20MCA104
ADVANCED COMPUTER NETWORKS

Module 2

TRANSPORT LAYER

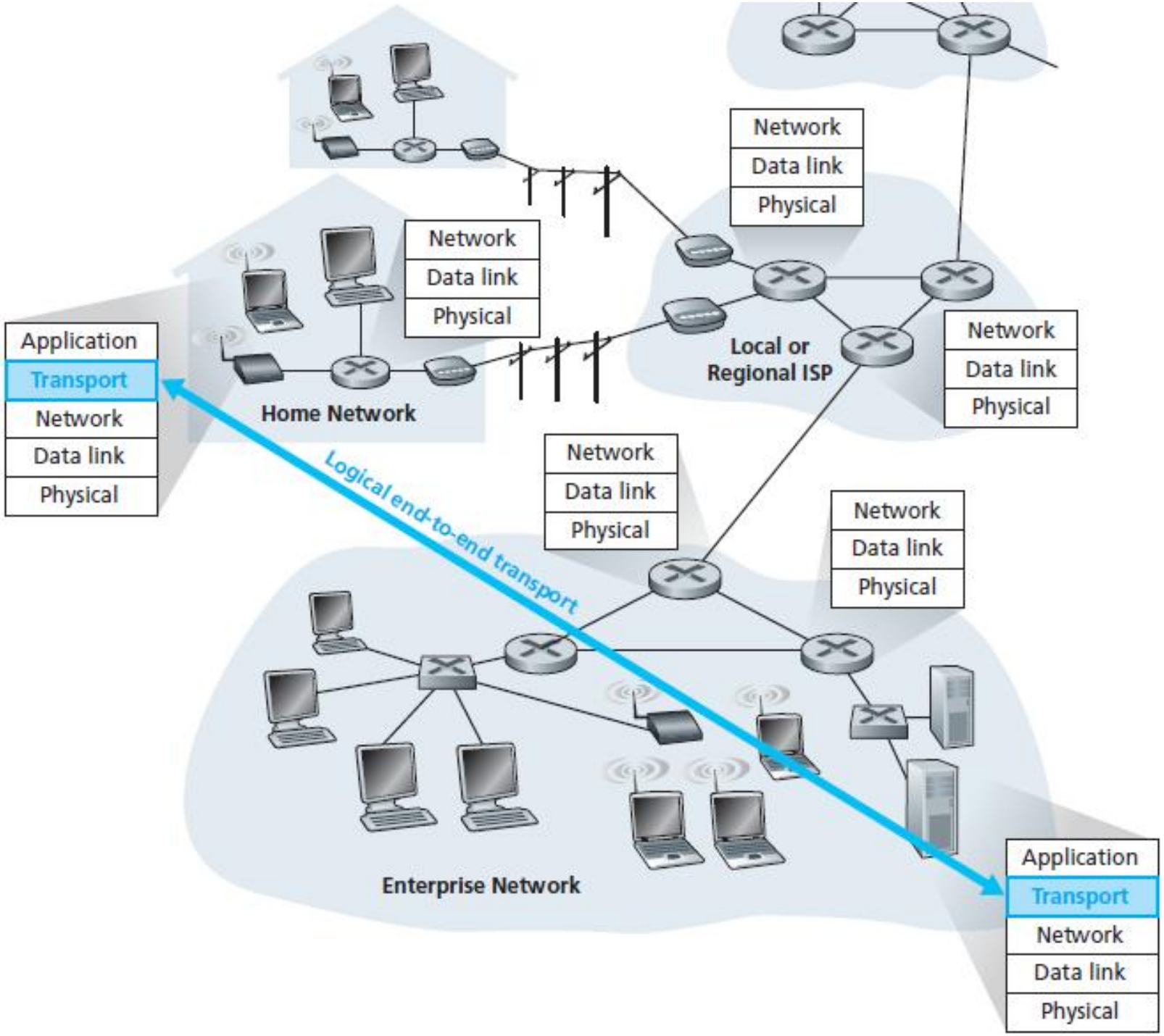
Transport Layer

- The critical function of the transport layer:
  - extending the network layer's delivery service between two end systems to a delivery service between two application-layer processes running on the end systems.
- Fundamental problems:
  - Handling lost and corrupt data.
  - Managing congestion within the network.

Transport-Layer Services

- Provide *logical communication* between application processes running on different hosts.

- Transport-layer protocols are implemented in the end systems.
  - but not in network routers.


Application
Transport
Network
Data link
Physical
Network
Data link
Physical
Home Network
Network
Data link
Physical
Local or
Regional ISP
Network
Data link
Physical
Network
Data link
Physical
Network
Data link
Physical
Logical end-to-end transport
Enterprise Network
Application
Transport
Network
Data link
Physical

- Sending side
  - Receives application-layer messages.
  - Converts into transport-layer packets, known as **segments**.
    - Breaks the application messages into smaller chunks.
    - Adds a transport-layer header to each chunk.
  - Passes the segment to the network layer.
- Receiving side
  - Network layer extracts segment from the datagram.
  - Passes the segment up to the transport layer.
  - The transport layer processes the received segment.
  - Passes the data in the segment to the receiving application.

Relationship Between Transport and Network Layers

- Network layer:
  - Logical communication between hosts.
- Transport layer:
  - Logical communication between processes.
    - Relies on network layer services.

Transport Layer Protocols

- **UDP (User Datagram Protocol)**
  - Provides an **unreliable, connectionless** service to the invoking application.
  - Minimal transport-layer services:
    - process-to-process data delivery and error checking
- **TCP (Transmission Control Protocol)**
  - Provides a **reliable, connection-oriented** service to the invoking application.
  - Services
    - process-to-process data delivery and error checking
    - congestion control
    - flow control

A few words to IP

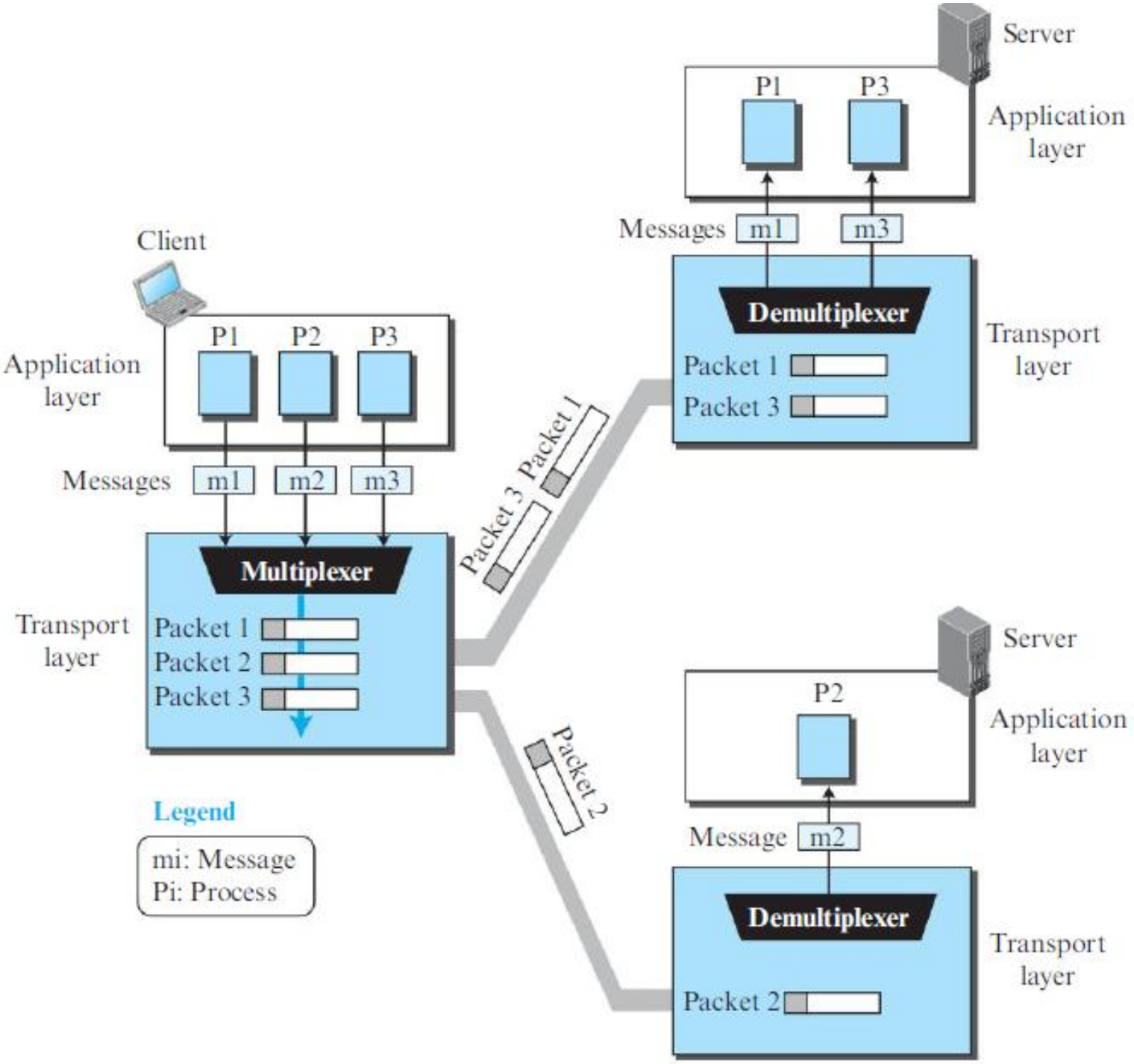
- IP - Internet protocol.
- Protocol in the Network layer.
- IP provides logical communication between **hosts**.
- The IP service model is a **best-effort delivery service.**
- This means that IP makes its "best effort" to deliver segments between communicating hosts, *but it makes no guarantees.*
- It does not guarantee
  - Segment delivery,
  - Orderly delivery of segments,
  - Integrity of the data in the segments.
- For these reasons, IP is said to be an **unreliable service.**
- *Each host has an **IP address**.*

Multiplexing and Demultiplexing

- The most fundamental responsibility of UDP and TCP is
  - 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**.

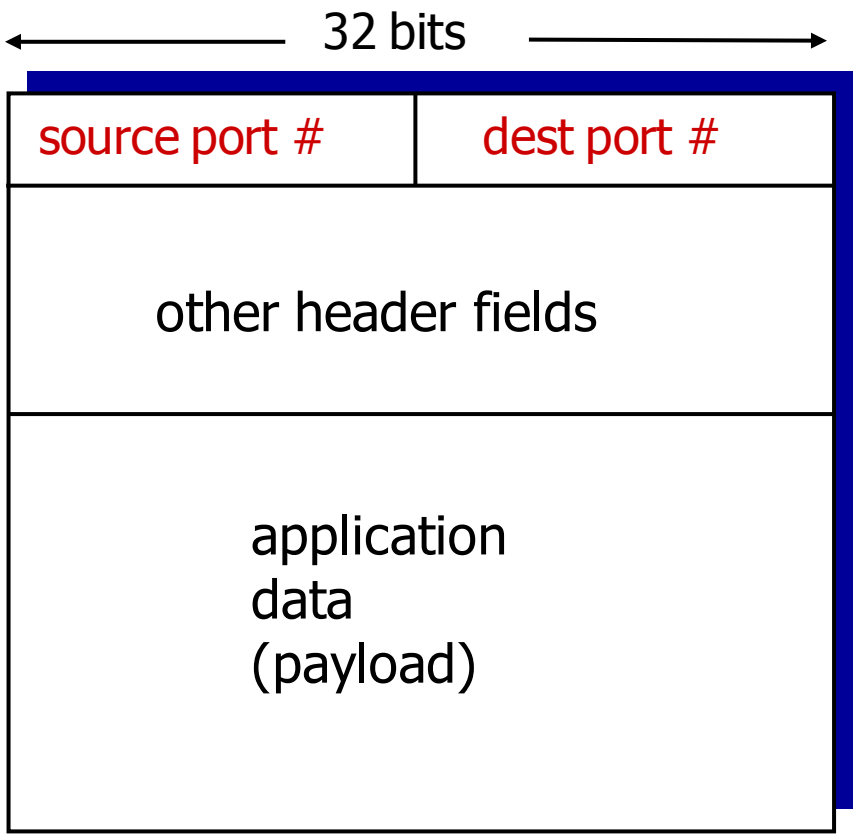
Multiplexing and Demultiplexing…

- The job of gathering data chunks at the source host from different sockets, encapsulating each data chunk with header information to create segments, and passing the segments to the network layer is called **multiplexing.**
- At the receiving end, the transport layer identify the receiving socket and then directs the segment to that socket. This job of delivering the data in a transport-layer segment to the correct socket is called **demultiplexing.**


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

How it works?

- Transport-layer multiplexing requires
  1. sockets have unique identifiers, and
  2. special fields that indicate the socket to which the segment is to be delivered.
     - **Source port number** field
     - **Destination port number** field.
- Each socket in the host could be assigned a port number.



TCP/UDP segment format

- Multiplexing service
  - Gather outgoing data from sockets,
  - Form transport-layer segments, and
  - Pass segments down to the network layer.

- Demultiplexing service:
  - When a segment arrives at the host, the transport layer examines the destination port number in the segment and directs the segment to the corresponding socket.
  - The segment's data then passes through the socket into the attached process.

What is the purpose of the source port number?

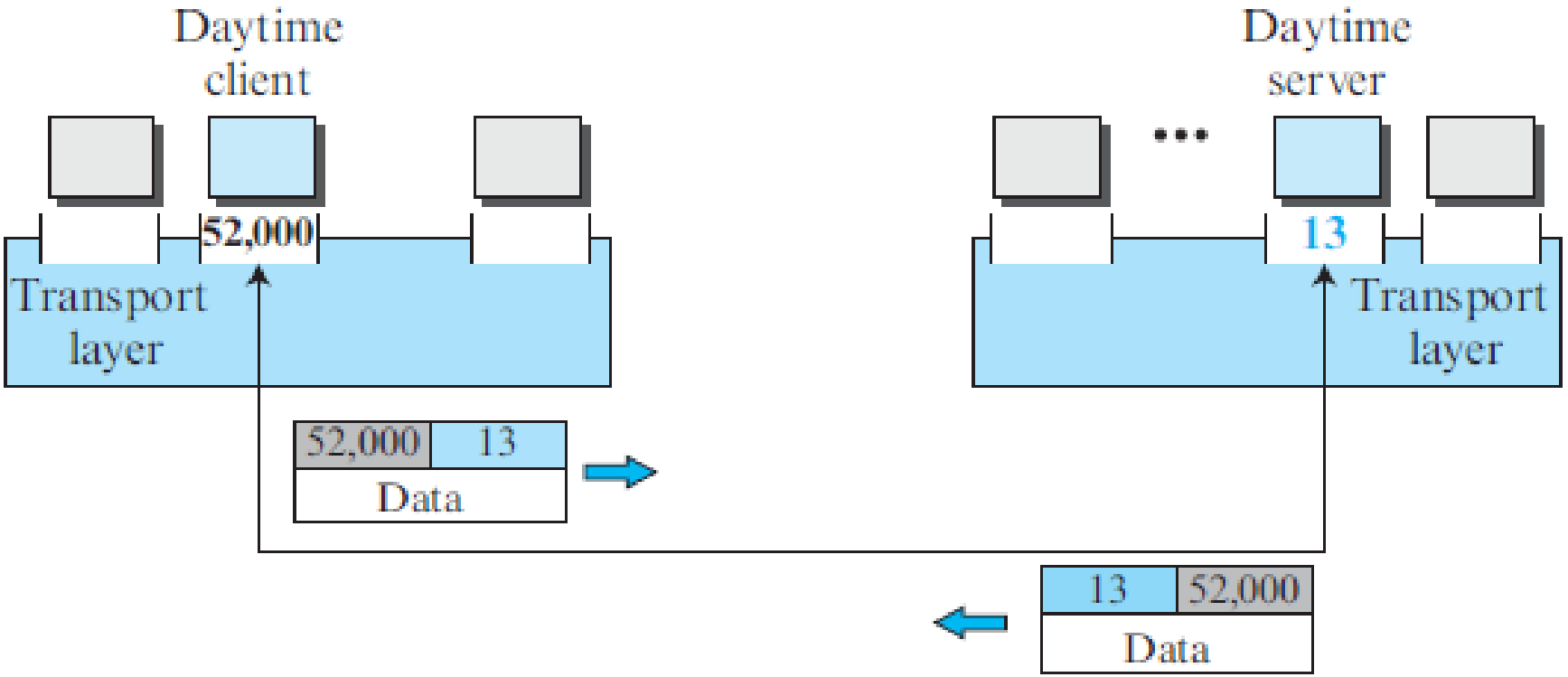


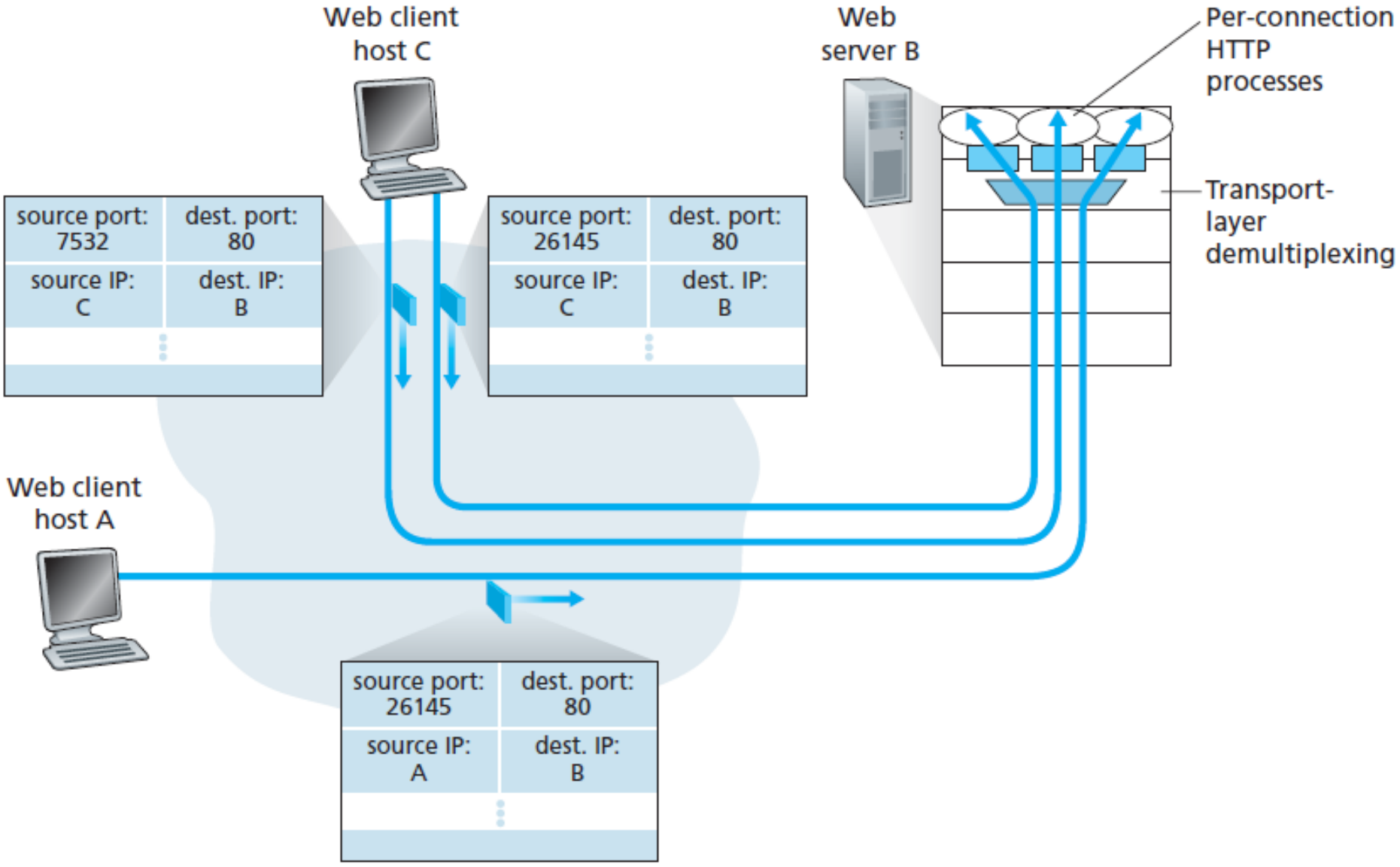
Fig: The inversion of source and destination port numbers

Connectionless demultiplexing

- When host receives UDP segment:
  - Checks destination port number in segment.
  - Directs UDP segment to socket with that port number.
- IP datagrams with *same destination port number,* but different source IP addresses and/or source port numbers will be directed to *same socket* at destination.

Connection-Oriented Multiplexing and Demultiplexing

- TCP socket identified by 4-tuple:
  - Source IP address
  - Source port number
  - Destination IP address
  - Destination port number
- Receiver uses all four values to direct segment to appropriate socket.
- Web servers have different sockets for each connecting client.


Web client host C
Web server B
Per-connection HTTP processes
Transport-layer demultiplexing
source port: 7532
dest. port: 80
source IP: C
dest. IP: B
source port: 26145
dest. port: 80
source IP: C
dest. IP: B
Web client host A
source port: 26145
dest. port: 80
source IP: A
dest. IP: B

Port Numbers

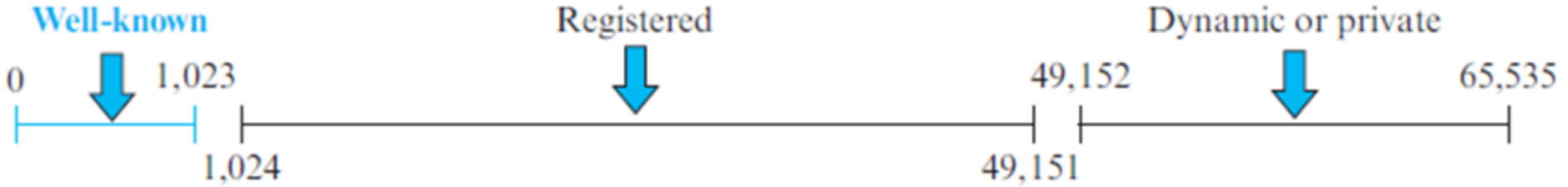
- Each port number is a 16-bit number, ranging from 0 to 65535.
- The port numbers ranging from **0** to **1023** are called **well-known port numbers.**
  - restricted, which means that they are reserved for use by well-known application protocols such as
    - HTTP (port number 80) and
    - FTP (port number 21).
  - The list of well-known port numbers is given in RFC 1700.
  - Updated at http://www.iana.org [RFC 3232].

Port Numbers…

- 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 1,023 for some client/server programs to work properly.

ICANN Ranges

- ICANN has divided the port numbers into three ranges:
- **Well-known ports**.
  - The ports ranging from 0 to 1,023 are assigned and controlled by ICANN.
- **Registered 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.
- **Dynamic ports**.
  - The ports ranging from 49,152 to 65,535 are neither controlled nor registered.
  - They can be used as temporary or private port numbers.



Principles of Reliable Data Transfer

- A **reliable data transfer protocol** implements the service abstraction:

  With a reliable channel, transferred data bits are:
  - Not corrupted (flipped from 0 to 1, or vice versa)
  - Not lost,
  - Delivered in the order in which they were sent.

- It is difficult to implement because
  - Layer *below* the reliable data transfer protocol may be unreliable.
    - E.g. TCP is a reliable data transfer protocol that is implemented on top of an unreliable (IP) end-to-end network layer.

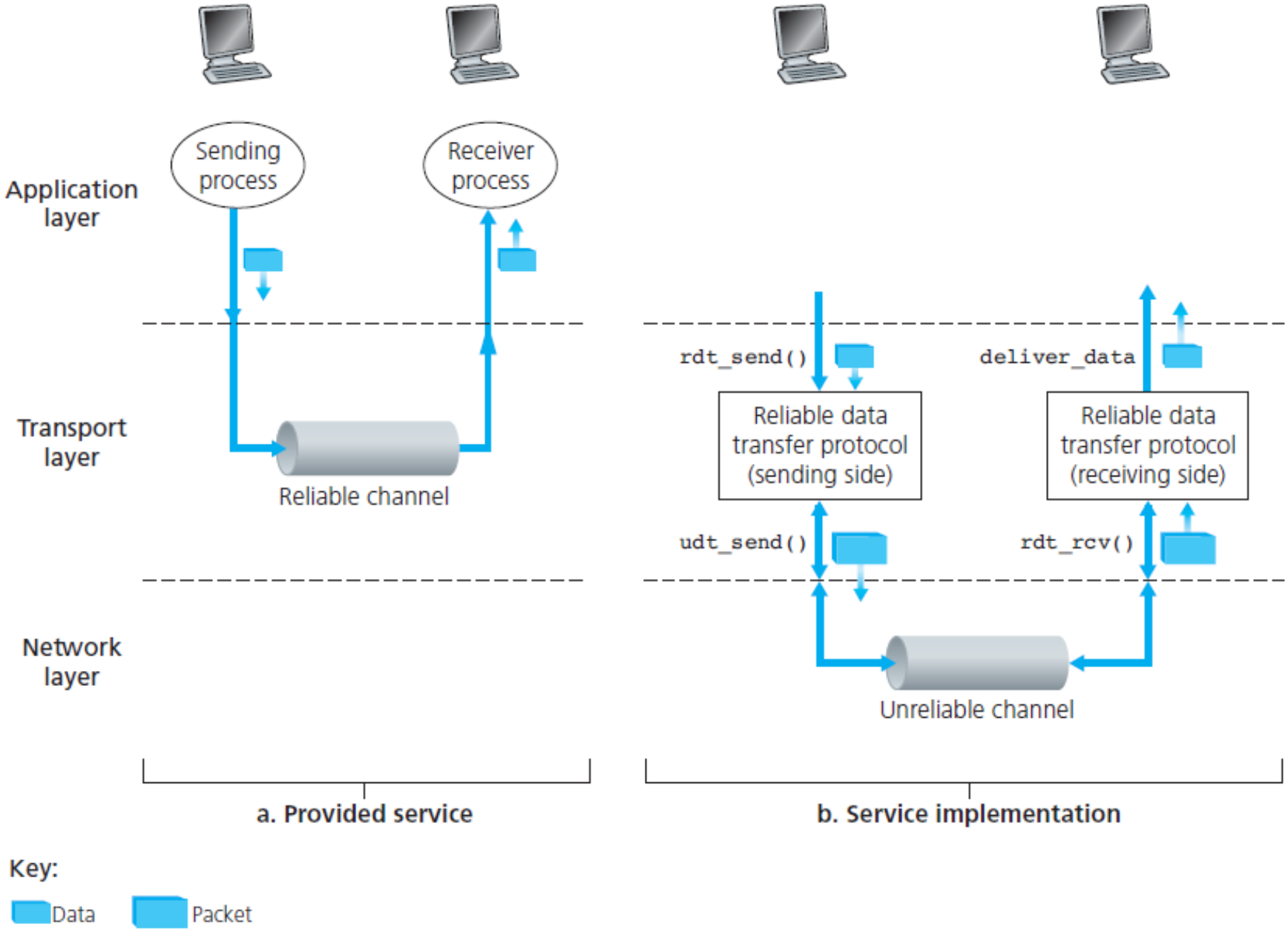


Figure : Reliable data transfer: Service model and service implementation

Finite State Machine

- The behavior of a transport-layer protocol can be better shown as a **finite state machine (FSM).**
- Using this tool, each transport layer (sender or receiver) is thought as a machine with a finite number of states.
- The machine is always in one of the states until an *event* occurs.
- Each event is associated with two reactions:
  - Defining the list of actions to be performed.
  - Determining the next state.
- One of the states must be defined as the *initial state*, the state in which the machine starts when it turns on.

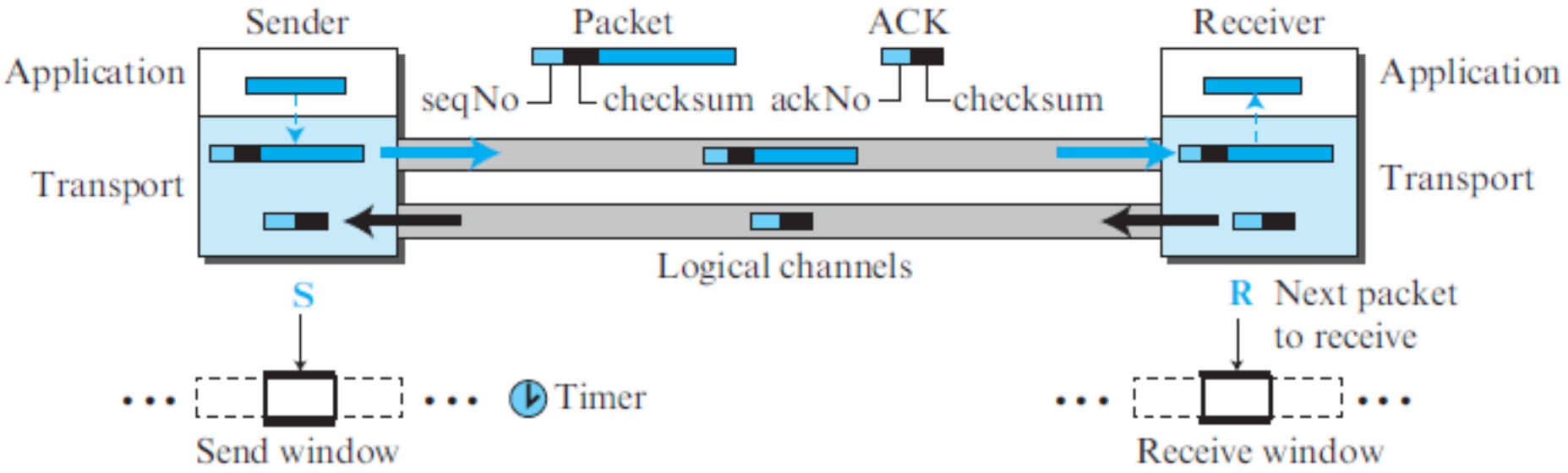
Stop-and-Wait Protocol

- The sender sends one packet at a time and waits for an acknowledgment before sending the next one.
- To detect corrupted packets, a checksum is added 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.
- The silence of the receiver is a signal for the sender that a packet was either corrupted or lost.

Stop-and-Wait Protocol…

- Every time the sender sends a packet, it starts a ***timer***.
- If an acknowledgment arrives before the timer expires, the timer is stopped and the sender sends the next packet.
- If the timer expires, the sender resends the previous packet, assuming that the packet was either lost or corrupted.
  - Sender needs to keep a copy of the packet until its acknowledgment arrives.
- Only one packet and one acknowledgment can be in the channels at any time.

Stop-and-Wait Protocol…



Sequence & Acknowledgment Numbers

- To prevent duplicate packets, the protocol uses sequence numbers and acknowledgment numbers.
- Assume that the sender has sent the packet with sequence number ***x***.
- Three things can happen.
  1. The packet arrives safe at the receiver site; the receiver sends an acknowledgment. The acknowledgment arrives at the sender site, causing the sender to send the next packet numbered x + 1.
  2. The packet is corrupted or never arrives at the receiver site; the sender resends the packet (numbered x) after the time-out. The receiver returns an acknowledgment.
  3. The packet arrives safe at the receiver site; the receiver sends an acknowledgment, but the acknowledgment is corrupted or lost. The sender resends the packet (numbered x) after the time-out. Note that the **packet** here **is a duplicate.**

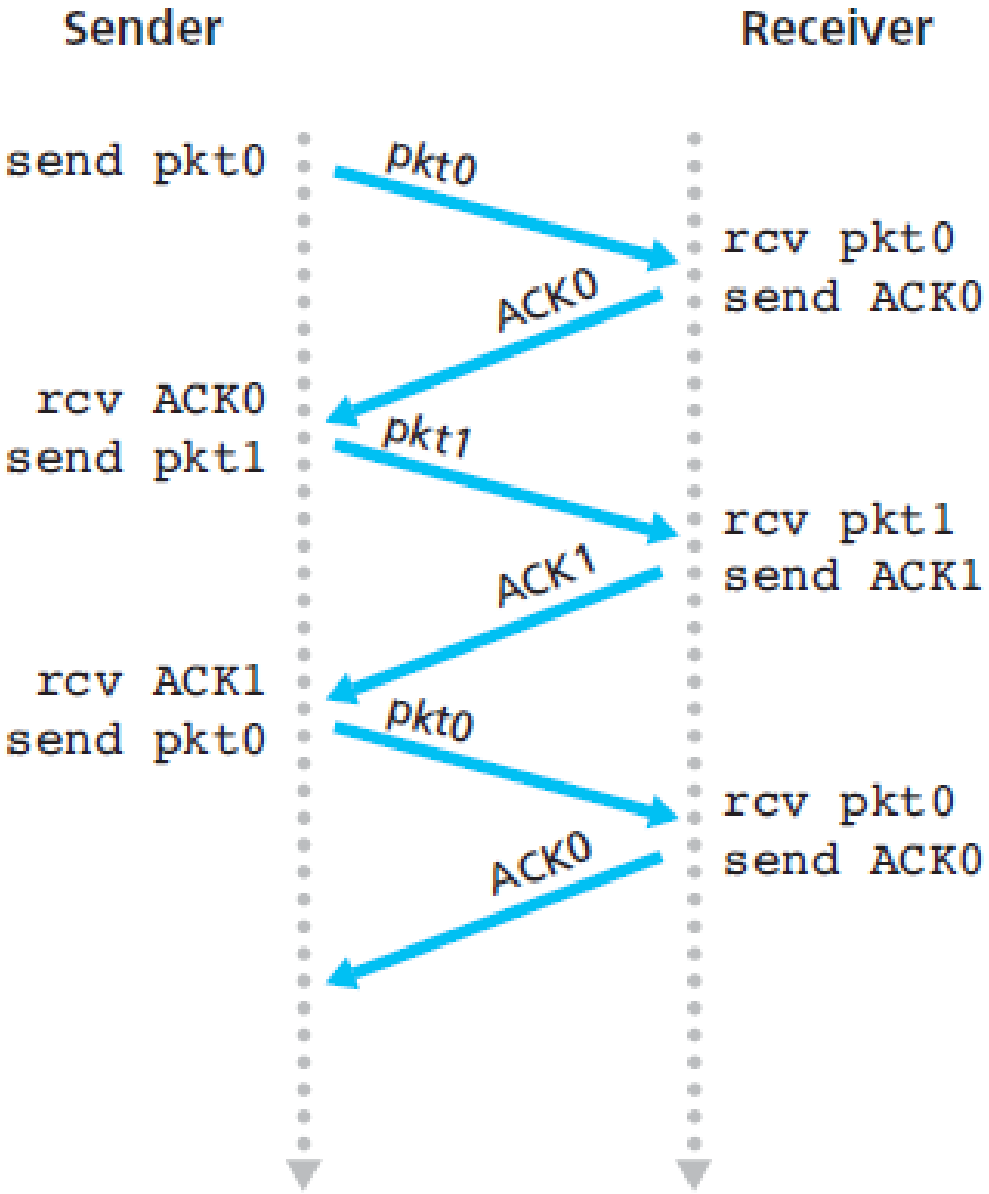
Sequence Numbers…

- There is a need for sequence numbers x and x+1 because the receiver needs to distinguish between case 1 and case 3.
- But there is no need for a packet to be numbered x+2.
- If only x and x+1 are needed, we can 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.

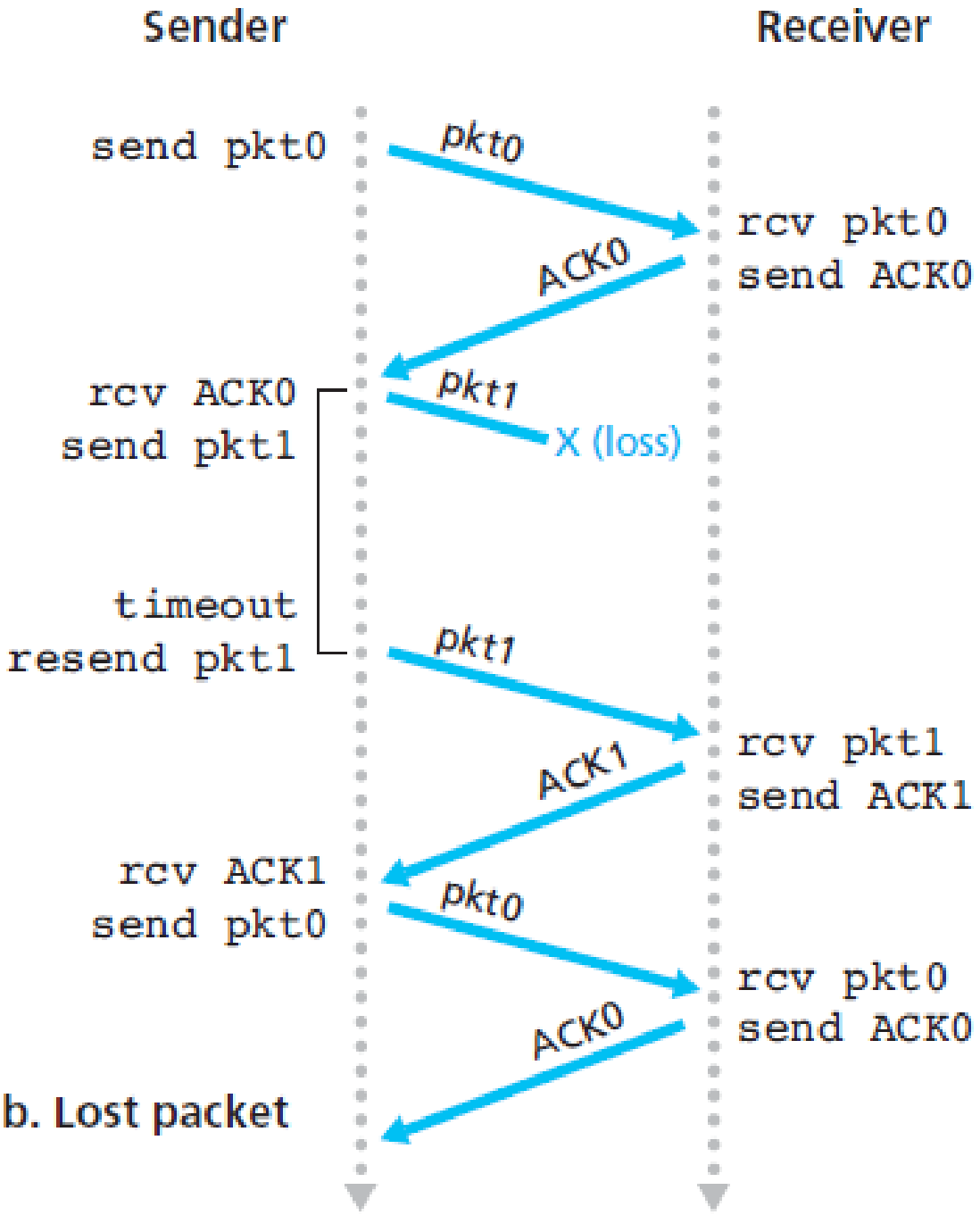
Acknowledgment Numbers…

- The acknowledgment numbers always announce the sequence number of the next packet expected by the receiver.
  - For example, if packet 0 has arrived safe, the receiver sends an ACK with acknowledgment 1 (meaning packet 1 is expected next).
  - If packet 1 has arrived safe, the receiver sends an ACK with acknowledgment 0 (meaning packet 0 is expected).

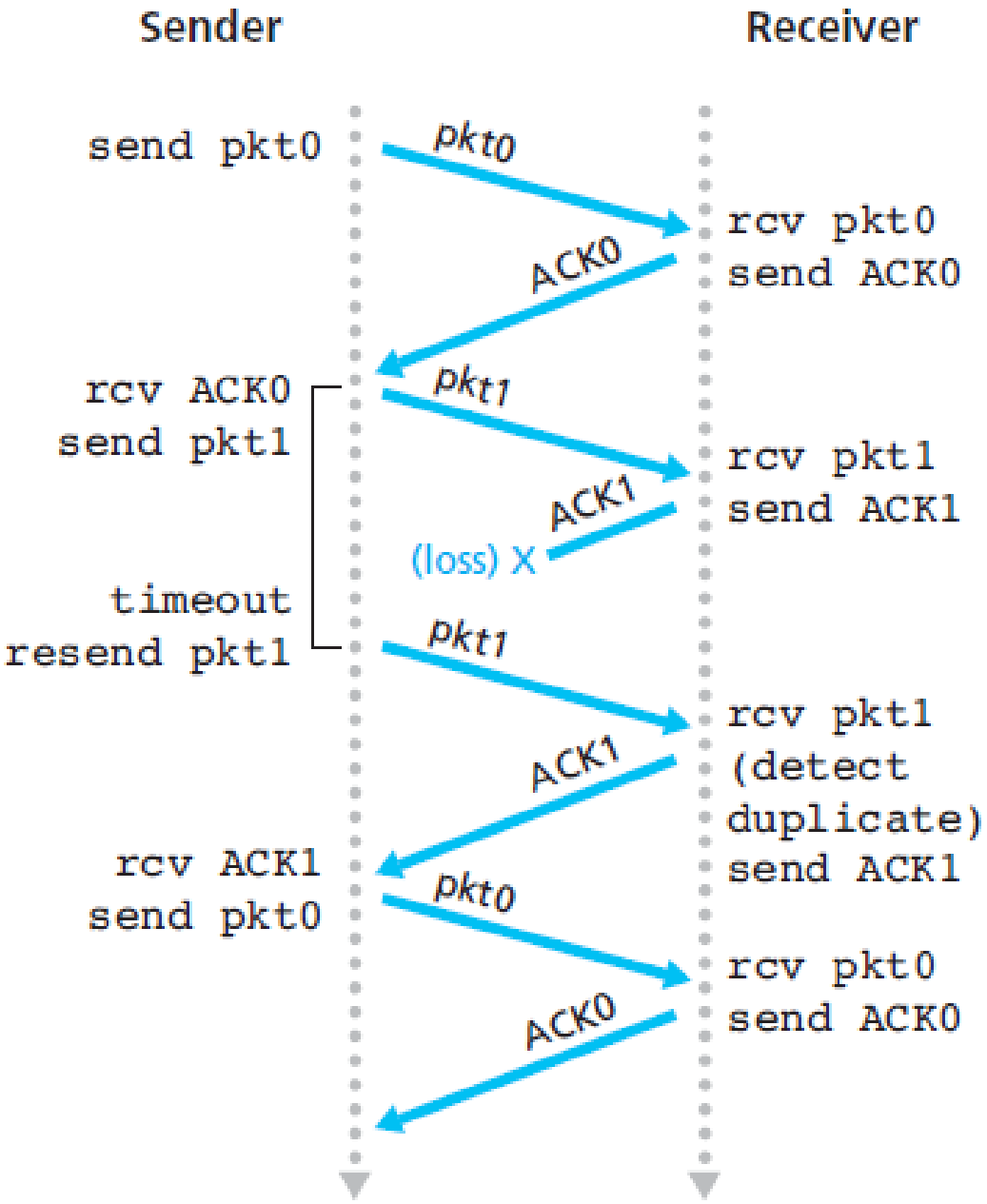
In the Stop-and-Wait protocol, the acknowledgment number always announces, in modulo-2 arithmetic, the sequence number of the next packet expected.



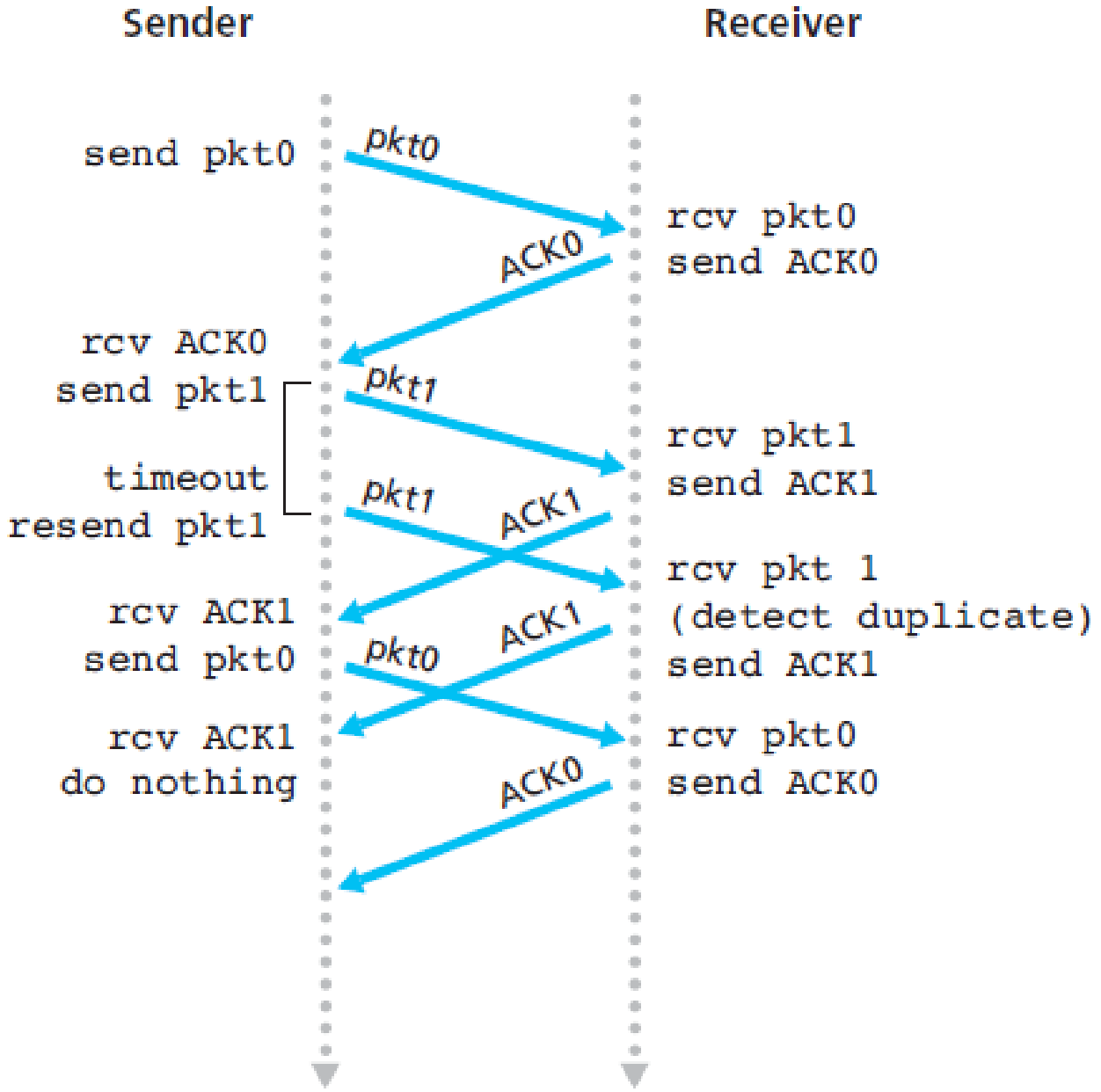
a. Operation with no loss



b. Lost packet

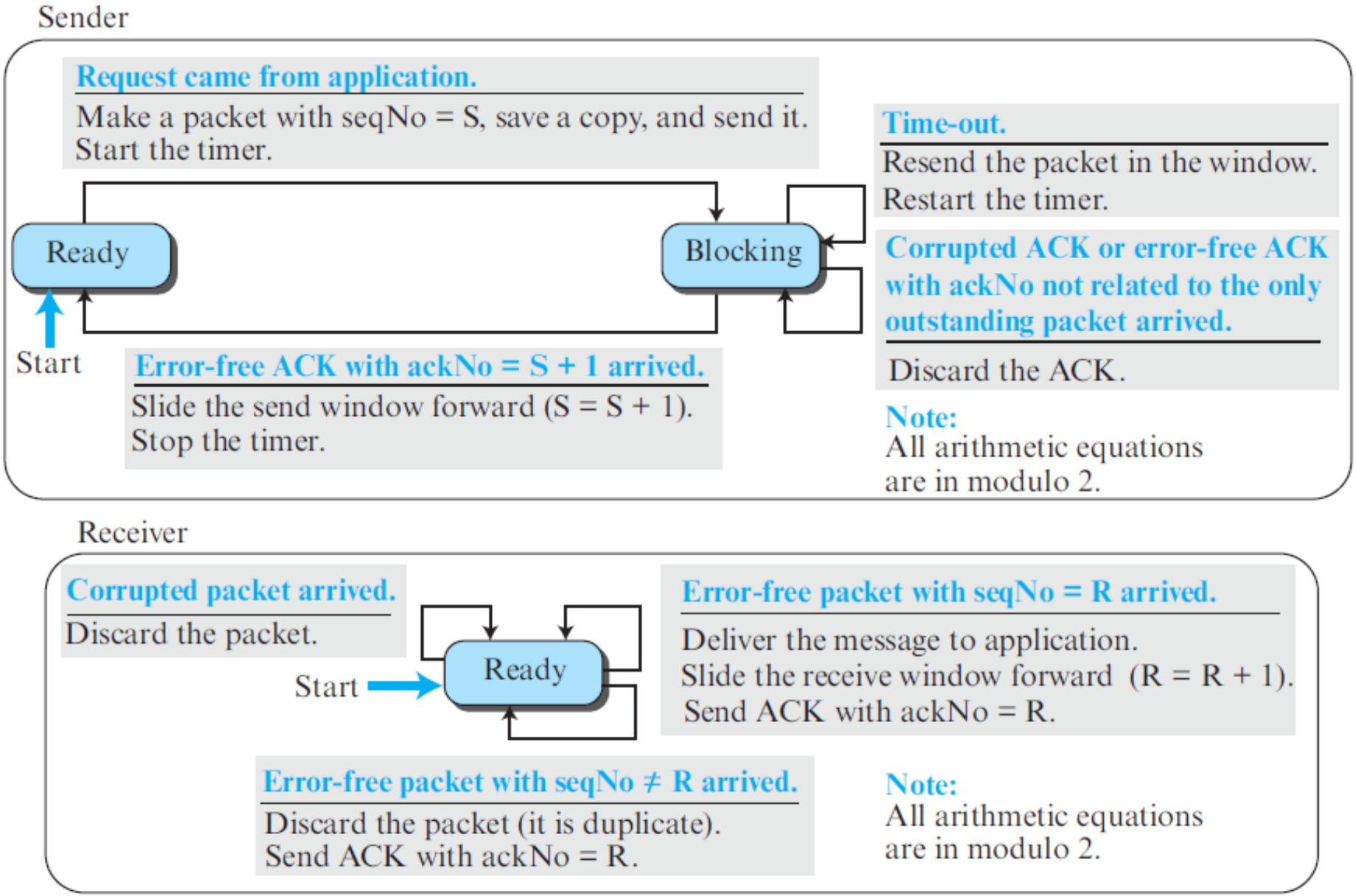

Sender
Receiver
send pkt0
pkt0
rcv pkt0
send ACK0
ACK0
rcv ACK0
send pkt1
pkt1
rcv pkt1
send ACK1
ACK1
(loss) X
timeout
resend pkt1
pkt1
rcv pkt1
(detect
duplicate)
send ACK1
ACK1
rcv ACK1
send pkt0
pkt0
rcv pkt0
send ACK0
ACK0

c. Lost ACK



d. Premature timeout

FSM for Stop-and-Wait protocol

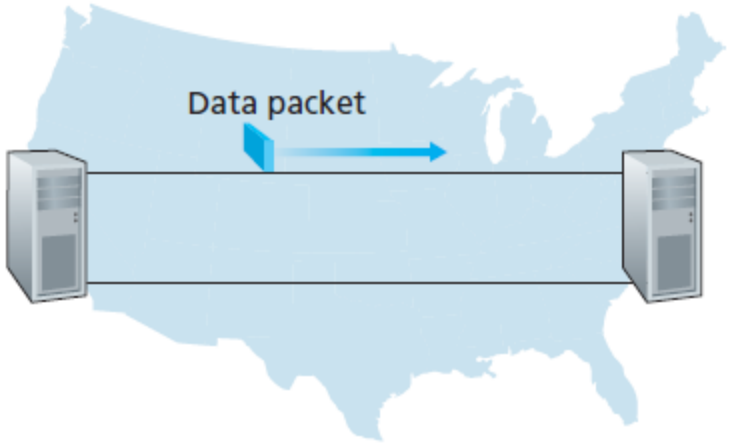


Stop-and-Wait protocol is inefficient!!

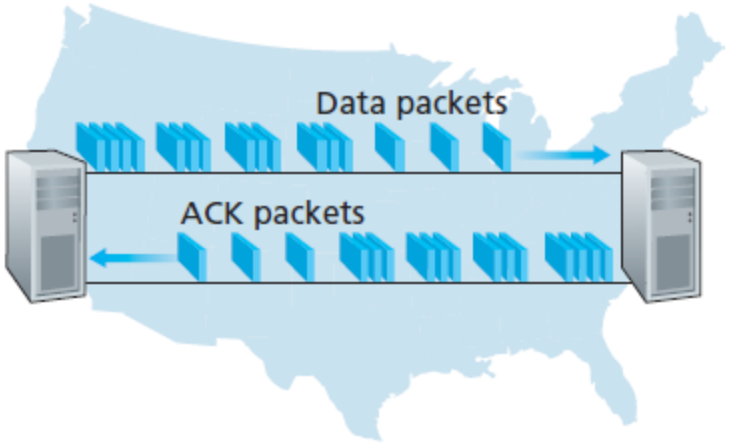
- The Stop-and-Wait protocol is very inefficient if channel is :
  - ***thick***
    - channel has a large bandwidth (high data rate);
  - ***long***
    - round-trip delay is long.
- The product of these two is called the **bandwidth delay product**.
  - A measure of the number of bits a sender can transmit through the system while waiting for an acknowledgment from the receiver.

Pipelined protocols

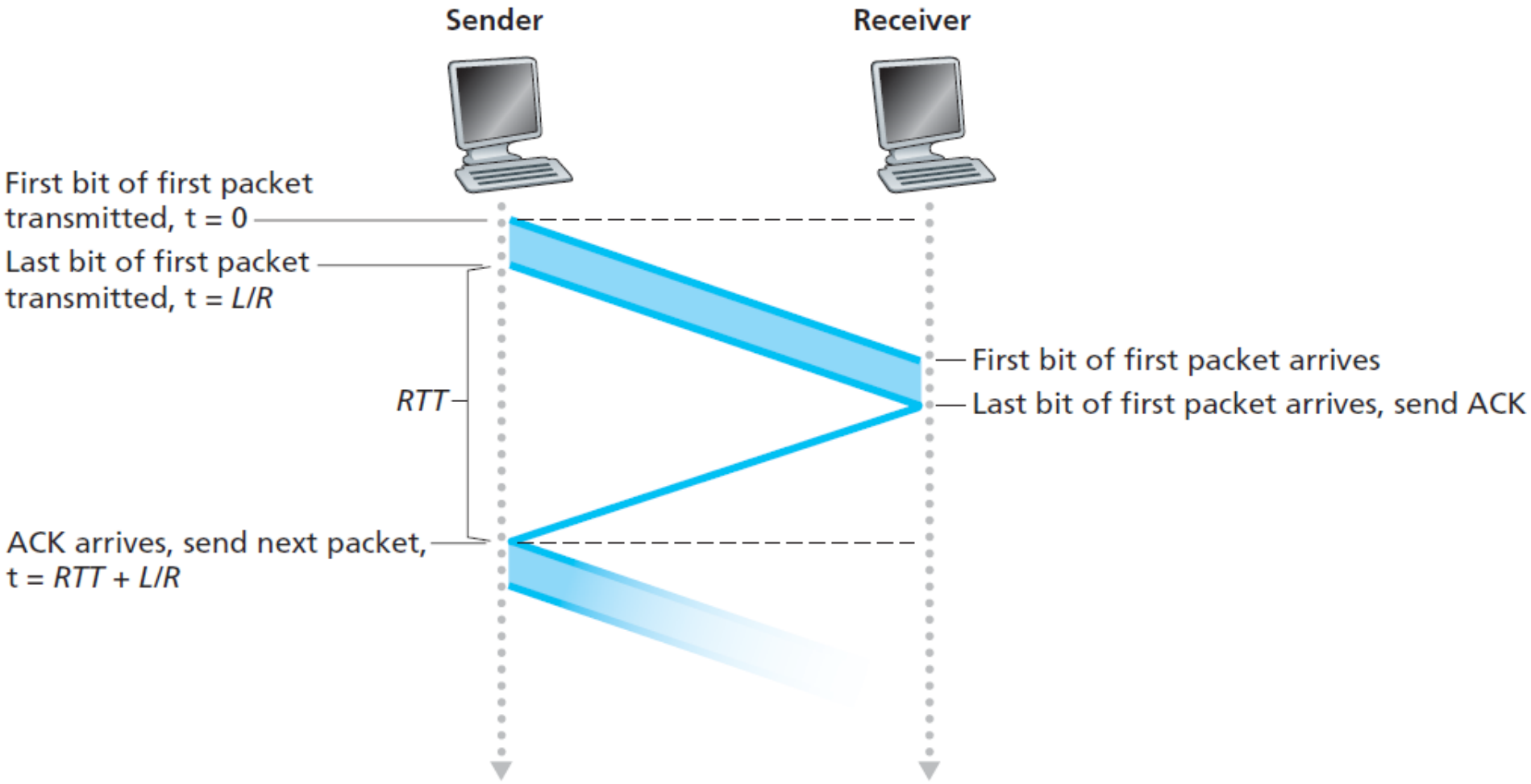
- A task is often begun before the previous task has ended. This is known as **pipelining**.
- Pipelining improves the efficiency of the transmission if the number of bits in transition is large with respect to the bandwidth delay product.
- Two generic forms of pipelined protocols: *Go-Back-N*, Selective Repeat



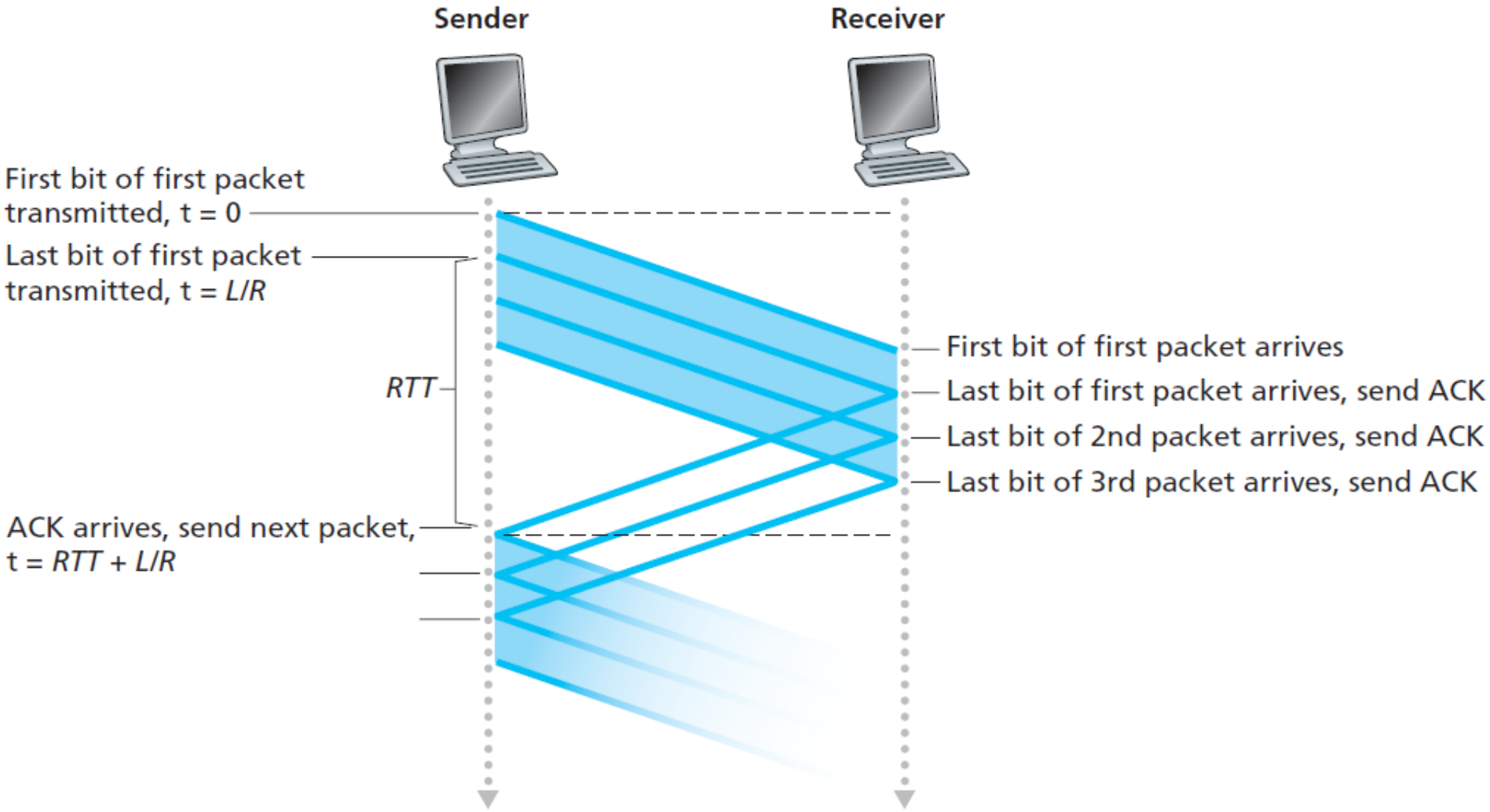
a. A stop-and-wait protocol in operation



b. A pipelined protocol in operation



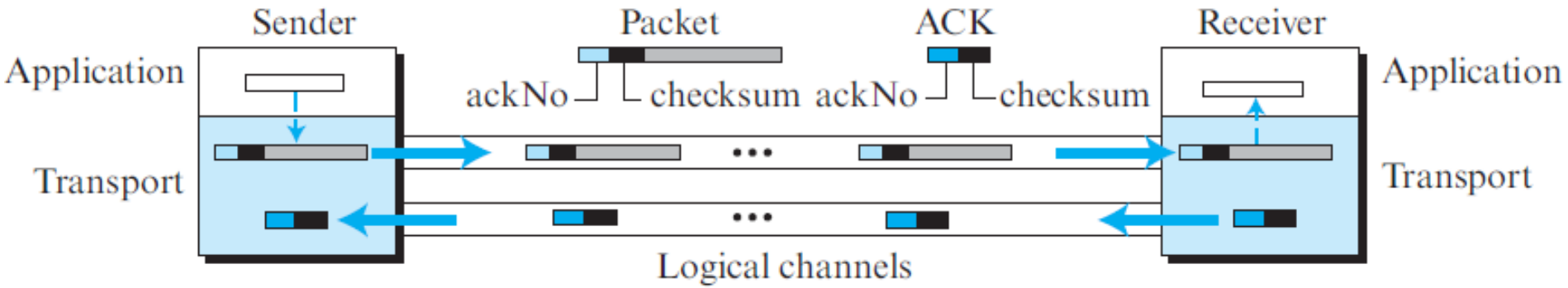
a. Stop-and-wait operation



b. Pipelined operation

Go-Back-N (GBN) Protocol

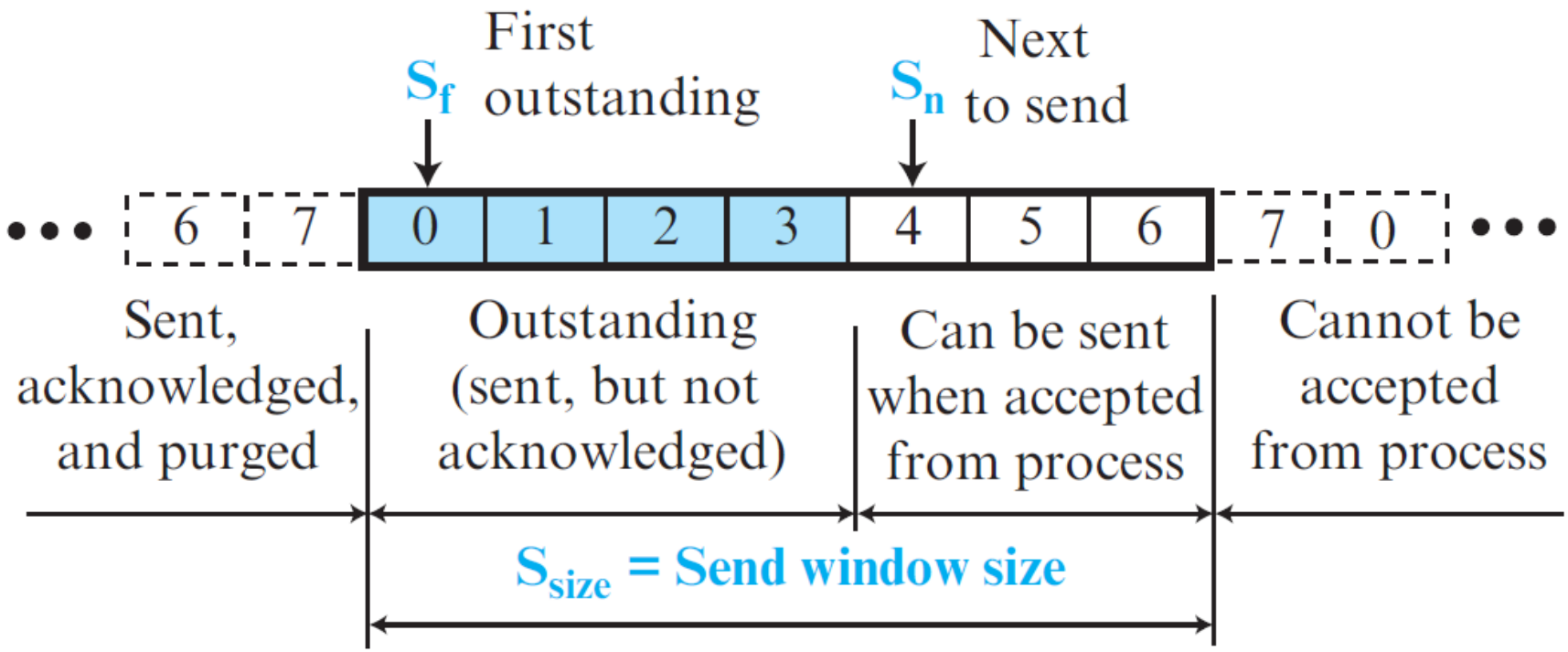
- Sender is allowed to transmit multiple packets without waiting for an acknowledgment.
  - No more than maximum allowable number, *N, of unacknowledged packets* in the pipeline.



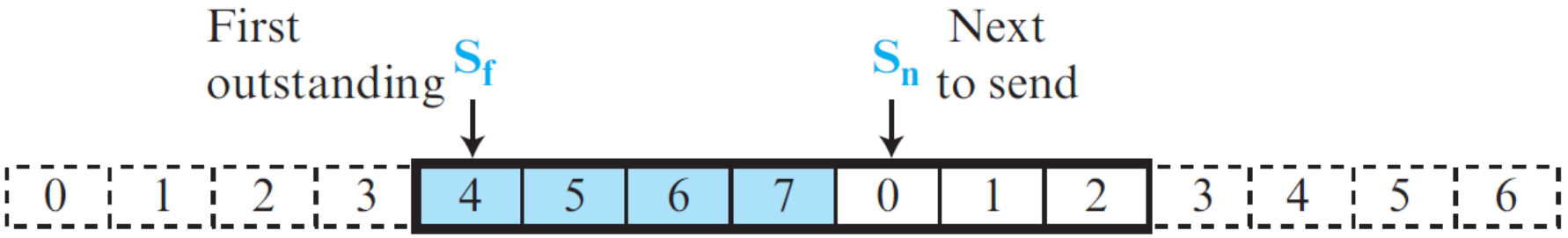
- https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/go-back-n-protocol/index.html

- Sequence Numbers
  - The sequence numbers are modulo 2^m,
    - where ***m*** is the size of the sequence number field in bits.
- Acknowledgment Numbers
  - The acknowledgment number is cumulative and defines the sequence number of the next packet expected to arrive.
  - Example
    - If the acknowledgment number (ackNo) is 7, it means
      - All packets with sequence number up to 6 have arrived
      - The receiver is expecting the packet with sequence number 7.

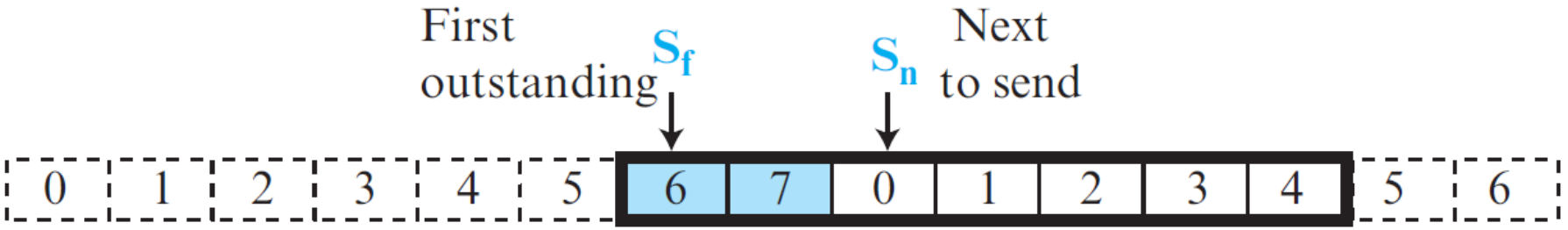
- Send Window
  - The maximum size of the window is $2^m - 1$.
  - The send window at any time divides the possible sequence numbers into four regions.



- Sliding the send window
  - The send window can slide one or more slots to the right when an error-free ACK with ackNo greater than or equal S_f and less than S_n arrives.

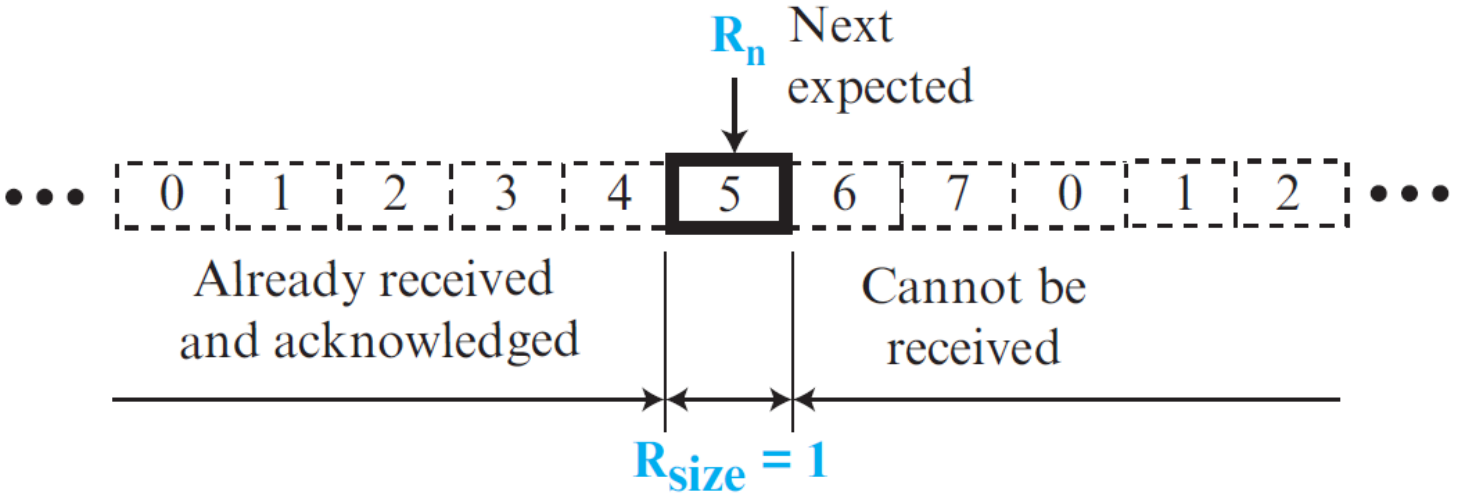


a. Window before sliding



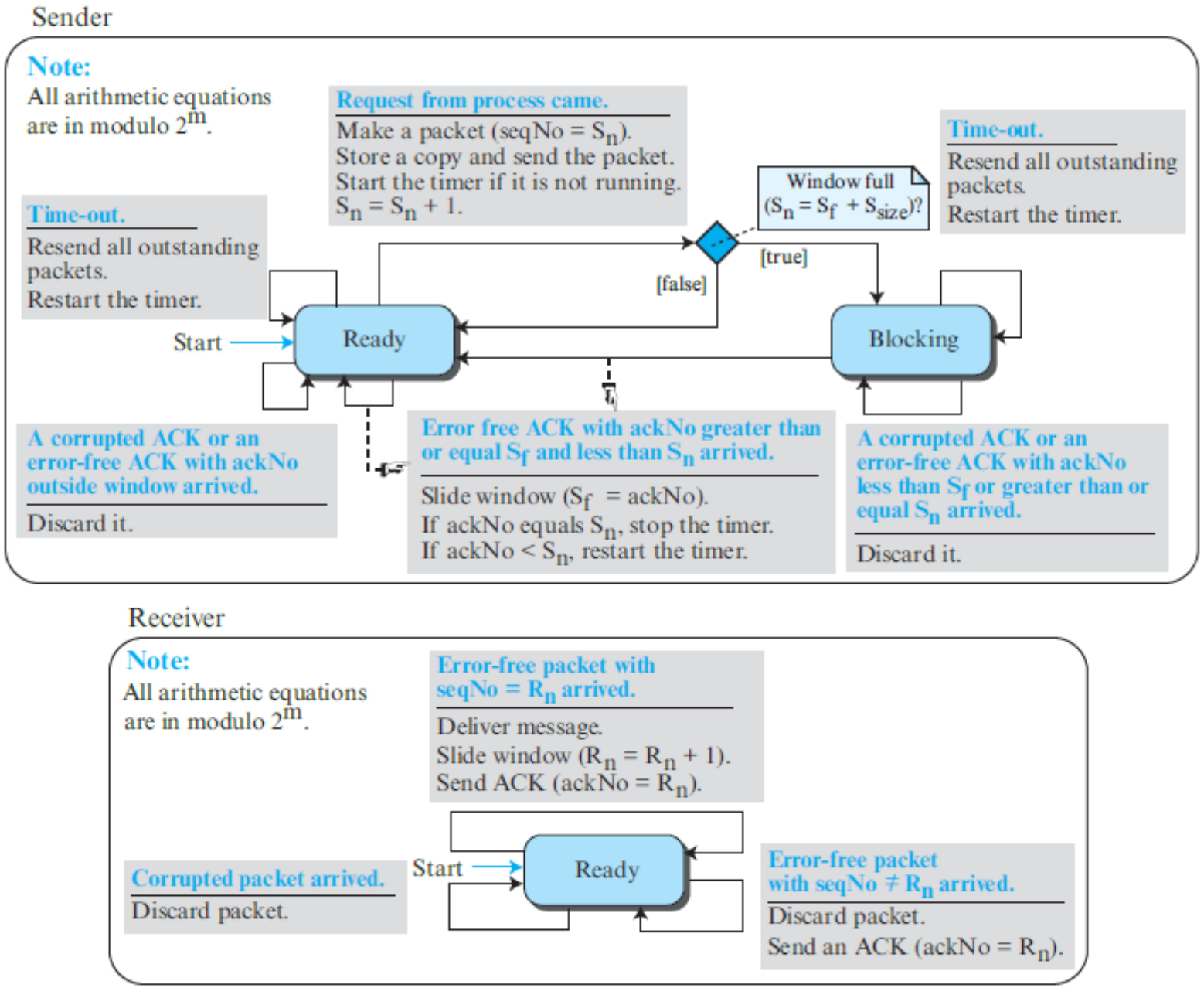
b. Window after sliding (an ACK with ackNo = 6 has arrived)

- Receive Window
  - The receive window makes sure that the correct data packets are received and that the correct acknowledgments are sent.
  - In Go-back-N, the size of the receive window is always 1.
  - The receiver is always looking for the arrival of a specific packet.
  - Any packet arriving out of order is discarded and needs to be resent.



- Timers
  - Although there can be a timer for each packet that is sent, the protocol uses only one.
  - The reason is that the timer for the first outstanding packet always expires first.
  - Resend all outstanding packets when this timer expires.
- Resending packets
  - When the timer expires, the sender resends all outstanding packets.
  - That is why the protocol is called **Go-Back-N.**
    - On a time-out, the machine goes back N locations and resends all packets.

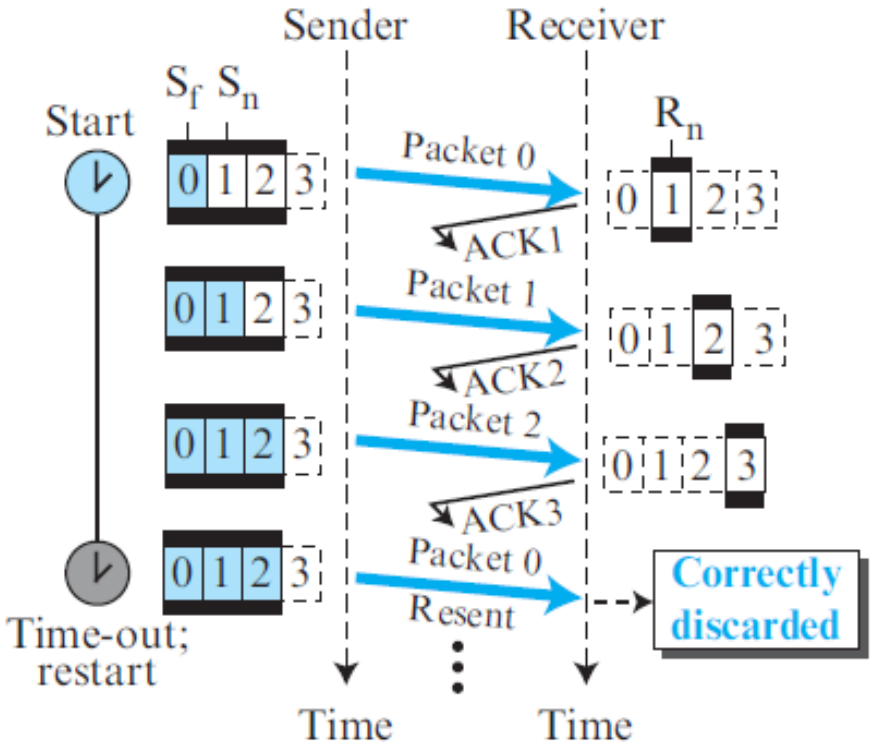
FSMs for the Go-Back-N protocol



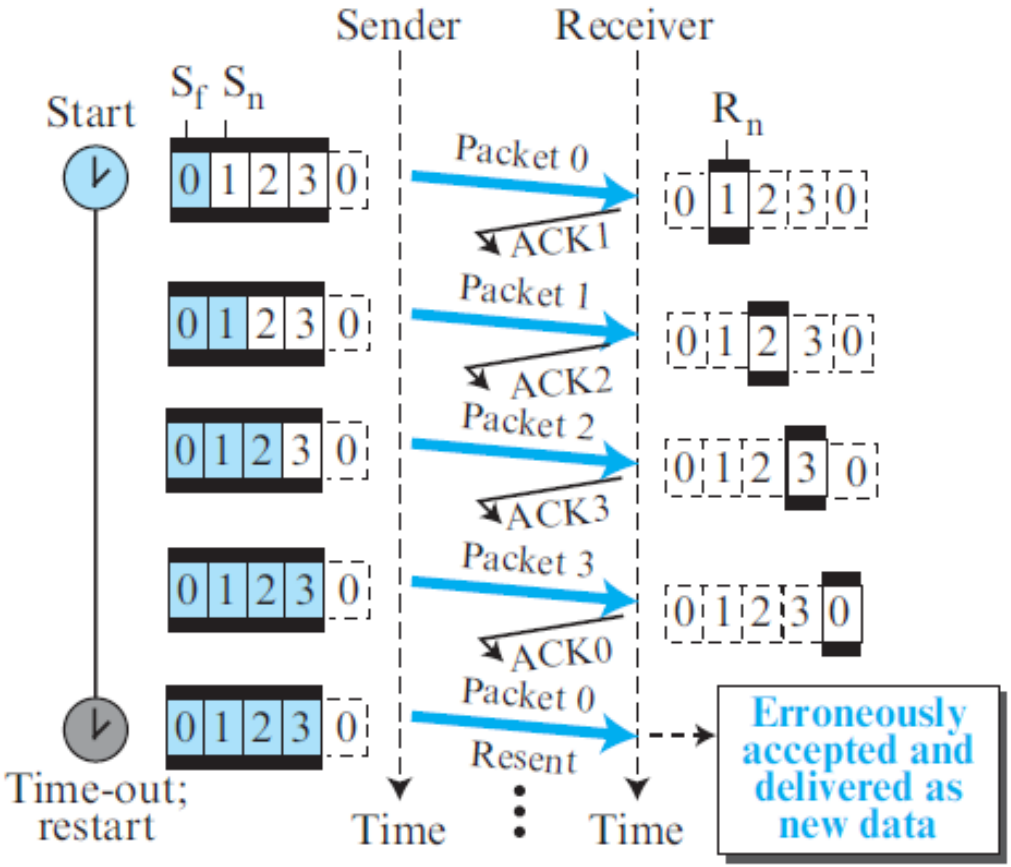
Why the size of the send window must be less than 2^m??

- Consider m=2.
- Case 1 : send window size = 3 ($< 2^m$) and all three acknowledgments are lost.
  - Timer expires and all three packets are resent.
  - The receiver is now expecting packet 3, not packet 0, so the duplicate packet is correctly discarded.
- Case 2 : send window size = 4 ($= 2^m$) and all the acknowledgments are lost.
  - Sender will send a duplicate of packet 0.
  - Window of the receiver expects to receive packet 0 (in the next cycle), so it accepts packet 0, not as a duplicate, but as the first packet in the next cycle. This is an error.
  - **Therefore, the size of the send window must be less than 2^m.**

Why the size of the send window must be less than 2^m??



a. Send window of size < 2^m



b. Send window of size = 2^m

Connection-Oriented Transport: TCP

Transmission Control Protocol (TCP)

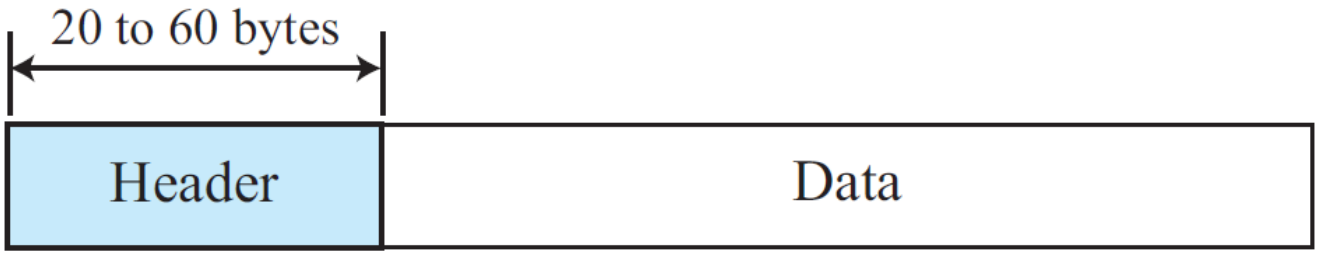
- A connection-oriented, reliable protocol.
  - TCP defines connection establishment, data transfer, and connection teardown phases to provide a connection-oriented service.
  - TCP uses a combination of GBN and SR protocols to provide reliability.
  - TCP uses
    - Checksum (for error detection),
    - Retransmission of lost or corrupted packets,
    - Cumulative and selective acknowledgments, and
    - Timers.

TCP: Overview

- Point-to-point:
  - One sender, one receiver
- Reliable, in-order *byte stream:*
  - No "message boundaries"
- Pipelined:
  - TCP congestion and flow control set window size
- Full duplex data:
  - Bi-directional data flow in same connection
  - MSS: maximum segment size
- Connection-oriented:
  - Handshaking initiates sender, receiver state before data exchange
- Flow controlled:
  - Sender will not overwhelm receiver

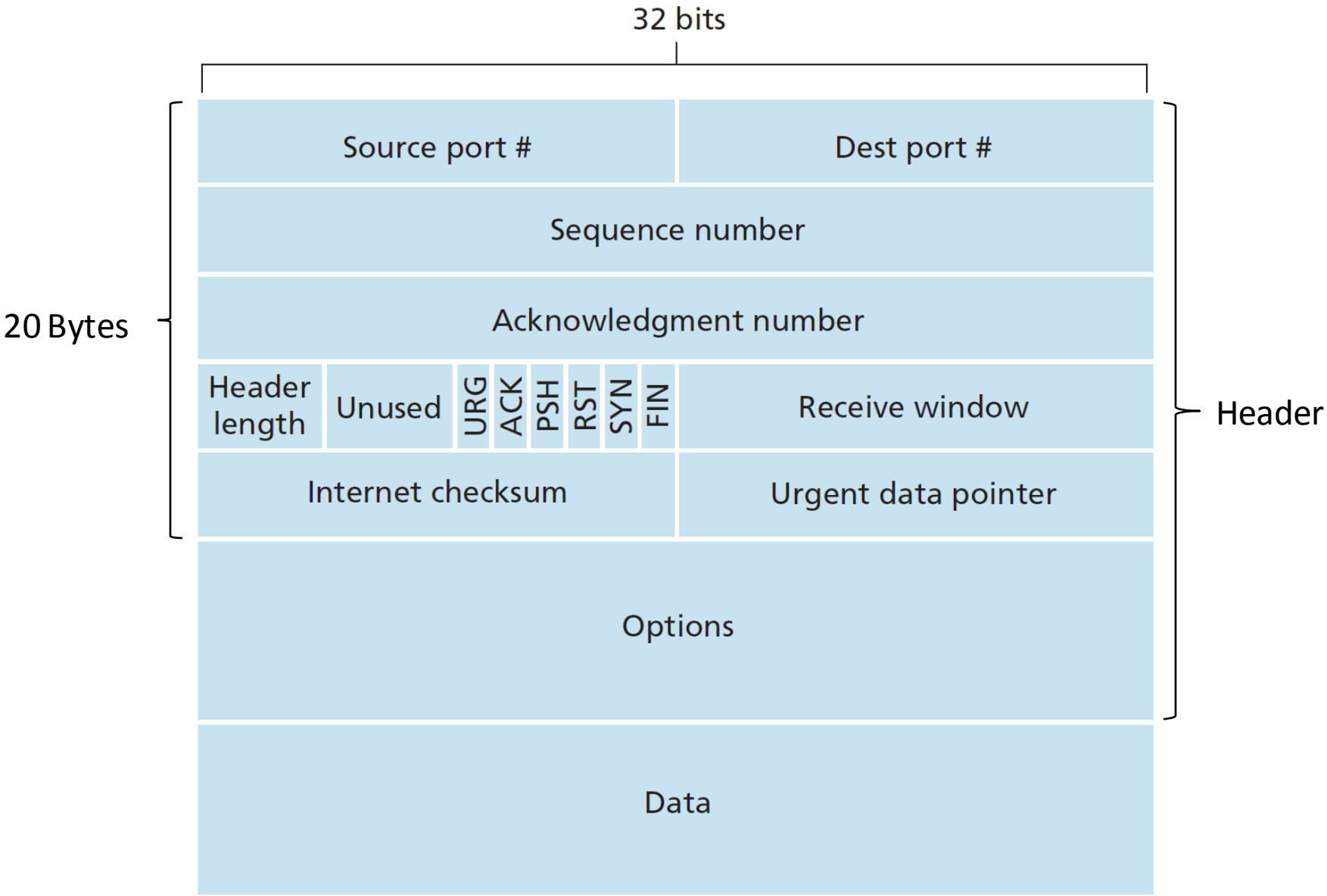
TCP Segment

- A packet in TCP is called a **segment**.
- The segment consists of a header of 20 to 60 bytes, followed by data from the application program.



  - The header is 20 bytes if there are no options and up to 60 bytes if it contains options.

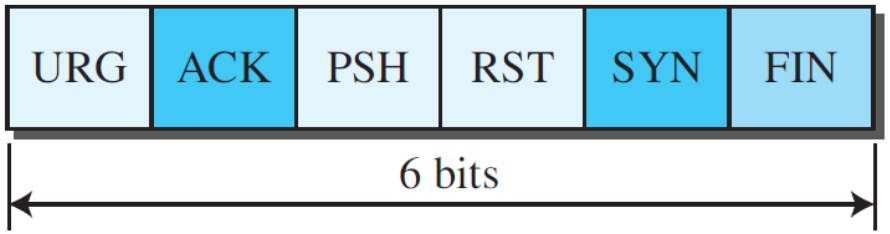
TCP Segment Structure



- **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.
  - During connection establishment each party uses a random number generator to create an **Initial Sequence Number (ISN)**, which is usually different in each direction.

- **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.
  - The length of the header can be between 20 and 60 bytes.
  - Therefore, the value of this field is always between 5 (5 × 4 = 20) and 15 (15 × 4 = 60).

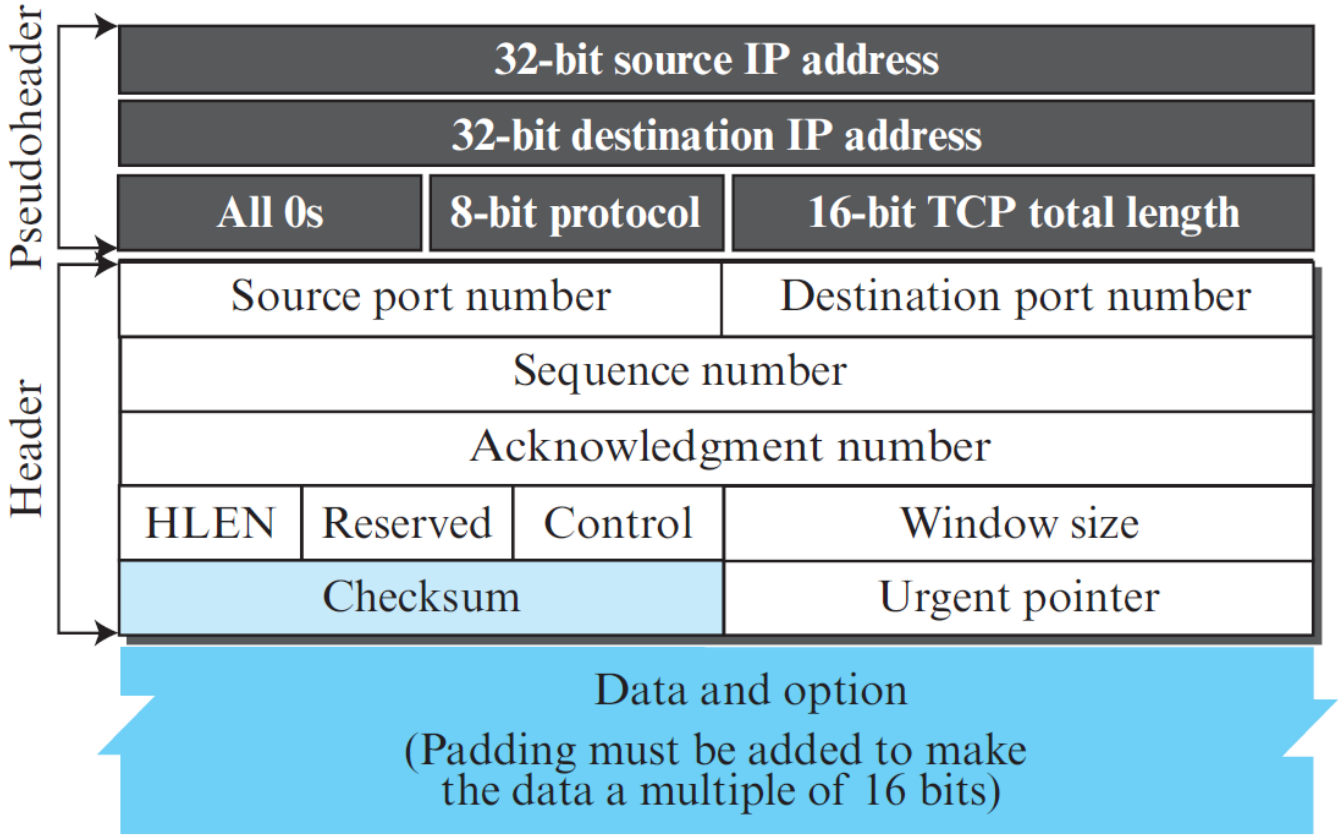
- **Flags**.
  - These bits enable flow control, connection establishment and termination, connection abortion, and the mode of data transfer in TCP.



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

- **Window size.**
  - This field defines the window size of the sending TCP in bytes.
  - The maximum size of the window is 65,535 bytes.
  - 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
  - A pseudoheader is added to the segment.



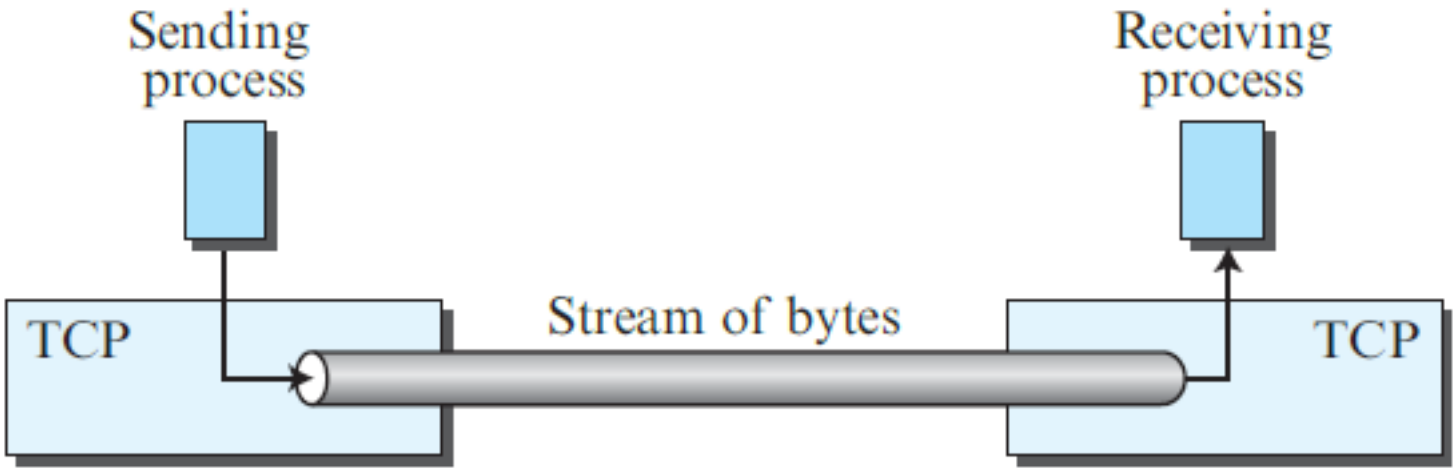
Procedure to calculate the checksum

Sender	*Receiver*
1. The message is divided into 16-bit words.	1. The message and the checksum is received.
2. The value of the checksum word is initially set to zero.	2. The message is divided into 16-bit words.
3. All words including the checksum are added using one's complement addition.	3. All words are added using one's complement addition.
4. The sum is complemented and becomes the checksum.	4. The sum is complemented and becomes the new checksum.
5. The checksum is sent with the data.	5. If the value of the checksum is 0, the message is accepted; otherwise, it is rejected.

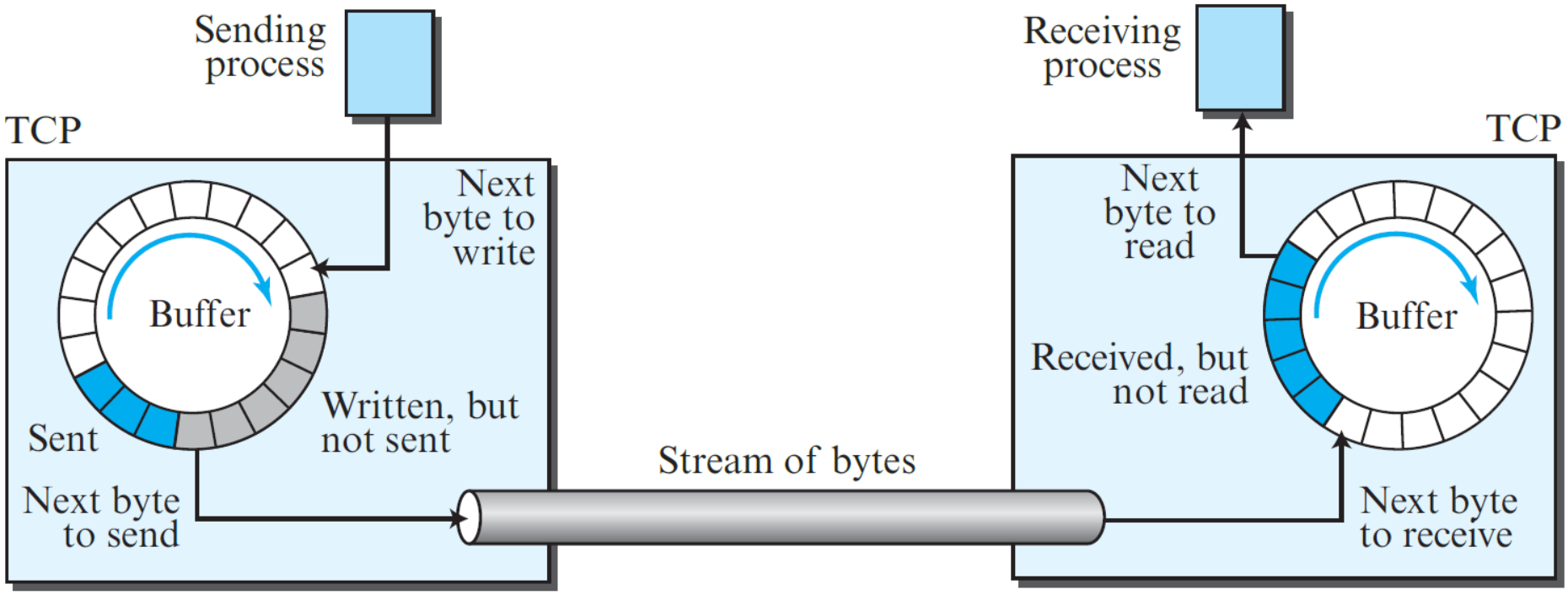
- **Urgent pointer.**
  - This 16-bit field is used when the segment contains urgent data.
  - It is valid only if the urgent flag is set.
  - The location of the last byte of the urgent data is indicated by the 16-bit **urgent data pointer field.**
- **Options.**
  - There can be up to 40 bytes of optional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

Stream delivery in TCP

- TCP is a stream-oriented protocol.
  - allows the sending process to deliver data as a stream of bytes and
  - allows the receiving process to obtain data as a stream of bytes.
- The sending process produces (writes to) the stream and the receiving process consumes (reads from) it.

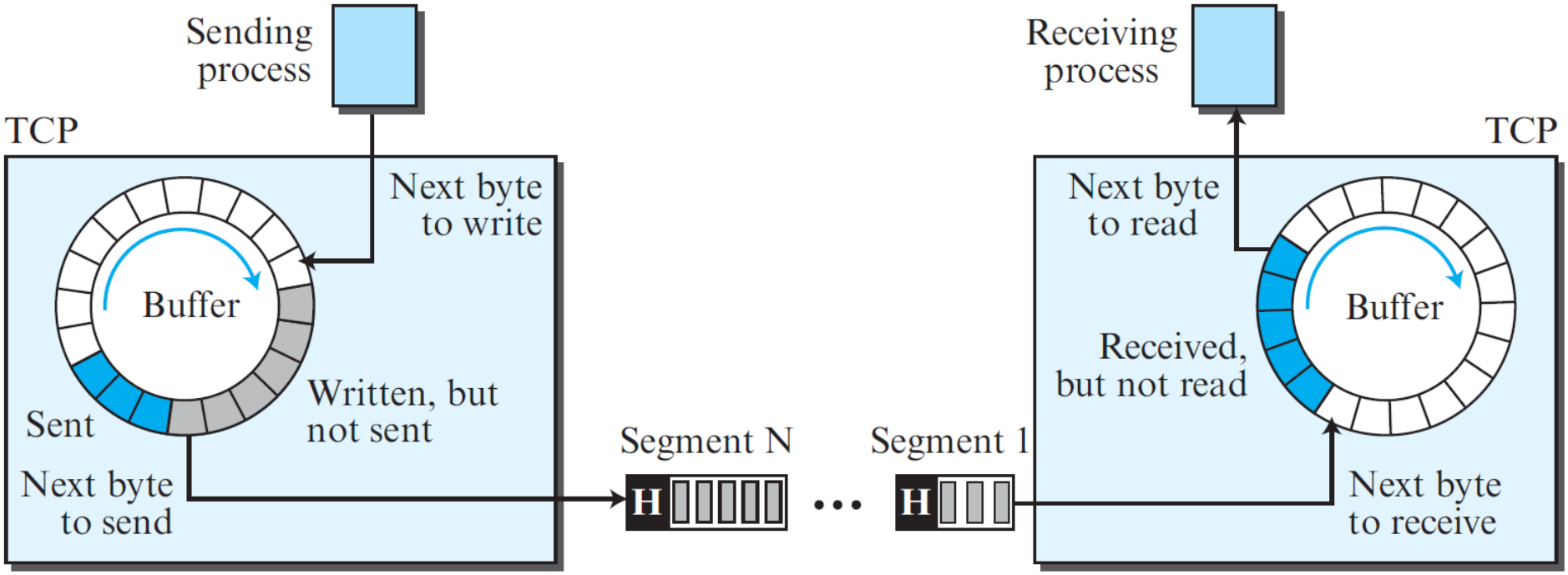


Sending and Receiving Buffers



How TCP Segments generated?

- TCP groups a number of bytes together into a packet called a segment.
- TCP adds a header to each segment.
- Segments are not necessarily all the same size



Numbering System

- Byte Number
  - TCP numbers all data bytes that are transmitted in a connection.
  - The numbering does not necessarily start from 0.
  - TCP chooses an arbitrary number between 0 and $2^{32} - 1$ for the number of the first byte.
- Sequence Number
  - TCP assigns a sequence number to each segment that is being sent.
  - The sequence number of the first segment is the ISN (initial sequence number), a random number.
  - The sequence number of any other segment is the sequence number of the previous segment plus the number of carried by the previous segment.
- Acknowledgment Number
  - Defines the number of the next byte that the party expects to receive.
  - Cumulative : number of the last byte that it has received + 1.

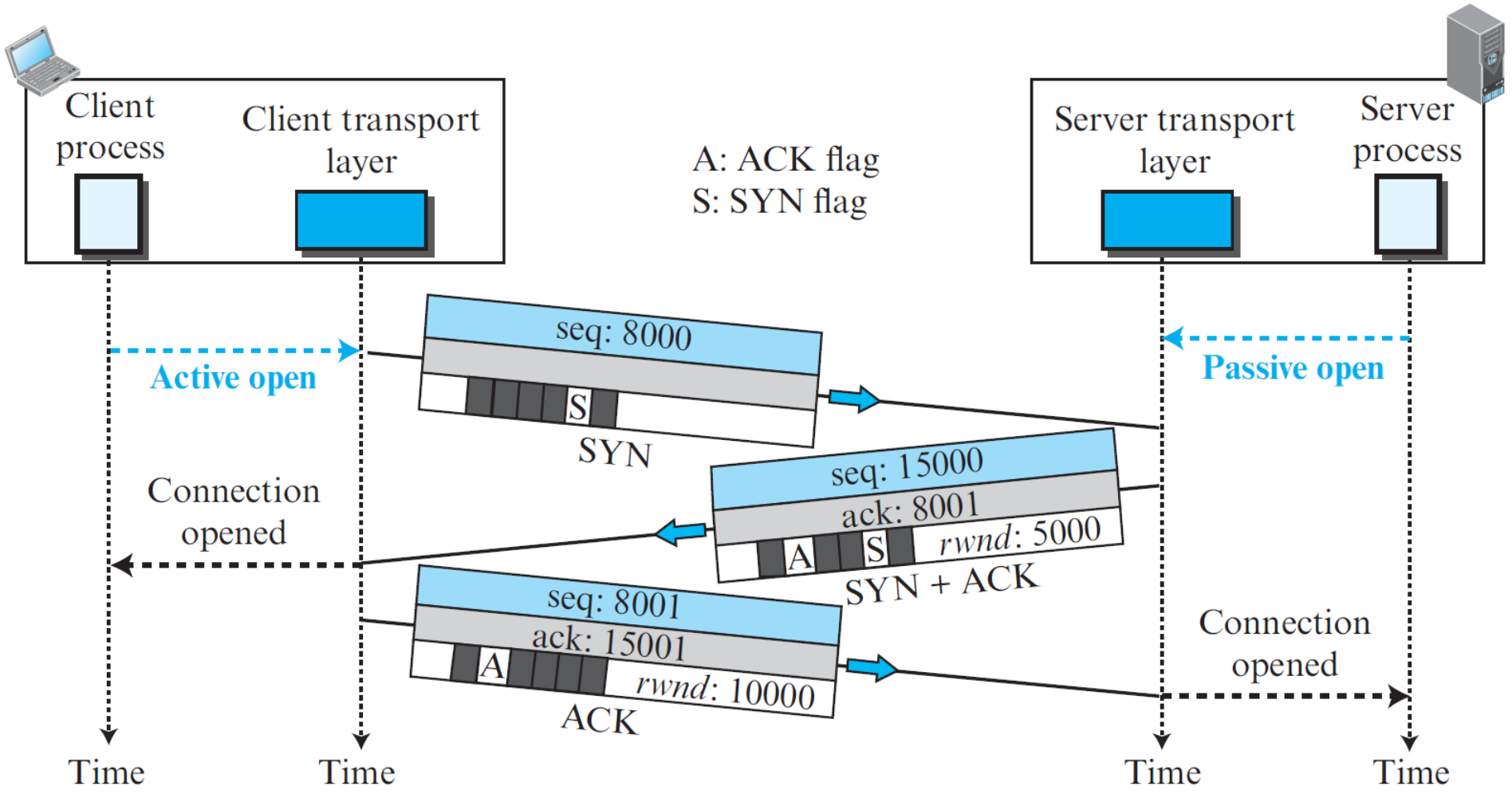
TCP Connection

- Establishes a logical path between the source and destination.
- All of the segments belonging to a message are then sent over this logical path.
- Three phases:
  - Connection establishment,
  - Data transfer, and
  - Connection termination.

Connection Establishment

- TCP transmits data in full-duplex mode.
  - When two TCPs in two machines are connected, they are able to send segments to each other simultaneously.
  - 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.**

Connection establishment using three-way handshaking

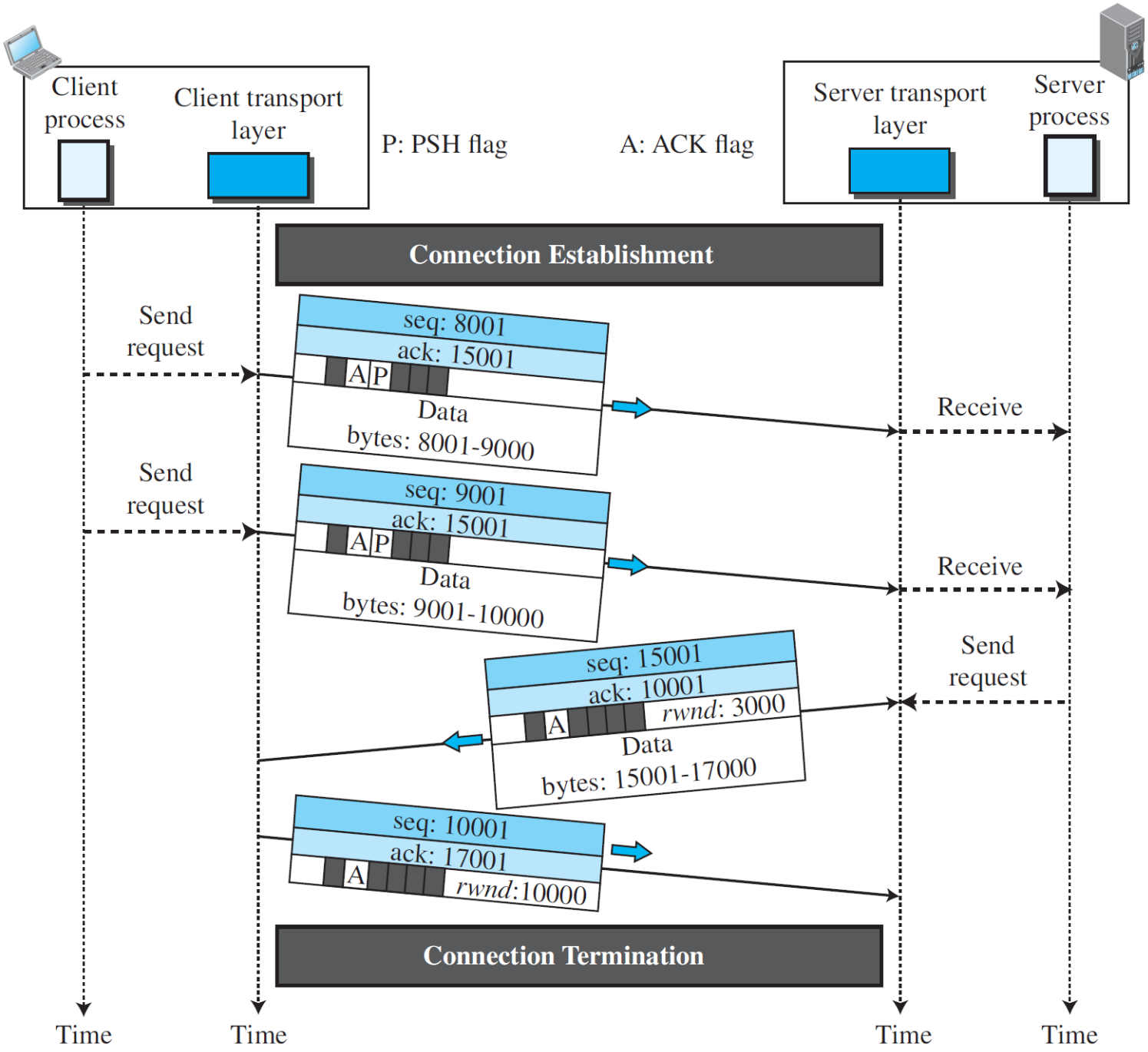


Connection Establishment...

- A SYN segment cannot carry data, but it consumes **one** sequence number.
- A SYN + ACK segment cannot carry data, but it does consume **one** sequence number.
- An ACK segment, if carrying no data, consumes **no** sequence number.

Data Transfer

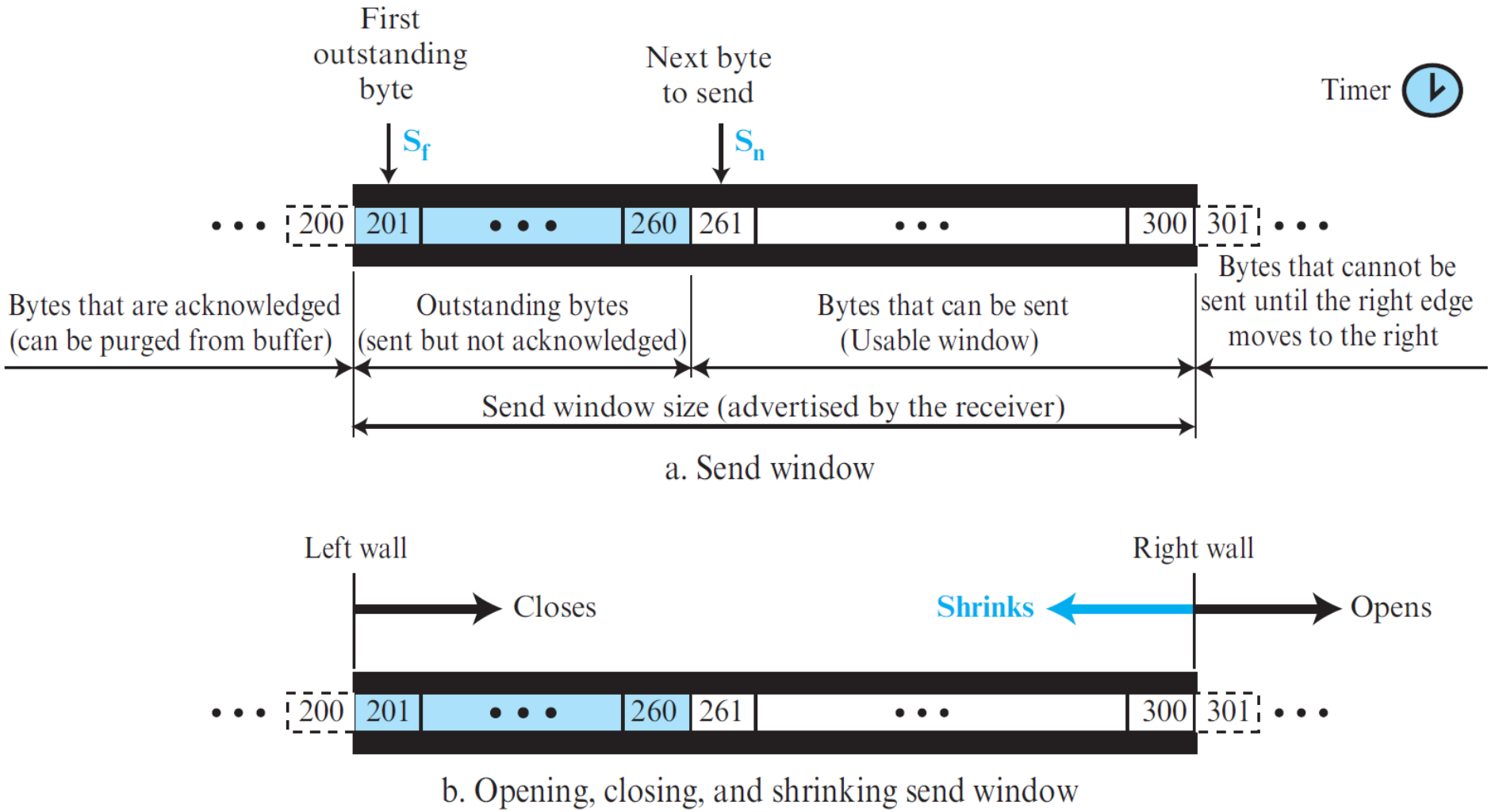
- After connection is established, bidirectional data transfer can take place.
- The client and server can send data and acknowledgments in both directions.
- Setting the **PSH** bit indicates that the receiver should pass the data to the upper layer immediately.
- The **URG** bit is used to indicate that there is data in this segment that the sending-side upper-layer entity has marked as “urgent.”


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
rwnd: 3000
A
Data
bytes: 15001-17000
seq: 10001
ack: 17001
A
rwnd:10000
Connection Termination
Time
Time
Time
Time

- **Piggybacking**
  - Method of attaching acknowledgment to the outgoing data packet.
  - Used to improve the efficiency of the bidirectional data transfer.
  - Example
    - When a packet is carrying data from A to B, it can also carry acknowledgment feedback about arrived packets from B and vice versa.

- **Cumulative Acknowledgments**
  - TCP only acknowledges bytes up to the first missing byte in the stream, so TCP is said to provide **cumulative acknowledgments.**
  - Example
    - Suppose that Host A has received one segment from Host B containing bytes 0-535 and another segment containing bytes 900-1,000.
    - For some reason Host A has not yet received bytes from 536-899.
    - Host A is still waiting for byte 536 (and beyond) in order to re-create B's data stream.
    - Thus, A's next segment to B will contain 536 in the acknowledgment number field.

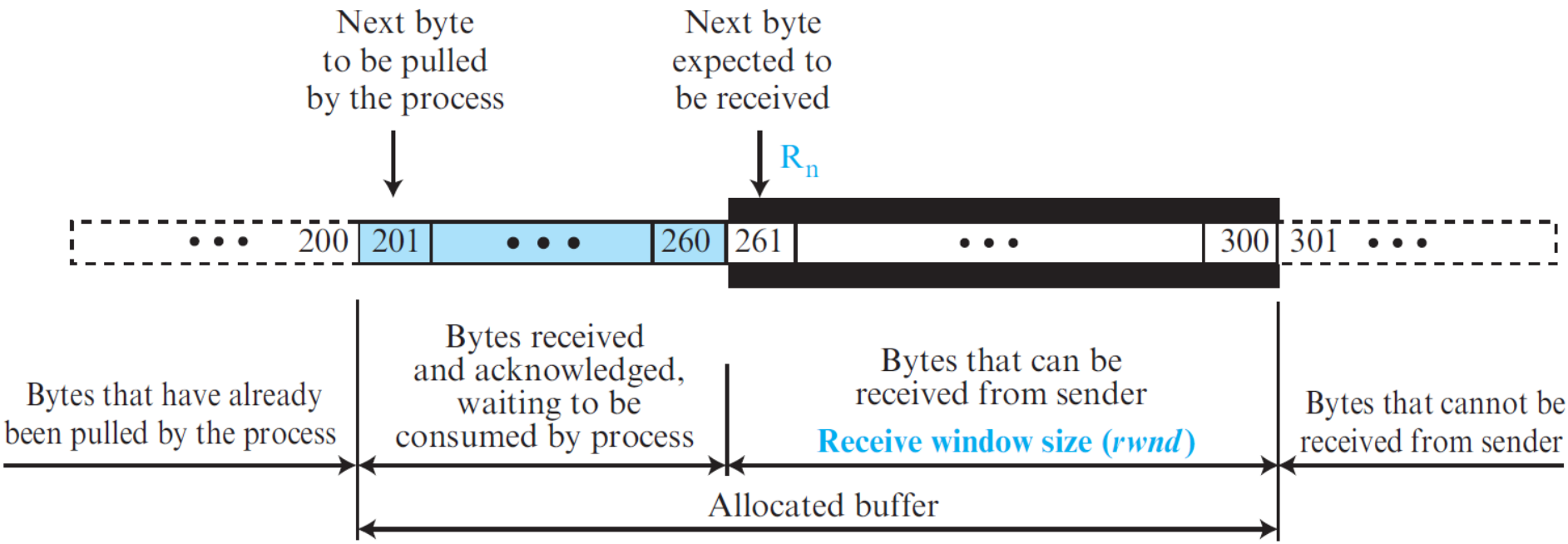
- **Send Window in TCP**



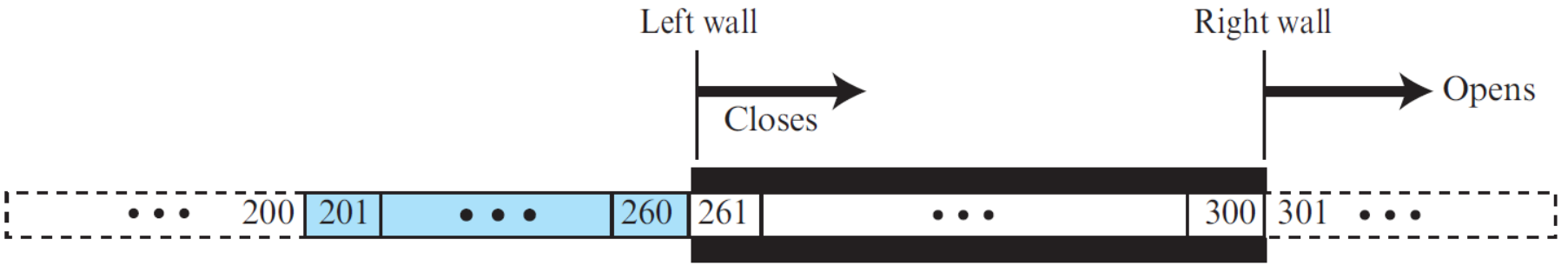
- **Receive Window**
  - The receive window size is always smaller or equal to the receiver buffer size.
  - The receive window size determines the number of bytes that the receive window can accept from the sender.
  - The receive window size, normally called **rwnd**, can be determined as:

rwnd = buffer size – number of waiting bytes to be pulled

• Receive Window in TCP



a. Receive window and allocated buffer

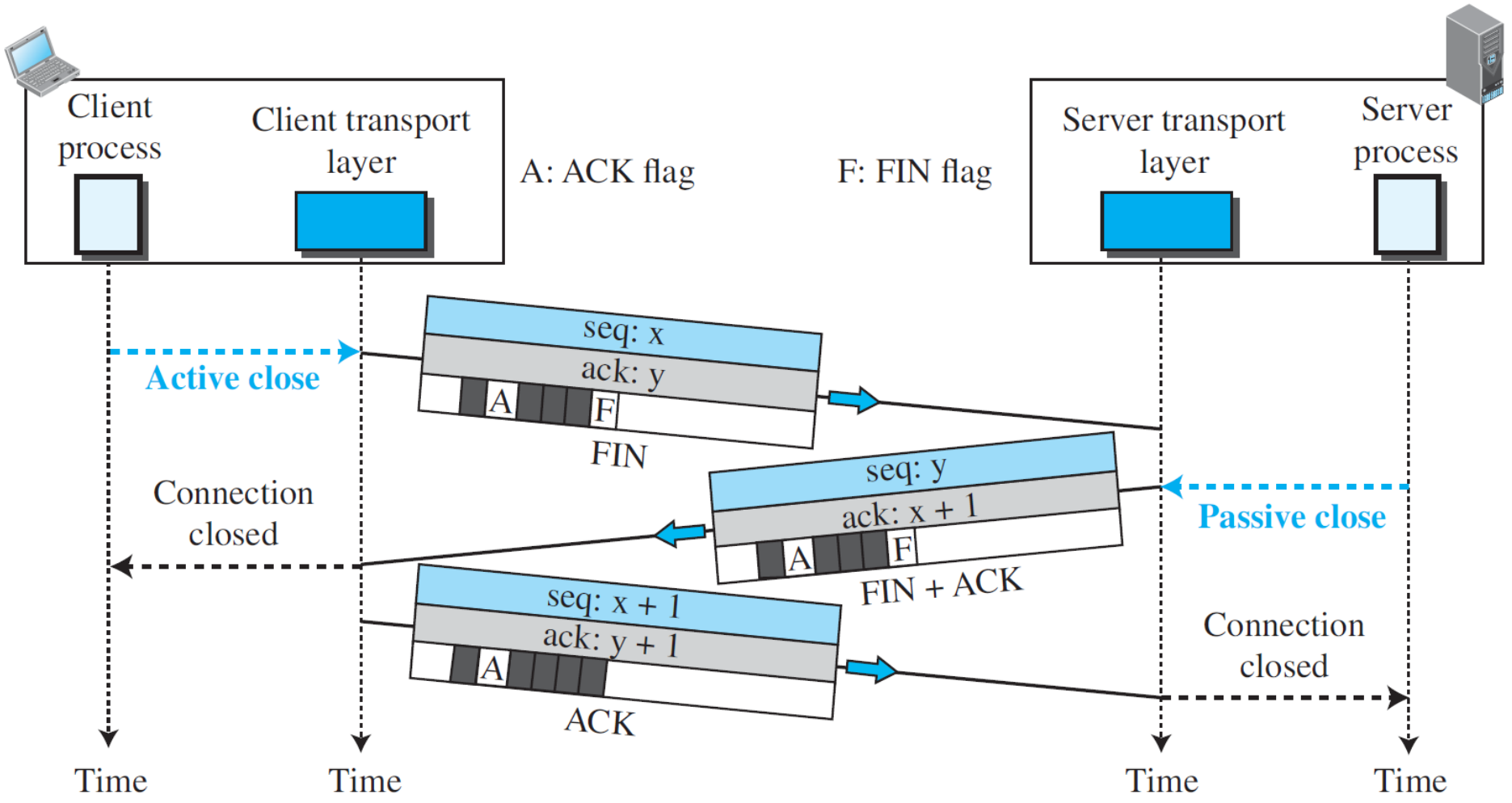


b. Opening and closing of receive window

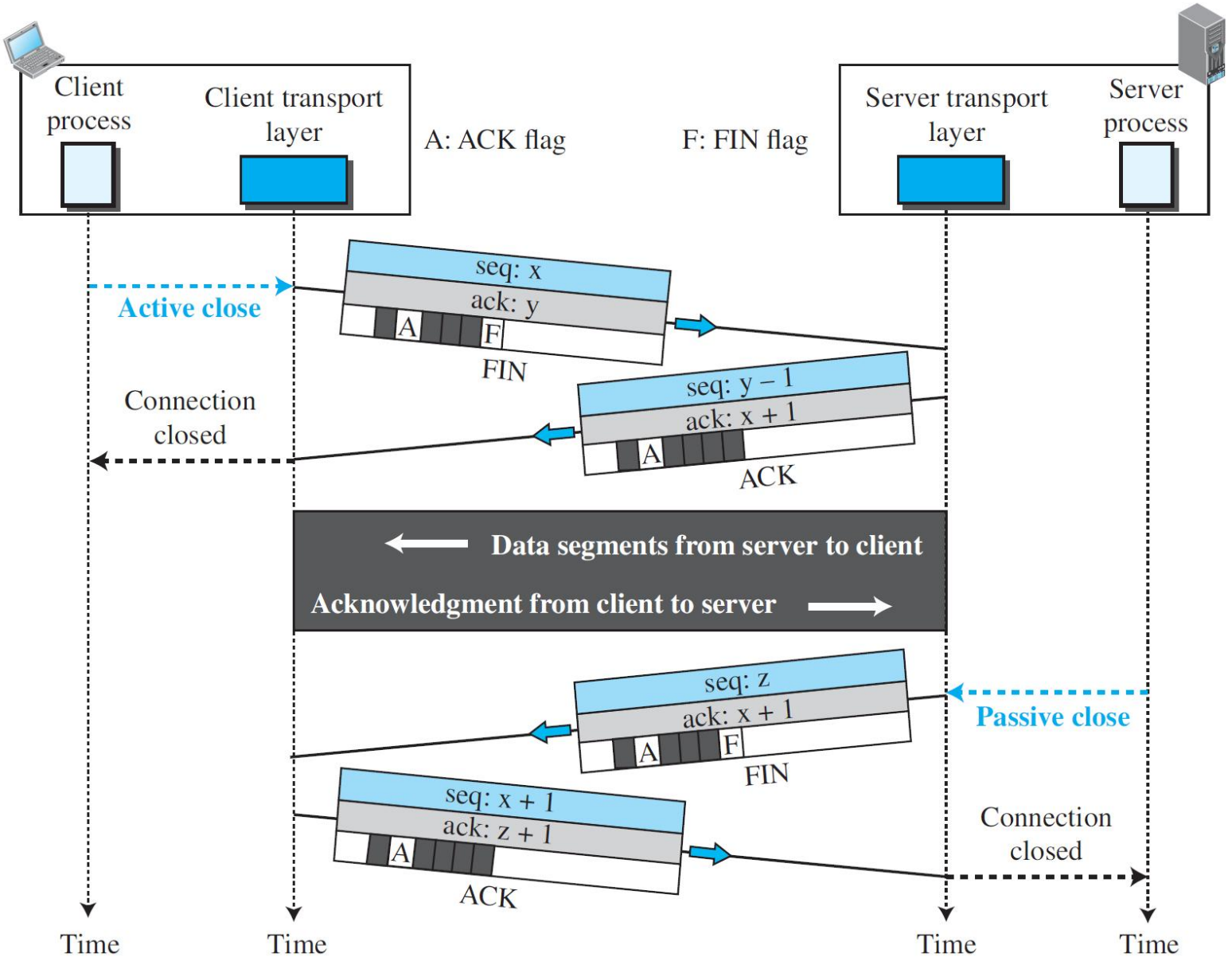
Connection Termination

- Either of the two parties involved in exchanging data (client or server) can close the connection.
- It is usually initiated by the client.
- Two options for connection termination:
  - Three-way handshaking and
  - Four-way handshaking with a half-close option.

Connection termination using three-way handshaking



Four-way handshaking with a half-close option



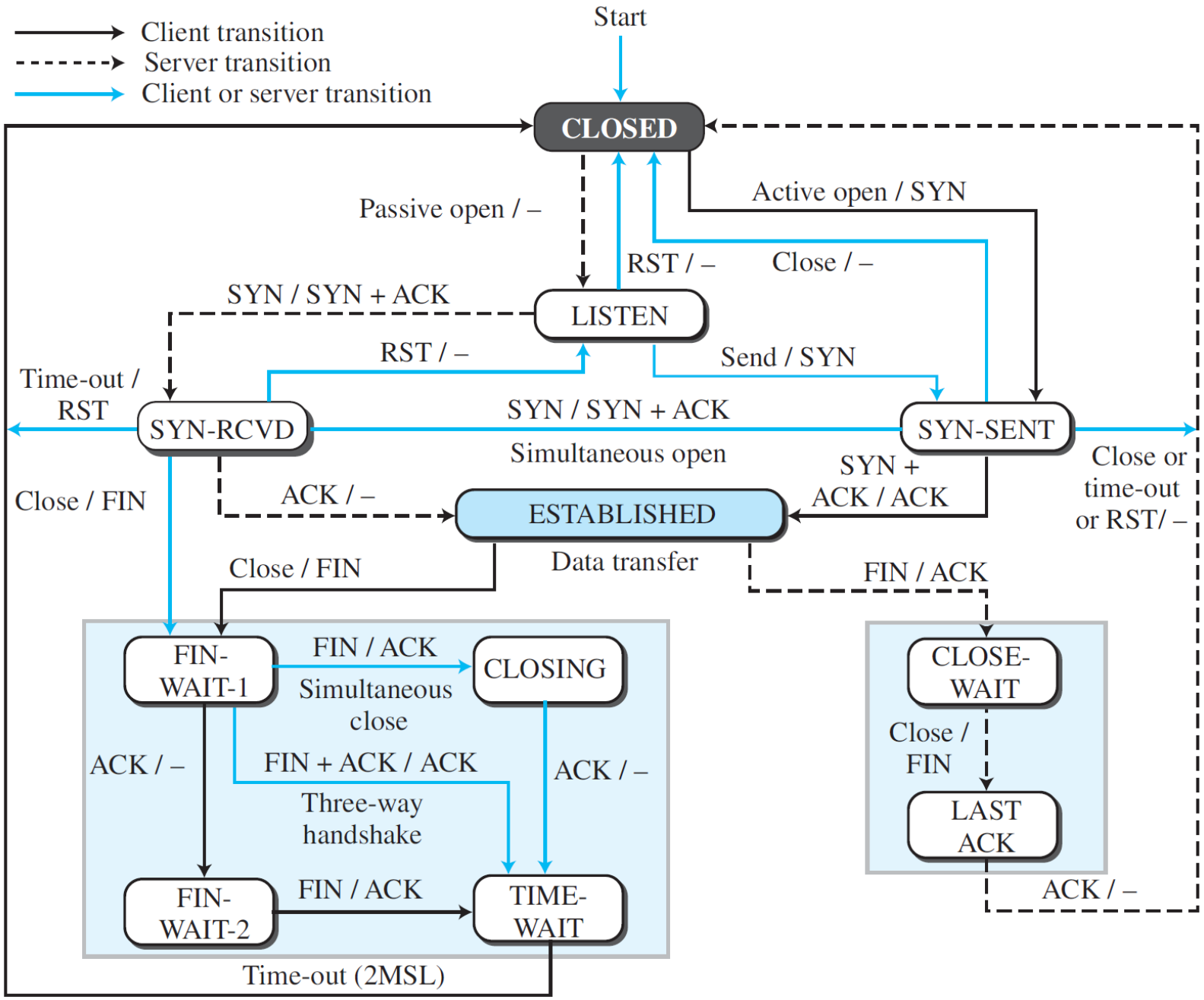
Connection Termination…

- The FIN segment consumes one sequence number if it does not carry data.
- The FIN + ACK segment consumes only one sequence number if it does not carry data.
- The ACK segment cannot carry data and consumes no sequence numbers.
- **Half-Close :**
  - Stop sending data while still receiving data.
  - Either the server or the client can issue a half-close request.

Connection Reset

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

State Transition Diagram

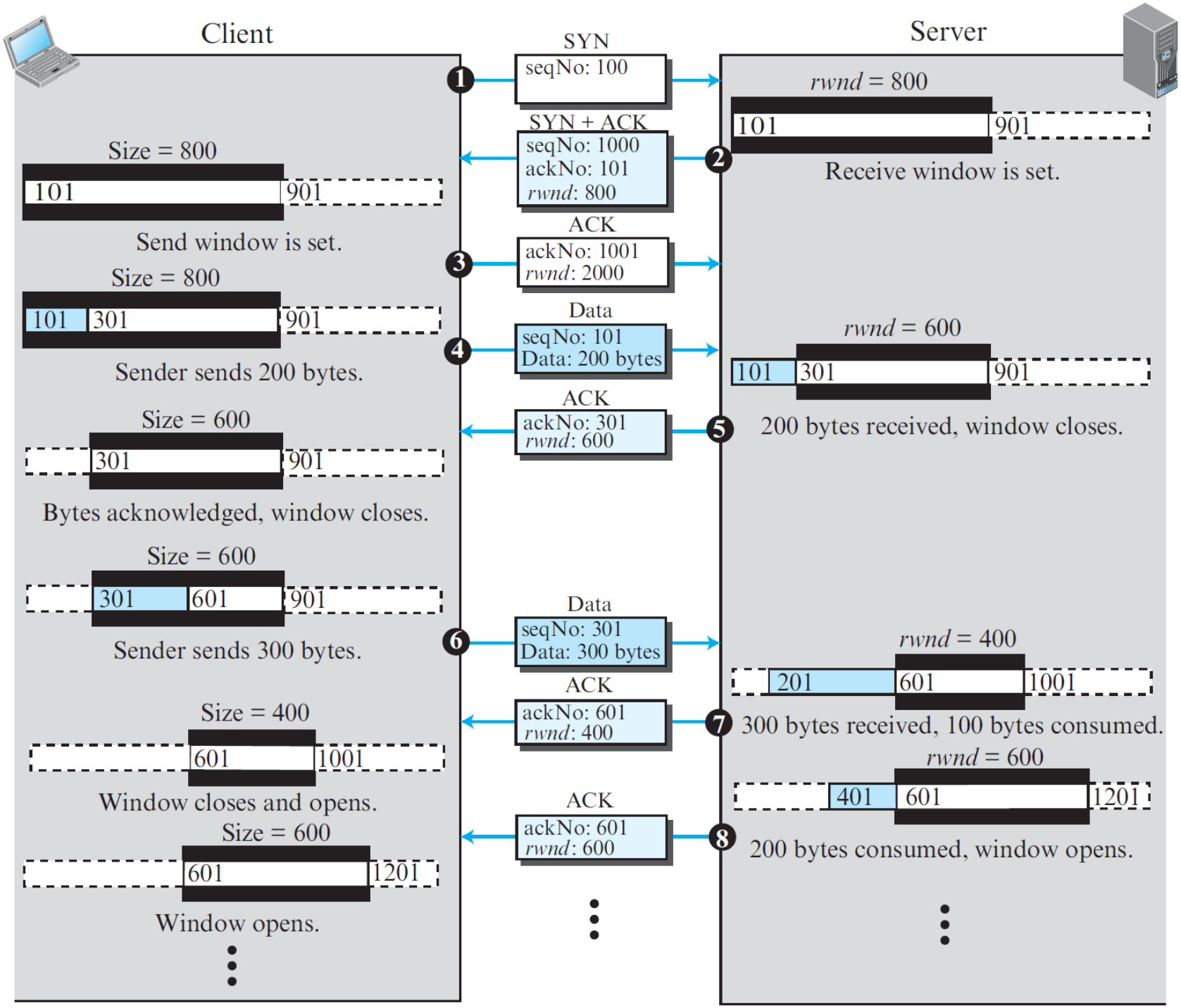


Flow Control

- TCP provides a **flow-control service** to its applications to eliminate the possibility of the sender overflowing the receiver's buffer.
- To achieve flow control, TCP forces the sender and the receiver to adjust their window sizes.

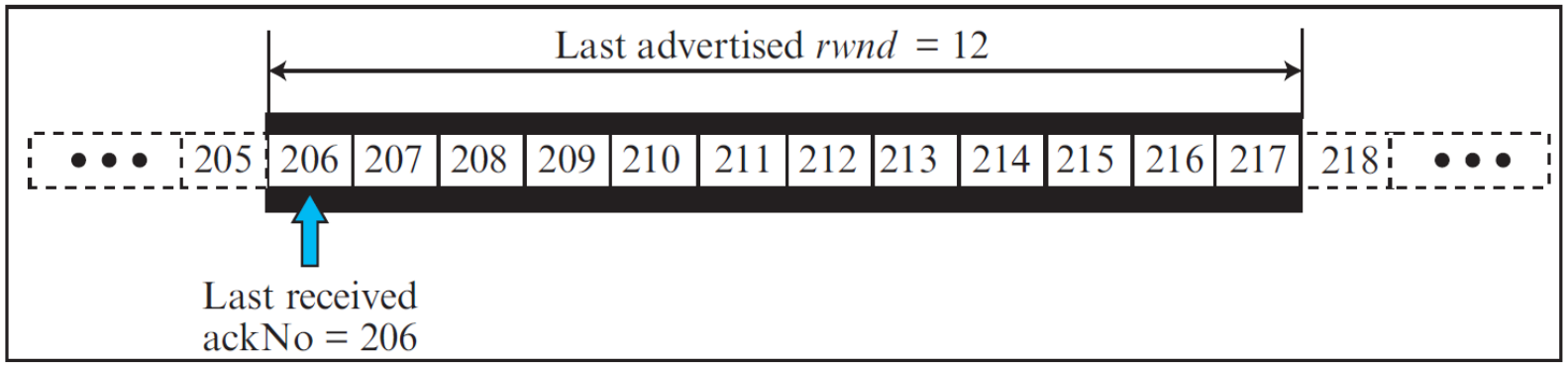
Opening and Closing Windows

- The receive window
  - closes (moves its left wall to the right) when more bytes arrive from the sender;
  - opens (moves its right wall to the right) when more bytes are pulled by the process.
- The send window
  - closes (moves its left wall to the right) when a new acknowledgment allows it to do so.
  - opens (its right wall moves to the right) when the receive window size (rwnd) advertised by the receiver allows it to do so (new ackNo + new rwnd > last ackNo + last rwnd).

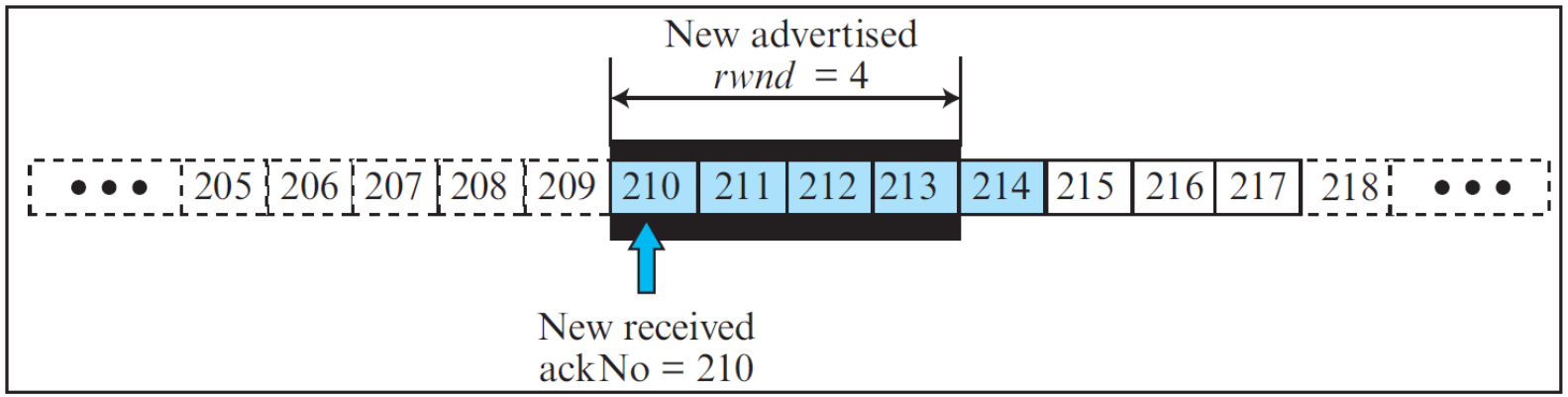

Client
Server
SYN
seqNo: 100
1
rwnd = 800
101
901
Receive window is set.
SYN + ACK
seqNo: 1000
ackNo: 101
rwnd: 800
2
Size = 800
101
901
Send window is set.
ACK
ackNo: 1001
rwnd: 2000
3
Size = 800
101
301
901
Sender sends 200 bytes.
Data
seqNo: 101
Data: 200 bytes
4
rwnd = 600
101
301
901
ACK
ackNo: 301
rwnd: 600
5
200 bytes received, window closes.
Size = 600
301
901
Bytes acknowledged, window closes.
Size = 600
301
601
901
Sender sends 300 bytes.
Data
seqNo: 301
Data: 300 bytes
6
rwnd = 400
201
601
1001
ACK
ackNo: 601
rwnd: 400
7
300 bytes received, 100 bytes consumed.
Size = 400
601
1001
Window closes and opens.
rwnd = 600
401
601
1201
ACK
ackNo: 601
rwnd: 600
8
200 bytes consumed, window opens.
Size = 600
601
1201
Window opens.

Shrinking of Windows

- The receive window cannot shrink.
- The send window can shrink if the receiver defines a value for ***rwnd*** that results in shrinking the window.



a. The window after the last advertisement



b. The window after the new advertisement; window has shrunk

Shrinking of Windows…

- Some implementations do not allow shrinking of the send window.
  - does not allow the right wall of the send window to move to the left.
- The receiver needs to keep the following relationship to prevent shrinking of the send window.

  new ackNo + new rwnd ≥ last ackNo + last rwnd
- Method : receiver postpone its feedback until enough buffer locations are available in its window.
  - The receiver should wait until more bytes are consumed by its process to meet the relationship.

Window Shutdown

- The receiver can temporarily shut down the window by sending a rwnd of 0.
  - if the receiver does not want to receive any data from the sender for a while.
- The sender does not actually shrink the size of the window, but stops sending data until a new advertisement has arrived.
- Even when the window is shut down by an order from the receiver, the sender can always send a segment with 1 byte of data.
- This is called **probing** and is used to prevent a deadlock.

Silly Window Syndrome

- Problem in the sliding window operation.
- Sending of data in very small segments.
  - Because either the sending application program creates data slowly or the receiving application program consumes data slowly, or both.
- Reduces the efficiency.
- Example
  - If TCP sends segments containing only 1 byte of data.
  - A 41-byte datagram (20 bytes of TCP header and 20 bytes of IP header) transfers only 1 byte of user data.

Error Control

- An application program delivers a stream of data to TCP.
- TCP delivers the entire stream to the application program on the other end
  - in order,
  - without error, and
  - without any part lost or duplicated.
- **TCP provides reliability using error control.**

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,** (refer previous slides)
  - **Acknowledgment,** and
  - **Time-out.**

Acknowledgment

- Confirm the receipt of data segments.
- ACK segments do not consume sequence numbers and are not acknowledged.
- **Acknowledgment Types:**
  - **Cumulative Acknowledgment (ACK)**
    - The receiver advertises the next byte it expects to receive, ignoring all segments received and stored out of order.
  - **Selective Acknowledgment (SACK)**
    - Reports a block of bytes that is out of order, and also a block of bytes that is duplicated.
    - SACK is implemented as an **option** at the end of the TCP header.

Rules for Generating Acknowledgments

1. When A sends a data segment to B, it must include (piggyback) an acknowledgment that gives the next sequence number it expects to receive.
2. When the receiver has no data to send and it receives an in-order segment 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.
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.
   - There should not be more than two in-order unacknowledged segments at any time.

Rules for Generating Acknowledgments…

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.
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.
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.

Retransmission

- The heart of the error control mechanism.
- When a segment is sent, it is stored in a queue until it is acknowledged.
- A segment is retransmitted when
  1. the retransmission timer expires or
  2. the sender receives three duplicate ACKs for the first segment in the queue.

Retransmission after RTO

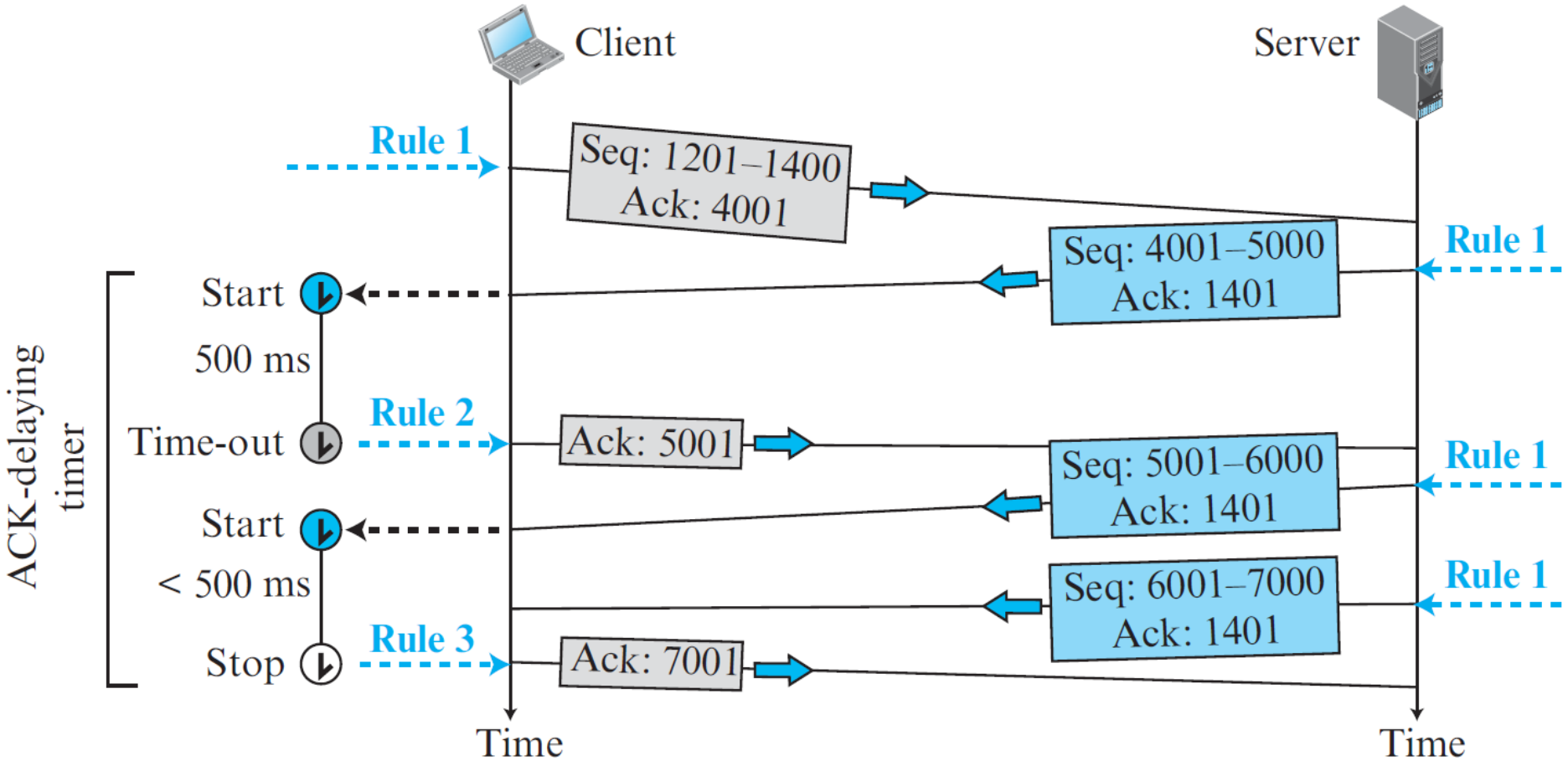
- The sending TCP maintains one retransmission time-out (RTO) for each connection.
- When the timer matures, i.e. times out, TCP resends the segment in the front of the queue and restarts the timer.
- The value of RTO is dynamic in TCP and is updated based on the round-trip time (RTT) of segments.
  - **RTT** is the time needed for a segment to reach a destination and for an acknowledgment to be received.
- Sufficient if the value of RTO is not large.

Fast Retransmission

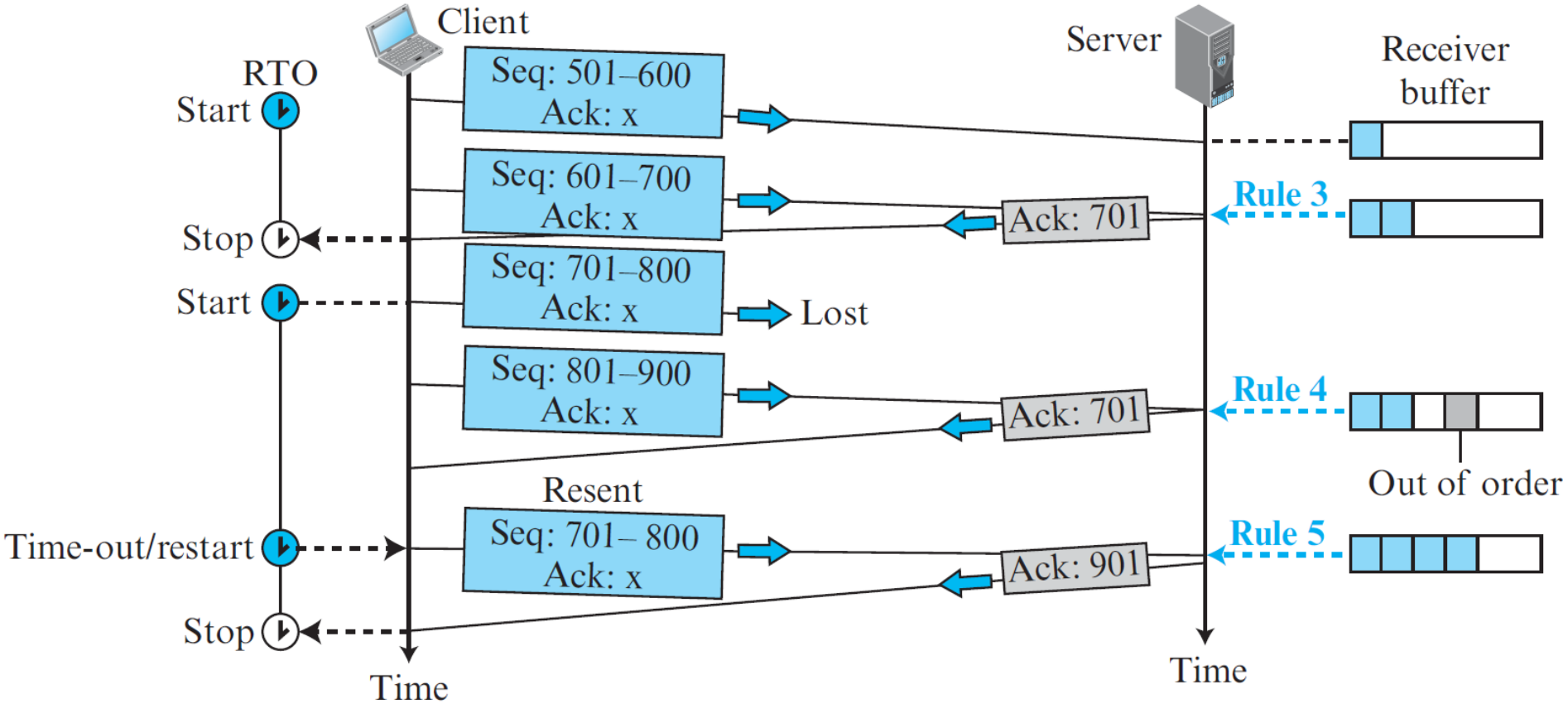
Retransmission after Three Duplicate ACK Segments

- Allow senders to retransmit without waiting for a time out.
- If three duplicate acknowledgments arrive for a segment
  - (i.e., an original ACK plus three exactly identical copies),
  - the next segment is retransmitted without waiting for the time-out.

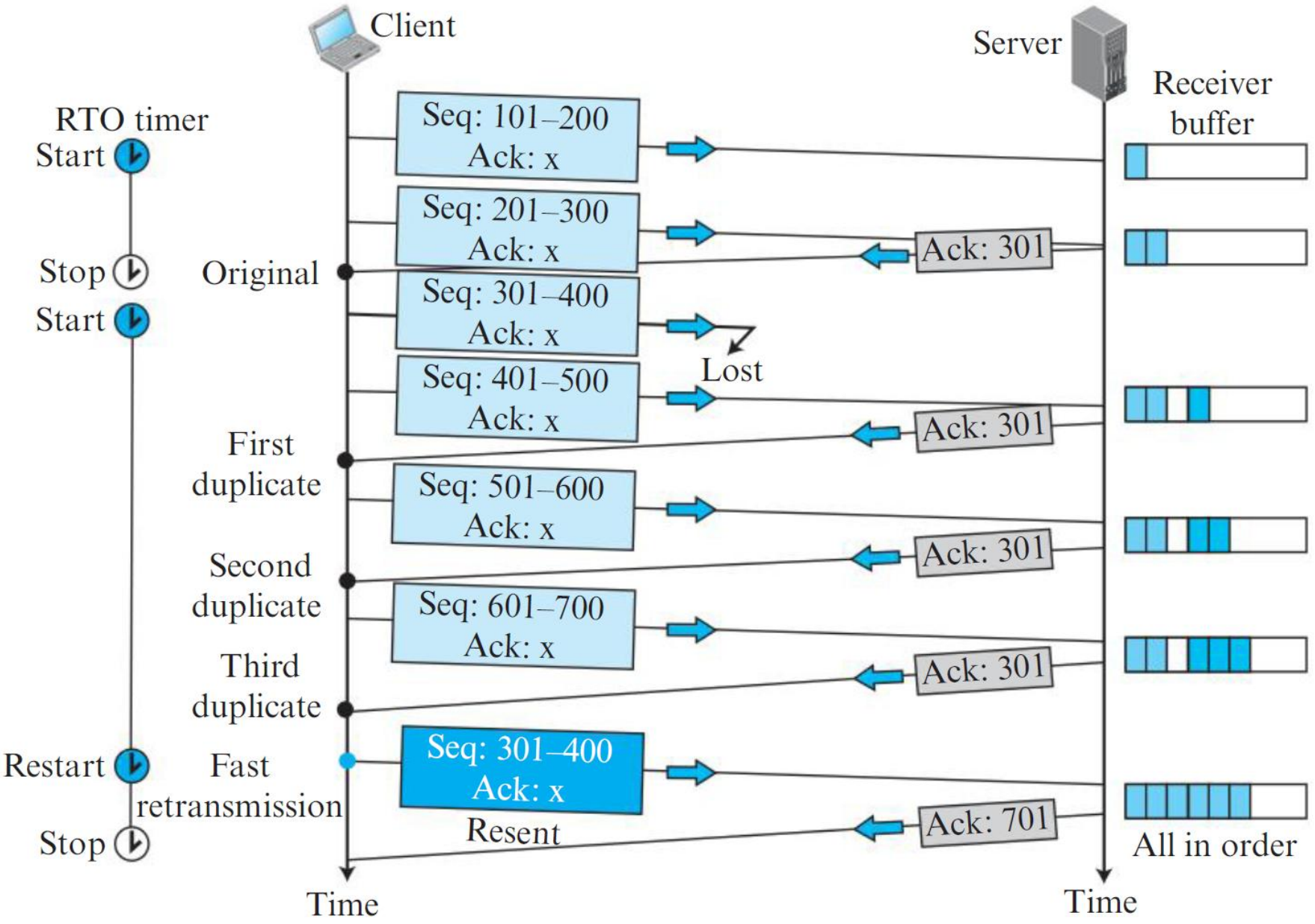
Normal Operation



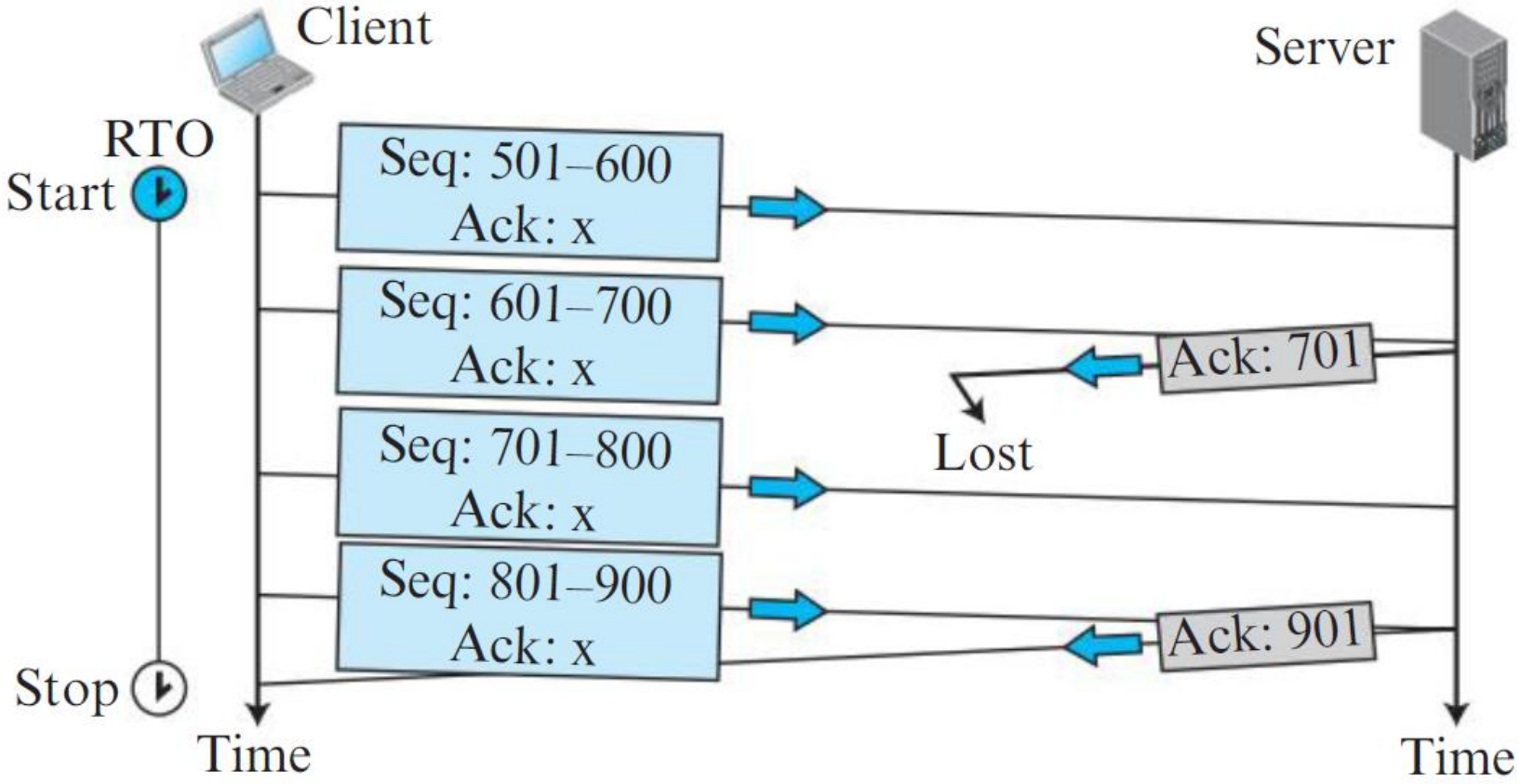
Lost segment



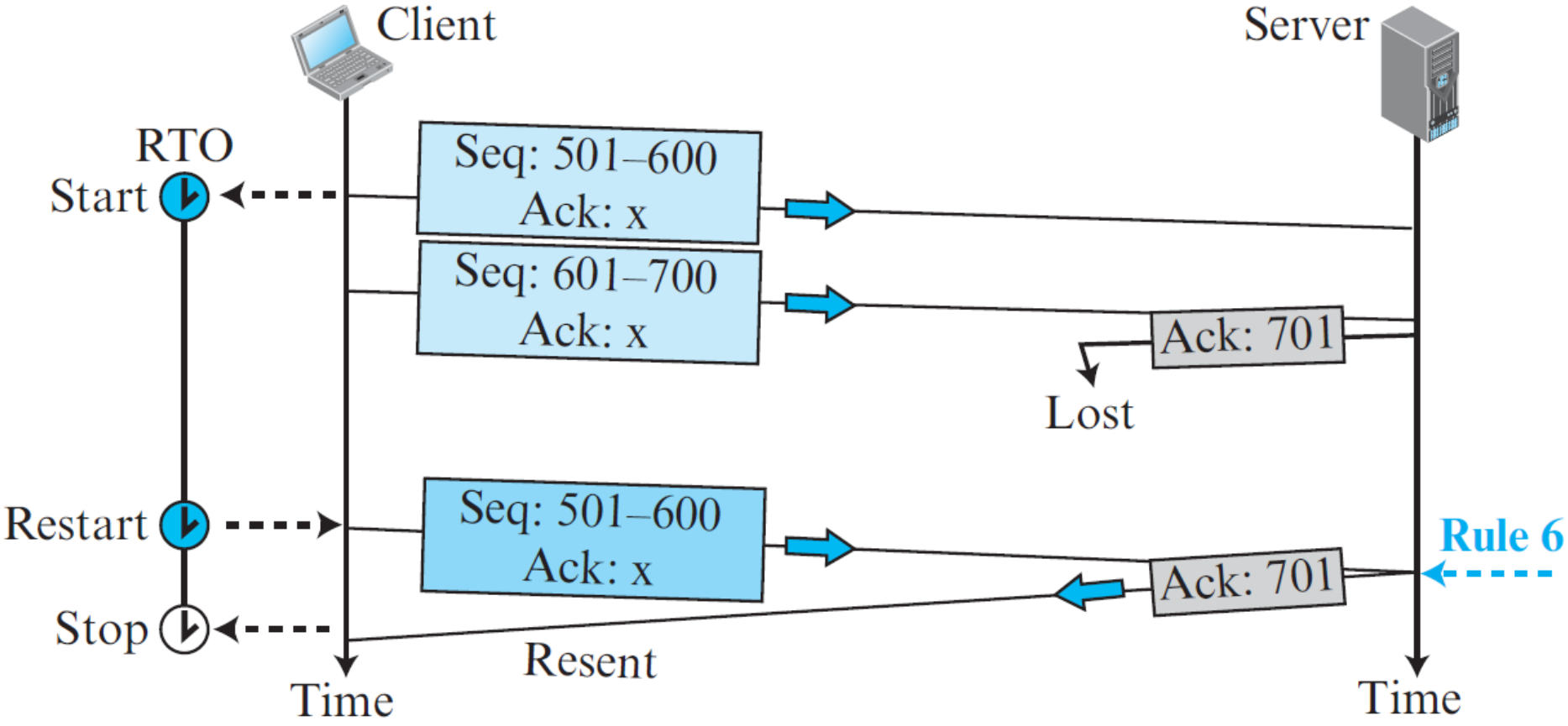
Fast retransmission



Lost acknowledgment



Lost acknowledgment corrected by resending a segment



Deadlock Created by Lost Acknowledgment

- A receiver sends an acknowledgment with ***rwnd*** set to 0 and requests that the sender shut down its window temporarily.
- After a while, the receiver wants to remove the restriction; if it has no data to send, it sends an ACK segment and removes the restriction with a nonzero value for *rwnd*.
- A problem arises if this acknowledgment is lost.
- The sender is waiting for an acknowledgment that announces the nonzero *rwnd*.
- The receiver thinks that the sender has received this and is waiting for data.
- This situation is called a **deadlock**.

Persistence Timer & Probing

- To prevent deadlock, TCP uses a persistence timer for each connection.
- When the sending TCP receives an acknowledgment with a window size of zero, it starts a persistence timer.
- When the persistence timer goes off, the sending TCP sends a special segment called a probe.
- This segment contains only 1 byte of new data.
- The probe causes the receiving TCP to resend the acknowledgment.
- The value of the persistence timer is set to the value of the retransmission time.
- If a response is not received from the receiver, another probe segment is sent and the value of the persistence timer is doubled and reset.
- The sender continues sending the probe segments and doubling and resetting the value of the persistence timer until the value reaches a threshold (usually 60 s).
- After that the sender sends one probe segment every 60 seconds until the window is reopened.

Estimating Round-Trip Time (RTT)

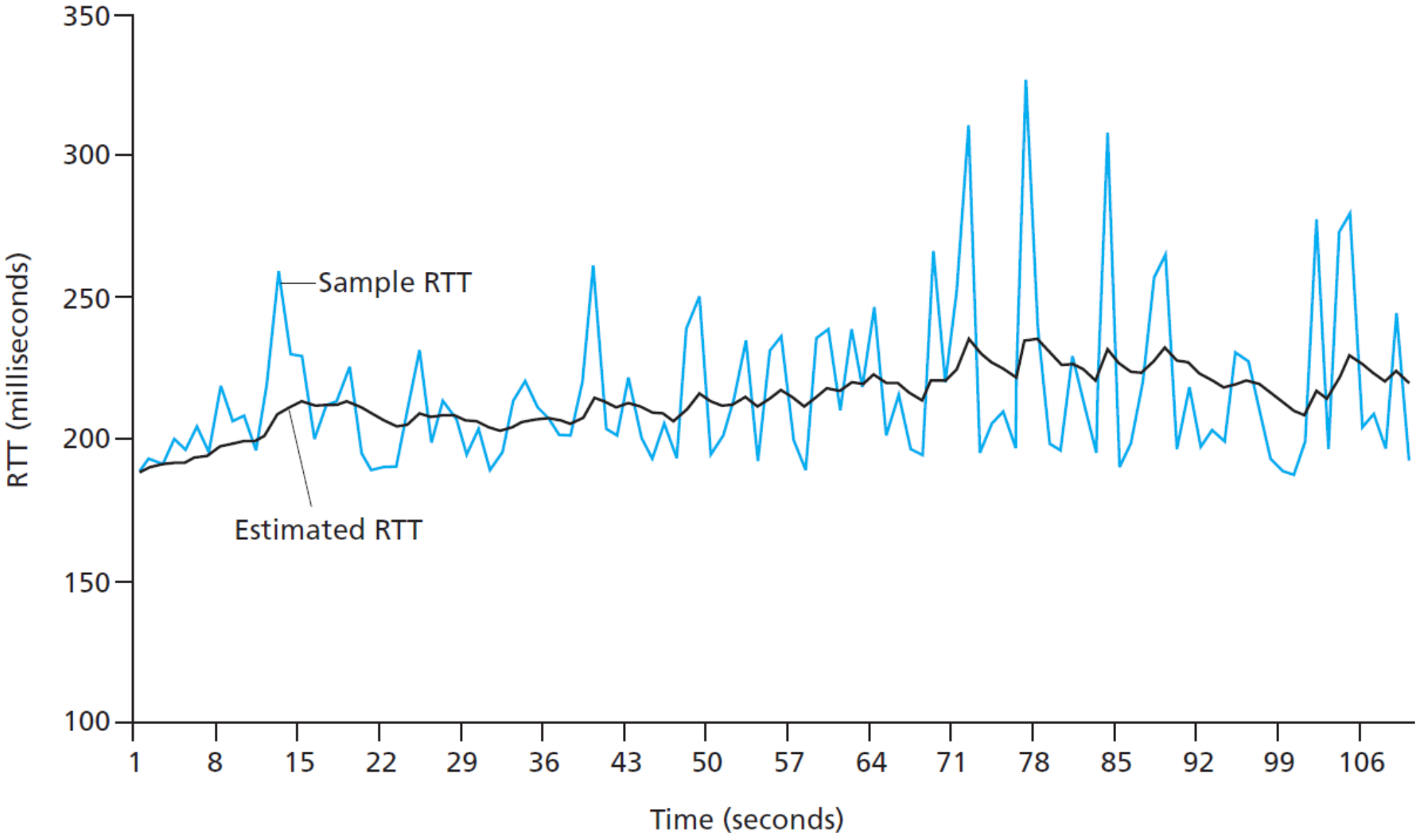
- RTT
  - The amount of time between when a segment is sent and when an acknowledgment for the segment is received.
- Measured RTT (SampleRTT) - RTT_M
  - The time required for the segment to reach the destination and be acknowledged, although the acknowledgment may include other segments.
  - In TCP, only one RTT measurement can be in progress at any time.
    - If an RTT measurement is started, no other measurement starts until the value of this RTT is finalized.
  - TCP never computes a RTT_M for a retransmitted segment.

Estimating Round-Trip Time (RTT)…

- RTT_M values will fluctuate from segment to segment due to congestion in the routers and to the varying load on the end systems.
- Smoothed RTT (EstimatedRTT) – RTT_S
  - TCP maintains an average of the SampleRTT values.

Initially	→	**No value**
After first measurement	→	$\mathbf{RTT_S = RTT_M}$
After each measurement	→	$\mathbf{RTT_S = (1 - \alpha)\ RTT_S + \alpha \times RTT_M}$

  - The recommended value of α is = 0.125 (that is, 1/8)


350
300
250
200
150
100
RTT (milliseconds)
Sample RTT
Estimated RTT
1
8
15
22
29
36
43
50
57
64
71
78
85
92
99
106
Time (seconds)

Estimating Round-Trip Time (RTT)…

- RTT Deviation (DevRTT) – RTT_D
  - Measure of the variability of the RTT.

Initially	$\rightarrow$	**No value**
After first measurement	$\rightarrow$	$RTT_D = RTT_M / 2$
After each measurement	$\rightarrow$	$RTT_D = (1 - \beta)\, RTT_D + \beta \times \lvert RTT_S - RTT_M \rvert$

  - The recommended value of β is 0.25.

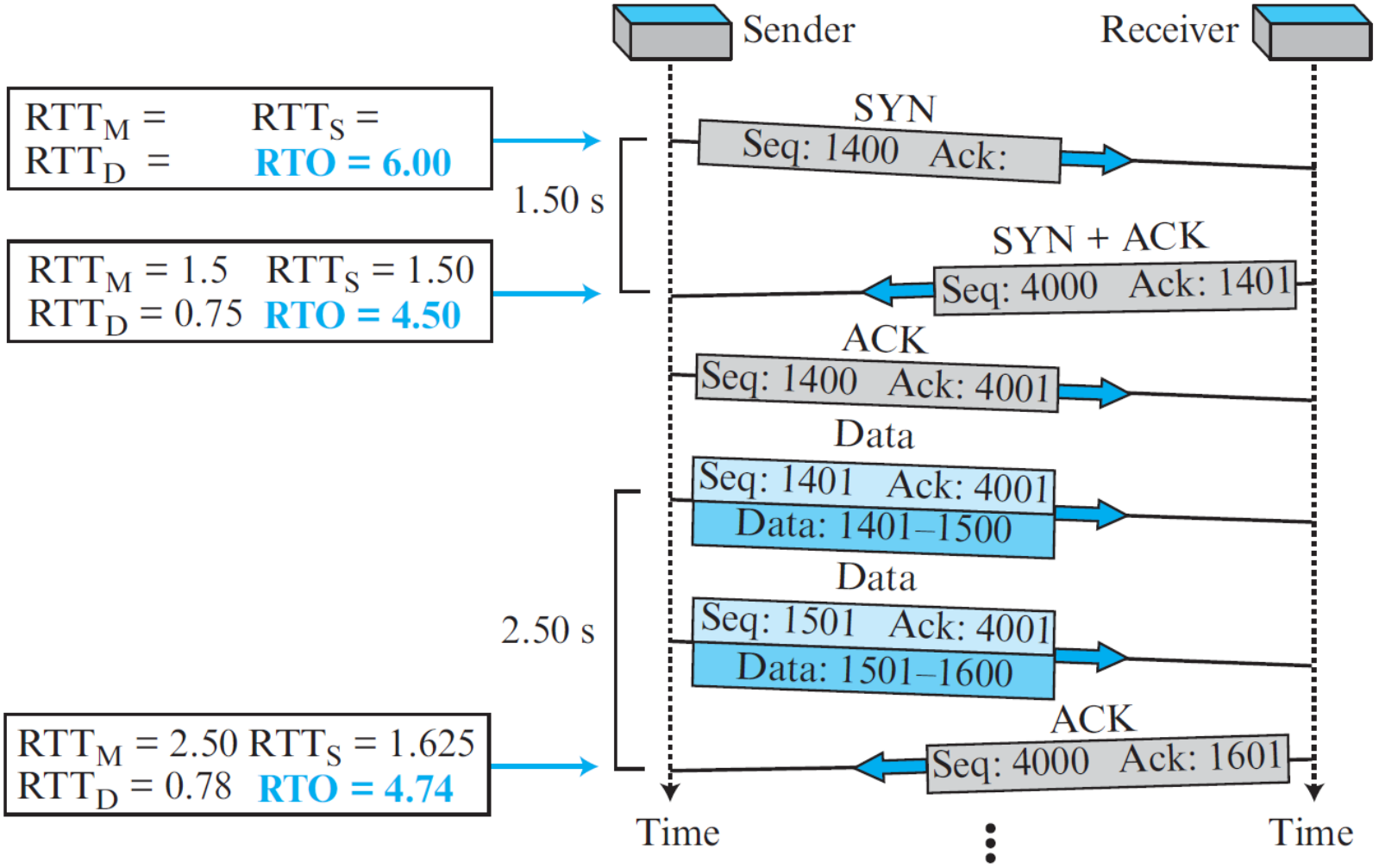
Estimating Retransmission Time-out (RTO)

- The value of RTO is based on the *smoothed RTT* and its deviation.
- An initial RTO value of 1 second is recommended by RFC.
- When a timeout occurs, the value of RTO is doubled.
  - to avoid a premature timeout occurring for a subsequent segment that will soon be acknowledged.
- As soon as a segment is received and *EstimatedRTT* is updated, the RTO is again computed using the formula.

Original	$\rightarrow$	**Initial value**
After any measurement	$\rightarrow$	$\mathbf{RTO} = \mathbf{RTT_S} + 4 \times \mathbf{RTT_D}$

Example

The initial value of RTO is set to 6.00 seconds.



TCP Congestion Control

- In flow control, *rwnd* strategy guarantees that the receive window is never overflowed with the received bytes (no end congestion).
- This does not mean that the intermediate buffers, buffers in the routers, do not become congested.
  - A router may receive data from more than one sender.
  - Buffers of a router may be overwhelmed with data, which results in dropping some segments sent by a specific TCP sender.
- 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.

Congestion Window

- 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.
  - cwnd is related to the congestion in the network.
  - rwnd is related to the congestion at the end.
- The actual size of the window is the minimum of these two.

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

Congestion Detection

- TCP detect the existence of congestion by the occurrence of two events:
  - Time-out
    - The sign of a strong congestion
  - Receiving three duplicate ACKs.
    - The sign of a weak congestion in the network.

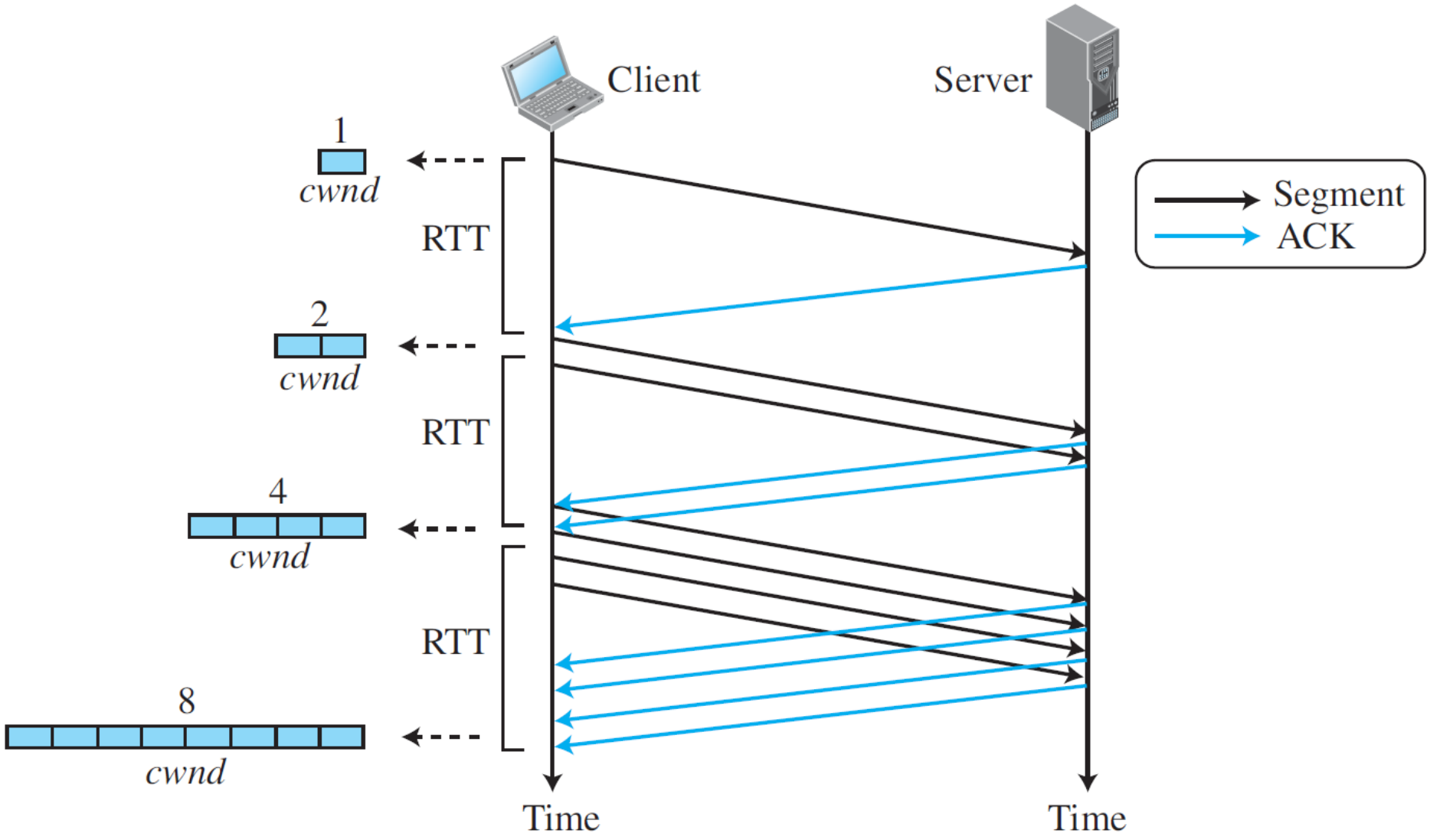
Congestion Policies

- TCP handles congestion is based on three algorithms:
  - **Slow start**
  - **Congestion Avoidance**
  - **Fast recovery**

Slow Start: Exponential Increase

- The size of the congestion window (cwnd) starts with one maximum segment size (MSS), but it increases one MSS each time one acknowledgment arrives.
  - MSS is a value set during the connection establishment.
- The algorithm starts slowly, but grows exponentially.
- Assume that
  - ***rwnd*** is much larger than ***cwnd***, so that the sender window size always equals ***cwnd***.
  - each segment is of the same size and carries MSS bytes
  - each segment is acknowledged individually.

Slow Start: Exponential Increase…



Slow Start: Exponential Increase…

- The sender starts with cwnd = 1.
  - i.e. sender can send only one segment.
- After the first ACK arrives, the size of the congestion window is also increased by 1.
- The size of the window is now 2.
- After sending two segments and receiving two individual acknowledgments for them, the size of the congestion window now becomes 4, and so on.
- The size of the congestion window in this algorithm is a function of the number of ACKs arrived.

If an ACK arrives, cwnd = cwnd + 1.

Slow Start: Exponential Increase…

- The growth rate is exponential in terms of each RTT.

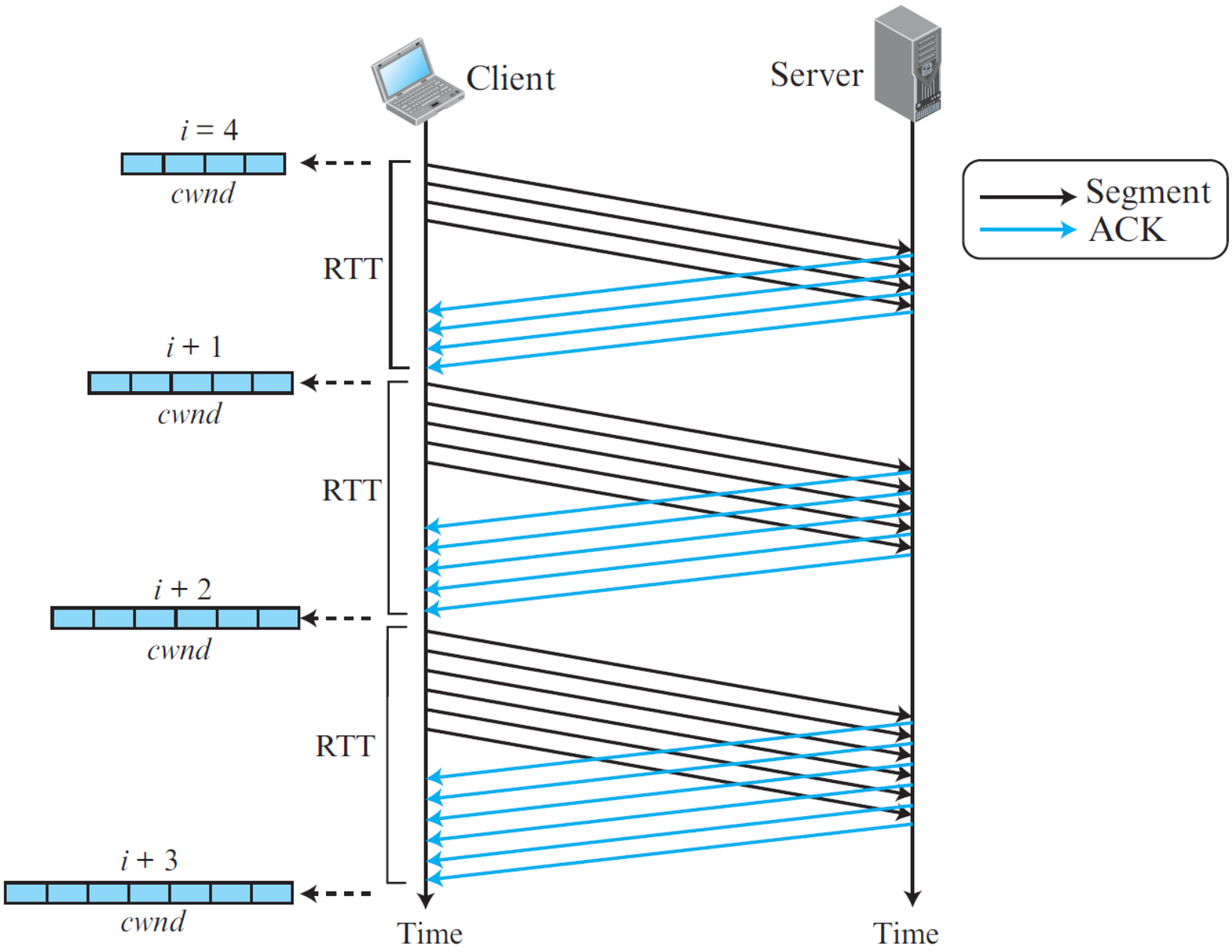
Start	$\rightarrow$	$cwnd = 1 \rightarrow 2^0$
After 1 RTT	$\rightarrow$	$cwnd = cwnd + 1 = 1 + 1 = 2 \rightarrow 2^1$
After 2 RTT	$\rightarrow$	$cwnd = cwnd + 2 = 2 + 2 = 4 \rightarrow 2^2$
After 3 RTT	$\rightarrow$	$cwnd = cwnd + 4 = 4 + 4 = 8 \rightarrow 2^3$

- The size of the congestion window increases exponentially until it reaches a threshold *ssthresh* (slow-start threshold).
- If two segments are acknowledged cumulatively, the size of the *cwnd* increases by only 1.
  - The growth is still exponential, it is a power of 1.5.

Congestion Avoidance: Additive Increase

- TCP defines an algorithm called **congestion avoidance**, which increases the cwnd additively instead of exponentially until congestion is detected.
- When the size of the congestion window reaches the slow-start threshold where **cwnd = i**, the slow-start phase stops and the additive phase begins.
- Each time the whole "window" of segments is acknowledged, the size of the congestion window is increased by one.

Congestion Avoidance: Additive Increase…



Congestion Avoidance: Additive Increase…

- The size of the congestion window in this algorithm is also a function of the number of ACKs that have arrived.

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

- The size of the window increases only 1/cwnd portion of MSS.
- The growth rate is linear in terms of each RTT.

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

Fast Recovery

- The fast-recovery algorithm starts when three duplicate ACKs arrives.
  - light congestion in the network.
- It 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.

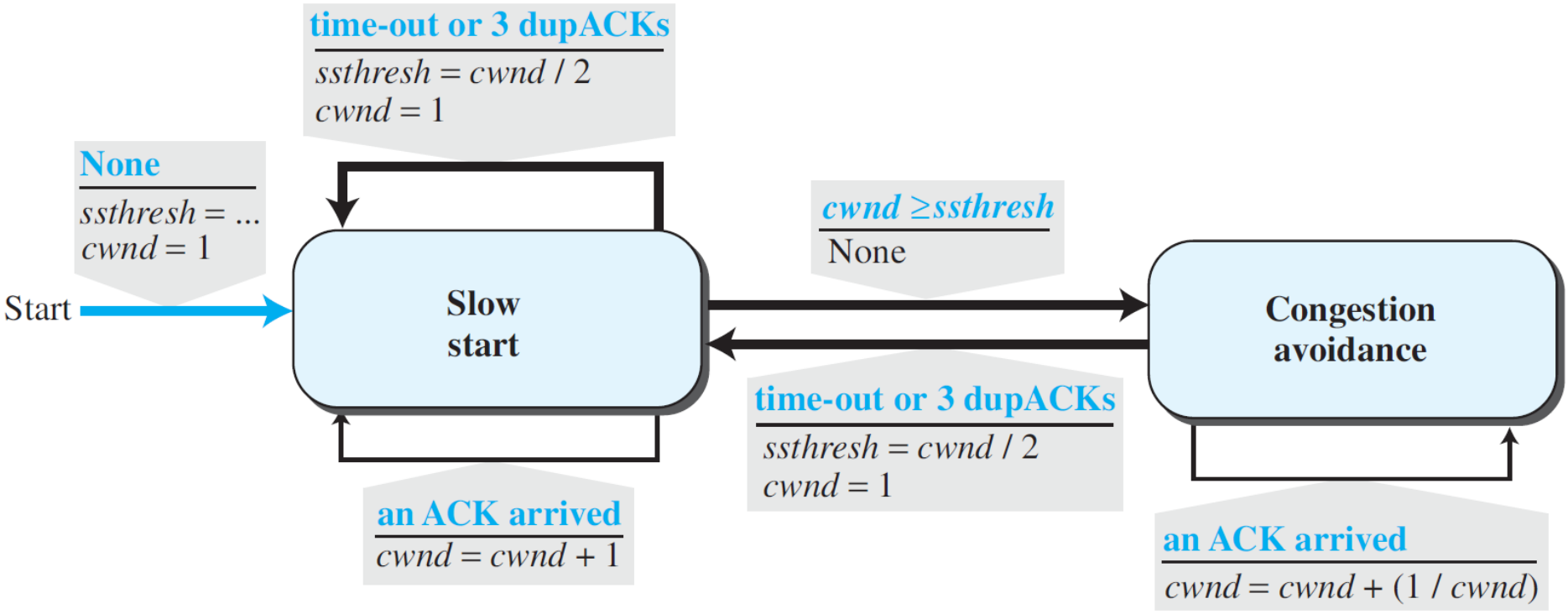
If a duplicate ACK arrives, cwnd = cwnd + (1 / cwnd).

- This algorithm is optional in TCP.

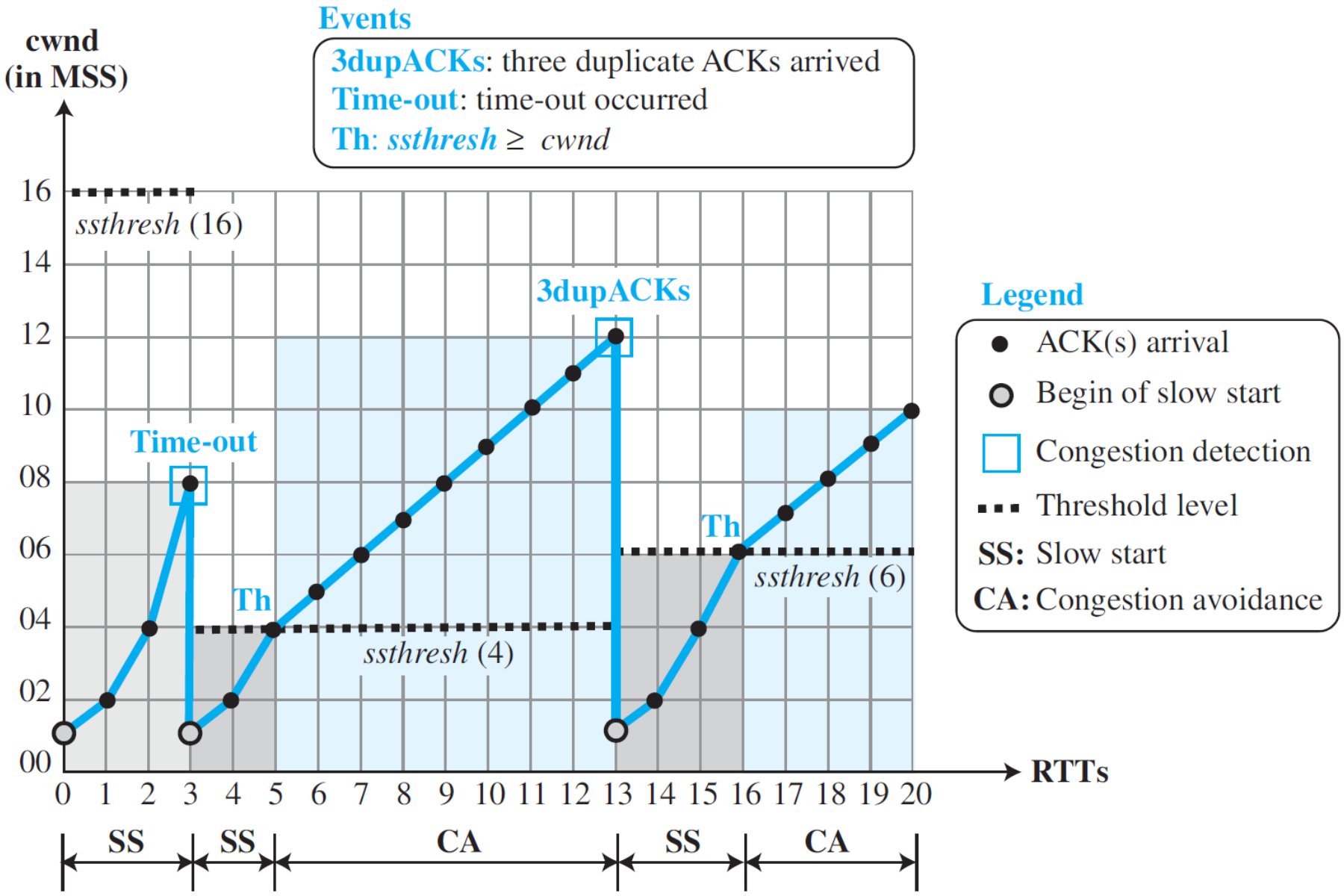
- TCP is said to be **self-clocking.**
  - Because TCP uses acknowledgments to trigger (or clock) its increase in congestion window size.

Taho TCP

- Uses only slow start and congestion avoidance policies.
- Starts the slow-start algorithm and sets the ssthresh variable to a pre-agreed value and the cwnd to 1 MSS.

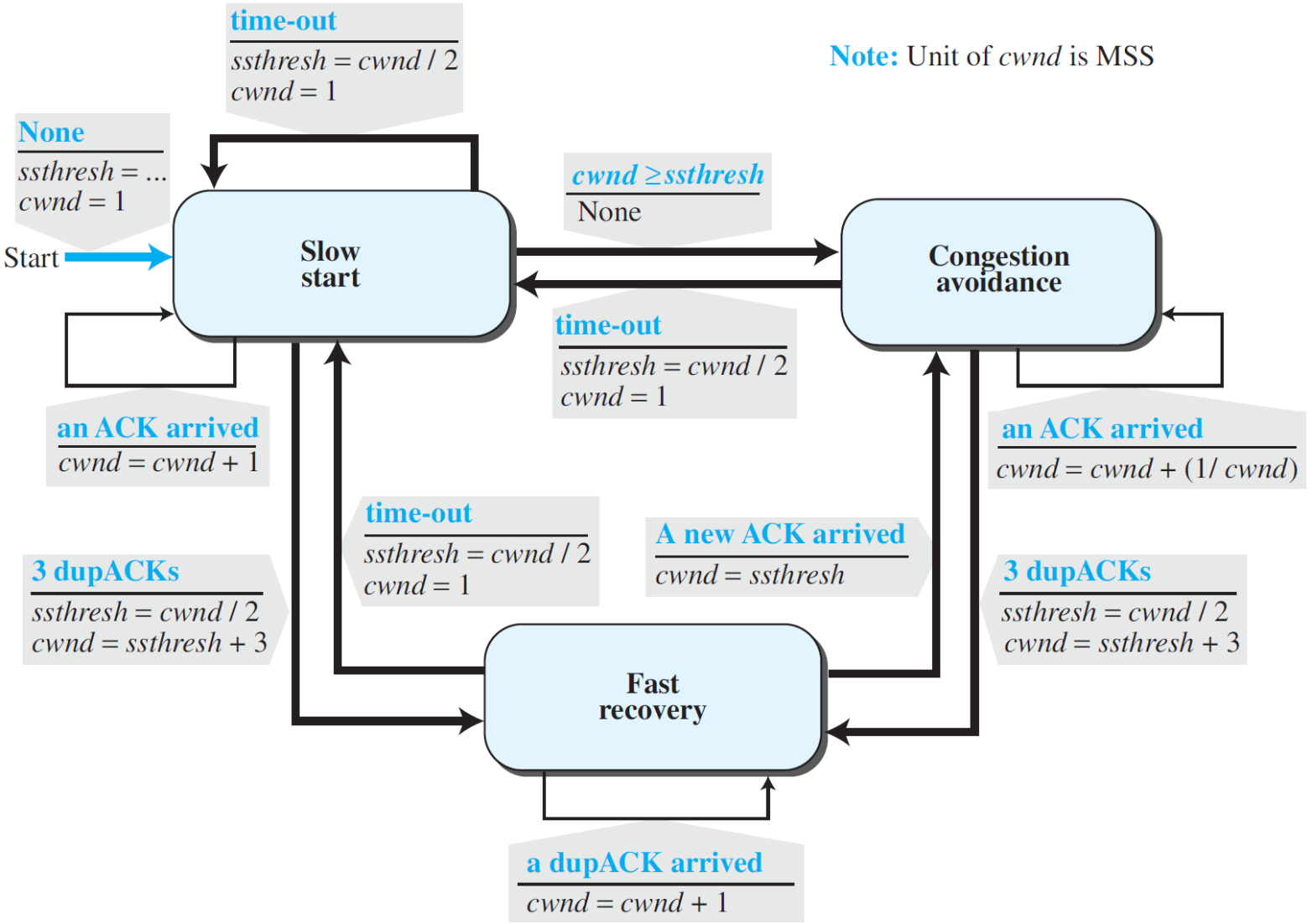


Example of Taho TCP

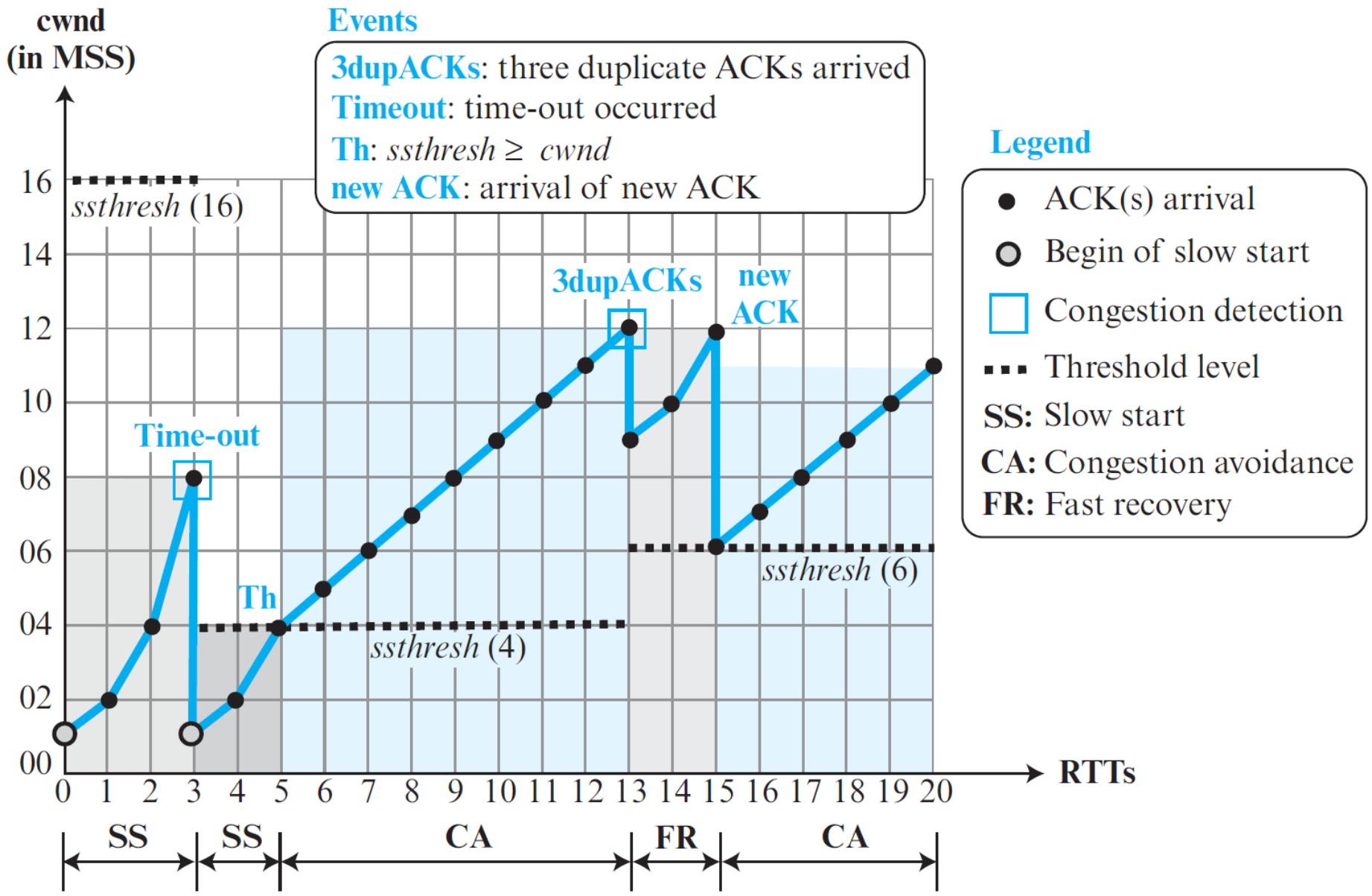


Reno TCP

- Treats the two signals of congestion, time-out and the arrival of three duplicate ACKs, differently.
- Uses all slow start, congestion avoidance and fast recovery policies.



Example of a Reno TCP



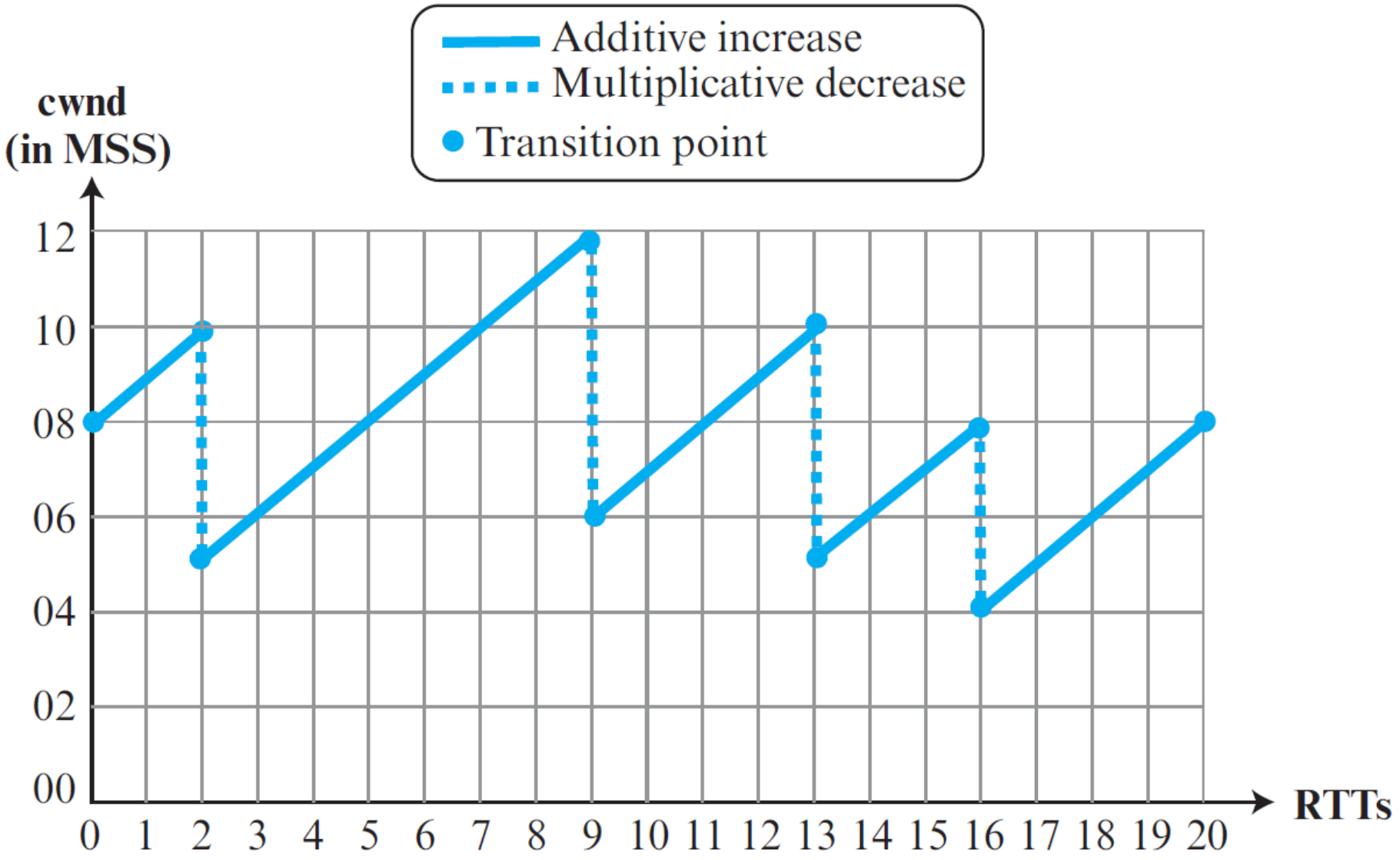
NewReno TCP

- 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, TCP is certain that only one segment was lost.
  - If the ACK number defines a position between the retransmitted segment and the end of the window, the segment defined by the ACK is also lost.
  - NewReno TCP retransmits this segment to avoid receiving more and more duplicate ACKs for it.

Additive Increase, Multiplicative Decrease

- Out of the three versions of TCP, the Reno version is most common today.
- In Reno TCP, most of the time the congestion is detected and taken care of by observing the three duplicate ACKs.
- The congestion window size, after it passes the initial slow start state, follows a saw tooth pattern called **additive increase, multiplicative decrease (AIMD)**.
- TCP congestion window is
  - cwnd = cwnd + (1/cwnd) when an ACK arrives (congestion avoidance) - **additive increase**
  - cwnd = cwnd / 2 when congestion is detected - **multiplicative decrease**

Additive Increase, Multiplicative Decrease



TCP Throughput

- TCP sends a *cwnd* bytes of data and receives acknowledgement for them in RTT time.
- i.e. *throughput = cwnd / RTT*.
- The behavior of TCP, is not a flat line (constant); it is like saw teeth, with many minimum and maximum values.
  - The value of the maximum is twice the value of the minimum because in each congestion detection the value of *cwnd* is set to half of its previous value.
- So, **Throughput = (0.75) W_{max} / RTT**
  - W_{max} is the average of window sizes when the congestion occurs.

TCP Throughput…

- **TCP Over High-Bandwidth Paths**
  - Consider the high-speed TCP connections that are needed for grid- and cloud-computing applications.
  - The throughput of a TCP connection as a function of the loss rate (*L*), the *round-trip time (RTT)*, and the *maximum segment size (MSS*):

$$Average\ Throughput = \frac{1.22\ \mathrm{MSS}}{\mathrm{RTT}\ \sqrt{\mathrm{L}}}$$

How should a TCP sender determine the rate at which it should send?

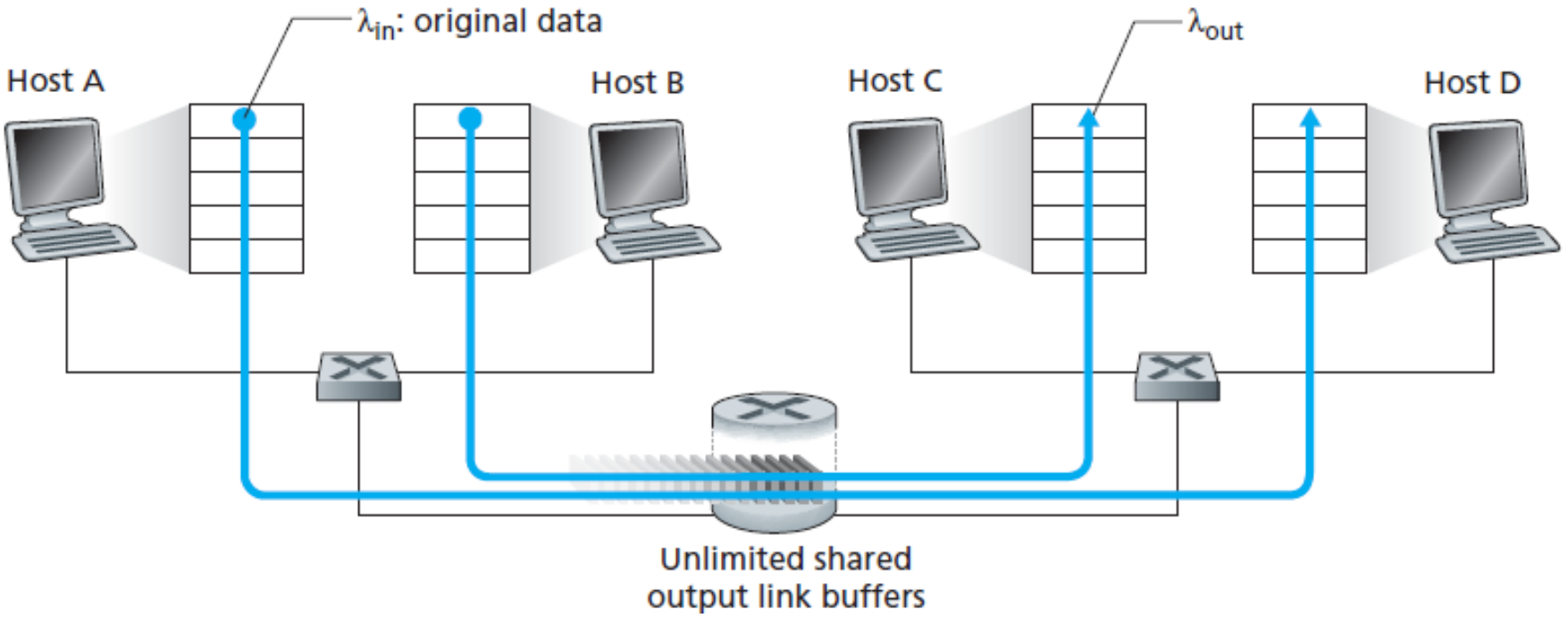
Guiding principles:

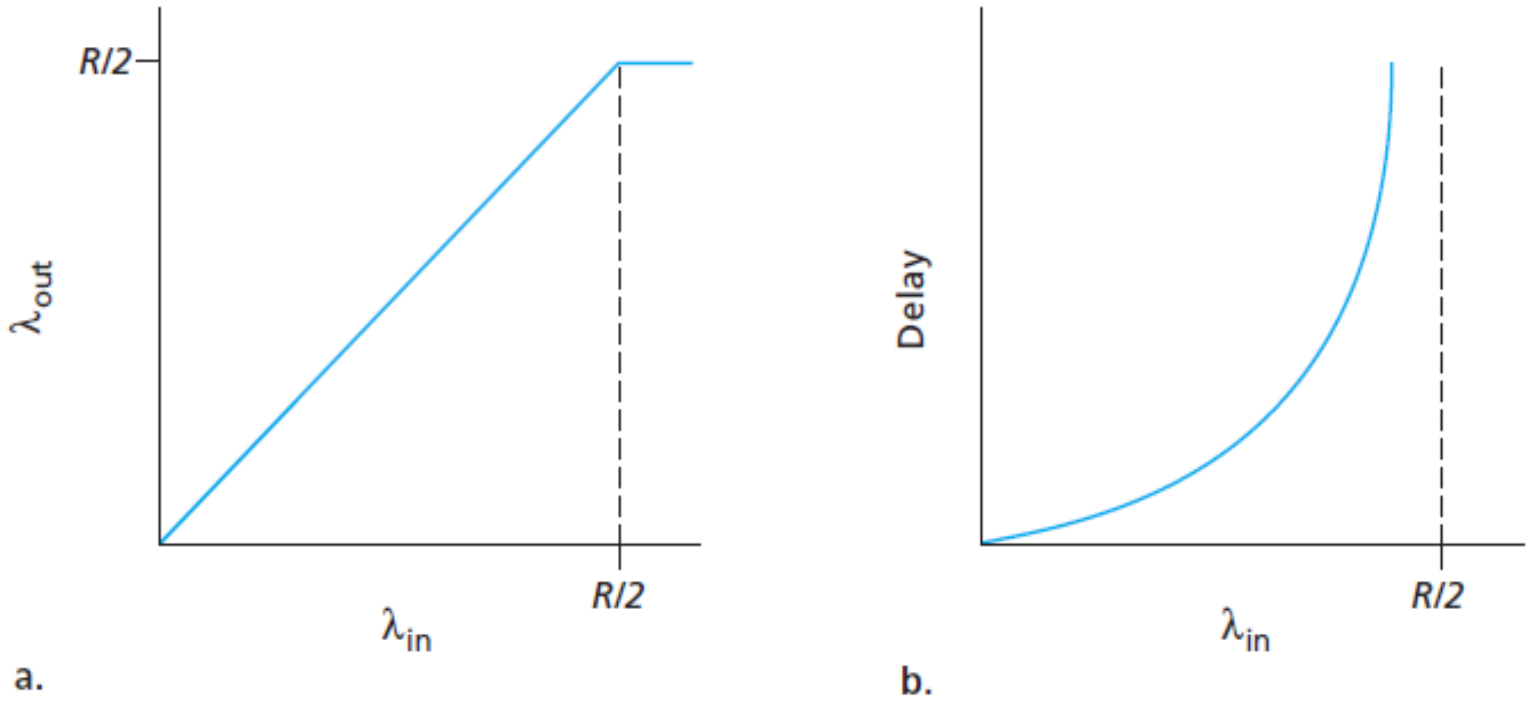
1. A lost segment implies congestion, and hence, the TCP sender's rate should be decreased when a segment is lost.
2. An acknowledged segment indicates that the network is delivering the sender's segments to the receiver, and hence, the sender's rate can be increased when an ACK arrives for a previously unacknowledged segment.
3. Bandwidth probing.
   - The TCP sender increases its transmission rate to probe for the rate that at which congestion onset begins, backs off from that rate, and then to begins probing again to see if the congestion onset rate has changed.

The Causes and the Costs of Congestion

- Scenario 1:
  - Two Senders, a Router with Infinite Buffers
- Scenario 2:
  - Two Senders and a Router with Finite Buffers
- Scenario 3:
  - Four Senders, Routers with Finite Buffers, and Multihop Paths

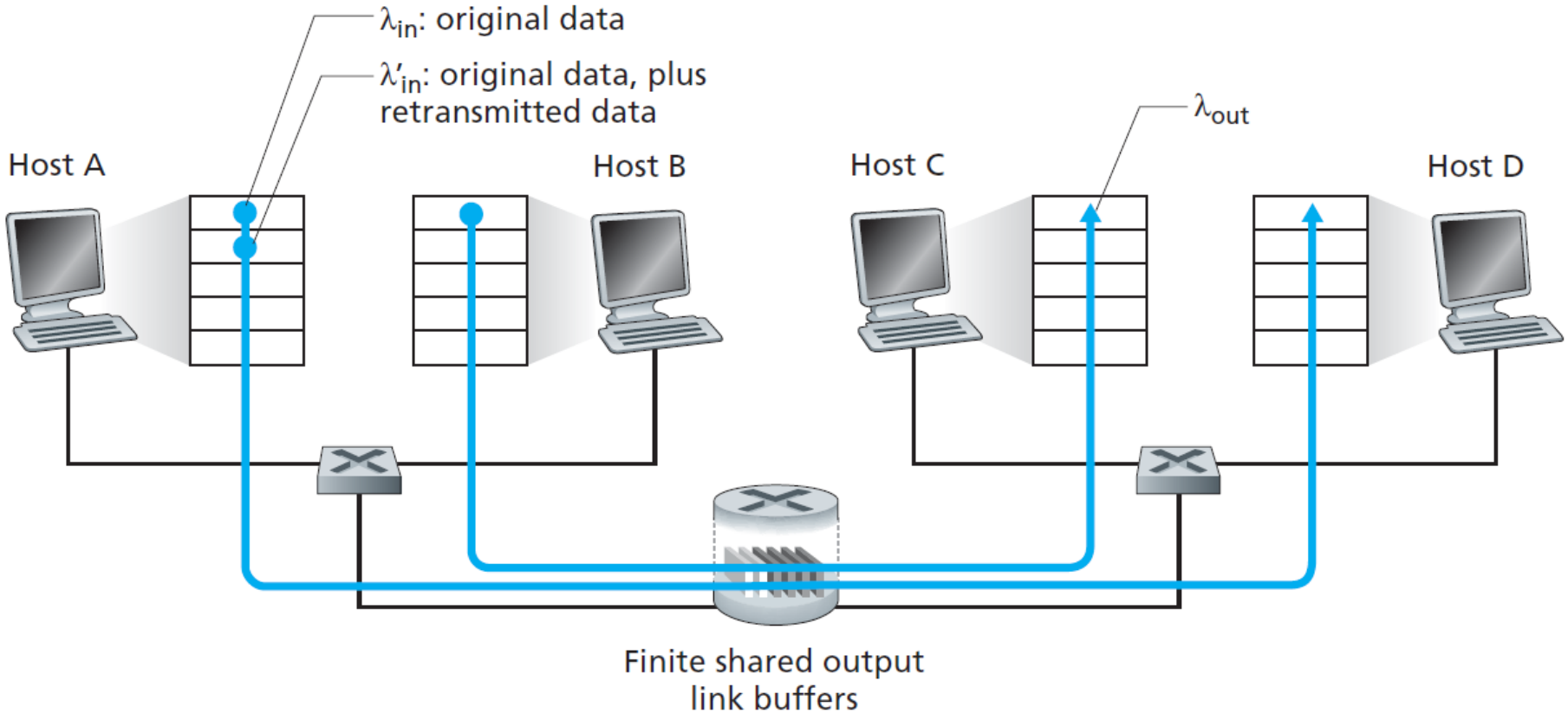
- **Scenario 1: Two Senders, a Router with Infinite Buffers**
  - *Cost of a congested network—large queuing delays are experienced as the packet arrival rate nears the link capacity.*





Throughput and delay as a function of host sending rate

- **Scenario 2: Two Senders and a Router with Finite Buffers**
  - Cost of a congested network— the sender must perform retransmissions in order to compensate for dropped (lost) packets due to buffer overflow.



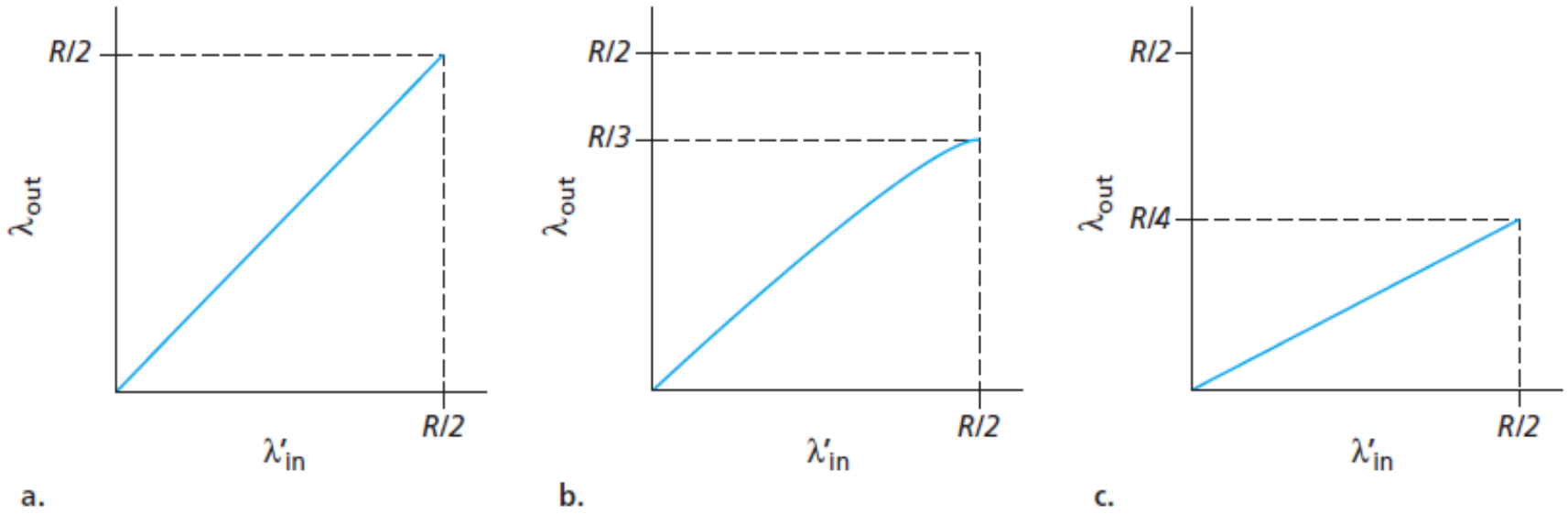
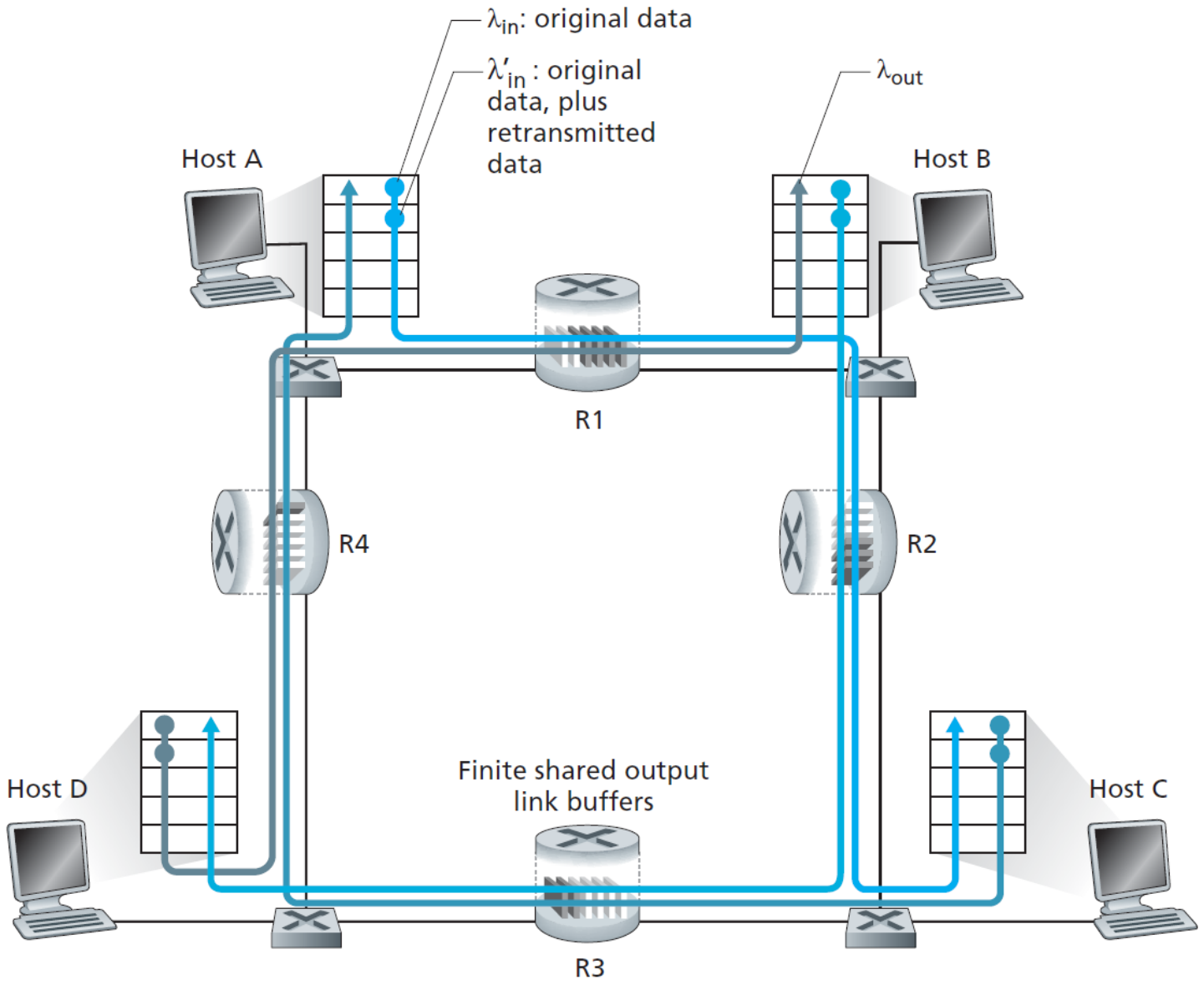


Fig : Performance with finite buffers

– *Another cost of a congested network—unneeded retransmissions by the sender in the face of large delays may cause a router to use its link bandwidth to forward unneeded copies of a packet.*

- **Scenario 3: Four Senders, Routers with Finite Buffers, and Multihop Paths**



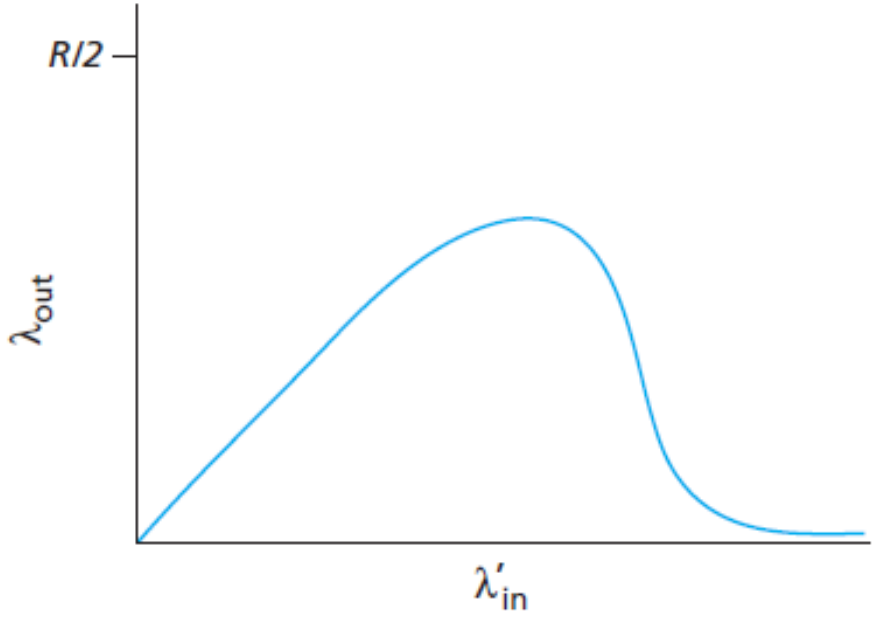


Fig : Performance with finite buffers and multihop paths

– *Cost of dropping a packet due to congestion* — *when a packet is dropped along a path, the transmission capacity that was used at each of the upstream links to forward that packet to the point at which it is dropped ends up having been wasted.*

Congestion-control Approaches

1. ***End-to-end congestion control.***
   - Network layer provides ***no*** *explicit* ***support*** *to the transport layer for congestion control* purposes.
2. ***Network-assisted congestion control.***
   - Network-layer components (that is, routers) provide explicit feedback to the sender regarding the congestion state in the network.
   - E.g. congestion control in the **available bit-rate (ABR)** service in **asynchronous transfer mode (ATM)** networks.

Connectionless transport : UDP

User Datagram Protocol (UDP)

- Connectionless, unreliable transport protocol.
- Provides process-to-process communication.
- Simple protocol using a minimum of overhead.
  - If a process wants to send a small message and does not care much about reliability, it can use UDP.
  - Sending a small message using UDP takes much less interaction between the sender and receiver than using TCP.

UDP…

- UDP – at sender
  - 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.
- UDP – at receiver
  - Uses the destination port number to deliver the segment's data to the correct application process.

UDP…

- UDP is *connectionless*.
  - There is no handshaking between sending and receiving transport-layer entities before sending a segment.
- **DNS** is an example of an application-layer protocol that typically uses UDP.

Why an application developer would choose to build an application over UDP rather than over TCP?

1. *Finer application-level control over what data is sent, and when.*
2. *No connection establishment.*
3. *No connection state.*
4. *Small packet header overhead.*

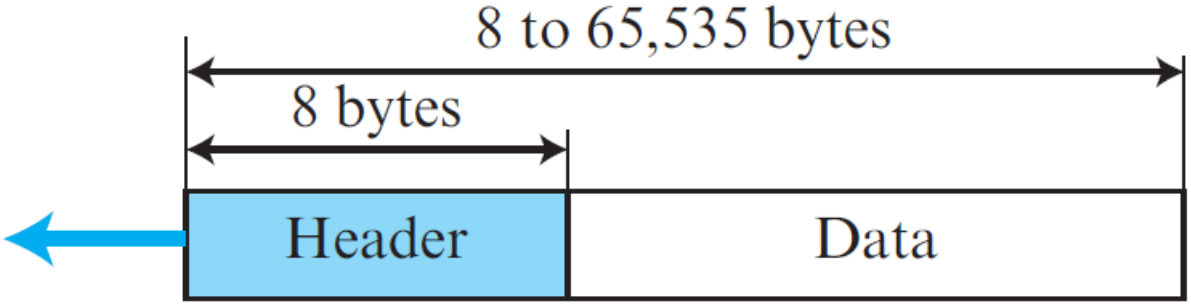
Refer Kurose and Ross, Pages No.199-200

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Remote file server	NFS	Typically UDP
Streaming multimedia	typically proprietary	UDP or TCP
Internet telephony	typically proprietary	UDP or TCP
Network management	SNMP	Typically UDP
Routing protocol	RIP	Typically UDP
Name translation	DNS	Typically UDP

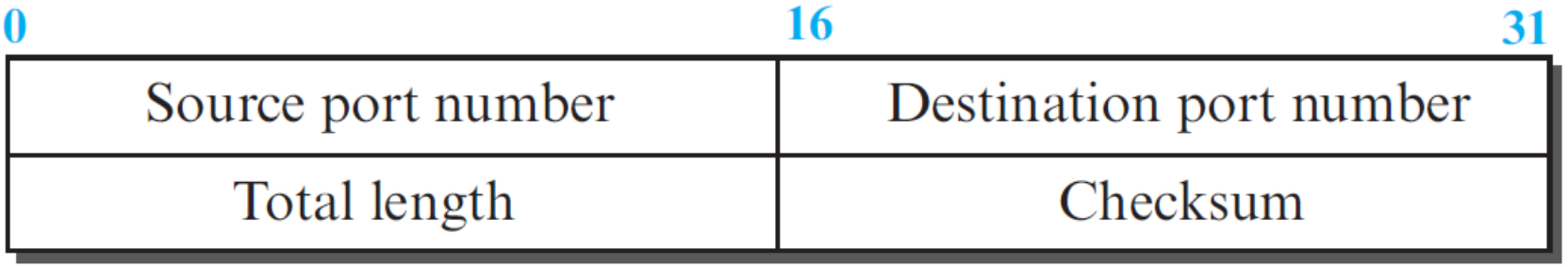
Fig : Popular Internet applications and their underlying transport protocols

UDP Segment Structure

- UDP packets, called user datagrams, have a fixed-size header of 8 bytes.



a. UDP user datagram



b. Header format

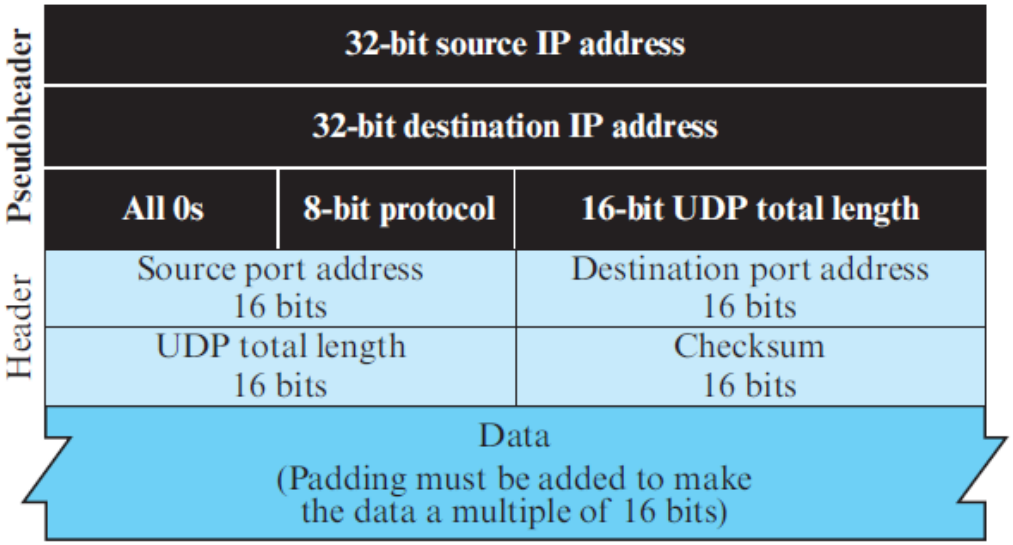
UDP Segment Structure…

- 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).
  - 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 detect errors have been introduced into the segment.

UDP Checksum

- UDP checksum calculation includes three sections:
  - A pseudoheader,
  - UDP header, and
  - Data coming from the application layer.
- The pseudoheader is the part of the header of the IP packet.
- The value of the protocol field for UDP is 17.

- Performs the 1s complement addition of all the 16-bit words.
- Result is put in the checksum field.



UDP Features

- Connectionless Service
  - No overhead to establish and close a connection.
  - Provides less delay.
- Lack of Error Control
- Lack of Congestion Control

UDP Applications

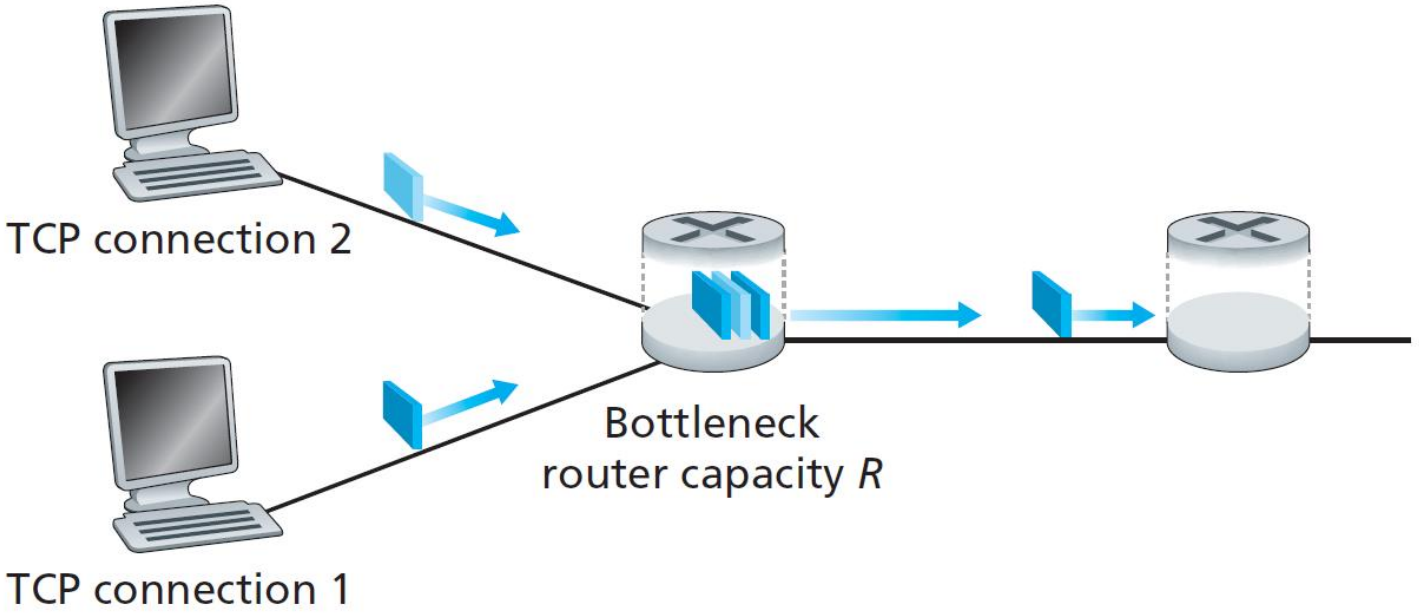
- Suitable for a process that requires simple request-response communication with little concern for flow and error control.
  - Not used for a process that needs to send bulk data (FTP).
- Suitable for a process with internal flow- and error-control mechanisms.
  - E.g. Trivial File Transfer Protocol (TFTP)
- Suitable transport protocol for multicasting.
- Used for management processes such as SNMP.
- Used for some route updating protocols such as Routing Information Protocol (RIP).
- Used for interactive real-time applications that
  - cannot tolerate uneven delay between sections of a received message and can tolerate some data loss.

Fairness

- Consider ***K*** *TCP connections,* each with a different end-to-end path, but all passing through a bottleneck link with transmission rate *R bps*.
- Suppose each connection is transferring a large file and there is no UDP traffic passing through the bottleneck link.
- A congestion-control mechanism is said to be ***fair*** *if the average transmission rate of each connection* is approximately ***R/K****;*
  - that is, each connection gets an equal share of the link bandwidth.

Example

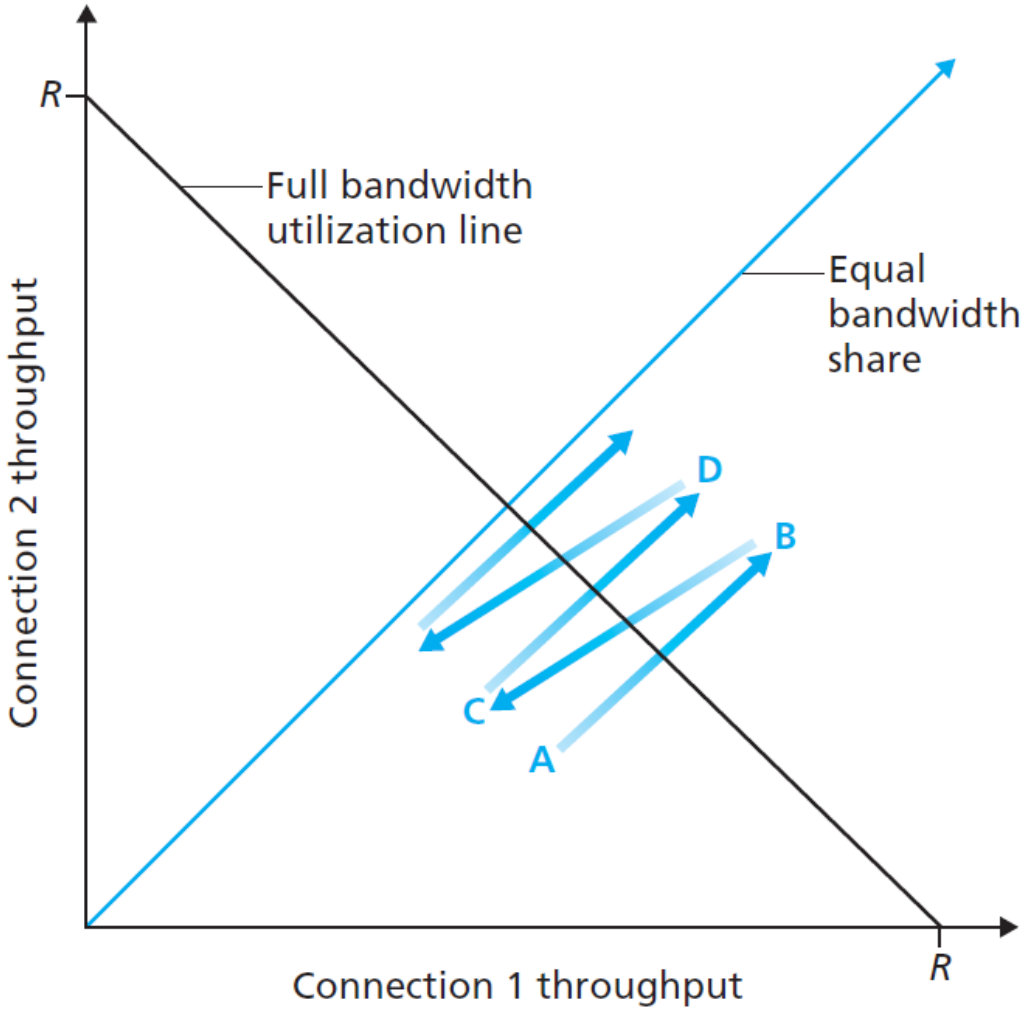
- Consider two TCP connections sharing a single link with transmission rate *R*.
- Assume that
  - The two connections have the same MSS and RTT.
  - They have a large amount of data to send.
  - Operating in CA mode (AIMD) at all times.



Example…

Throughput

- Ideally, the sum of the two throughputs should equal *R.*
- In practice, applications can obtain very unequal portions of link bandwidth.
- When multiple connections share a common bottleneck, smaller RTT sessions enjoy higher throughput than larger RTT sessions .



Fairness and UDP

- Many multimedia applications, such as Internet phone and video conferencing, prefer to run over UDP, which does not have built-in congestion control.
- Applications can pump their audio and video into the network at a constant rate and occasionally lose packets, rather than reduce their rates to “fair” levels at times of congestion and not lose any packets.
  - Multimedia applications running over UDP are not being fair—they do not cooperate with the other connections nor adjust their transmission rates appropriately.
- UDP sources to crowd out TCP traffic.

Fairness and Parallel TCP Connections

- When an application uses multiple parallel connections, it gets a larger fraction of the bandwidth in a congested link.
  - Web browsers use multiple parallel TCP connections to transfer the multiple objects within a Web page.