



TY BTech Trimester-VII (AY 2019-2020)

Computer Science and Engineering

Disclaimer:

- Information included in these slides came from multiple sources. We have tried our best to cite the sources. Please refer to the [references](#) to learn about the sources, when applicable.
- The slides should be used only for preparing notes, academic purposes (e.g. in teaching a class), and should not be used for commercial purposes.

Unit-III

TRANSPORT LAYER

Unit-III

- Transport Layer Services: Transport layer functionalities, Sockets,
- Transport Layer Protocols: UDP, RTP, TCP: 3- way Handshake, TCP Transmission Policy (Sliding Window),
- TCP Congestion Control Algorithms: Leaky Bucket, Token Bucket, Congestion Avoidance, Quality of Service

Books Referred:

- Behrouz A. Forouzan, 'Data Communications and Networking', 5th Edition, McGraw- Hill Publishing Company
- Tanenbaum A. S., 'Computer Networks', Pearson Education , 4th Edition, 2008, ISBN-978-81-7758-165-2

Introduction

- ❏ The transport layer in the TCP/IP suite is located between the application layer and the network layer.
- ❏ It provides services to the application layer and receives services from the network layer.
- ❏ The transport layer acts as a liaison between a client program and a server program, **a process-to-process connection**.
- ❏ The transport layer is the heart of the TCP/IP protocol suite.

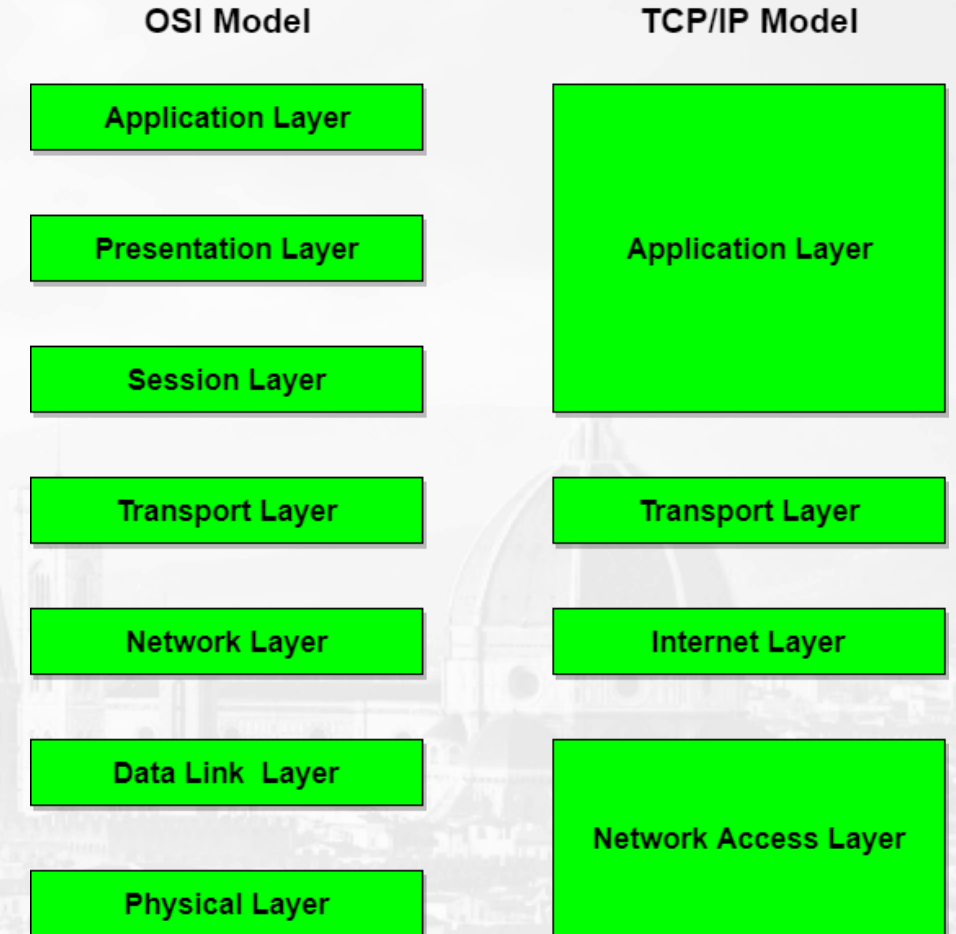


Figure: Transport Layer in OSI and TCP/IP Model

Transport Layer Services

- ❑ The transport layer provides
- ❑ efficient, reliable, and cost-effective data transmission service to its users.
- ❑ It provides a **process-to-process communication** between two application layers.
- ❑ A transport-layer protocol provides for **logical communication** between application processes running on different hosts.
- ❑ Host-to-host communication vs. process-to-process communication.

Transport Layer Services (Cntd.)

- ❑ **Process to process delivery –**
- ❑ **End-to-end Connection between hosts –**
- ❑ **Multiplexing and Demultiplexing –**
- ❑ **Congestion Control –**
 - ❑ **open loop congestion control**
 - ❑ **closed loop congestion control**
- ❑ **Data integrity and Error correction –**
- ❑ **Flow control –**

Transport Primitives

- Transport entity -
- The (logical) relationship of the network, transport, and application layers is illustrated in Figure.
- On the sending side, the transport layer converts the application-layer messages it receives from a sending application process into transport-layer packets, known as transport-layer **segments** in Internet terminology.

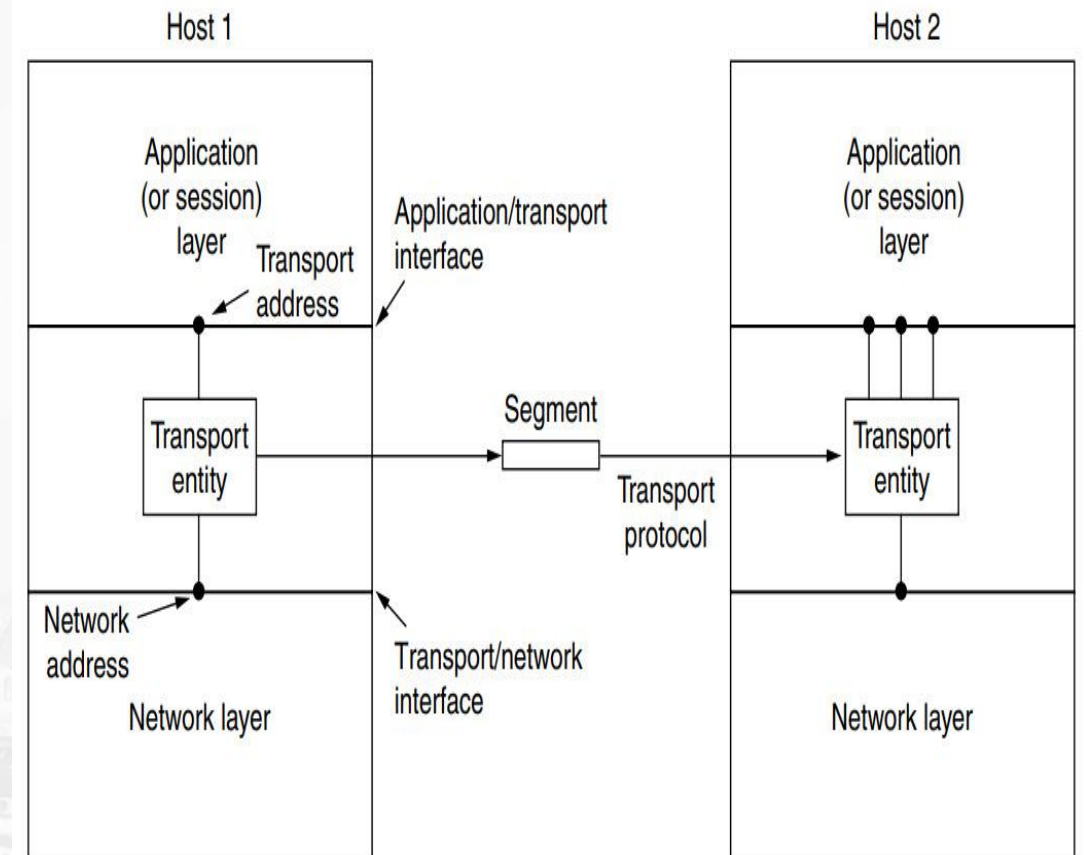


Figure: The (logical) relationship of the network, transport, and application layers

Transport Primitives (Cntd.)

- a segments.
- a transport-layer header.
- a network-layer packet (a datagram).

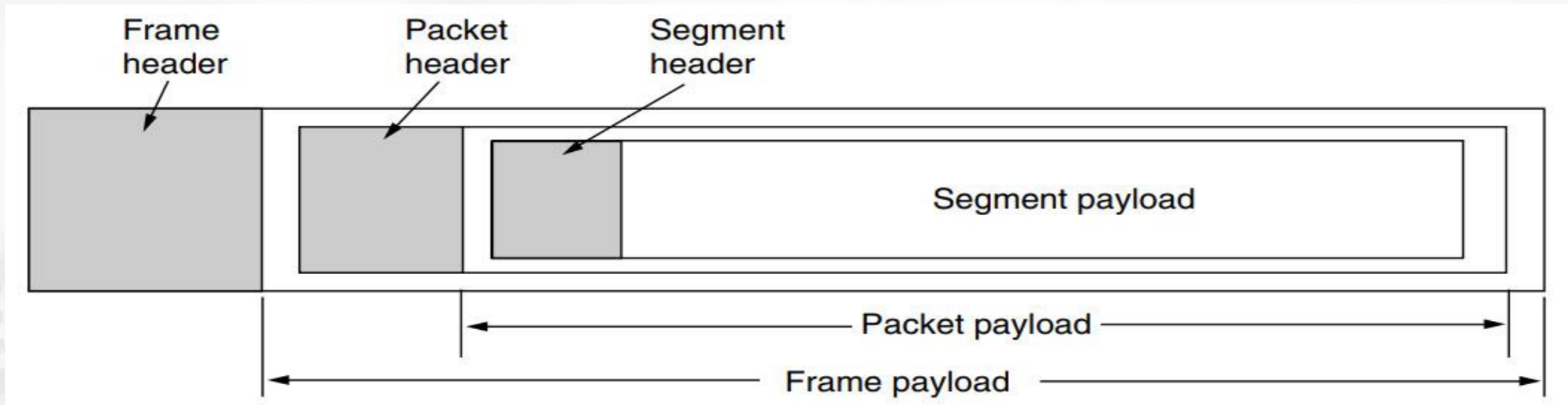


Figure: Nesting of segments, packets, and frames.

Transport Primitives (Cntd.)

- ❏ To allow users to access the transport service, the transport layer must provide, **a transport service interface**.
- ❏ The transport service is similar to the network service, but there are also some important differences. The **main difference** is that the network service is intended to model the service offered by real networks, warts and all. Real networks can lose packets, so the network service is generally unreliable. The connection-oriented transport service, in contrast, is reliable.
- ❏ Of course, real networks are not error-free, but that is precisely the **purpose of the transport layer—to provide a reliable service on top of an unreliable network**.
- ❏ As an aside, the transport layer can also provide unreliable (datagram) service.

Transport Primitives (Cntd.)

Transport Protocols –

- Internet Protocol (IP) provides a packet delivery service across an internet
- However, IP cannot distinguish between multiple processes (applications) running on the same computer
- Fields in the IP datagram header identify only computers
- A protocol that allows an application to serve as an end-point of communication is known as a transport protocol or an end-to-end protocol
- The TCP/IP protocol suite provides two transport protocols:
 1. The User Datagram Protocol (UDP)
 2. The Transmission Control Protocol (TCP)

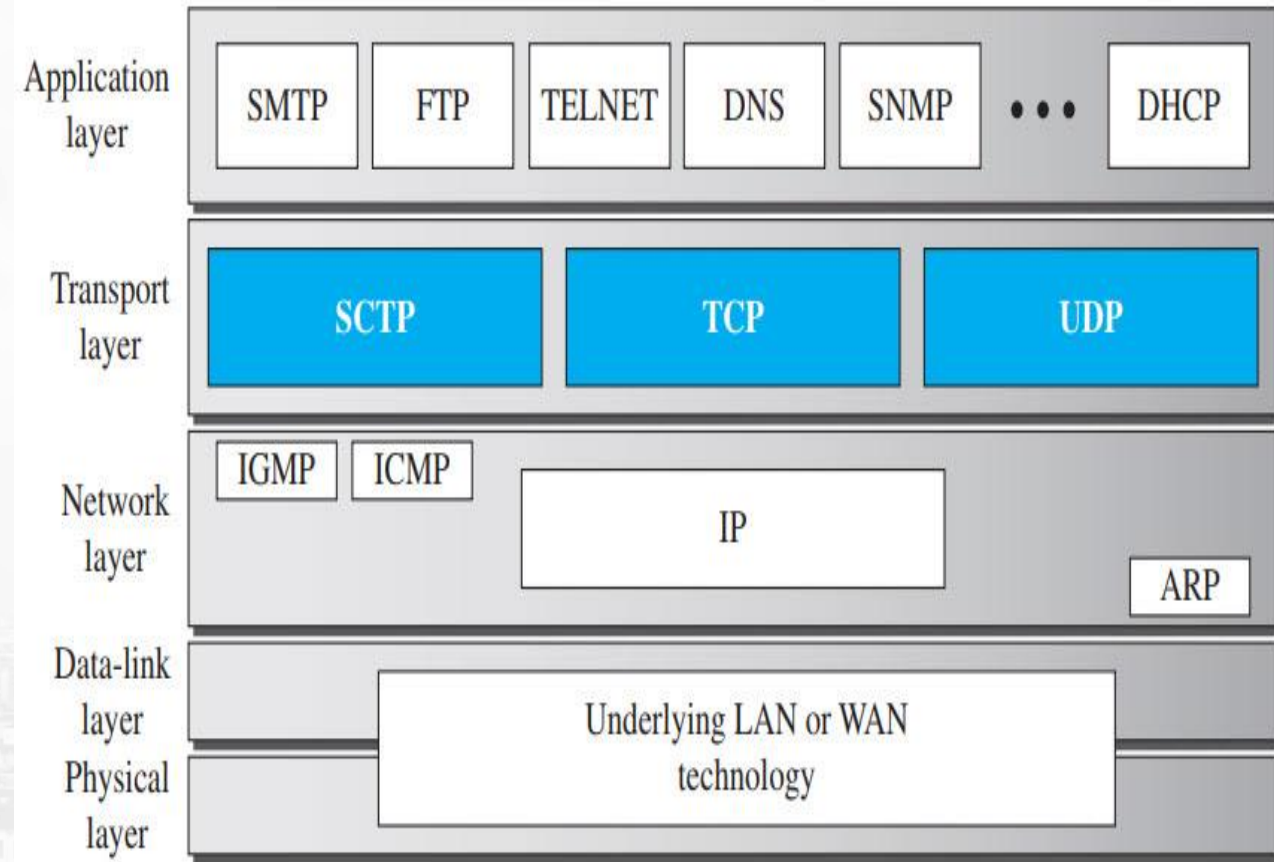


Figure: Position of transport-layer protocols in the TCP/IP protocol suite

Transport Primitives (Cntd.)

Transport Protocols Services –

- Each protocol provides a different type of service and should be used appropriately.

- ✓ UDP

UDP is an unreliable connectionless transport-layer protocol used for its simplicity and efficiency in applications where error control can be provided by the application-layer process.

- ✓ TCP

TCP is a reliable connection-oriented protocol that can be used in any application where reliability is important.

- ✓ SCTP

SCTP is a new transport-layer protocol that combines the features of UDP and TCP.

Transport Layer Vs. Network Layer

- ❑ **IP provides logical communication between hosts.**
- ❑ **The IP service model is a best-effort delivery service. IP is said to be an unreliable service.**
- ❑ **On The other hand, transport layer protocols like UDP and TCP have The most fundamental responsibility to extend IP's delivery service between two end systems to a delivery service between two processes running on the end systems. Extending host-to-host delivery to process-to-process delivery is called transport-layer multiplexing and demultiplexing.**
- ❑ **UDP and TCP also provide integrity checking by including error detection fields in their segments' headers.**

Port Numbers

- On a TCP/IP network the IP address identifies the device e.g. computer.
- However an IP address alone is not sufficient for running network applications, as a computer can run multiple applications and/or services.
- Just as the IP address identifies the computer, The network **port** identifies the application or service running on the computer.

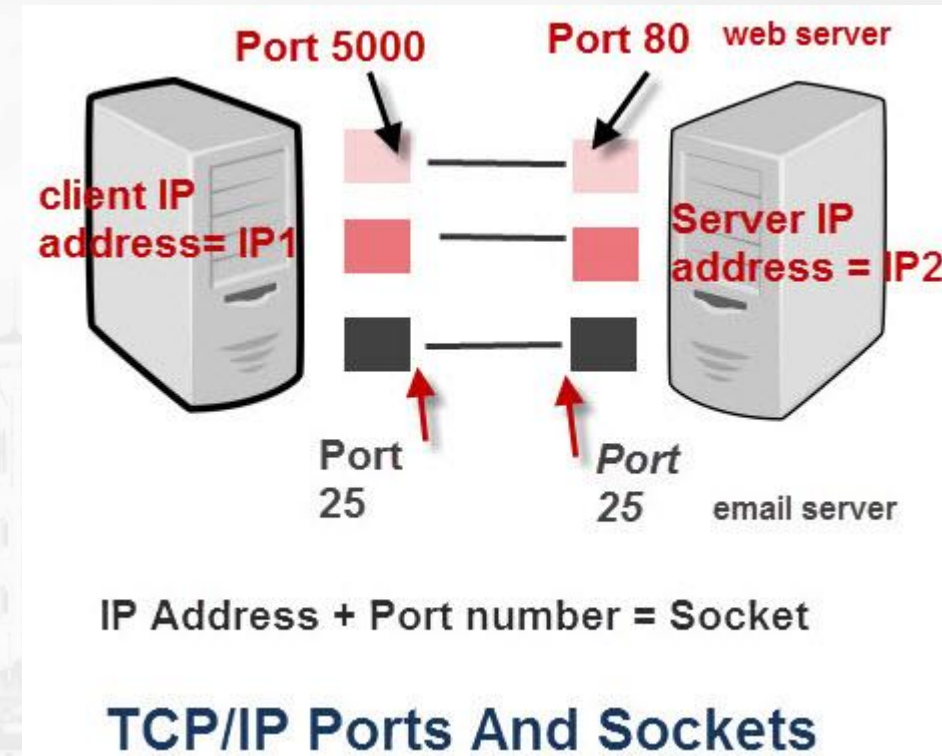


Figure: TCP/IP ports and sockets

Port Numbers (Cntd.)

- ❏ A transport-layer protocol usually has several responsibilities. One is to create a process-to-process communication; these protocols use port numbers to accomplish this.
- ❏ Port numbers provide end-to-end addresses at the transport layer and allow multiplexing and demultiplexing at this layer, just as IP addresses do at the network layer.

Port Numbers (Cntd.)

- 📖 **Analogy** – Street address and Flat numbers in residential address.
- 📖 A port number uses **16 bits** and so can therefore **have a value from 0 to 65535** decimal.
- 📖 Port numbers are divided into ranges as follows:
 - **Port numbers 0-1023 – Well known ports** – e.g. Web servers normally use port 80 and SMTP servers use port 25 (see previous diagram).
 - **Ports 1024-49151 – Registered Port.**
 - **Ports 49152-65535 – Dynamic Ports** – These are used by client programs. Also known as **ephemeral ports**.
- 📖 Following figure gives some common port numbers being used.

Port Numbers (Cntd.)

Port	Protocol	UDP	TCP	SCTP	Description
7	Echo	✓	✓	✓	Echoes back a received datagram
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
21	FTP-21		✓	✓	File Transfer Protocol
23	TELNET		✓	✓	Terminal Network
25	SMTP		✓	✓	Simple Mail Transfer Protocol
53	DNS	✓	✓	✓	Domain Name Service
67	DHCP	✓	✓	✓	Dynamic Host Configuration Protocol
69	TFTP	✓	✓	✓	Trivial File Transfer Protocol
80	HTTP		✓	✓	HyperText Transfer Protocol
111	RPC	✓	✓	✓	Remote Procedure Call
123	NTP	✓	✓	✓	Network Time Protocol
161	SNMP-server	✓			Simple Network Management Protocol
162	SNMP-client	✓			Simple Network Management Protocol

Figure: Some well-known ports used with UDP and TCP

Port Numbers (Cntd.)

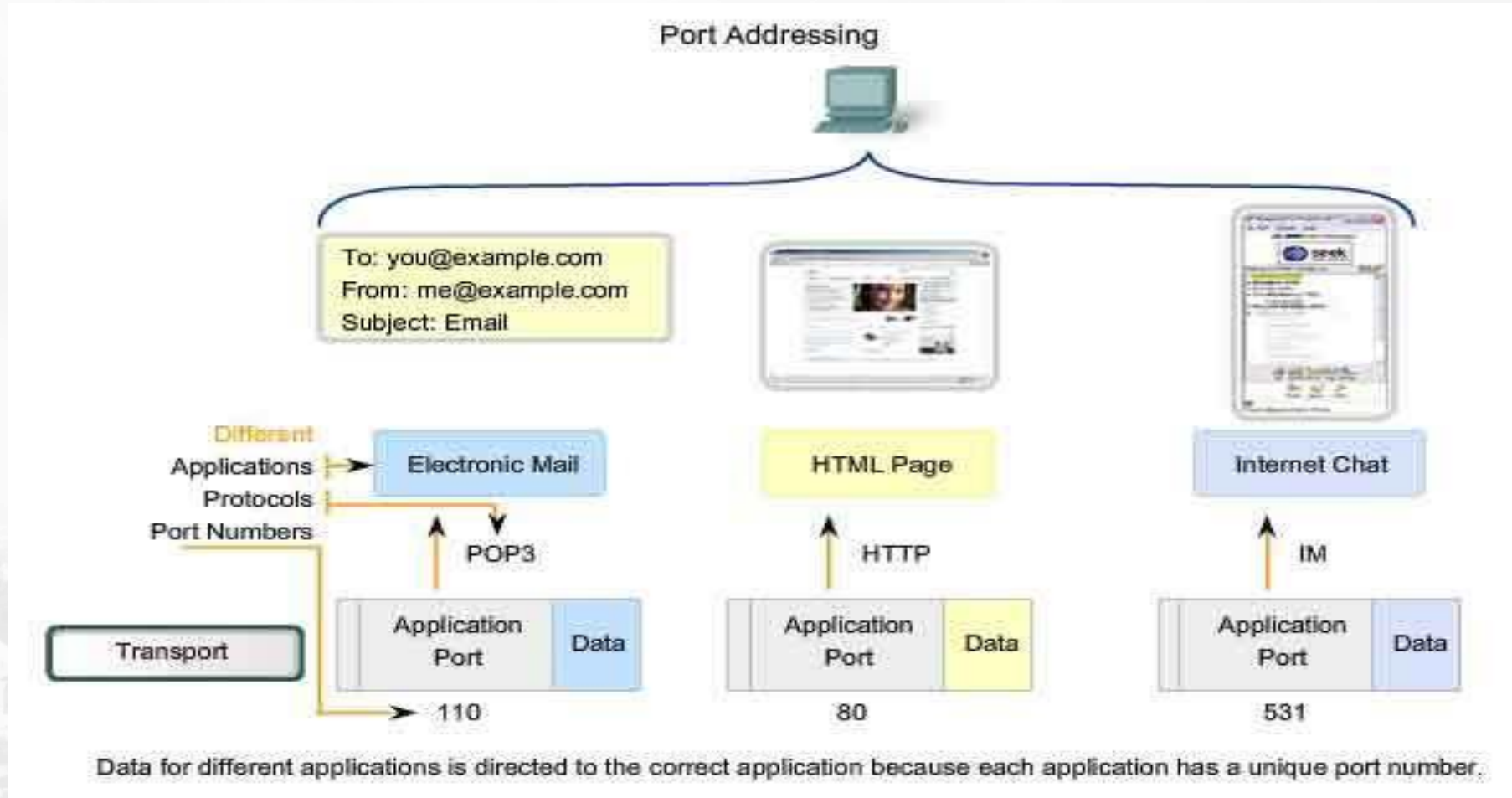



Figure: Port Addressing

Sockets

 **A process sends messages into, and receives messages from, the network through a software interface called a socket.**

 **Figure illustrates socket communication between two processes that communicate over the Internet. (Figure assumes that the underlying transport protocol used by the processes is the Internet's TCP protocol.)**

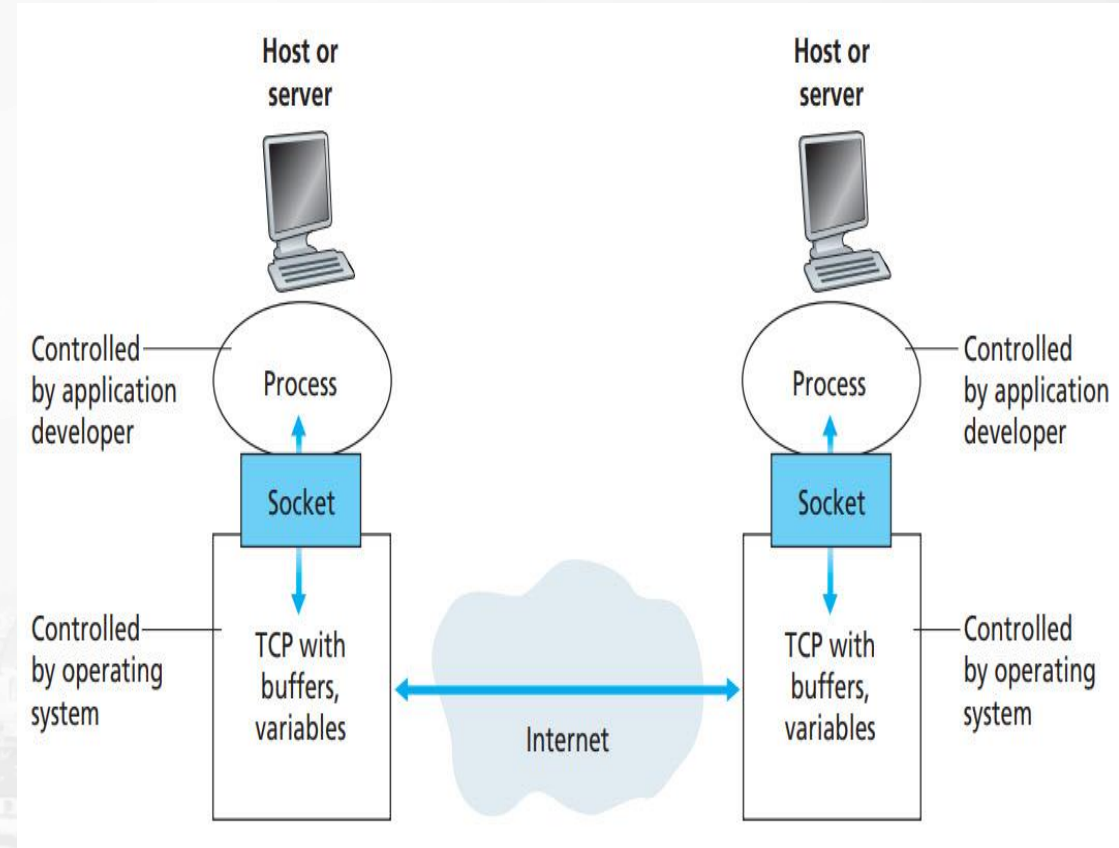
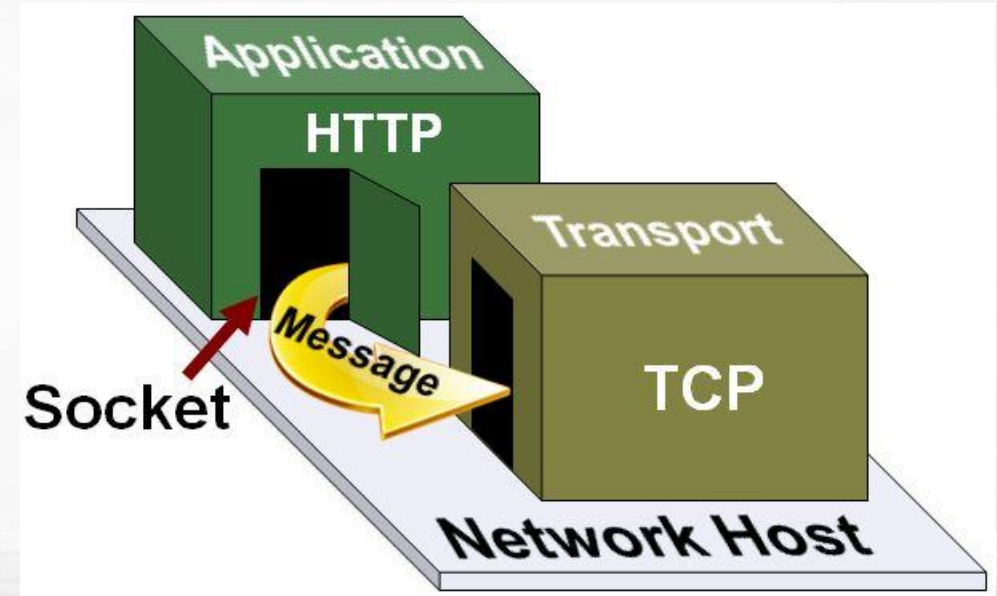


Figure: Application processes, sockets, and underlying transport protocol

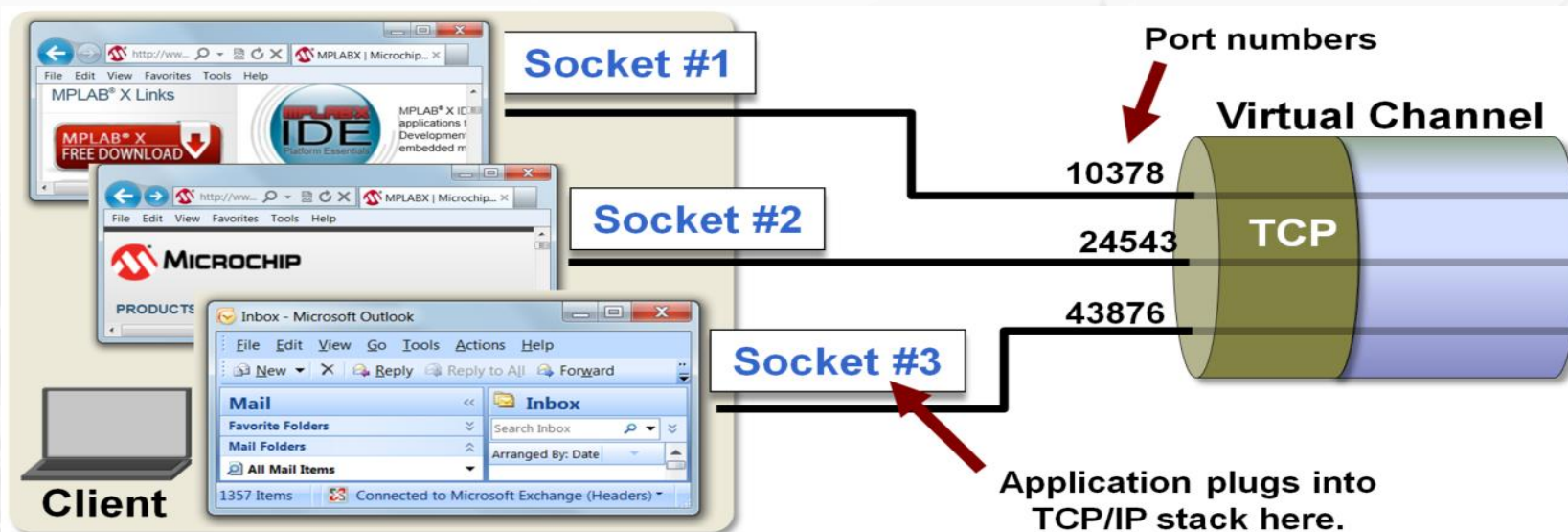
Sockets (Cntd.)

- ❏ A **socket** is a software concept for a connection. Sockets enable applications to connect to a Transmission Control Protocol/Internet Protocol (TCP/IP) network.
- ❏ An application running on a host creates a socket or doorway to connect with an application on another host. Messages pass through this socket or doorway.



Sockets (Cntd.)

- ❏ Sockets enable virtual TCP or UDP communication channels between hosts.
- ❏ This example shows three applications requiring three TCP communication channels: Two channels for each of the two web browsers acting as HTTP clients, and one for the email application acting as an SMTP client.



Sockets (Cntd.)

Berkeley Socket:

- 1. Berkeley sockets is an application programming interface (API) for Internet sockets and UNIX domain sockets.**
- 2. It is used for inter-process communication (IPC).**
- 3. It is commonly implemented as a library of linkable modules.**
- 4. It originated with the 4.2BSD UNIX released in 1983.**

Sockets (Cntd.)

Primitives used in Berkeley Socket:

Primitives	Meaning
SOCKET	Create a New Communication Endpoint.
BIND	Attach a Local Address to a SOCKET.
LISTEN	Shows the Willingness to Accept Connections.
ACCEPT	Block the Caller until a Connection Attempts Arrives.
CONNECT	Actively Attempt to Establish a Connection.
SEND	Send Some Data over Connection.
RECEIVE	Receive Some Data from the Connection.
CLOSE	Release the Connection.

Multiplexing & Demultiplexing

- ❏ Whenever an entity accepts items from more than one source, this is referred to as **multiplexing** (many to one); whenever an entity delivers items to more than one source, this is referred to as **demultiplexing** (one to many).
- ❏ **Objective** is to extend the host-to-host delivery service provided by the network layer to a process-to-process delivery service for applications running on the hosts.
- ❏ The transport layer at the source performs multiplexing; the transport layer at the destination performs demultiplexing. (See Figure)

Multiplexing & Demultiplexing (Cntd.)

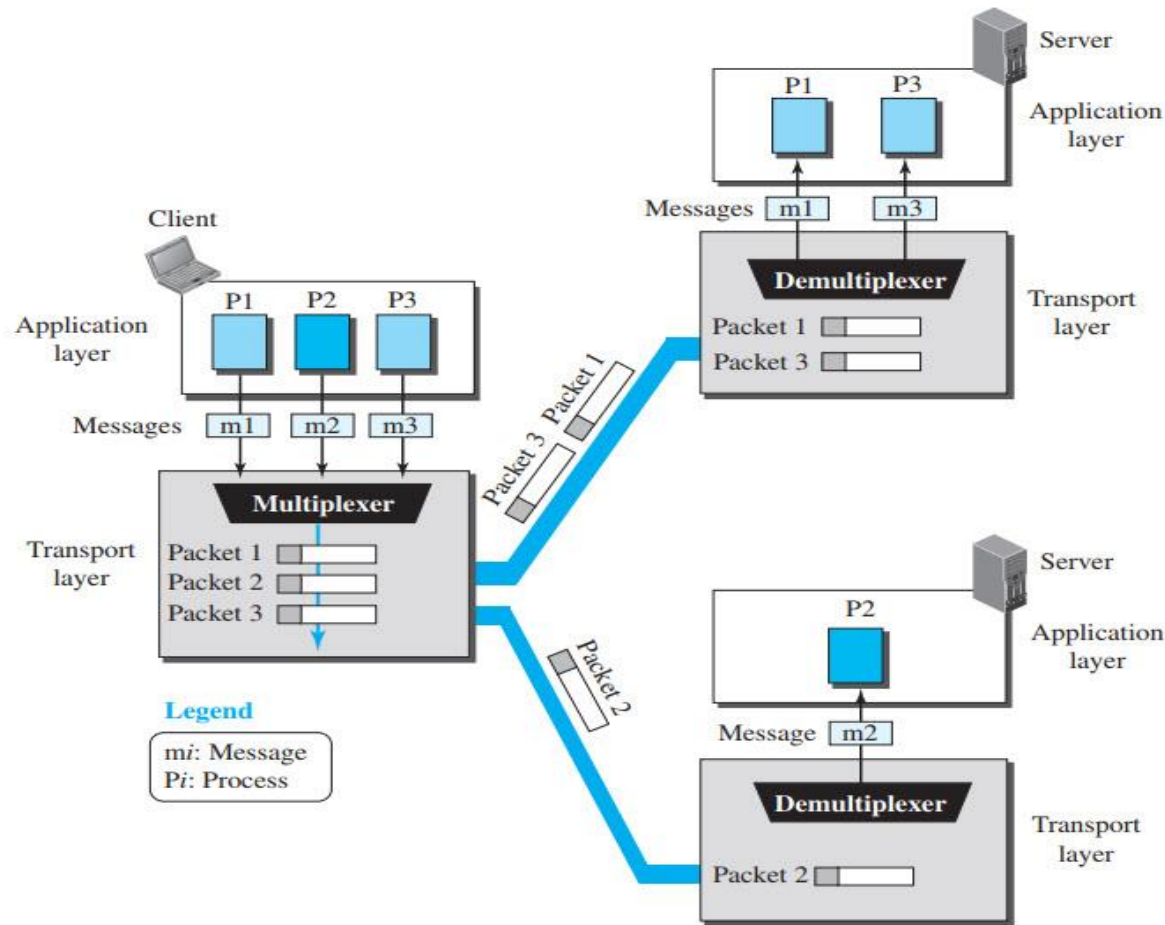


Figure: Multiplexing and demultiplexing

Multiplexing & Demultiplexing (Cntd.)

❏ Transport-layer multiplexing requires

1. that sockets have unique identifiers, and
2. that each segment have special fields that indicate the socket to which the segment is to be delivered.

❏ These special fields, are the **source port number field** and the **destination port number field**. (This port addressing concept we have discussed earlier. The UDP and TCP segments have other fields as well.)

❏ **How the transport layer could implement the demultiplexing service?**

Connectionless and Connection Oriented Protocol

- A transport-layer protocol, like a network-layer protocol, can provide **two types of services: connectionless and connection-oriented**.
- The nature of these services at the transport layer, however, is different from the ones at the network layer.
 - At the **network layer**, a connectionless service may mean different paths for different datagrams belonging to the same message; connection-oriented means dedicated path established before communication initiated.
 - At the **transport layer**, we are not concerned about the physical paths of packets (we assume a logical connection between two transport layers). Connectionless service at the transport layer means independency between packets; connection-oriented means dependency.
- The **connectionless protocol is UDP**.
- The **connection-oriented protocol is TCP**.

Connectionless and Connection Oriented Protocol (Cntd.)

Connectionless Service:

➤ (See the Figure).

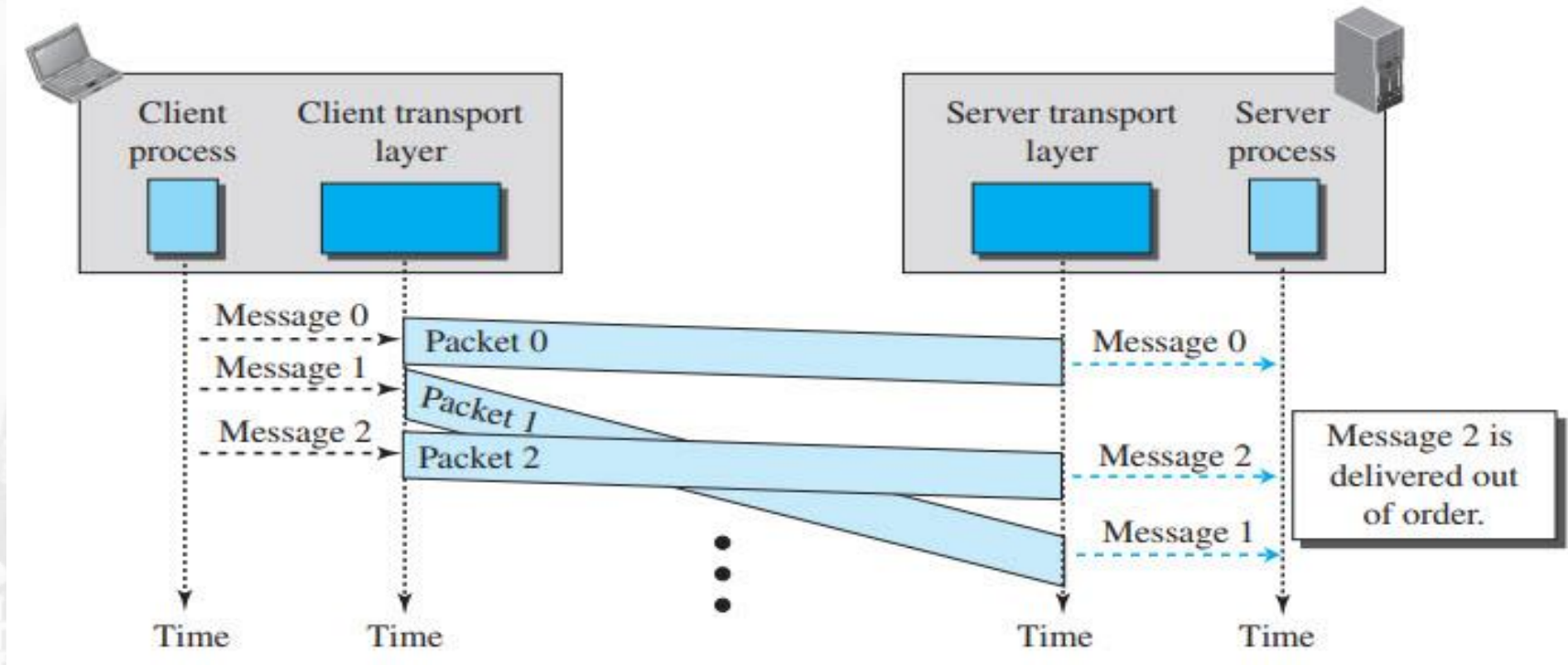


Figure: Connectionless service

Connectionless and Connection Oriented Protocol (Cntd.)

Connection-oriented Service:

➤ (See the Figure).

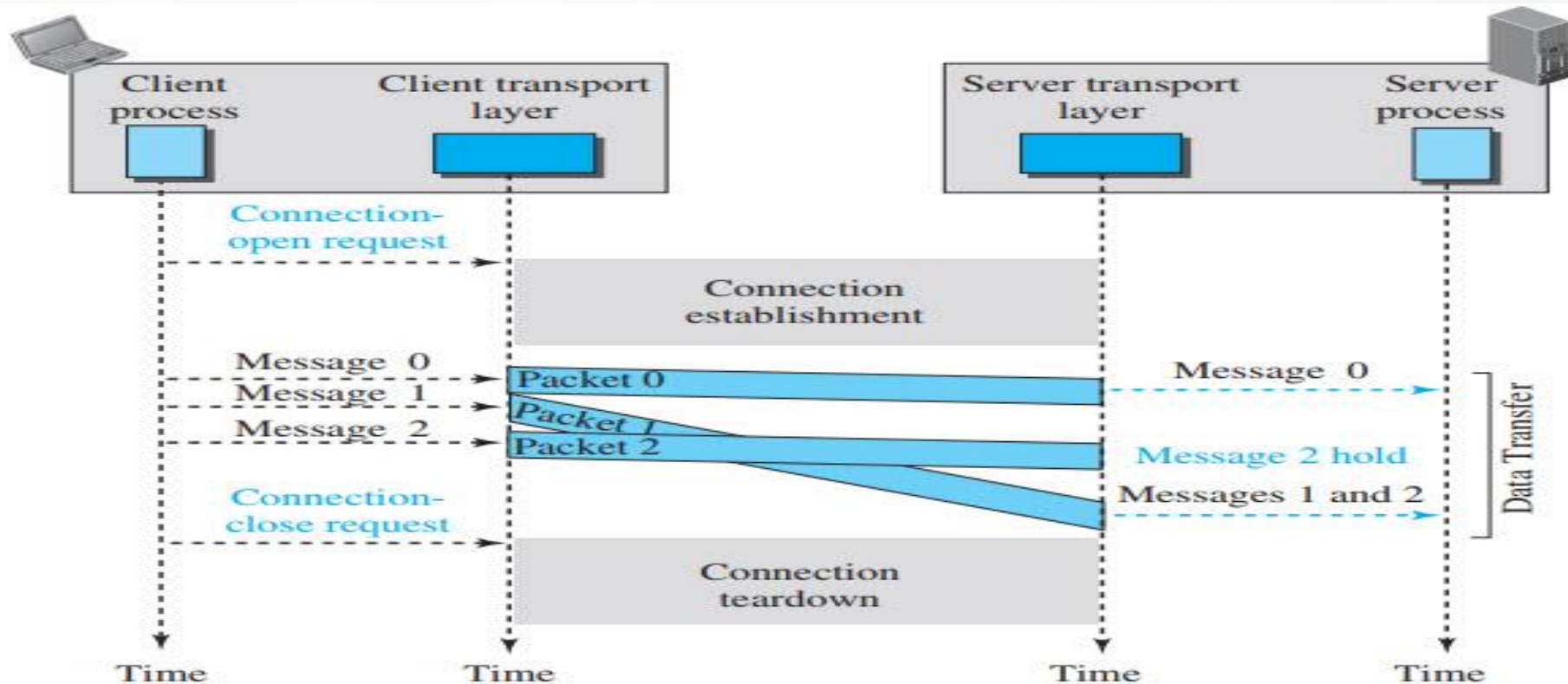


Figure: Connection-oriented service

Connectionless and Connection Oriented Protocol (Cntd.)

Note:
The colored arrow shows the starting state.

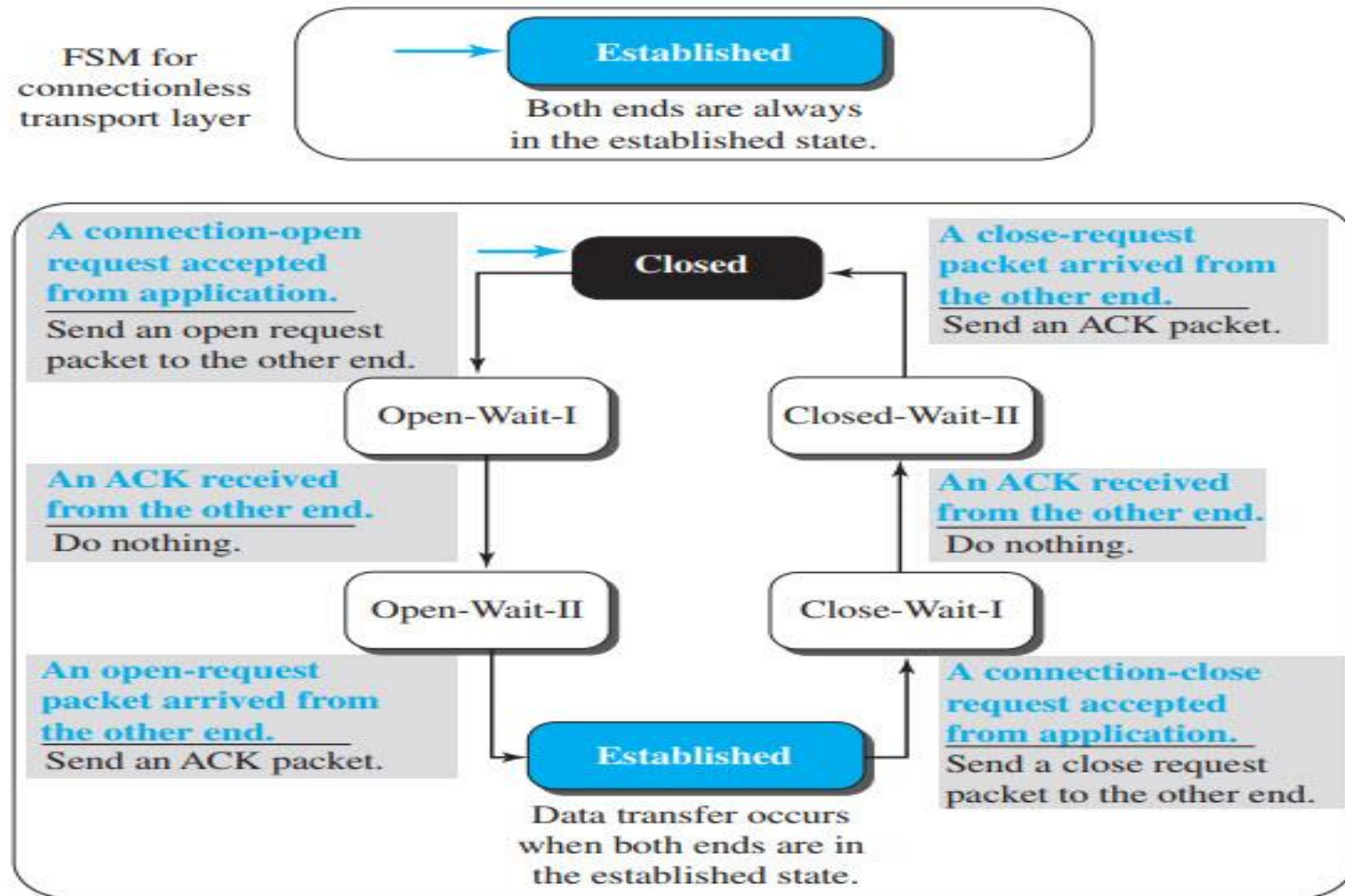


Figure: Connectionless and connection-oriented service represented as FSMs

Elements of Transport Protocols

🖥 In some ways, transport protocols resemble the data link protocols we have already studied.

🖥 However, **significant differences** between the two also exist. These differences are **due to major dissimilarities** between the environments in which the two protocols operate.

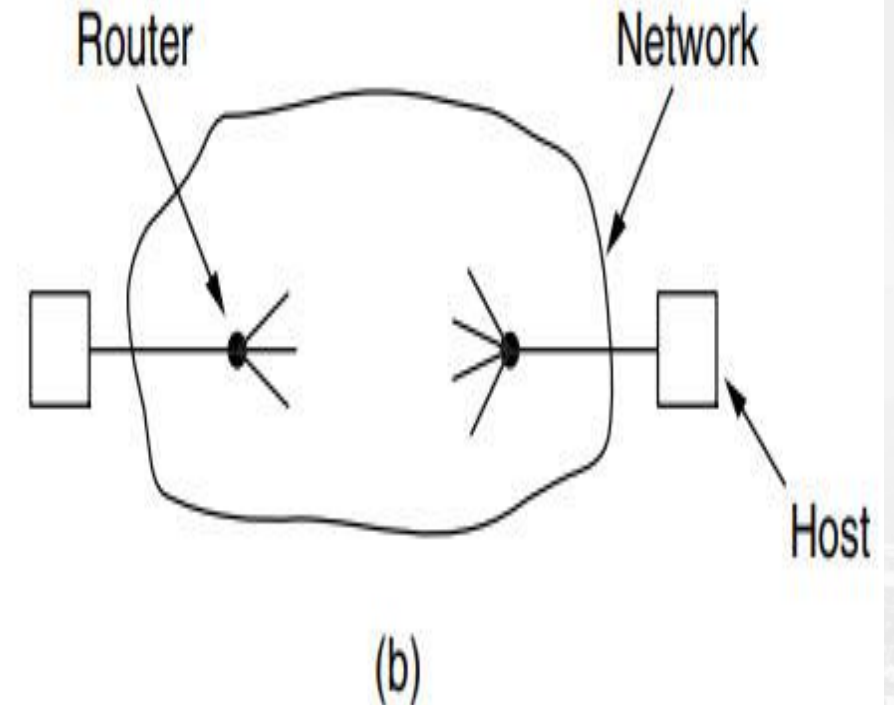
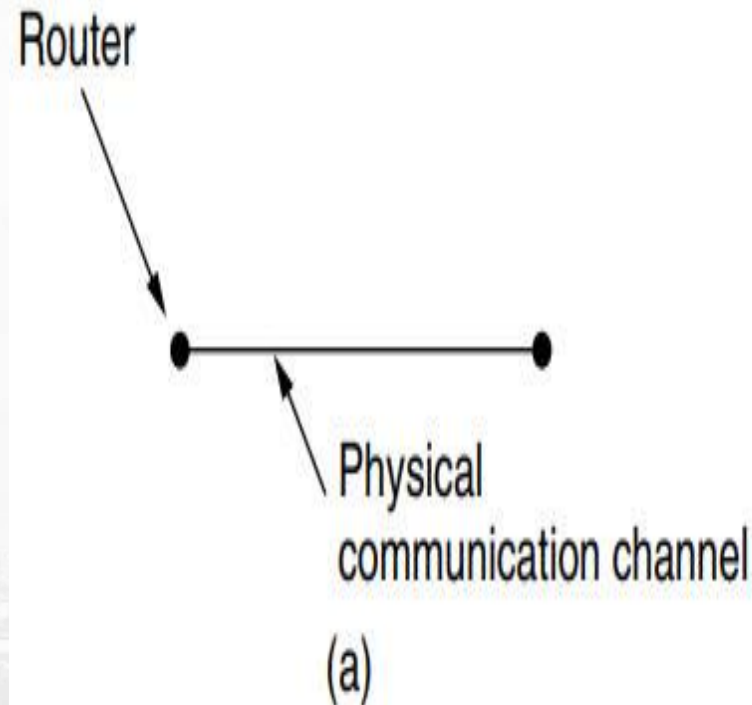


Figure: (a) Environment of the data link layer.
(b) Environment of the transport layer.

Elements of Transport Protocols (Cntd.)

Other Differences -

- **User Oriented:**
- **Guarantee Service:**
- **Addressing:**
- **Connection establishment:**
- **Storage capacity of the subnet:**
- **We need dynamic flow control mechanism:**
- **Addressing:**

Elements of Transport Protocols (Cntd.)

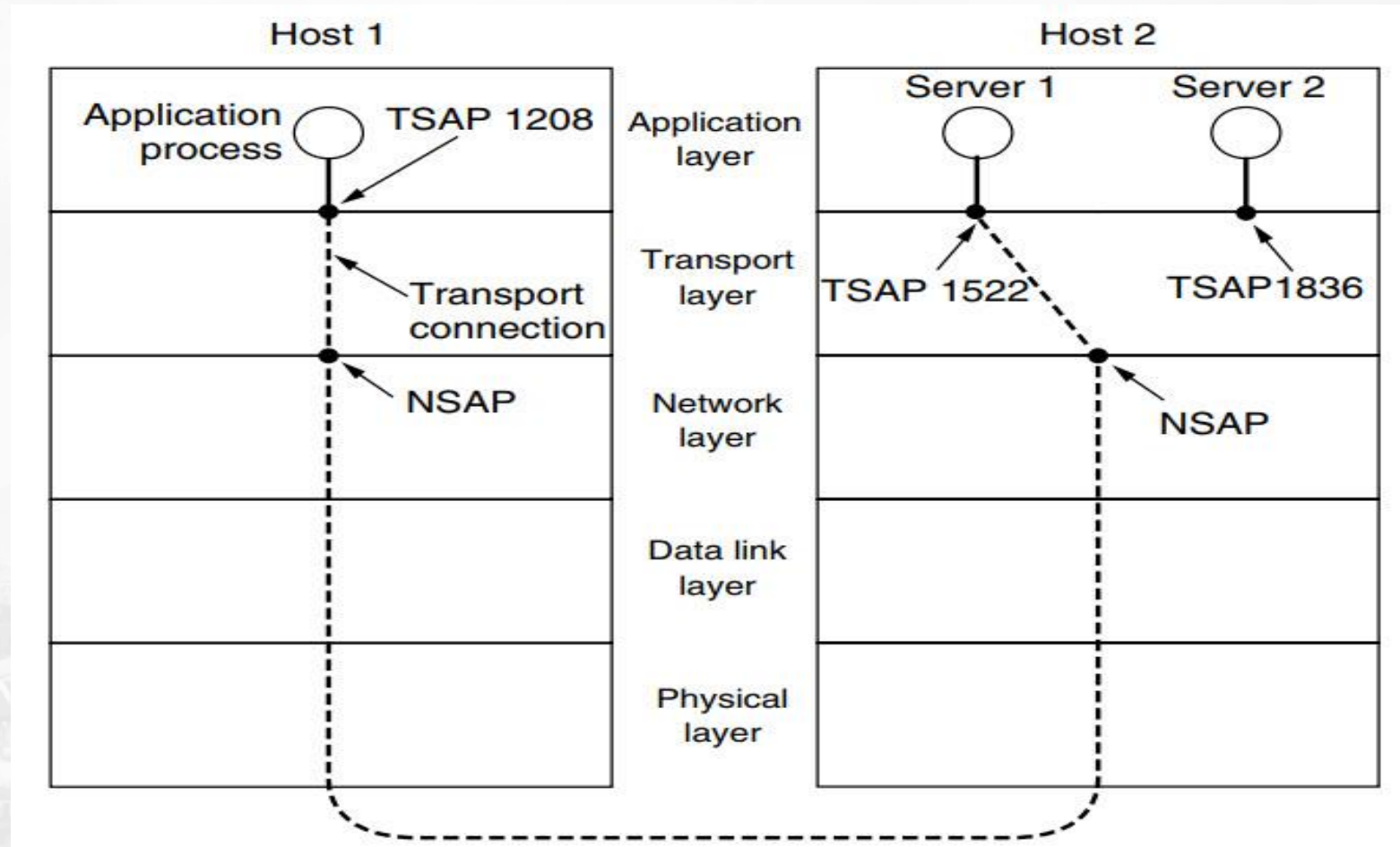


Figure: TSAPs, NSAPs, and transport connections.

Elements of Transport Protocols (Cntd.)

Connection Establishment: (Works only for TCP)

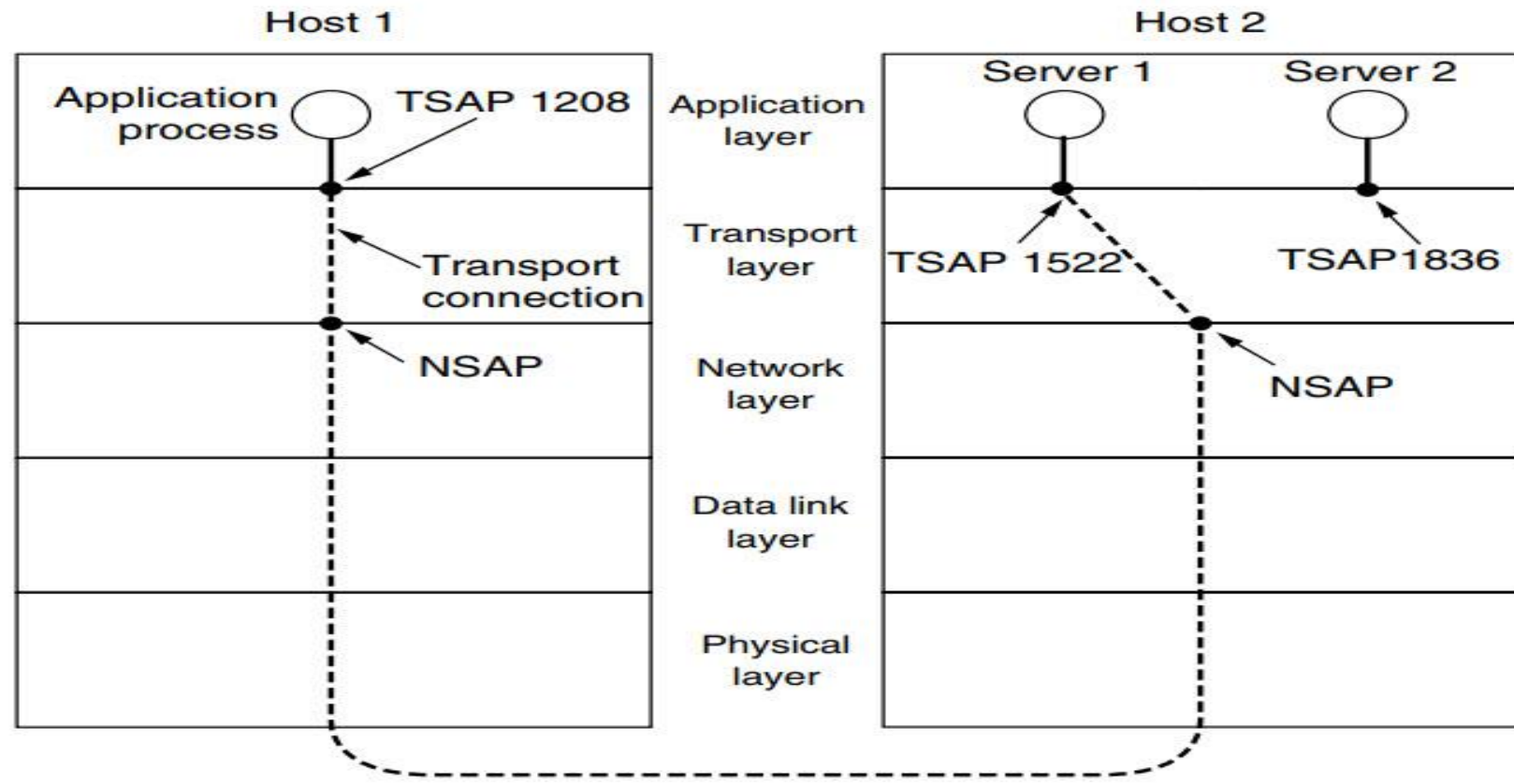


Figure: TSAPs, NSAPs, and transport connections.

User Datagram Protocol [UDP]

- ❑ Connectionless Transport Protocol.
- ❑ UDP transmits segments consisting of an 8-byte fixed size header followed by the payload.
- ❑ The header is shown below

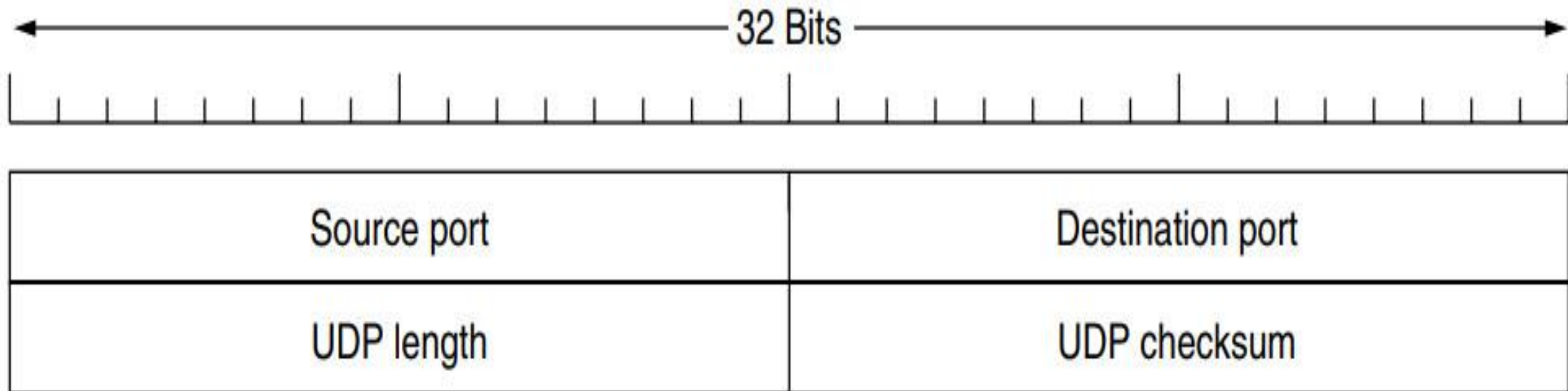


Figure: The UDP header.

User Datagram Protocol [UDP] (Cntd.)

 **Question:** The following is the content of a UDP header in hexadecimal format.

CB84000D001C001C

- What is the source port number?
- What is the destination port number?
- What is the total length of the user datagram?
- What is the length of the data?
- Is the packet directed from a client to a server or vice versa?
- What is the client process?

 **Solution:** Try to Solve It.....

User Datagram Protocol [UDP] (Cntd.)

Solution:

CB84000D001C001C

- The source port number is the first four hexadecimal digits (CB84)₁₆, which means that the source port number is 52100.
- The destination port number is the second four hexadecimal digits (000D)₁₆, which means that the destination port number is 13.
- The third four hexadecimal digits (001C)₁₆ define the length of the whole UDP packet as 28 bytes.
- The length of the data is the length of the whole packet minus the length of the header, or $28 - 8 = 20$ bytes.
- Since the destination port number is 13 (well-known port), the packet is from the client to the server.
- The client process is the Daytime (see slide no. 19).

User Datagram Protocol [UDP] (Cntd.)

UDP Services:

- Process to process communication – using socket address
- Connectionless services
- Flow Control – Not present
- Error Control – Not present
- Checksum – based on pseudo header, UDP header, and data (Can be Optional)
- Congestion Control – Not present
- Encapsulation & Decapsulation
- Queuing
- Multiplexing & Demultiplexing

User Datagram Protocol [UDP] (Cntd.)

UDP Applications:

- UDP is suitable for a process that requires simple request-response communication with little concern for flow and error control.
- UDP is suitable for a process with internal flow- and error-control mechanisms. For example, the Trivial File Transfer Protocol (TFTP).
- UDP is a suitable transport protocol 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).
- UDP is normally used for interactive real-time applications that cannot tolerate uneven delay between sections of a received message. E.g. multimedia applications.
- Client-server RPC is one area in which UDP is widely used.

Real-Time Transport Protocol [RTP]

- Client-server RPC is one area in which UDP is widely used. Another one is for real-time multimedia applications.
- Multimedia applications such as Internet radio, Internet telephony, music-on-demand, videoconferencing, video-on-demand, and other are reinventing more or less the same real-time transport protocol.
- Thus generic real-time transport protocol for multiple applications would be a good idea. i.e. RTP (Real-time Transport Protocol).
- Aspects of real-time transport
 - for transporting audio and video data in packets.
 - the processing that takes place, mostly at the receiver, to play out the audio and video at the right time.

Real-Time Transport Protocol [RTP] (Cntd.)

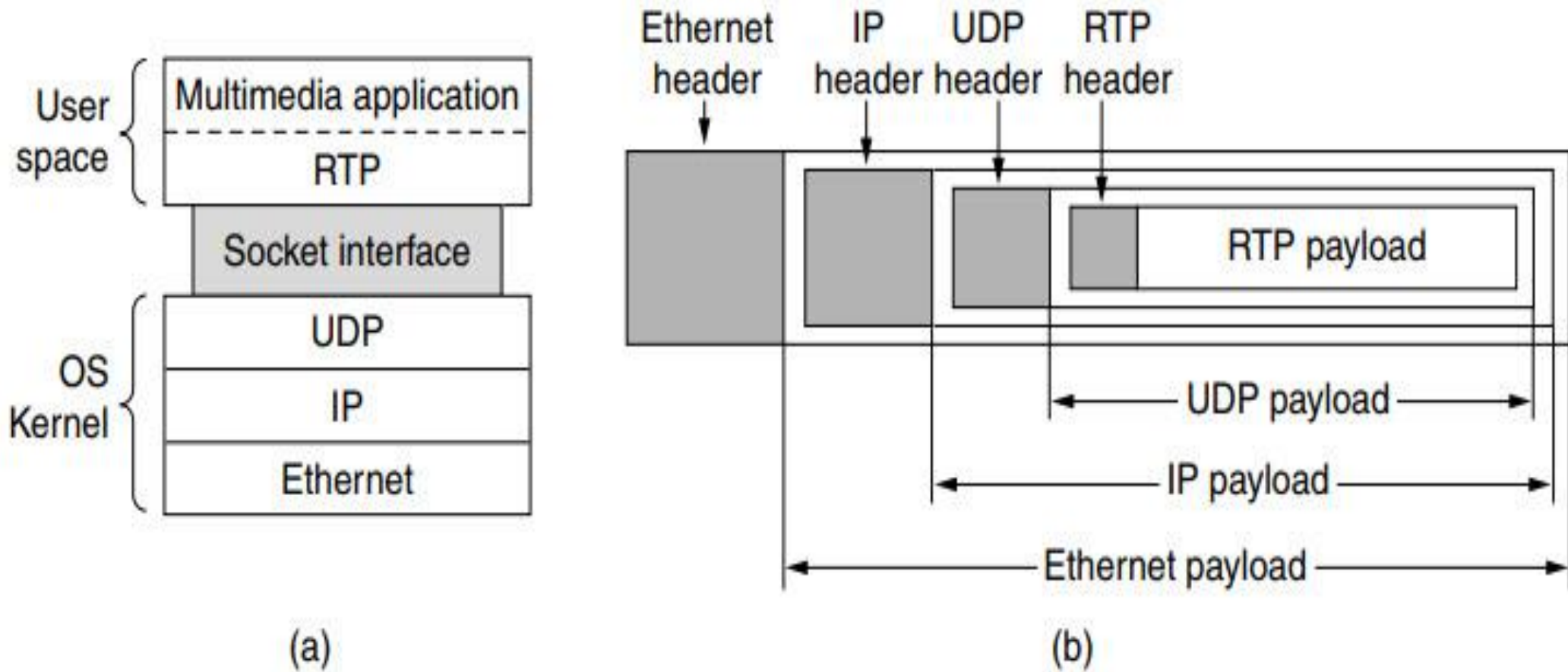
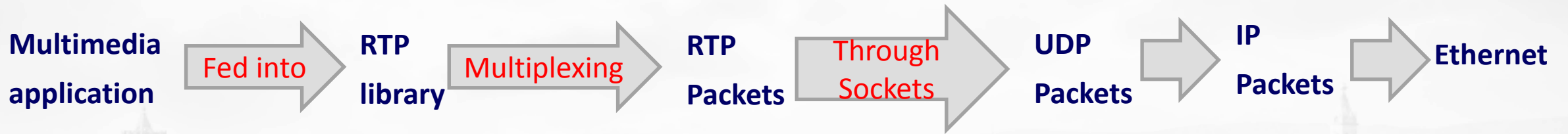


Figure: (a) The position of RTP in the protocol stack. (b) Packet nesting.

Real-Time Transport Protocol [RTP] (Cntd.)

- RTP normally runs in user space over UDP (in the operating system).

- Its works as follows



- Because of this design, it is a little hard to say which layer RTP is in. (see previous figure)
 - looks like an application protocol – because it runs in user space and is linked to the application program.
 - looks like a transport protocol – because it is a generic, application independent protocol that just provides transport facilities

Real-Time Transport Protocol [RTP] (Cntd.)

- **Basic function – multiplex several real-time data streams onto a single stream of UDP packets.**
- **Unicasting or Multicasting can be done.**
- **There are no special guarantees about delivery, and packets may be lost, delayed, corrupted, etc.**
- **The RTP format contains several features to help receivers work with multimedia information.**
 - ✓ **Sequence Numbering**
 - ✓ **Encoding for Different profiles such as audio, video etc.**
 - ✓ **Time stamping**

Real-Time Transport Protocol [RTP] (Cntd.)

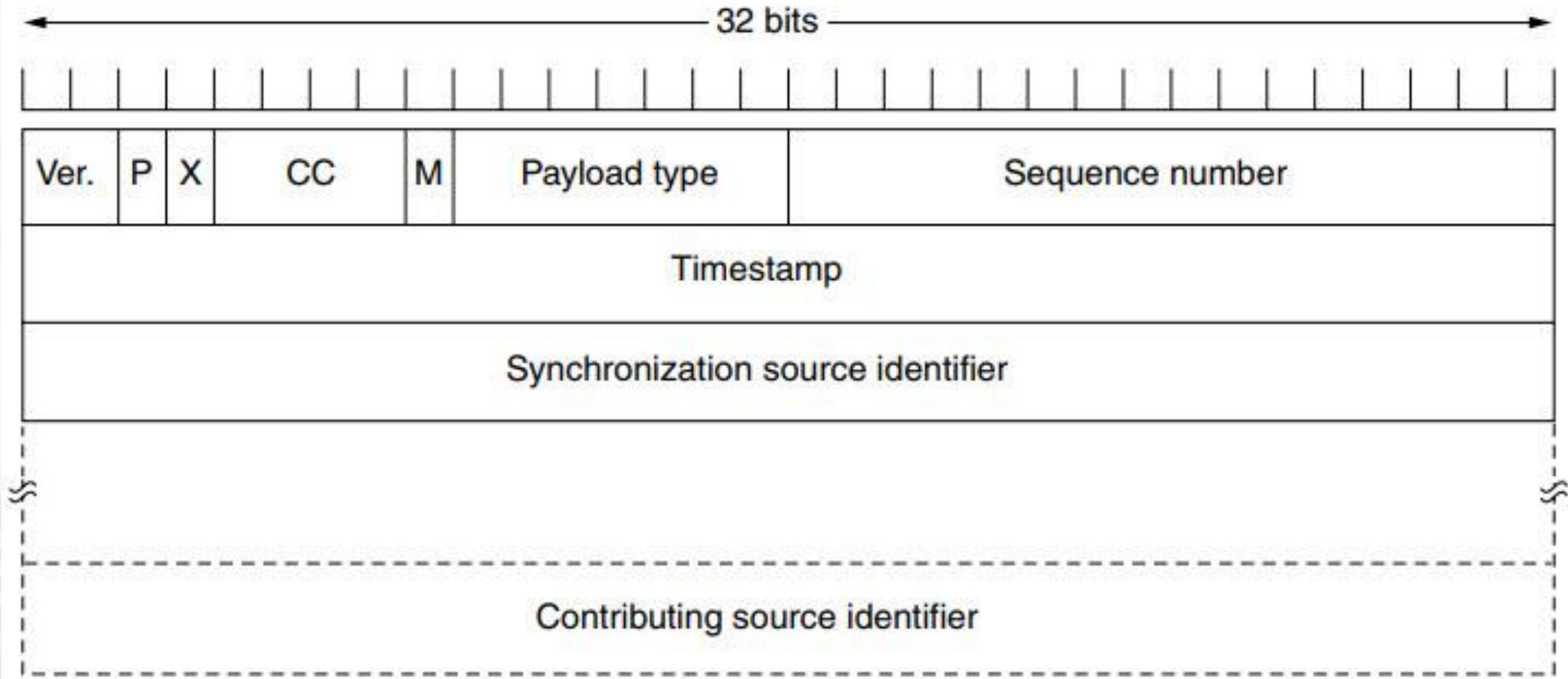


Figure: The RTP header.

Transmission Control Protocol [TCP]

- Specifically designed to provide a reliable end-to-end byte stream over an unreliable internetwork. Transmission Control Protocol (TCP) is a connection-oriented, reliable protocol.
- Was designed to dynamically adapt to properties of the internetwork and to be robust in the face of many kinds of failures.
- Connection-Oriented Service – explicitly defines connection establishment, data transfer, and connection teardown phases.
- Reliability – Combination of GBN and SR protocols is used, uses checksum (for error detection), retransmission of lost or corrupted packets, cumulative and selective acknowledgments, and timers.

Transmission Control Protocol [TCP] (Cntd.)

TCP Services:

- Process to process communication – using socket address
- Stream delivery service –
 - ✓ Sending/Receiving process to deliver/obtain data as a stream of bytes.
 - ✓ Sending and Receiving buffers
 - ✓ TCP Segments
- Full Duplex Communication
- Multiplexing and Demultiplexing – multiplexing at sender & demultiplexing at receiver. However connection need to be established for each pair of processes
- Connection-Oriented service – Connection establishment, data transfer, connection termination
- Reliable Service – use of acknowledgement.

Transmission Control Protocol [TCP] (Cntd.)

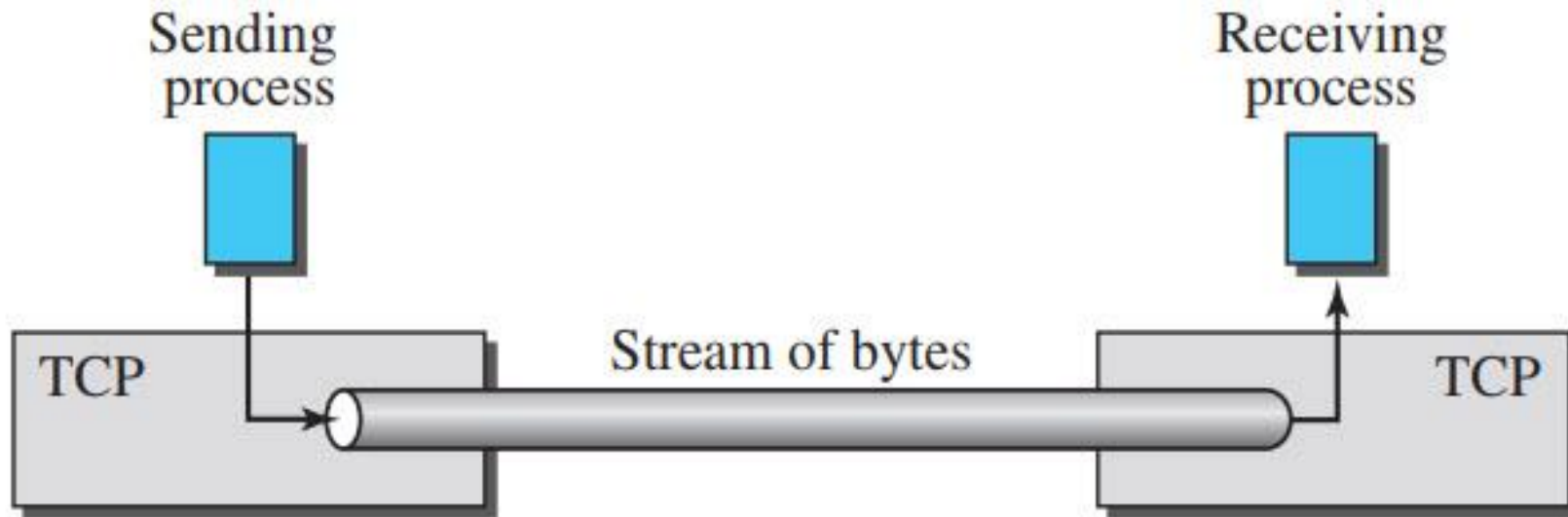


Figure: Stream delivery.

Transmission Control Protocol [TCP] (Cntd.)

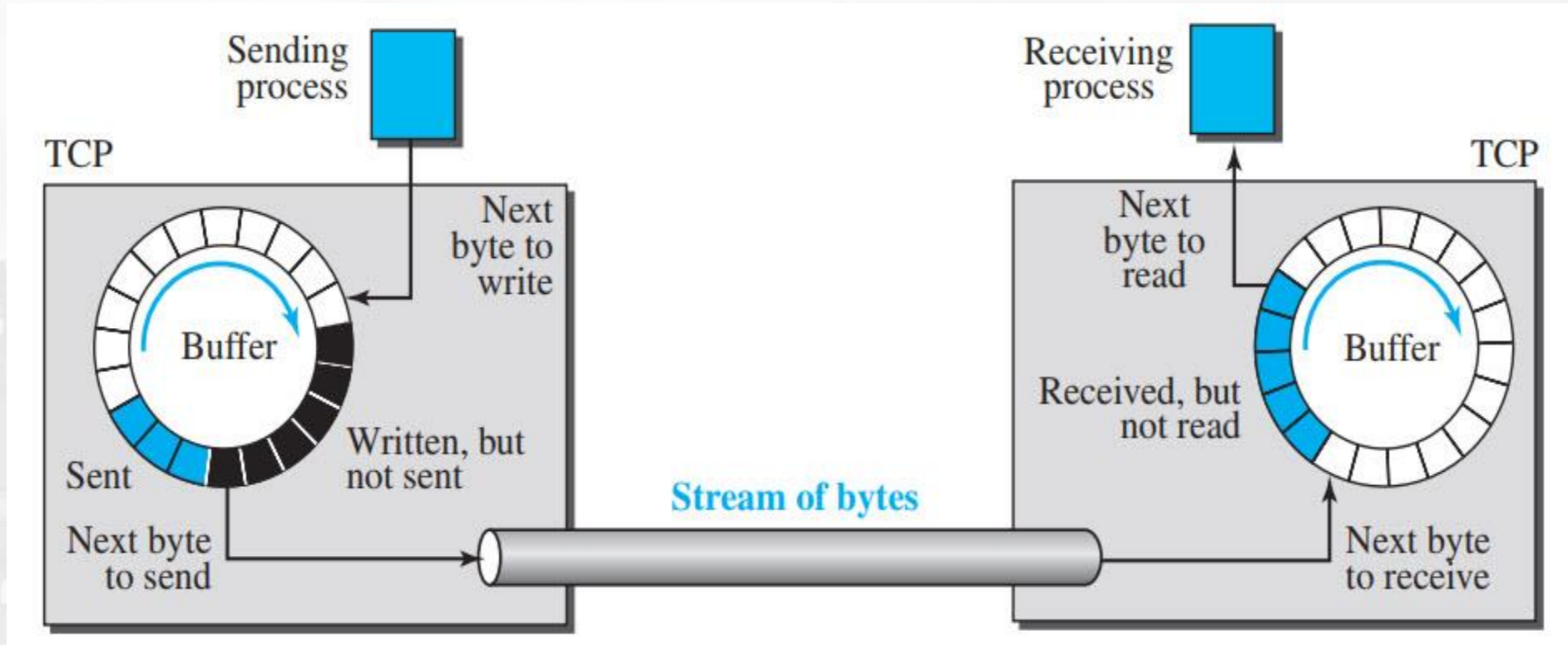


Figure: Sending and receiving buffers.

Transmission Control Protocol [TCP] (Cntd.)

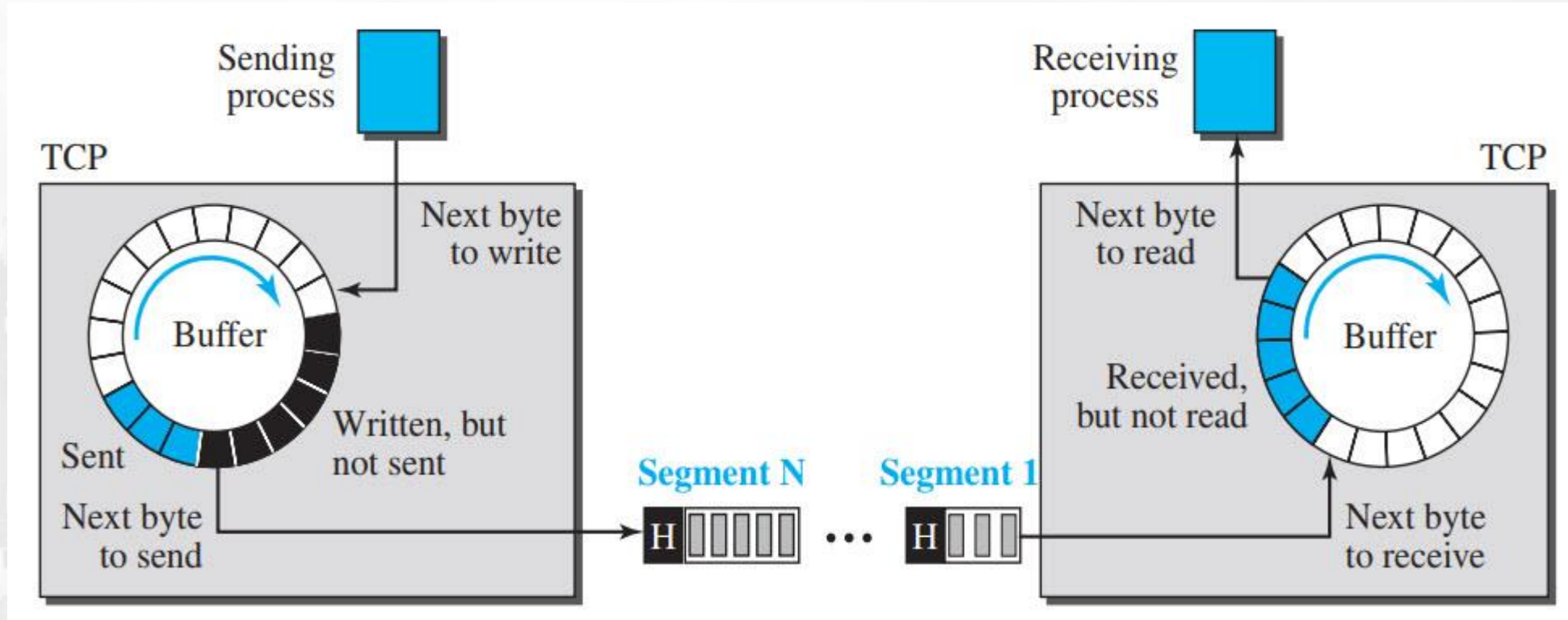


Figure: TCP segments.

Transmission Control Protocol [TCP] (Cntd.)

TCP Features:

➤ Numbering System –

✓ Byte Number –

- ✓ The bytes of data being transferred in each connection are numbered by TCP.
- ✓ Numbering is independent in each direction.
- ✓ The numbering starts with an arbitrarily generated number.

✓ Sequence Number –

- ✓ defines the number assigned to the first data byte contained in that segment.

✓ Acknowledgement Number –

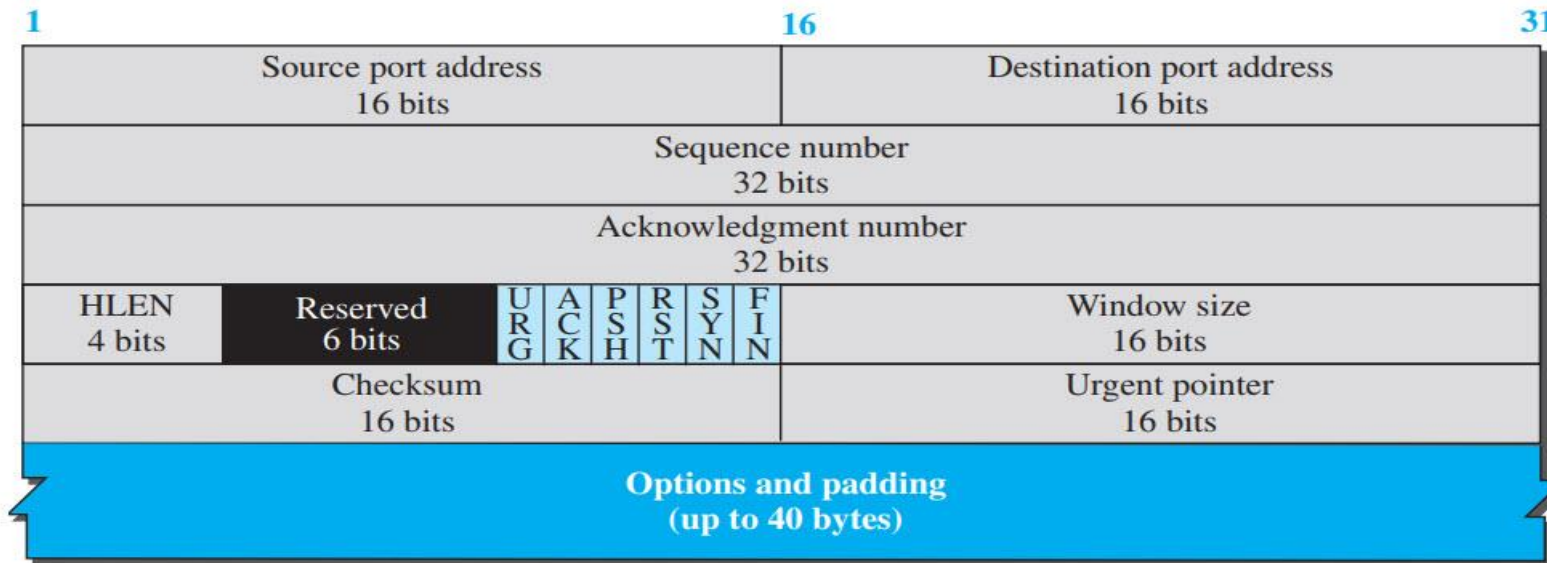
- ✓ defines the number of the next byte a party expects to receive.
- ✓ The acknowledgment number is cumulative.

Transmission Control Protocol [TCP] (Cntd.)

Segment:



a. Segment



b. Header

Figure: TCP segment format.

Transmission Control Protocol [TCP] (Cntd.)

TCP Connection:

- In TCP, connection-oriented transmission requires three phases: connection establishment, data transfer, and connection termination.

Transmission Control Protocol [TCP] (Cntd.)

Connection Establishment: Three way Handshaking

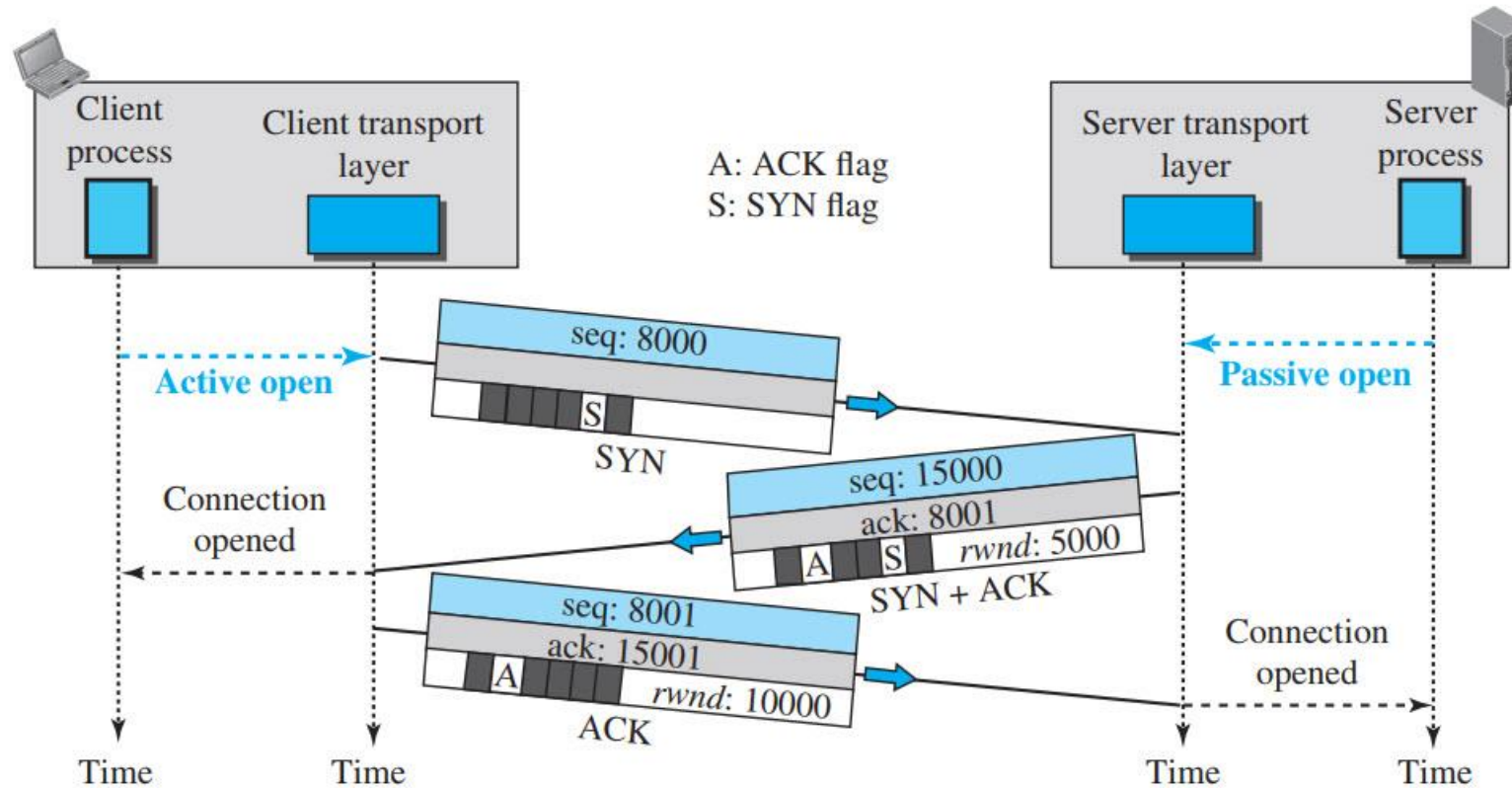


Figure: Connection establishment using three-way handshaking.

Transmission Control Protocol [TCP] (Cntd.)

Connection Establishment: SYN flooding attack

- TCP Connection establishment phase is susceptible to SYN flooding attack.
- belongs to a group of security attacks known as a denial of service attack.
- Defending against SYN flooding attacks
 - ✓ Some implementations of TCP have strategies to alleviate the effect of a SYN attack.
 - ✓ Some have imposed a limit of connection requests during a specified period of time.
 - ✓ Some try to filter out datagrams coming from unwanted source addresses.
 - ✓ Recent strategy postpone resource allocation until the server can verify that the connection request is coming from a valid IP address, by using a cookie.

Transmission Control Protocol [TCP] (Cntd.)

Data Transfer:

- **Bidirectional data transfer – data and acknowledgements can be transferred.**
- **Pushing Data:**
 - ✓ Application program on sender can request a push operation, means that the sending TCP must not wait for the window to be filled. It must create a segment and send it immediately.
- **Urgent Data:**
 - ✓ There are occasions in which an application program needs to send urgent bytes.

Transmission Control Protocol [TCP] (Cntd.)

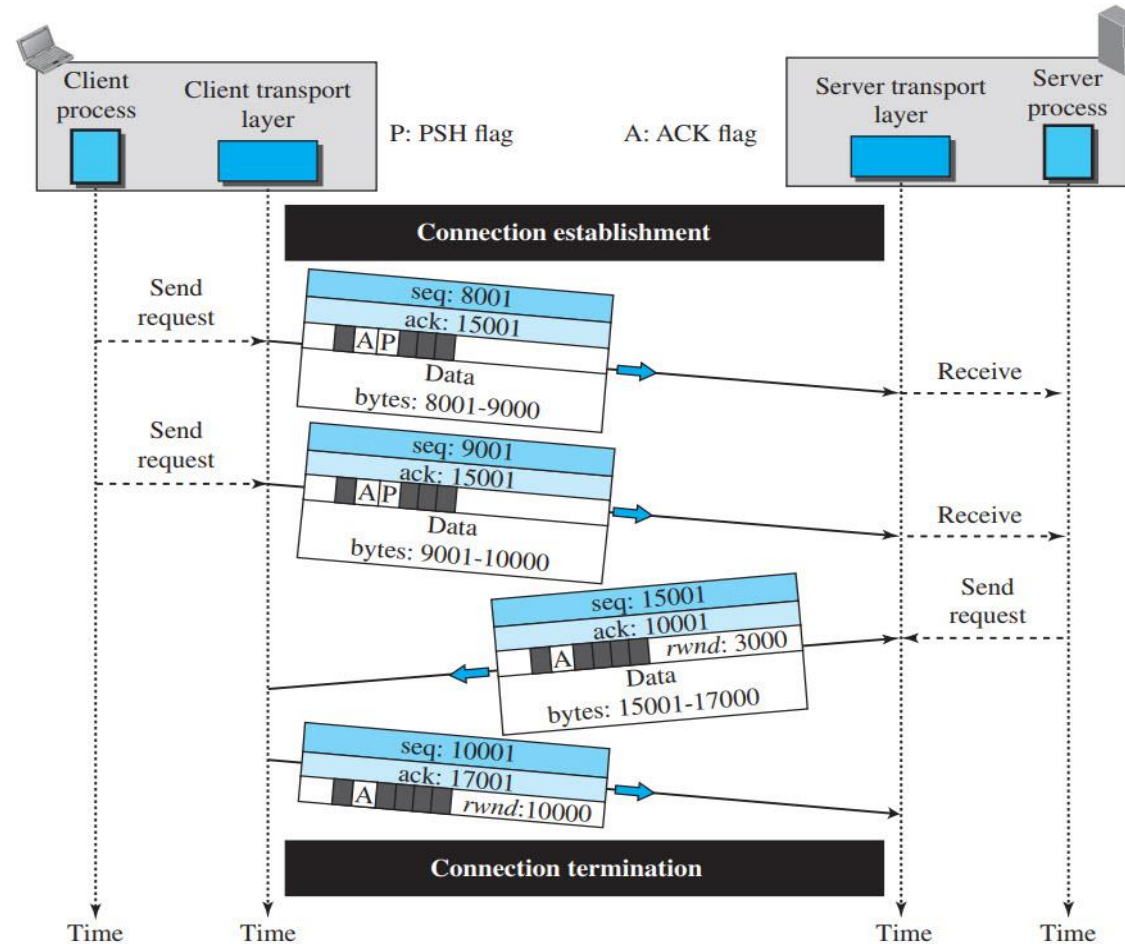


Figure: Data transfer.

Transmission Control Protocol [TCP] (Cntd.)

Connection Termination: Three way Handshaking

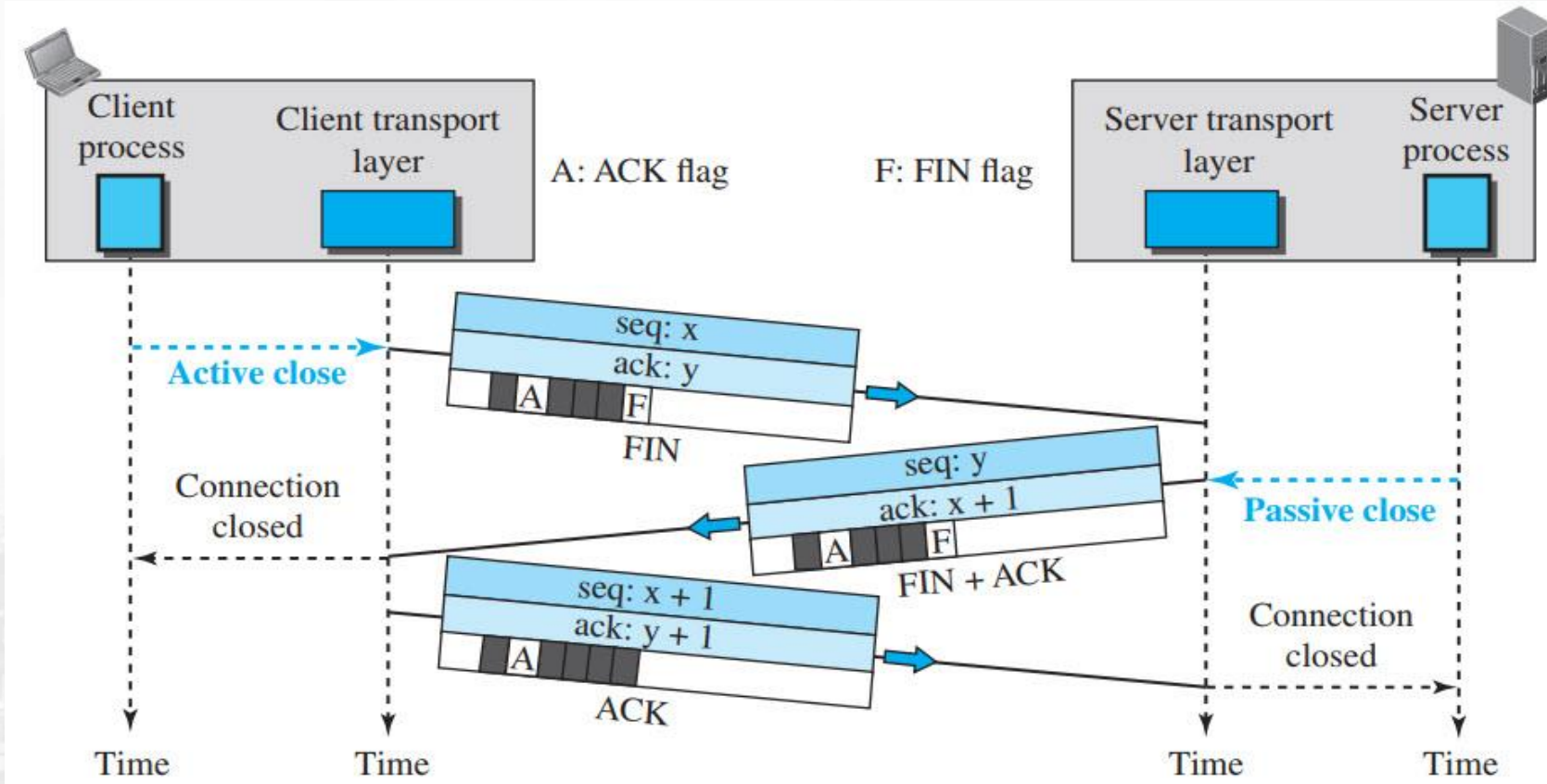


Figure: Connection termination using three-way handshaking.

Figure: Half-close.

Transmission Control Protocol [TCP] (Cntd.)

TCP Connection Management Modeling: State Transition Diagram

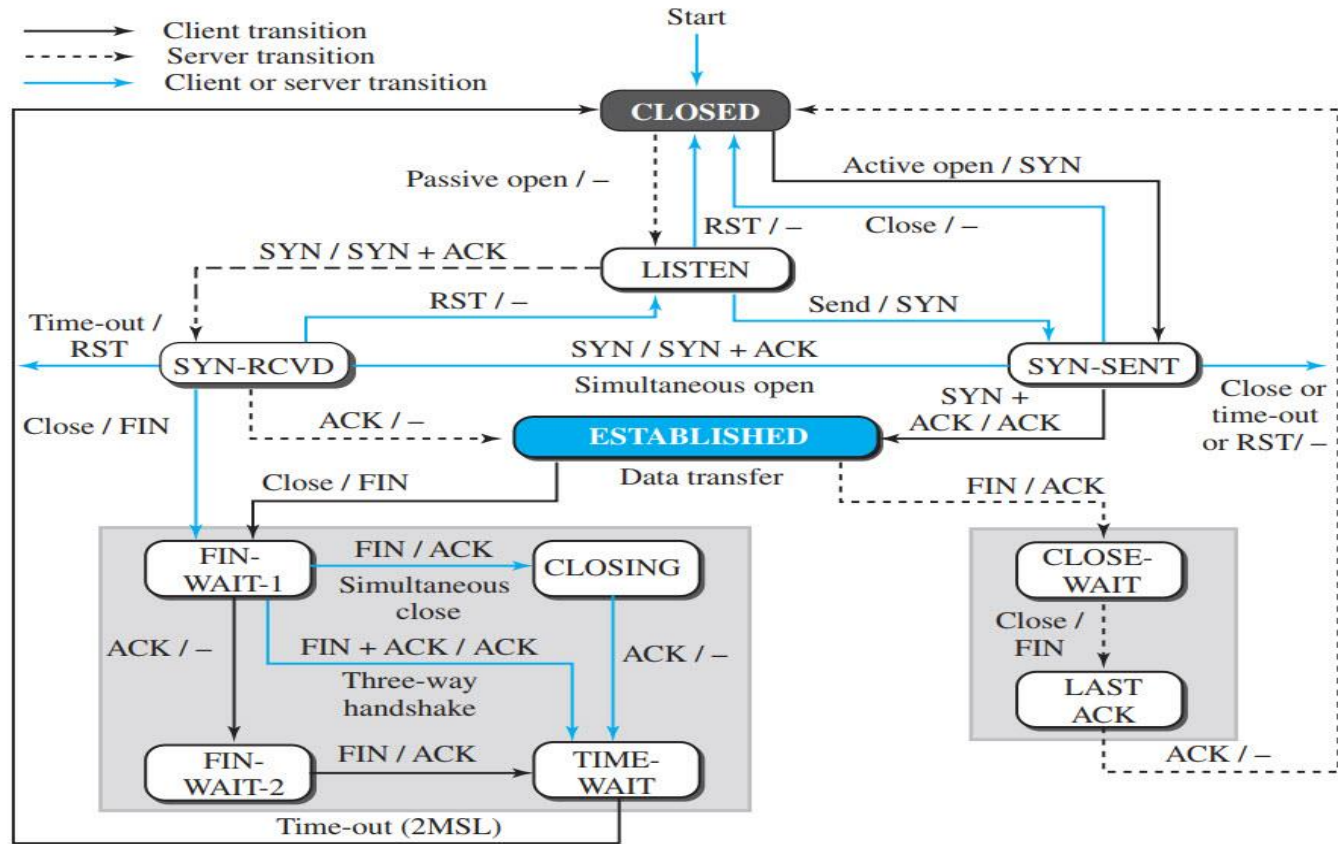


Figure: State transition diagram.

Transmission Control Protocol [TCP] (Cntd.)

TCP Connection Management Modeling: State Transition Diagram - States of TCP

<i>State</i>	<i>Description</i>
CLOSED	No connection exists
LISTEN	Passive open received; waiting for SYN
SYN-SENT	SYN sent; waiting for ACK
SYN-RCVD	SYN + ACK sent; waiting for ACK
ESTABLISHED	Connection established; data transfer in progress
FIN-WAIT-1	First FIN sent; waiting for ACK
FIN-WAIT-2	ACK to first FIN received; waiting for second FIN
CLOSE-WAIT	First FIN received, ACK sent; waiting for application to close
TIME-WAIT	Second FIN received, ACK sent; waiting for 2MSL time-out
LAST-ACK	Second FIN sent; waiting for ACK
CLOSING	Both sides decided to close simultaneously

Figure: States for TCP.

Transmission Control Protocol [TCP] (Cntd.)

TCP Connection Management Modeling: State Transition Diagram - Scenarios

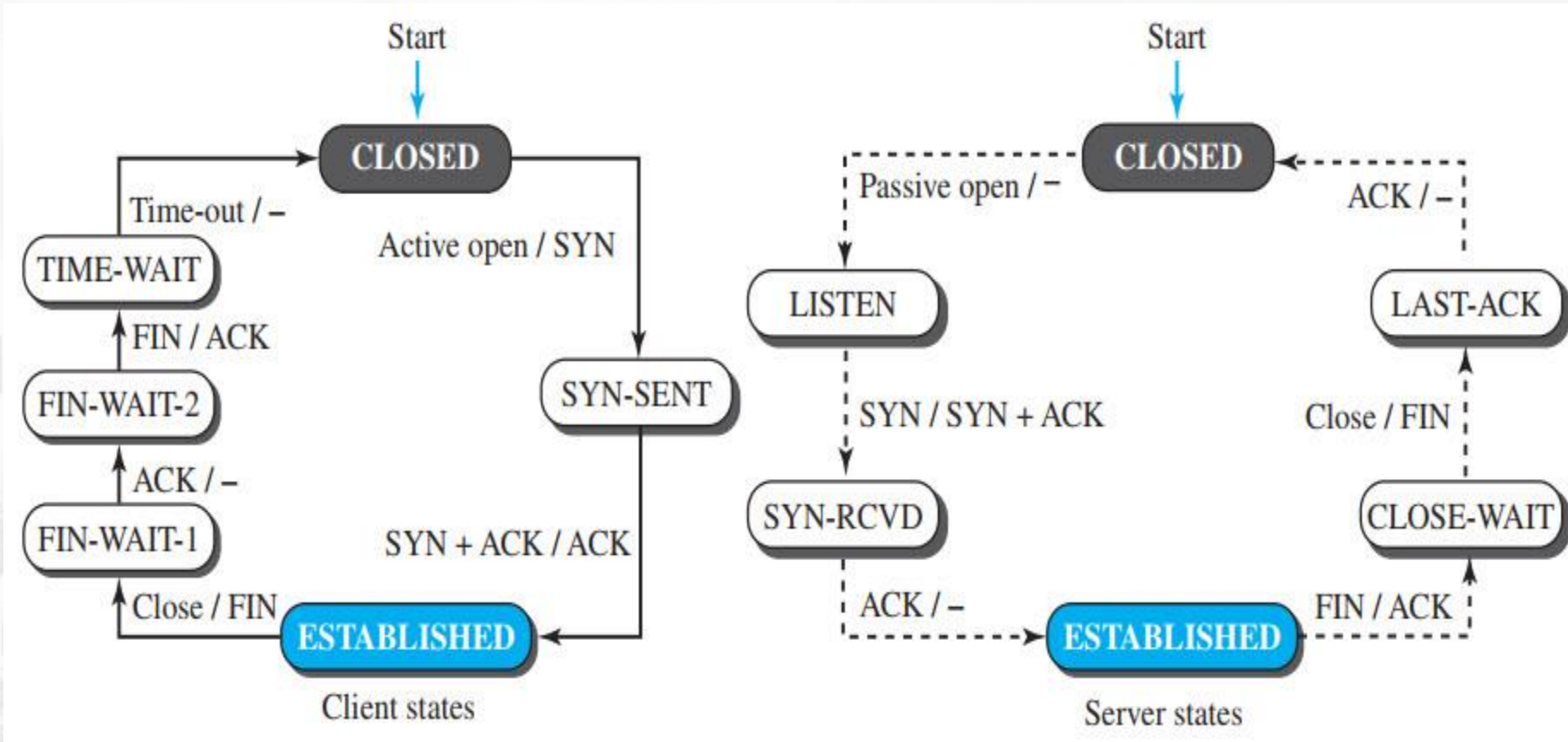


Figure: Transition diagram with half-close connection termination.

Transmission Control Protocol [TCP] (Cntd.)

TCP Connection Management Modeling: State Transition Diagram - Scenarios

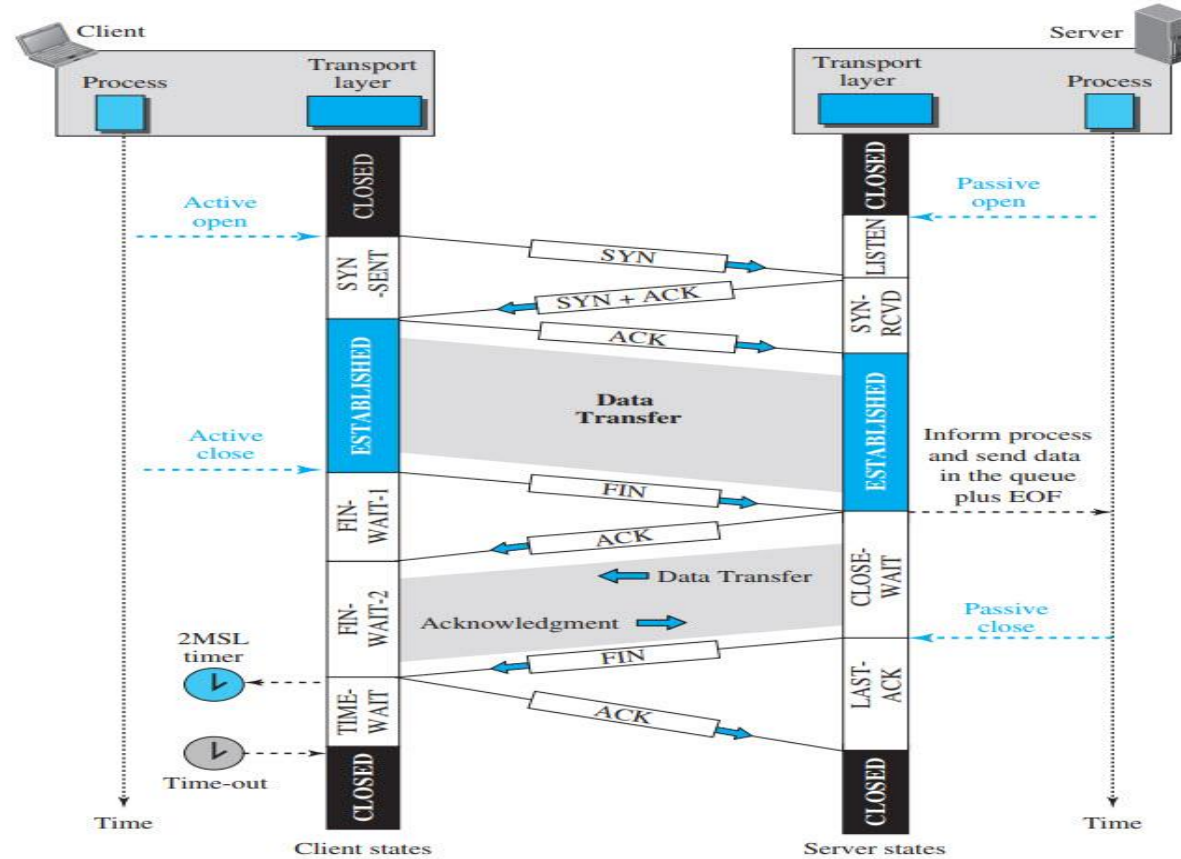


Figure: Time-line diagram for a common scenario.

Transmission Control Protocol [TCP] (Cntd.)

Flow Control:

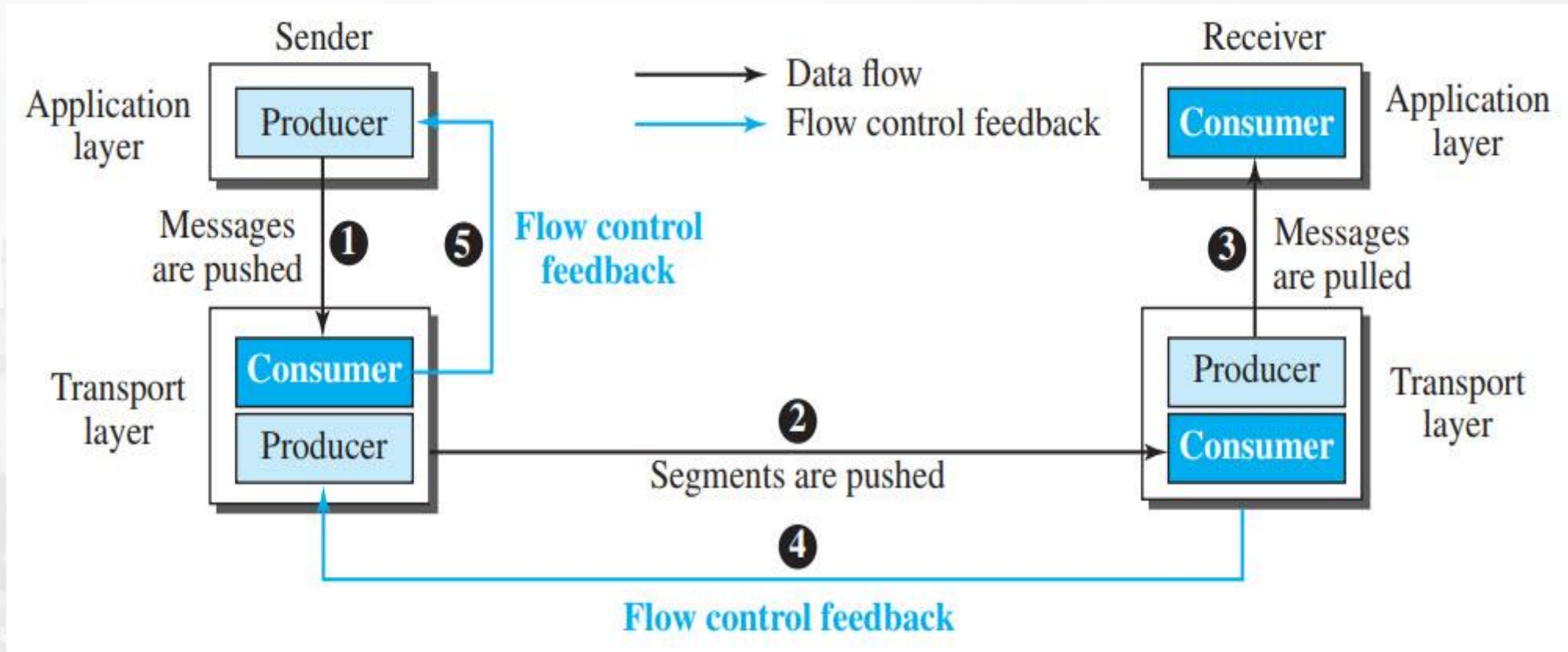


Figure: Data flow and flow control feedbacks in TCP.

Transmission Control Protocol [TCP] (Cntd.)

Flow Control (Cntd.):

- Flow control balances the rate a producer creates data with the rate a consumer can use the data.
- TCP separates flow control from error control.
- To achieve flow control, TCP forces the sender and the receiver to adjust their window sizes, although the size of the buffer for both parties is fixed when the connection is established.

Transmission Control Protocol [TCP] (Cntd.)

Windows in TCP:

- TCP uses two windows (send window and receive window) for each direction of data transfer.
- Window management in TCP decouples the issues of acknowledgement of the correct receipt of segments and receiver buffer allocation.

Transmission Control Protocol [TCP] (Cntd.)

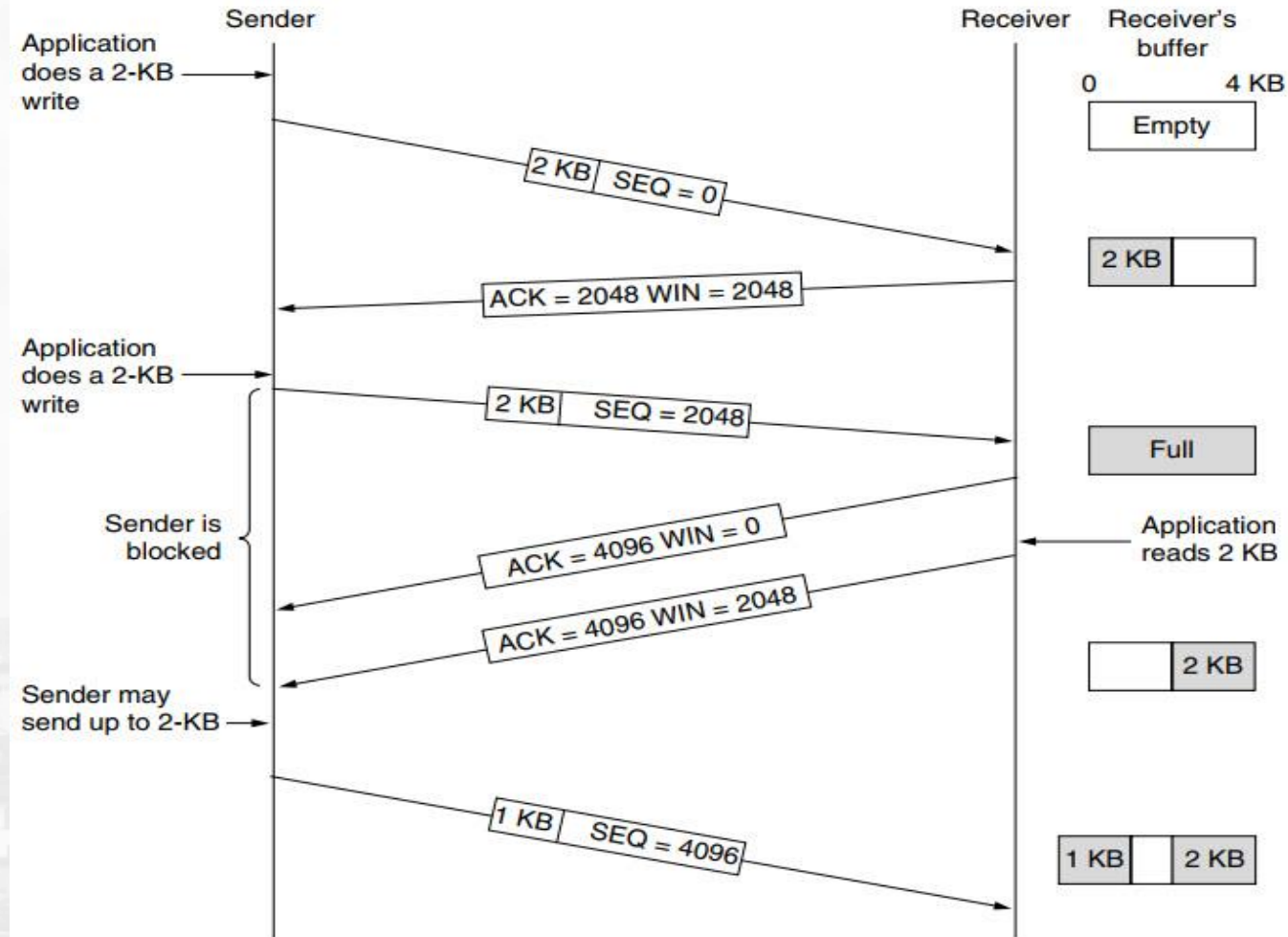


Figure: TCP Sliding window example.

Transmission Control Protocol [TCP] (Cntd.)

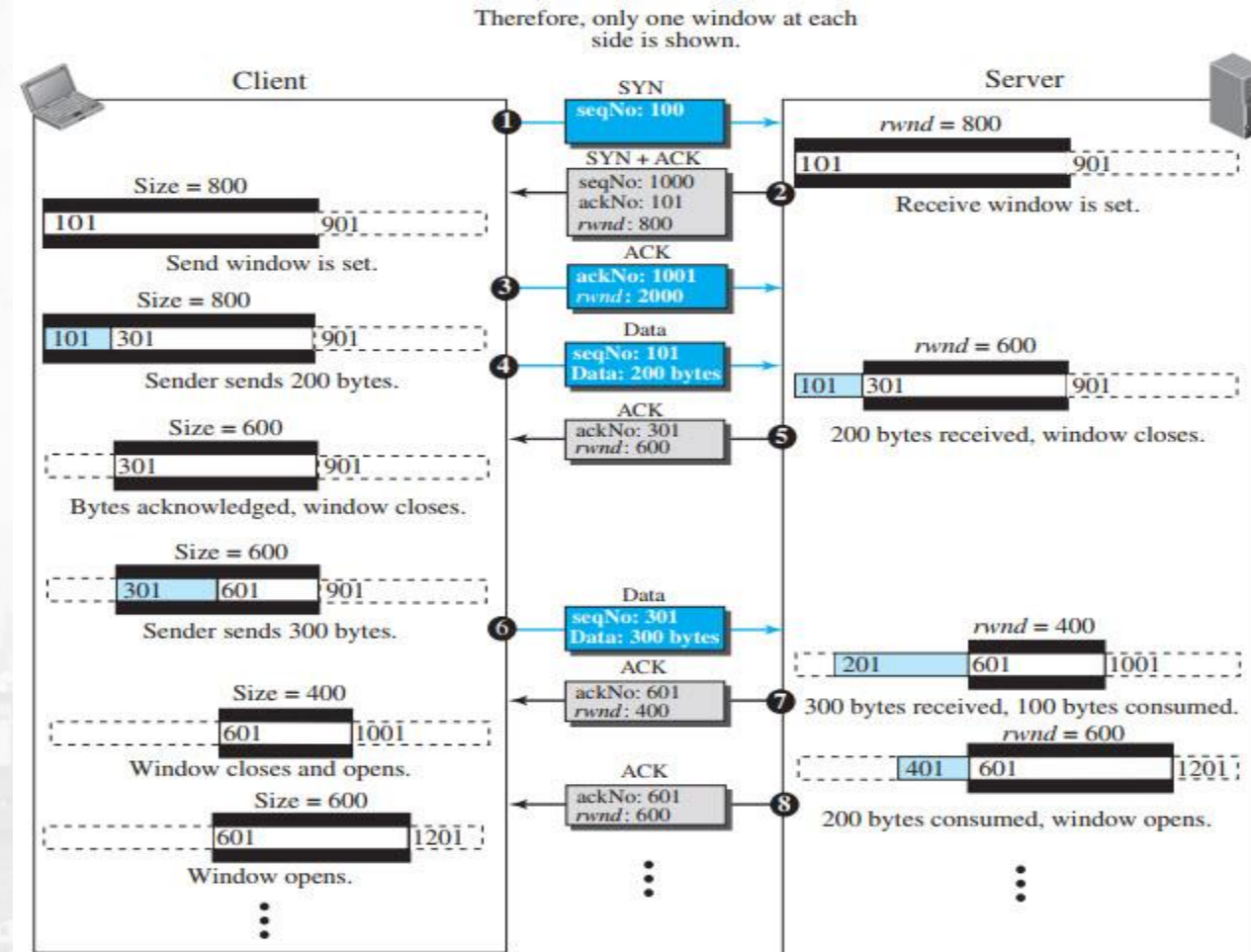


Figure: An example of flow control.

Transmission Control Protocol [TCP] (Cntd.)

Windows in TCP (Cntd.): Send Window

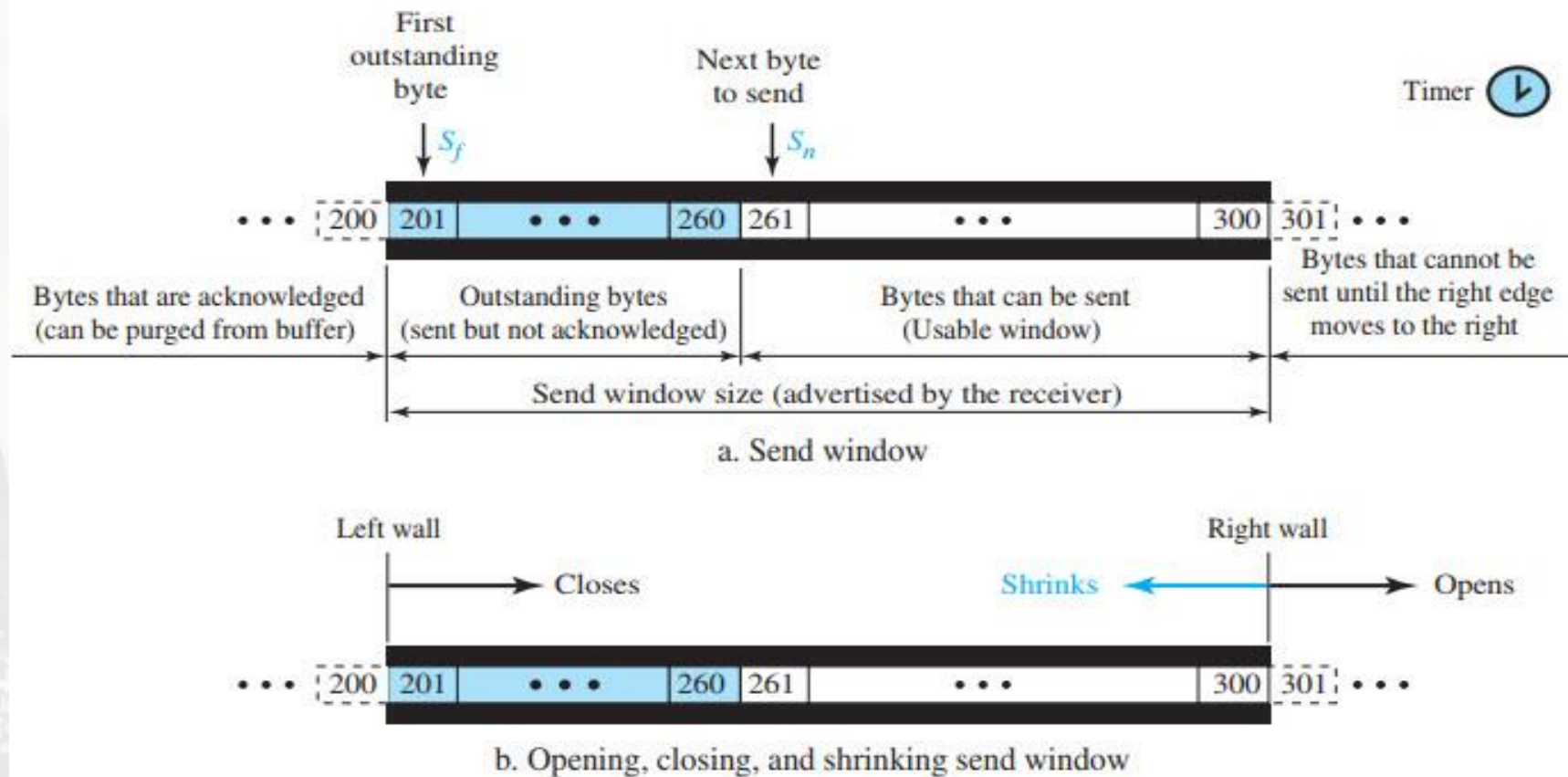


Figure: Send window in TCP.

Transmission Control Protocol [TCP] (Cntd.)

Windows in TCP (Cntd.): Send Window (Cntd.)

- The send window in TCP is similar to the one used with the Selective-Repeat(SR) protocol, but with some differences:
 - The window size in SR is the number of packets, but the window size in TCP is the number of bytes.
 - In some implementations, TCP can store data received from the process and send them later, but we assume that the sending TCP is capable of sending segments of data as soon as it receives them from its process.
 - Selective-Repeat protocol may use several timers for each packet sent, but TCP protocol uses only one timer.
- The opening, closing, and shrinking of the send window is controlled by the receiver. (Check example scenario discussed above)

Transmission Control Protocol [TCP] (Cntd.)

Windows in TCP (Cntd.): Receive Window

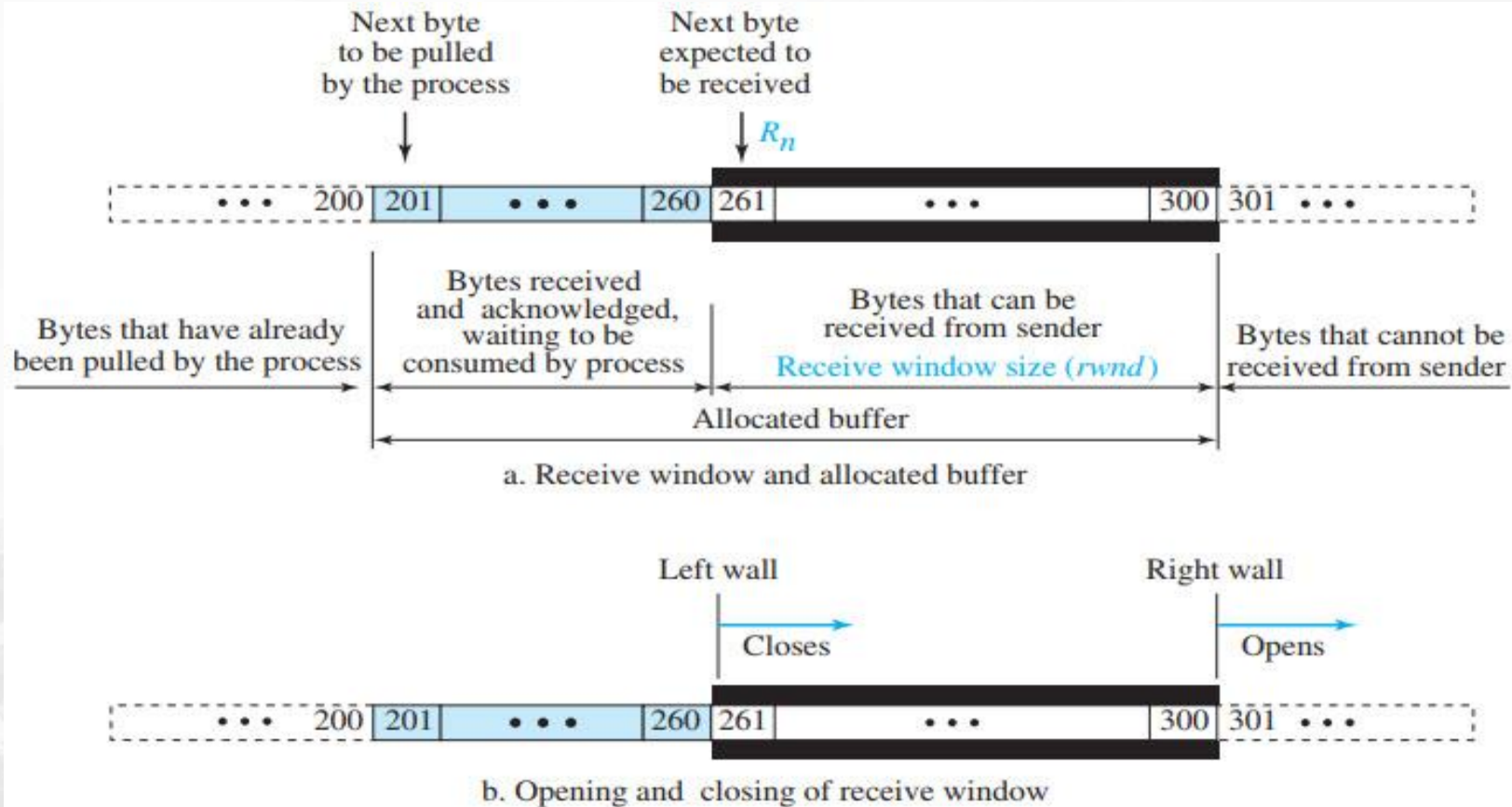


Figure: Receive window in TCP.

Transmission Control Protocol [TCP] (Cntd.)

Windows in TCP (Cntd.): Receive Window (Cntd.)

- The receive window opens and closes; in practice, the window should never shrink.
- There are two differences between the receive window in TCP and the one we used for SR:
 - TCP allows the receiving process to pull data at its own pace.
 - ✓ The receive window size is then always smaller than or equal to the buffer size, as shown in Figure.
 - ✓ The receive window size determines the number of bytes that the receive window can accept from the sender before being overwhelmed (flow control). In other words, the receive window size, normally called *rwnd*, can be determined as:

$$rwnd = \text{buffer size} - \text{number of waiting bytes to be pulled}$$

- The major acknowledgment mechanism in TCP is a cumulative acknowledgment (Acknowledgements can be sent only when all the data up to the byte acknowledged have been received.) announcing the next expected byte to receive. (in this way TCP looks like GBN)

Transmission Control Protocol [TCP] (Cntd.)

Windows in TCP (Cntd.): Opening and Closing Windows

- The send window closes (moves its left wall to the right) when a new acknowledgment allows it to do so. The send window opens (its right wall moves to the right) when the receive window size (rwnd) advertised by the receiver allows it to do so ($\text{new ackNo} + \text{new rwnd} > \text{last ackNo} + \text{last rwnd}$). The send window shrinks in the event this situation does not occur.
- The receive window closes (moves its left wall to the right) when more bytes arrive from the sender; it opens (moves its right wall to the right) when more bytes are pulled by the process. We assume that it does not shrink (the right wall does not move to the left).

Transmission Control Protocol [TCP] (Cntd.)

Windows in TCP (Cntd.): Shrinking of Windows

- The receive window cannot shrink.
- The send window, on the other hand, can shrink if the receiver defines a value for *rwnd* that results in shrinking the window.
- Some implementations do not allow shrinking of the send window. The receiver needs to keep the following relationship between the last and new acknowledgment and the last and new *rwnd* values to prevent shrinking of the send window.

$$new\ ackNo + new\ rwnd \geq last\ ackNo + last\ rwnd$$

Transmission Control Protocol [TCP] (Cntd.)

Windows in TCP (Cntd.): Silly Window Syndrome

- The receive window cannot shrink.
- The send window, on the other hand, can shrink if the receiver defines a value for *rwnd* that results in shrinking the window.
- Some implementations do not allow shrinking of the send window. The receiver needs to keep the following relationship between the last and new acknowledgment and the last and new *rwnd* values to prevent shrinking of the send window.

$$new\ ackNo + new\ rwnd \geq last\ ackNo + last\ rwnd$$

Transmission Control Protocol [TCP] (Cntd.)

Windows in TCP (Cntd.): Silly Window Syndrome

- Silly window syndrome is a problem that can arise in poor implementations of the transmission control protocol (TCP).
- It can lead to a significant reduction in network performance and can indicate an overloaded server or a sending application that is limiting throughput.
- There are situations, resulting sending of data in very small segments, which reduces the efficiency of the operation.
 - when either the sending application program creates data slowly
 - or the receiving application program consumes data slowly
 - or both.
- Inefficiently using the capacity of the network. The inefficiency is even worse after accounting for the data-link layer and physical-layer overhead. This problem is called the **silly window syndrome**.

Transmission Control Protocol [TCP] (Cntd.)

Windows in TCP (Cntd.): Silly Window Syndrome (Cntd.)

➤ Syndrome Created by Sender:

- ✓ The sending TCP may create a silly window syndrome if it is serving an application program that creates data slowly.
- ✓ The solution is to prevent the sending TCP from sending the data byte by byte. The sending TCP must be forced to wait and collect data to send in a larger block.
- ✓ How long should the sending TCP wait? – elegant solution is Nagle's algorithm.

Transmission Control Protocol [TCP] (Cntd.)

Windows in TCP (Cntd.): Silly Window Syndrome (Cntd.)

➤ Syndrome Created by Sender (Cntd.):

✓ Nagle's algorithm :

1. The sending TCP sends the first piece of data it receives from the sending application program even if it is only 1 byte.
2. After sending the first segment, the sending TCP accumulates data in the output buffer and waits until either the receiving TCP sends an acknowledgment or until enough data have accumulated to fill a maximum-size segment. At this time, the sending TCP can send the segment.
3. Step 2 is repeated for the rest of the transmission. Segment 3 is sent immediately if an acknowledgment is received for segment 2, or if enough data have accumulated to fill a maximum-size segment.

Transmission Control Protocol [TCP] (Cntd.)

Windows in TCP (Cntd.): Silly Window Syndrome (Cntd.)

➤ Syndrome Created by the Receiver:

- ✓ The receiving TCP may create a silly window syndrome if it is serving an application program that consumes data slowly.
- ✓ Two solutions have been proposed to prevent the silly window syndrome.
 - ✓ **Solution 1:** Clark's solution is to send an acknowledgment as soon as the data arrive, but to announce a window size of zero until either there is enough space to accommodate a segment of maximum size or until at least half of the receive buffer is empty.
 - ✓ **Solution 2:** The second solution is to delay sending the acknowledgment. This means that when a segment arrives, it is not acknowledged immediately. The receiver waits until there is a decent amount of space in its incoming buffer before acknowledging the arrived segments. The delayed acknowledgment prevents the sending TCP from sliding its window. After the sending TCP has sent the data in the window, it stops. This kills the syndrome. Delayed acknowledgment also has another advantage: it reduces traffic. There also is a disadvantage in that the delayed acknowledgment may result in the sender unnecessarily retransmitting the unacknowledged segments.

Transmission Control Protocol [TCP] (Cntd.)

Error Control:

- **TCP is a reliable transport-layer protocol. TCP provides reliability using error control.**
- **Error control includes mechanisms for**
 - ✓ detecting and resending corrupted segments
 - ✓ resending lost segments
 - ✓ storing out-of-order segments until missing segments arrive
 - ✓ and detecting and discarding duplicated segments.
- **Error control in TCP is achieved through the use of three simple tools:**
 - Checksum
 - Acknowledgment
 - and Time-out.

Transmission Control Protocol [TCP] (Cntd.)

Error Control (Cntd.): Checksum

- Each segment includes a checksum field, which is used to check for a corrupted segment.
- If a segment is corrupted, as detected by an invalid checksum, the segment is discarded by the destination TCP and is considered as lost.
- TCP uses a 16-bit checksum that is mandatory in every segment.

Transmission Control Protocol [TCP] (Cntd.)

Error Control (Cntd.): Acknowledgment

- TCP uses acknowledgments to confirm the receipt of data segments.
- Control segments that carry no data, but consume a sequence number, are also acknowledged.
- ACK segments do not consume sequence numbers and are not acknowledged.
- Acknowledgment Type:
 - Cumulative Acknowledgment (ACK) – positive cumulative acknowledgment
 - Selective Acknowledgment (SACK) – A SACK does not replace an ACK, but reports additional information to the sender.

Transmission Control Protocol [TCP] (Cntd.)

Error Control (Cntd.): Common Rules for Generating Acknowledgments

➤ Rule 1:

When end A sends a data segment to end B, it must include (piggyback) an acknowledgment that gives the next sequence number it expects to receive. This rule decreases the number of segments needed and therefore reduces traffic.

➤ Rule 2:

When the receiver has no data to send and it receives an in-order segment (with expected sequence number) and the previous segment has already been acknowledged, the receiver delays sending an ACK segment until another segment arrives or until a period of time (normally 500 ms) has passed. In other words, the receiver needs to delay sending an ACK segment if there is only one outstanding in-order segment. This rule reduces ACK segments.

Transmission Control Protocol [TCP] (Cntd.)

Error Control (Cntd.): Common Rules for Generating Acknowledgments (Cntd.)

➤ Rule 3:

When a segment arrives with a sequence number that is expected by the receiver, and the previous in-order segment has not been acknowledged, the receiver immediately sends an ACK segment. In other words, there should not be more than two in-order unacknowledged segments at any time. This prevents the unnecessary retransmission of segments that may create congestion in the network.

➤ Rule 4:

When a segment arrives with an out-of-order sequence number that is higher than expected, the receiver immediately sends an ACK segment announcing the sequence number of the next expected segment. This leads to the fast retransmission of missing segments (discussed later).

Transmission Control Protocol [TCP] (Cntd.)

Error Control (Cntd.): Common Rules for Generating Acknowledgments (Cntd.)

➤ Rule 5:

When a missing segment arrives, the receiver sends an ACK segment to announce the next sequence number expected. This informs the receiver that segments reported missing have been received.

➤ Rule 6:

If a duplicate segment arrives, the receiver discards the segment, but immediately sends an acknowledgment indicating the next in-order segment expected. This solves some problems when an ACK segment itself is lost.

Transmission Control Protocol [TCP] (Cntd.)

Error Control (Cntd.): Retransmission

- The heart of the error control mechanism is the retransmission of segments.
- When retransmission is possible?: When a segment is sent, it is stored in a queue until it is acknowledged.
 - ✓ When the retransmission timer expires, or
 - ✓ When the sender receives three duplicate ACKs for the first segment in the queue, that segment is retransmitted.

Transmission Control Protocol [TCP] (Cntd.)

Error Control (Cntd.): Retransmission (Cntd.)

➤ Retransmission after retransmission time-out (RTO):

- ✓ Sending TCP maintains one retransmission time-out (RTO) for each connection.
- ✓ When the timer matures, i.e. times out, TCP resends the segment.
- ✓ The value of RTO is dynamic in TCP and is updated based on the round-trip time (RTT) of segments.

➤ Retransmission after Three Duplicate ACK Segments:

- ✓ The previous rule about retransmission of a segment is sufficient if the value of RTO is not large.
- ✓ Goal in this retransmission method is to allowing senders to retransmit without waiting for a time out
- ✓ if three duplicate acknowledgments (i.e., an original ACK plus three exactly identical copies) arrive for a segment, the next segment is retransmitted without waiting for the time-out.
- ✓ This feature is called fast retransmission.

Transmission Control Protocol [TCP] (Cntd.)

Error Control (Cntd.): Out of Order segments

- **Data may arrive out of order and be temporarily stored by the receiving TCP, but TCP guarantees that no out-of-order data are delivered to the process.**

Transmission Control Protocol [TCP] (Cntd.)

Error Control (Cntd.): Scenarios

➤ Normal operation

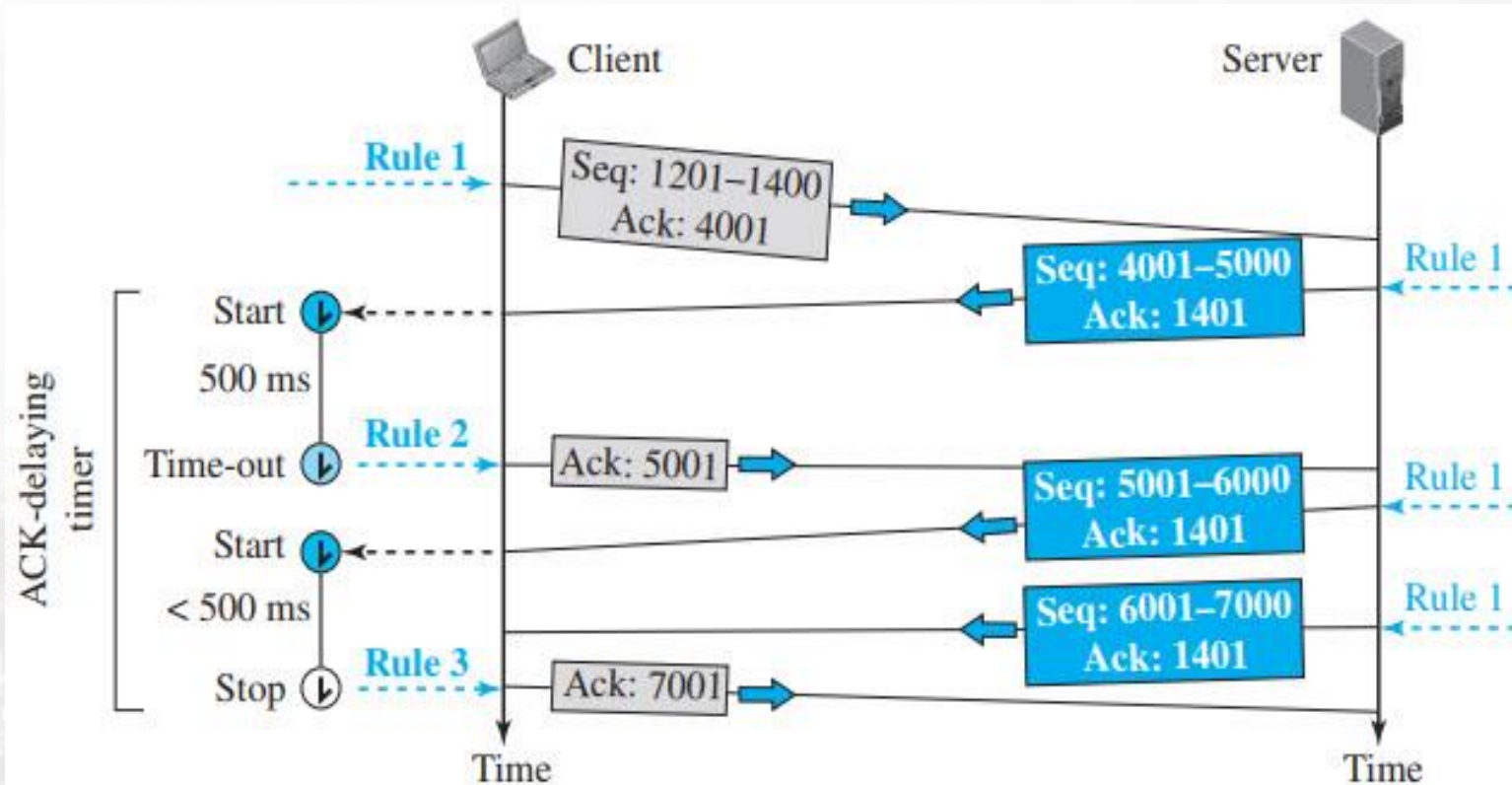


Figure: Normal operation.

Transmission Control Protocol [TCP] (Cntd.)

Error Control (Cntd.): Scenarios (Cntd.)

➤ Lost segment

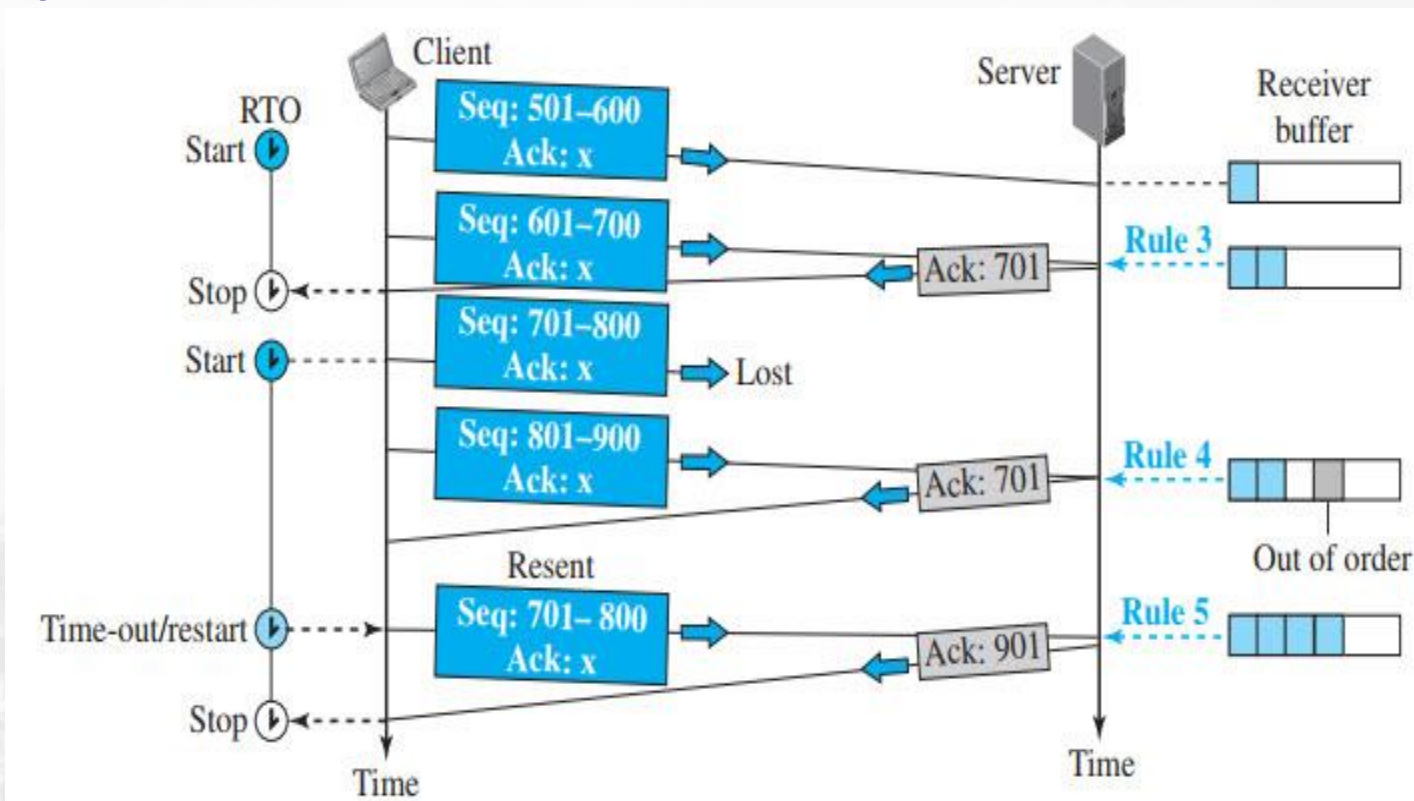


Figure: Lost segment.

Transmission Control Protocol [TCP] (Cntd.)

Error Control (Cntd.): Scenarios (Cntd.)

➤ Fast retransmission

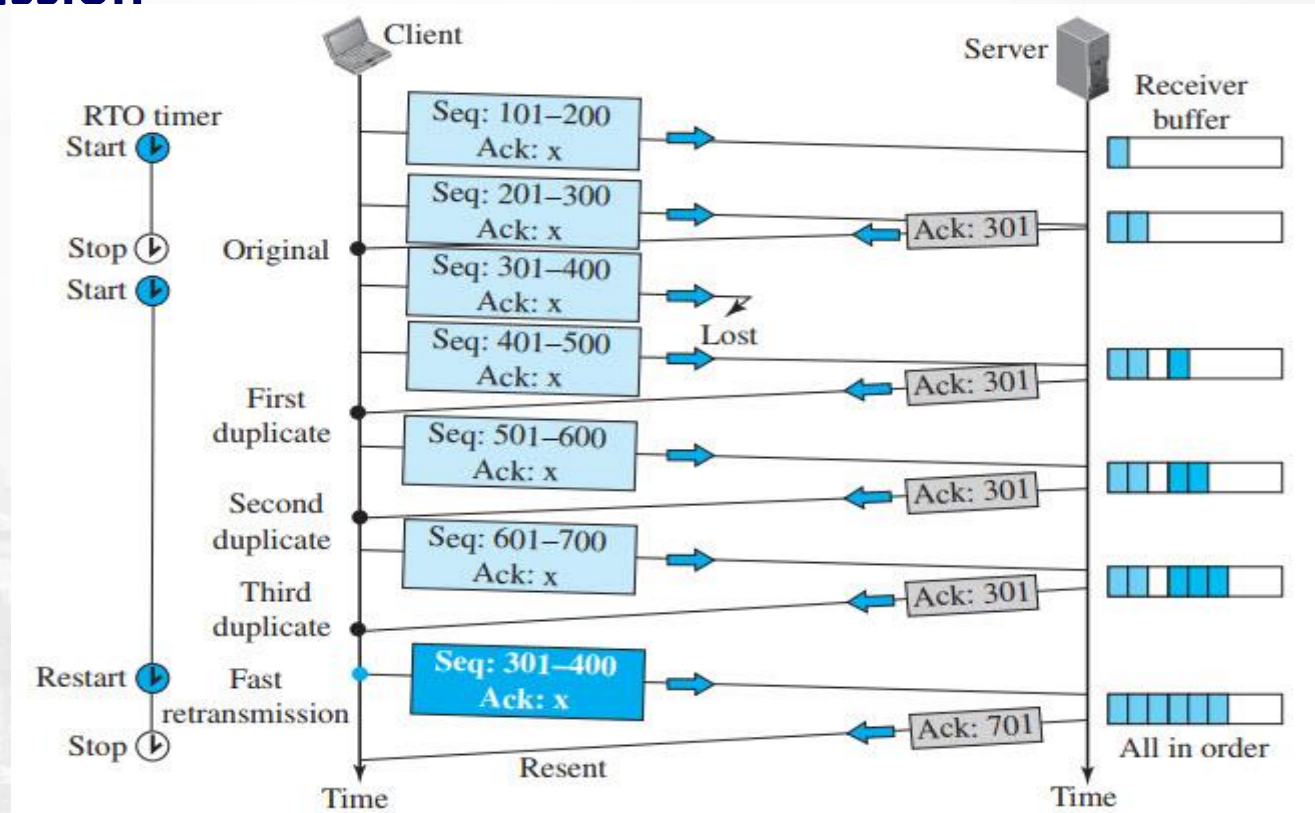


Figure: Fast retransmission.

Transmission Control Protocol [TCP] (Cntd.)

Error Control (Cntd.): Scenarios (Cntd.)

- **Delayed Segment**
- **Duplicate Segment**
- **Automatically Corrected Lost ACK**
- **Lost Acknowledgment Corrected by Resending a Segment**

Transmission Control Protocol [TCP] (Cntd.)

Error Control (Cntd.): Scenarios (Cntd.)

➤ Lost acknowledgment

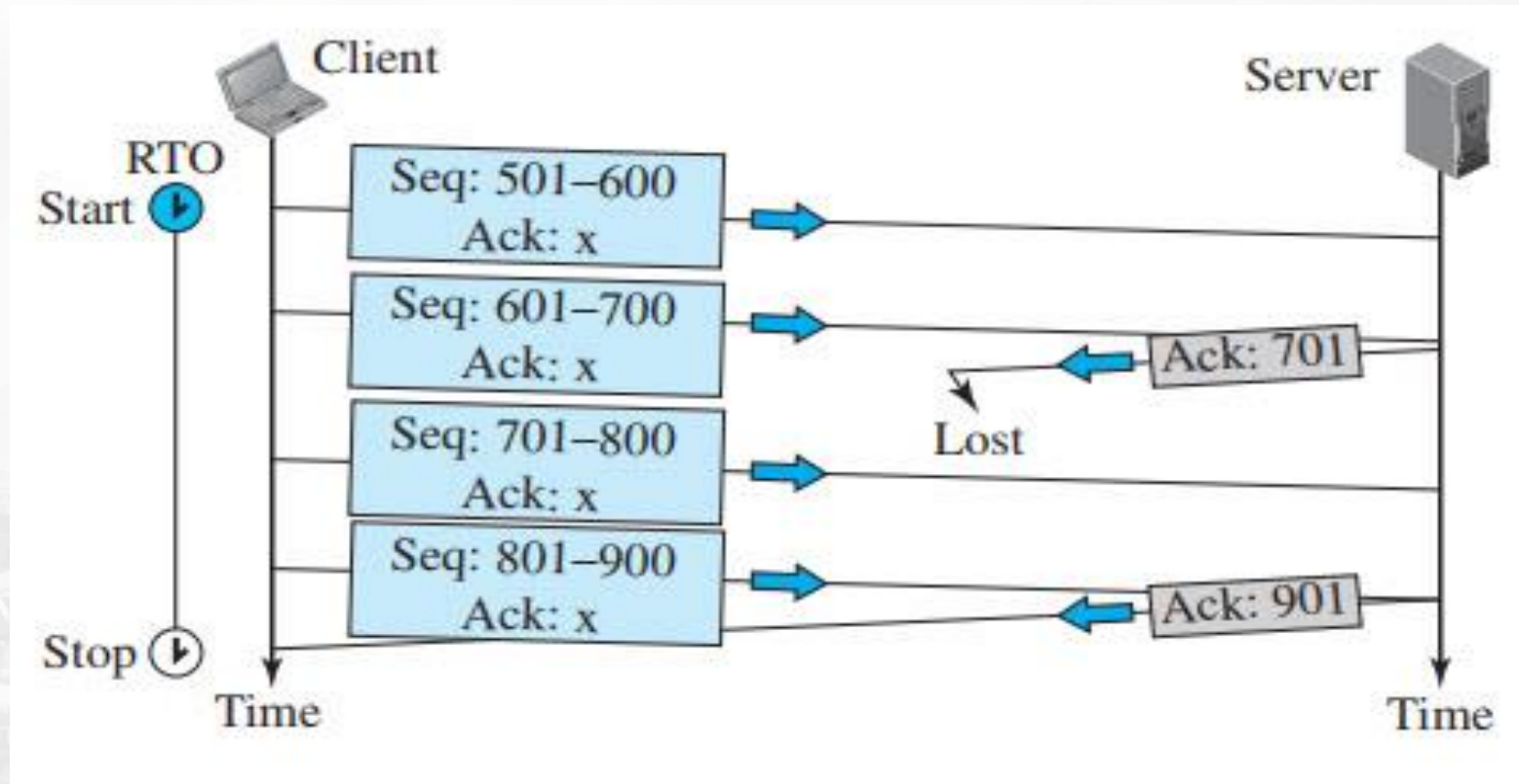


Figure: Lost acknowledgment.

Transmission Control Protocol [TCP] (Cntd.)

Error Control (Cntd.): Scenarios (Cntd.)

➤ Lost acknowledgment corrected by resending a segment

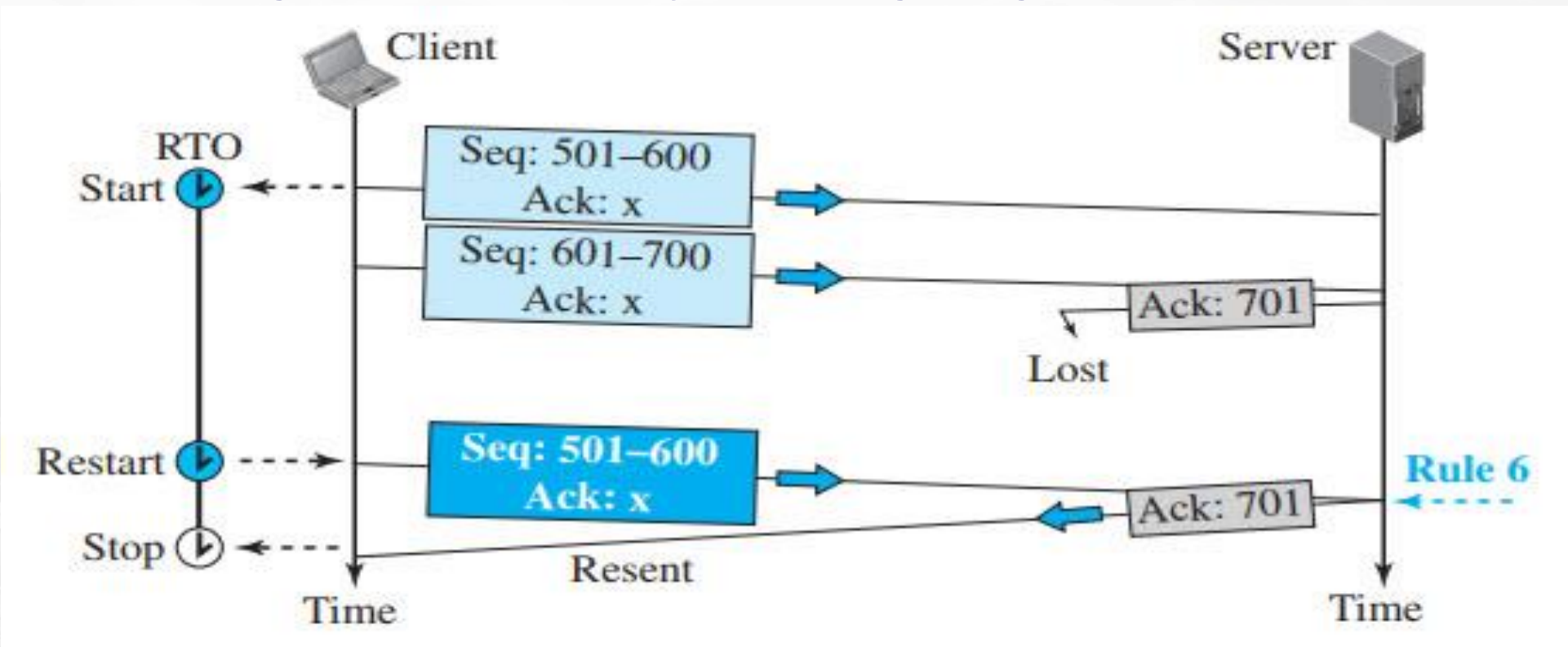


Figure: Lost acknowledgment corrected by resending a segment.

Transmission Control Protocol [TCP] (Cntd.)

Error Control (Cntd.): Scenarios (Cntd.)

➤ Deadlock Created by Lost Acknowledgment

Lost acknowledgments may create deadlock if they are not properly handled.

Transmission Control Protocol [TCP] (Cntd.)

Layer	Policies
Transport	<ul style="list-style-type: none">• Retransmission policy• Out-of-order caching policy• Acknowledgement policy• Flow control policy• Timeout determination
Network	<ul style="list-style-type: none">• Virtual circuits versus datagram inside the subnet• Packet queueing and service policy• Packet discard policy• Routing algorithm• Packet lifetime management
Data link	<ul style="list-style-type: none">• Retransmission policy• Out-of-order caching policy• Acknowledgement policy• Flow control policy

Figure: Policies that affect congestion at different layers.

Transmission Control Protocol [TCP] (Cntd.)

Congestion Control:

- Congestion control refers to techniques and mechanisms that can either prevent congestion before it happens or remove congestion after it has happened.
- In the Internet, congestion control could be implemented either in the network layer or the transport layer.
- The congestion problem was clearly identified in the later 1980s and the researchers who developed techniques to solve the problem opted for a solution in the transport layer.
- Adding congestion control to the transport layer makes sense since this layer provides a reliable data transfer and avoiding congestion is a factor in this reliable delivery.

Transmission Control Protocol [TCP] (Cntd.)

Congestion Control (cntd.):

- The difference between flow control and congestion control is that **Flow control**, in contrast, relates to the traffic between a particular sender and a particular receiver. Its job is to make sure that a fast sender cannot continually transmit data faster than the receiver is able to absorb it and **Congestion control** has to do with making sure the network is able to carry the offered traffic. It is a global issue, involving the behavior of all the hosts and routers.
- We can divide congestion control mechanisms into two broad categories: **open-loop congestion control (prevention)** and **closed-loop congestion control (removal)**.

Transmission Control Protocol [TCP] (Cntd.)

- 🖥️ **Congestion Control: Open-Loop Congestion Control [Prevention]**
 - In open-loop congestion control, policies are applied to prevent congestion before it happens.
 - In these mechanisms, congestion control is handled by either the source or the destination.

Transmission Control Protocol [TCP] (Cntd.)

Congestion Control: Open-Loop Congestion Control [Prevention] (Cntd.)

➤ There is a brief list of policies that can prevent congestion.

1. Retransmission Policy
2. Window Policy
3. Acknowledgement Policy
4. Discarding Policy
5. Admission Policy

Transmission Control Protocol [TCP] (Cntd.)

- ❏ **Congestion Control: Closed-Loop Congestion Control [Removal]**
- **Closed-loop congestion control mechanisms try to alleviate congestion after it happens.**

Transmission Control Protocol [TCP] (Cntd.)

Congestion Control: Closed-Loop Congestion Control [Removal] (Cntd.)

➤ Several mechanisms have been used by different protocols. We describe a few of them here.

1. Backpressure
2. Choke Packet
3. Implicit Signaling
4. Explicit Signaling

Transmission Control Protocol [TCP] (Cntd.)

Congestion Control: Congestion Window

- We have already seen that, the size of the send window is controlled by the receiver using the value of **rwnd**, which is advertised in each segment traveling in the opposite direction. This strategy ensures that the receiver window is never overwhelmed.
- But this doesn't mean that the congestion is not at all present, congestion may not be present at ends, **but there is possibility that it may be present at intermediate router** as it may be receiving data from multiple senders.
- TCP needs to worry about congestion in the middle because many segments lost may seriously affect the error control.
- More segment loss means resending the same segments again, resulting in worsening the congestion, and finally the collapse of the communication.

Transmission Control Protocol [TCP] (Cntd.)

Congestion Control: Congestion Window (Cntd.)

- The congestion in the router is in the IP territory and should be taken care of by IP. However, IP is a simple protocol with no congestion control. Therefore TCP, itself, needs to be responsible for this problem.
- TCP needs to define policies that accelerate the data transmission when there is no congestion and decelerate the transmission when congestion is detected.
- To control the number of segments to transmit, TCP uses another variable called a congestion window, *cwnd*, whose size is controlled by the congestion situation in the network.
- The *cwnd* variable and the *rwnd* variable together define the size of the send window in TCP. The first is related to the congestion in the middle (network); the second is related to the congestion at the end. The actual size of the window is the minimum of these two.

$$\text{Actual window size} = \text{minimum}(\text{rwnd}, \text{cwnd})$$

Transmission Control Protocol [TCP] (Cntd.)

Congestion Control: Congestion Detection

- How a TCP sender can detect the possible existence of congestion in the network?
- Sender rely on feedback from the other end to detect congestion: ACKs.
- The TCP sender uses the occurrence of two events as signs of congestion in the network: time-out and receiving three duplicate ACKs.
 - ✓ Time Out - If a TCP sender does not receive an ACK for a segment or a group of segments before the time-out occurs, it assumes that the corresponding segment or segments are lost and the loss is due to congestion.
 - ✓ Duplicate ACKs - receiving of three duplicate ACKs (four ACKs with the same acknowledgment number). When a TCP receiver sends a duplicate ACK, it is the sign that a segment has been delayed, but sending three duplicate ACKs is the sign of a missing segment.
- The congestion in the case of three duplicate ACKs can be less severe than in the case of time-out.

Transmission Control Protocol [TCP] (Cntd.)

Congestion Control: Congestion Policies

➤ TCP's general policy for handling congestion is based on three algorithms:

- ✓ Slow Start Algorithm: Exponential Increase
- ✓ Congestion Avoidance Algorithm: Additive Increase
- ✓ Fast Recovery Algorithm

➤ Policy Transition : three versions of TCP

- ✓ Tahoe TCP
- ✓ Reno TCP
- ✓ New Reno TCP

Transmission Control Protocol [TCP] (Cntd.)

📖 Congestion Control: Congestion Policies (Cntd.)

➤ Slow Start – Exponential Increase:

- ✓ Based on the idea that the size of the congestion window (cwnd) starts with one maximum segment size (MSS), but it increases one MSS each time an acknowledgment arrives.
- ✓ MSS is a value negotiated during the connection establishment, using an option of the same name.
- ✓ the size of the congestion window in this algorithm is a function of the number of ACKs arrived and can be determined as follows.

If an ACK arrives, $cwnd = cwnd + 1$.

- ✓ If we look at the size of the cwnd in terms of round-trip times (RTTs), we find that the growth rate is exponential in terms of each round trip time, which is a very aggressive approach:

Start $\rightarrow cwnd = 1 \rightarrow 2^0$

After 1 RTT $\rightarrow cwnd = cwnd + 1 = 1 + 1 = 2 \rightarrow 2^1$

After 2 RTT $\rightarrow cwnd = cwnd + 2 = 2 + 2 = 4 \rightarrow 2^2$

After 3 RTT $\rightarrow cwnd = cwnd + 4 = 4 + 4 = 8 \rightarrow 2^3$

Transmission Control Protocol [TCP] (Cntd.)

Congestion Control: Congestion Policies (Cntd.)

➤ Slow Start – Exponential Increase (Cntd.):

- ✓ A slow start cannot continue indefinitely. There must be a threshold to stop this phase. The sender keeps track of a variable named ssthresh (slow-start threshold). When the size of the window in bytes reaches this threshold, slow start stops and the next phase starts.
- ✓ slow-start strategy is slower in the case of delayed acknowledgments.
- ✓ The algorithm starts slowly, but grows exponentially.
- ✓ In the slow-start algorithm, the size of the congestion window increases exponentially until it reaches a threshold. (See figure)

Transmission Control Protocol [TCP] (Cntd.)

🖥️ Congestion Control: Congestion Policies (Cntd.)

➤ Slow Start – Exponential Increase (Cntd.):

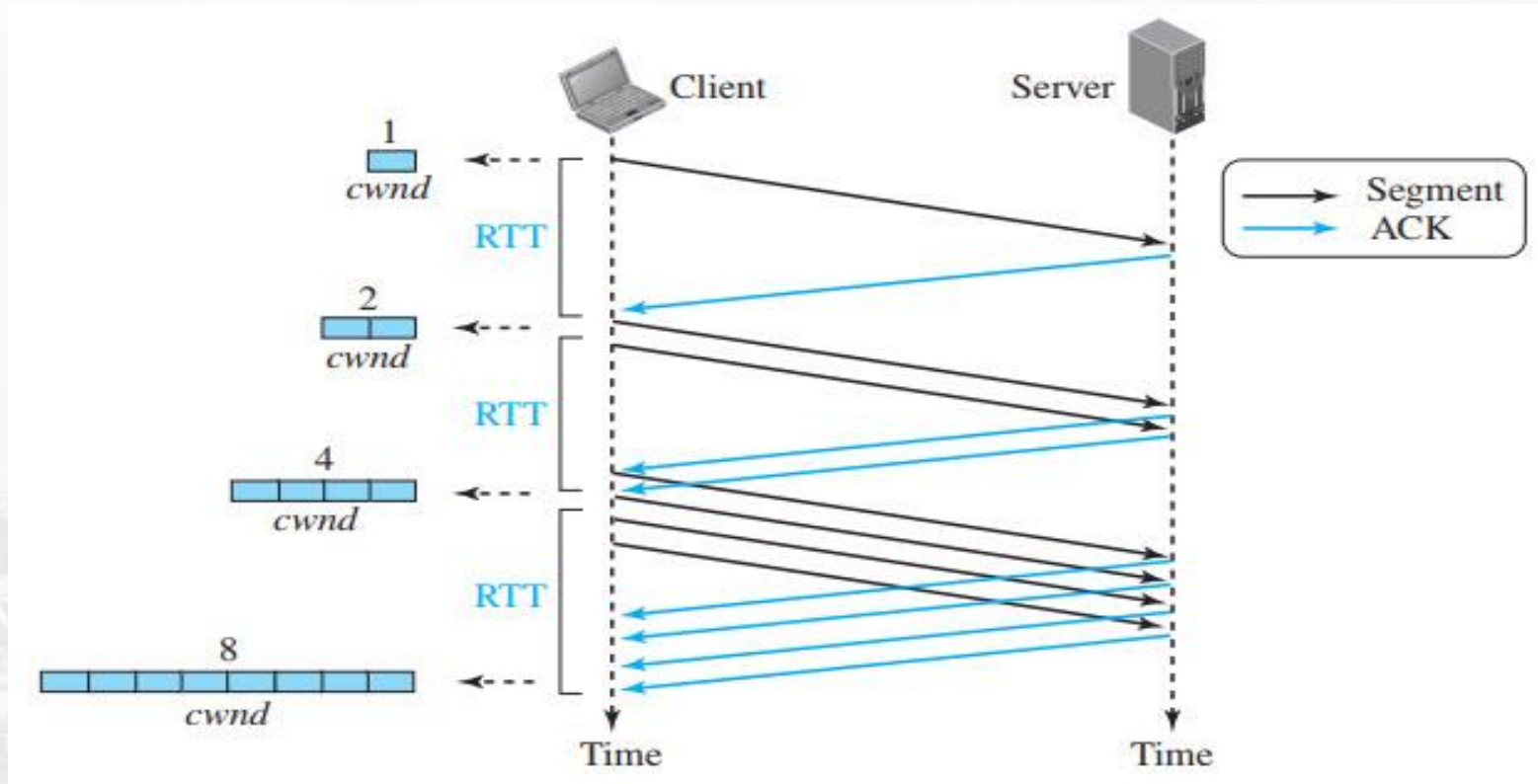


Figure: Slow start, exponential increase.

Transmission Control Protocol [TCP] (Cntd.)

🖥️ Congestion Control: Congestion Policies (Cntd.)

➤ Congestion Avoidance – Additive Increase:

- ✓ increases the *cwnd* additively instead of exponentially.
- ✓ When the size of the congestion window reaches the slow-start threshold in the case where *cwnd* = *ssthresh*, the slow-start phase stops and the additive phase begins.
- ✓ In this algorithm, each time the whole “window” of segments is acknowledged, the size of the congestion window is increased by one. (Refer Figure)
- ✓ A window is the number of segments transmitted during RTT.
- ✓ The size of the congestion window in this algorithm is also a function of the number of ACKs that have arrived and can be determined as follows:

If an ACK arrives, $cwnd = cwnd + (1/cwnd)$.

- ✓ The size of the window increases only $1/cwnd$ portion of MSS (in bytes).

Transmission Control Protocol [TCP] (Cntd.)

📖 Congestion Control: Congestion Policies (Cntd.)

➤ Congestion Avoidance – Additive Increase (Cntd.):

- ✓ Size of the *cwnd* in terms of round-trip times (RTTs): the growth rate is linear in terms of each round-trip time

Start → $cwnd = i$

After 1 RTT → $cwnd = i + 1$

After 2 RTT → $cwnd = i + 2$

After 3 RTT → $cwnd = i + 3$

- ✓ In the congestion-avoidance algorithm, the size of the congestion window increases additively until congestion is detected.

Transmission Control Protocol [TCP] (Cntd.)

- 🖥️ **Congestion Control: Congestion Policies (Cntd.)**
- **Congestion Avoidance – Additive Increase (Cntd.):**

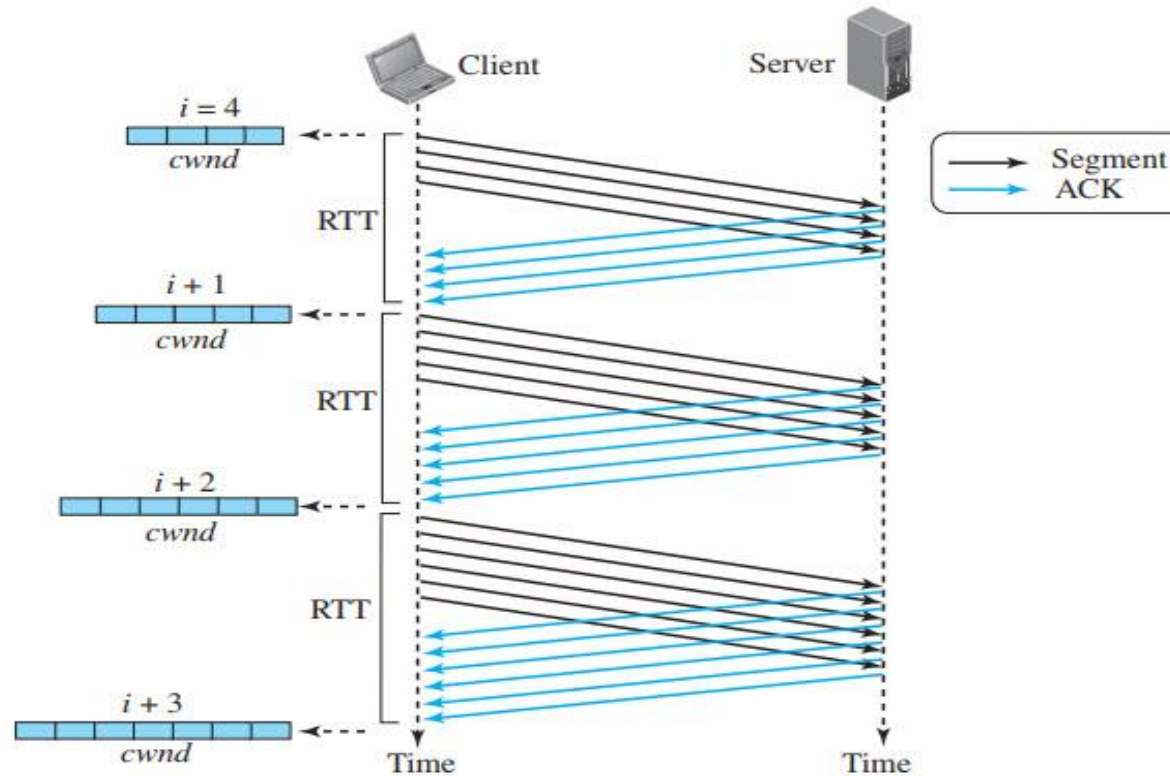


Figure: Congestion avoidance, additive increase.

Transmission Control Protocol [TCP] (Cntd.)

📖 Congestion Control: Congestion Policies (Cntd.)

➤ Fast Recovery:

- ✓ The fast-recovery algorithm is optional in TCP. The old version of TCP did not use it, but the new versions try to use it.
- ✓ It starts when three duplicate ACKs arrive, which is interpreted as light congestion in the network.
- ✓ Like congestion avoidance, this algorithm is also an additive increase, but it increases the size of the congestion window when a duplicate ACK arrives (after the three duplicate ACKs that trigger the use of this algorithm).

If a duplicate ACK arrives, $cwnd = cwnd + (1 / cwnd)$.

Transmission Control Protocol [TCP] (Cntd.)

Congestion Control: Congestion Policies (Cntd.)

➤ Policy Transition:

Tahoe TCP:

- ✓ The early TCP.
- ✓ Used only two different algorithms in their congestion policy: slow start and congestion avoidance.

Transmission Control Protocol [TCP] (Cntd.)

Congestion Control: Congestion Policies (Cntd.)

➤ Policy Transition (Cntd.):

Tahoe TCP (cntd.):

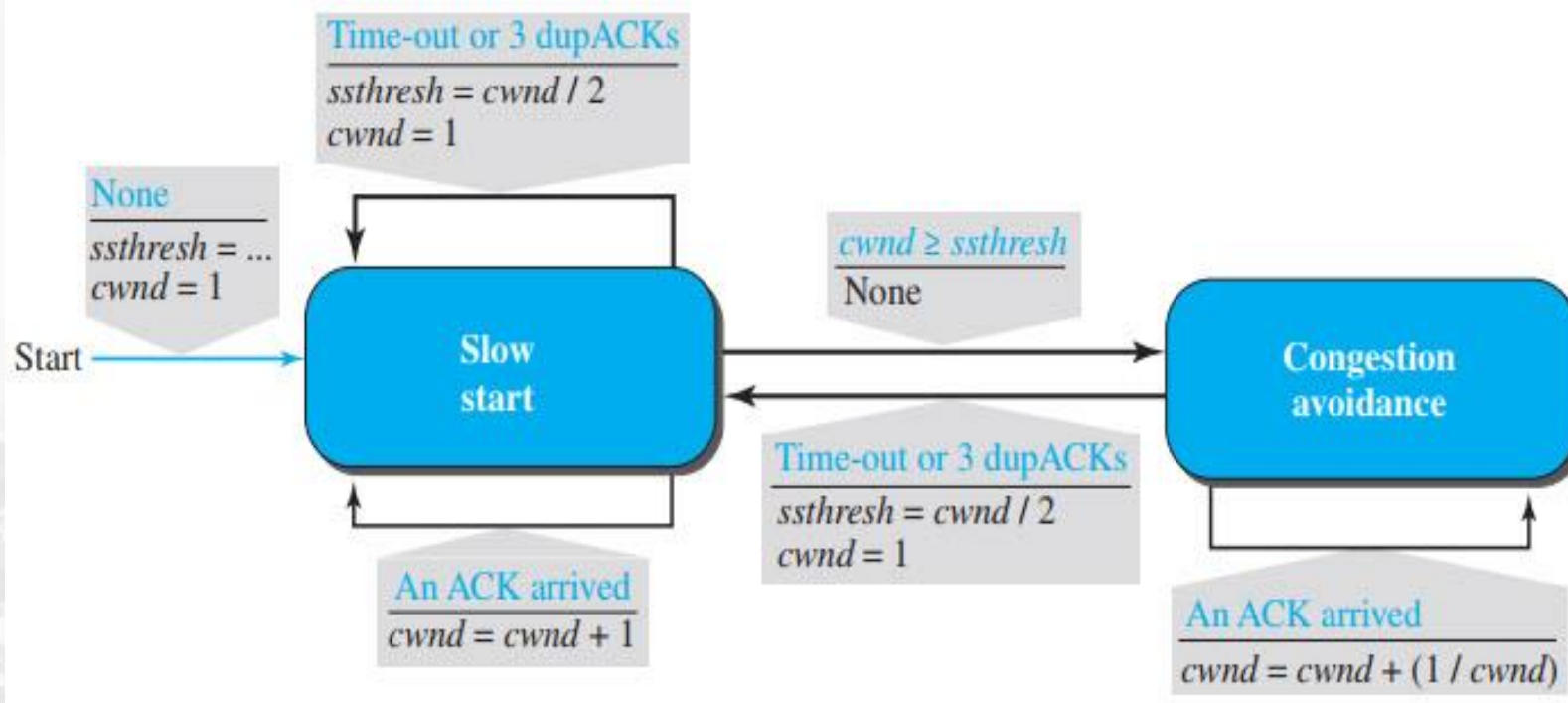


Figure: FSM for Tahoe TCP.

Transmission Control Protocol [TCP] (Cntd.)

Congestion Control: Congestion Policies (Cntd.)

➤ Policy Transition (Cntd.):

Reno TCP:

- ✓ Newer version of TCP.
- ✓ added a new state to the congestion-control FSM, called the fast-recovery state.
- ✓ if a time-out occurs, TCP moves to the slow-start state (or starts a new round if it is already in this state)
- ✓ if three duplicate ACKs arrive, TCP moves to the fast-recovery state and remains there as long as more duplicate ACKs arrive. The fast-recovery state is a state somewhere between the slow-start and the congestion-avoidance states.
- ✓ if a new (nonduplicate) ACK arrives, TCP moves to the congestion-avoidance state.

Transmission Control Protocol [TCP] (Cntd.)

Congestion Control: Congestion Policies (Cntd.)

➤ Policy Transition (Cntd.):

Reno TCP (cntd.):

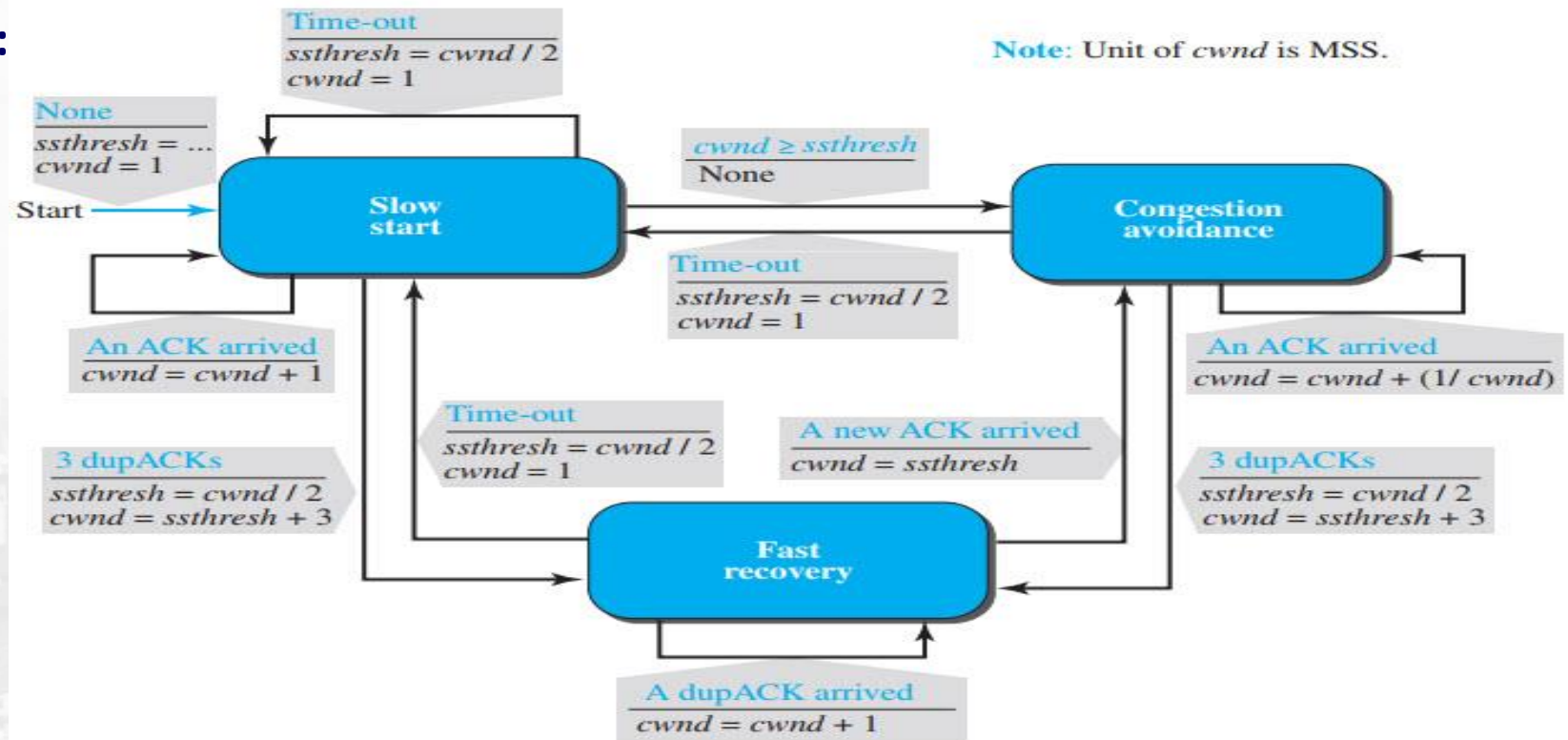


Figure: FSM for Reno TCP.

Transmission Control Protocol [TCP] (Cntd.)

Congestion Control: Congestion Policies (Cntd.)

➤ Policy Transition (Cntd.):

New Reno TCP:

- ✓ An extra optimization on the Reno TCP.
- ✓ TCP checks to see if more than one segment is lost in the current window when three duplicate ACKs arrive.
- ✓ When TCP receives three duplicate ACKs, it retransmits the lost segment until a new ACK (not duplicate) arrives.
- ✓ If the new ACK defines the end of the window when the congestion was detected, TCP is certain that only one segment was lost. However, if the ACK number defines a position between the retransmitted segment and the end of the window, it is possible that the segment defined by the ACK is also lost. New Reno TCP retransmits this segment to avoid receiving more and more duplicate ACKs for it.

Transmission Control Protocol [TCP] (Cntd.)

🖥️ Congestion Control: Congestion Policies (Cntd.)

➤ Policy Transition (Cntd.):

Additive Increase, Multiplicative Decrease (AIMD):

- ✓ Out of the three versions of TCP, the Reno version is most common today.
- ✓ It has been observed that, in this version, most of the time the congestion is detected and taken care of by observing the three duplicate ACKs.
- ✓ Even if there are some time-out events, TCP recovers from them by aggressive exponential growth.
- ✓ That is, if we ignore the slow-start states and short exponential growth during fast recovery, the TCP congestion window is:
 1. when an ACK arrives (congestion avoidance): $cwnd = cwnd + (1 / cwnd)$. (Additive Increase)
 2. when congestion is detected: $cwnd = cwnd / 2$. (Multiplicative Decrease)
- ✓ This means that the congestion window size, after it passes the initial slow-start state, follows a saw tooth pattern called additive increase, multiplicative decrease (AIMD). (See Figure)

Transmission Control Protocol [TCP] (Cntd.)

🖥️ Congestion Control: Congestion Policies (Cntd.)

➤ Policy Transition (Cntd.):

Additive Increase, Multiplicative Decrease (AIMD):

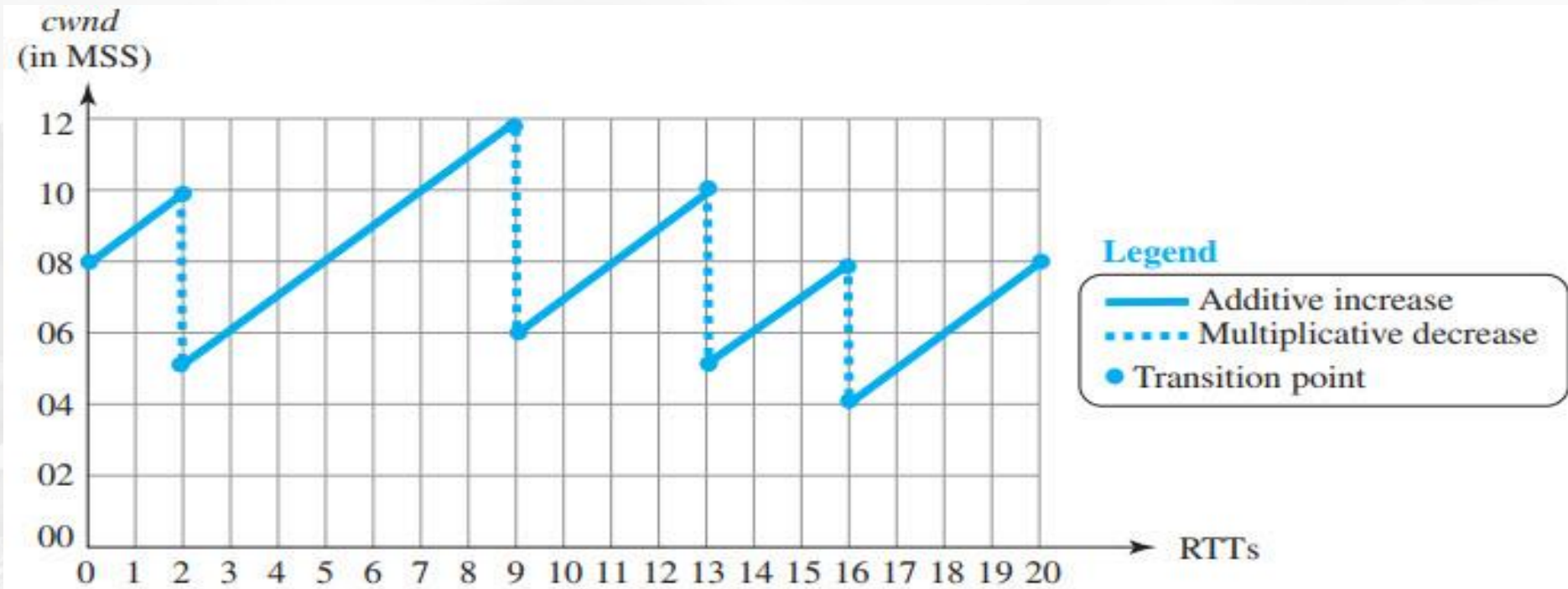


Figure: Additive increase, multiplicative decrease (AIMD).

Transmission Control Protocol [TCP] (Cntd.)

Congestion Control: (Cntd.)

➤ Actual timeout events behavior :

State	Event	Sender Action	Comment
Slow Start (SS)	New ACK received	Congestion Window = Congestion Window + MSS. if (Congestion Window \geq Threshold) set state to CA.	Congestion Window doubles every RTT.
Congestion Avoidance (CA)	New ACK received	Congestion Window = Congestion Window + $MSS(MSS/Congestion\ Window)$.	Congestion Window increases by 1 MSS every RTT.
SS or CA	Triple Duplicate ACK	Threshold = Congestion Window/2. Congestion Window = Threshold. set state to CA.	Fast recover; multiplicative decrease.
SS or CA	Timeout	Threshold = Congestion Window/2. Congestion Window = 1 MSS. Set state to SS.	
SS or CA	Duplicate ACK received	Increment duplicate ACK count for segment.	

Transmission Control Protocol [TCP] (Cntd.)

- Multiple senders sending data streams quickly into network results in congestion, which eventually leads to degraded performance of the network because of delayed or lost packets.
- Therefore Controlling congestion is the combined responsibility of the network and transport layers.
- Congestion occurs at routers so it is detected at network layer, however transport layer traffic sent thorough network layer is mainly responsible for congestion.
- Therefore effective way to control congestion is to send packets slowly into network.

Transmission Control Protocol [TCP] (Cntd.)

Desirable Bandwidth Allocation:

- Our goal is not only to avoid congestion but also to find a good allocation of bandwidth to the transport entities that are using the network.
- Good allocation of bandwidth ensures uses of all available bandwidth with congestion avoidance.

Transmission Control Protocol [TCP] (Cntd.)

Desirable Bandwidth Allocation (Cntd.): Efficiency and Power

- the goodput (or rate of useful packets arriving at the receiver) as a function of the offered load.
- the delay as a function of the offered load

Maximum Carrying Capacity of Subnet

Packets Delivered
i.e.
Throughput

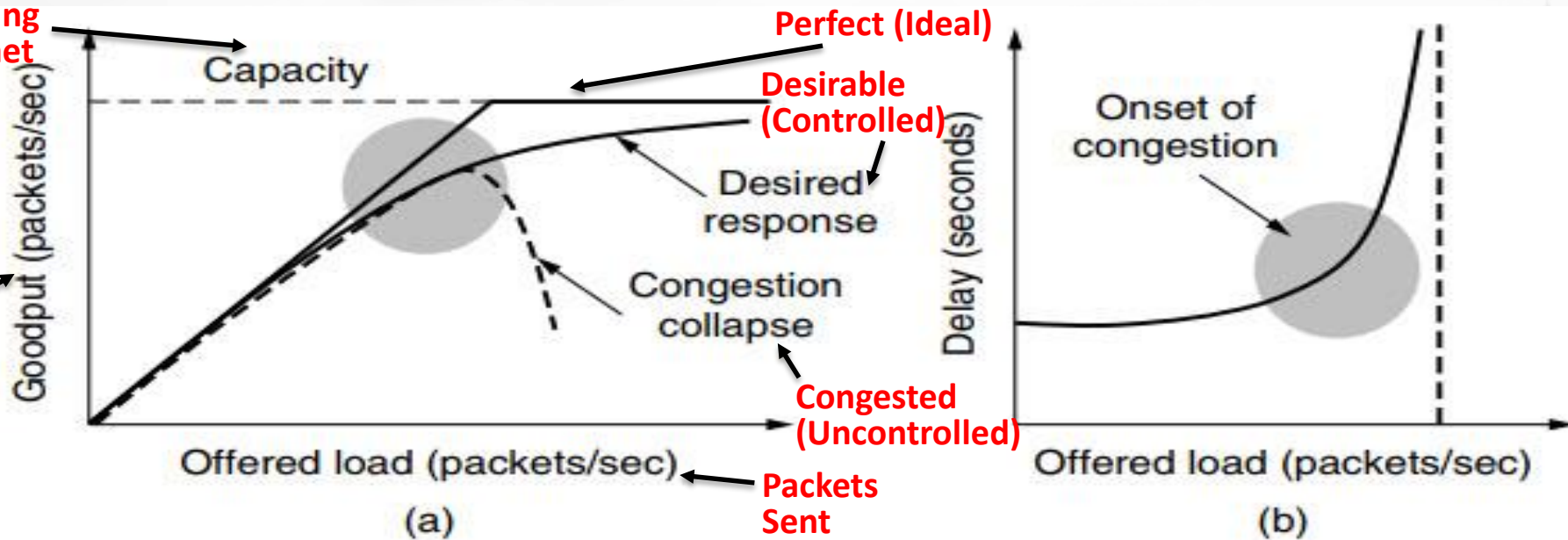


Figure: (a) Goodput and (b) delay as a function of offered load.

Transmission Control Protocol [TCP] (Cntd.)

Desirable Bandwidth Allocation (Cntd.): Efficiency and Power

➤ Kleinrock (1979) proposed the metric of power:

$$power = \frac{load}{delay}$$

Transmission Control Protocol [TCP] (Cntd.)

Desirable Bandwidth Allocation (Cntd.): Max-Min Fairness

- How to divide bandwidth between different transport senders?
- There are several considerations
 1. What this problem has to do with congestion control? IP routers often have all connections competing for the same bandwidth. In this situation, it is the congestion control mechanism that is allocating bandwidth to the competing connections.
 2. What a fair portion means for flows in a network?
 3. We will adopt a notion of fairness that does not depend on the length of the network path.
- The form of fairness that is often desired for network usage is **max-min fairness**.
- An allocation is max-min fair if the bandwidth given to one flow cannot be increased without decreasing the bandwidth given to another flow with an allocation that is no larger.

Transmission Control Protocol [TCP] (Cntd.)

Desirable Bandwidth Allocation (Cntd.): Max-Min Fairness (Cntd.)

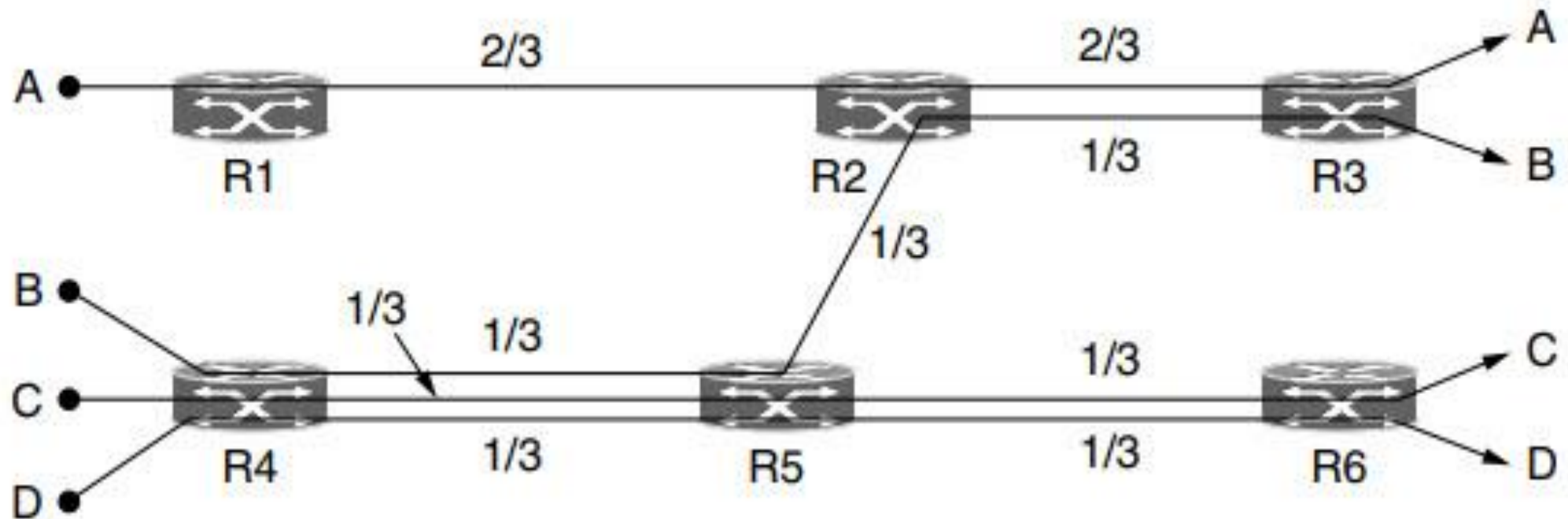


Figure: Max-min bandwidth allocation for four flows.

Transmission Control Protocol [TCP] (Cntd.)

Desirable Bandwidth Allocation (Cntd.): Convergence

- The congestion control algorithm converge quickly to a fair and efficient allocation of bandwidth.
- Connections are always coming and going in a network, and the bandwidth needed by a given connection will vary over time too.
- Because of the variation in demand, the ideal operating point for the network varies over time. A good congestion control algorithm should rapidly converge to the ideal operating point, and it should track that point as it changes over time.
- Check example on next slide.....

Transmission Control Protocol [TCP] (Cntd.)

Desirable Bandwidth Allocation (Cntd.): Convergence (Cntd.)

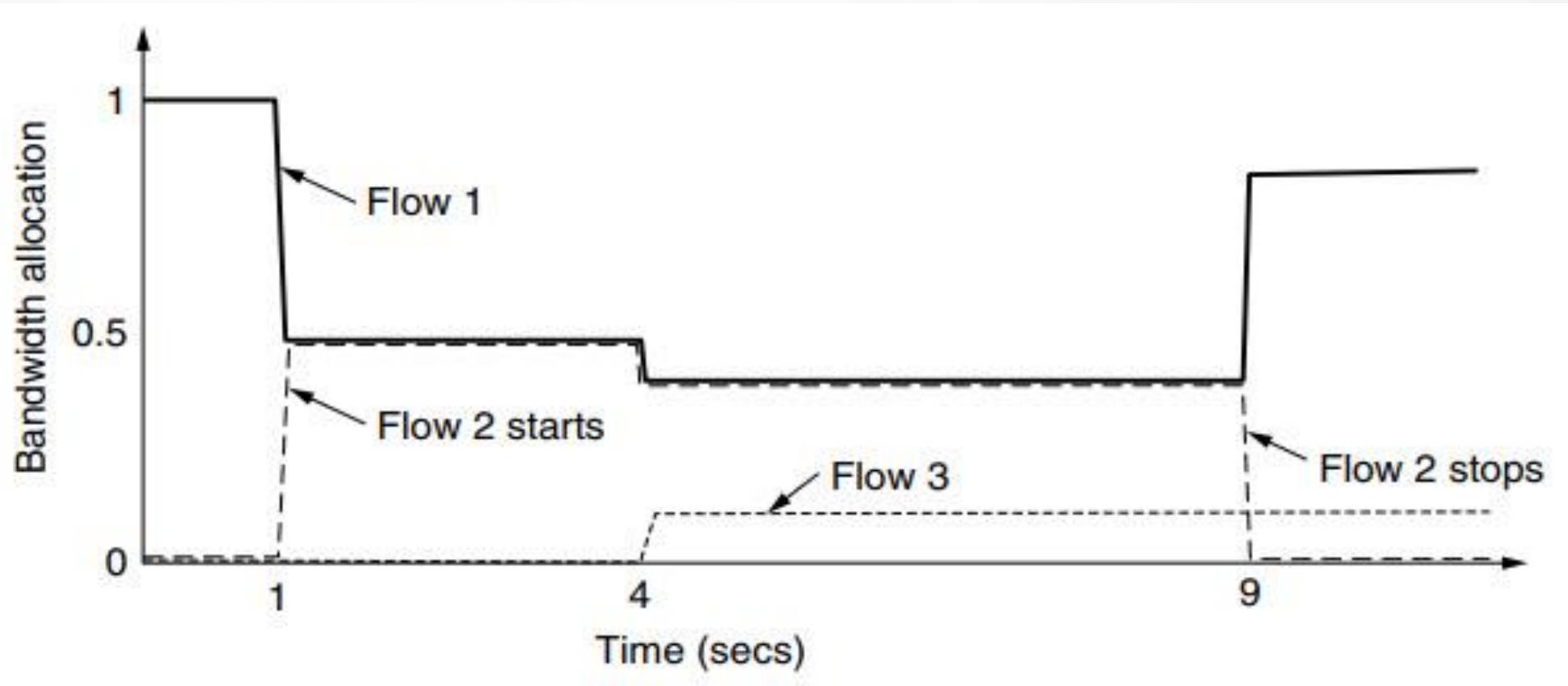


Figure: Changing bandwidth allocation over time.

Transmission Control Protocol [TCP] (Cntd.)

Regulating the Sending Rate:

- The sending rate may be limited by two factors.
 - ✓ The first is flow control, in the case that there is insufficient buffering at the receiver. (Refer figure)
 - ✓ The second is congestion, in the case that there is insufficient capacity in the network. (Refer figure)

Transmission Control Protocol [TCP] (Cntd.)

Regulating the Sending Rate (Cntd.):

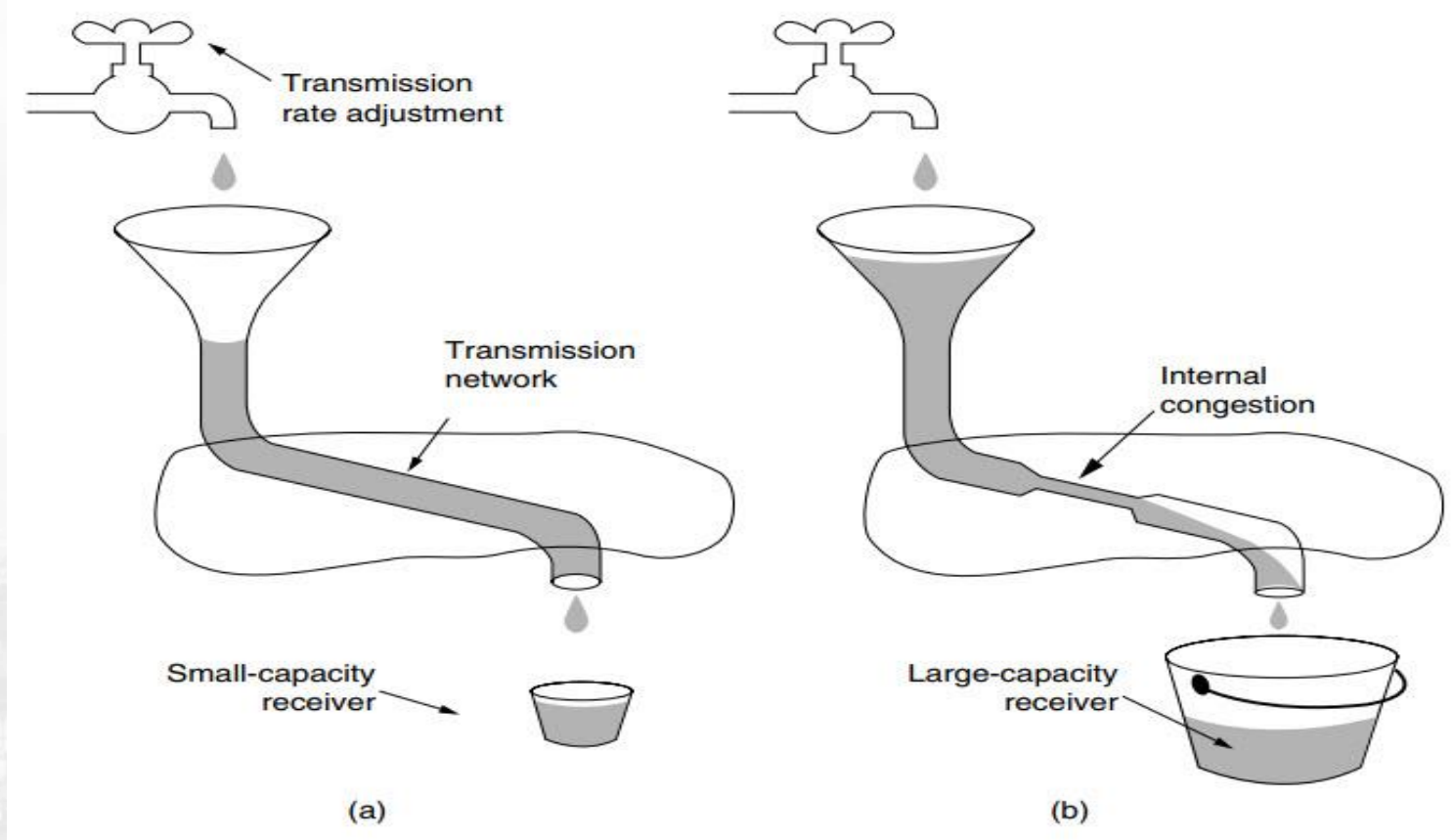


Figure: (a) A fast network feeding a low-capacity receiver. (b) A slow network feeding a high-capacity receiver.

Transmission Control Protocol [TCP] (Cntd.)

Regulating the Sending Rate (Cntd.):

- The way that a transport protocol should regulate the sending rate depends on the form of the feedback returned by the network. Different network layers may return different kinds of feedback. The feedback may be explicit or implicit, and it may be precise or imprecise.

Protocol	Signal	Explicit?	Precise?
XCP	Rate to use	Yes	Yes
TCP with ECN	Congestion warning	Yes	No
FAST TCP	End-to-end delay	No	Yes
Compound TCP	Packet loss & end-to-end delay	No	Yes
CUBIC TCP	Packet loss	No	No
TCP	Packet loss	No	No

Figure: Signals of some congestion control protocols.

Transmission Control Protocol [TCP] (Cntd.)

Regulating the Sending Rate (Cntd.): Leaky Bucket

- A leaky bucket algorithm shapes bursty traffic into fixed-rate traffic by averaging the data rate. It may drop the packets if the bucket is full.

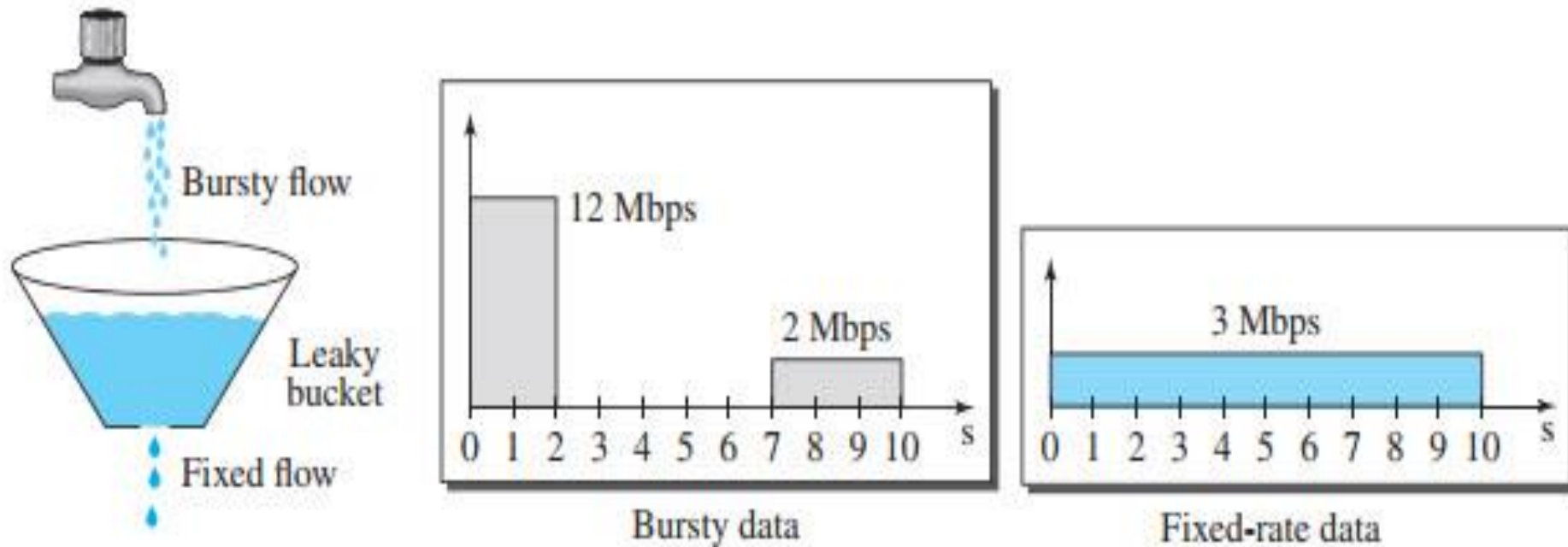


Figure: Leaky bucket.

Transmission Control Protocol [TCP] (Cntd.)

Regulating the Sending Rate (Cntd.): Token Bucket

- The token bucket allows bursty traffic at a regulated maximum rate.

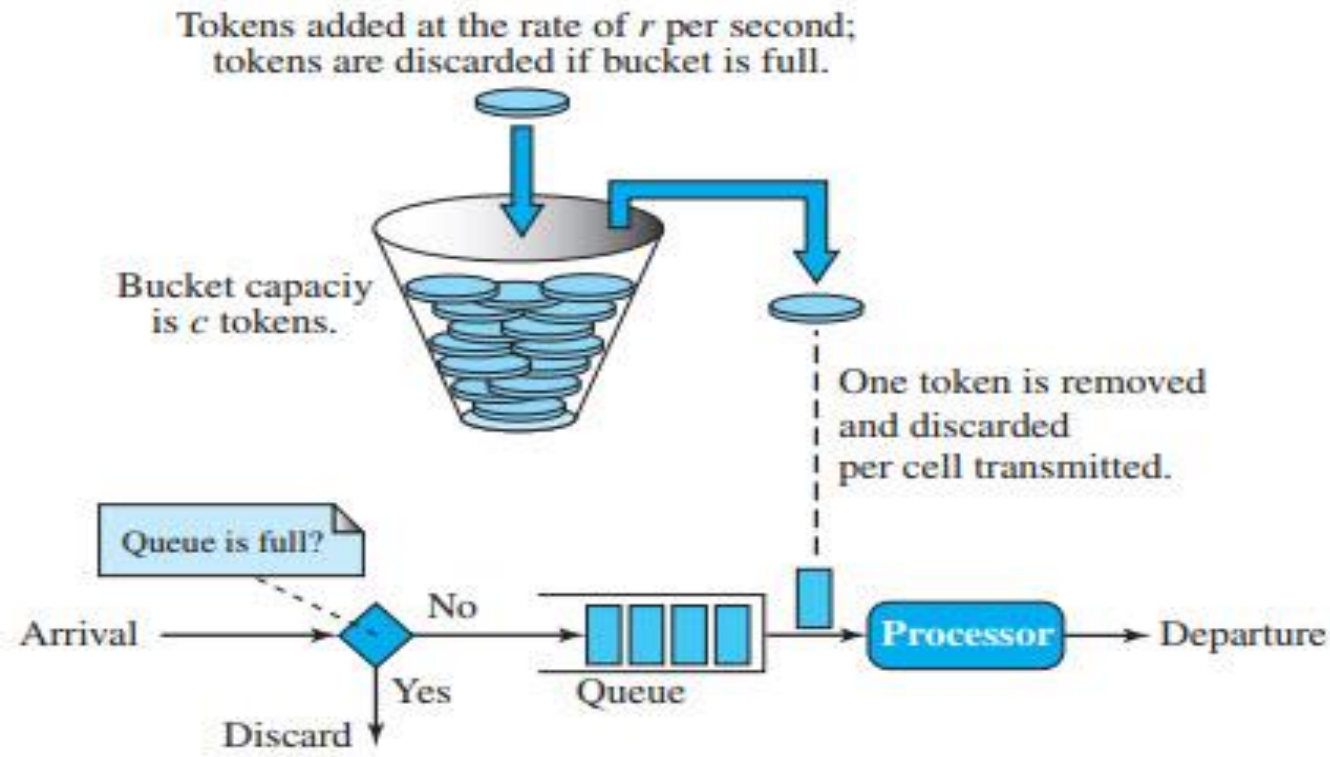



Figure: Token bucket.

Transmission Control Protocol [TCP] (Cntd.)

-  **Regulating the Sending Rate (Cntd.): Combining Token Bucket and Leaky Bucket**
- **The two techniques can be combined to credit an idle host and at the same time regulate the traffic. The leaky bucket is applied after the token bucket; the rate of the leaky bucket needs to be higher than the rate of tokens dropped in the bucket.**

Quality of Service [QoS]

- Another way of looking at the transport layer is to regard its primary function as enhancing the QoS (Quality of Service) provided by the network layer.
- If the network service is impeccable, the transport layer has an easy job. If however, the network service is poor, the transport layer has to bridge the gap between what the transport users want and what the network layer provides.
- The transport service may allow the user to specify preferred, acceptable, and minimum values for various service parameters at the time a connection is setup.
- Now we will discuss some of the QoS parameters.

Quality of Service [QoS] (Cntd.)

QoS parameters

- The connection establishment delay:
- The connection establishment failure probability:
- Throughput:
- Transit delay:
- Residual error ratio:
- Protection:
- Priority:
- Resilience:

Questions on Transport Layer

- 1. List out services provide by transport layer?**
- 2. Explain connection establishment and connection release in transport layer?**
- 3. Draw TCP Header and explain all fields of TCP header?**
- 4. Explain Real-time Transport Protocol in detail?**
- 5. Explain concepts of Socket and Ports?**
- 6. Explain Concept of 3-Way Handshaking?**
- 7. Discuss UDP Applications?**
- 8. Explain Silly Window Syndrome Problem?**
- 9. How Error Control is achieved in TCP?**
- 10. Draw and Explain UDP Frame format?**
- 11. Discuss some of the QoS parameters at Transport Layer?**

NPTEL Video Lecture Links on Transport Layer

1. <https://www.youtube.com/watch?v=8-3CSAkcYU>
2. <https://www.youtube.com/watch?v=fBPDLGwfSUM>
3. <https://www.youtube.com/watch?v=c8Wv3b7f9XY>
4. <https://www.youtube.com/watch?v=qclg6FY-FGM>
5. <https://www.youtube.com/watch?v=VUdfS70puWI>
6. <https://www.youtube.com/watch?v=bKHRbqwkMkg>
7. <https://www.youtube.com/watch?v=b-nCA0tZU-0>
8. <https://www.youtube.com/watch?v=EqzDTO9tdqs>
9. <https://www.youtube.com/watch?v=5ex1s4lURto>
10. <https://www.youtube.com/watch?v=xS202cWCc2Q>
11. <https://www.youtube.com/watch?v=8NxJGHXDOGc>
12. <https://www.youtube.com/watch?v=SHO9eeWxPxY>
13. <https://www.youtube.com/watch?v=f1y25BfOH9I>
14. <https://www.youtube.com/watch?v=x7obJLN7FcQ>
15. <https://www.youtube.com/watch?v=JExfKvUgrtl>

NPTEL Web Links on Transport Layer

1. <https://nptel.ac.in/courses/106106091/36>
2. <https://nptel.ac.in/courses/106106091/37>
3. <https://nptel.ac.in/courses/106106091/38>
4. <https://nptel.ac.in/courses/106106091/39>
5. <https://nptel.ac.in/courses/106105080/32>