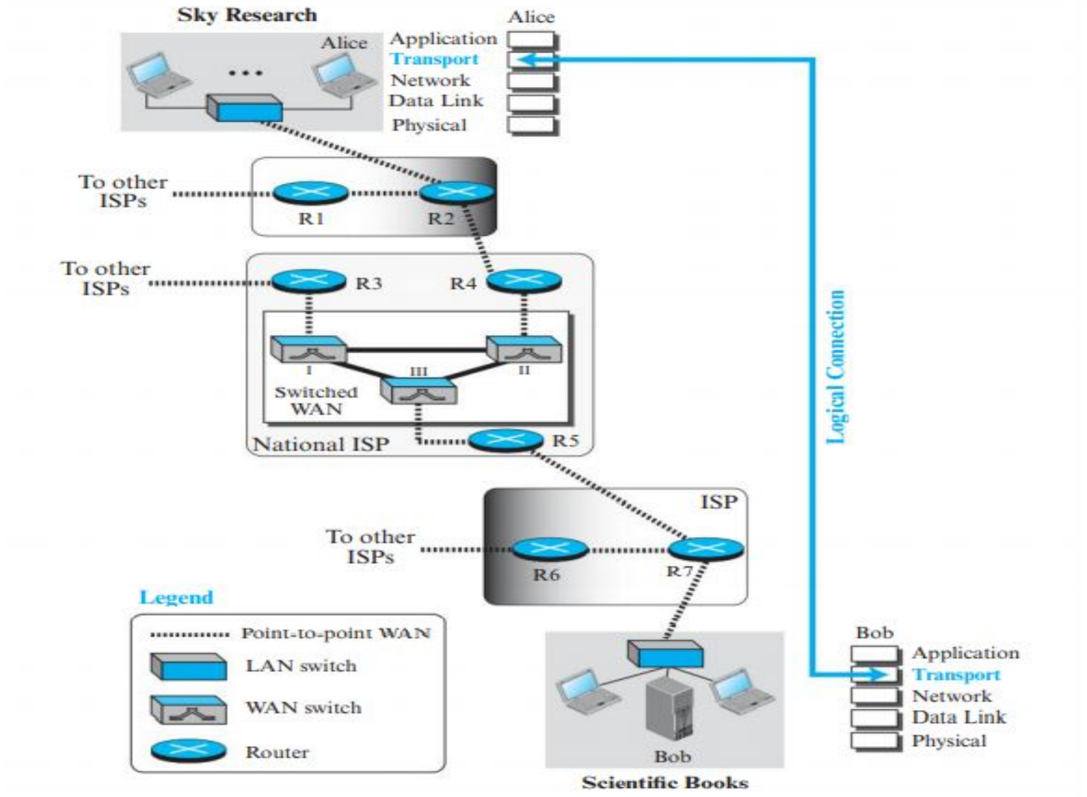
COMPUTER NETWORKS

II MODULE - TRANSPORT LAYER

Transport layer

- It provides a process-to-process communication between two application layers running on different hosts.
- Communication is provided using a logical connection.
- 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.
-

Figure *Logical connection at the transport layer*



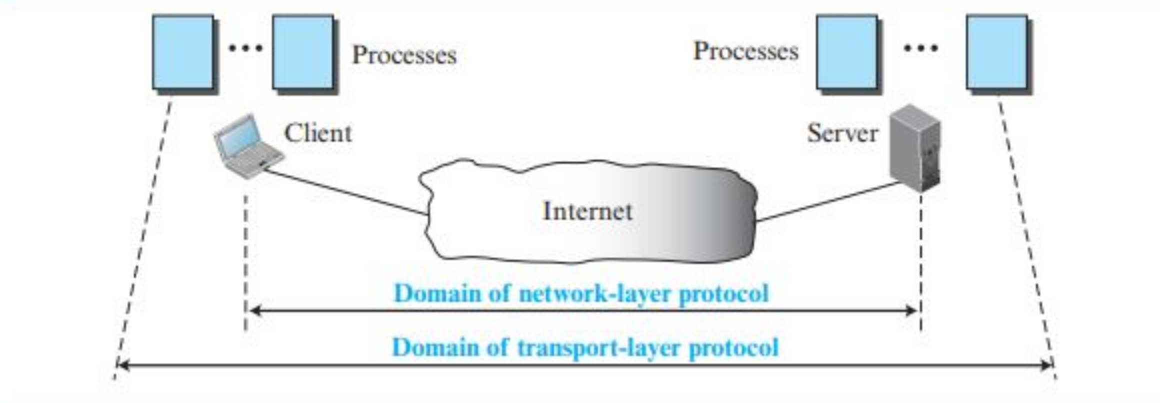
Transport-Layer Services

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

Transport-Layer Services:

i) Process-to-Process Communication:

Figure Network layer versus transport layer


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

•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

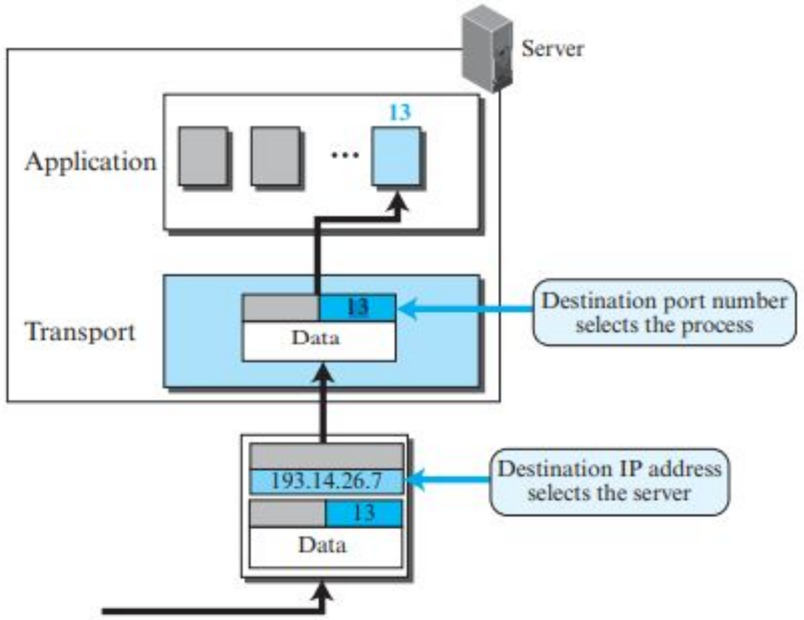
- The network layer is responsible for communication at the computer level (host-to-host communication).
- A network-layer protocol can deliver the message only to the destination computer.
- The message still needs to be handed to the correct process. This is done by transport-layer protocol.
- A transport-layer 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.

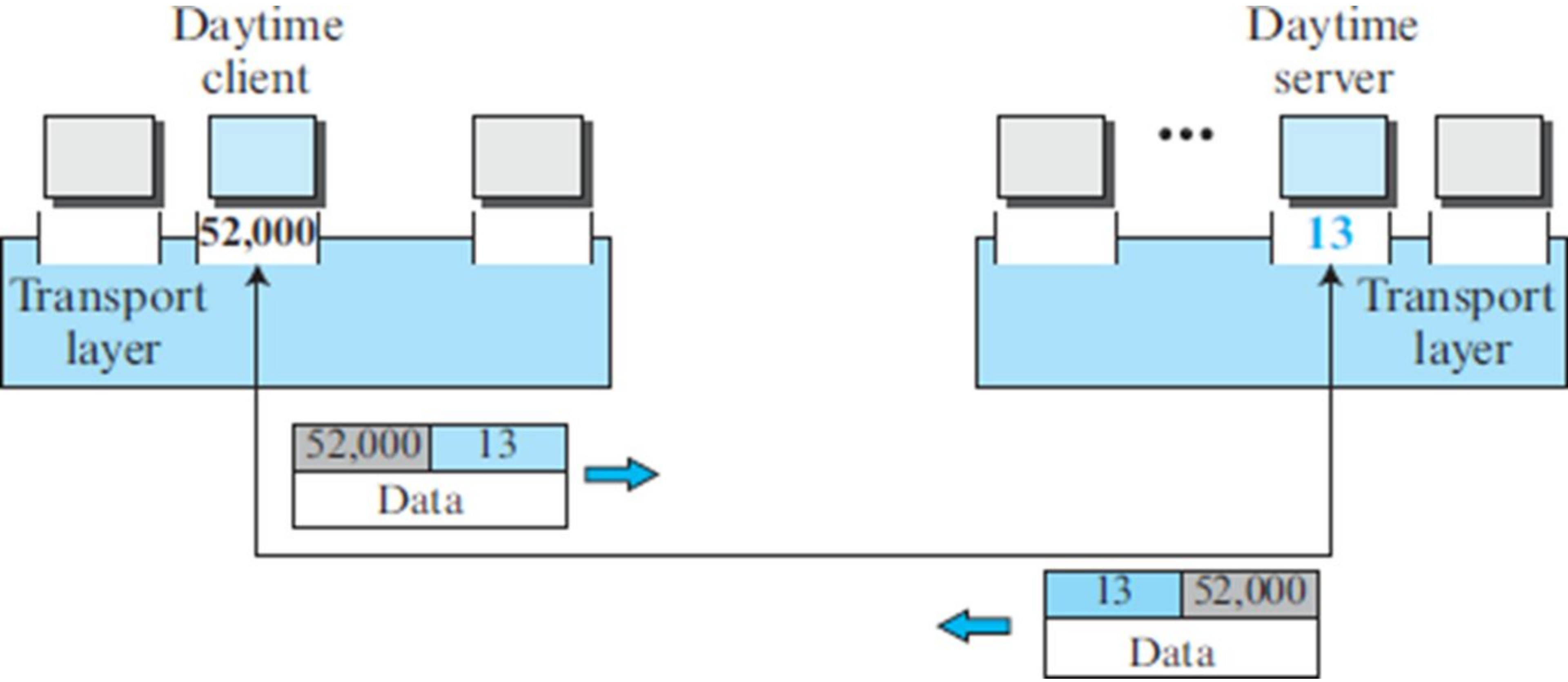
ii) Addressing:

Port Numbers:

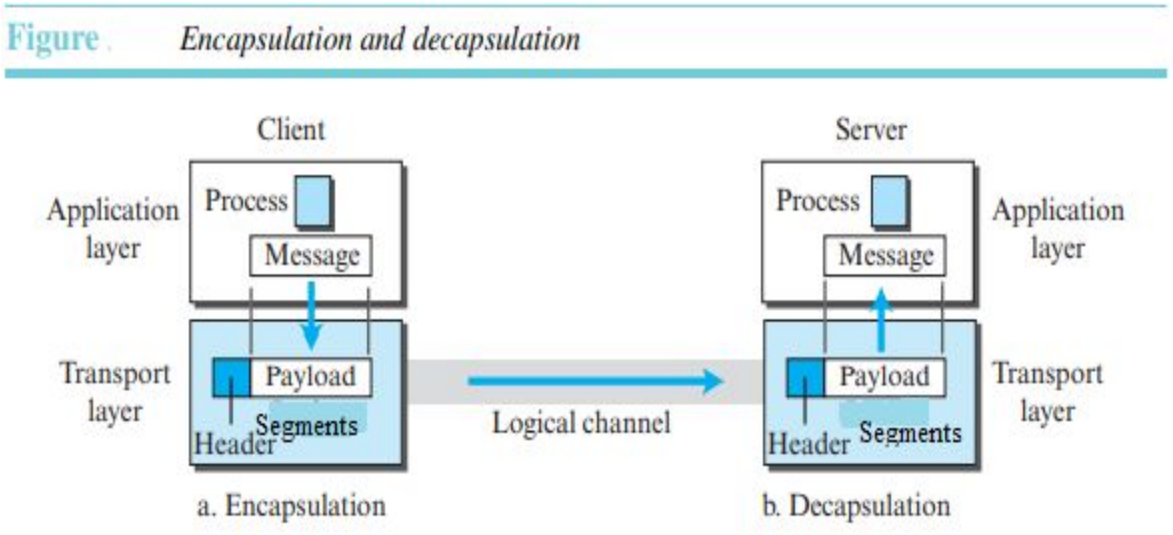
- For communication, we must define the local host, local process, remote host, and remote process.
- The local host and the remote host are defined using IP addresses .
- Each process running on a machine is defined using an identifier called port number.
- Each port number is a 16-bit number, ranging from 0 to 65535.
- The client program defines itself with a port number, called the ephemeral port number.
- Universal port numbers assigned to the servers are called well-known port numbers.
- The port numbers ranging from 0 to 1023 are called well-known port numbers and are reserved.

Figure *IP addresses versus port numbers*


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


Daytime client
Daytime server
52,000
13
Transport layer
Transport layer
52,000
13
Data
13
52,000
Data

iii) 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.

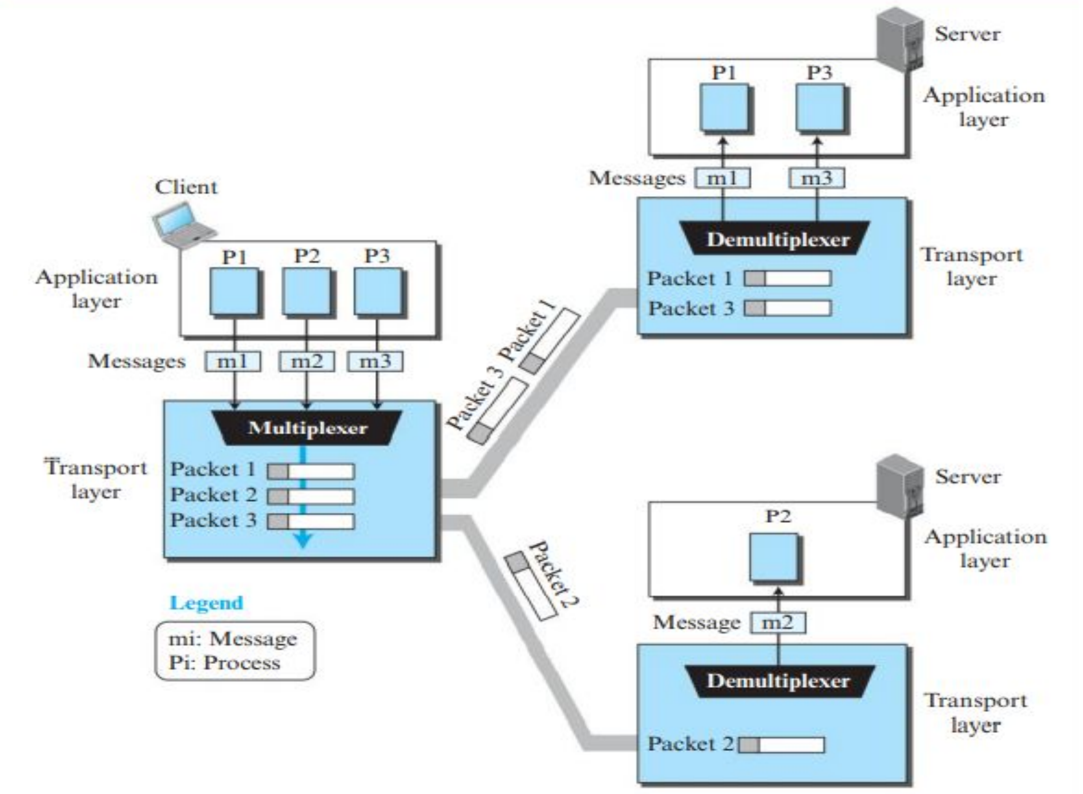
•When a process has a message to send, it passes the message to the transport layer along with a pair of socket addresses and some other pieces of information, which depend on the transport-layer protocol.
• The transport layer receives the data and adds the transport-layer header.
• The packets at the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

•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.

iv) Multiplexing and Demultiplexing:

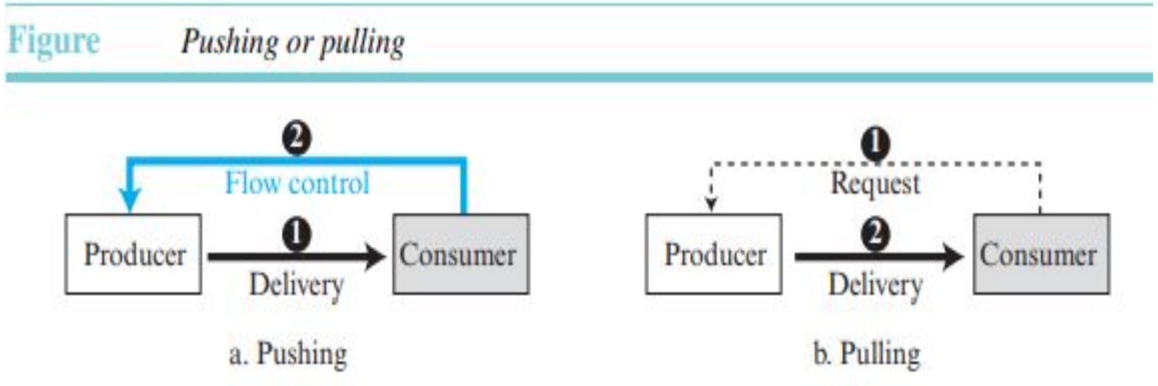
Figure *Multiplexing and demultiplexing*



•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.

v) Flow Control:

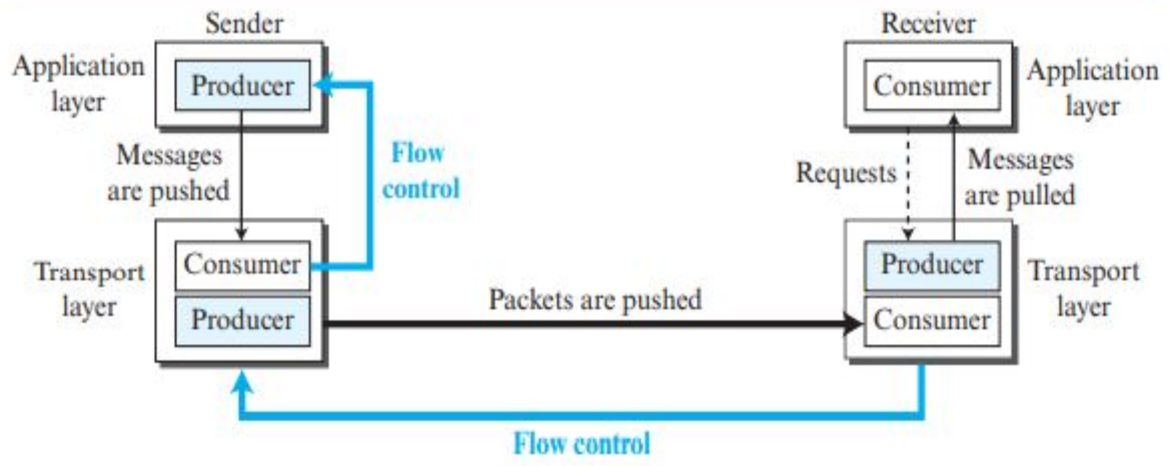


- When the consumer pulls the items, it requests them when it is ready. In this case, there is no need for flow control.
- When the producer pushes the items, the consumer needs to warn the producer to stop the delivery and to inform it when it is ready again to receive the items.

•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.

•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.

Figure . *Flow control at the transport layer*



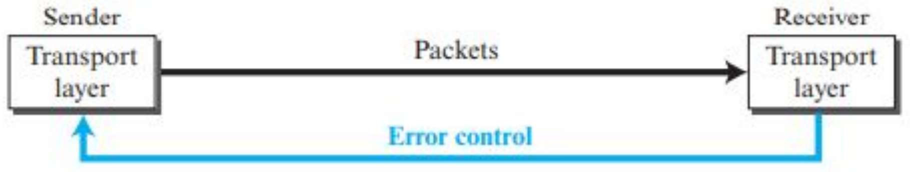
- Flow control can be implemented by the use of buffers.

vi) Error Control:

- Error control at the transport layer is responsible for:

    1. Detecting and discarding corrupted packets.
    2. Keeping track of lost and discarded packets and re-sending them.
    3. Recognizing duplicate packets and discarding them.
    4. Buffering out-of-order packets until the missing packets arrive.

Figure *Error control at the transport layer*

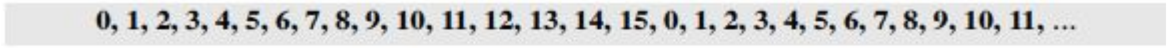


•We can add a field to the transport-layer packet to hold the sequence number of the packet.
•When a packet is corrupted or lost, the receiving transport layer can somehow inform the sending transport layer to resend that packet using the sequence number.
•For error control, the sequence numbers are 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 re-sends 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.

- *Sequence Numbers:*
    - Each packet is assigned a sequence number.
    - It helps to identify lost/duplicated/corrupted/out-of-order packets and thus to retransmit them if needed.
    - If m bits are used to represent the sequence number, then the possible values range from 0 to $2^m - 1$.
    - if m is 4, the only sequence numbers are 0 through 15.
    - the sequence numbers are modulo 2^m.



0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...

- *Acknowledgment:*
    - The receiver sends an acknowledgment (ACK) for each of a collection of packets that have arrived safe and sound.

- 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.
- Duplicate packets can be silently discarded by the receiver.
- Out-of-order packets can be either discarded or stored until the missing ones arrives.

Combination of Flow and Error Control:

-

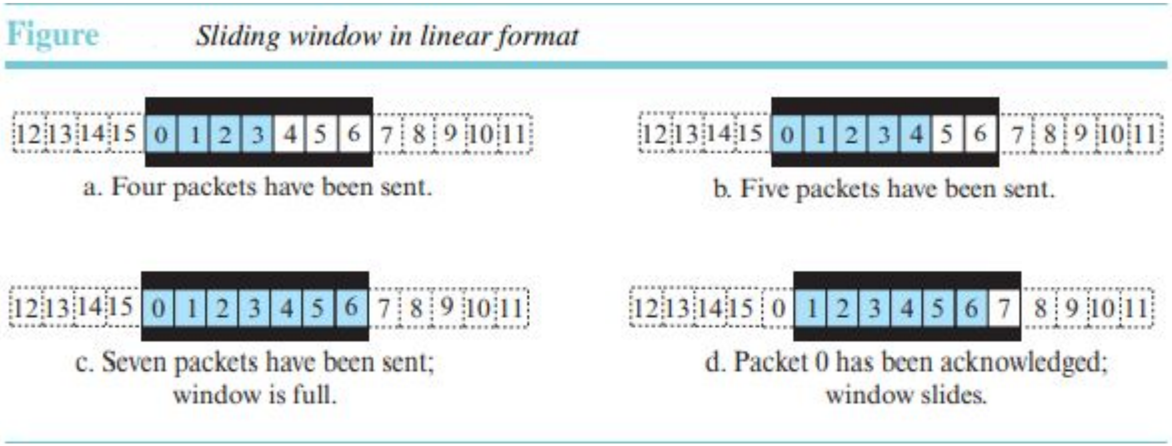


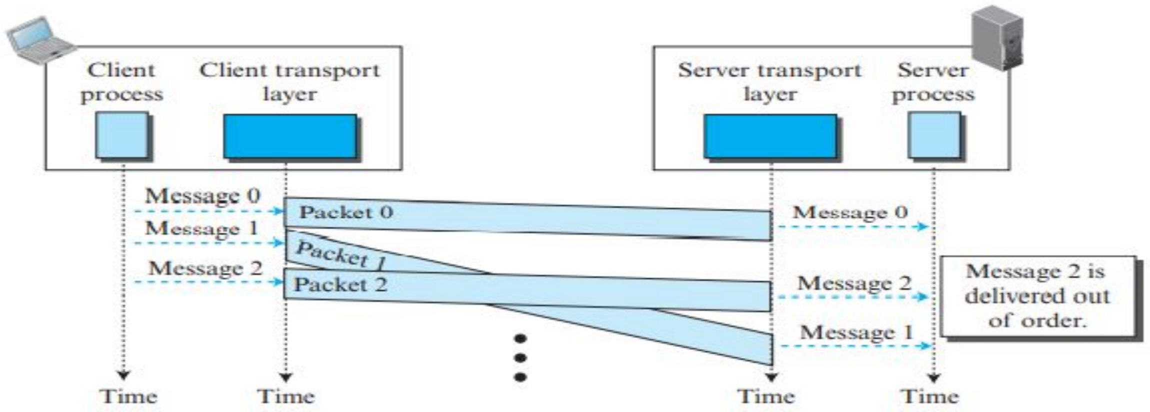
Figure Sliding window in linear format
12 13 14 15 0 1 2 3 4 5 6 7 8 9 10 11
a. Four packets have been sent.
12 13 14 15 0 1 2 3 4 5 6 7 8 9 10 11
b. Five packets have been sent.
12 13 14 15 0 1 2 3 4 5 6 7 8 9 10 11
c. Seven packets have been sent; window is full.
12 13 14 15 0 1 2 3 4 5 6 7 8 9 10 11
d. Packet 0 has been acknowledged; window slides.

Connectionless and Connection-Oriented Services:

•Connectionless service at the transport layer means independency between packets;

•Connection-oriented means dependency.

Figure *Connectionless service*



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.

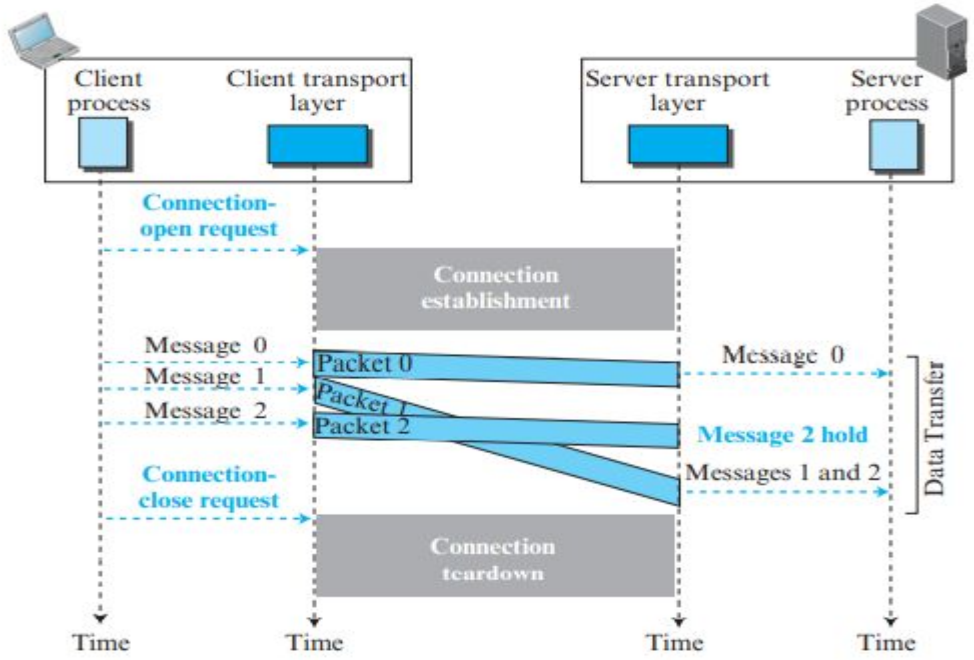
•The chunks are handed over to the connectionless transport protocol in order.

•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.
•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.

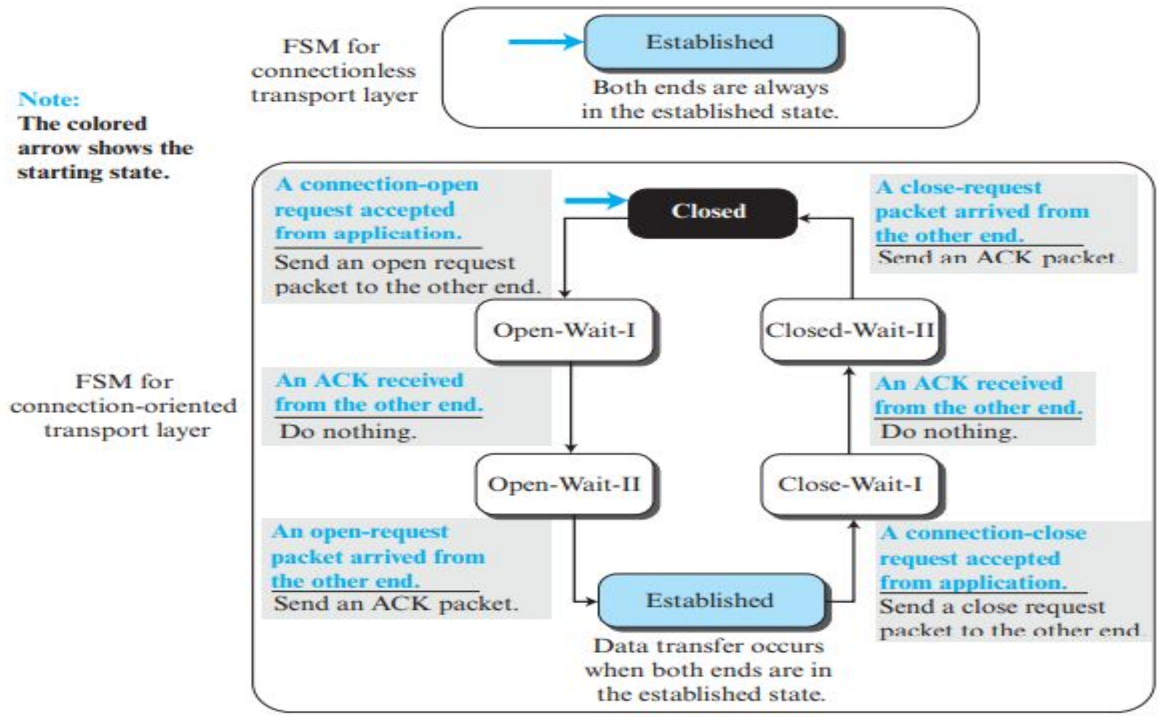
Figure *Connection-oriented service*



Finite State Machine:

- The behavior of a transport-layer protocol can be represented as a FSM.
- each transport layer (sender or receiver) is considered 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 and determining the next state (which can be the same as the current state).
- One of the states must be defined as the initial state
- rounded-corner rectangles to show states, color text to show events, and regular black text to show actions. A horizontal line is used to separate the event from the actions and arrows represent the movement to the next state.

Figure *Connectionless and connection-oriented service represented as FSMs*


FSM for connectionless transport layer
Established
Both ends are always in the established state.
Note: The colored arrow shows the starting state.
FSM for connection-oriented transport layer
A connection-open request accepted from application.
Send an open request packet to the other end.
Closed
A close-request packet arrived from the other end.
Send an ACK packet.
Open-Wait-I
Closed-Wait-II
An ACK received from the other end.
Do nothing.
An ACK received from the other end.
Do nothing.
Open-Wait-II
Close-Wait-I
An open-request packet arrived from the other end.
Send an ACK packet.
Established
A connection-close request accepted from application.
Send a close request packet to the other end.
Data transfer occurs when both ends are in the established state.

TRANSPORT-LAYER PROTOCOLS

i) Simple Protocol:

- a simple connectionless protocol with neither flow nor error control

Figure *Simple protocol*

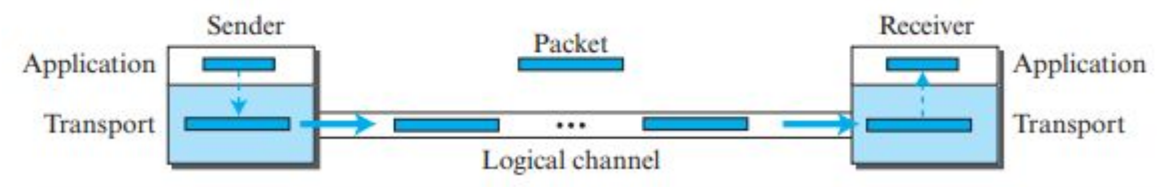


Figure *FSMs for the simple protocol*

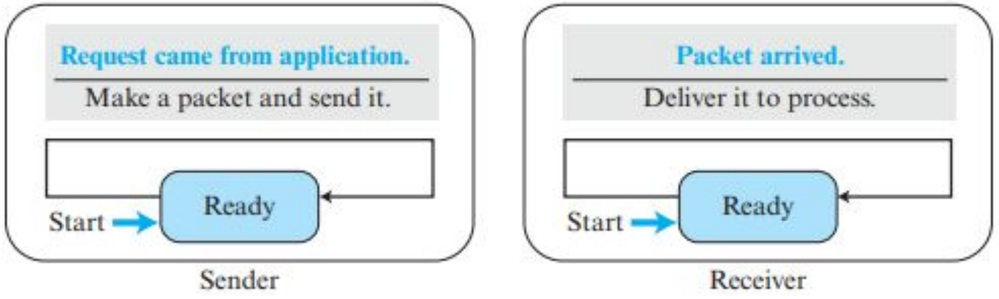
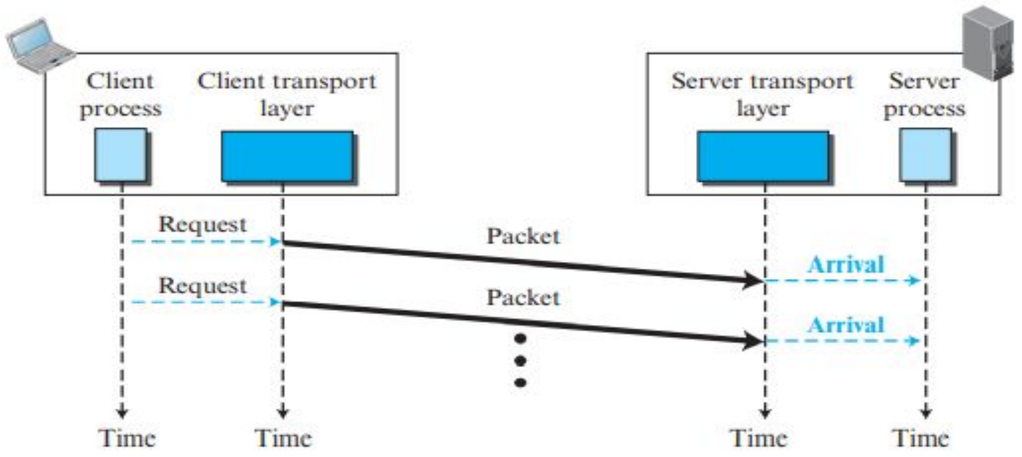
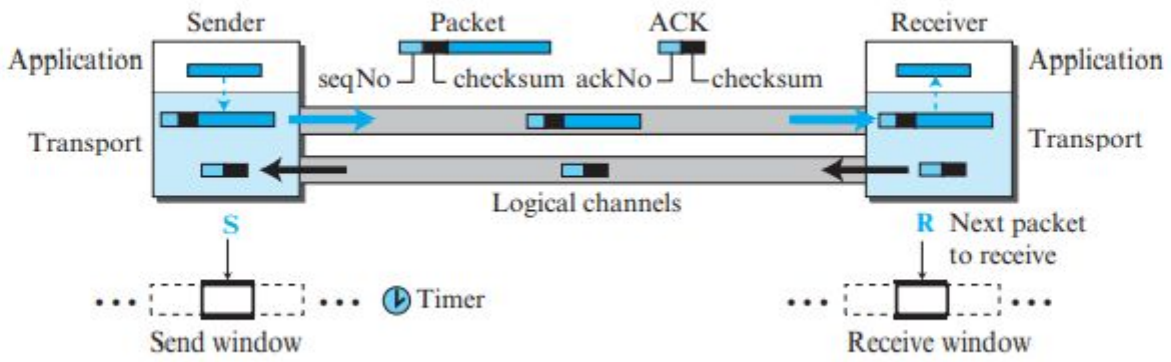


Figure *Flow diagram for Example 3.3*



ii) Stop-and-Wait Protocol:

Figure 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.
Sequence numbers are always modulo arithmetic.
the acknowledgment number always announces, in modulo-2 arithmetic, the sequence number of the next packet expected.

•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.

Figure *FSM for the Stop-and-Wait protocol*

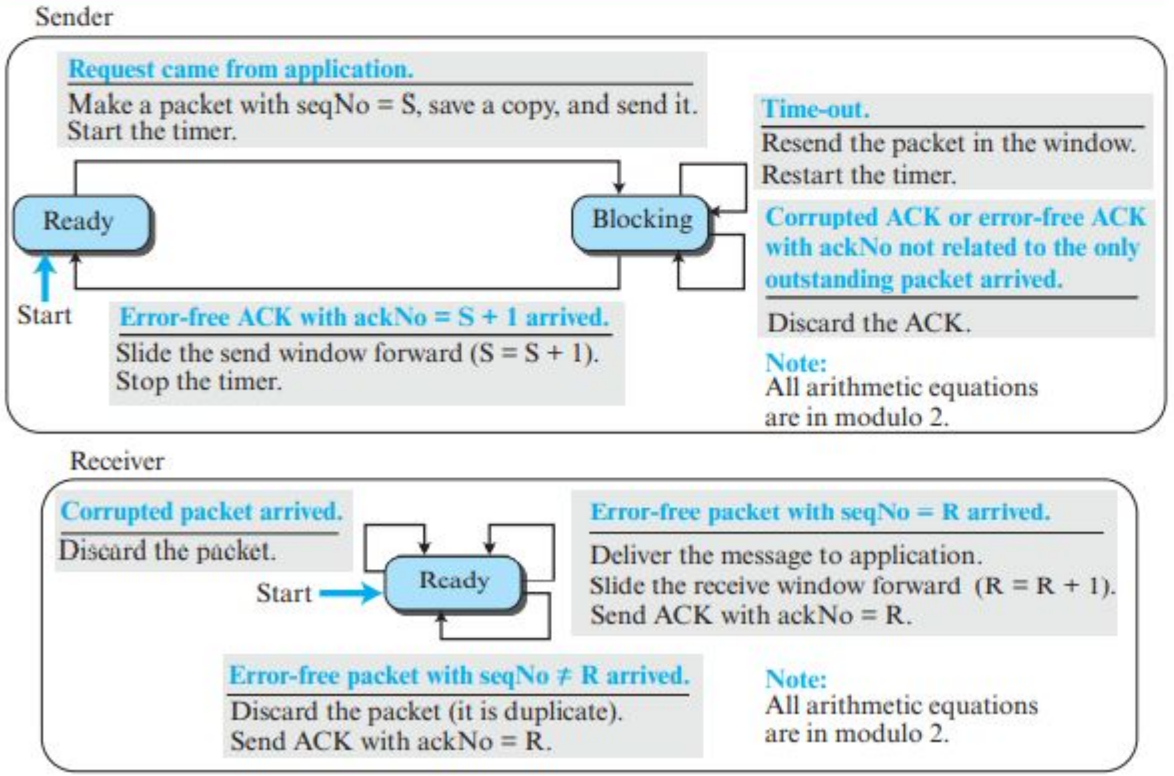
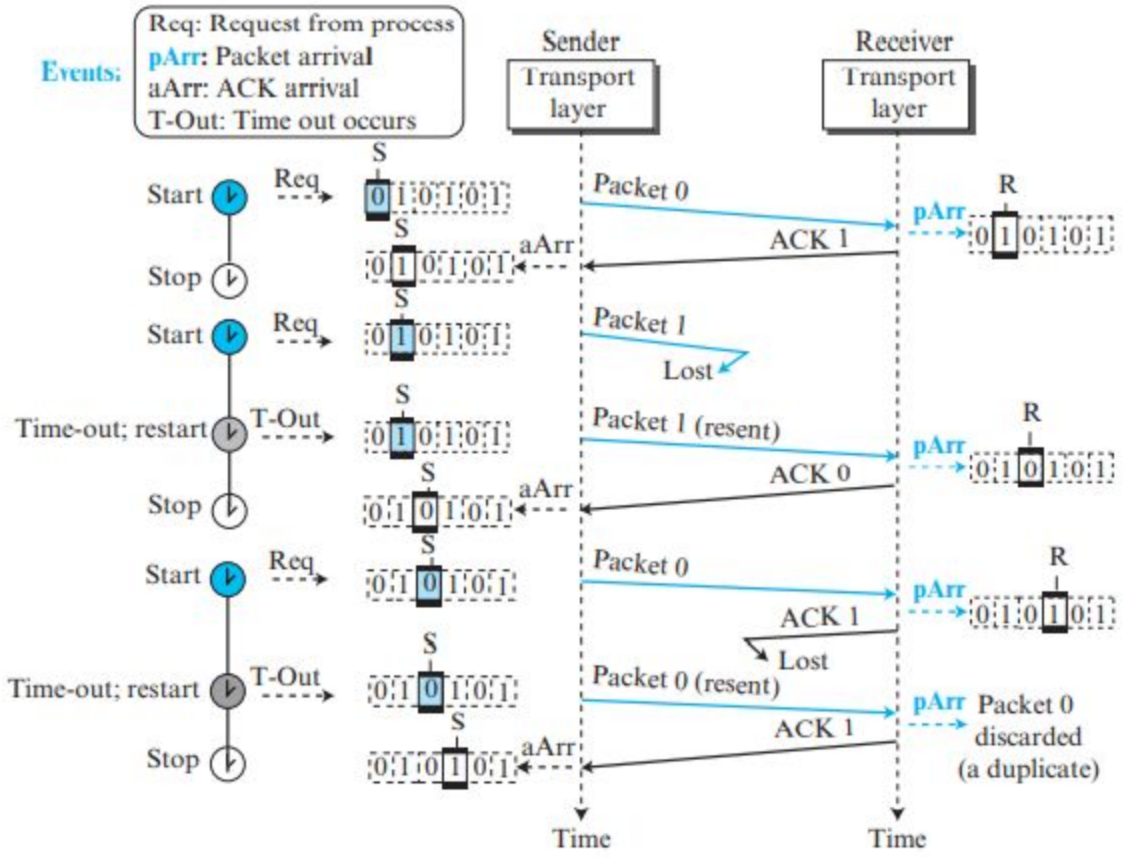

Sender
Request came from application.
Make a packet with seqNo = S, save a copy, and send it. Start the timer.
Time-out.
Resend the packet in the window. Restart the timer.
Ready
Blocking
Corrupted ACK or error-free ACK with ackNo not related to the only outstanding packet arrived.
Discard the ACK.
Start
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.
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.
Start
Ready
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.

Figure *Flow diagram*



Let us check the efficiency of stop and wait protocol:

- Suppose the transmission rate of the channel is 1 Mbps, and 1 bit takes 20 milliseconds to make a round trip. If the system data packets are 1,000 bits in length, what is the utilization percentage of the channel?
  -

$$(1 \times 10^6) \times (20 \times 10^{-3}) = 20{,}000 \text{ bits.}$$

This is the bandwidth-delay product. Here the system sends only 1,000 bits. So the channel utilization is only 1,000/20,000, or 5 percent.

- What is the utilization percentage of the above said link if we have a protocol that can send up to 15 packets before stopping and worrying about the acknowledgments?

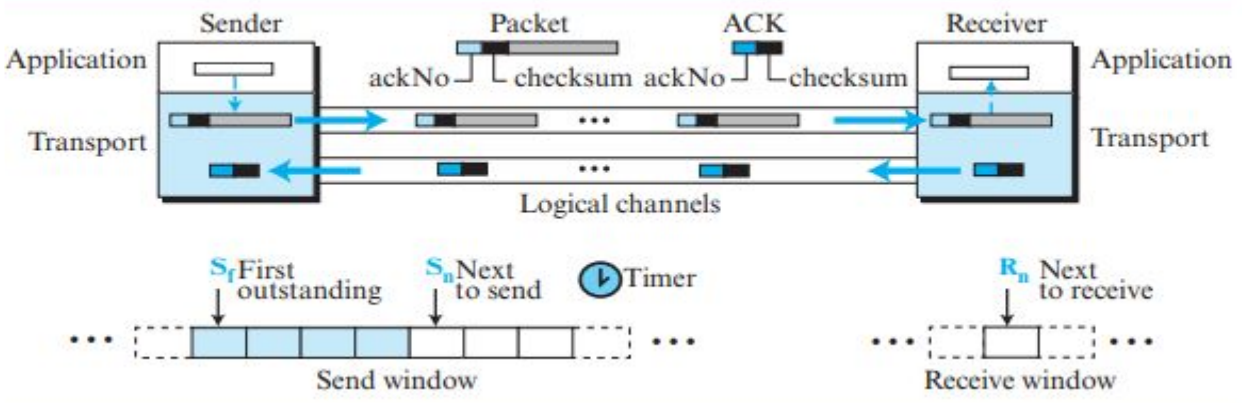
Pipelining

- It is the process where a task is often begun before the previous task has ended.
- There is no pipelining in the Stop-and-Wait protocol
- Through pipelining, several packets can be sent before a sender receives feedback about the previous packets.
- Pipelining improves the efficiency of the transmission.
- Go-back-N and Selective Repeat protocols implement pipelining.

iii) Go-Back-N Protocol (GBN):

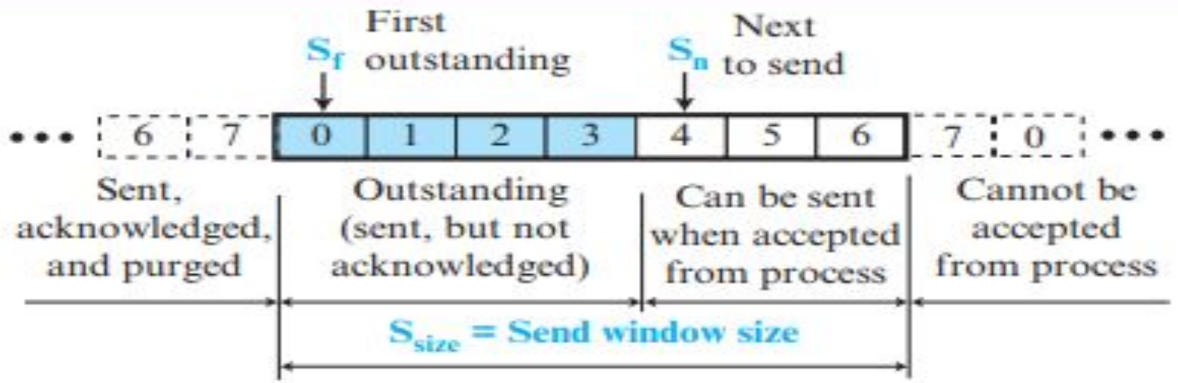
- To improve the efficiency of transmission (to fill the pipe), multiple packets must be in transition while the sender is waiting for acknowledgment.
- The key to Go-back-N is that we can send several packets before receiving acknowledgments, but the receiver can only buffer one packet.

Figure *Go-Back-N protocol*



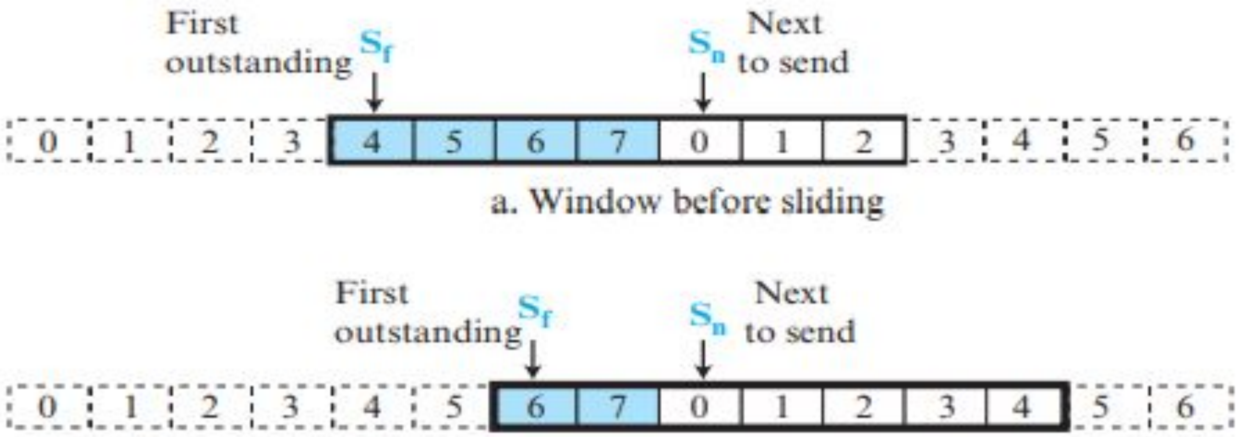
- Sequence Numbers: modulo 2^m, where m is the size of the sequence number field in bits.
- Acknowledgment Numbers: It is cumulative and defines the sequence number of the next packet expected to arrive.
- Send Window:
    - The send window is an abstract concept defining an imaginary box of maximum size = $2^m - 1$ with three variables: S_f, S_n, and S_{size}.
    - The variable S_f defines the sequence number of the first (oldest) outstanding packet.
    - The variable S_n holds the sequence number that will be assigned to the next packet to be sent.
    - The variable S_{size} defines the size of the window, which is fixed in this protocol.

Figure *Send window for Go-Back-*N



S_f First outstanding
S_n Next to send
6
7
0
1
2
3
4
5
6
7
0
Sent, acknowledged, and purged
Outstanding (sent, but not acknowledged)
Can be sent when accepted from process
Cannot be accepted from process
S_{size} = Send window size

Figure *Sliding the send window*



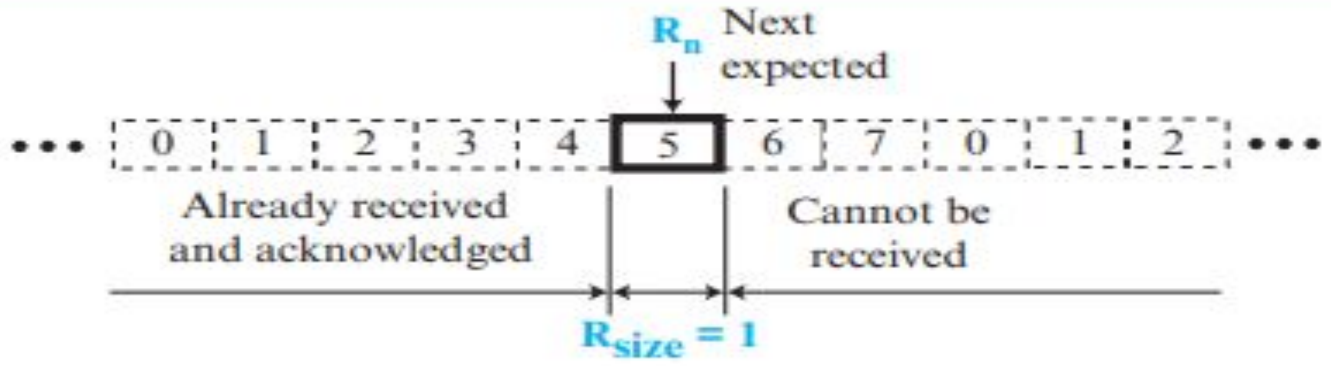
a. Window before sliding

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

The send window can slide one or more slots when an error-free ACK with ackNo greater than or equal S_f and less than S_n (in modular arithmetic) arrives.

- Receive Window:
    - 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.
    - The sequence numbers to the left of the window belong to the packets already received and acknowledged;
    - the sequence numbers to the right of this window define the packets that cannot be received.
    - Any received packet with a sequence number in these two regions is discarded.
    - Only a packet with a sequence number matching the value of Rn is accepted and acknowledged.
    - The window also slides, but only one slot at a time.
    - When a correct packet is received, the window slides, $R_n = (R_n + 1)$ modulo 2^m.

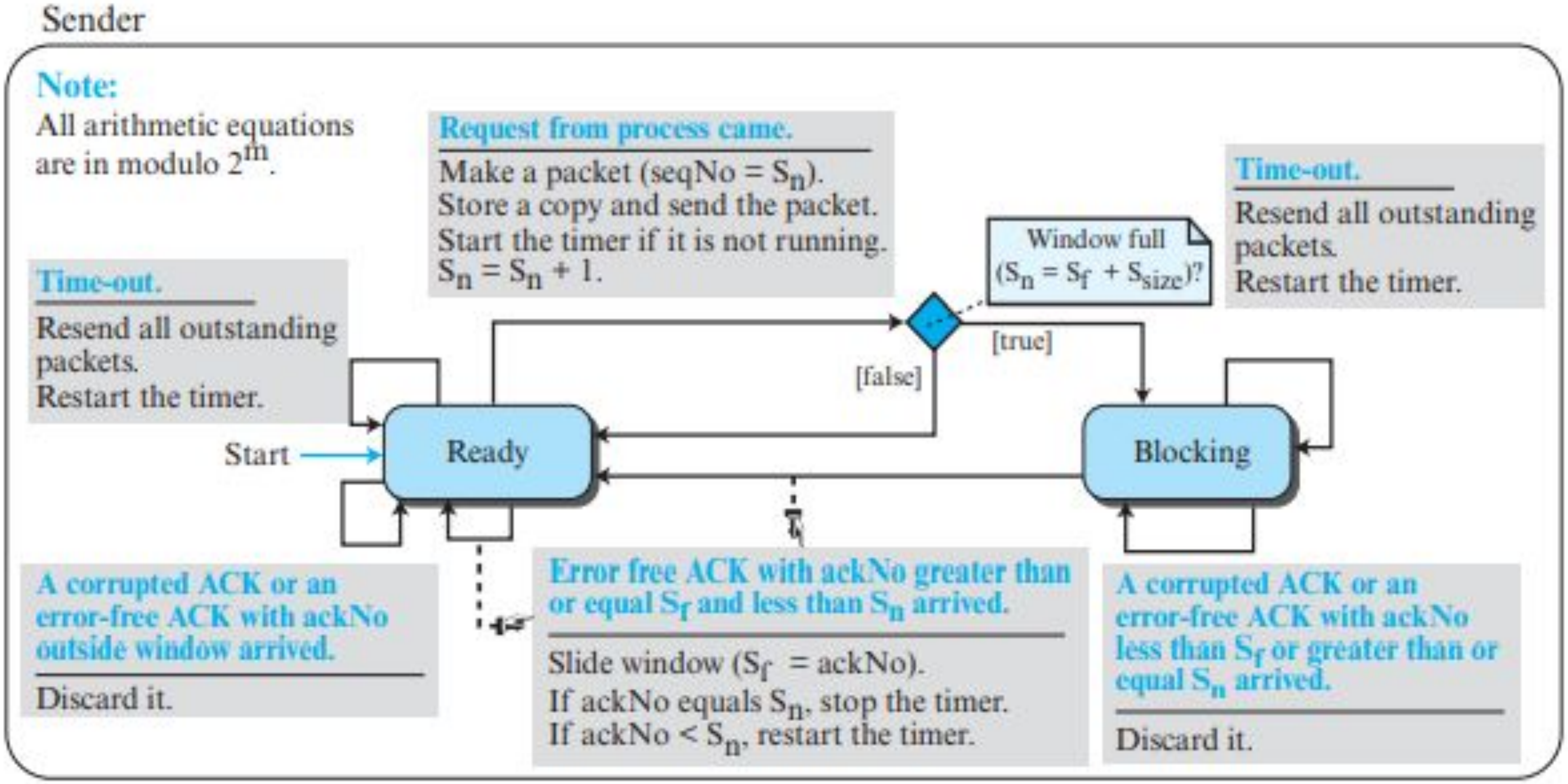
Figure *Receive window for Go-Back-*N



Timers:

- When the timer for the first outstanding packet expires, it resend all outstanding packets.
- On a time-out, the machine goes back N locations and resends all packets. So this protocol is called Go-Back-N.

FSMs:


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 ($S_n = S_f + S_{size}$)?
Time-out.
Resend all outstanding packets.
Restart the timer.
Time-out.
Resend all outstanding packets.
Restart the timer.
[true]
[false]
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.

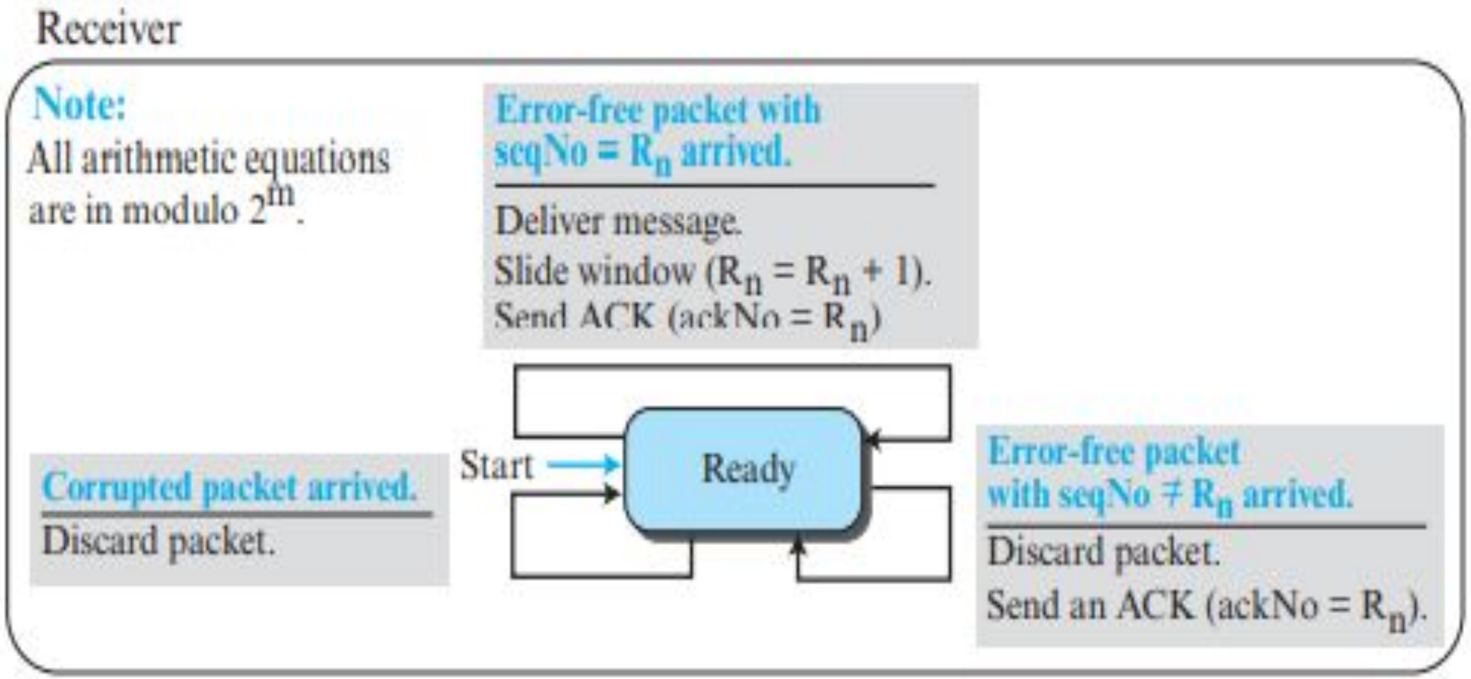
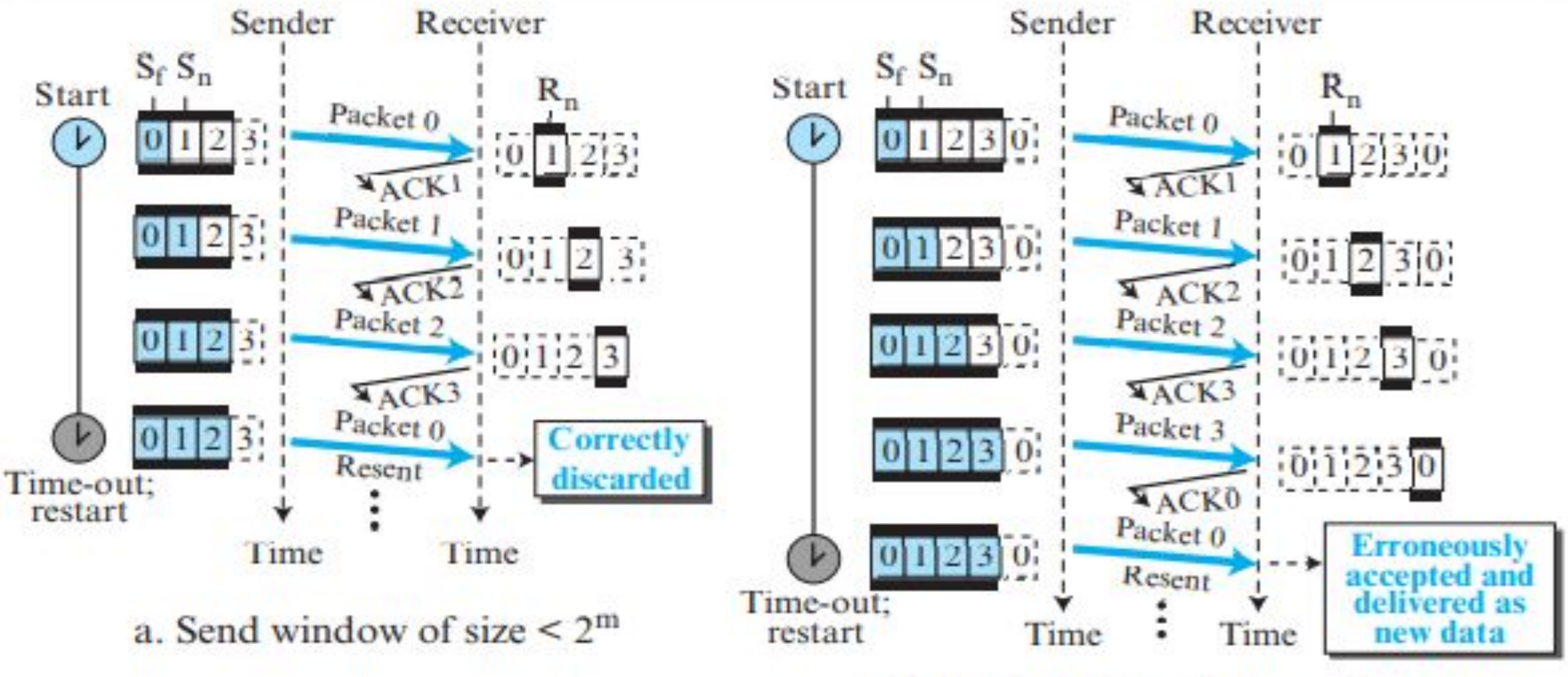

Receiver
Note:
All arithmetic equations are in modulo 2^m.
Error-free packet with seqNo = R_n arrived.
Deliver message.
Slide window ($R_n = R_n + 1$).
Send ACK ($ackNo = R_n$)
Start
Ready
Corrupted packet arrived.
Discard packet.
Error-free packet with seqNo ≠ R_n arrived.
Discard packet.
Send an ACK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

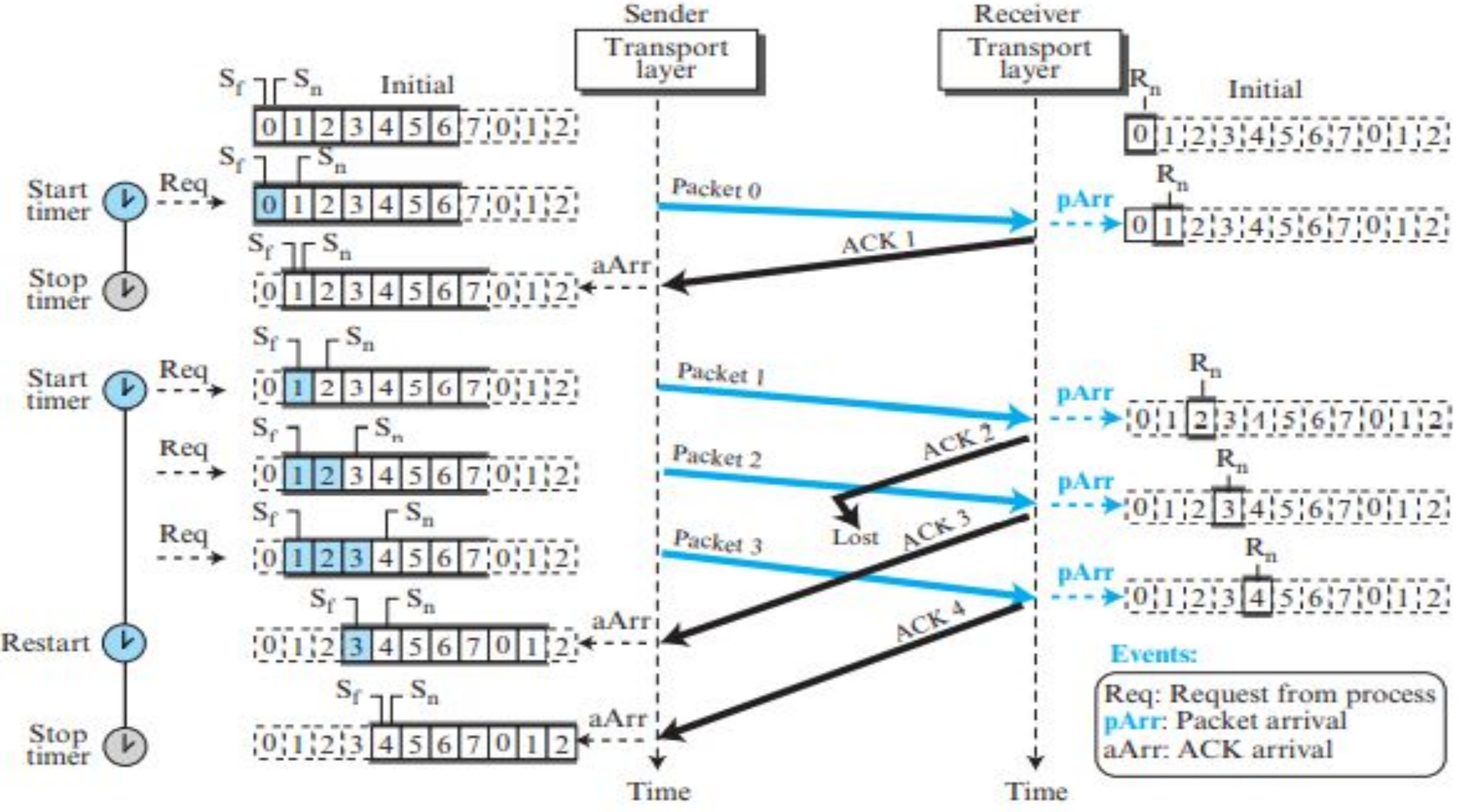
Figure *Send window size for Go-Back-N*



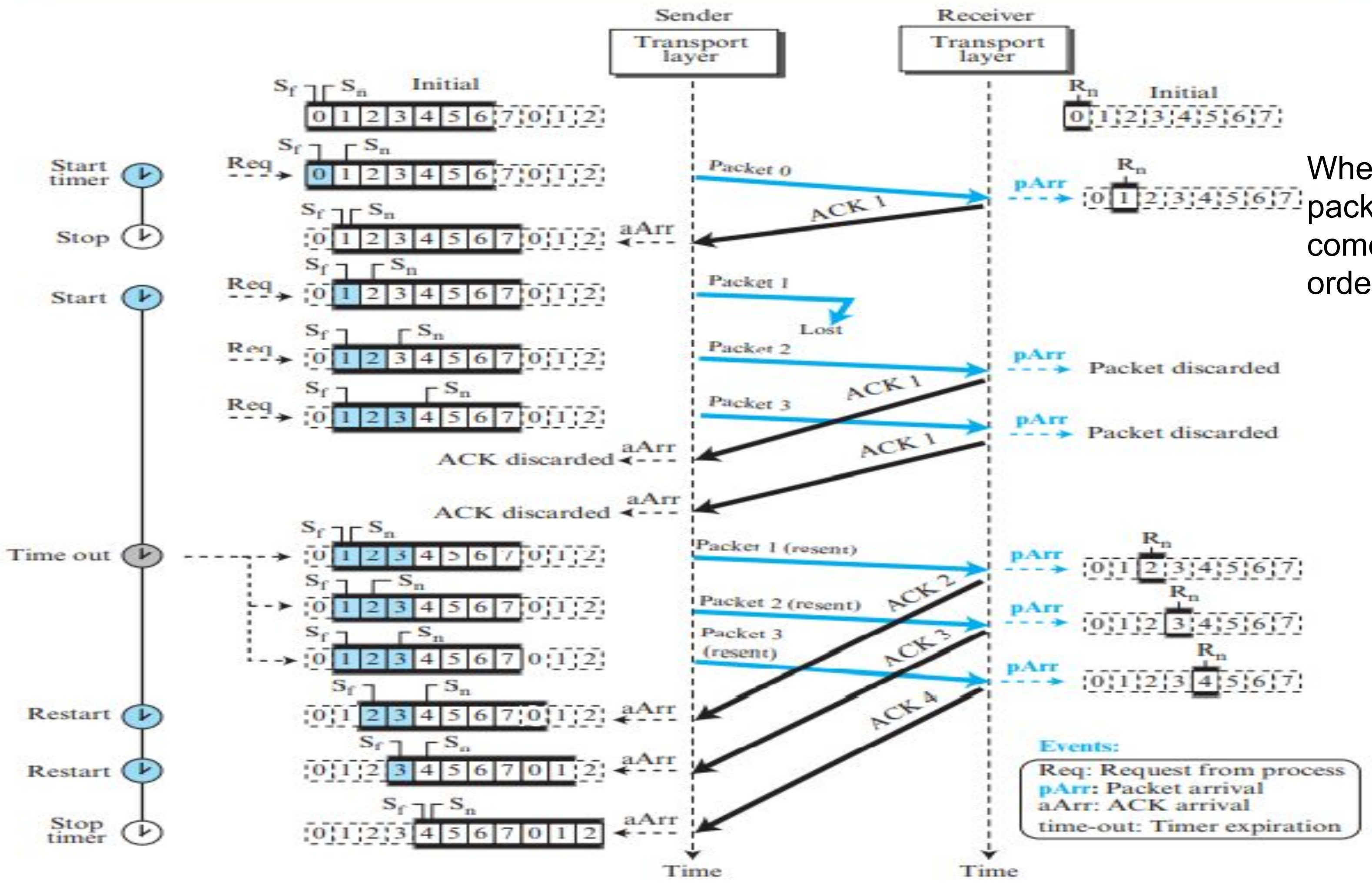
a. Send window of size $< 2^m$

b. Send window of size $= 2^m$

Figure *Flow diagram for Example 3.7*



When ACKs are delayed or lost



When packets come out of order

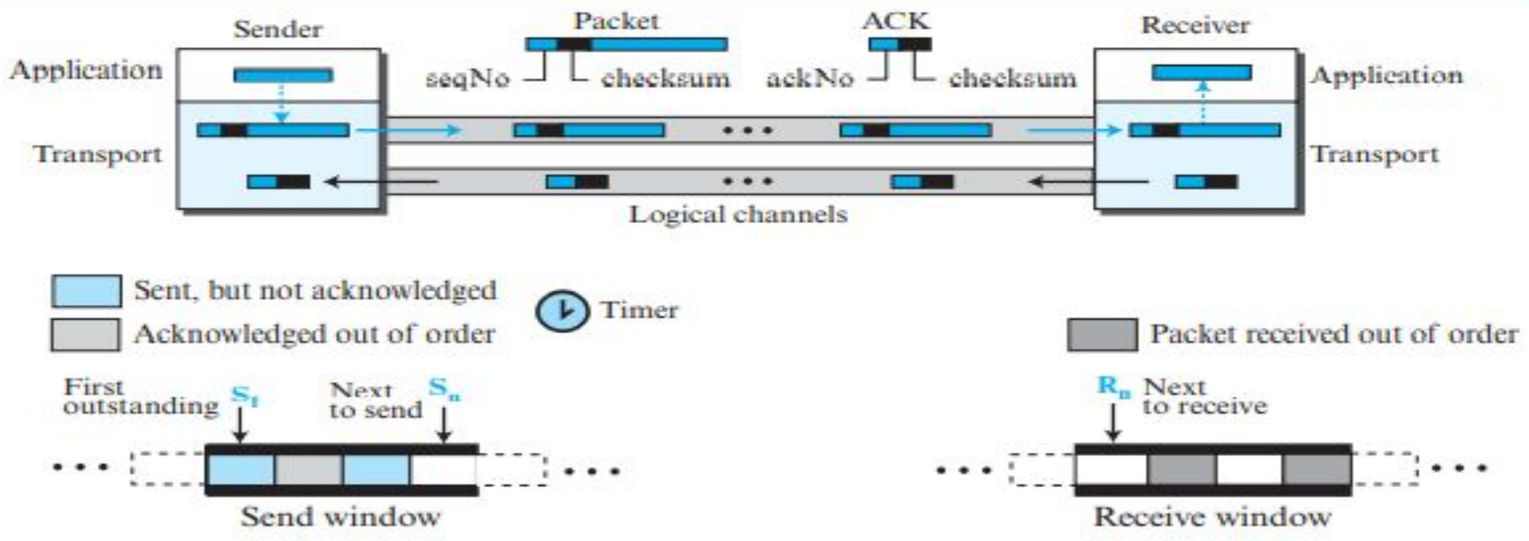
Disadvantages of Go-back-N:

- The receiver keeps track of only one variable, and there is no need to buffer out-of-order packets; they are simply discarded.
- 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.

iv) Selective Repeat (SR) protocol:

- It resends only selective packets, those that are actually lost.
- Only the erroneous or lost packets are retransmitted, while correct packets are received and buffered.

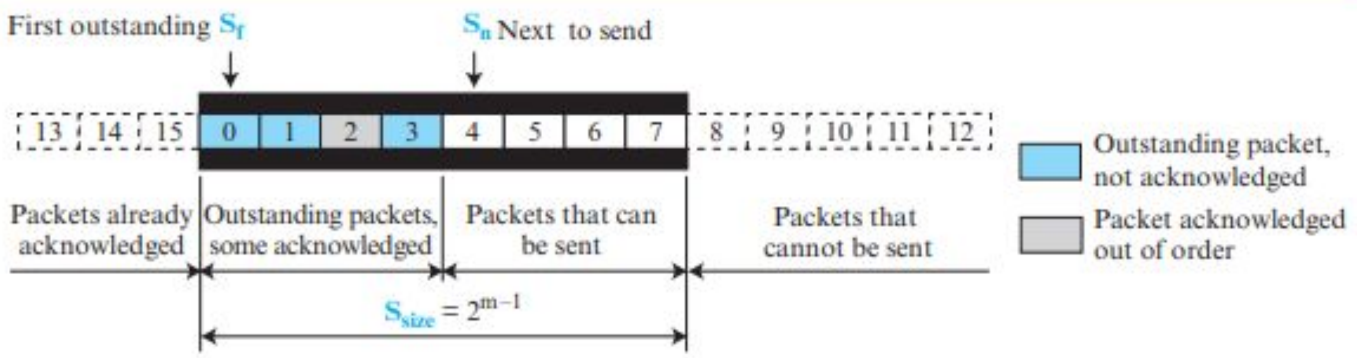
Figure 3.31 *Outline of Selective-Repeat*



Windows:

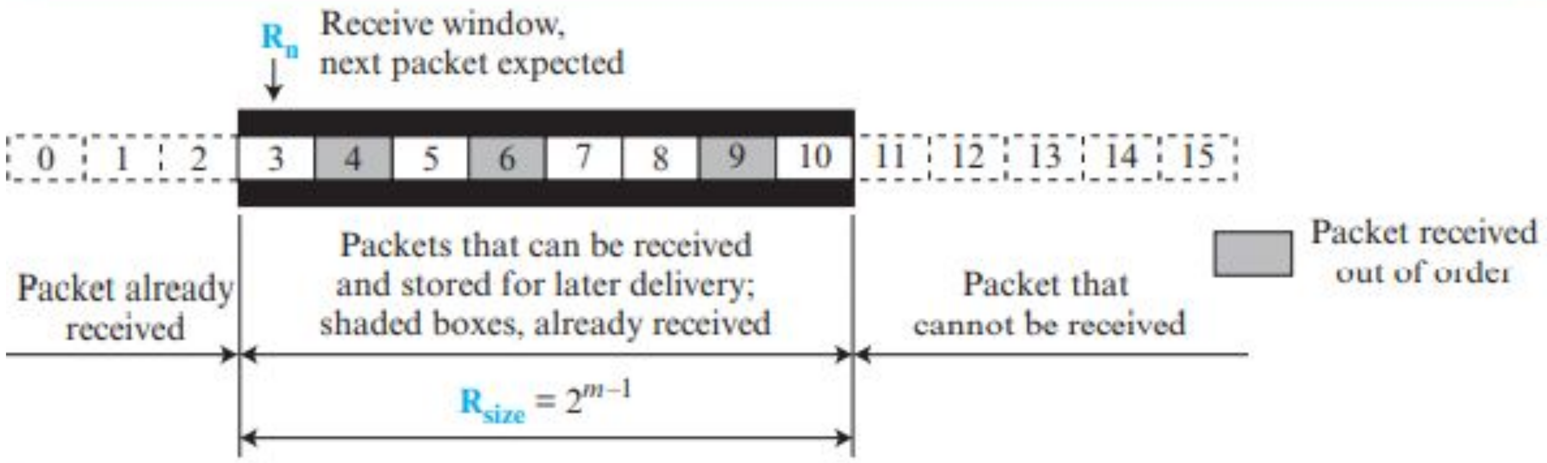
- It uses two windows: a send window and a receive window.
- the maximum size of the send window is 2^{m-1}.
- the receive window is the same size as the send window.

Figure 3.32 *Send window for Selective-Repeat protocol*



First outstanding S_f
S_n Next to send
13
14
15
0
1
2
3
4
5
6
7
8
9
10
11
12
Packets already acknowledged
Outstanding packets, some acknowledged
Packets that can be sent
Packets that cannot be sent
$S_{size} = 2^{m-1}$
Outstanding packet, not acknowledged
Packet acknowledged out of order

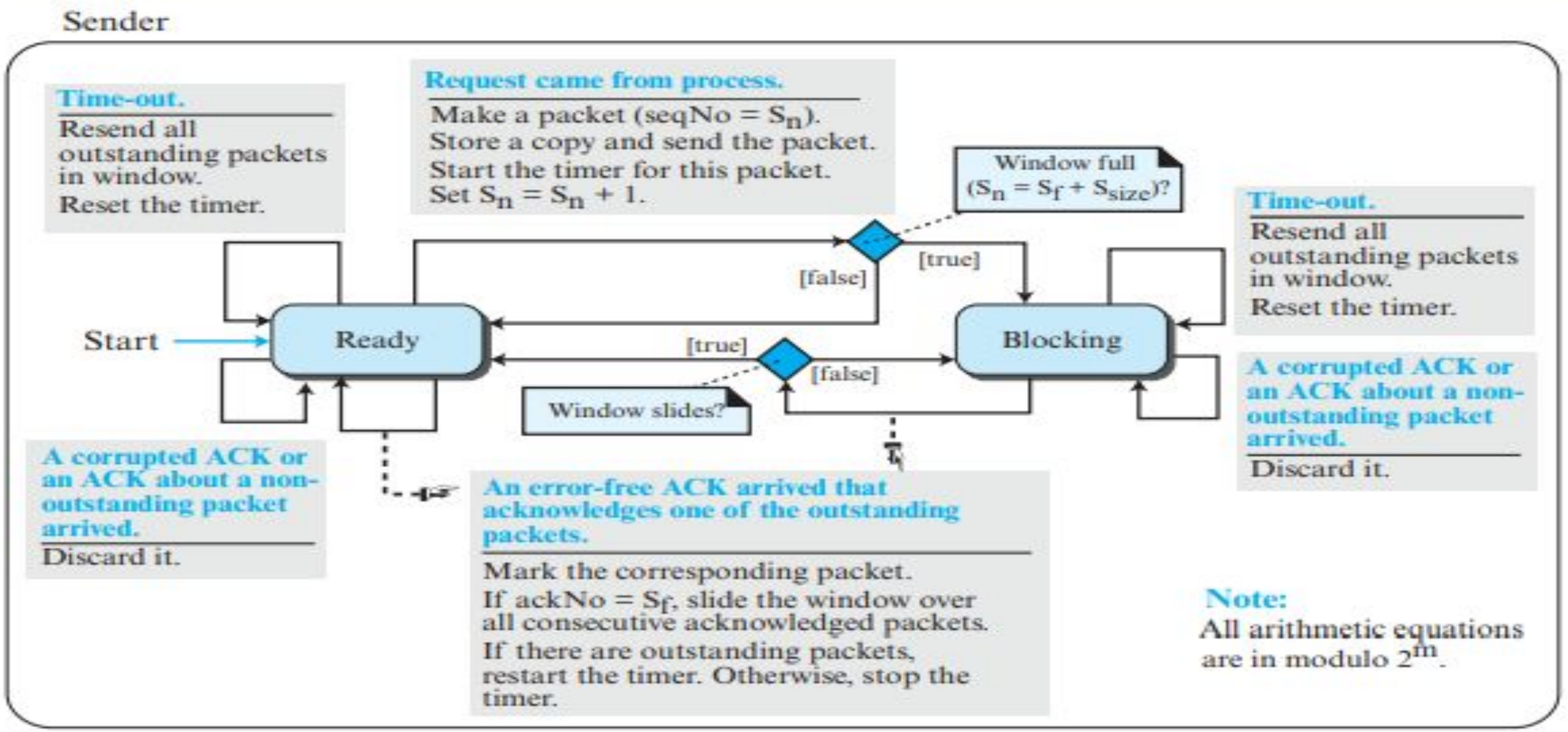
Figure 3.33 *Receive window for Selective-Repeat protocol*



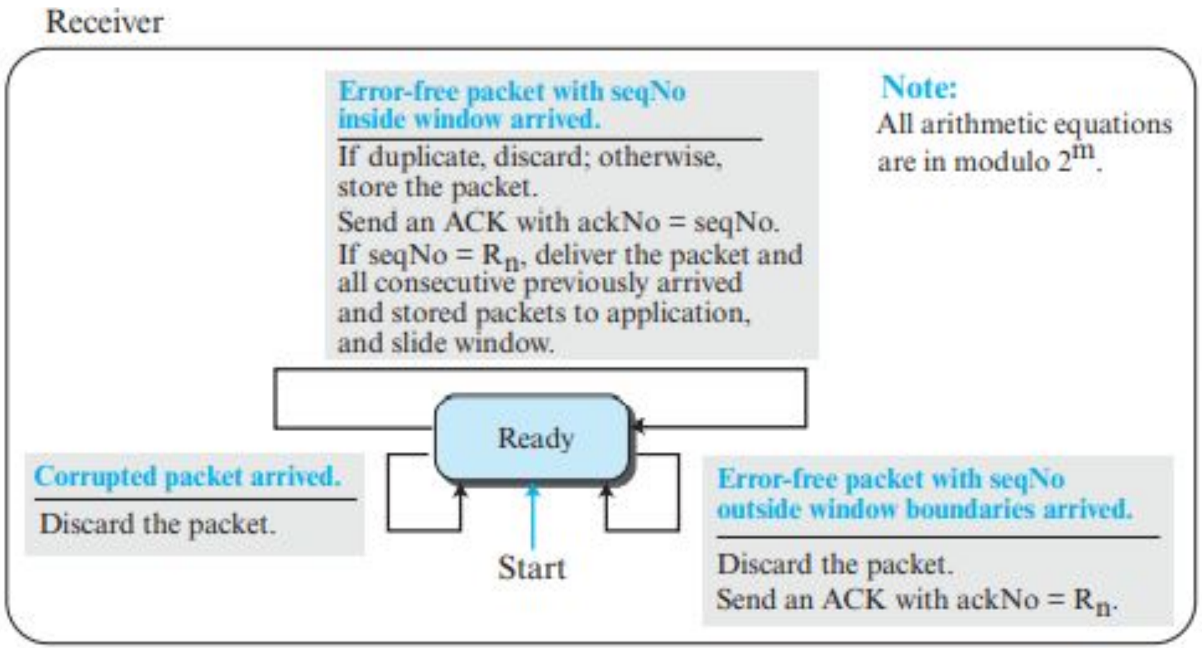
R_n
Receive window, next packet expected
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Packet already received
Packets that can be received and stored for later delivery; shaded boxes, already received
Packet that cannot be received
Packet received out of order
$R_{size} = 2^{m-1}$

In the Selective-Repeat protocol, an acknowledgment number defines the sequence number of the error-free packet received.

Finite State Machines:



Sender
Time-out.
Resend all outstanding packets in window.
Reset the timer.
Request came from process.
Make a packet (seqNo = S_n).
Store a copy and send the packet.
Start the timer for this packet.
Set $S_n = S_n + 1$.
Window full ($S_n = S_f + S_{size}$)?
[false]
[true]
Time-out.
Resend all outstanding packets in window.
Reset the timer.
Start
Ready
Blocking
[true]
[false]
Window slides?
A corrupted ACK or an ACK about a non-outstanding packet arrived.
Discard it.
A corrupted ACK or an ACK about a non-outstanding packet arrived.
Discard it.
An error-free ACK arrived that acknowledges one of the outstanding packets.
Mark the corresponding packet.
If ackNo = S_f, slide the window over all consecutive acknowledged packets.
If there are outstanding packets, restart the timer. Otherwise, stop the timer.
Note:
All arithmetic equations are in modulo 2^m.


Receiver
Error-free packet with seqNo inside window arrived.
If duplicate, discard; otherwise, store the packet.
Send an ACK with ackNo = seqNo.
If seqNo = R_n, deliver the packet and all consecutive previously arrived and stored packets to application, and slide window.
Note:
All arithmetic equations are in modulo 2^m.
Ready
Corrupted packet arrived.
Discard the packet.
Start
Error-free packet with seqNo outside window boundaries arrived.
Discard the packet.
Send an ACK with ackNo = R_n.

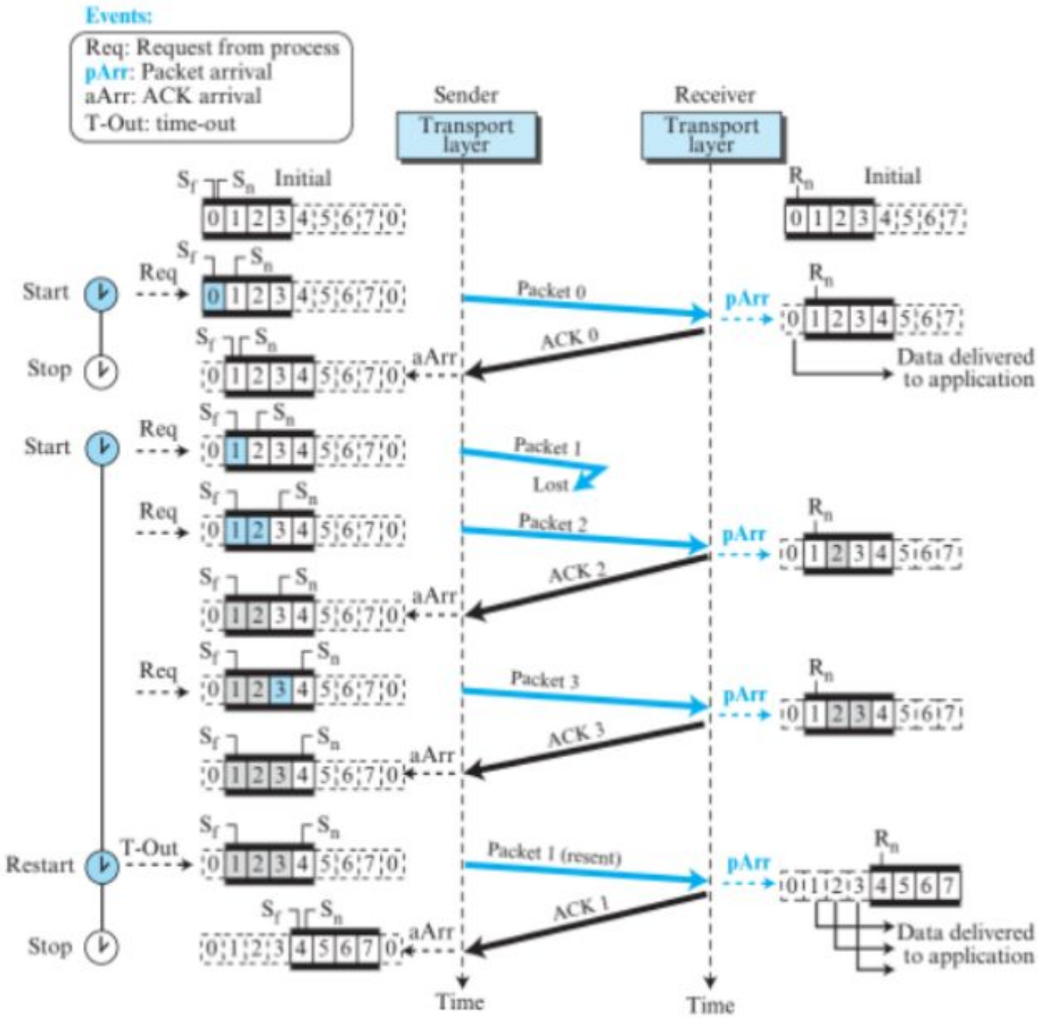
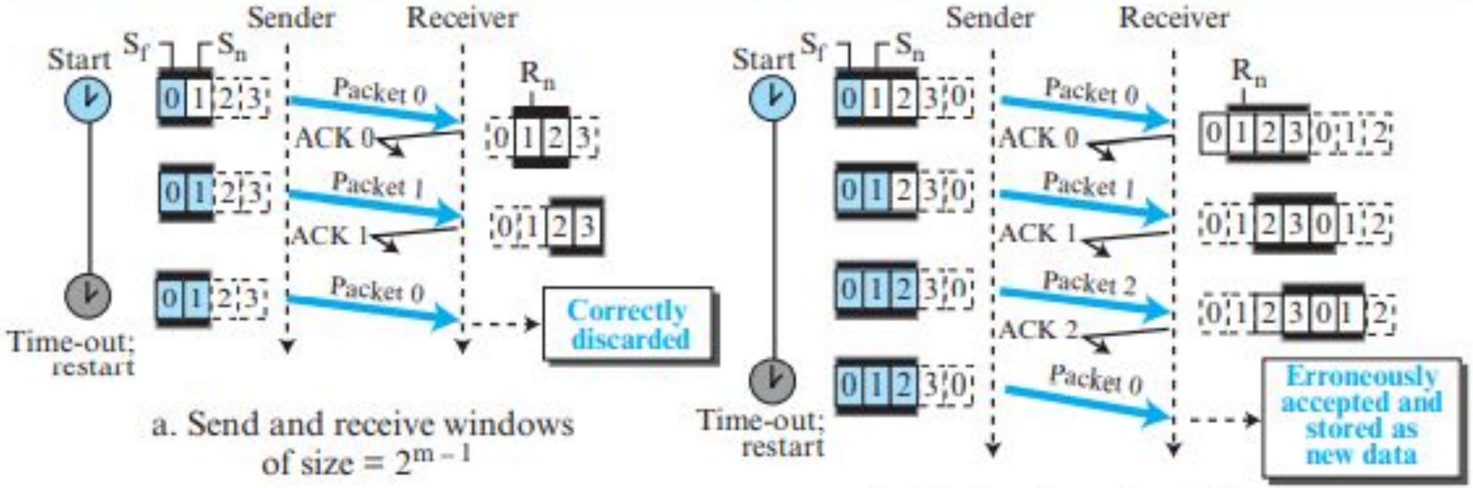

Events:
Req: Request from process
pArr: Packet arrival
aArr: ACK arrival
T-Out: time-out
Sender
Transport layer
Receiver
Transport layer
Initial
Start
Stop
Restart
Req
T-Out
Packet 0
ACK 0
Packet 1
Lost
Packet 2
ACK 2
Packet 3
ACK 3
Packet 1 (resent)
ACK 1
aArr
pArr
Data delivered to application
Time

Figure *Selective-Repeat, window size*



a. Send and receive windows of size = 2^{m-1}

b. Send and receive windows of size > 2^{m-1}

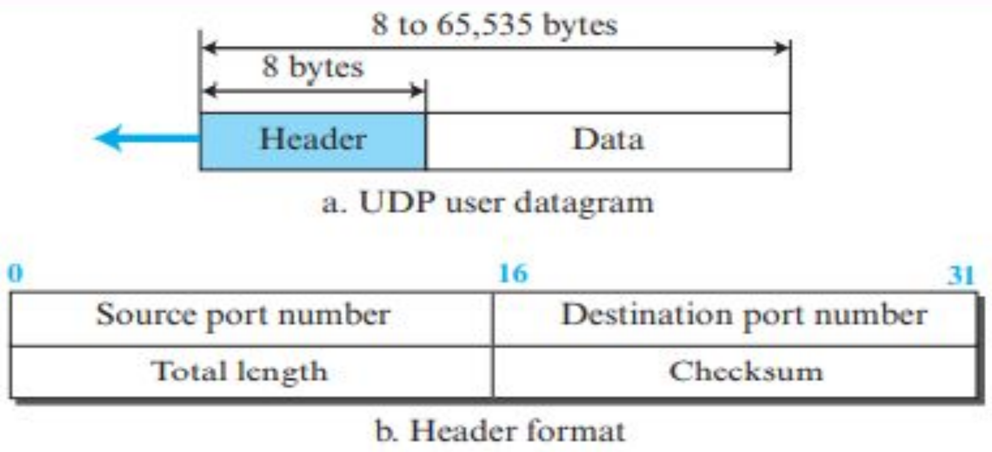
Bidirectional Protocols: Piggybacking

- Data packets normally flow in both directions: from client to server and from server to client.
- Thus 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.

USER DATAGRAM PROTOCOL (UDP) segment format:

- is a connectionless, unreliable transport protocol.

Figure 3.39 *User datagram packet format*



How checksum is calculated at the sender side?

- It is used for error detection.
- 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 is the checksum.
- Suppose that we have the following three 16-bit words:

  0110011001100000

  0101010101010101

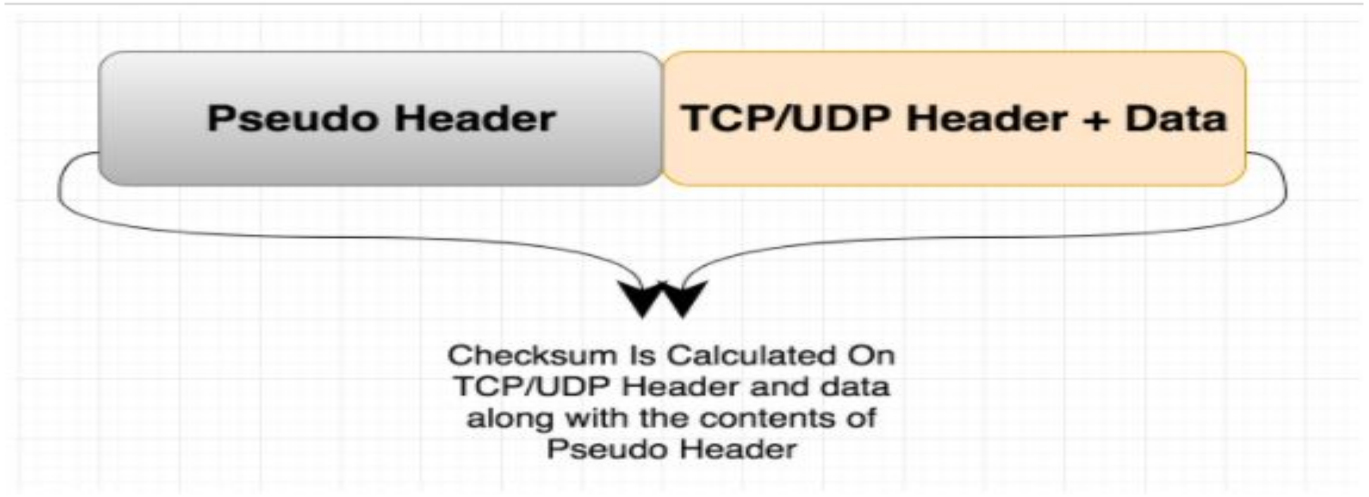
  1000111100001100

- The sum of these 16-bit words is 0100101011000010.
- Then take the 1s complement of the sum; thus the 1s complement of the sum 0100101011000010 is 1011010100111101.
- This is the checksum.(1011010100111101)

How error is detected at the receiver side?

- At the receiver, all four 16-bit words are added, including the checksum.
- If the output is all 1s ie., 1111111111111111; then no error in the received segment.
- If one of the bits is a 0, then errors have been introduced into the segment.

What all are the part of checksum calculation?

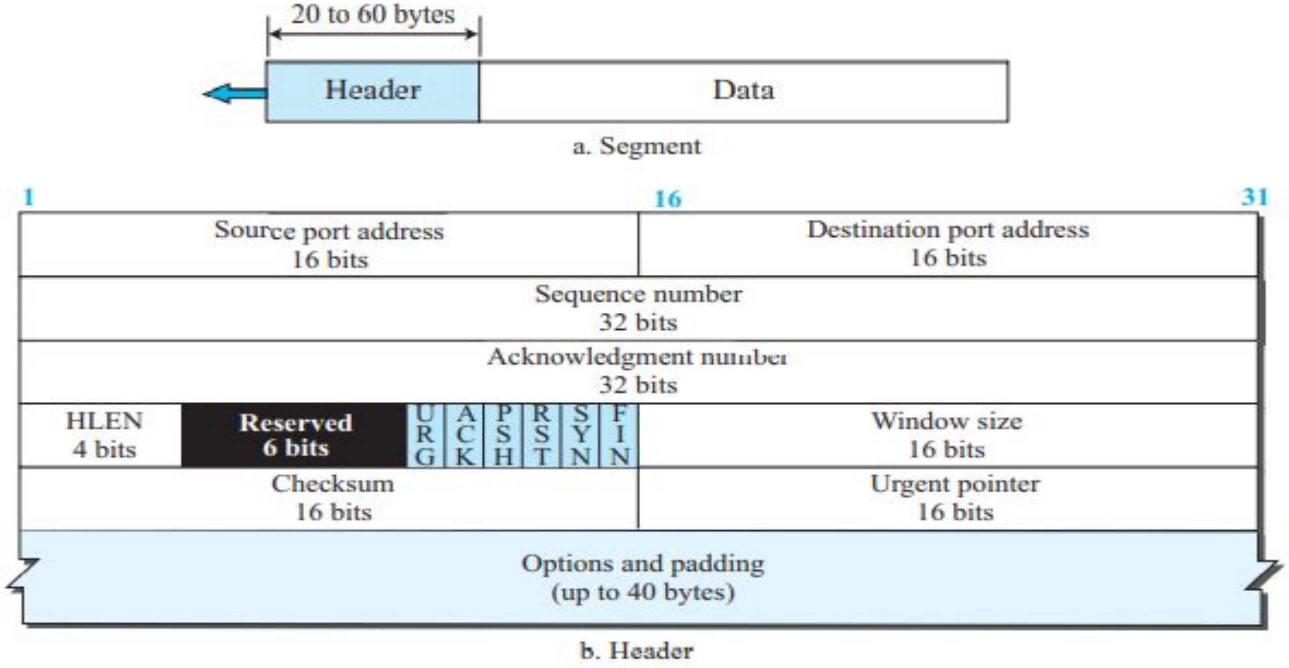

Pseudo Header
TCP/UDP Header + Data
Checksum Is Calculated On TCP/UDP Header and data along with the contents of Pseudo Header

:Psuedo header consists of the following things from the IP header.

- Source Address(IP)
- Destination Address(IP)
- Protocol type
- Length
- Reserved 8 Bit

TCP Segment format:

Figure 3.44 *TCP segment format*



TCP Congestion Control

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(receiver window) variable together define the size of the send window in TCP.
- cwnd is related to the congestion in the middle (network); rwnd is related to the congestion at the end
- Actual window size = minimum (rwnd, cwnd)

Congestion Detection:

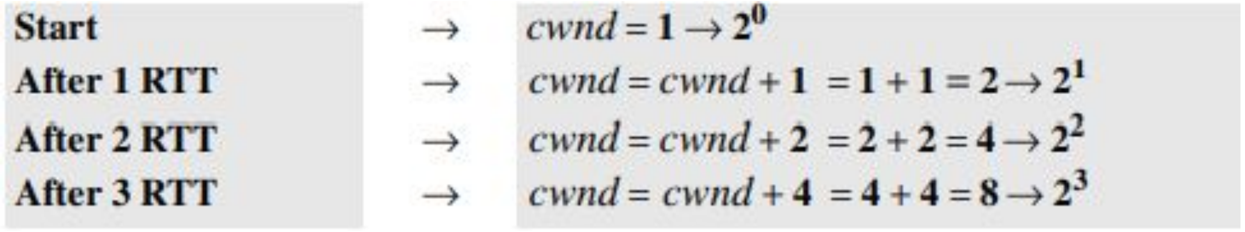
- The TCP sender uses the occurrence of two events as signs of congestion in the network: time-out and receiving three duplicate ACKs.
  - Time-out:
    - If a TCP sender does not receive an ACK for a segment or a group of segments before the time-out occurs, it assumes that the corresponding segment or segments are lost and the loss is due to congestion.
    - is the sign of a strong congestion

- receiving three duplicate ACKs:
    - when a TCP receiver sends a duplicate ACK, it is the sign that a segment has been delayed, but sending three duplicate ACKs is the sign of a missing segment, which can be due to congestion in the network.
    - is the sign of a weak congestion in the network.

<u>Congestion Policies:</u>

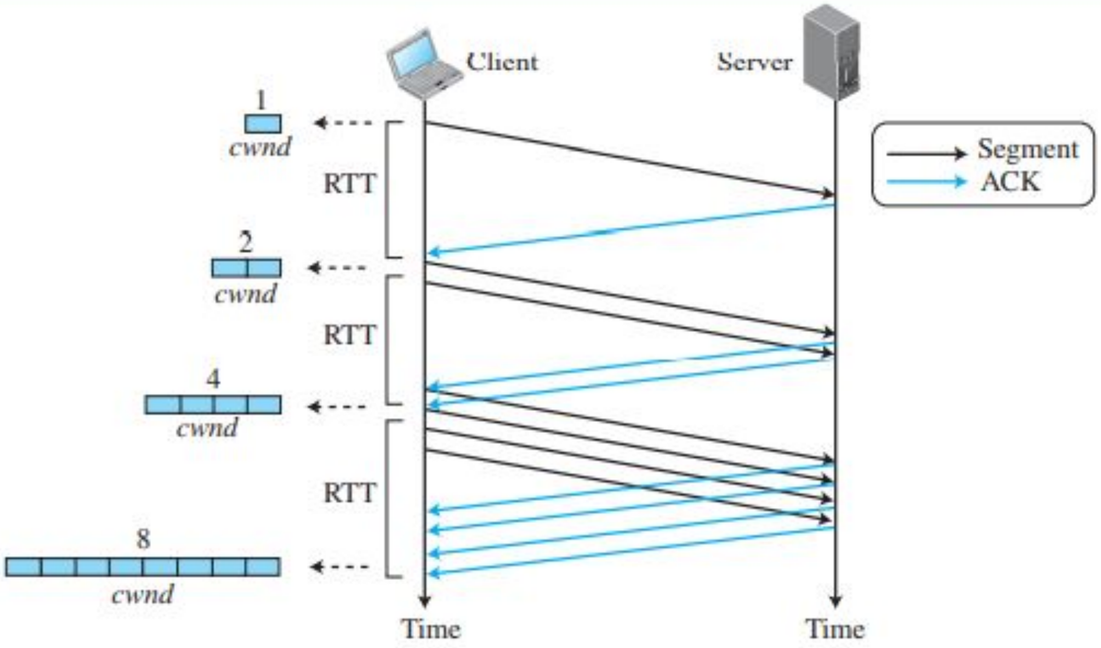
i) Slow Start algorithm: Exponential Increase:

- Initially, the size of the congestion window (cwnd) is one maximum segment size (MSS), but it increases one MSS each time one acknowledgment arrives.
- **If an ACK arrives, cwnd = cwnd + 1**



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

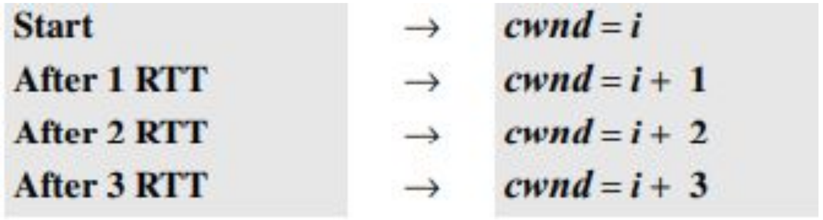
Figure *Slow start, exponential increase*



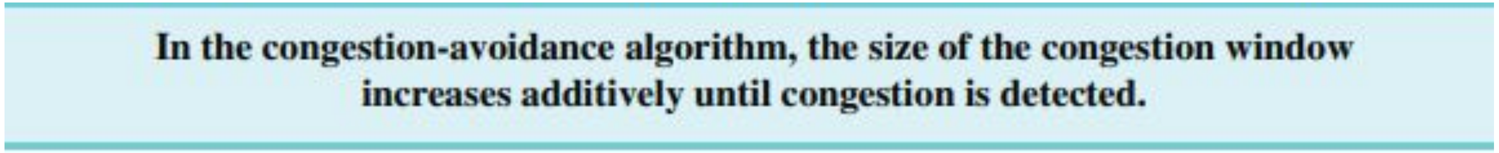
In the slow-start algorithm, the size of the congestion window increases exponentially until it reaches a threshold.

ii) Congestion Avoidance: Additive Increase

- cwnd is increased additively instead of exponentially.
- When the size of the congestion window reaches the slow-start threshold in the case 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.
- **If an ACK arrives, cwnd = cwnd + (1/cwnd).**
-

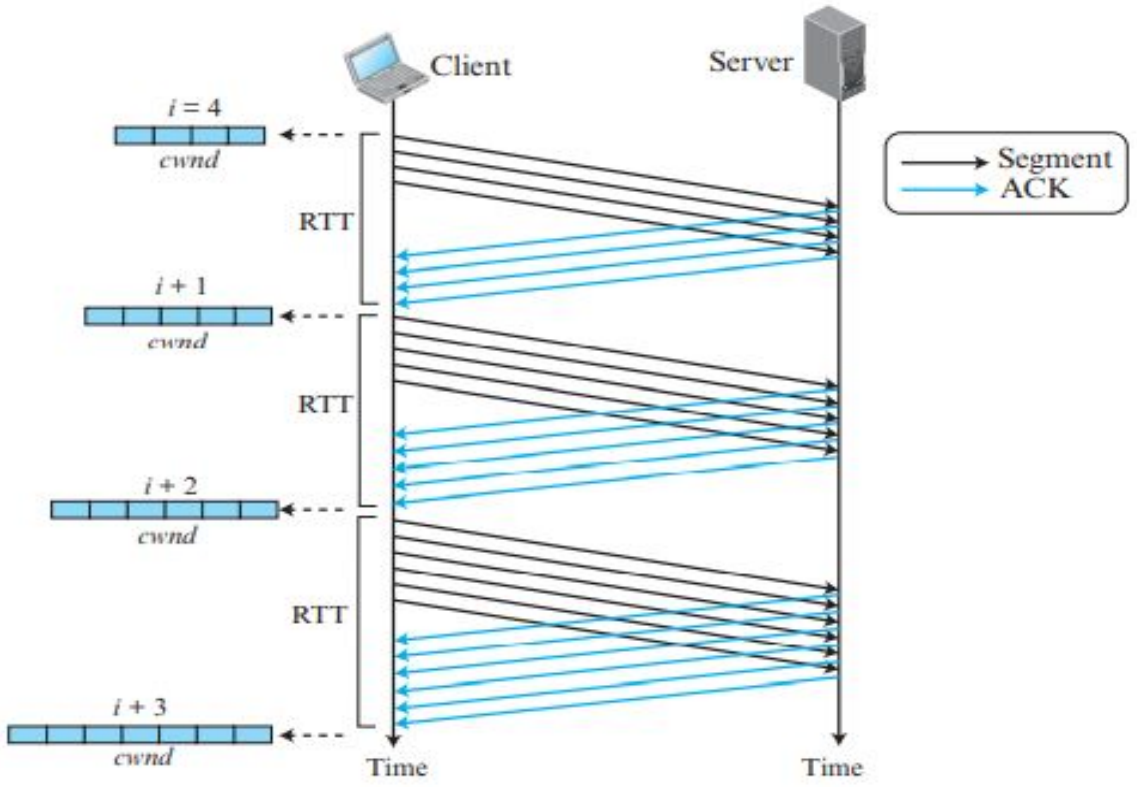


Start	$\rightarrow$	$cwnd = i$
After 1 RTT	$\rightarrow$	$cwnd = i + 1$
After 2 RTT	$\rightarrow$	$cwnd = i + 2$
After 3 RTT	$\rightarrow$	$cwnd = i + 3$



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

Figure *Congestion avoidance, additive increase*



iii) Fast Recovery:

- It starts when three duplicate ACKs arrives that is interpreted as light congestion in the network
- it increases the size of the congestion window when a duplicate ACK arrives
- **If a duplicate ACK arrives, cwnd = cwnd + (1 / cwnd)**