Module2

Transport Layer

Transport Layer

- 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;
- it is the end-to-end logical vehicle for transferring data from one point to another in the Internet.

Transport-Layer Services

- Process-to-Process Communication
- Addressing: Port Numbers
- Encapsulation and Decapsulation
- Multiplexing and Demultiplexing
- Flow Control
- Error Control
- Congestion Control

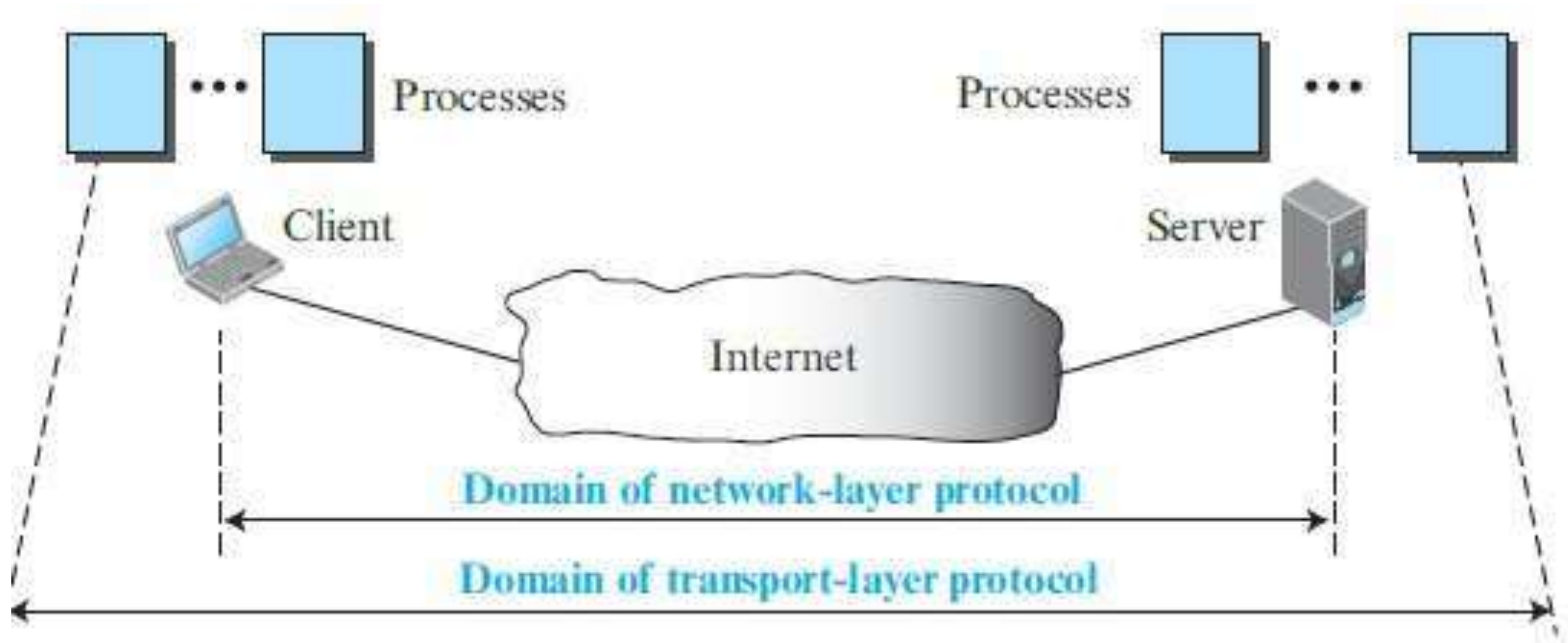
process-to-process communication

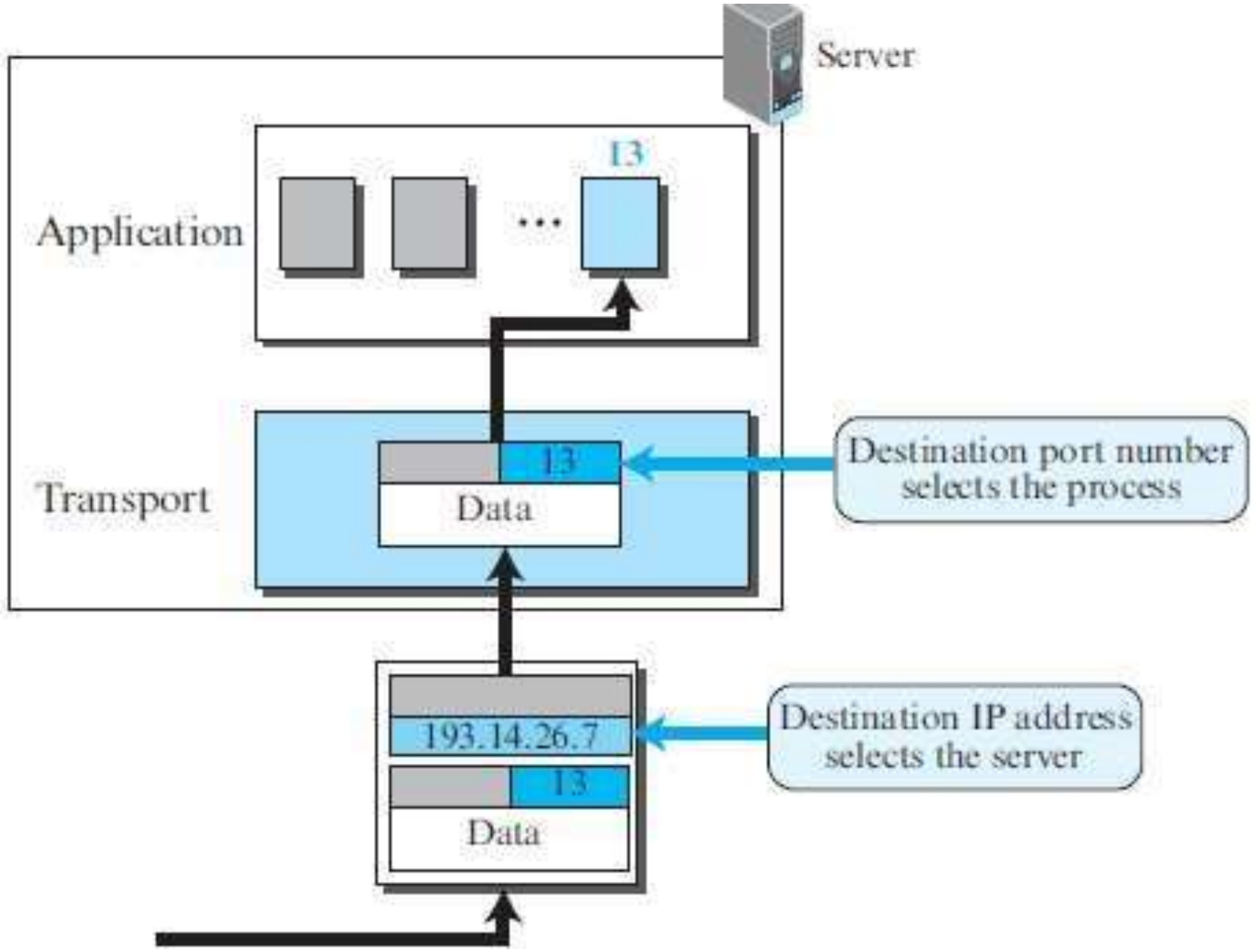
- The first duty of a transport-layer protocol is to provide process-to- process communication.
- A process is an application-layer entity (running program) that uses the services of the transport layer.
- A process on the local host, called a client, needs services from a process usually on the remote host, called a server.
- Both processes (client and server) have the same name.

Difference between host-to-host communication and process-to-process communication

- The network layer is responsible for communication at the computer level (host-to-host communication).
- A network-layer protocol can deliver the message only to the destination computer.
- The message still needs to be handed to the correct process. This is done by transport-layer protocol.
- A transport-layer protocol is responsible for delivery of the message to the appropriate process.
- The destination IP address defines the host among the different hosts in the world.
- After the host has been selected, the port number defines one of the processes on this particular host.

Network layer versus transport layer

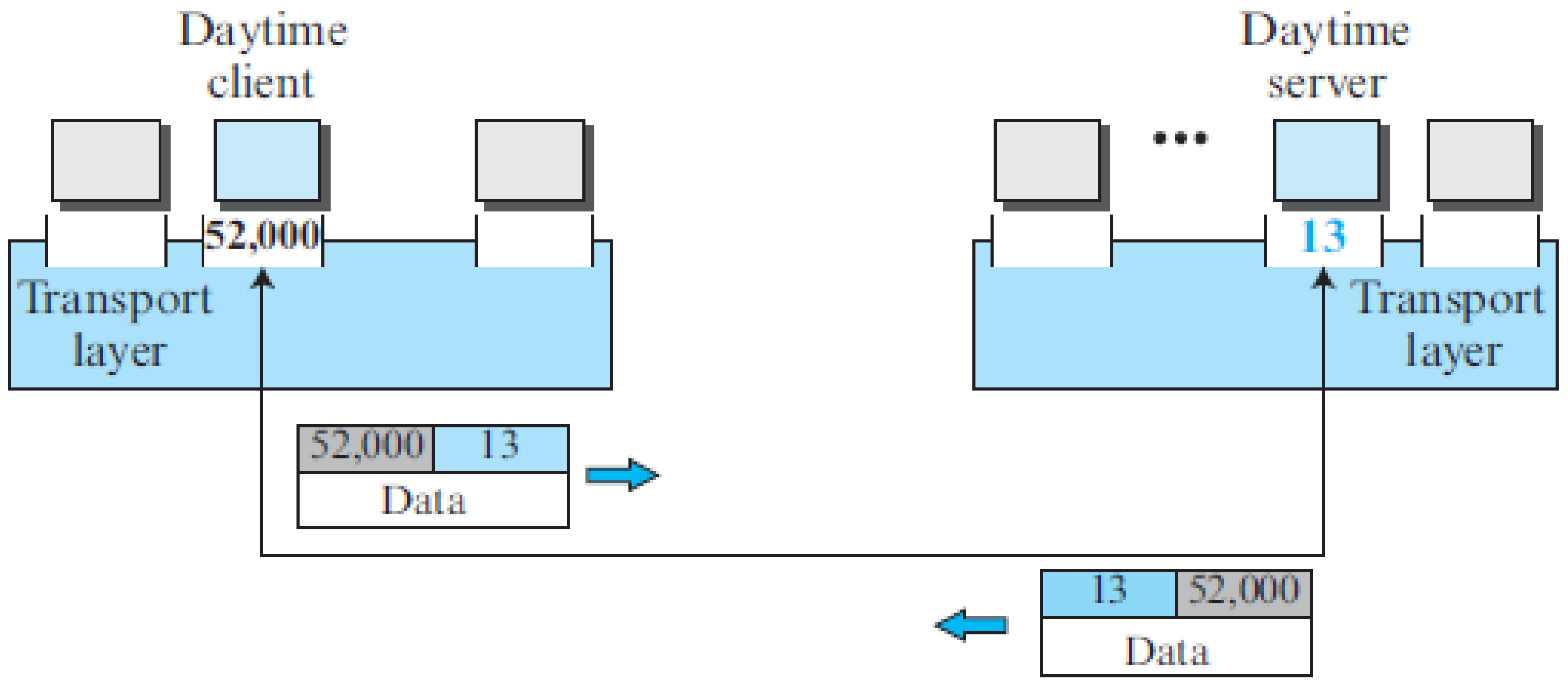

Processes
Processes
Client
Server
Internet
Domain of network-layer protocol
Domain of transport-layer protocol


Server
Application
13
...
Transport
13
Data
Destination port number selects the process
193.14.26.7
13
Data
Destination IP address selects the server

Addressing: Port Numbers

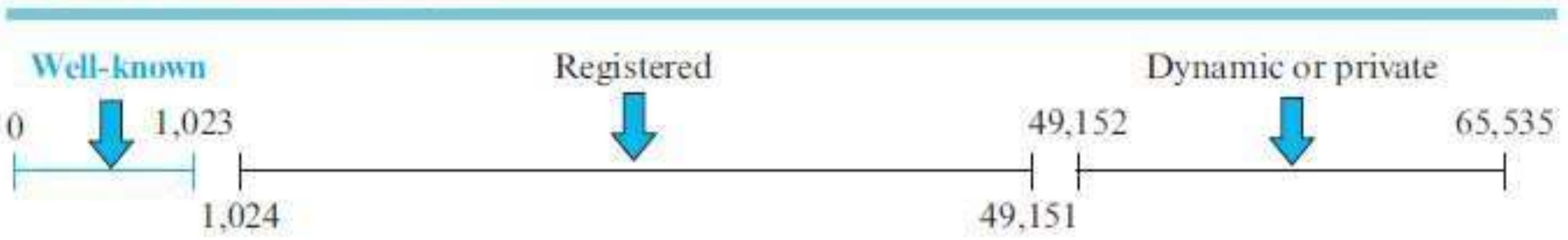
- For communication, we must define the local host, local process, remote host, and remote process.
- The local host and the remote host are defined using IP addresses .
- To define the processes, we need second identifiers, called port numbers.
- In the TCP/IP protocol suite, the port numbers are integers between 0 and 65,535 (16 bits).
- The client program defines itself with a port number, called the ephemeral port number.
- The word ephemeral means short-lived and is used because the life of a client is normally short.
- An ephemeral port number is recommended to be greater than

- The server process must also define itself with a port number.
- This port number cannot be chosen randomly.
- TCP/IP has decided to use universal port numbers for servers; these are called well-known port numbers.



- ICANN has divided the port numbers into three ranges: well-known, registered, and dynamic (or private).

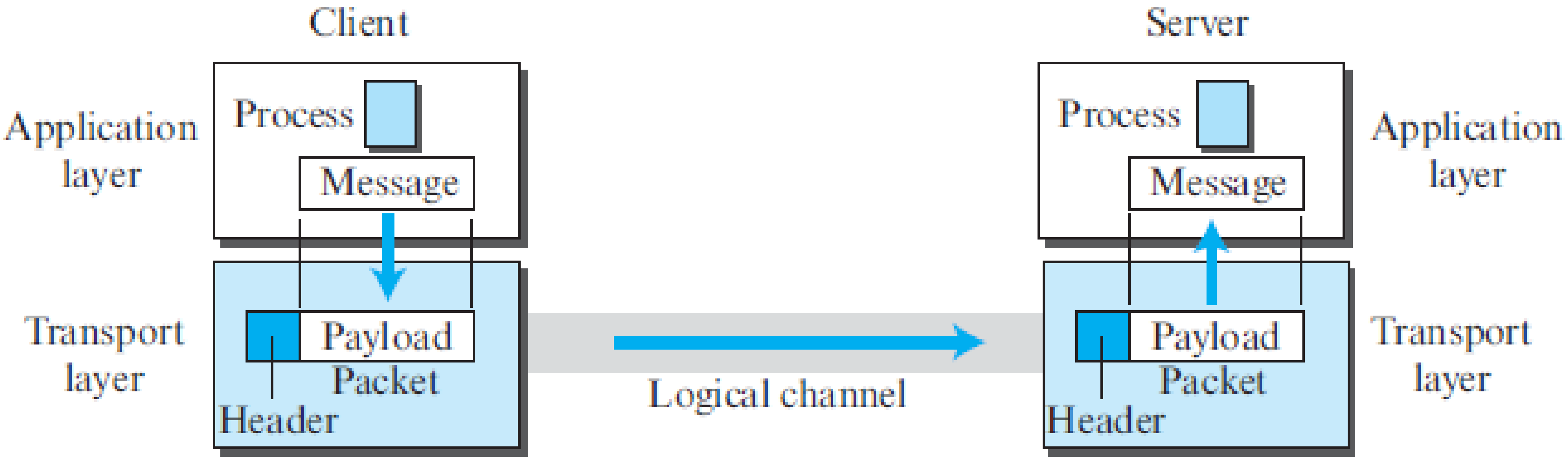
Figure 3.5 *ICANN ranges*



- The ports ranging from 0 to 1,023 are assigned and controlled by ICANN. These are the well-known ports.
- The ports ranging from 1,024 to 49,151 are not assigned or controlled by ICANN. They can only be registered with ICANN to prevent duplication.
- The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used as temporary or private port numbers.

Encapsulation and Decapsulation

- To send a message from one process to another, the transport-layer protocol encapsulates and decapsulates messages.
- Encapsulation happens at the sender site.
- When a process has a message to send, it passes the message to the transport layer along with a pair of socket addresses and some other pieces of information, which depend on the transport-layer protocol.
- The transport layer receives the data and adds the transport-layer header.
- The packets at the transport layers in the Internet are called user datagrams, segments, or packets, depending on what transport-layer protocol we use.



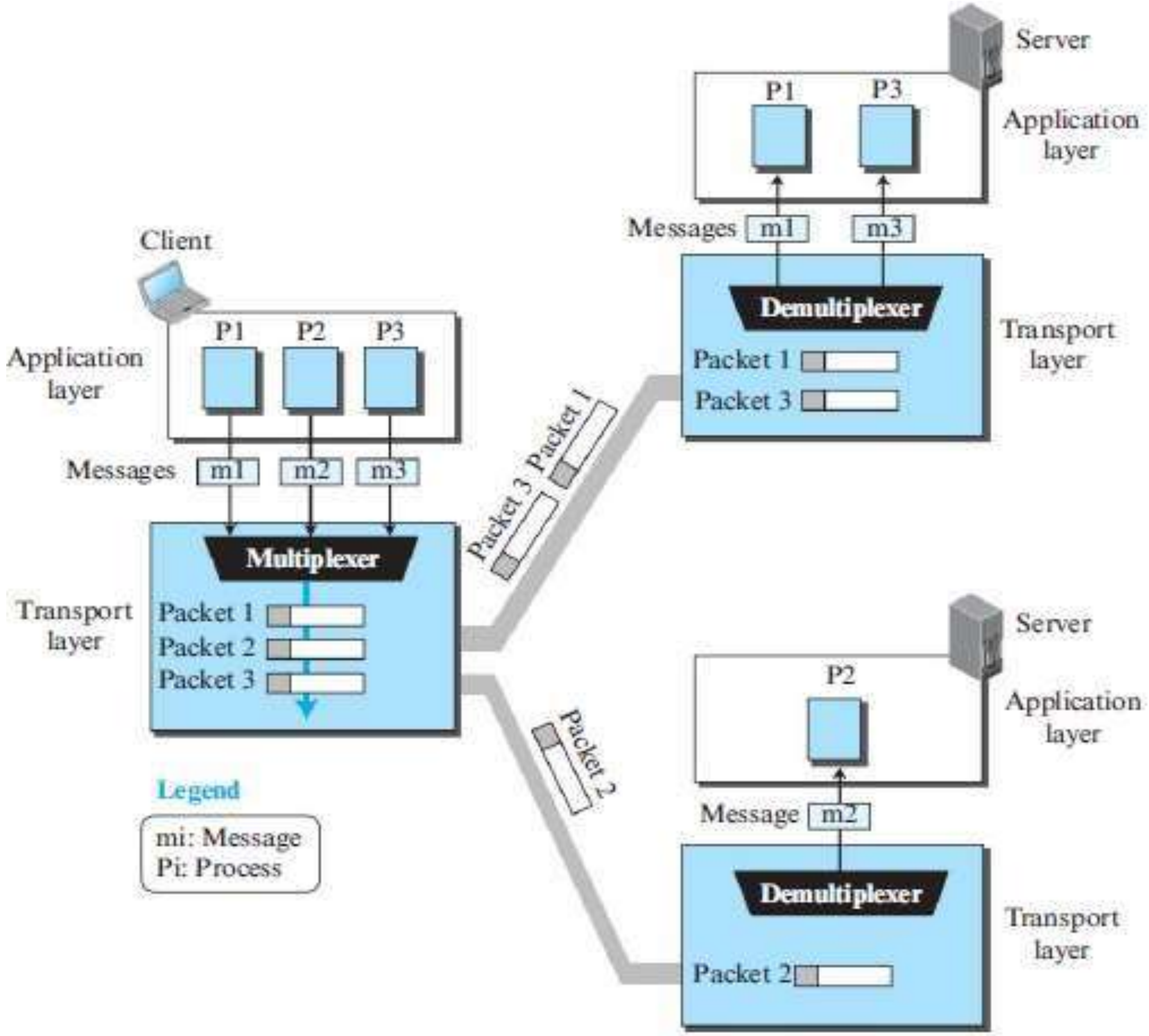
a. Encapsulation

b. Decapsulation

- Decapsulation happens at the receiver site.
- When the message arrives at the destination transport layer, the header is dropped and the transport layer delivers the message to the process running at the application layer.
- The sender socket address is passed to the process in case it needs to respond to the message received.

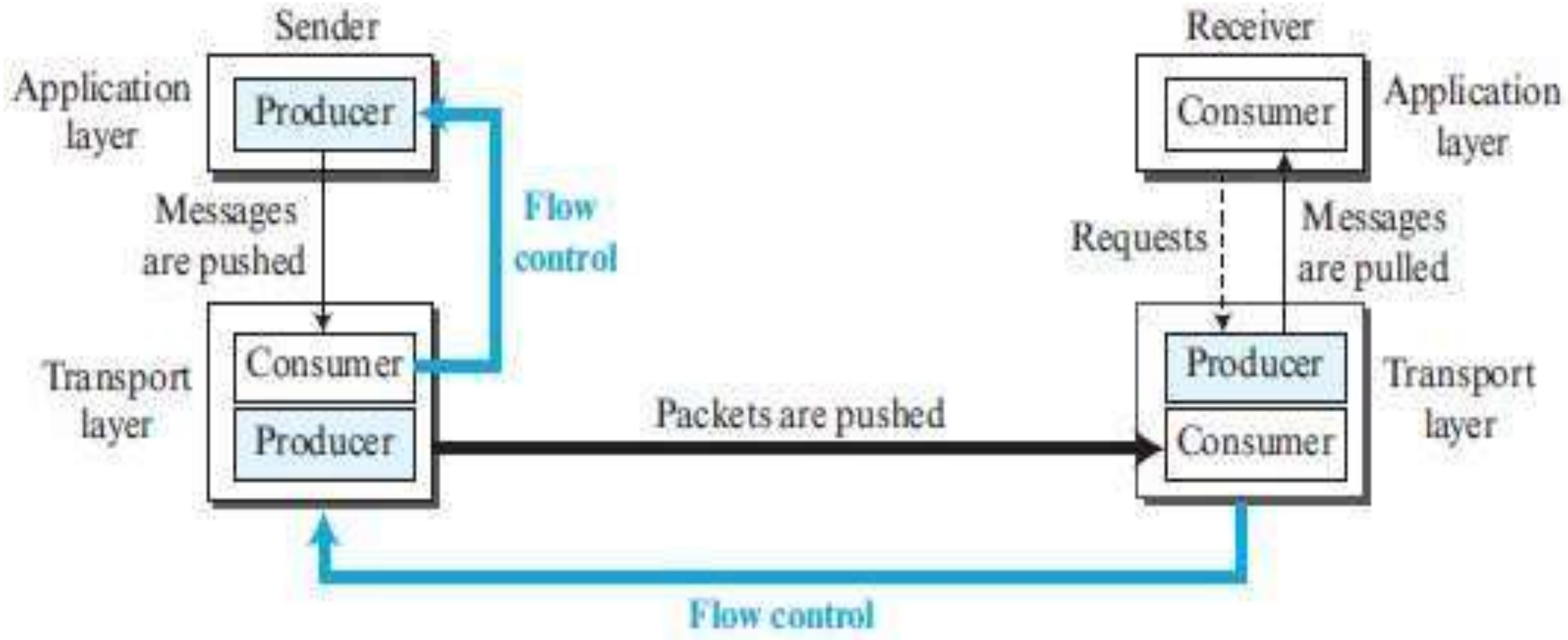
Multiplexing and Demultiplexing

- 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).
- The transport layer at the source performs multiplexing; the transport layer at the destination performs demultiplexing.
- Although there is only one message, we use demultiplexer.


Server
P1
P3
Application layer
Messages
m1
m3
Demultiplexer
Transport layer
Packet 1
Packet 3
Client
P1
P2
P3
Application layer
Messages
m1
m2
m3
Multiplexer
Transport layer
Packet 1
Packet 2
Packet 3
Packet 3
Packet 1
Packet 2
Server
P2
Application layer
Message
m2
Demultiplexer
Transport layer
Packet 2
Legend
mi: Message
Pi: Process

Flow Control

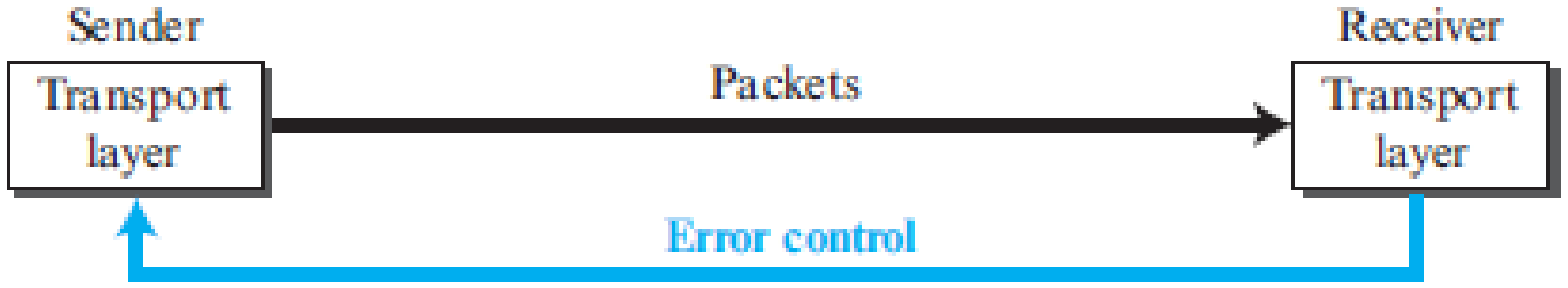
- If the items are produced faster than they can be consumed, the consumer can be overwhelmed and may need to discard some items.
- Flow control is related to this issue.
- We need to prevent losing the data items at the consumer site.
- If the sender delivers items whenever they are produced without a prior request from the consumer–the delivery is referred to as pushing.
- If the producer delivers the items after the consumer has requested them, the delivery is referred to as pulling.
- When the producer pushes the items, the consumer may be overwhelmed and there is a need for flow control, in the opposite direction, to prevent discarding of the items.
- we need at least two cases of flow control: from the sending transport layer to the sending application layer and from the receiving transport layer to the sending transport layer.


Sender
Application layer
Producer
Messages are pushed
Flow control
Transport layer
Consumer
Producer
Packets are pushed
Receiver
Consumer
Application layer
Requests
Messages are pulled
Producer
Consumer
Transport layer
Flow control

- One of the solutions is normally to use two buffers: one at the sending transport layer and the other at the receiving transport layer.
- A buffer is a set of memory locations that can hold packets at the sender and receiver.
- The flow control communication can occur by sending signals from the consumer to the producer.
- When the buffer of the sending transport layer is full, it informs the application layer to stop passing chunks of messages; when there are some vacancies, it informs the application layer that it can pass message chunks again.
- When the buffer of the receiving transport layer is full, it informs the sending transport layer to stop sending packets. When there are some vacancies, it informs the sending transport layer that it can send packets again.

Error Control

- In the Internet, since the underlying network layer (IP) is unreliable, we need to make the transport layer reliable if the application requires reliability.
- Reliability can be achieved to add error control services to the transport layer.
- Error control at the transport layer is responsible for
- 1. Detecting and discarding corrupted packets.
- 2. Keeping track of lost and discarded packets and resending them.
- 3. Recognizing duplicate packets and discarding them.
- 4. Buffering out-of-order packets until the missing packets arrive.

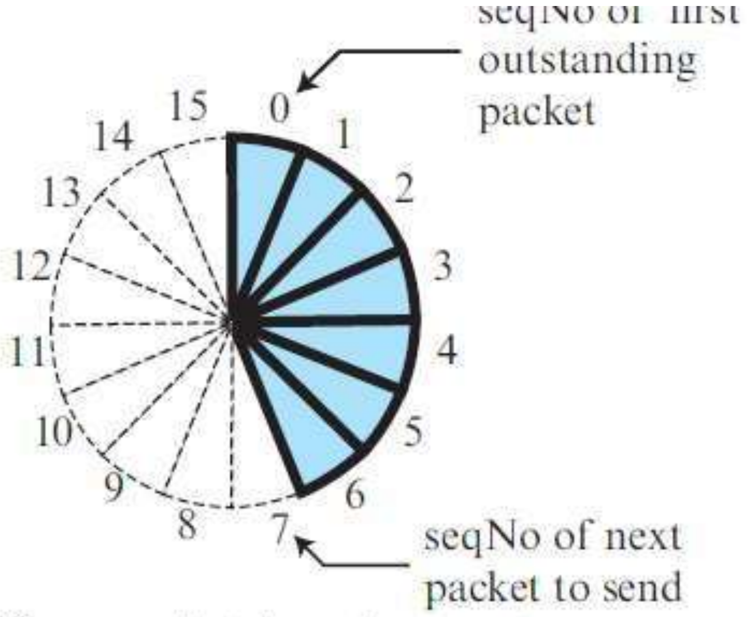


- We can add a field to the transport-layer packet to hold the sequence number of the packet.
- When a packet is corrupted or lost, the receiving transport layer can somehow inform the sending transport layer to resend that packet using the sequence number.
- For error control, the sequence numbers are modulo 2^m, where m is the size of the sequence number field in bits.
- The receiver side can send an acknowledgment (ACK) for each of a collection of packets.
- The sender can detect lost packets if it uses a timer.
- When a packet is sent, the sender starts a timer. If an ACK does not arrive before the timer expires, the sender resends the packet.

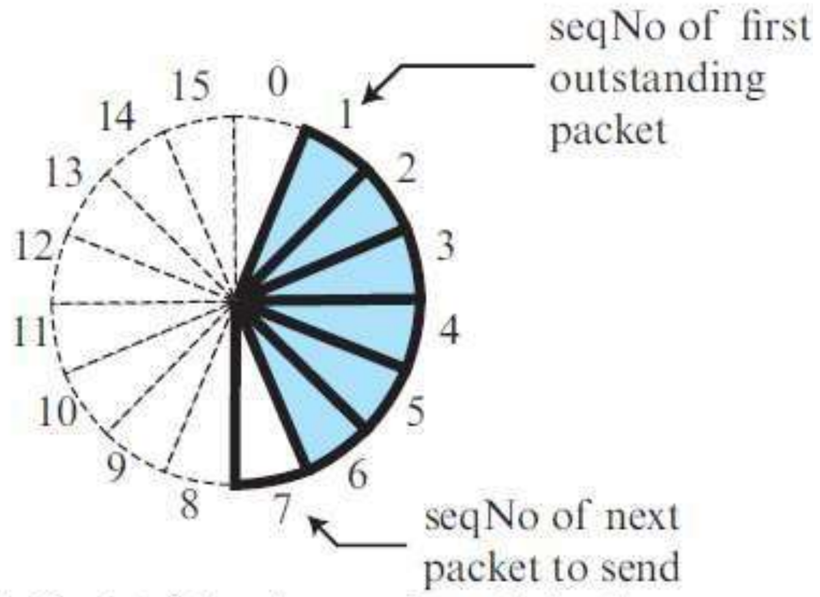
- These two requirements can be combined if we use two numbered buffers, one at the sender, one at the receiver.
- At the sender, when a packet is prepared to be sent, we use the number of the next free location, x, in the buffer as the sequence number of the packet.
- When the packet is sent, a copy is stored at memory location x, awaiting the acknowledgment from the other end.
- When an acknowledgment related to a sent packet arrives, the packet is purged and the memory location becomes free.
- At the receiver, when a packet with sequence number y arrives, it is stored at the memory location y until the application layer is ready to receive it.
- An acknowledgment can be sent to announce the arrival of packet y.

Sliding Window

- A circle can represent the sequence numbers from 0 to 2^m − 1
- The buffer is represented as a set of slices, called the sliding window, that occupies part of the circle at any time.



. Seven packets have been sent;
window is full.



d. Packet 0 has been acknowledged;
window slides.

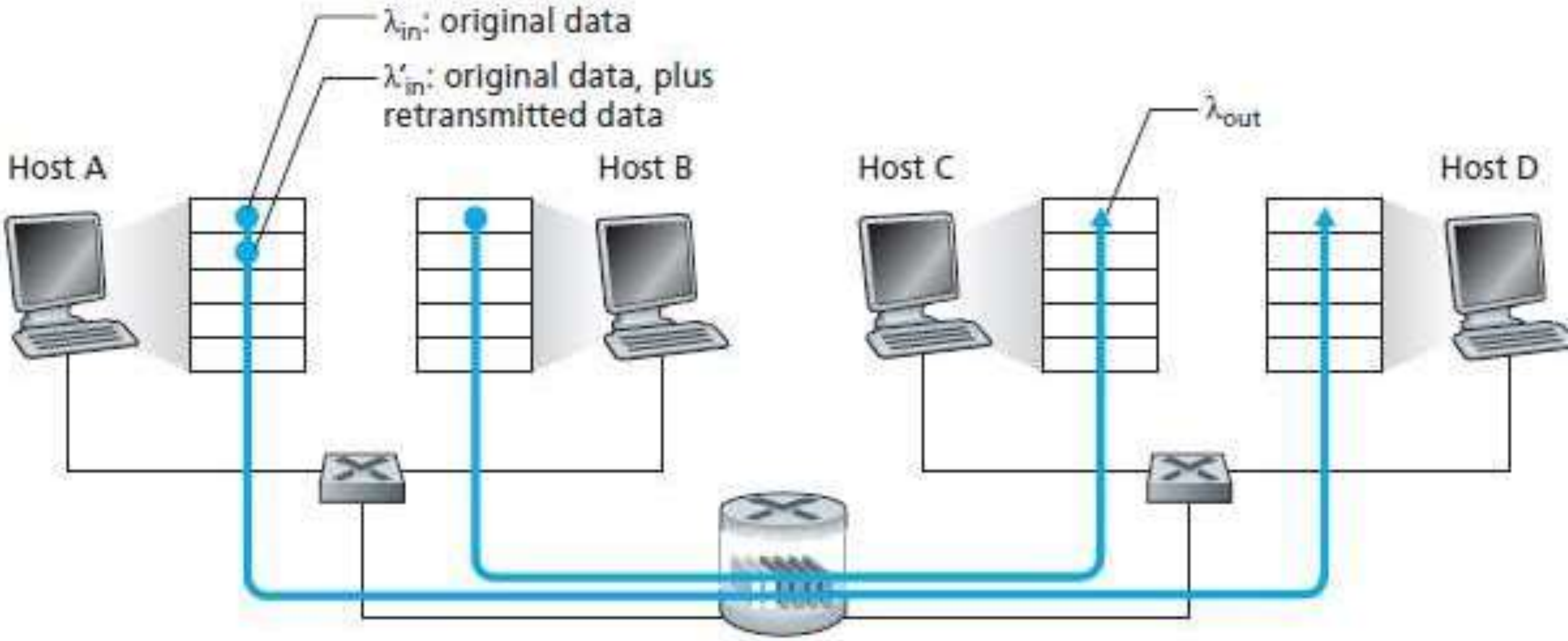
- At the sender site, when a packet is sent, the corresponding slice is marked.
- When an acknowledgment arrives, the corresponding slice is unmarked.

Congestion Control

- Congestion in a network may occur if the load on the network—the number of packets sent to the network—is greater than the capacity of the network—the number of packets a network can handle.
- Congestion control refers to the mechanisms and techniques that control the congestion and keep the load below the capacity.
- Congestion in a network or internetwork occurs because routers and switches have queues—buffers that hold the packets before and after processing.
- A router, for example, has an input queue and an output queue for each interface.
- If a router cannot process the packets at the same rate at which they arrive, the queues become overloaded and congestion occurs.
- Congestion at the transport layer is actually the result of congestion at the network layer.

Principles of Congestion Control

- A TCP sender can also be throttled due to congestion within the IP network; this form of sender control is referred to as **congestion control.**
- Specific TCP mechanisms are used to provide for a reliable data transfer service in the face of packet loss.
- Such loss typically results from the overflowing of router buffers as the network becomes congested.
- Packet retransmission thus treats a symptom of network congestion (the loss of a specific transport-layer segment) but does not treat the cause of network congestion.


λ_{in}: original data
λ'_{in}: original data, plus retransmitted data
λ_{out}
Host A
Host B
Host C
Host D

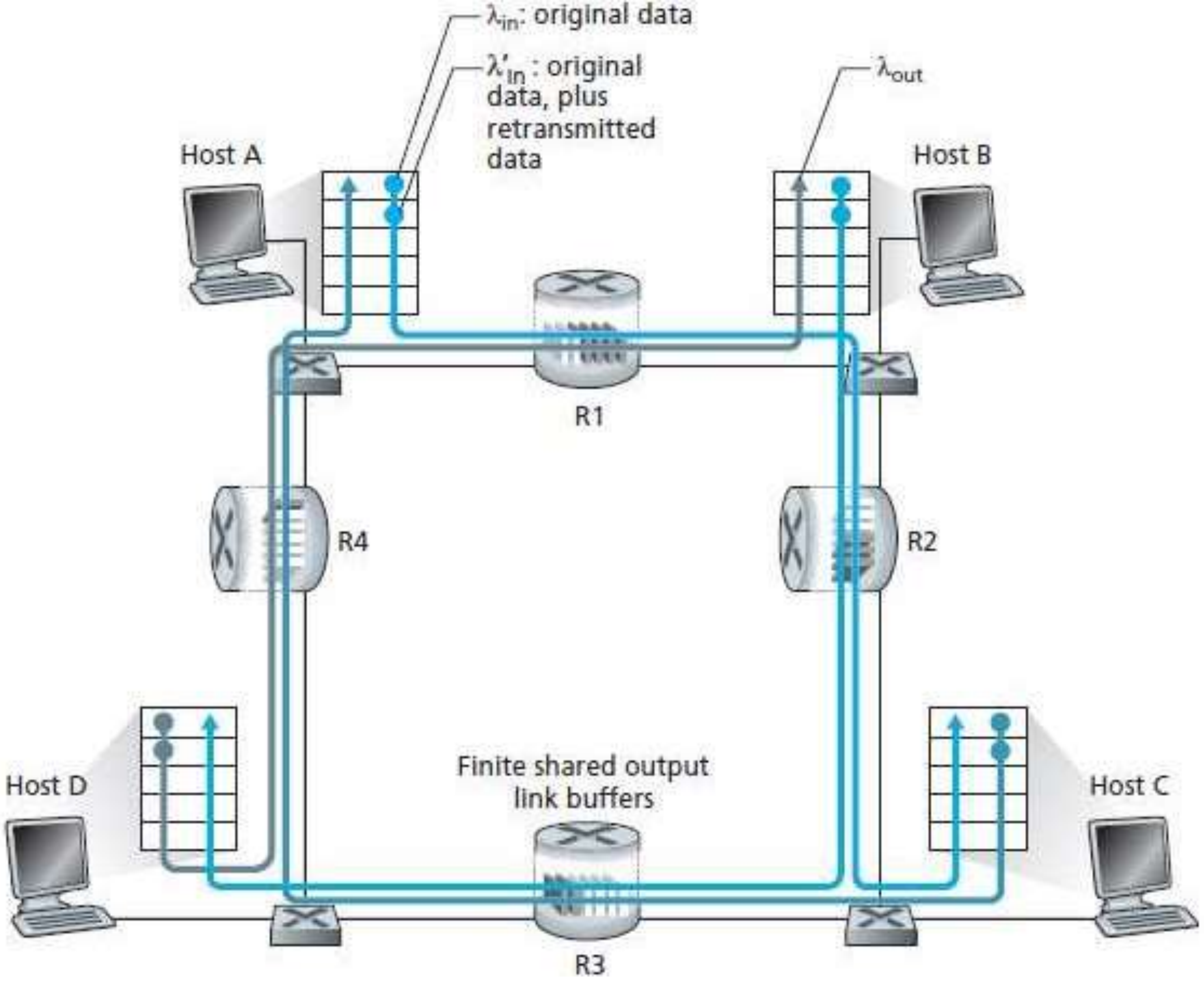
Finite shared output link buffers

Scenario : Two Senders and a Router with Finite Buffers

- The amount of router buffering is assumed to be finite.
- A consequence of this real-world assumption is that packets will be dropped when arriving to an already fullbuffer.
- Second, we assume that each connection is reliable. If a packet containing a transport-level segment is dropped at the router, the sender will eventually retransmit it.

Scenario : Four Senders, Routers with Finite Buffers, and Multihop Paths

- Each host uses a timeout/retransmission mechanism to implement a reliable data transfer service, that all hosts have the same value of Yin, and that all router links have capacity *R* bytes/sec.
- In the high-traffic scenario, whenever a packet is dropped at a second-hop router, the work done by the first-hop router in forwarding a packet to the second-hop router ends up being "wasted."


λ_{in}: original data
λ'_{in}: original data, plus retransmitted data
λ_{out}
Host A
Host B
R1
R4
R2
Host D
Host C
Finite shared output link buffers
R3

Approaches to Congestion Control

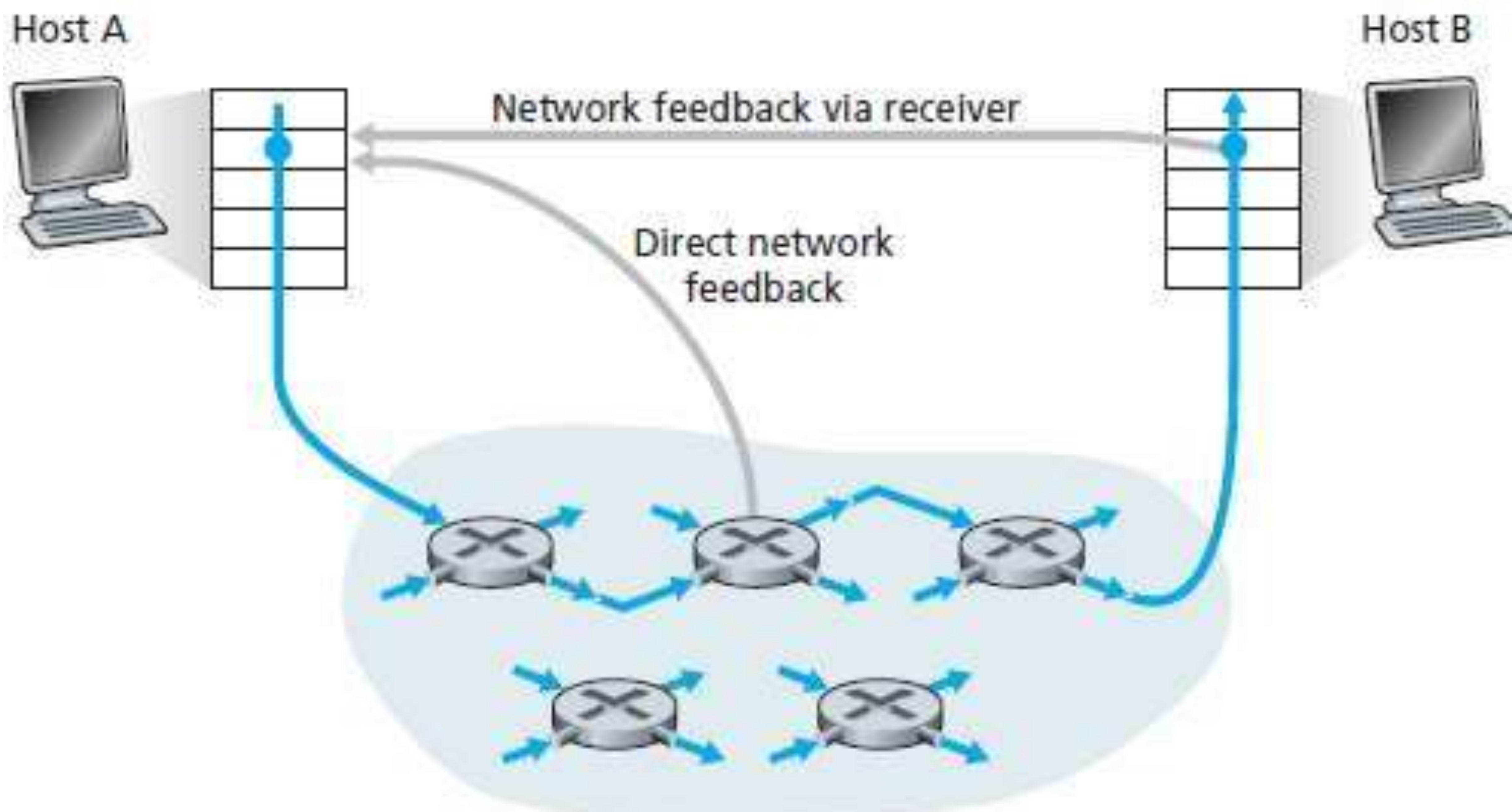
- *End-to-end congestion control*
- *Network-assisted congestion control.*

End-to-end congestion control.

- The network layer provides *no explicit support* to the transport layer for congestion control purposes.
- Even the presence of congestion in the network must be inferred by the end systems based only on observed network behaviour.
- TCP must necessarily take this end-to- end approach toward congestion control, since the IP layer provides no feedback to the end systems regarding network congestion.
- TCP segment loss (as indicated by a timeout or a triple duplicate acknowledgment) is taken as an indication of network congestion and TCP decreases its window size accordingly.

Network-assisted congestion control.

- The network-layer components (that is, routers) provide explicit feedback to the sender regarding the congestion state in the network.
- This feedback may be as simple as a single bit indicating congestion at a link.
- Direct feedback may be sent from a network router to the sender. This form of notification typically takes the form of a **choke packet** (essentially saying, “I’m congested!”).
- The second form of notification occurs when a router marks/updates a field in a packet flowing from sender to receiver to indicate congestion.
- Upon receipt of a marked packet, the receiver then notifies the sender of the congestion indication.


Host A
Host B
Network feedback via receiver
Direct network feedback

Network-Assisted Congestion-Control Example: ATM ABR Congestion Control

- A protocol that takes a network-assisted approach toward congestion control.
- **Asynchronous Transfer Mode Available Bit-Rate**
- ATM ABR congestion control is a rate-based approach.
- That is, the sender explicitly computes a maximum rate at which it can send and regulates itself accordingly.
- ABR provides three mechanisms for signaling congestion-related information from the switches to the receiver:

- *EFCI bit.* Each *data cell* contains an **explicit forward congestion indication (EFCI) bit**. A congested network switch can set the EFCI bit in a data cell to 1 to signal congestion to the destination host.
- **congestion indication (CI) bit** and a **no increase (NI) bit** that can be set by a congested network switch.
- **explicit rate (ER) field**. A congested switch may lower the value contained in the ER field in a passing packet.

CONGESTION CONTROL

TCP uses a **congestion window and a congestion policy** that avoid congestion and detect and alleviate congestion after it has occurred.

Congestion Window

- The sender window size is determined by the available buffer space in the receiver (rwnd).

- It is only the receiver that can dictate to the sender the size of the sender’s window.

Congestion Window

- If the network cannot deliver the data as fast as it is created by the sender, it must tell the sender to slow down.

- In other words, in addition to the receiver, the network is a second entity that determines the size of the sender’s window.

Congestion Window

- The sender has two pieces of information:
  - the receiver-advertised window size
  - and the congestion window size.
- The actual size of the window is the minimum of these two.

- **Actual window size = minimum (rwnd, cwnd)**

Congestion Policy

- TCP’s general policy for handling congestion is based on three phases:

- Slow start, congestion avoidance, and congestion detection.

- In the slow start phase, the sender starts with a slow rate of transmission, but increases the rate rapidly to reach a threshold.

- When the threshold is reached, the rate of increase is reduced.

- Finally if ever congestion is detected, the sender goes back to the slow start or congestion avoidance phase, based on how the congestion is detected.

Slow Start: Exponential Increase

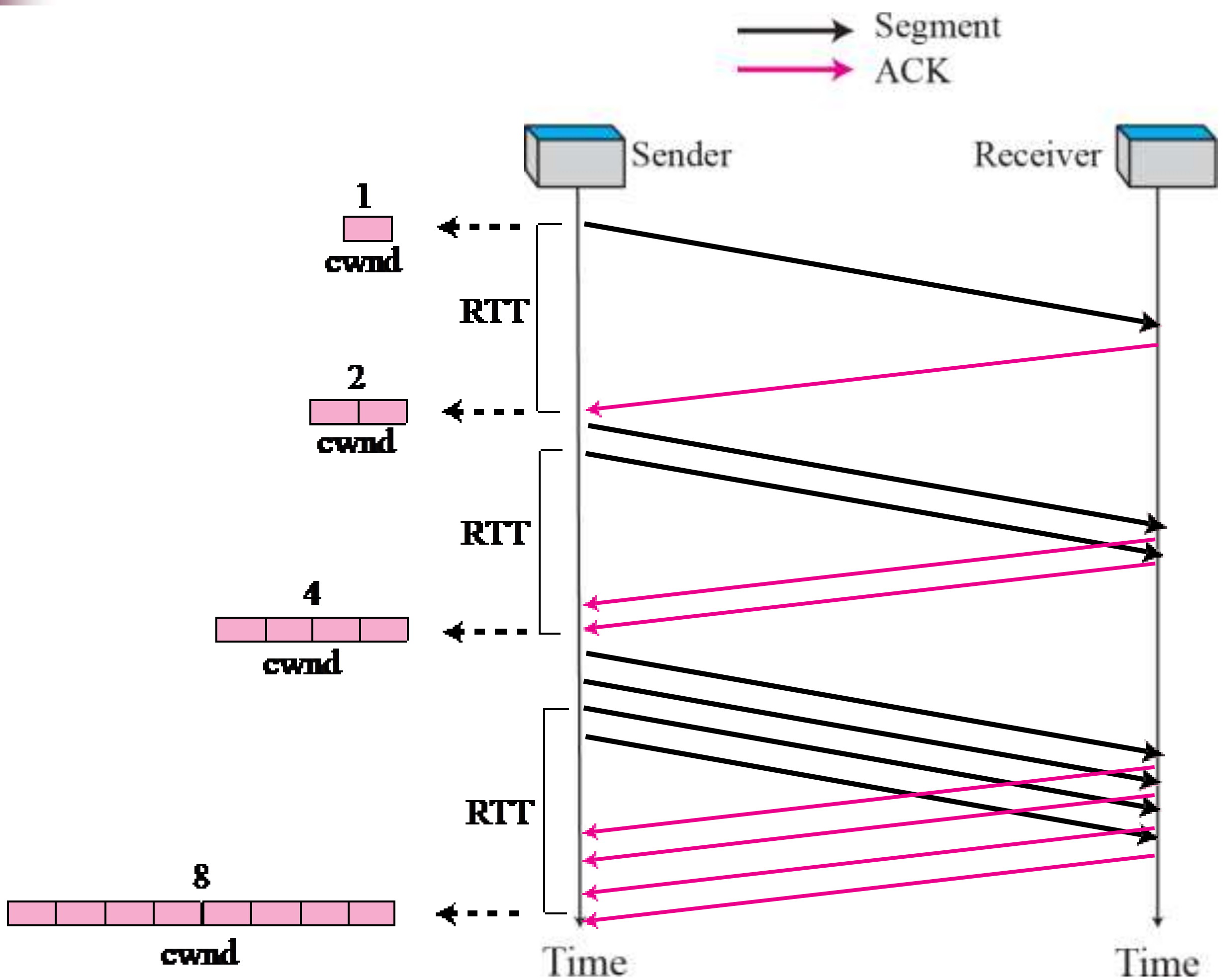
- The **slow start algorithm is based on the idea that the size of the congestion window** (cwnd) starts with one maximum segment size (MSS).

- The MSS is determined during connection establishment using an option of the same name.

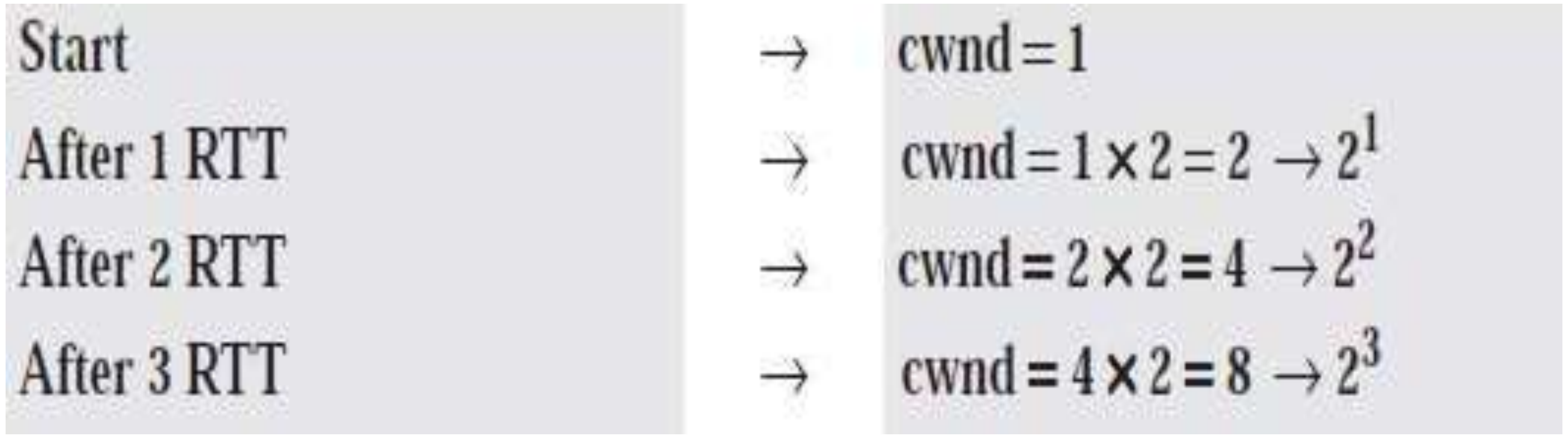
- The size of the window increases one MSS each time one acknowledgement arrives.

- As the name implies, the algorithm starts slowly, but grows exponentially.

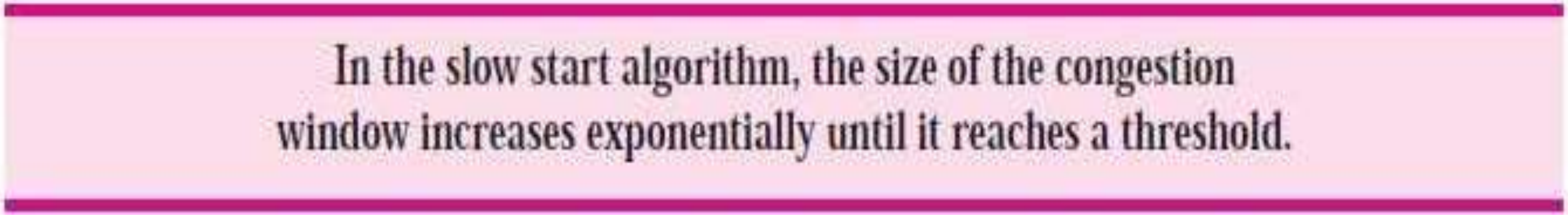
Figure 15.34 *Slow start, exponential increase*



Slow start, exponential increase



Start	$\rightarrow$	$cwnd = 1$
After 1 RTT	$\rightarrow$	$cwnd = 1 \times 2 = 2 \rightarrow 2^1$
After 2 RTT	$\rightarrow$	$cwnd = 2 \times 2 = 4 \rightarrow 2^2$
After 3 RTT	$\rightarrow$	$cwnd = 4 \times 2 = 8 \rightarrow 2^3$



In the slow start algorithm, the size of the congestion window increases exponentially until it reaches a threshold.

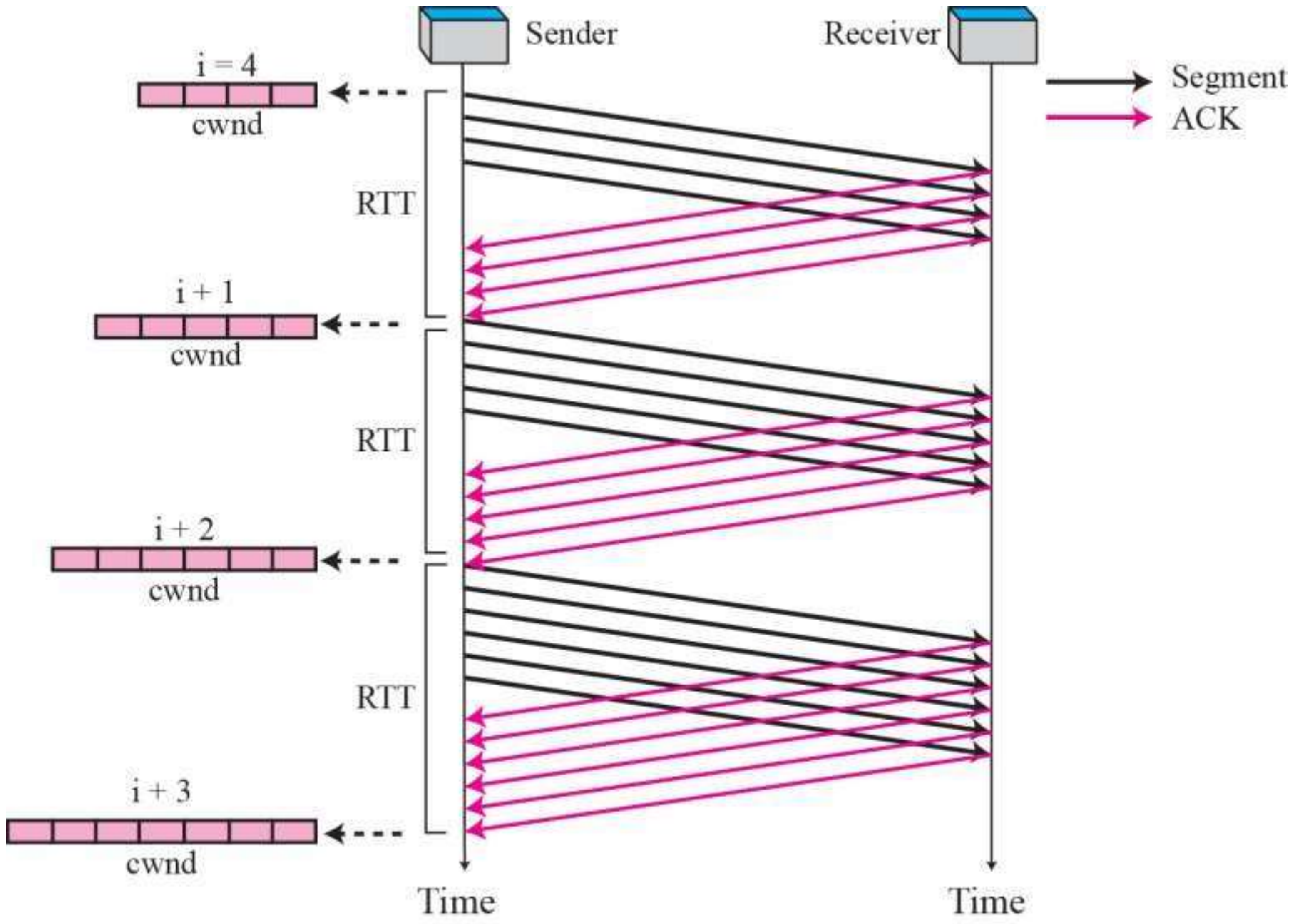
Congestion Avoidance: Additive Increase

- If we start with the slow start algorithm, the size of the congestion window increases exponentially.

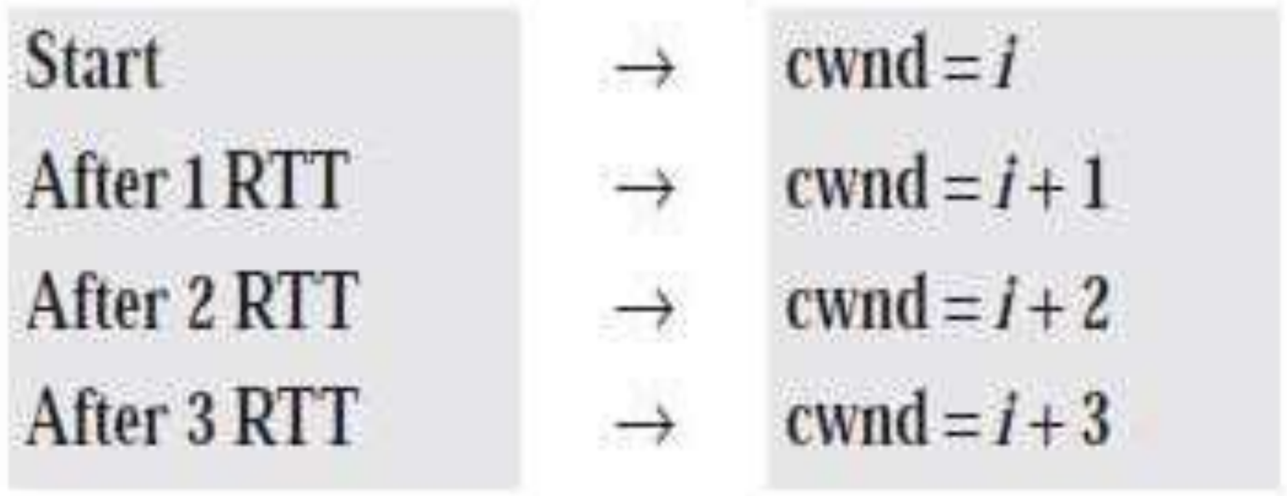
- To avoid congestion before it happens, one must slow down this exponential growth.

- TCP defines another algorithm called **congestion avoidance, which** increases the cwnd additively instead of exponentially.

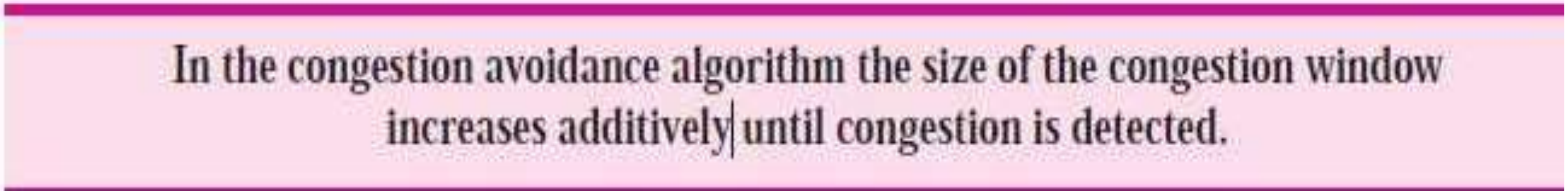
Figure 15.35 *Congestion avoidance, additive increase*



Congestion Avoidance: Additive Increase



Start	$\rightarrow$	cwnd $= i$
After 1 RTT	$\rightarrow$	cwnd $= i + 1$
After 2 RTT	$\rightarrow$	cwnd $= i + 2$
After 3 RTT	$\rightarrow$	cwnd $= i + 3$



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

Congestion Detection: Multiplicative Decrease

- If congestion occurs, the congestion window size must be decreased.
- The only way a sender can guess that congestion has occurred is the need to retransmit a segment.
- This is a major assumption made by TCP.
- Retransmission is needed to recover a missing packet which is assumed to have been dropped (i.e., lost) by a router that had so many incoming packets, that had to drop the missing segment, i.e., the router/network became overloaded or congested.

- However, retransmission can occur in one of two cases:
- when the RTO timer times out or
- when three duplicate ACKs are received.

- In both cases, the size of the threshold is dropped to half (**multiplicative decrease).**

1. **If a time-out occurs, there is a stronger possibility of congestion; a segment has** probably been dropped in the network and there is no news about the following sent segments. In this case TCP reacts strongly:

a. It sets the value of the threshold to half of the current window size.

b. It reduces cwnd back to one segment.

c. It starts the slow start phase again.

2. **If three duplicate ACKs are received, there is a weaker possibility of congestion; a** segment may have been dropped but some segments after that have arrived safely since three duplicate ACKs are received. This is called fast transmission and fast recovery.

In this case, TCP has a weaker reaction as shown below:

a. It sets the value of the threshold to half of the current window size.

b. It sets cwnd to the value of the threshold (some implementations add three segment sizes to the threshold).

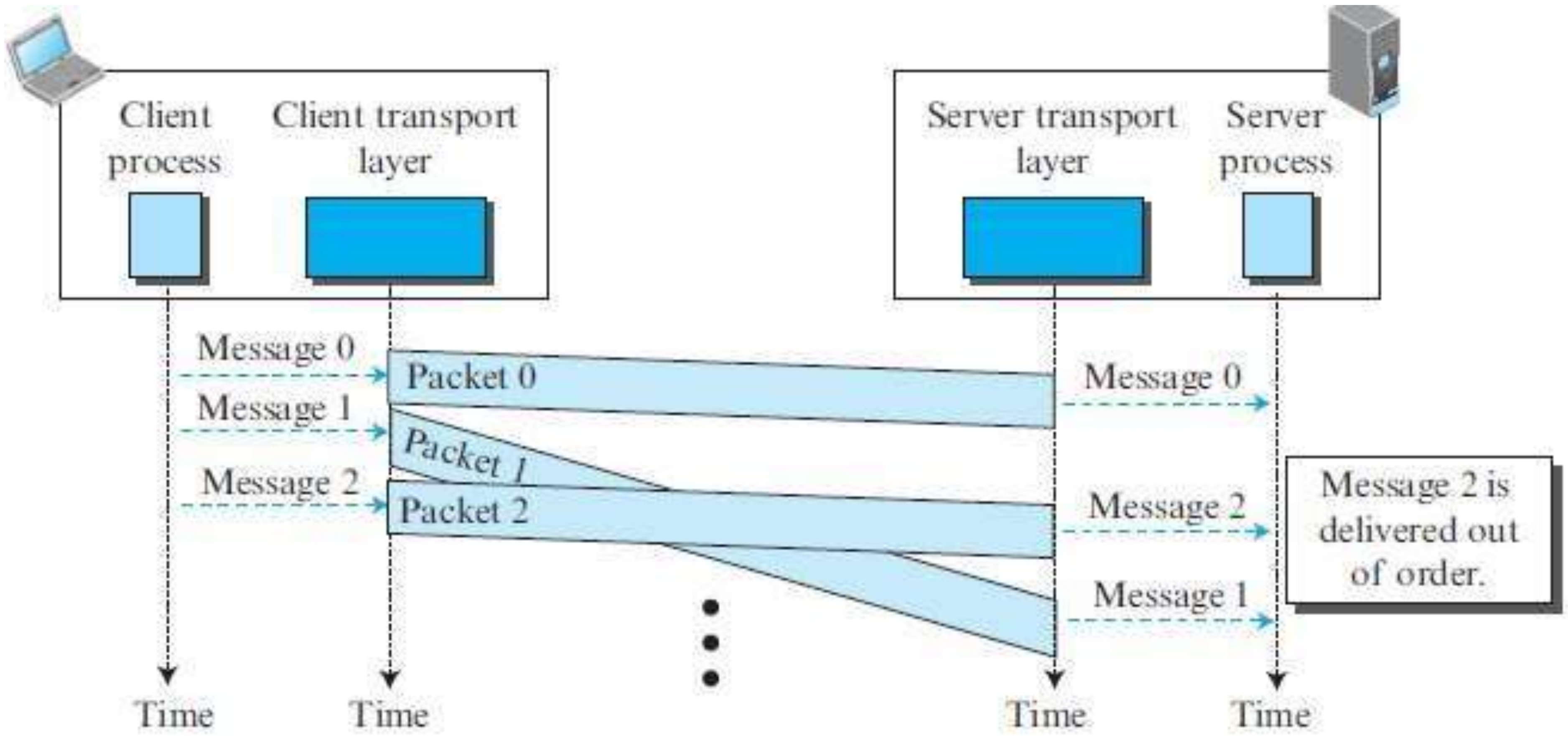
c. It starts the congestion avoidance phase.

Connectionless and Connection-Oriented Services

- Connectionless service at the transport layer means independency between packets;
- Connection-oriented means dependency.

Connectionless Service

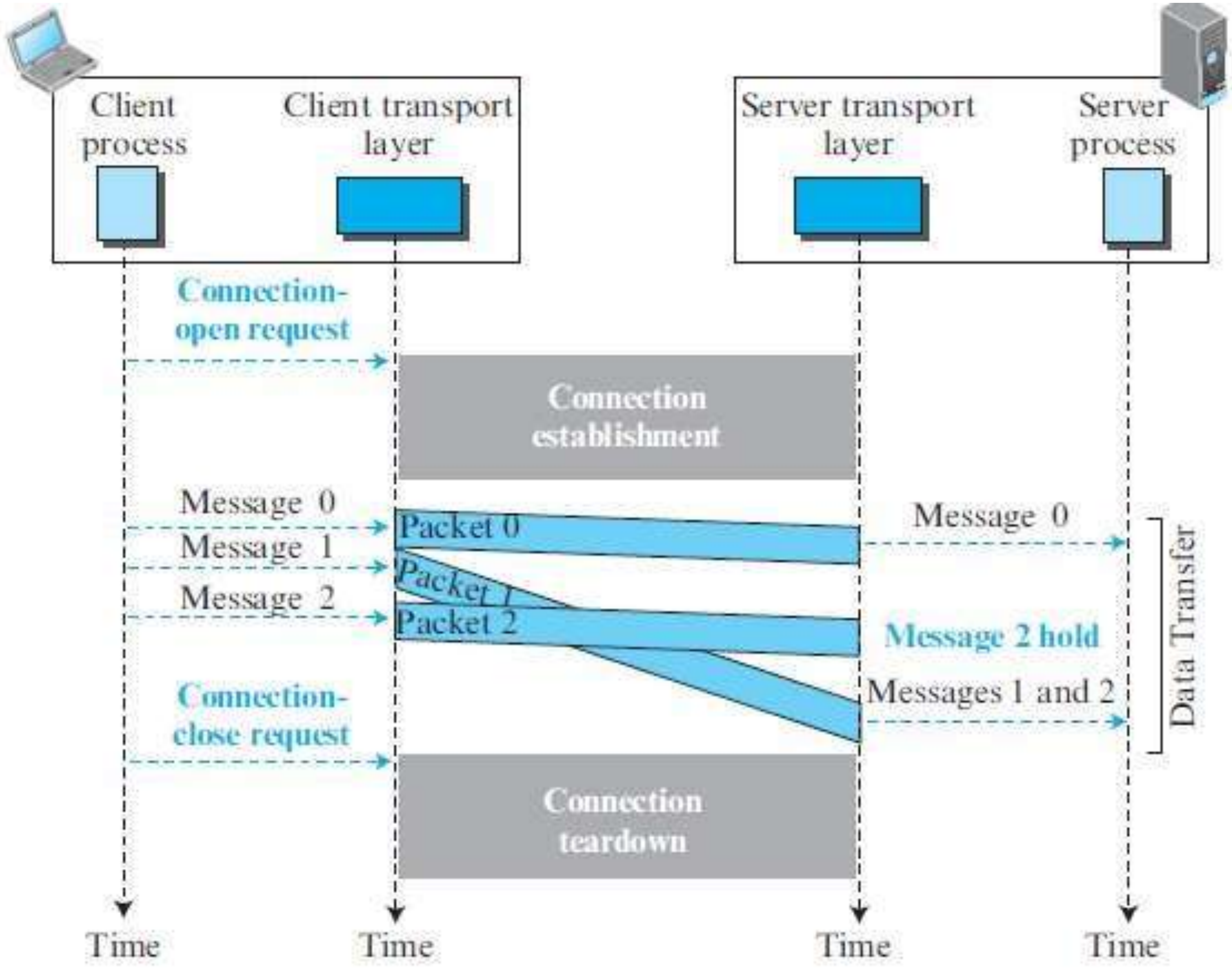
- In a connectionless service, the source process (application program) needs to divide its message into chunks of data of the size acceptable by the transport layer and deliver them to the transport layer one by one.
- When a chunk arrives from the application layer, the transport layer encapsulates it in a packet and sends it.
- Assume that a client process has three chunks of messages to send to a server process.
- The chunks are handed over to the connectionless transport protocol in order.
- However, since there is no dependency between the packets at the transport layer, the packets may arrive out of order at the destination and will be delivered out of order to the server process.


Client process
Client transport layer
Server transport layer
Server process
Message 0
Packet 0
Message 1
Packet 1
Message 2
Packet 2
Message 0
Message 2
Message 1
Message 2 is delivered out of order.
Time
Time
Time
Time

- If these three chunks of data belong to the same message, the server process may have received a strange message.
- Since there is no numbering on the packets, the receiving transport layer has no idea that one of the messages has been lost.
- no flow control, error control, or congestion control can be effectively implemented in a connectionless service.

Connection-Oriented Service

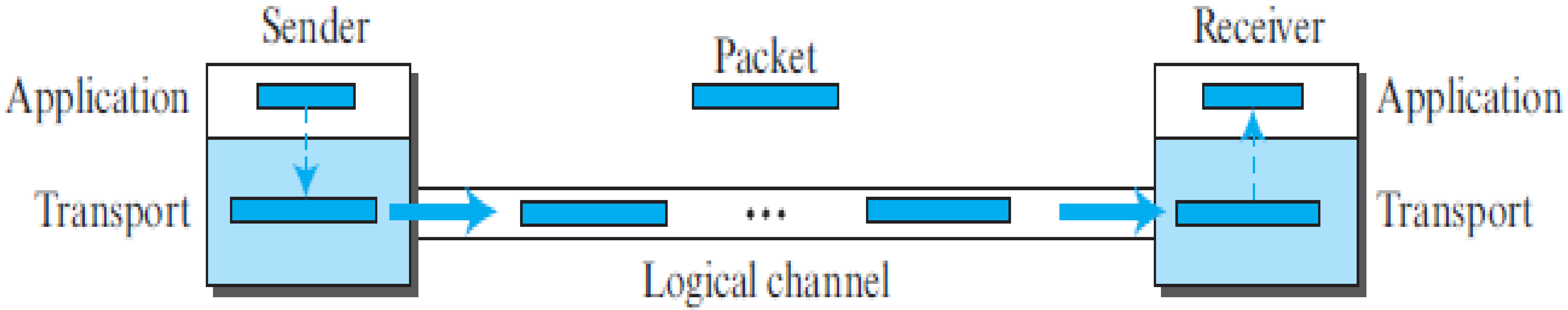
- The client and the server first need to establish a logical connection between themselves.
- The data exchange can only happen after the connection establishment.
- After data exchange, the connection needs to be turn down.
- We can implement flow control, error control, and congestion control in a connection oriented protocol.


Client process
Client transport layer
Server transport layer
Server process
Connection-open request
Connection establishment
Message 0
Message 1
Message 2
Packet 0
Packet 1
Packet 2
Message 0
Message 2 hold
Messages 1 and 2
Data Transfer
Connection-close request
Connection teardown
Time
Time
Time
Time

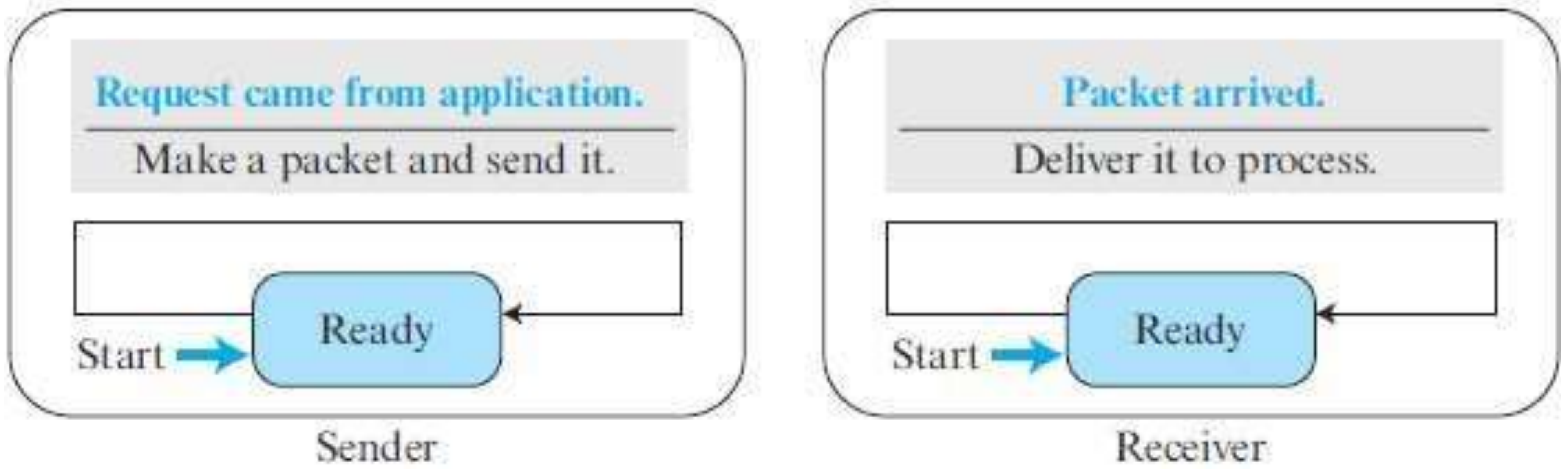
PROTOCOLS FOR RELIABLE DATA TRANSFER

- Simple Protocol
- Stop-and-Wait Protocol
- Go-Back-N Protocol (GBN)
- Selective-Repeat Protocol

Simple Protocol

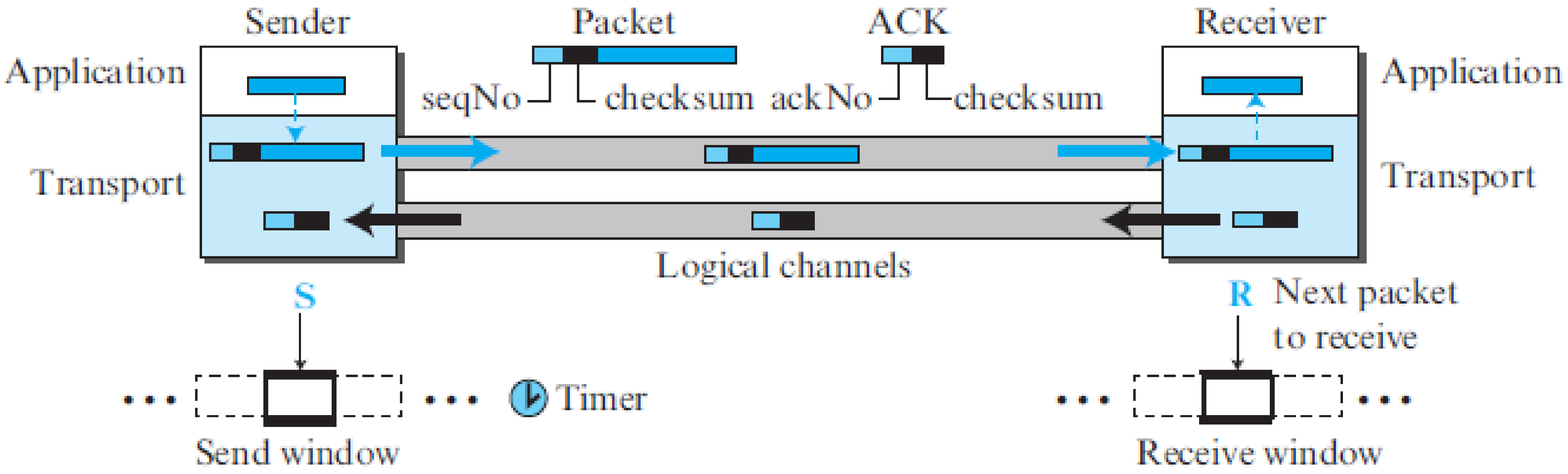


States of SimpleProtocol

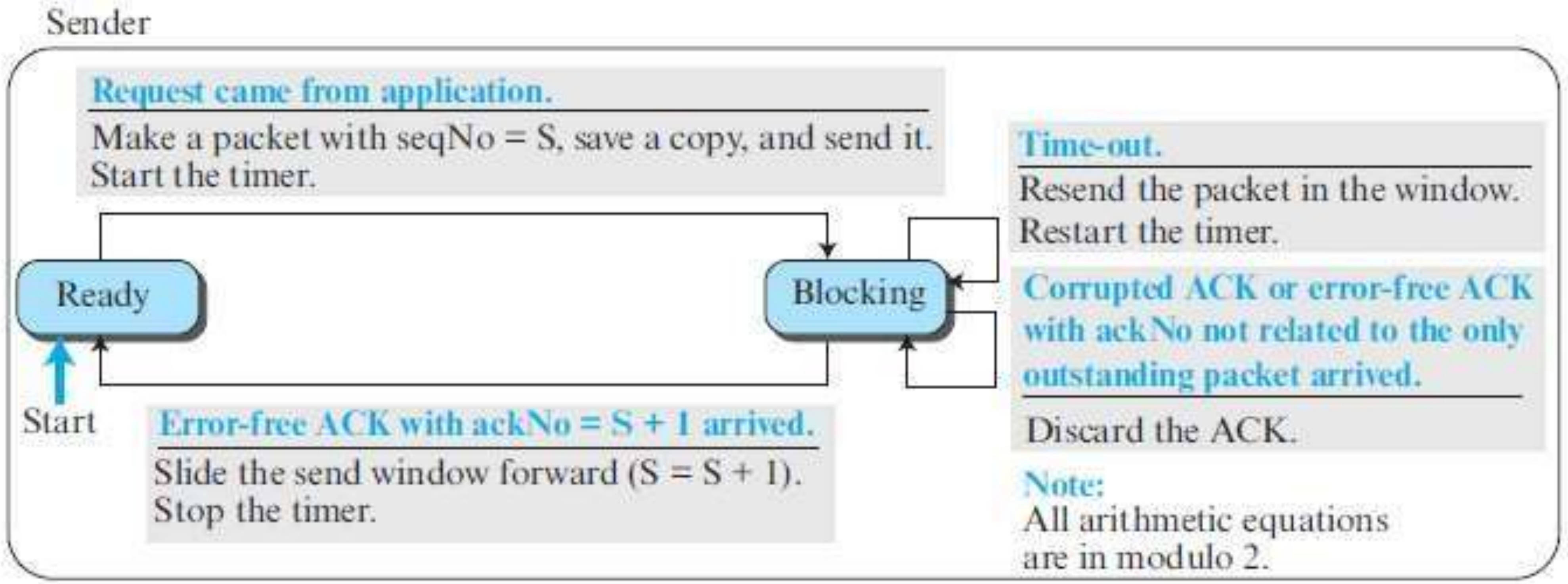


Stop-and-Wait Protocol

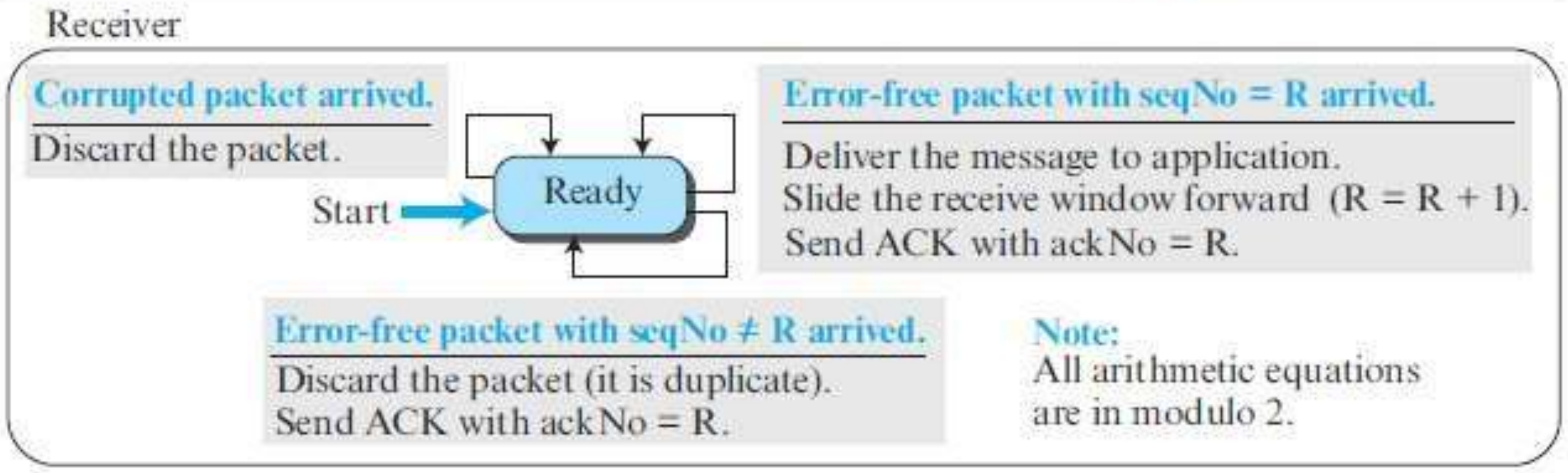
- It is a connection-oriented protocol, which uses both flow and error control.
- Both the sender and the receiver use a sliding window of size 1.
- The sender sends one packet at a time and waits for an acknowledgment before sending the next one.
- To detect corrupted packets, we need to add a checksum to each data packet.
- When a packet arrives at the receiver site, it is checked. If its checksum is incorrect, the packet is corrupted and silently discarded.
- Every time the sender sends a packet, it starts a timer. If the timer expires, the sender resends the previous packet.
- To prevent duplicate packets, the protocol uses sequence numbers and acknowledgment numbers.


Sender
Packet
ACK
Receiver
Application
seqNo
checksum
ackNo
checksum
Application
Transport
Transport
Logical channels
S
R
Next packet to receive
Timer
Send window
Receive window

- In the Stop-and-Wait protocol, the acknowledgment number always announces, in modulo-2 arithmetic, the sequence number of the next packet expected.
- The new packet is x + 1, not x + 2. If only x and x + 1 are needed let x = 0 and x + 1 = 1.
- This means that the sequence is 0, 1, 0, 1, 0, and so on. This is referred to as modulo 2 arithmetic.
- The Stop-and-Wait protocol is very inefficient if our channel has a large bandwidth and the round-trip delay is long .
- The product of these two is called the bandwidth delay product.
- The bandwidth-delay product is a measure of the number of bits a sender can transmit through the system while waiting for an acknowledgment from the receiver.
- There is no pipelining in the Stop-and-Wait protocol because a sender must wait for a packet to reach the destination and be acknowledged before the next packet can be sent.



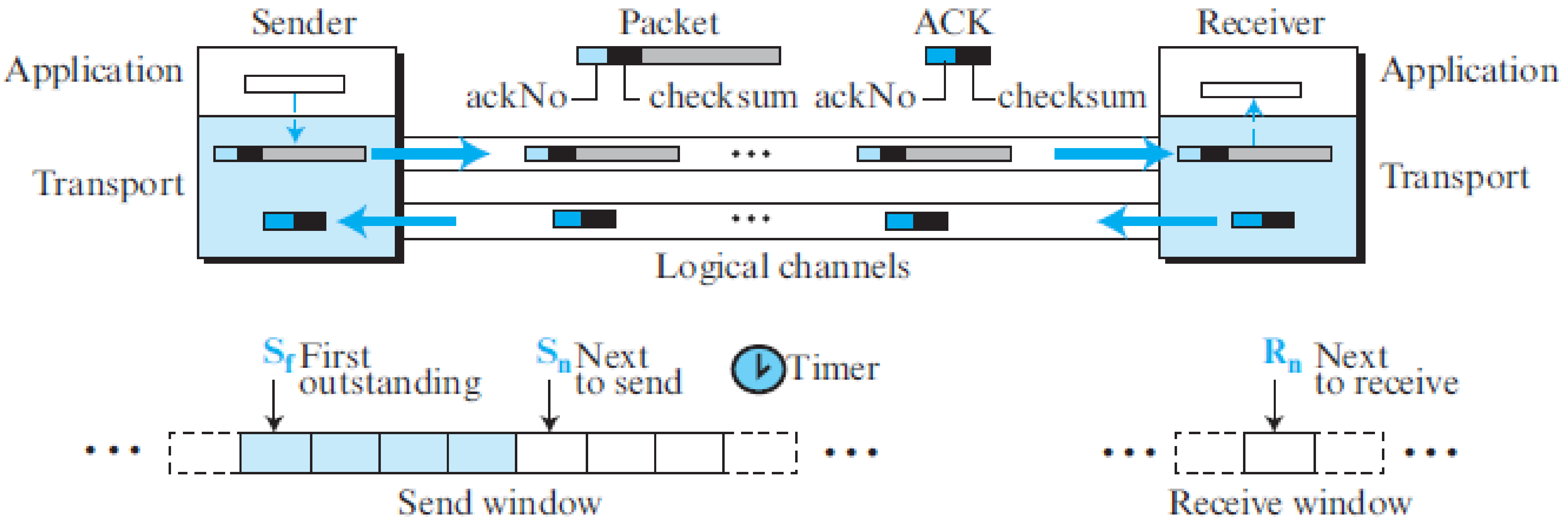
Sender
Request came from application.
Make a packet with seqNo = S, save a copy, and send it.
Start the timer.
Time-out.
Resend the packet in the window.
Restart the timer.
Ready
Blocking
Corrupted ACK or error-free ACK with ackNo not related to the only outstanding packet arrived.
Discard the ACK.
Start
Error-free ACK with ackNo = S + 1 arrived.
Slide the send window forward (S = S + 1).
Stop the timer.
Note:
All arithmetic equations are in modulo 2.



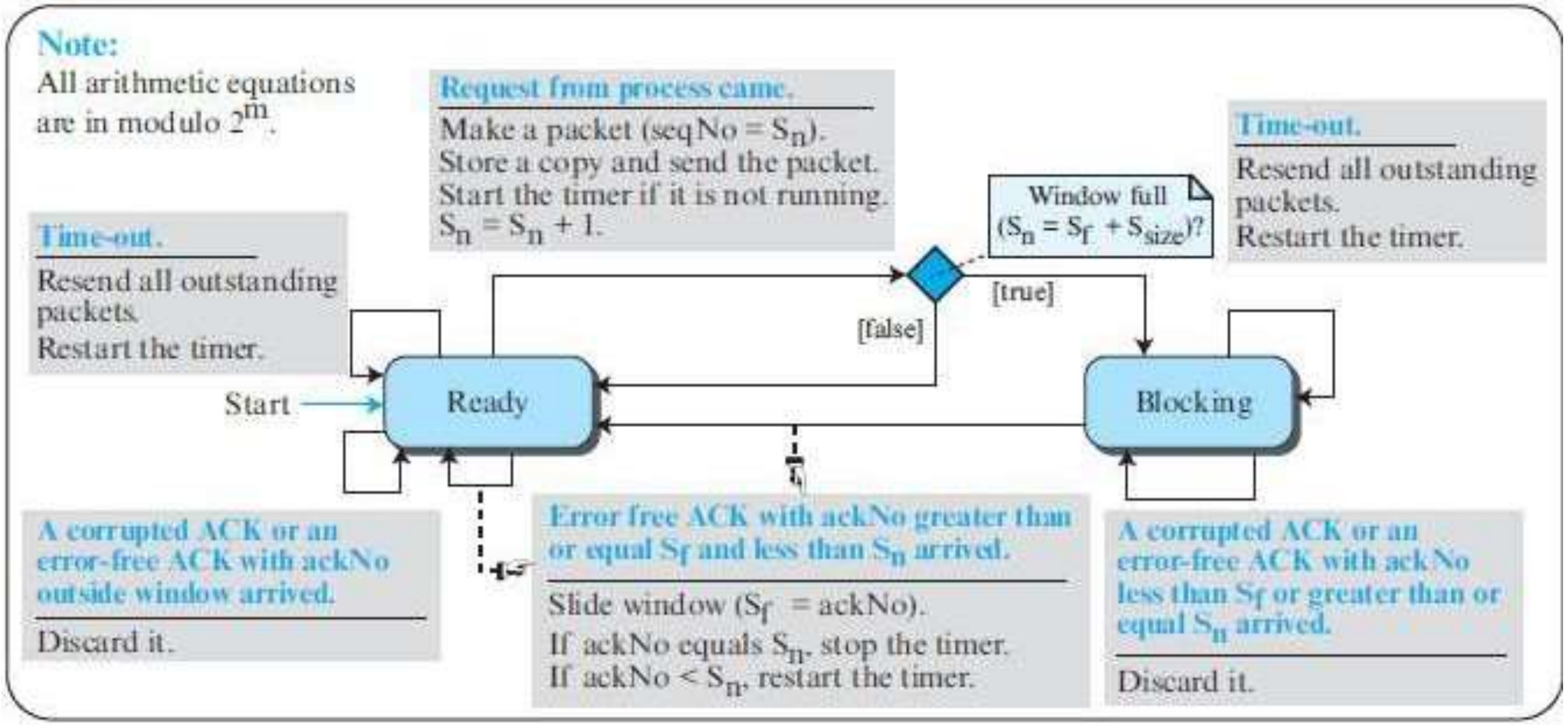
Receiver
Corrupted packet arrived.
Discard the packet.
Ready
Start
Error-free packet with seqNo = R arrived.
Deliver the message to application.
Slide the receive window forward (R = R + 1).
Send ACK with ackNo = R.
Error-free packet with seqNo ≠ R arrived.
Discard the packet (it is duplicate).
Send ACK with ackNo = R.
Note:
All arithmetic equations are in modulo 2.

Go-Back-N Protocol(GBN)

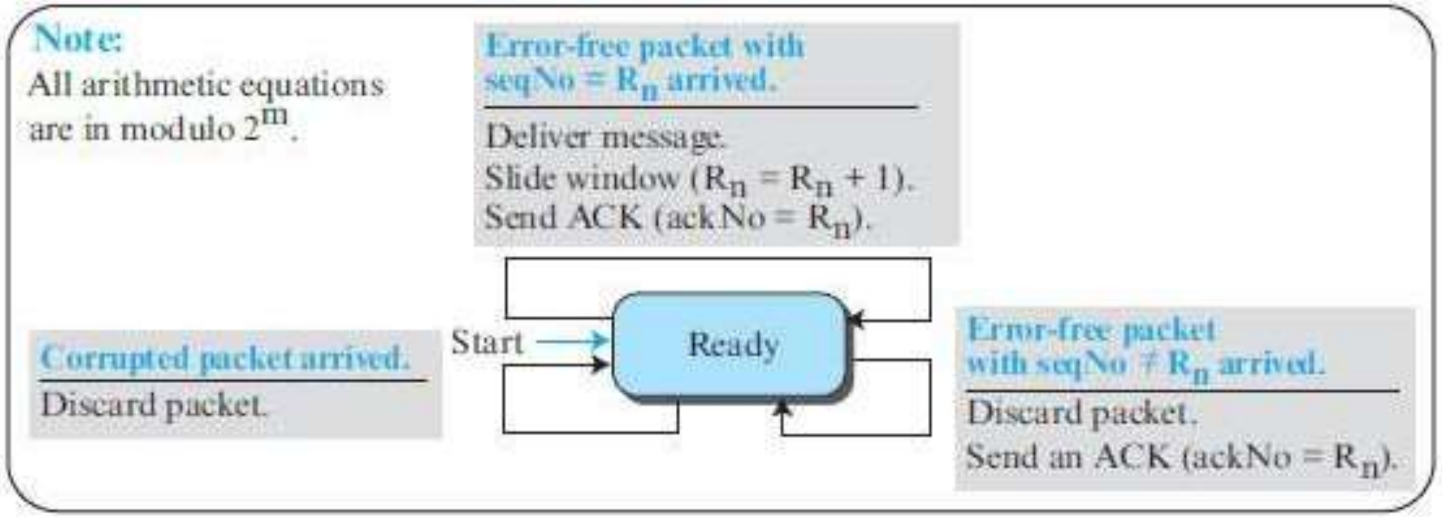
- The key to Go-back-N is that we can send several packets before receiving acknowledgments, but the receiver can only buffer one packet.
- We keep a copy of the sent packets until the acknowledgments arrive.
- several data packets and acknowledgments can be in the channel at the same time.
- if the acknowledgment number (ackNo) is 7, it means all packets with sequence number up to 6 have arrived, safe and sound, and the receiver is expecting the packet with sequence number 7.


Sender
Packet
ACK
Receiver
Application
ackNo
checksum
ackNo
checksum
Application
Transport
Transport
Logical channels
S_f First outstanding
S_n Next to send
Timer
R_n Next to receive
Send window
Receive window

Sender


Note:
All arithmetic equations are in modulo 2^m.
Request from process came.
Make a packet (seqNo = S_n).
Store a copy and send the packet.
Start the timer if it is not running.
$S_n = S_n + 1$.
Window full ($S_n = S_f + S_{size}$)?
[true]
[false]
Time-out.
Resend all outstanding packets.
Restart the timer.
Time-out.
Resend all outstanding packets.
Restart the timer.
Start
Ready
Blocking
A corrupted ACK or an error-free ACK with ackNo outside window arrived.
Discard it.
Error free ACK with ackNo greater than or equal S_f and less than S_n arrived.
Slide window (S_f = ackNo).
If ackNo equals S_n, stop the timer.
If ackNo < S_n, restart the timer.
A corrupted ACK or an error-free ACK with ackNo less than S_f or greater than or equal S_n arrived.
Discard it.

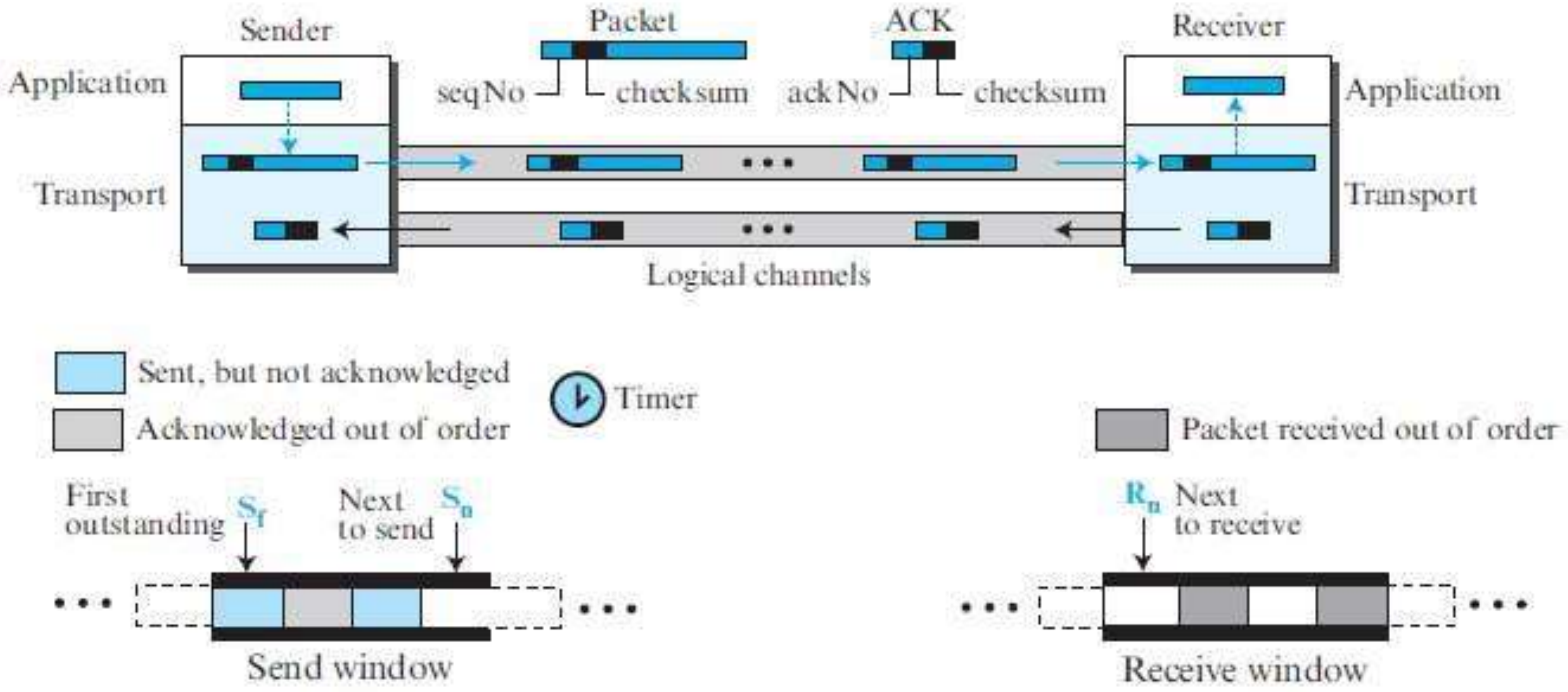
Receiver

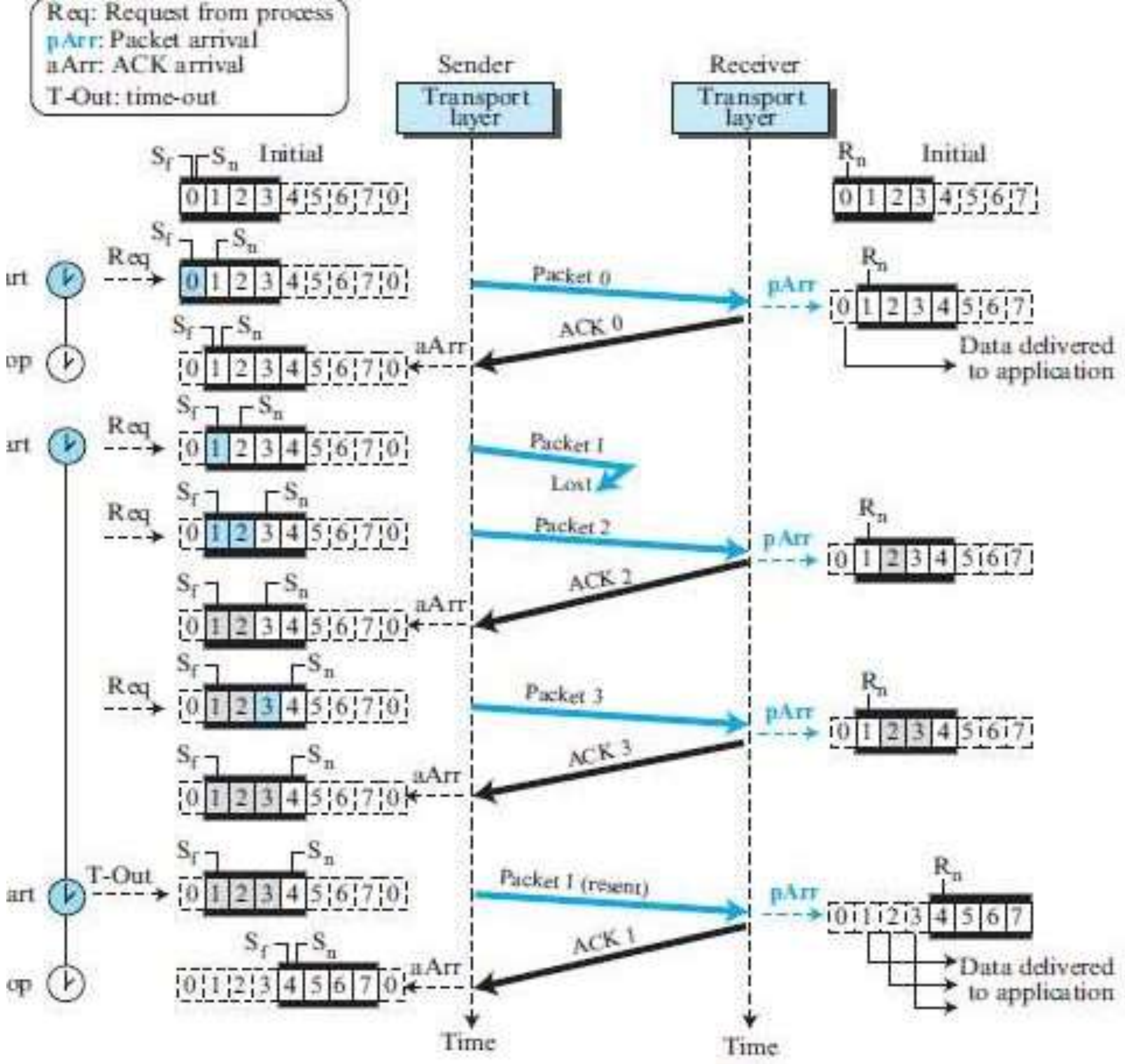

Note:
All arithmetic equations are in modulo 2^m.
Error-free packet with seqNo = R_n arrived.
Deliver message.
Slide window ($R_n = R_n + 1$).
Send ACK (ackNo = R_n).
Start
Ready
Corrupted packet arrived.
Discard packet.
Error-free packet with seqNo ≠ R_n arrived.
Discard packet.
Send an ACK (ackNo = R_n).

- This protocol is inefficient if the underlying network protocol loses a lot of packets.
- Each time a single packet is lost or corrupted, the sender resends all outstanding packets, even though some of these packets may have been received safe and sound but out of order.
- If the network layer is losing many packets because of congestion in the network, the resending of all of these outstanding packets makes the congestion worse, and eventually more packets are lost.
- This has an avalanche effect that may result in the total collapse of the network.

Selective-Repeat Protocol

- It resends only selective packets, those that are actually lost.
- The Selective-Repeat protocol uses two windows: a send window and a receive window.
- The receive window is the same size as the send window.
- ackNo defines the sequence number of one single packet that is received safe and sound.

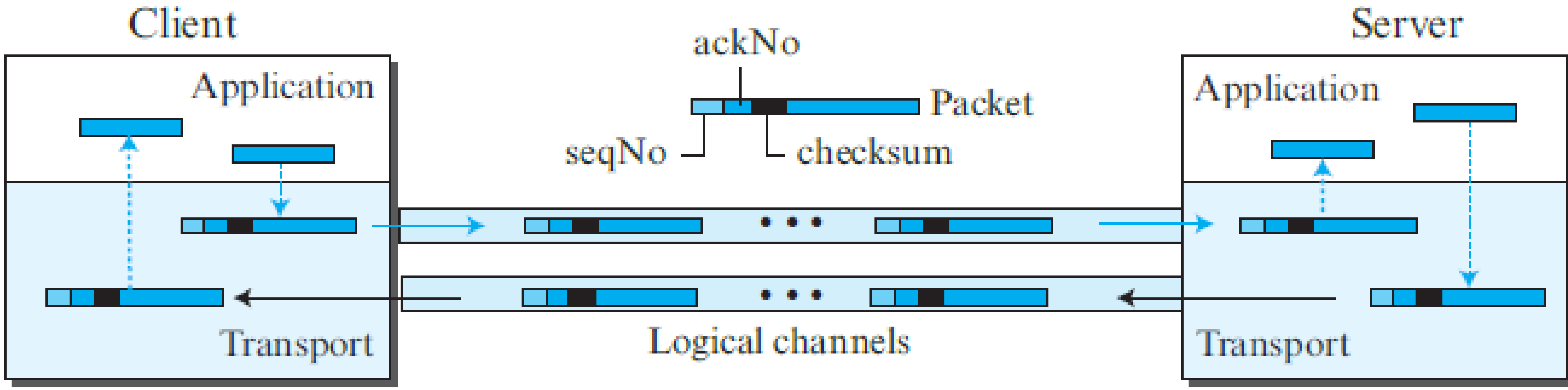

Sender
Packet
ACK
Receiver
Application
seqNo
checksum
ackNo
checksum
Application
Transport
Transport
Logical channels
Sent, but not acknowledged
Acknowledged out of order
Timer
Packet received out of order
First outstanding
S_f
Next to send
S_n
R_n
Next to receive
Send window
Receive window



Req: Request from process
pArr: Packet arrival
aArr: ACK arrival
T-Out: time-out
Sender
Transport layer
Receiver
Transport layer
Initial
Req
pArr
aArr
T-Out
Packet 0
ACK 0
Packet 1
Lost
Packet 2
ACK 2
Packet 3
ACK 3
Packet 1 (resent)
ACK 1
Data delivered to application
Data delivered to application
Time
Time

Bidirectional Protocols: Piggybacking

- In real life, data packets are normally flowing in both directions: from client to server and from server to client.
- This means that acknowledgments also need to flow in both directions.
- A technique called piggybacking is used to improve the efficiency of the bidirectional protocols.
- When a packet is carrying data from A to B, it can also carry acknowledgment feedback about arrived packets from B.


Client
Application
Transport
ackNo
Packet
seqNo
checksum
Logical channels
Server
Application
Transport

Internet Transport-Layer Protocols

- Although the Internet uses several transport-layer protocols, we discuss only two : UDP and TCP
- These protocols are located between the application layer and the network layer and serve as the intermediary between the application programs and the network operations.
- 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 is a reliable connection-oriented protocol that can be used in any application where reliability is important.
- Both are responsible to create a process-to-process communication

Figure 3.38 *Position of transport-layer protocols in the TCP/IP protocol suite*

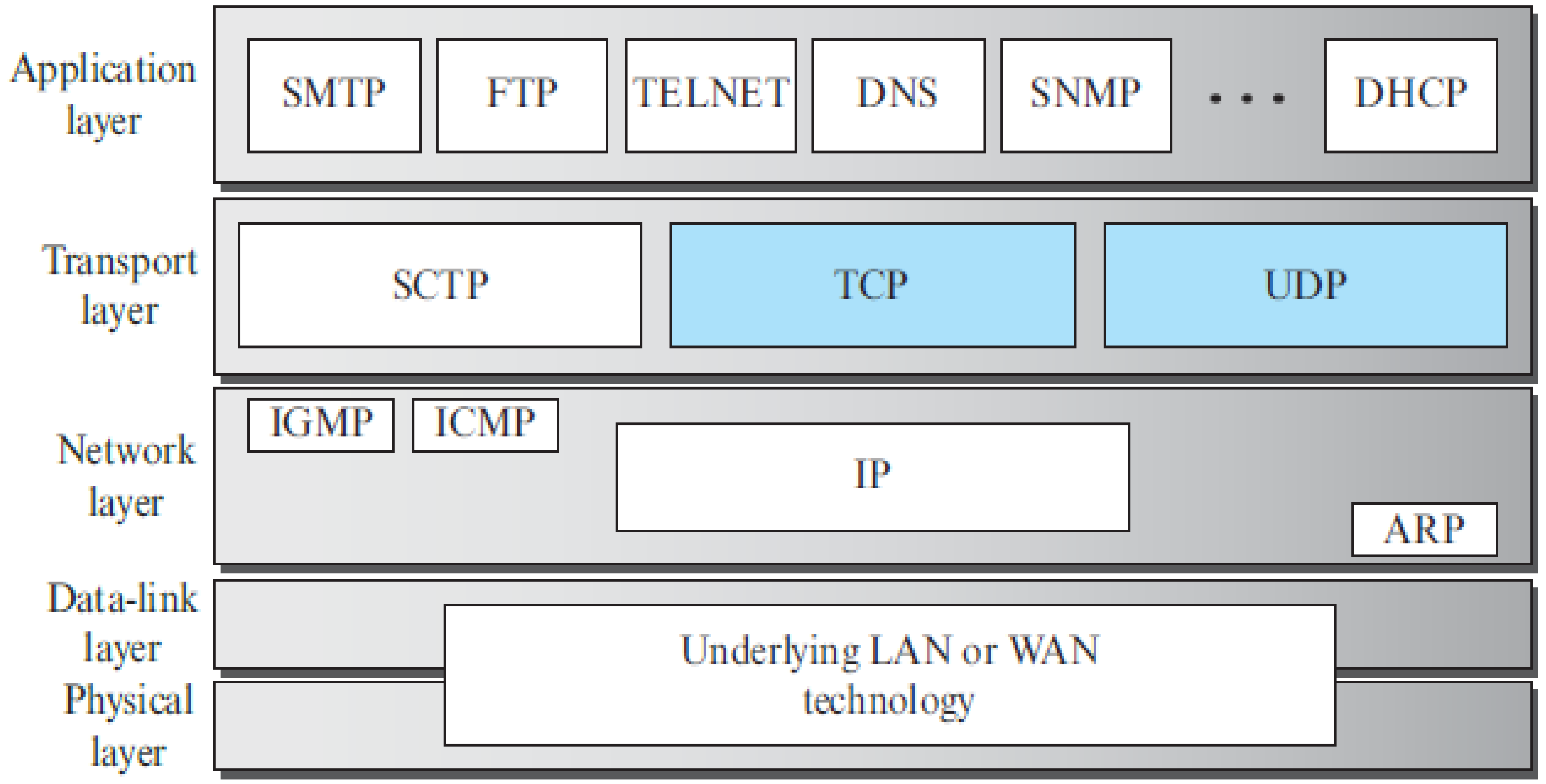


Table 3.1 *Some well-known ports used with UDP and TCP*

Port	*Protocol*	*UDP*	*TCP*	*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

CONNECTIONLESS TRANSPORT: UDP

CONNECTIONLESS TRANSPORT: UDP

- UDP does just about as little as a transport protocol can do.

- Aside from the multiplexing/demultiplexing function and some light error checking, it adds nothing to IP.

- UDP takes messages from the application process, attaches source and destination port number fields for the multiplexing/demultiplexing service, adds two other small fields, and passes the resulting segment to the network layer.

CONNECTIONLESS TRANSPORT: UDP

- **The network layer encapsulates the transport-layer segment into an IP datagram and then makes a best-effort attempt to deliver the segment to the receiving host.**

- If the segment arrives at the receiving host, UDP uses the destination port number to deliver the segment's data to the correct application process.

- Note that with UDP there is no handshaking between sending and receiving transport-layer entities before sending a segment. For this reason, **UDP is said to be *connectionless.***

CONNECTIONLESS TRANSPORT: UDP

Many applications are better suited for UDP for the following reasons:

1. ***application-level control over what data is sent, and when.***

2. ***No connection establishment***-UDP does not introduce any delay to establish a connection. This is probably the principal reason why DNS runs over UDP rather than TCP.

3. ***No connection state***.

4. ***Small packet header overhead****. The TCP segment has 20 bytes of header overhead in every segment, whereas UDP has only 8 bytes of overhead.*

CONNECTIONLESS TRANSPORT: UDP

Explanation

application-level control over what data is sent, and when

- an application process passes data to UDP, UDP will package the data inside a UDP segment and immediately pass the segment to the network layer.

- TCP, on the other hand, has a congestion-control mechanism that throttles the transport-layer TCP sender.

- TCP will also continue to resend a segment until the receipt of the segment has been acknowledged by the destination, regardless of how long reliable delivery takes.

- Since real-time applications often require a minimum sending rate, do not want to overly delay segment transmission, and can tolerate some data loss, TCP’s service model is not particularly well matched to these applications’ needs.

CONNECTIONLESS TRANSPORT: UDP

Explanation

No connection state

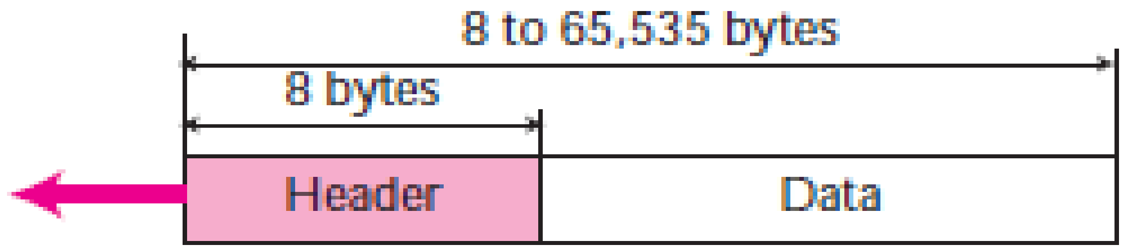
- *TCP maintains connection state in the end systems. This* connection state includes receive and send buffers, congestion-control parameters, and sequence and acknowledgment number parameters.

- UDP, on the other hand, does not maintain connection state and does not track any of these parameters. For this reason, a server devoted to a particular application can typically support many more active clients when the application runs over UDP rather than TCP.

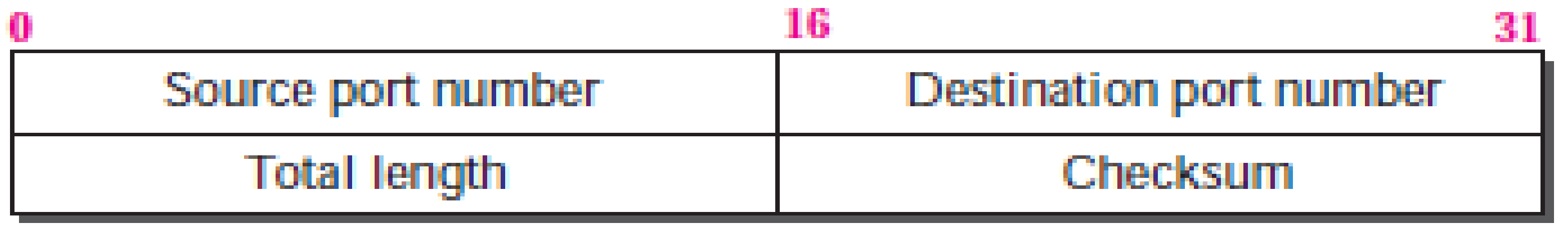
POPULAR INTERNET APPLICATIONS AND THEIR UNDERLYING TRANSPORT PROTOCOLS

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Network management	SNMP	Typically UDP
Routing protocol	RIP	Typically UDP
Name translation	DNS	Typically UDP

UDP SEGMENT STRUCTURE



a. UDP user datagram



b. Header format

UDP SEGMENT STRUCTURE

- The UDP header has only four fields, each consisting of two bytes.
- The port numbers allow the destination host to pass the application data to the correct process running on the destination end system (that is, to perform the demultiplexing function).

- The length field specifies the number of bytes in the UDP segment (header plus data).
- An explicit length value is needed since the size of the data field may differ from one UDP segment to the next.

- The checksum is used by the receiving host to check whether errors have been introduced into the segment.

UDP CHECKSUM

- The UDP checksum provides for error detection.

- That is, the checksum is used to determine whether bits within the UDP segment have been altered (for example, by noise in the links or while stored in a router) as it moved from source to destination.

- UDP at the sender side performs the 1s complement of the sum of all the 16-bit words in the segment, with any overflow encountered during the sum being wrapped around.

- This result is put in the checksum field of the UDP segment.

UDP CHECKSUM

0110011001100000

0101010101010101

1000111100001100

The sum of first two of these 16-bit words is

```
0110011001100000
0101010101010101
----------------
1011101110110101
```

Adding the third word to the above sum gives

```
   1011101110110101
   1000111100001100
-------------------
1,0100101011000001
```

```
0100101011000001
               1
----------------------
0100101011000010
```

UDP CHECKSUM

- Note that this last addition had overflow, which was wrapped around.

- The 1s complement is obtained by converting all the 0s to 1s and converting all the 1s to 0s.

- Thus the **1s complement of the sum 0100101011000010 is 1011010100111101, which becomes the checksum.**

- At the receiver, all four 16-bit words are added, including the checksum. If no errors are introduced into the packet, then clearly the sum at the receiver will be 1111111111111111.

- If one of the bits is a 0, then we know that errors have been introduced into the packet.

UDP CHECKSUM

0110011001100000

0101010101010101

1000111100001100

1011010100111101

1111111111111111

Message is accepted.

TRANSMISSION CONTROL PROTOCOL (TCP)

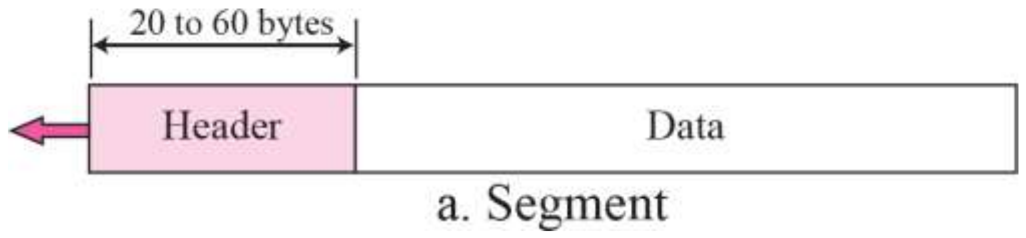
- Transmission Control Protocol (TCP) is a connection-oriented, reliable protocol.
- TCP explicitly defines connection establishment, data transfer, and connection teardown phases to provide a connection-oriented service.
- TCP uses a combination of GBN and SR protocols to provide reliablity.
- To achieve this goal, TCP uses checksum (for error detection), retransmission of lost or corrupted packets, cumulative and selective acknowledgments, and timers.
- TCP is the most common transport-layer protocol in the Internet.

TCP Services

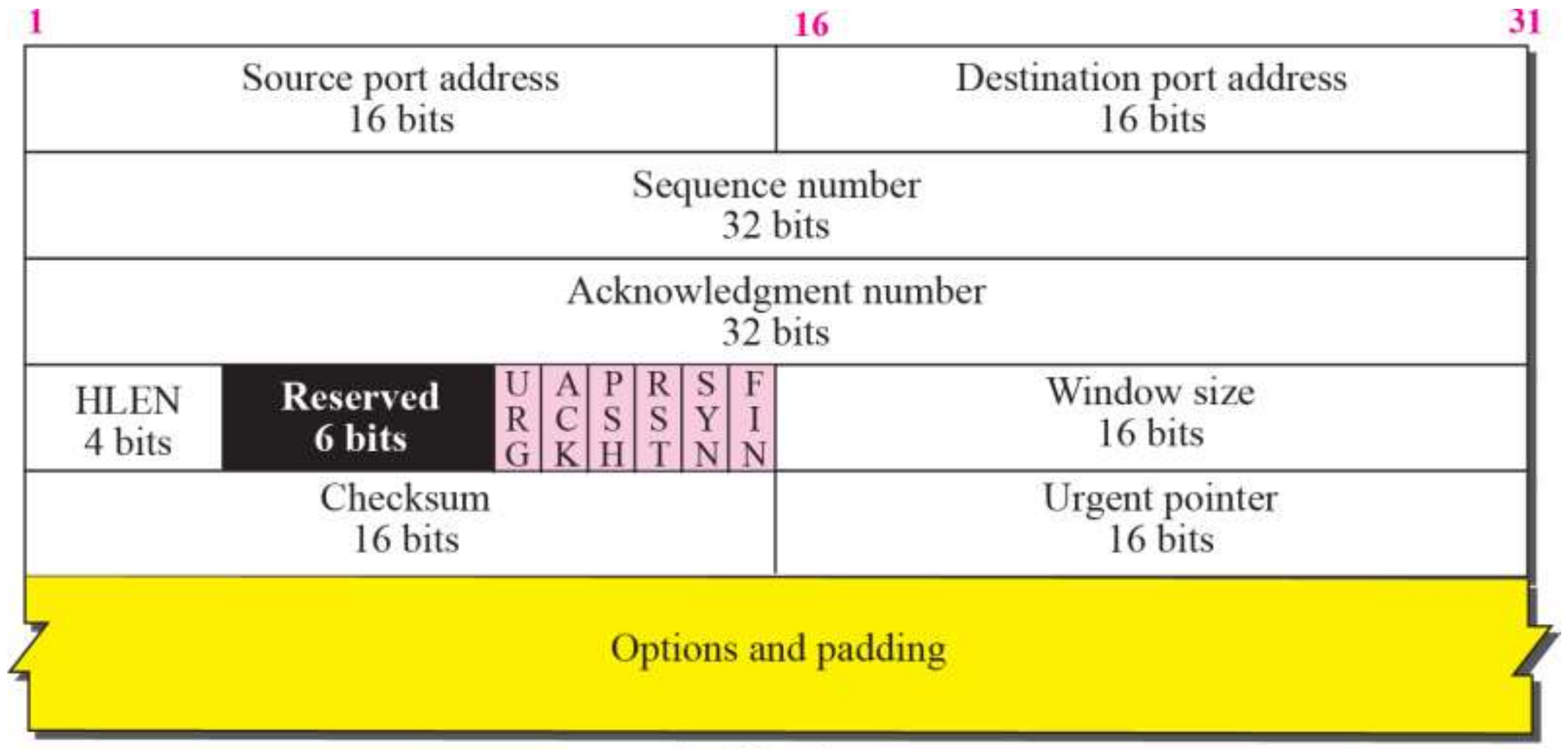
Services

- ✓ **Process-to-Process Communication**
- ✓ **Stream Delivery Service**
- ✓ **Full-Duplex Communication**
- ✓ **Multiplexing and Demultiplexing**
- ✓ **Connection-Oriented Service**
- ✓ **Reliable Service**

TCP segment format



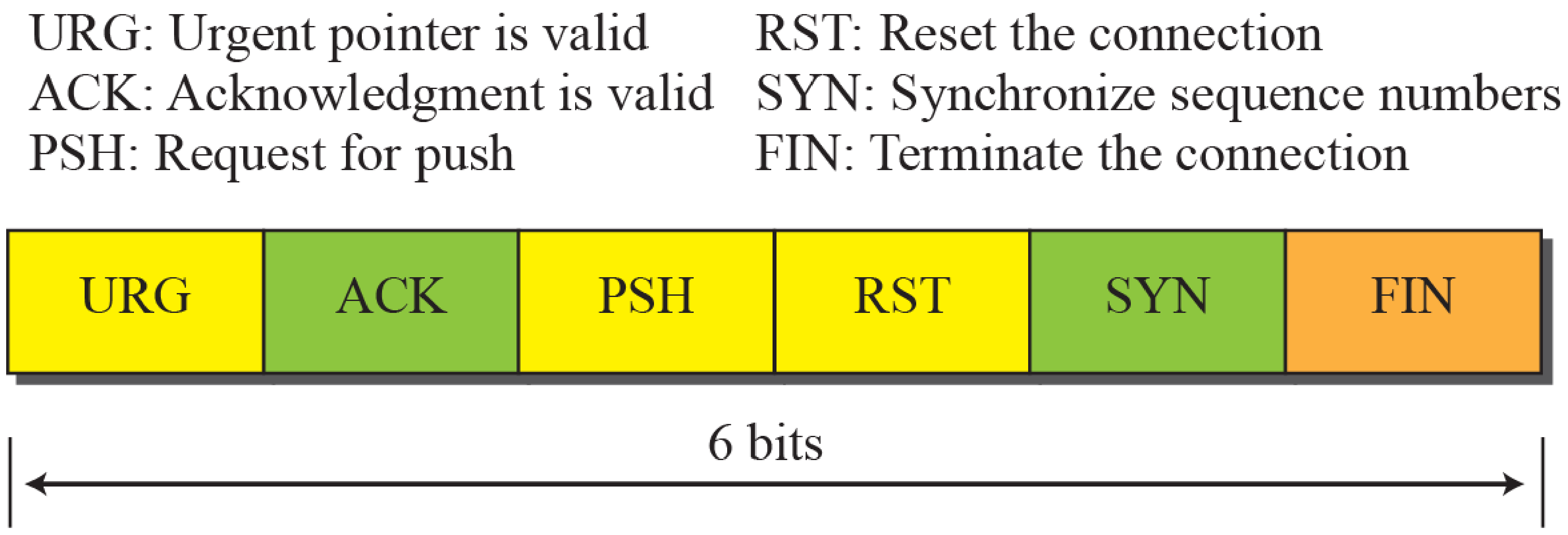
a. Segment



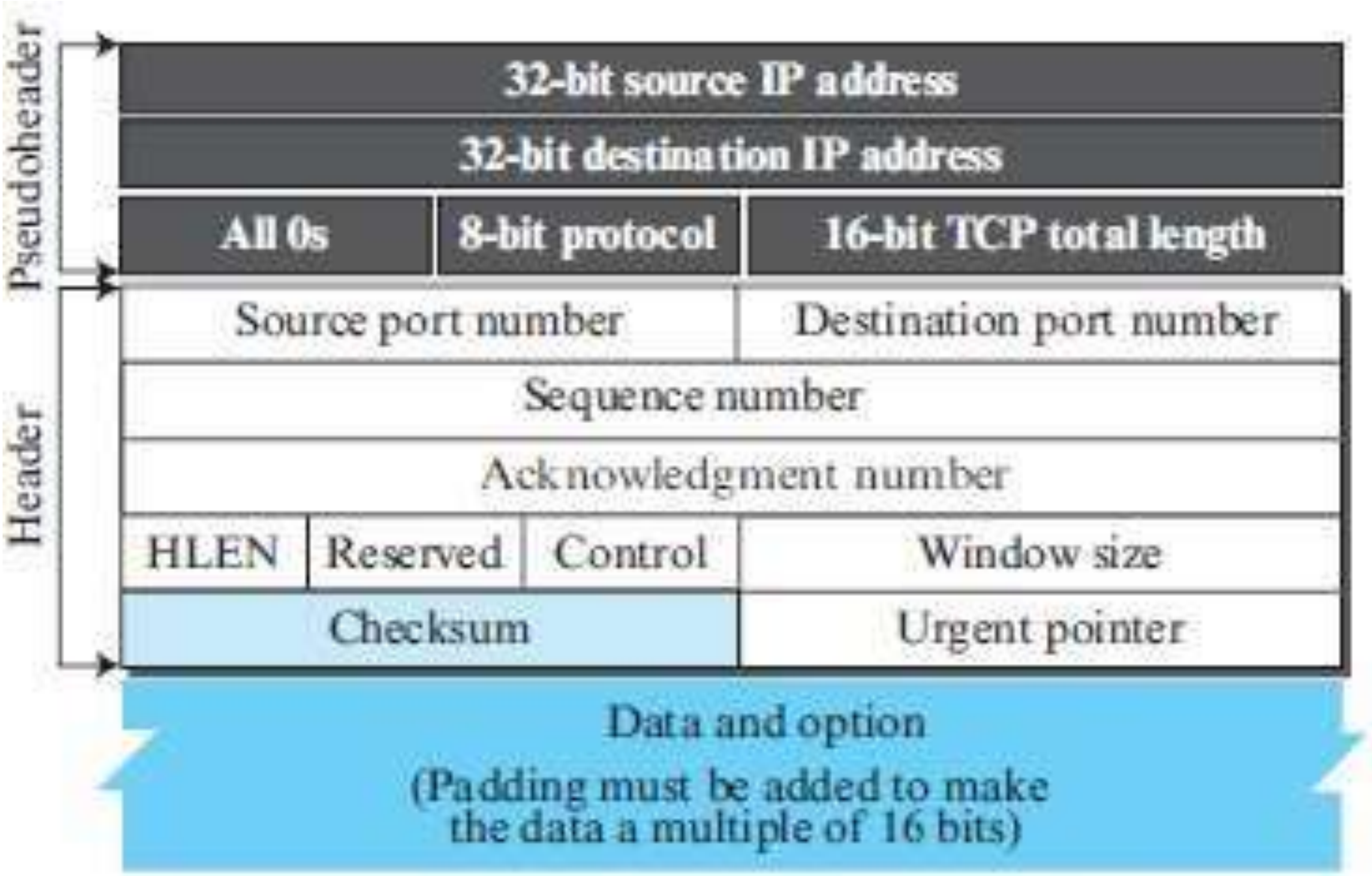
b. Header

- **Source port address**. This is a 16-bit field that defines the port number of the application program in the host that is sending the segment.
- **Destination port address**. This is a 16-bit field that defines the port number of the application program in the host that is receiving the segment.
- **Sequence number.** This 32-bit field defines the number assigned to the first byte of data contained in this segment.
- **Acknowledgment number.** This 32-bit field defines the byte number that the receiver of the segment is expecting to receive from the other party.
- **Header length**. This 4-bit field indicates the number of 4-byte words in the TCP header.

- **Control.** This field defines 6 different control bits or flags.
- One or more of these bits can be set at a time.
- These bits enable flow control, connection establishment and termination, connection abortion, and the mode of data transfer in TCP.


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
URG
ACK
PSH
RST
SYN
FIN
6 bits

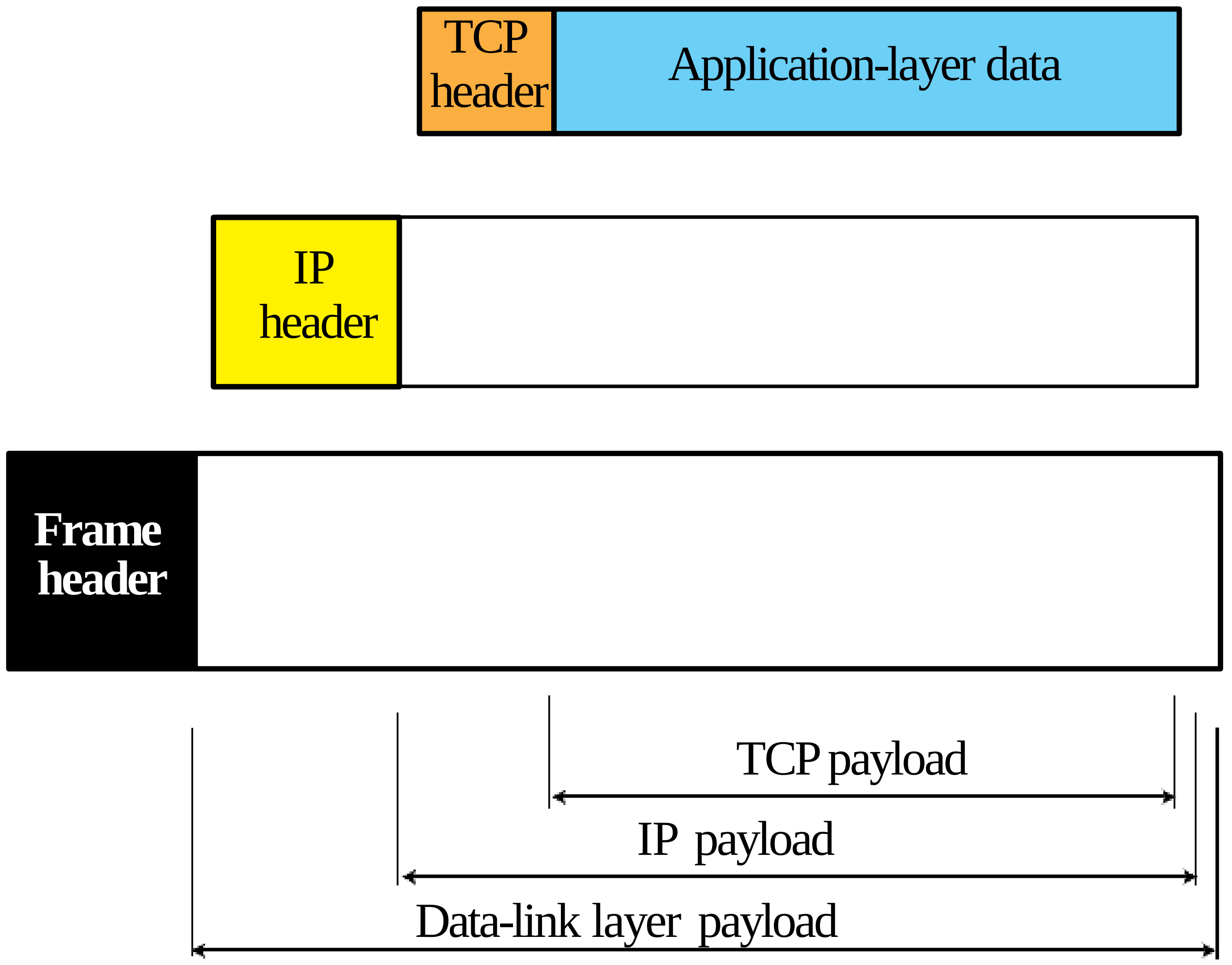
- **Window size.** This field defines the window size of the sending TCP in bytes.The length of this field is 16 bits. This value is normally referred to as the receiving window (rwnd) and is determined by the receiver.
- **Checksum.** This 16-bit field contains the checksum. The calculation of the checksum for TCP follows the same procedure as the one described for UDP. However, the use of the checksum in the UDP datagram is optional, whereas the use of the checksum for TCP is mandatory.
- The **pseudoheader** is added to the segment. For the TCP pseudo header, the value for the protocol field is 6.



Pseudoheader
Header
32-bit source IP address
32-bit destination IP address
All 0s
8-bit protocol
16-bit TCP total length
Source port number
Destination port number
Sequence number
Acknowledgment number
HLEN
Reserved
Control
Window size
Checksum
Urgent pointer
Data and option
(Padding must be added to make the data a multiple of 16 bits)

- **Urgent pointer.** This 16-bit field, which is valid only if the urgent flag is set, is used when the segment contains urgent data.
- **Options.** There can be up to 40 bytes of optional information in the TCP header.

Figure 15.8 *Encapsulation*



Three phases

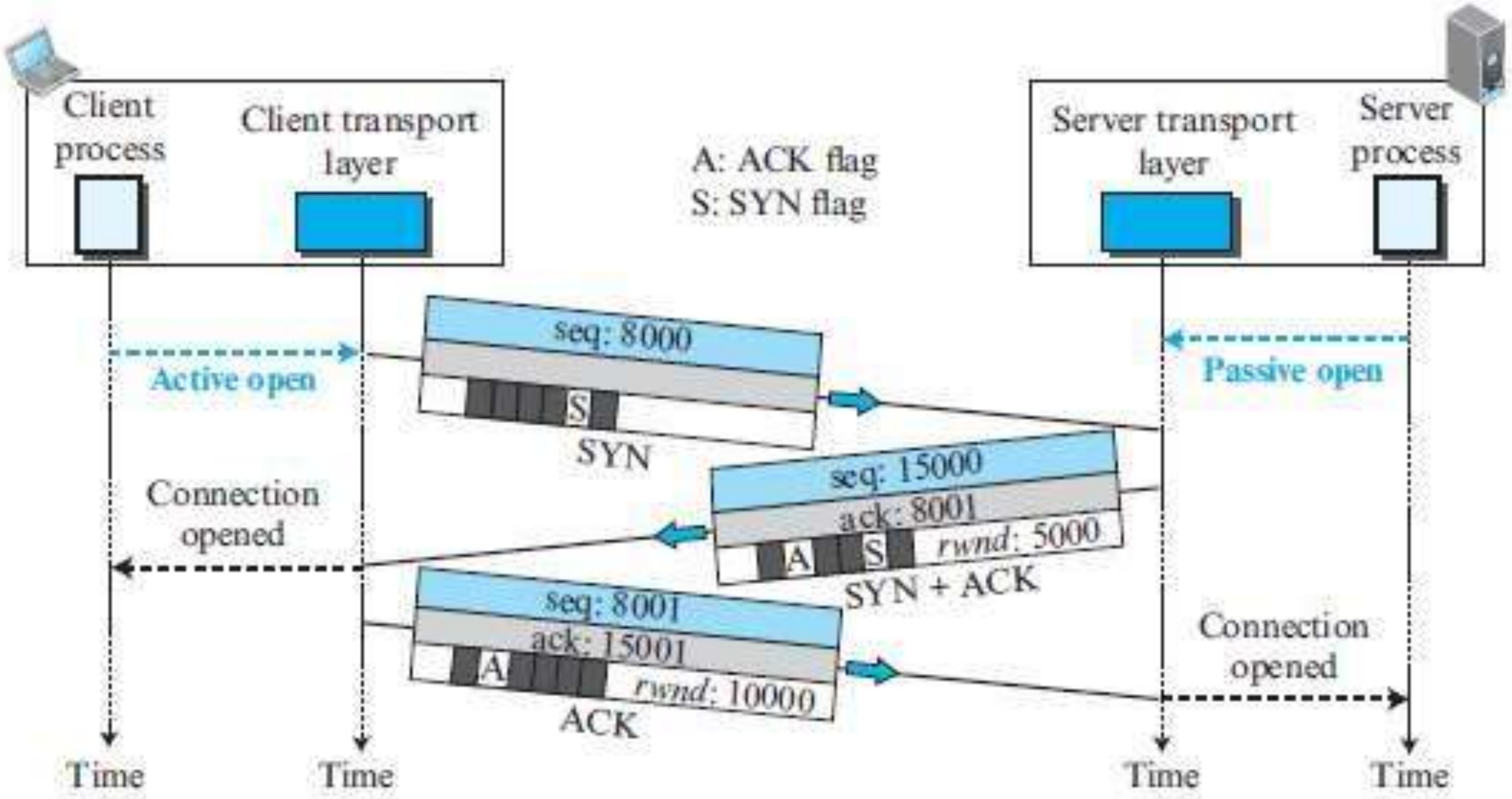
- ✓ **Connection Establishment**
- ✓ **Data Transfer**
- ✓ **Connection Termination**
- ✓ **Connection Reset**

TCP Connection phases

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

Connection Establishment

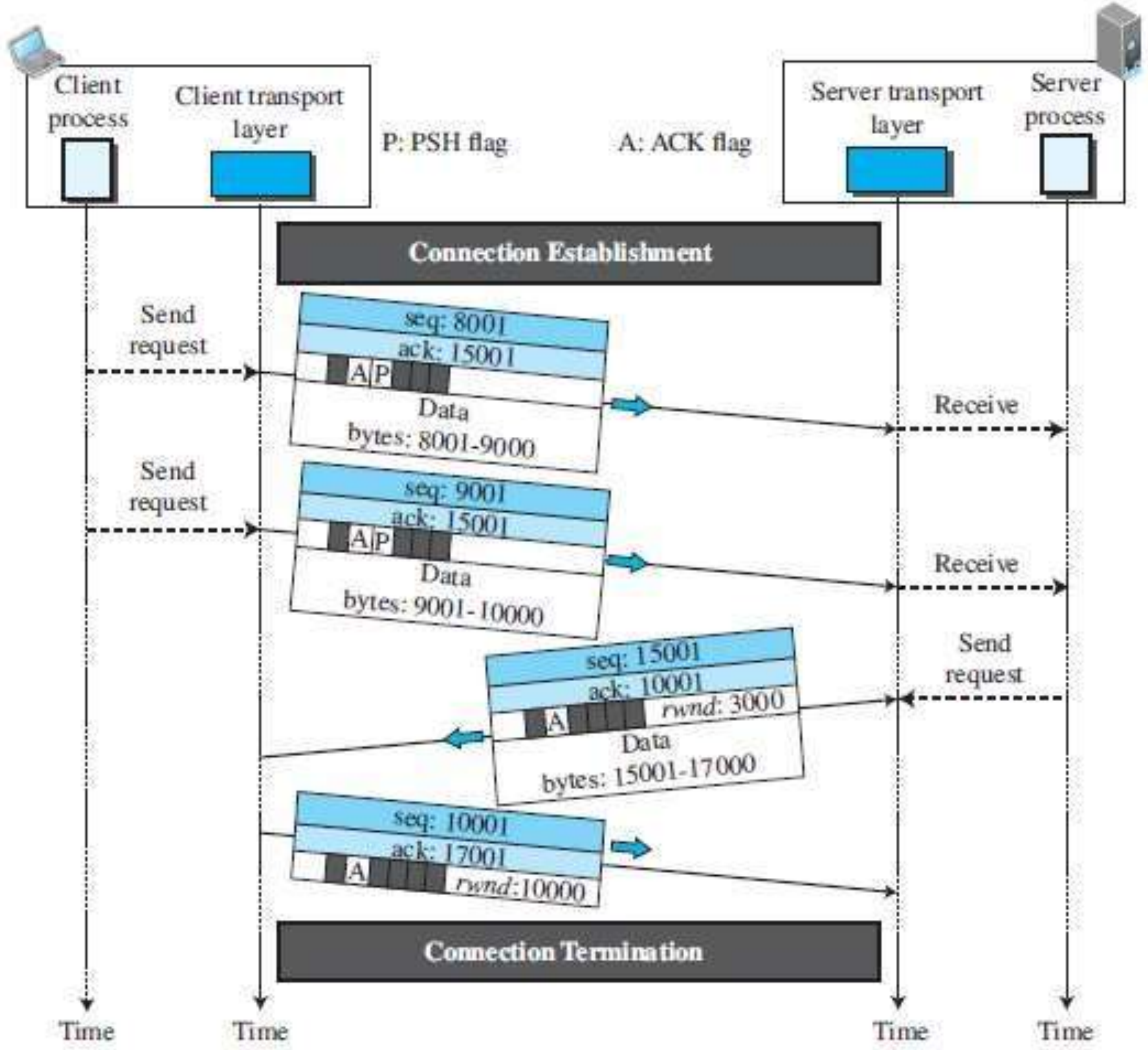
- TCP transmits data in full-duplex mode.
- Each party must initialize communication and get approval from the other party before any data are transferred.
- The connection establishment in TCP is called **three-way handshaking**.
- The process starts with the server. The server program tells its TCP that it is ready to accept a connection. This request is called a **passive open.**
- The client program issues a request for an **active open**.


Client process
Client transport layer
A: ACK flag
S: SYN flag
Server transport layer
Server process
seq: 8000
Active open
S
SYN
Passive open
seq: 15000
ack: 8001
A
S
rwnd: 5000
SYN + ACK
Connection opened
seq: 8001
ack: 15001
A
rwnd: 10000
ACK
Connection opened
Time
Time
Time
Time

- **1.** The client sends the first segment, a SYN segment, in which only the SYN flag is set.
- This segment is for synchronization of sequence numbers.
- The SYN segment is a control segment and carries no data.
- However, it consumes one sequence number because it needs to be acknowledged.
- **2.** The server sends the second segment, a SYN + ACK segment.
- This segment has a dual purpose. First, it is a SYN segment for communication in the other direction. The server also acknowledges the receipt of the SYN segment from the client by setting the ACK flag. It also needs to define the receive window size, rwnd (to be used by the client).
- **3.** The client sends the third segment. This is just an ACK segment. An ACK segment, if carrying no data, consumes no sequence number

Data Transfer

- After connection is established, bidirectional data transfer can take place. The client and server can send data and acknowledgments in both directions.
- In this example, after a connection is established, the client sends 2,000 bytes of data in two segments. The server then sends 2,000 bytes in one segment. The client sends one more segment.
- The data segments sent by the client have the PSH (push) flag set.


Client process
Client transport layer
P: PSH flag
A: ACK flag
Server transport layer
Server process
Connection Establishment
Send request
seq: 8001
ack: 15001
A P
Data bytes: 8001-9000
Receive
Send request
seq: 9001
ack: 15001
A P
Data bytes: 9001-10000
Receive
Send request
seq: 15001
ack: 10001
A
rwnd: 3000
Data bytes: 15001-17000
seq: 10001
ack: 17001
A
rwnd:10000
Connection Termination
Time
Time
Time
Time

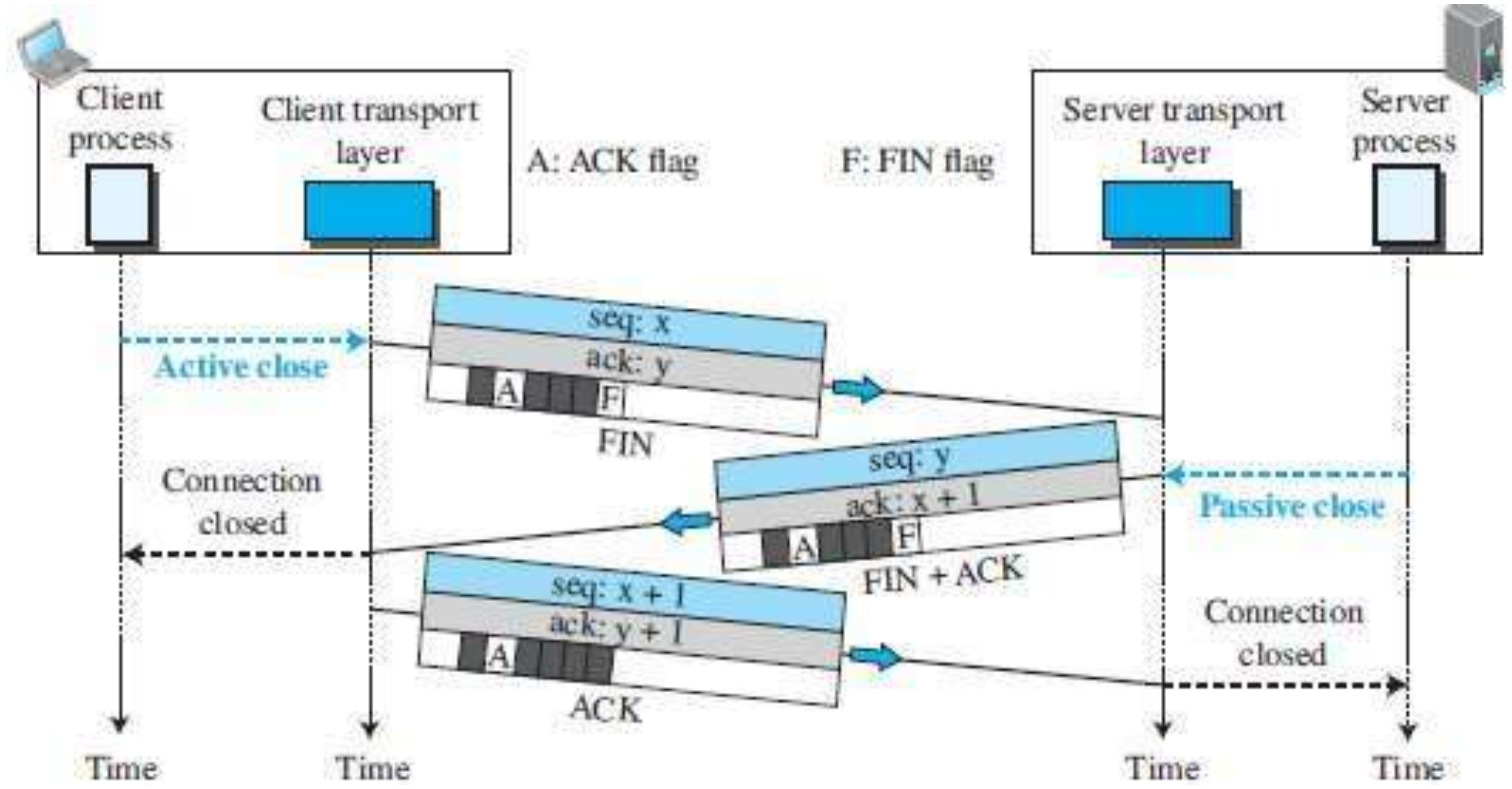
Connection Termination

- Either 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.
- In TCP, one end can stop sending data while still receiving data. This is called **a halfclose**. Either the server or the client can issue a half-close request. It can occur when the server needs all the data before processing can begin.

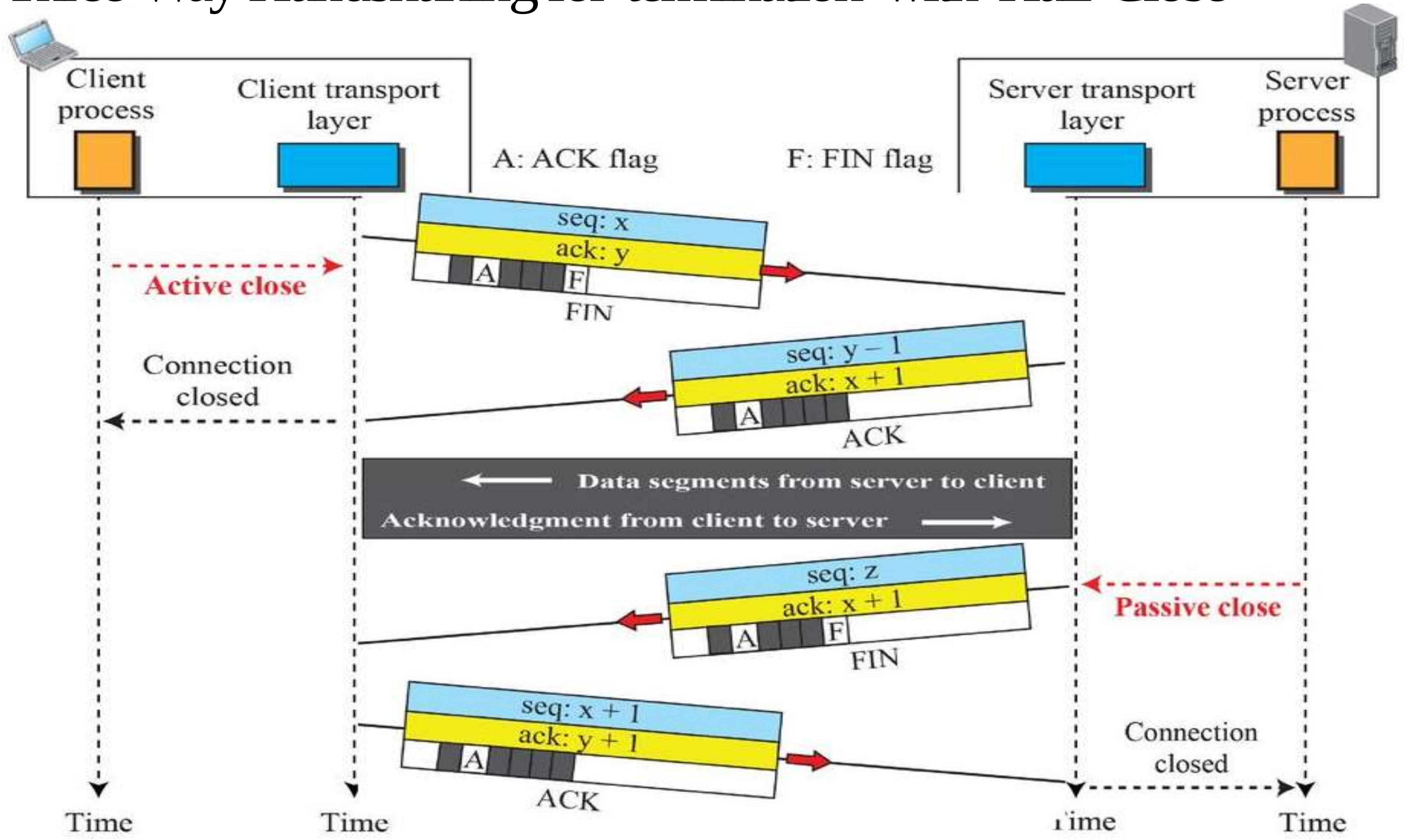
Three-Way Handshaking fortermination

- **1.** In this situation, the client TCP, after receiving a close command from the client process, sends the first segment, a FIN segment in which the FIN flag is set.
- **2.** The server TCP, after receiving the FIN segment, informs its process of the situation and sends the second segment, a FIN+ACK segment.
- **3.** The client TCP sends the last segment, an ACK segment, to confirm the receipt of the FIN segment from the TCP server.

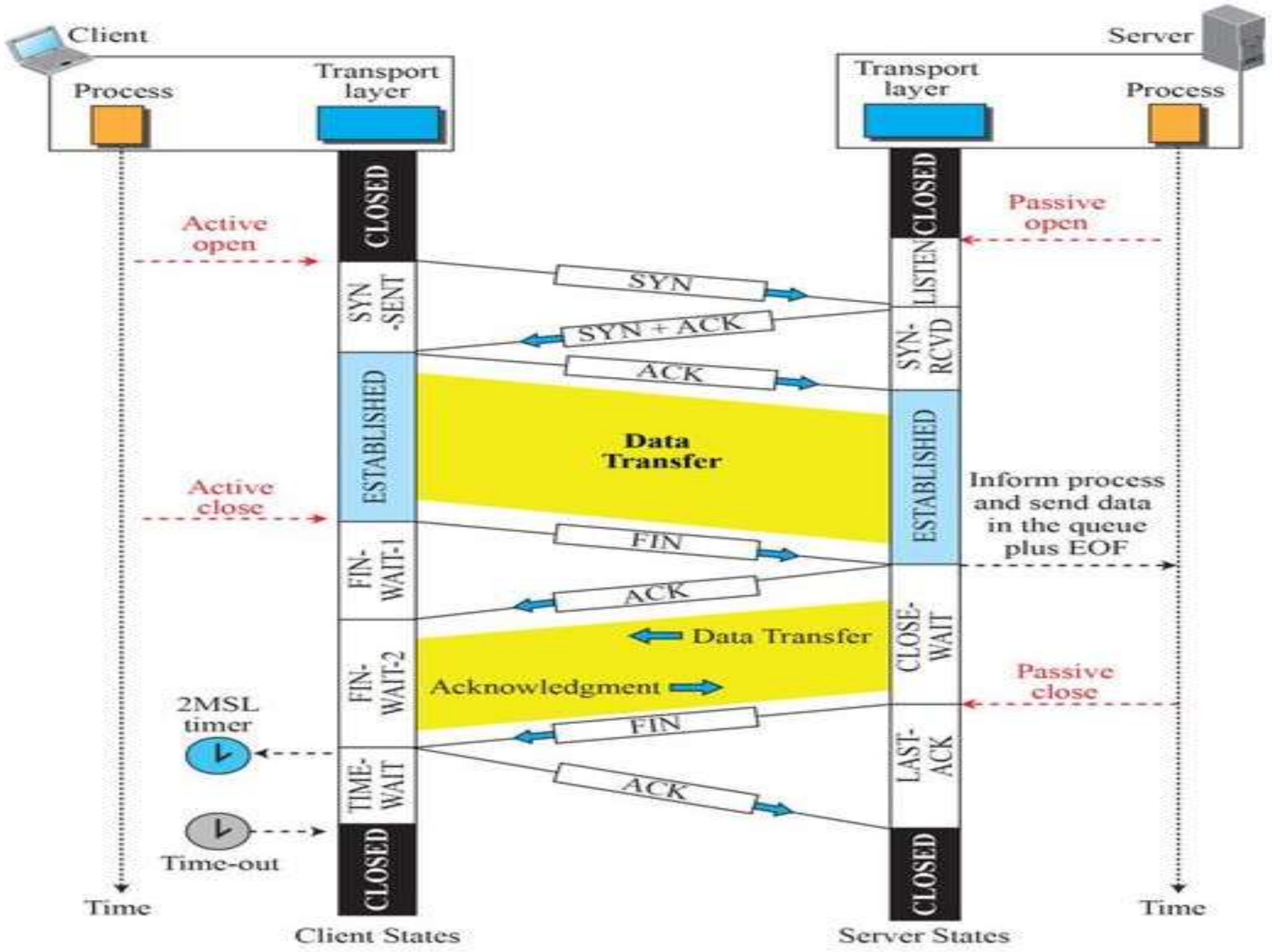
Three way Handshaking for termination



Three-Way Handshaking for termination with Half-Close



Time-line diagram for a common scenario



Connection Reset

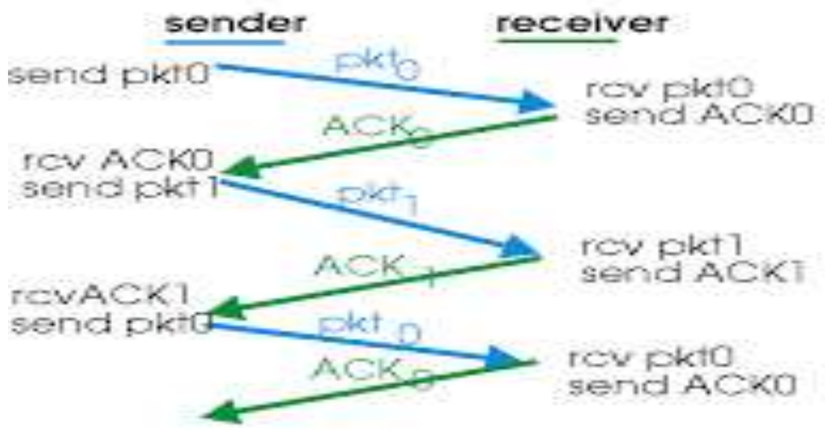
- TCP at one end may deny a connection request, may abort an existing connection, or may terminate an idle connection.
- All of these are done with the RST (reset) flag.

Mechanisms for reliable data transfer

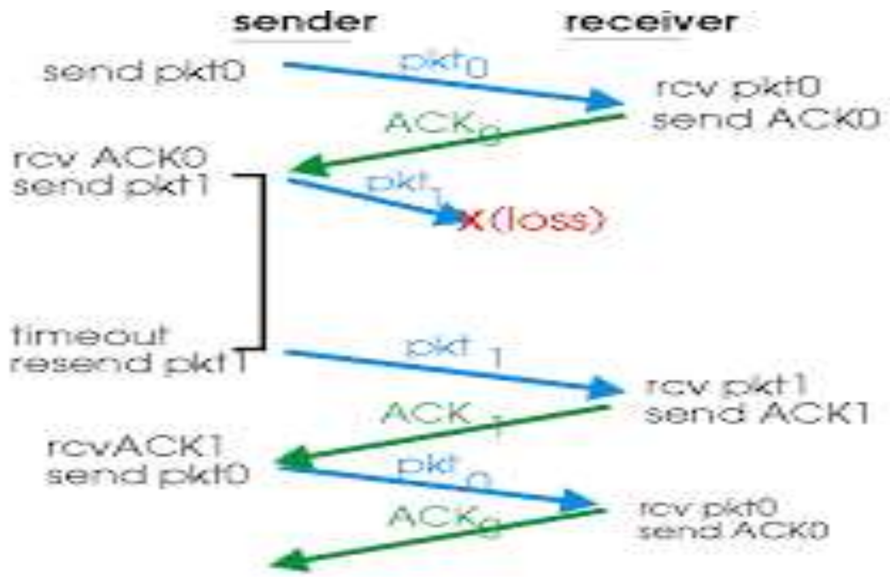
- Checksum
  - Used to detect bit errors in a transmitted packet.
- Timer
  - Used to timeout/retransmit a packet, possibly because the packet (or its ACK) was lost within the channel.
- Sequence number
  - Used for sequential numbering of packets of data flowing from sender to receiver.
- Acknowledgment
  - Used by the receiver to tell the sender that a packet or set of packets has been received correctly.
- Negative acknowledgment
  - Used by the receiver to tell the sender that a packet has not been received correctly.
- Window, pipelining
  - The sender may be restricted to sending only packets with sequence numbers that fall within a given range.

Error Control : Alternating-bit protocol.

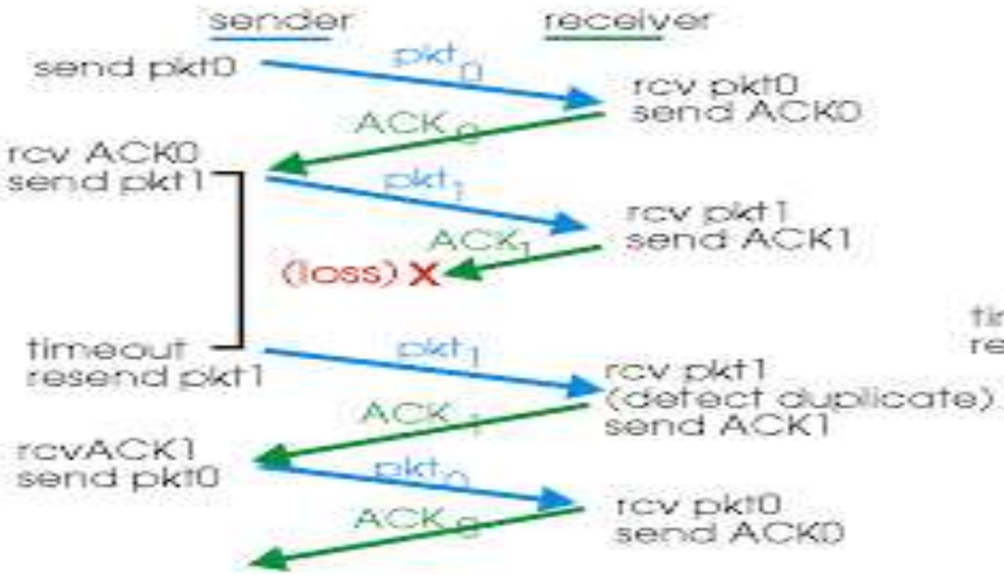
- If the packet sequence numbers alternate between 0 and 1,then the protocol is known as the **alternating-bit protocol**.
- The operations with this protocol are:
- **a. Operation with no loss**
- **b. Lost packet**
- **c. Lost ACK**
- **d. Premature timeout**



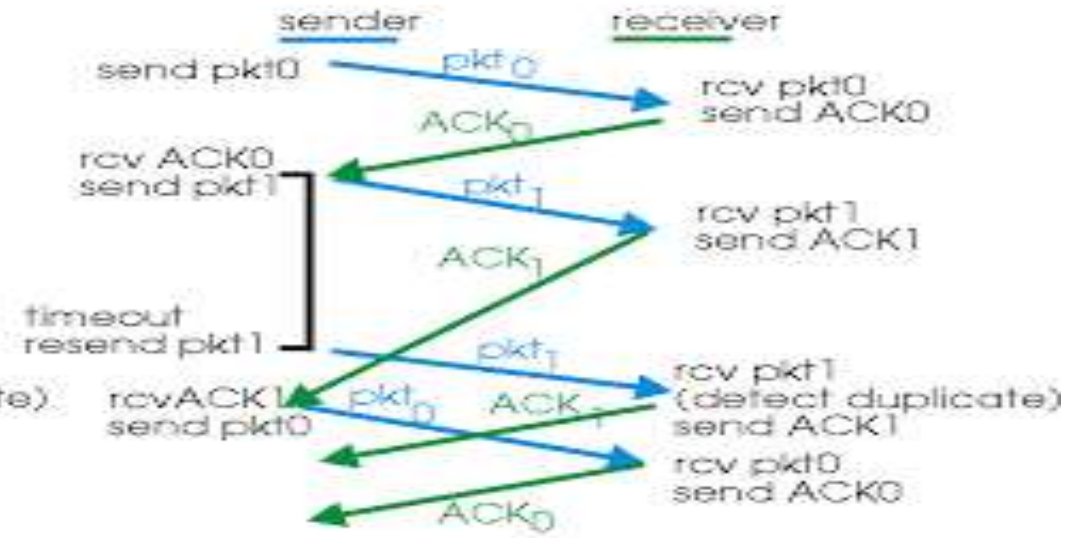
(a) operation with no loss



(b) lost packet



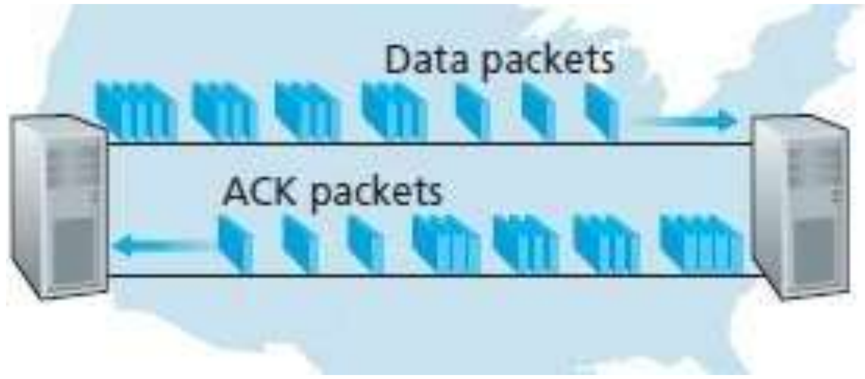
(c) lost ACK



(d) premature timeout

pipelining

- The sender is allowed to send multiple packets without waiting for acknowledgments.
- Since the many in-transit sender-to-receiver packets can be visualized as filling a pipeline, this technique is known as **pipelining**.

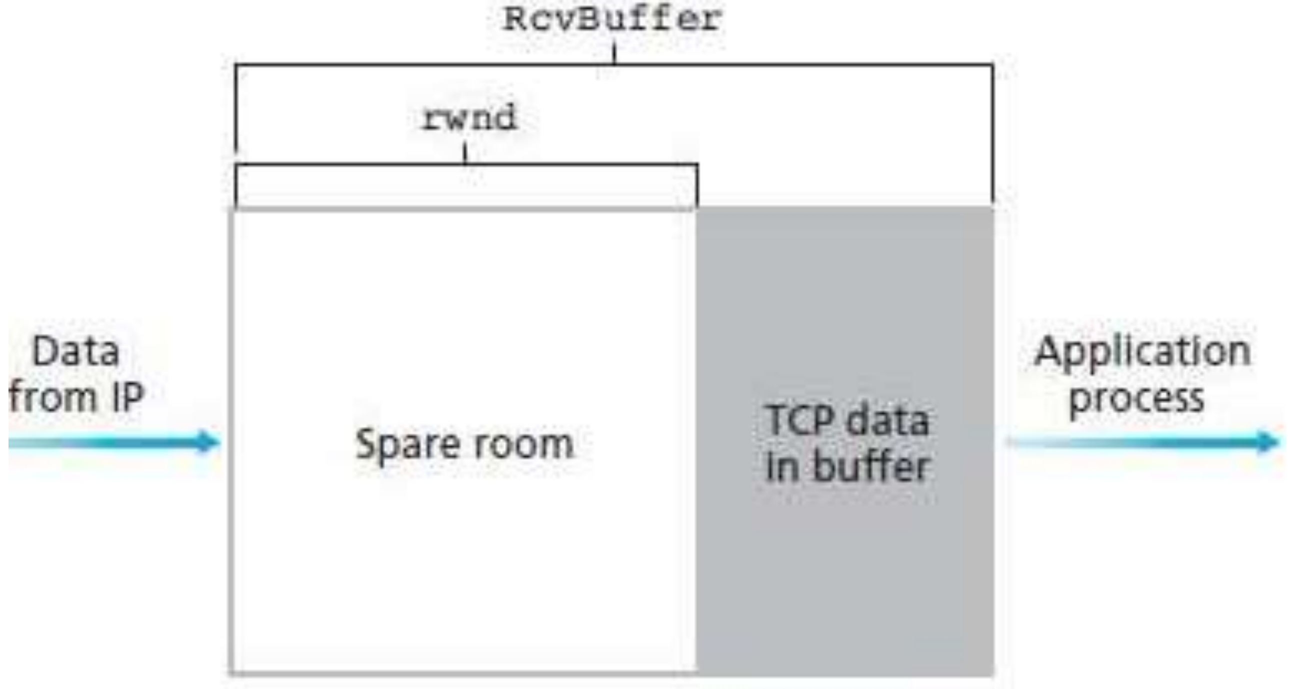


- Pipelining has the following consequences for reliable data transfer protocols:

- **The range of sequence numbers must be increased**, since each in- transit packet (not counting retransmissions) must have a unique sequence number and there may be multiple, in-transit, unacknowledged packets.
- **The sender and receiver sides of the protocols may have to buffer** more than one packet.
- **A data transfer protocol responds** to lost, corrupted, and overly delayed packets.
- Two basic approaches toward pipelined error recovery can be identified: **Go-Back-N** and **selective repeat**.

Flow control in TCP

- TCP provides a **flow-control service** to its applications to eliminate the possibility of the sender overflowing the receiver's buffer.
- Flow control is thus a speed-matching service—matching the rate at which the sender is sending against the rate at which the receiving application is reading.
- TCP provides flow control by having the *sender* maintain a variable called the **receive window**.
- Informally, the receive window is used to give the sender an idea of how much free buffer space is available at the receiver.



38 • The receive window (rwnd) and the receive buffer (RcvBuffer)

- Because TCP is not permitted to overflow the allocated buffer, we must have
- LastByteRcvd – LastByteRead <= RcvBuffer
- The receive window, denoted rwnd is set to the amount of spare room in the buffer:
- rwnd = RcvBuffer – [LastByteRcvd – LastByteRead]

- Suppose that Host A is sending a large file to Host B over a TCP connection.
- Host B allocates a receive buffer to this connection; denote its size by RcvBuffer.
- Host B tells Host A how much spare room it has in the connection buffer by placing its current value of rwnd in the receive window field of every segment it sends to A.
- By keeping the amount of unacknowledged data less than the value of rwnd, Host A is assured that it is not overflowing the receive buffer at Host B.
- Thus, Host A makes sure throughout the connection’s life that
- LastByteSent – LastByteAcked <= rwnd

SYN Flooding Attack

- The connection establishment procedure in TCP is susceptible to a serious security problem called SYN flooding attack.
- This happens when one or more malicious attackers send a large number of SYN segments to a server pretending that each of them is coming from a different client by faking the source IP addresses in the datagrams.
- The server, assuming that the clients are issuing an active open, allocates the necessary resources, such as creating transfer control block (TCB) tables and setting timers.
- The TCP server then sends the SYN + ACK segments to the fake clients, which are lost.

- When the server waits for the third leg of the handshaking process, however, resources are allocated without being used.
- If, during this short period of time, the number of SYN segments is large, the server eventually runs out of resources and may be unable to accept connection requests from valid clients.
- This SYN flooding attack belongs to a group of security attacks known as a denial of service attack, in which an attacker monopolizes a system with so many service requests that the system overloads and denies service to valid requests.
- Fortunately, an effective defense known as SYN cookies are now deployed in most major operating systems.