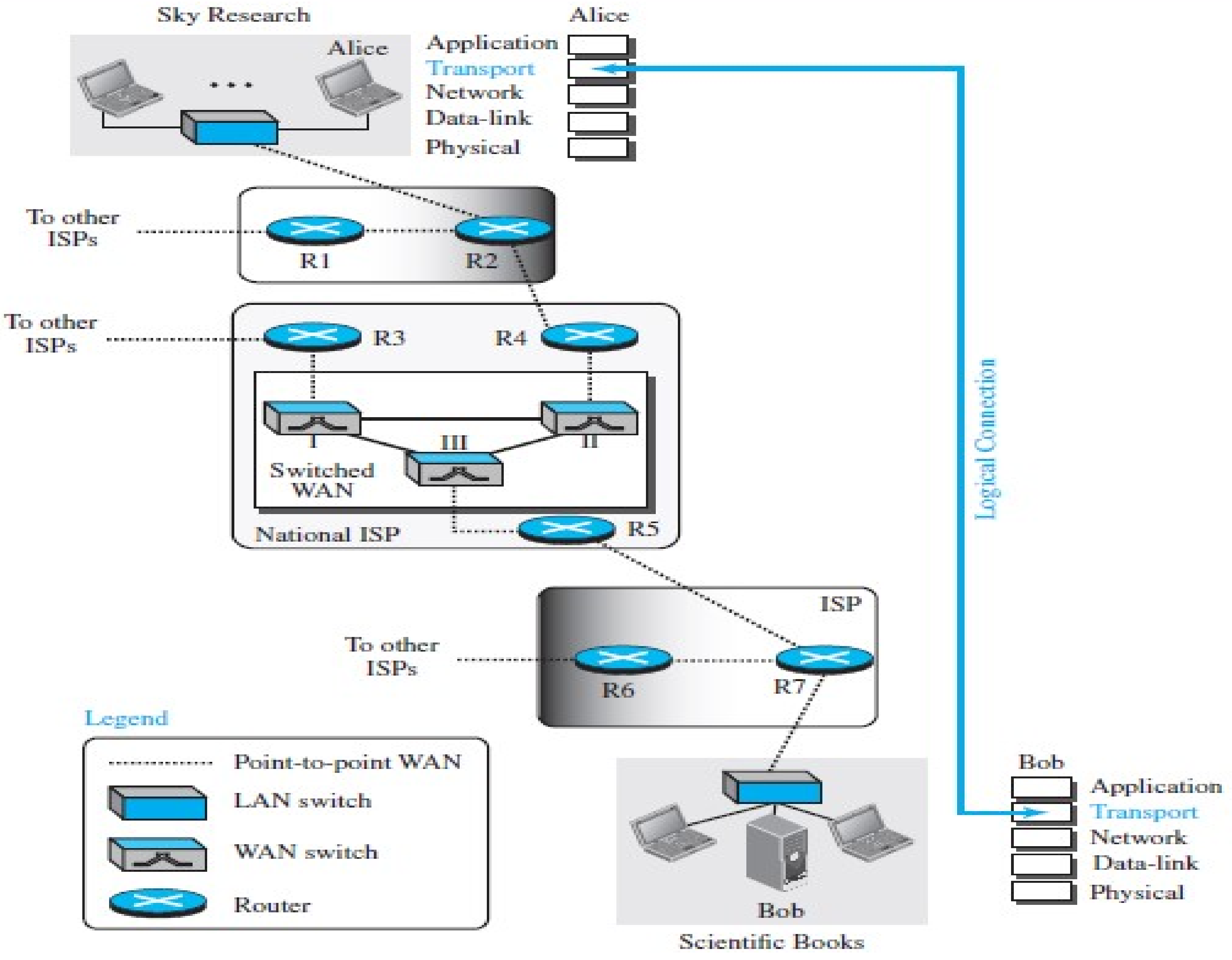
Agenda

- Process-To-Process Delivery: Client/Server Paradigm
- Multiplexing and Demultiplexing
- Connectionless Versus Connection-Oriented Service
- Reliable Versus Unreliable User Datagram Protocol (UDP): Well-Known Ports for UDP
- User Datagram Checksum
- UDP Operation
- Use of UDP TCP Services
- TCP Features, Segment
- Segment Header Format
- A TCP Connection
- Flow Control, Error Control
- Congestion Control.

Process-to-Process Delivery

- The transport layer is located between the application layer and the network layer. It provides a process-to-process communication between two application layers, one at the local host and the other at the remote host.
- Communication is provided using a logical connection, which means that the two application layers, which can be located in different parts of the globe, assume that there is an imaginary direct connection through which they can send and receive messages.
- Figure 23.1 shows the idea behind this logical connection.

Figure 23.1 *Logical connection at the transport layer*



Process-to-Process Delivery

- Alice’s host in the Sky Research company creates a logical connection with Bob’s host in the Scientific Books company at the transport layer.
- The two companies communicate at the transport layer as though there is a real connection between them.
- Figure 23.1 shows that only the two end systems (Alice’s and Bob’s computers) use the services of the transport layer; all intermediate routers use only the first three layers.

Process-to-Process Delivery

- Data link layer is responsible for node-to-node delivery of frames
- Network layer is responsible for host-to-host delivery of datagrams
- Real communication takes place between two processes
- Several processes may be running on source host and destination host
- Transport layer is responsible for process-to-process delivery of packets

Client/Server Paradigm

- The most common way to achieve process-to-process communication is through the client/server paradigm.
- A process on the local host, called a client, needs services from a process usually on the remote host, called a server.
- Both processes (client and server) have the same name.
- For example, to get the day and time from a remote machine, we need a Daytime client process running on the local host and a Daytime server process running on a remote machine.

Client/Server Paradigm

- Operating systems today support both multiuser and multiprogramming environments. A remote computer can run several server programs at the same time, just as local computers can run one or more client programs at the same time.
- For communication, we must define the following:
  - 1. Local host
  - 2. Local process
  - 3. Remote host
  - 4. Remote process

Addressing

- Data Link Layer – MAC address
- Network Layer – IP address
- Transport Layer – port number
- Every processes running is assigned port number
- Destination port number is used for delivery and source port number for reply

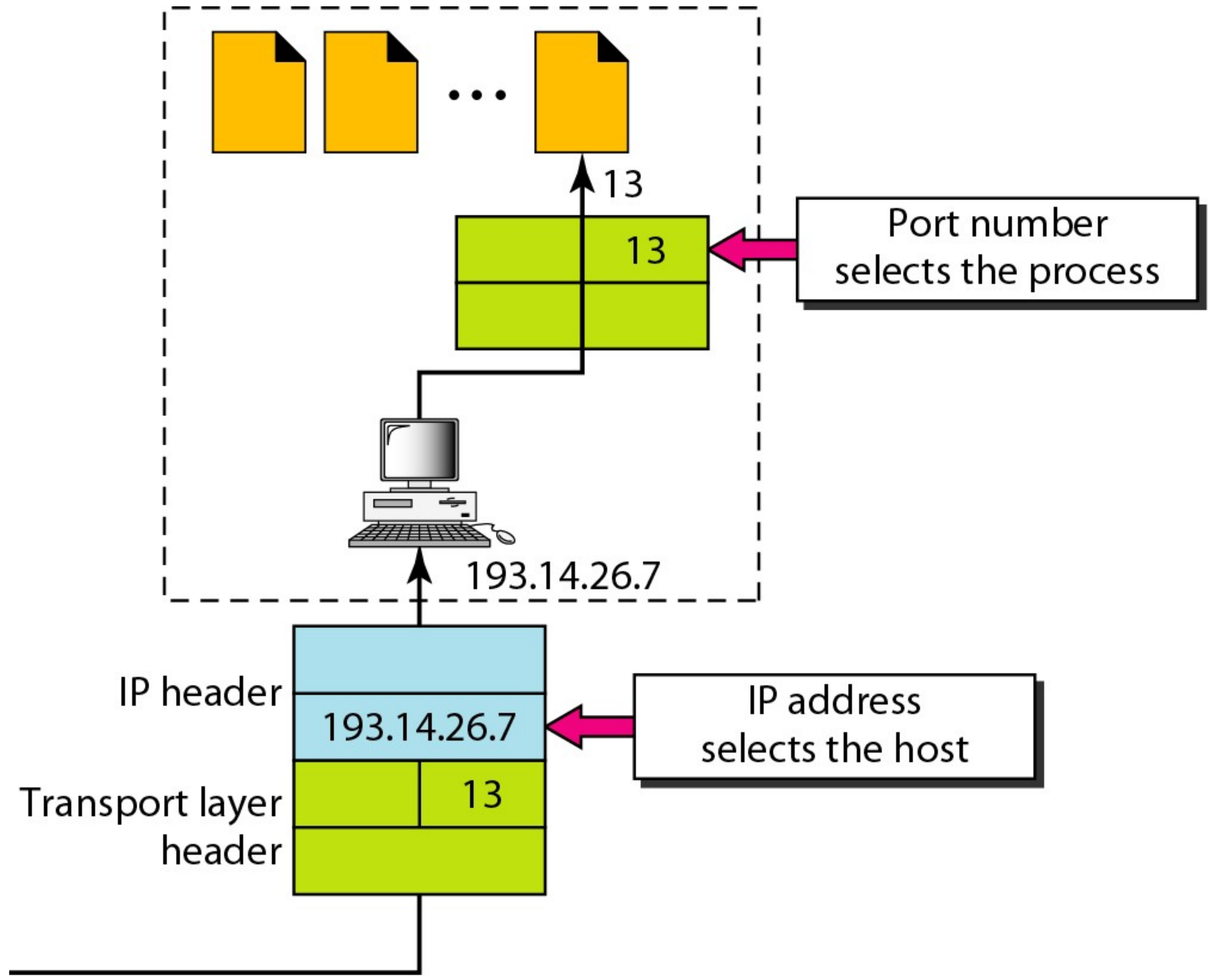
Addressing

- At the data link layer, we need a MAC address to choose one node among several nodes if the connection is not point-to-point.
- A frame in the data link layer needs a destination MAC address for delivery and a source address for the next node's reply.
- At the transport layer, we need a transport layer address, called a port number, to choose among multiple processes running on the destination host.
- The destination port number is needed for delivery; the source port number is needed for the reply.
- In the Internet model, the port numbers are 16-bit integers between 0 and 65,535.
- The client program defines itself with a port number, chosen randomly by the transport layer software running on the client host. This is the ephemeral port number.

Addressing

- The server process must also define itself with a port number. This port number, however, cannot be chosen randomly.
- The Internet has decided to use universal port numbers for servers; these are called well-known port numbers. There are some exceptions to this rule; for example, there are clients that are assigned well-known port numbers.
- Every client process knows the well-known port number of the corresponding server process.
- The IP addresses and port numbers play different roles in selecting the final destination of data.
  - The destination IP address defines the host among the different hosts in the world.
  - After selecting the host, the port number defines one of the processes on this particular host

Figure 23.3 *IP addresses versus port numbers*



Addressing

- Port number is 16-bit and thus has values between 0 to 65535.
- IANA (Internet Assigned Number Authority) has divided port numbers into three ranges:
  - **Well-known ports** - Range from 0 to 1023 assigned and controlled by IANA
  - **Registered ports** - Range from 1024 to 49,151 not assigned or controlled by IANA. They can be registered with IANA to prevent duplication.
  - **Dynamic ports** - Range from 49,152 to 65,535 neither controlled nor registered but can be used by any process and are ephemeral ports.

Addressing

- Process-to-process delivery needs two identifiers, IP address and the port number, to identify process uniquely.
- Combination of IP address and port number is required called as **socket address.**
- A transport layer protocol needs a pair of socket addresses: the client socket address and the server socket address.
- These four pieces of information are part of the IP header and the transport layer protocol header.
- The IP header contains the IP addresses; the UDP or TCP header contains the port numbers.

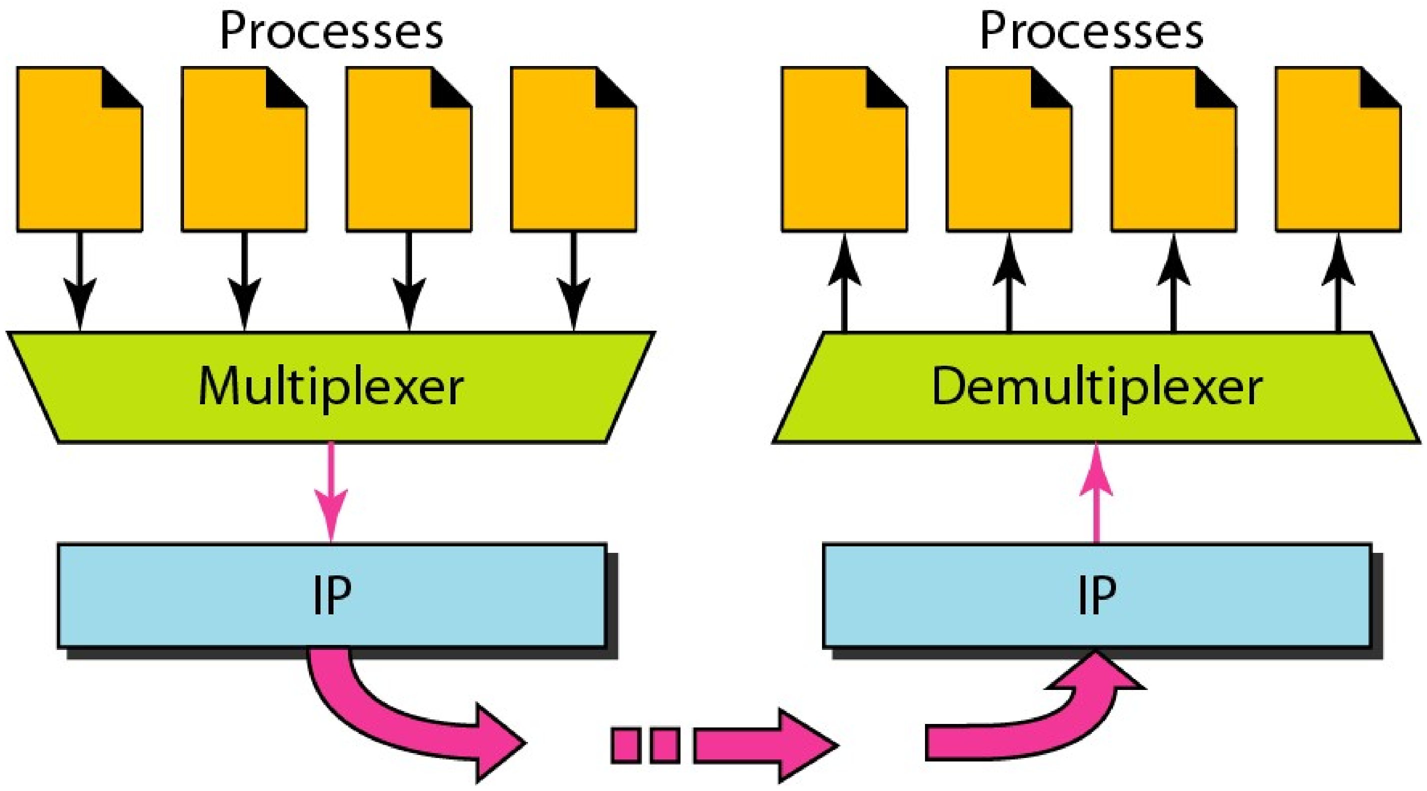
Multiplexing and Demultiplexing

- *Multiplexing*
- At sender site, there may be several processes that need to send packets (many-to-one relationship)
- But there is only one transport layer protocol
- Protocol accepts messages from different processes, differentiated by port numbers
- After adding header, transport layer passes packet to network layer

Multiplexing and Demultiplexing

- *Demultiplexing*
- At receiver site, transport layer receives datagrams from network layer (one-to-many relation)
- After error checking and dropping of header, transport layer delivers each message to appropriate process

Figure 23.6 *Multiplexing and demultiplexing*



Connectionless v/s Connection-Oriented Service

- Transport layer protocol can be connectionless or connection oriented
- Connectionless service
    - No connection establishment
    - Packets are not numbered
    - Packets may be delayed, lost or out of sequence
    - No acknowledgment
- Connection-oriented service
    - Connection is established

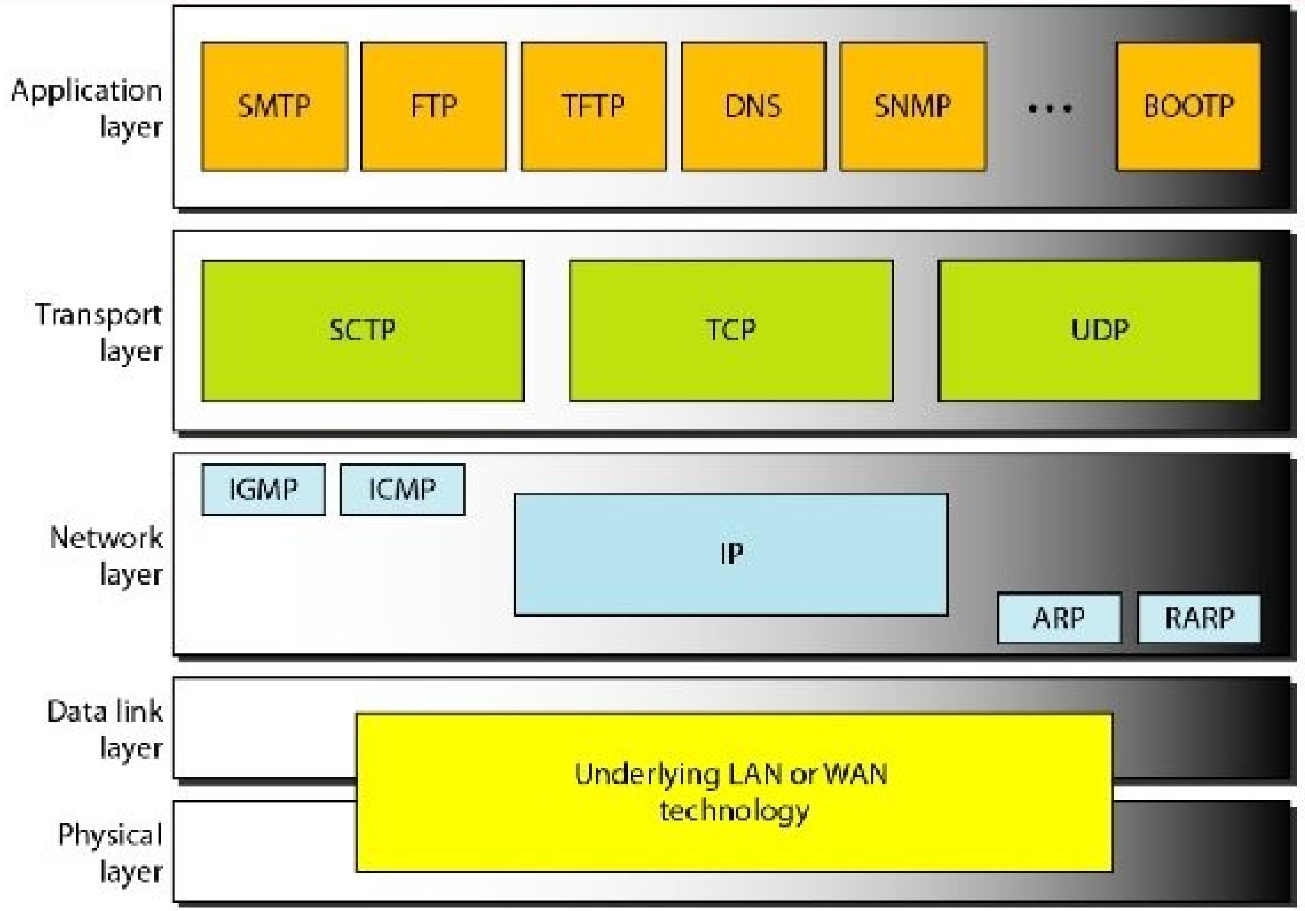
Reliable v/s Unreliable

- Transport layer service can be reliable or unreliable
- If application program needs reliability, reliable transport layer protocol is used
- It implements flow and error control at transport layer
- But slow and more complex service
- If the application program does not need reliability but fast service, then unreliable transport layer protocol is selected
- Data link layer provides reliability between two nodes
- Reliability at the data link layer is between two nodes; we need reliability between two ends.
- Because the network layer in the Internet is unreliable (best-effort delivery), we need to implement reliability at the transport layer.

Transport Layer Protocols

- Three protocols are provided at transport layer in TCP/IP suite:
  - UDP (User Datagram Protocol) – connectionless and unreliable
  - TCP (Transmission Control Protocol) – connection oriented and reliable
  - SCTP (Stream Control Transmission Protocol) – connection oriented and reliable.

Figure 23.8 *Position of UDP, TCP, and SCTP in TCP/IP suite*



User Datagram Protocol (UDP)

- The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol.
- It does not add anything to the services of IP except to provide process-to-process communication instead of host-to-host communication along with limited error checking.
- UDP is connectionless, unreliable transport protocol
- It just perform process-to-process delivery of messages
- Advantage:
  - Minimum overhead (Sending a small message by using UDP takes much less interaction between the sender and receiver than using TCP or SCTP.)

Table 23.1

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
53	Nameserver	Domain Name Service
67	BOOTPs	Server port to download bootstrap information
68	BOOTPc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

Example 23.1

In UNIX, the well-known ports are stored in a file called fetcfservices. Each line in this file gives the name of the server and the well-known port number. We can use the *grep* utility to extract the line corresponding to the desired application. The following shows the port for FTP. Note that FTP can use port 21 with either UDP or TCP.

```
$grep    ftp   fetclservices
ftp       21ftcp
fip       211udp
```

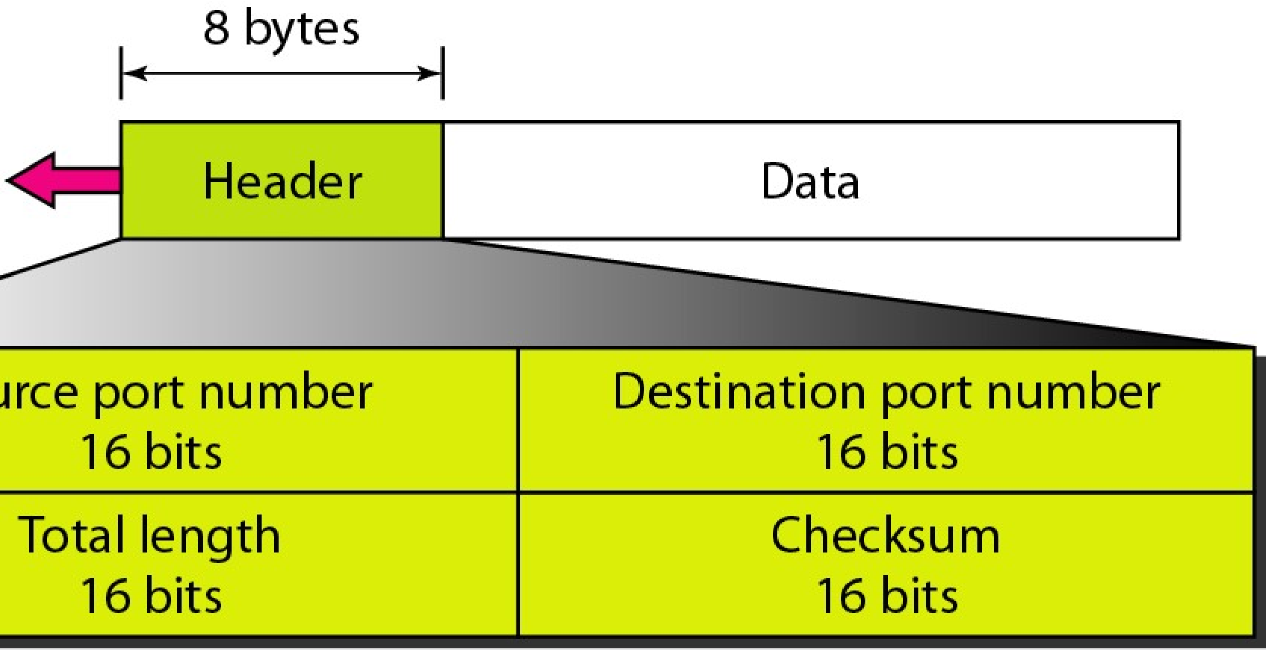
SNMP uses two port numbers (161 and 162), each for a different purpose, as we will see in Chapter 28.

```
$grep     snmp fetclservices
snmp            161ftcp          #Simple Net  Mgmt Proto
snmp            1611udp          #Simple Net  Mgmt Proto
snmptrap        162/udp          #Traps for SNMP
```

User Datagram

- UDP packets are called as user datagrams. Header is of fixed size ie 8 bytes. Fields are as follows:
- **Source port number – 16 bits**
  - Port number used by process running on source host
  - If source host is client then port number is ephemeral port number
  - If source host is server port number, is well-known port number
- **Destination port number-16 bits**
  - Used by the process running on the destination host.
  - If the destination host is the server (a client sending a request), the port number, in most cases, is a well-known port number.
  - If the destination host is the client (a server sending a response), the port number, in most cases, is an ephemeral port number.

Figure 23.9 *User datagram format*



User Datagram

- Length – 16 bits
  - Defines the total length of the user datagram i.e. header + data
  - Length can be from 0 to 65535 bytes
  - But actual length is much less because UDP datagram is stored in IP datagram with total length of 65535 bytes.
  - Length of UDP datagram can be calculated from IP
  - UDP Length = IP Length – IP header length
- Checksum
- Detect errors for entire user datagram I.e. header + data

User Datagram

- The UDP checksum calculation is different from the one for IP and ICMP. Here the checksum includes three sections: a pseudoheader, the UDP header, and the data coming from the application layer.
- The pseudoheader is the part of the header of the IP packet in which the user datagram is to be encapsulated with some fields filled with Os.
- If the checksum does not include the pseudoheader, a user datagram may arrive safe and sound. However, if the IP header is corrupted, it may be delivered to the wrong host.
- The protocol field is added to ensure that the packet belongs to UDP, and not to other transport-layer protocols.

User Datagram

- The value of the protocol field for UDP is 17.
- If this value is changed during transmission, the checksum calculation at the receiver will detect it and UDP drops the packet. It is not delivered to the wrong protocol.
- Optional Use of the Checksum
- The calculation of the checksum and its inclusion in a user datagram are optional. If the checksum is not calculated, the field is filled with 1s.
- Note that a calculated checksum can never be all 1s because this implies that the sum is all Os, which is impossible because it requires that the value of fields to be Os.

Figure 23.10 *Pseudoheader for checksum calculation*

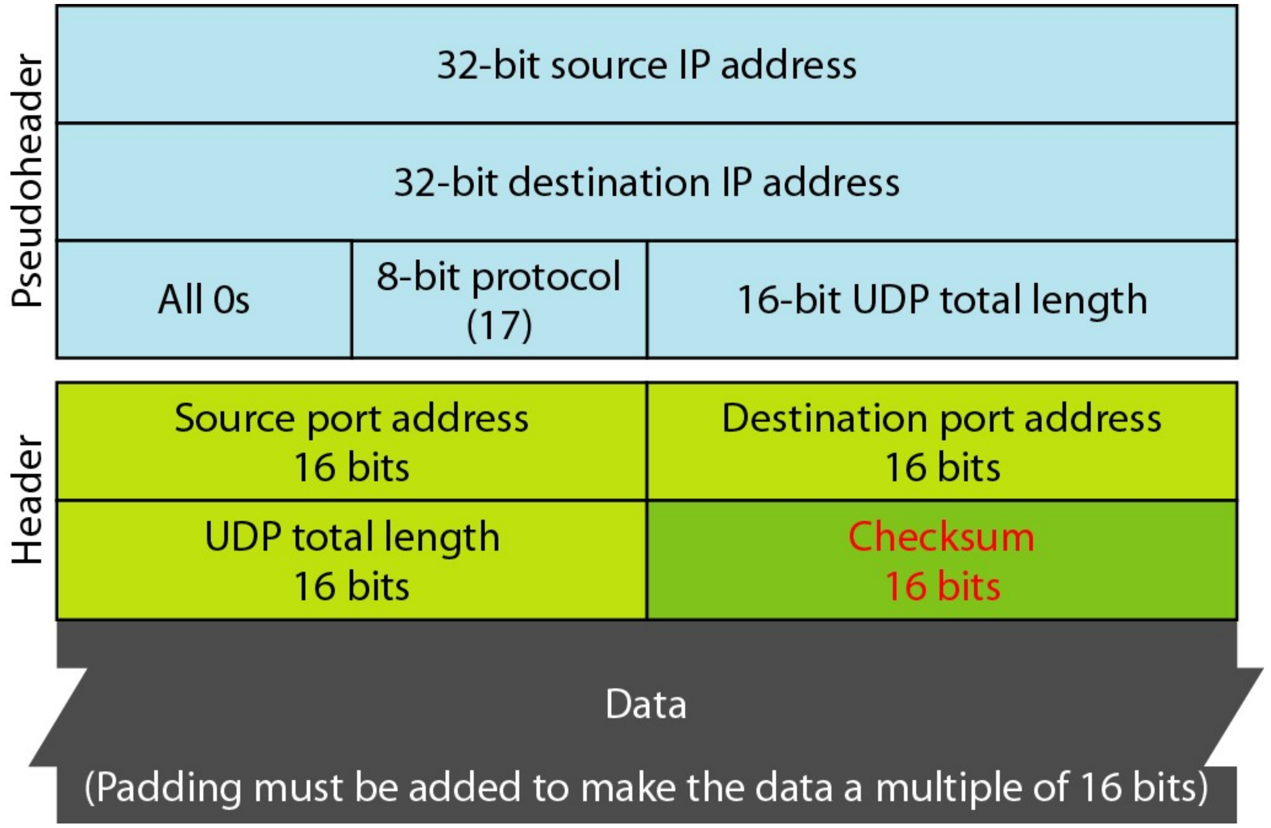
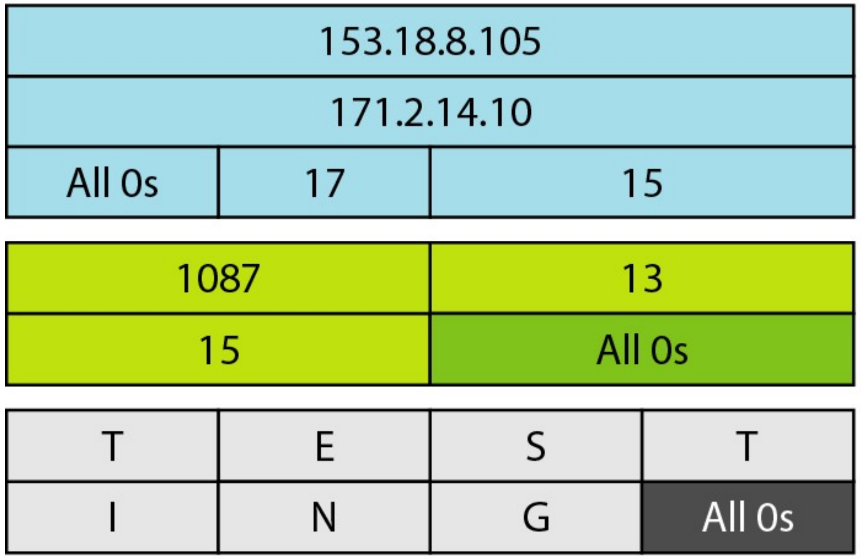

Pseudoheader
32-bit source IP address
32-bit destination IP address
All 0s
8-bit protocol (17)
16-bit UDP total length
Header
Source port address 16 bits
Destination port address 16 bits
UDP total length 16 bits
Checksum 16 bits
Data
(Padding must be added to make the data a multiple of 16 bits)

Figure 23.11 *Checksum calculation of a simple UDP user datagram*



```
10011001 00010010 ──→ 153.18
00001000 01101001 ──→ 8.105
10101011 00000010 ──→ 171.2
00001110 00001010 ──→ 14.10
00000000 00010001 ──→ 0 and 17
00000000 00001111 ──→ 15
00000100 00111111 ──→ 1087
00000000 00001101 ──→ 13
00000000 00001111 ──→ 15
00000000 00000000 ──→ 0 (checksum)
01010100 01000101 ──→ T and E
01010011 01010100 ──→ S and T
01001001 01001110 ──→ I and N
01000111 00000000 ──→ G and 0 (padding)
─────────────────
10010110 11101011 ──→ Sum
01101001 00010100 ──→ Checksum
```

User Datagram

- UDP Operation:
- Connectionless services
  - UDP provides connectionless service
  - User datagrams are not numbered, no connection establishment and no connection termination and hence each user datagram can travel on a different path
  - One change is that UDP does not take stream of data and chop them into datagrams
  - Each request must be small enough to fit in one user datagram

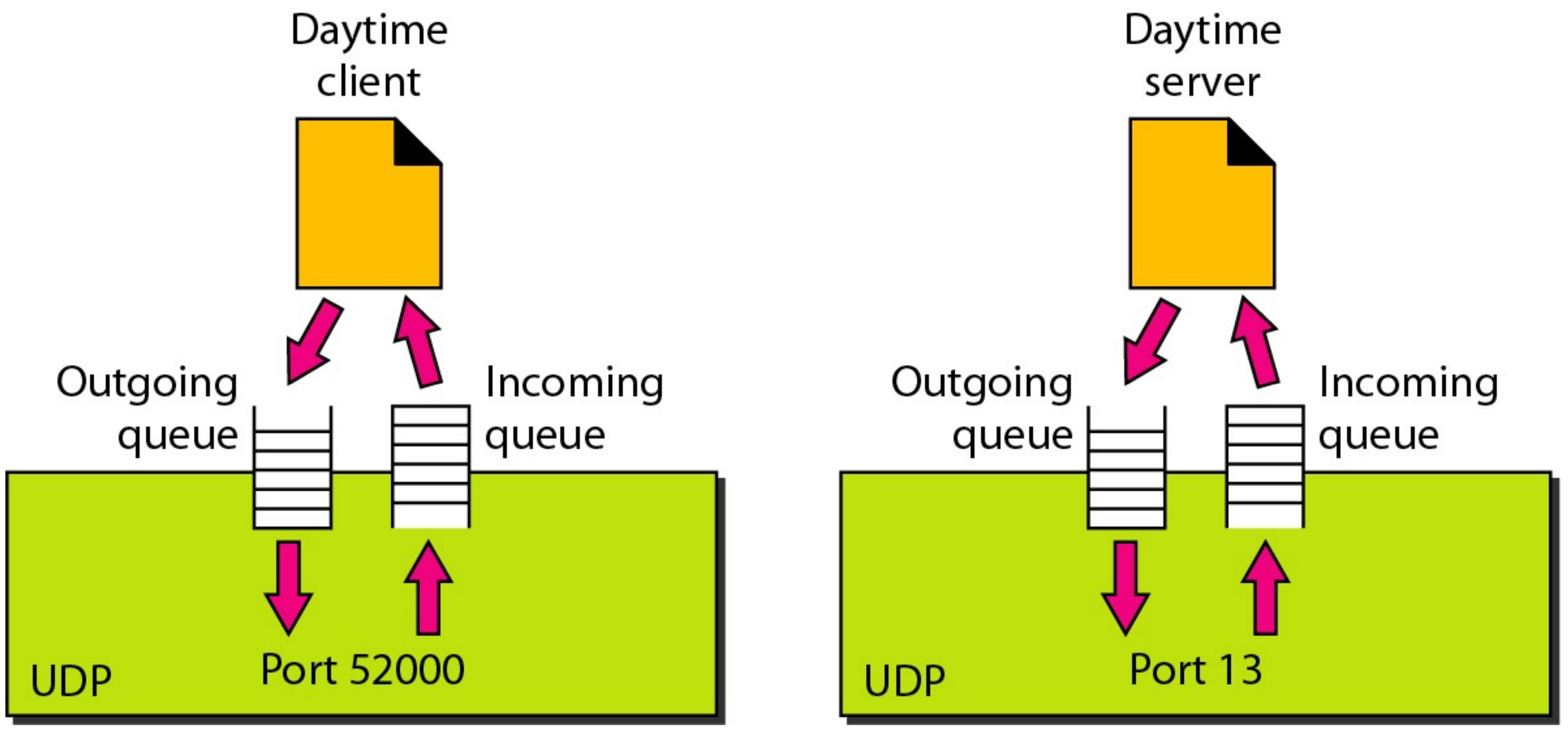
User Datagram

- Flow and Error Control
  - UDP is unreliable transport protocol with no flow control and hencet he receiver may overflow with incoming messages.
  - There is no error control mechanism in UDP except for the checksum. This means that the sender does not know if a message has been lost or duplicated. When the receiver detects an error through the checksum, the user datagram is silently discarded.
  - The lack of flow control and error control means that the process using UDP should provide these mechanisms.

User Datagram

- If error is detected using checksum, datagram is discarded.
- Encapsulation and Decapsulation
  - To send messages, UDP protocol encapsulates and decapsulates messages in IP datagram
- Queuing
  - Queues are associated with ports
  - (Some implementations create both an incoming and an outgoing queue associated with each process. Other implementations create only an incoming queue associated with each process.)

Figure 23.12 *Queues in UDP*



User Datagram

- Queue at client site
  - When process starts, client requests for port number from operating system
  - Even if a process wants to communicate with multiple processes, it obtains only one port number and eventually one outgoing and one incoming queue
  - When process terminates, queues are destroyed
  - Client send messages to outgoing queue using port number
  - UDP removes messages one by one and add UDP header to each message, delivers to IP

User Datagram

- If outgoing queue overflows, OS asks client process to wait
- When message arrives, UDP checks incoming queue for destination port number is created or not
- If queue is present, UDP sends received datagram to end of queue
- If no queue, UDP discards datagram
- It asks ICMP protocol to send port unreachable message to server
- All incoming messages even if coming from different servers are sent to same queue

User Datagram

- Incoming queue may overflow
- In this, UDP discards datagram and asks for port unreachable message to server
- Queue at server site
- Similar process at server
- Server asks for incoming and outgoing queues using its well-known port, when it starts running
- Queues remain open as long as server is running

User Datagram

<u>Uses of UDP</u>

- UDP is suitable for process that requires simple request-response communication with little concern for flow and error control. Not suitable for bulk data.
- UDP is suitable for process with internal flow and error control mechanisms. Example Trivial File Transfer Protocol (TFTP) UDP is suitable for multicasting
- UDP is used for management processes such as SNMP
- UDP is used for some route updating protocols such as Routing Information Protocol (RIP)

Transmission Control Protocol (TCP)

- TCP, like UDP, is a process-to-process (program-to-program) protocol. TCP, therefore, like UDP, uses port numbers.
- TCP is called a connection-oriented, reliable transport protocol. It adds connection-oriented and reliability features to the services of IP.

TCP Services

- ***Process-to-Process Communication***
- Like UDP, TCP provides process-to-process communication using port numbers.
- Table 23.2 lists some well-known port numbers used by TCP.

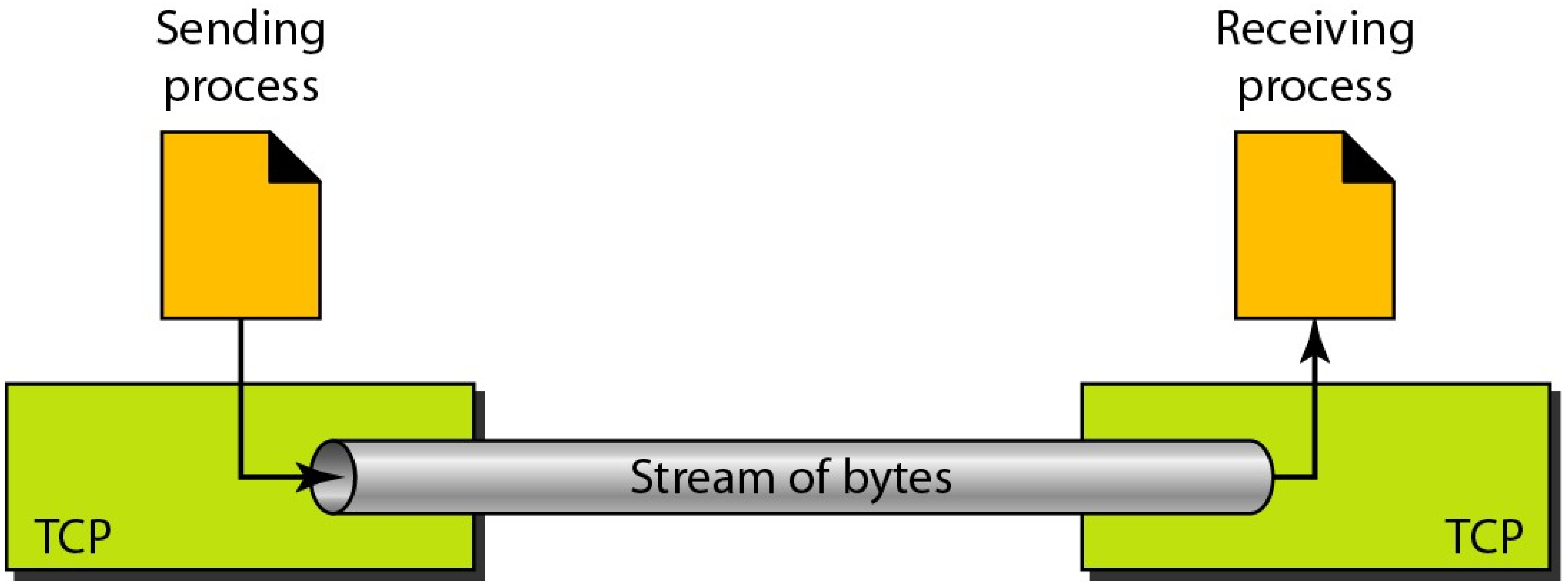
Table 23.2 *Well-known ports used by TCP*

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20	FTP, Data	File Transfer Protocol (data connection)
21	FTP, Control	File Transfer Protocol (control connection)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol
111	RPC	Remote Procedure Call

TCP Services

- ***Stream Delivery Service***
  - TCP, unlike UDP is stream-oriented protocol
  - In UDP, process sends messages with predefined boundaries
  - TCP delivers data as stream of bytes and receives data as stream of bytes
  - TCP creates an environment in which two processes seem to be connected by an imaginary "tube” that carries data

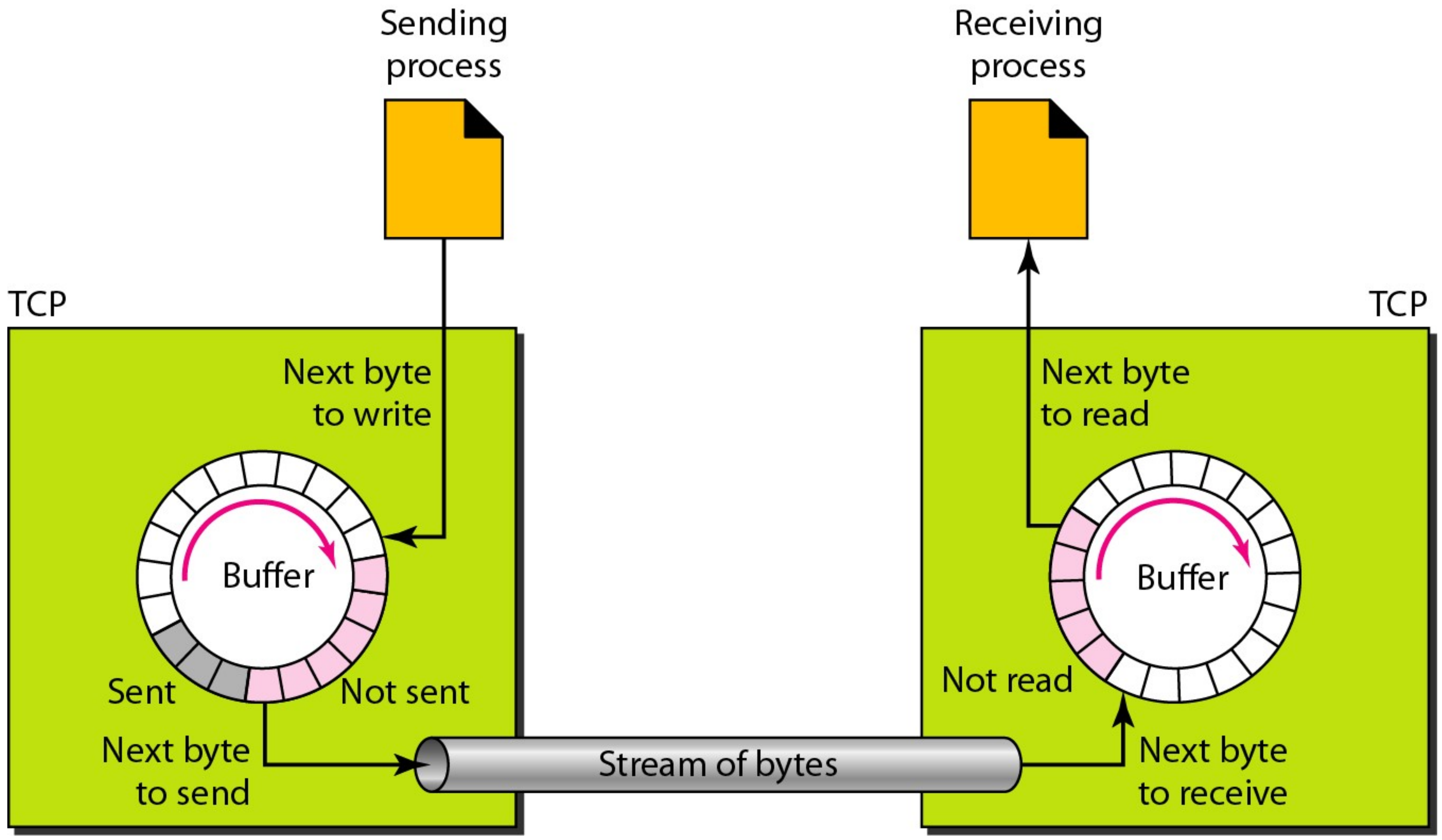
Figure 23.13 *Stream delivery*



TCP Services

- ***Sending and Receiving Buffers***
  - Sending and receiving processes may not write or read data at same speed
  - TCP uses sending and receiving buffers for storage
  - Buffer is implemented as circular array of 1-byte
  - Both buffers may or may not be of same size

Figure 23.14 *Sending and receiving buffers*



TCP Services

- ***Sending and Receiving Buffers***
  - Sender side buffer has three sections:
    - Empty chambers that can be filled by sending process
    - Area that holds bytes that have been sent out but not acknowledged
    - Area of bytes to be sent by sending TCP
    - After bytes are acknowledged, chambers are recycled and available for use by sending process
    - Thus it is circular buffer

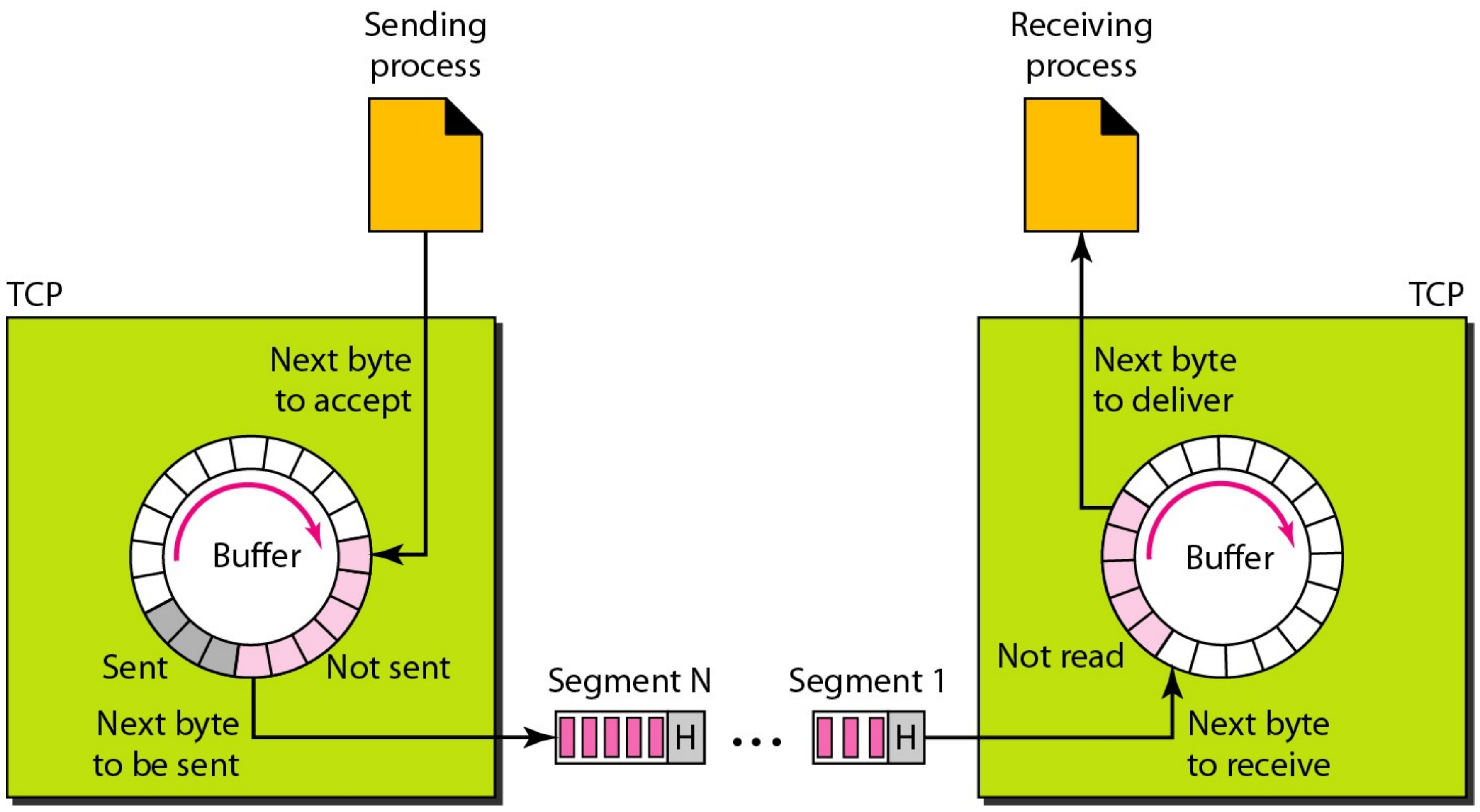
TCP Services

- ***Sending and Receiving Buffers***
  - Receiver site buffer
  - Circular buffer is divided into two areas:
    - Empty chambers to be filled by bytes received from network
    - Area of received bytes that can be read by receiving process
    - When byte is read by receiving process, chamber is recycled and added to pool of empty chambers

TCP Services

- ***Segments***
  - IP layer needs to send data in packets not as stream of bytes
  - TCP groups number of bytes together into a packet called as segment
  - TCP adds header to each segment and delivers to IP layer for transmission

Figure 23.15 *TCP segments*



TCP Services

- ***Full-Duplex Communication***
  - TCP offers full-duplex service
  - Data can flow in both directions at same time
  - Each TCP has sending and receiving buffer
- ***Connection-Oriented Service***
  - When process at site A wants to send and receive data from another process at site B following steps occurs:
    1. Two TCPs establish connection
    2. Data are exchanged in both directions
    3. Connection is terminated

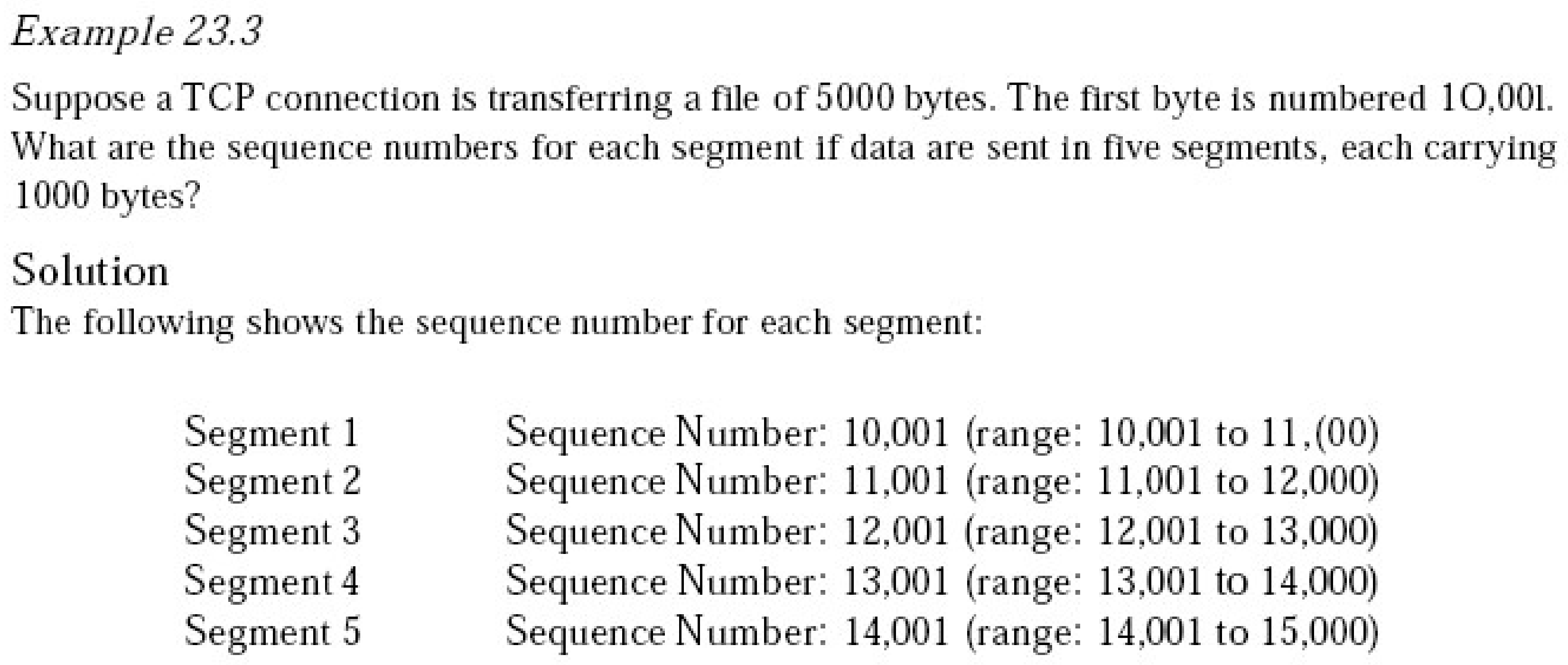
TCP Services

  - Connection is virtual connection, not physical connection
  - TCP segment is encapsulated in IP datagram which is connectionless protocol
  - IP delivers individual segments but connection is controlled by TCP
  - If segment is lost or corrupted, it is retransmitted
  - IP is unaware of this retransmission
- ***Reliable Service***
  - TCP is reliable transport protocol
  - It uses acknowledgment mechanism

TCP Services

- ***Numbering System***
  - To keep track of segments being transmitted or received, two fields called **sequence number** and **acknowledgment number** are used
  - They refer to byte number and not segment number
- ***Byte Number***
  - TCP numbers all data bytes that are transmitted in a connection
  - Numbering is independent in each direction

TCP Services



Example 23.3

Suppose a TCP connection is transferring a file of 5000 bytes. The first byte is numbered 10,001. What are the sequence numbers for each segment if data are sent in five segments, each carrying 1000 bytes?

Solution

The following shows the sequence number for each segment:

Segment 1	Sequence Number: 10,001 (range: 10,001 to 11,(00)
Segment 2	Sequence Number: 11,001 (range: 11,001 to 12,000)
Segment 3	Sequence Number: 12,001 (range: 12,001 to 13,000)
Segment 4	Sequence Number: 13,001 (range: 13,001 to 14,000)
Segment 5	Sequence Number: 14,001 (range: 14,001 to 15,000)

TCP Features

  - Numbering does not necessarily start from 0
  - TCP generates random number between 0 and 2^{32} - 1 for number of first byte
- ***Sequence Number***
  - After bytes have been numbered, TCP assigns sequence number to each segment that is being sent
  - Sequence number is number of first byte carried in that segment
  - When segment carries data and control information (piggybacking), it uses sequence number

TCP Features

- If segment does not carry user data, it does not logically define sequence number
- Field will have invalid value
- But some segments carrying only control information for connection establishment need sequence number
- Such segments consumes one sequence number as though it carried 1 byte, but there are no actual data
- If randomly generated sequence number is *x, first data byte is numbered x + 1*
- *Byte x is considered as* phony byte used for control segment

TCP Features

- ***Acknowledgment Number***
  - Each party uses acknowledgment number to confirm bytes received
  - Acknowledgment number defines number of next expected byte
  - Party takes number of last byte that it has received, safe and sound, adds 1 to it, and announces this sum as acknowledgment number

TCP Features

- Acknowledgment number is cumulative
- Cumulative means that if party uses 5643 as acknowledgment number, means it has received all bytes from the beginning up to 5642
- But this does not mean it has received 5642 bytes because first byte number does not have to start from 0

- ***Flow Control***
  - Receiver controls amount of data that are to be sent by sender
  - Numbering system allows TCP to use byte-oriented flow control

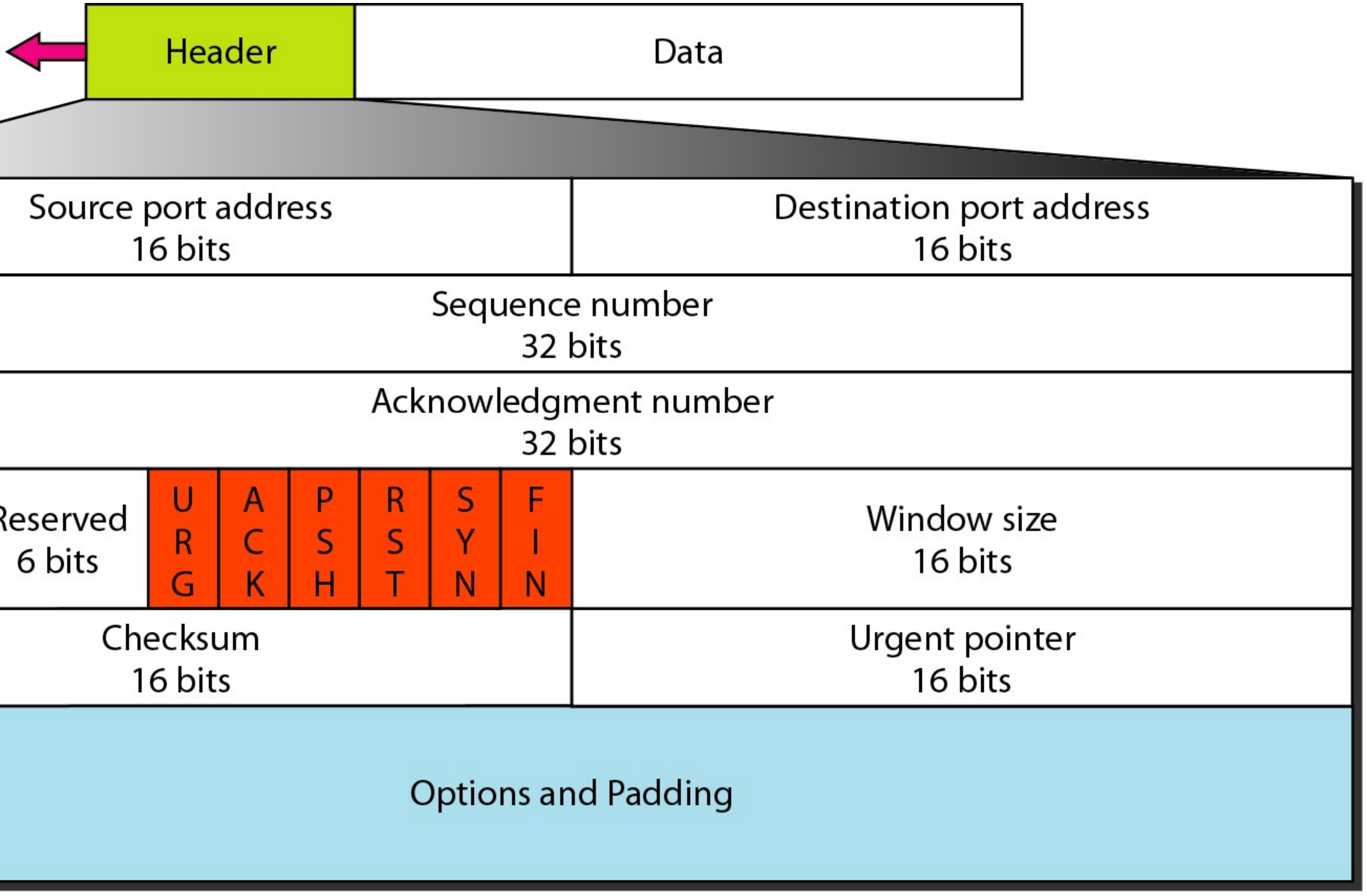
TCP Features

- ***Error Control***
  - To provide reliable service, TCP implements error control mechanism
  - Error control is byte-oriented
- ***Congestion Control***
  - TCP, takes into account congestion in network
  - Amount of data sent is controlled by receiver as well as level of congestion in network

TCP Features

- ***Segment***
- TCP packets are called as **segment**
- Segment has 20 to 60 bytes of header followed by data from application layer
- Header is of 20 bytes without option and upto 60 bytes with option

Figure 23.16 *TCP segment format*



TCP Features

- Following fields:
- **Source port address** – 16 bits
  - Port number of application program host that is sending segment
- **Destination port address** – 16 bits
  - Port number of application program in host that is receiving segment
- **Sequence number**-32 bits
  - Number assigned to first byte of data contained in this segment
  - The sequence number tells the destination which byte in this sequence comprises the first byte in the segment.

TCP Features

  - During connection establishment each party uses random number generator to create initial sequence number (ISN), which is usually different in each direction
- **Acknowledgment number** – 32 bits
  - Byte number that receiver of segment is expecting
  - If receiver has successfully received byte number *x, it* defines *x + 1 as acknowledgment number*
- **Header length** – 4 bits
  - Number of 4-byte words in TCP header
- **Reserved** – 6 bits
  - Reserved for future use

TCP Features

- **Control** – 6 bits
  - This field defines 6 control bits or flags
  - One or more of these bits can be set at time
  - These bits enable connection establishment, termination, abortion and data transfer

Figure 23.17 *Control field*

URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection

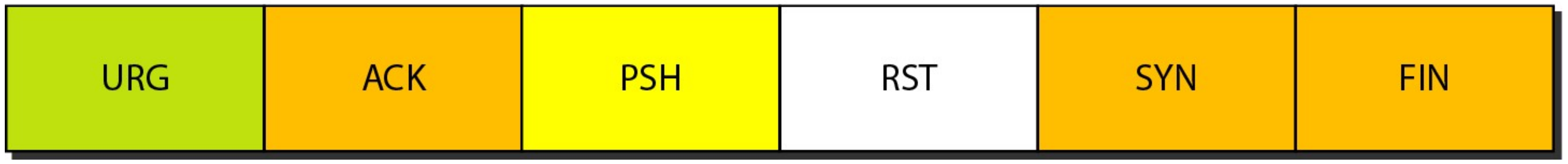


Table 23.3 Description offlags in the control field

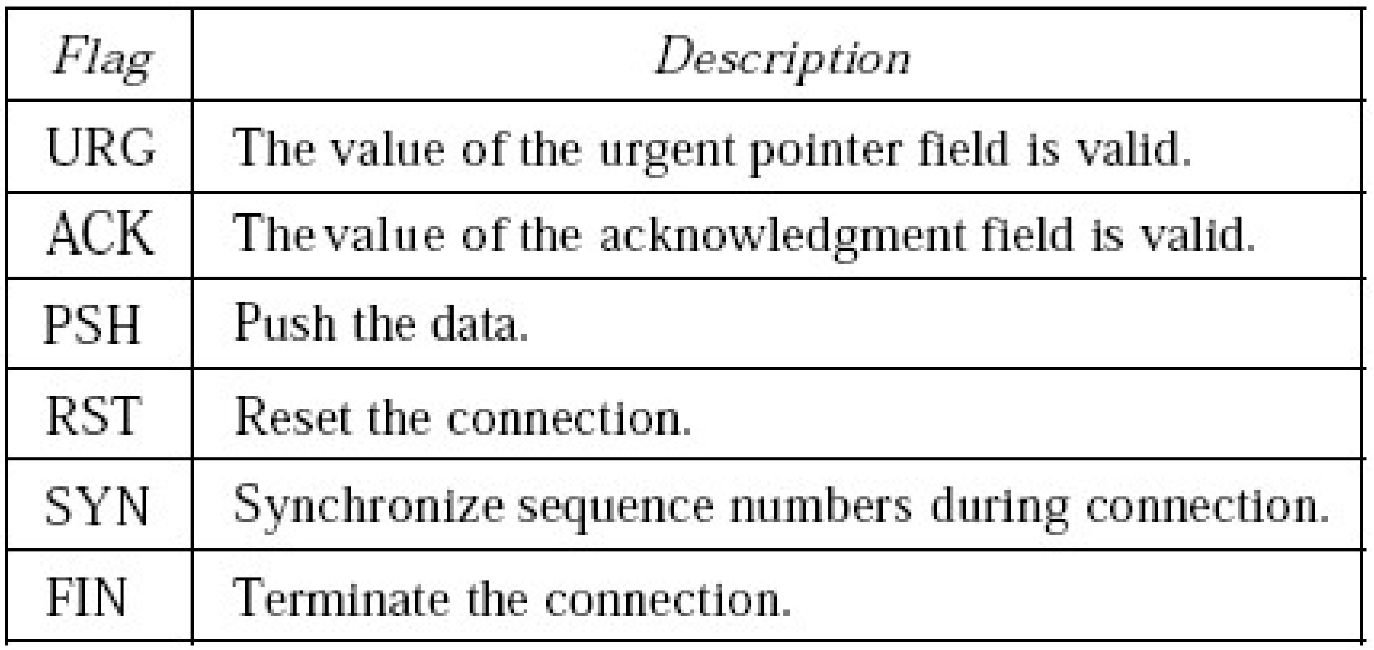


Table 23.3 *Description offlags in the control field*

Flag	*Description*
URG	The value of the urgent pointer field is valid.
ACK	The value of the acknowledgment field is valid.
PSH	Push the data.
RST	Reset the connection.
SYN	Synchronize sequence numbers during connection.
FIN	Terminate the connection.

TCP Features

- **Window size** – 16 bits
  - Defines size of window in bytes that other party must maintain
  - Maximum size of window is 65,535 bytes
  - This value is normally referred to as receiving window (rwnd) and is determined by receiver
- **Checksum** – 16 bits
  - Calculation of checksum is same as UDP
  - But inclusion of checksum for TCP is mandatory
  - For TCP value of protocol field is 6

TCP Features

- **Urgent pointer** – 16 bits
  - This field is valid only if urgent flag is set
  - Used when segment contains urgent data
  - It defines number that must be added to sequence number to obtain number of last urgent byte in data
- **Options** – 40 bytes
  - Up to 40 bytes of optional information

A TCP Connection

- TCP is connection-oriented.
- A connection-oriented transport protocol establishes a virtual path between the source and destination. All the segments belonging to a message are then sent over this virtual path.
- Using a single virtual pathway for the entire message facilitates the acknowledgment process as well as retransmission of damaged or lost frames.
- A connection-oriented TCP uses services of connectionless IP as TCP connection is virtual, not physical.
- TCP uses the services of IP to deliver individual segments to the receiver, but it controls the connection itself. If a segment is lost or corrupted, it is retransmitted without IP knowing it.

A TCP Connection

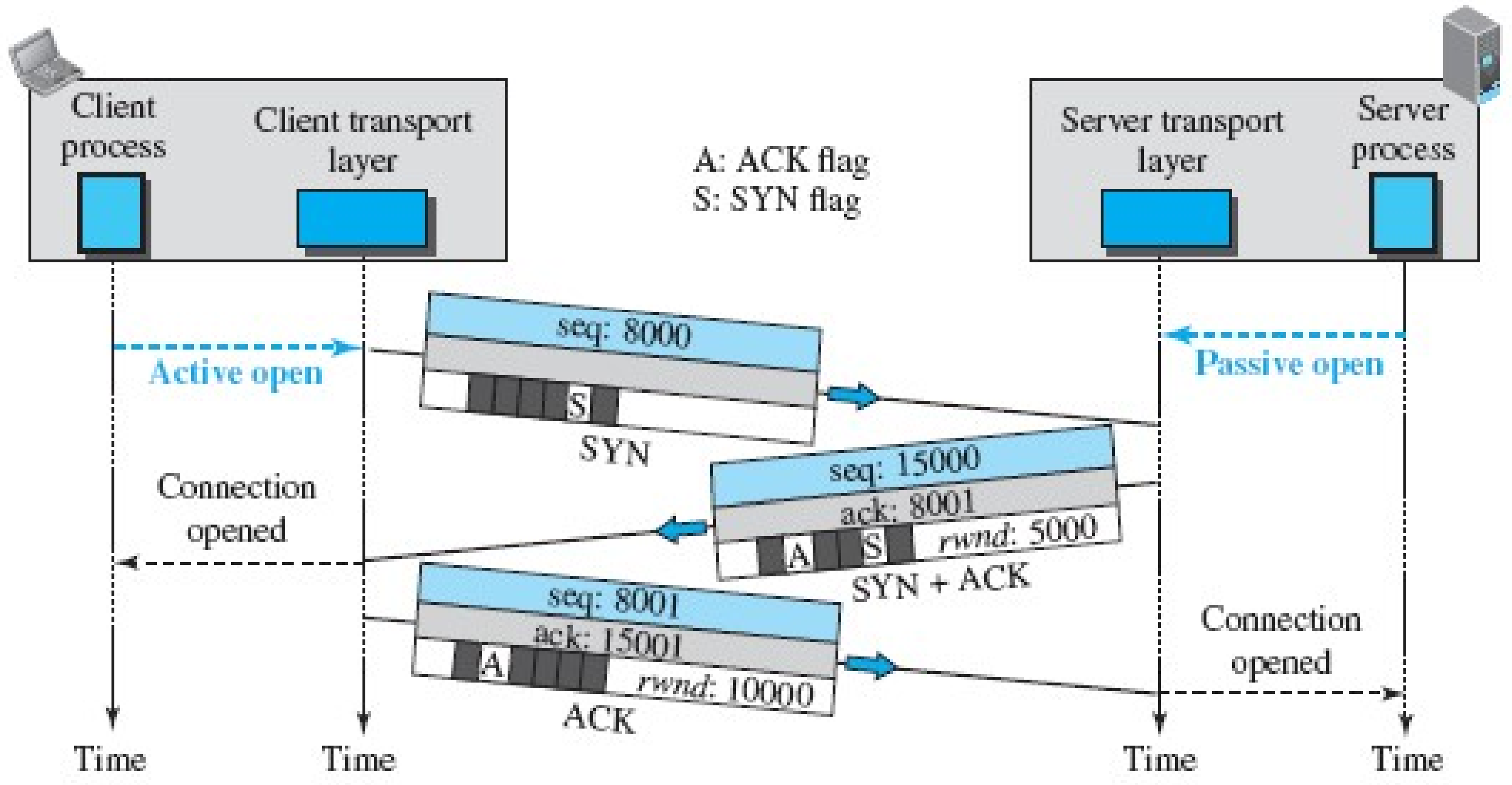
- In TCP, connection-oriented transmission requires three phases: connection establishment, data transfer, and connection termination.
- *Connection Establishment*
- TCP transmits data in full-duplex mode. When two TCPs in two machines are connected, they are able to send segments to each other simultaneously.
- This implies that each party must initialize communication and get approval from the other party before any data are transferred

A TCP Connection

- ***Three-Way Handshaking***
- The connection establishment in TCP is called threeway handshaking.
- The process starts with the server program telling its TCP that it is ready to accept a connection.
- This is called a request for a passive open. But the server TCP cannot make the connection itself.
- The client program issues a request for an active open. A client that wishes to connect to an open server tells its TCP that it needs to be connected to that particular server.
- TCP can now start the three-way handshaking process as shown in Figure 23.18.

A TCP Connection

Figure 23.18 *Connection establishment using three-way handshaking*



A TCP Connection

- ***Three-Way Handshaking***
- We show only the few fields necessary to understand each phase.
- We show the sequence number, the acknowledgment number, the control flags (only those that are set), and the window size, if not empty.
- The three steps in this phase are as follows.
- **1.** The client sends the first segment, a SYN segment, in which only the SYN flag is set. It consumes one sequence number.
- When the data transfer starts, the sequence number is incremented by 1. We can say that the SYN segment carries no real data

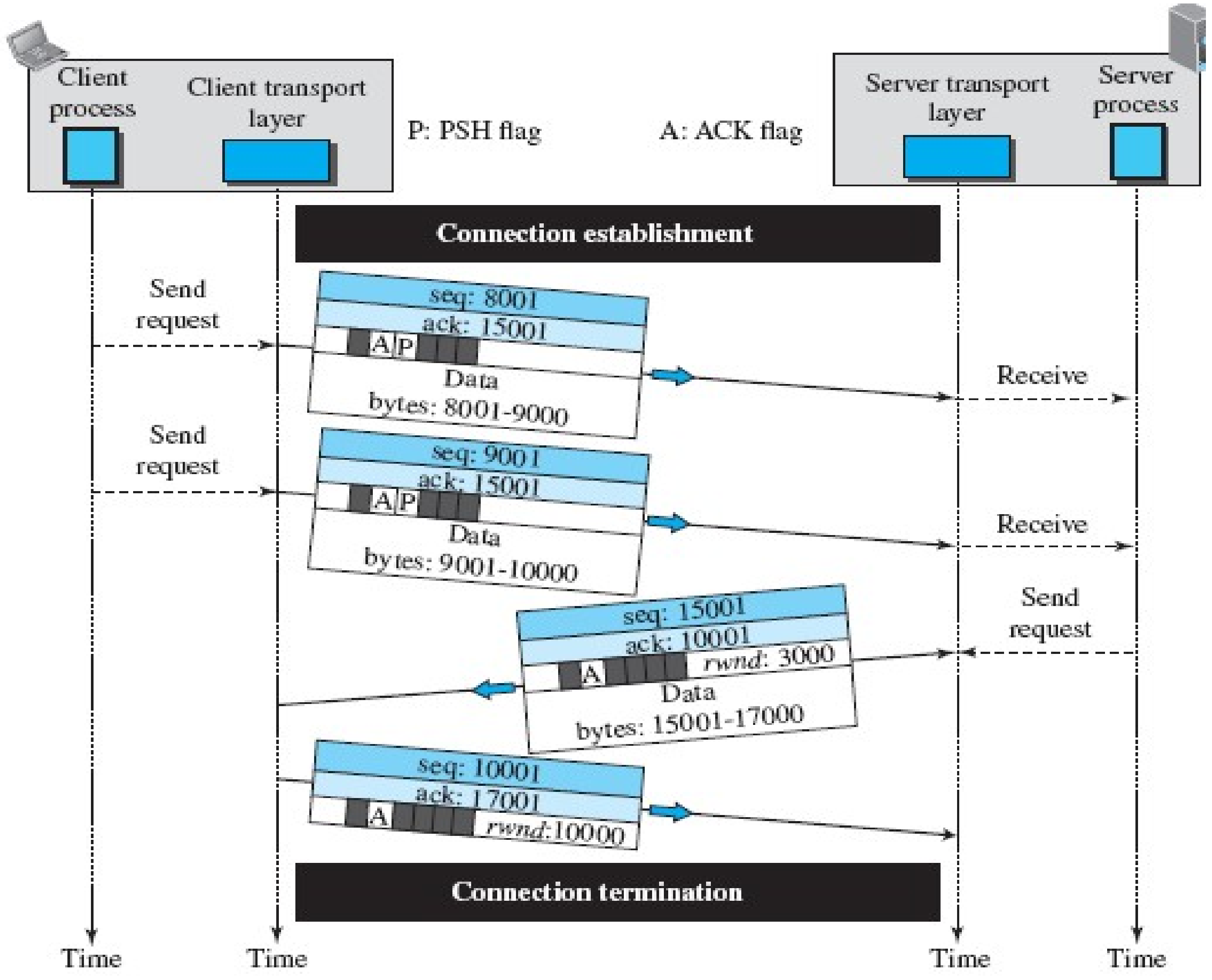
A TCP Connection

- ***Three-Way Handshaking***
- **2.** The server sends the second segment, a SYN +ACK segment, with 2 flag bits set:SYN and ACK.
- It is a SYN segment for communication in the other direction and serves as the acknowledgment for the SYN segment. It consumes one sequence number.
- A SYN +ACK segment cannot carry data, but does consume one sequence number.
- **3.** The client sends the third segment. This is just an ACK segment. It acknowledges the receipt of the second segment with the ACK flag and acknowledgment number field (the same as the one in the SYN segment).

Data Transfer

- After connection is established, bidirectional data transfer can take place. The client and server can both send data and acknowledgments.
- Data traveling in the same direction as an acknowledgment are carried on the same segment.
- Figure 23.19 shows an example.

Figure 23.19 *Data transfer*



Data Transfer

- ***Pushing Data***
- TCP uses buffer to store stream of data from sending application program .
- Receiving TCP also buffers data when they arrive and delivers to application program when it is ready .
- There are situations when delayed transmission or delayed delivery of data is not acceptable by application program.
- Application program at sending site can request push operation i.e. sending TCP must not wait for window to be filled.
- It must create segment and send it immediately.
- Sending TCP must also set push bit (PSH) to let receiving TCP know that segment must be delivered to receiving application program as soon as possible.

Data Transfer

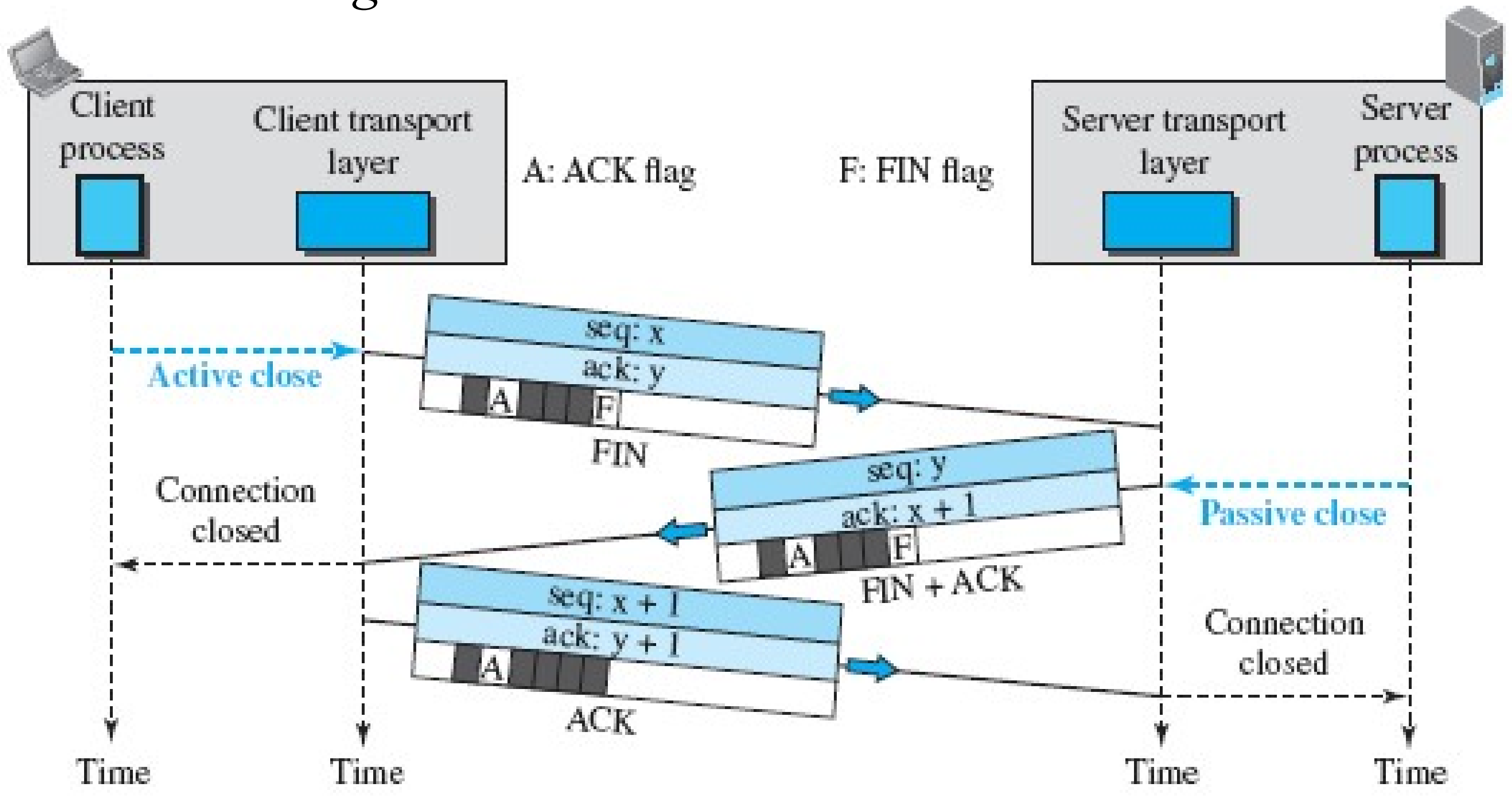
- ***Urgent Data***
- TCP represents data as stream of bytes
- Each byte of data has position in stream
- But sometimes an application program wants piece of data to be read out of order by receiving application program
- Sending TCP creates segment and inserts urgent data at beginning of segment and URG bit is set
- Rest of segment can have normal data
- Urgent pointer field defines end of urgent data and start of normal data
- When receiving TCP receives such segment, it extracts urgent data using urgent pointer and deliver out of order to application program

Connection Termination

- Any of the two parties involved in exchanging data (client or server) can close the connection, although it is usually initiated by the client.
- Most implementations today allow two options for connection termination: three-way handshaking and four-way handshaking with a half-close option.
- Three-Way Handshaking Most implementations today allow three-way handshaking for connection termination as shown in Figure 23.20.

Connection Termination

Figure 23.20 *Connection termination using three-way handshaking*



Connection Termination

- *Three-Way Handshaking*
- Client TCP receives close command from client process
  - It sends first segment, FIN segment with FIN flag set
  - FIN segment can include last chunk of data
  - Or it can be just control segment consuming only one sequence number
- Server TCP sends second segment, FIN +ACK segment, to confirm the receipt of FIN segment
  - It also announces closing of connection in other direction
  - This segment can also contain last chunk of data from server
  - If it does not carry data, it consumes only one sequence number
- Client TCP sends last segment, ACK segment, to confirm receipt of FIN segment
  - Acknowledgment number 1 plus sequence number received in FIN segment from server
  - This segment cannot carry data and consumes no sequence numbers

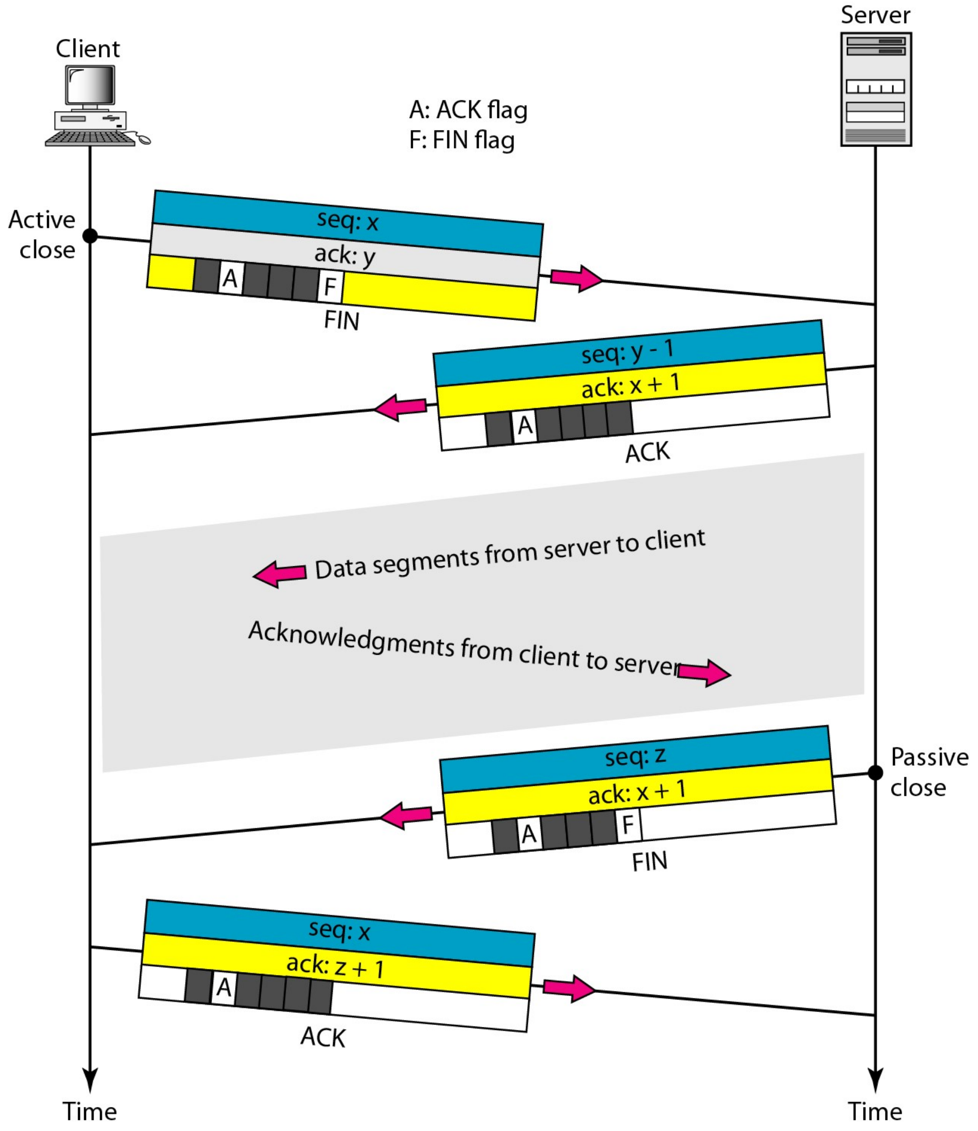
Connection Termination

- *Half-Close*
- One end can stop sending data while still receiving data This is called a half-close Client half-closes connection by sending FIN segment
- Server accepts half-close by sending ACK segment
- Data transfer from client to server stops
- Server can still send data
- When the server has sent all processed data, it sends FIN segment, which is acknowledged by ACK from client

Connection Termination

- *Half-Close*
- Second segment (ACK) consumes no sequence number
- Although client has received sequence number *y -1 and is expecting y server sequence number is still y – 1*
- *When the connection* finally closes sequence number of last ACK segment is still *x, because no* sequence numbers are consumed during data transfer in that direction

Figure 23.21 *Half-close*

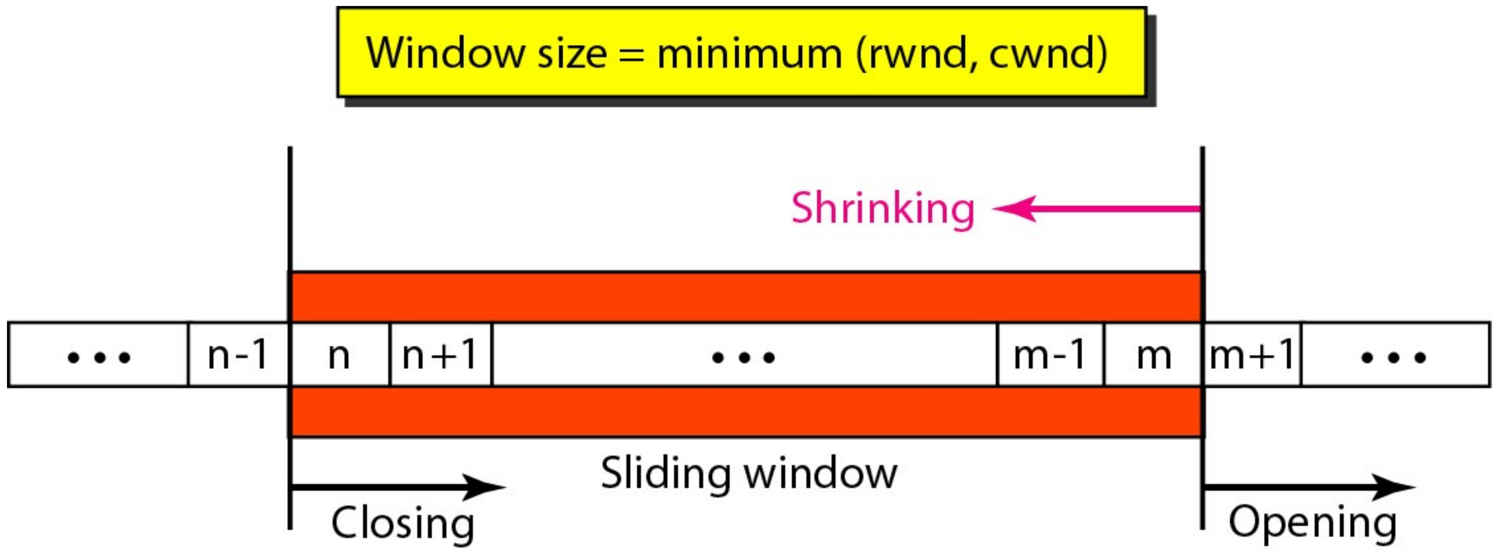


Flow Control

- TCP uses sliding window to handle flow control
- Sliding window is byte oriented and variable size
- Window spans portion of buffer containing bytes received from process
- Bytes inside window are bytes that can be in transit
- Size of window at one end is determined by lesser of two values: Receiver window (rwnd) or congestion window (cwnd)
- Receiver window value is given by opposite end in a segment containing acknowledgment
- It is number of bytes other end can accept before its buffer overflows
- Congestion window value is determined by network to avoid congestion
- Window has two walls: left and right
- Window can be one of three states :*Opened, Closed and Shrunk.*
- States of window is controlled by receiver not sender

Flow Control

Figure 23.22 *Sliding window*



Flow Control

- Opening window means moving right wall to right
- Allow new bytes in buffer eligible for sending
- Closing window means moving left wall to right
- Some bytes have been acknowledged and sender need not worry about them
- Shrinking window means moving right wall to left
- Strongly discouraged and not allowed in some implementations
- It means revoking eligibility of some bytes for sending
- Problem if sender has already sent these bytes

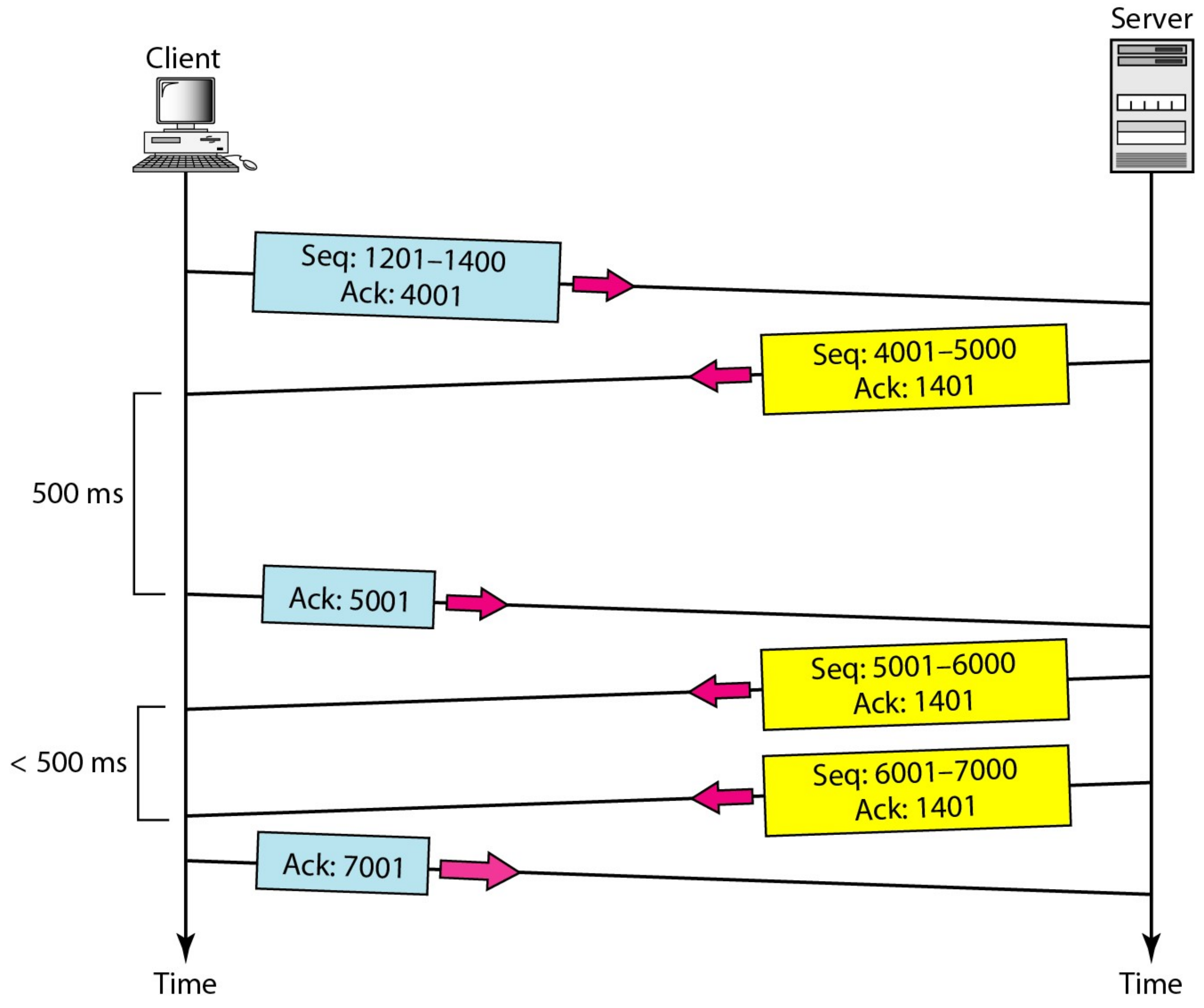
Flow Control

- Left wall cannot move to left
- This would revoke some of previously sent acknowledgment
- Some points about TCP sliding windows:
- The size of the window is the lesser of rwnd and cwnd.
- The source does not have to send a full window's worth of data.
- The window can be opened or closed by the receiver, but should not be shrunk.
- The destination can send an acknowledgment at any time as long as it does not result in a shrinking window.
- The receiver can temporarily shut down the window; the sender, however, can always send a segment of 1 byte after the window is shut down

Flow Control

- Error control includes detection of corrupted, lost, out-of-order and duplicated segments
- Uses three mechanisms: checksum, acknowledgement and time-out
- ***Checksum***
- Each segment includes checksum field
- ***Acknowledgment***
- Acknowledgments confirm receipt of data segments
- It also confirms control segments that carry no data but consume sequence number
- ACK segments itself are never acknowledged

Figure 23.24 *Normal operation*



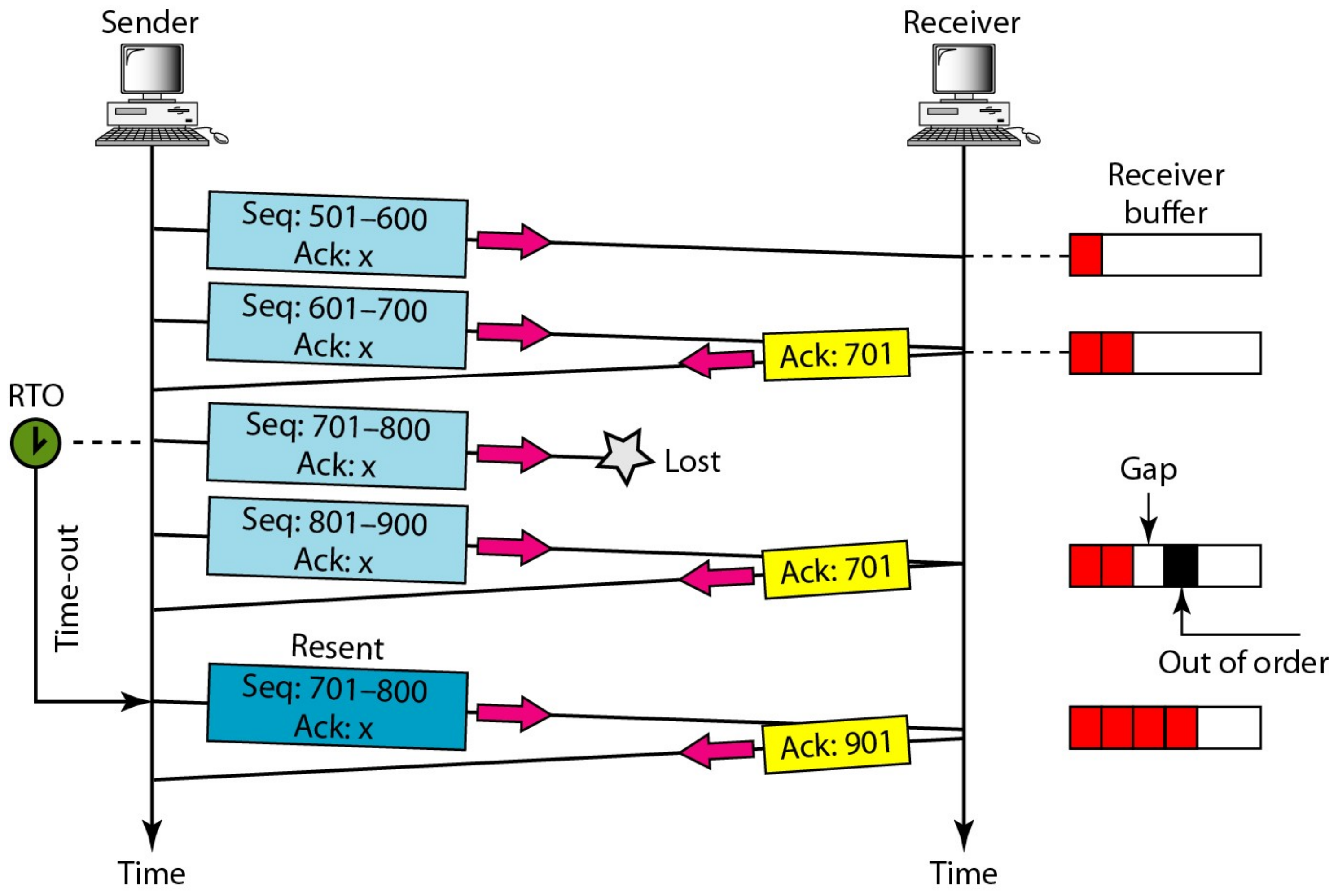
Error Control

- ***Retransmission***
- When segment is corrupted, lost, or delayed, it is retransmitted
- Segment is retransmitted on two occasions:
- Retransmission timer expires
- Sender receives three duplicate ACKs
- **Retransmission After RTO**
- TCP maintains one retransmission time-out (RTO) timer for all outstanding segments
- When timer matures, earliest outstanding segment is retransmitted

Error Control

- Time-out timer is not set for segment that carries only acknowledgment
- Value of RTO is dynamic and is updated based on round-trip time (RTT) of segments
- Out-of-order segments
- When segment is delayed, lost or discarded, other following segments arrive out of order
- These segments are not discarded
- Instead they are stored temporarily and marked as out-of-order
- When missing segment arrives, are delivered to process

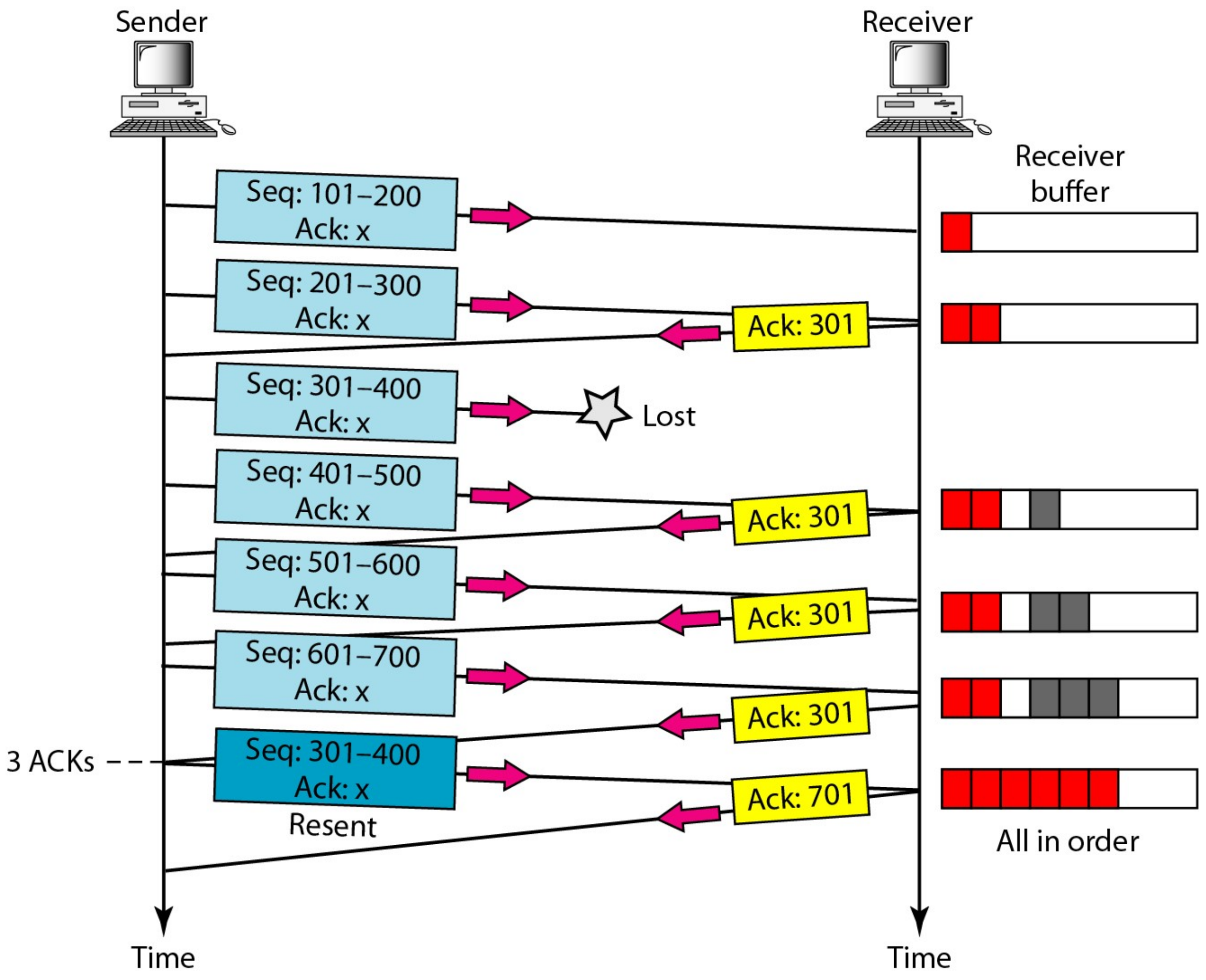
Figure 23.25 *Lost segment*



Error Control

- ***Retransmission after three duplicate ACK Segments***
- If value of RTO is large then it causes problem
- When one segment is lost, receiver receives many out-of-order segments that cannot be saved
- According to three-duplicate-ACKs rule, missing segment is transmitted immediately
- These feature is called as **fast retransmission**

Figure 23.26 *Fast retransmission*


Sender
Receiver
Receiver buffer
Seq: 101–200 Ack: x
Seq: 201–300 Ack: x
Ack: 301
Seq: 301–400 Ack: x
Lost
Seq: 401–500 Ack: x
Ack: 301
Seq: 501–600 Ack: x
Ack: 301
Seq: 601–700 Ack: x
Ack: 301
3 ACKs
Seq: 301–400 Ack: x
Resent
Ack: 701
All in order
Time
Time