Chapter 3

Transport Layer

Introduction to Transport Layer

- The transport layer is

    1. located between the application layer and the network layer

    2. provides a process-to-process communication between two systems

Transport Layer Services

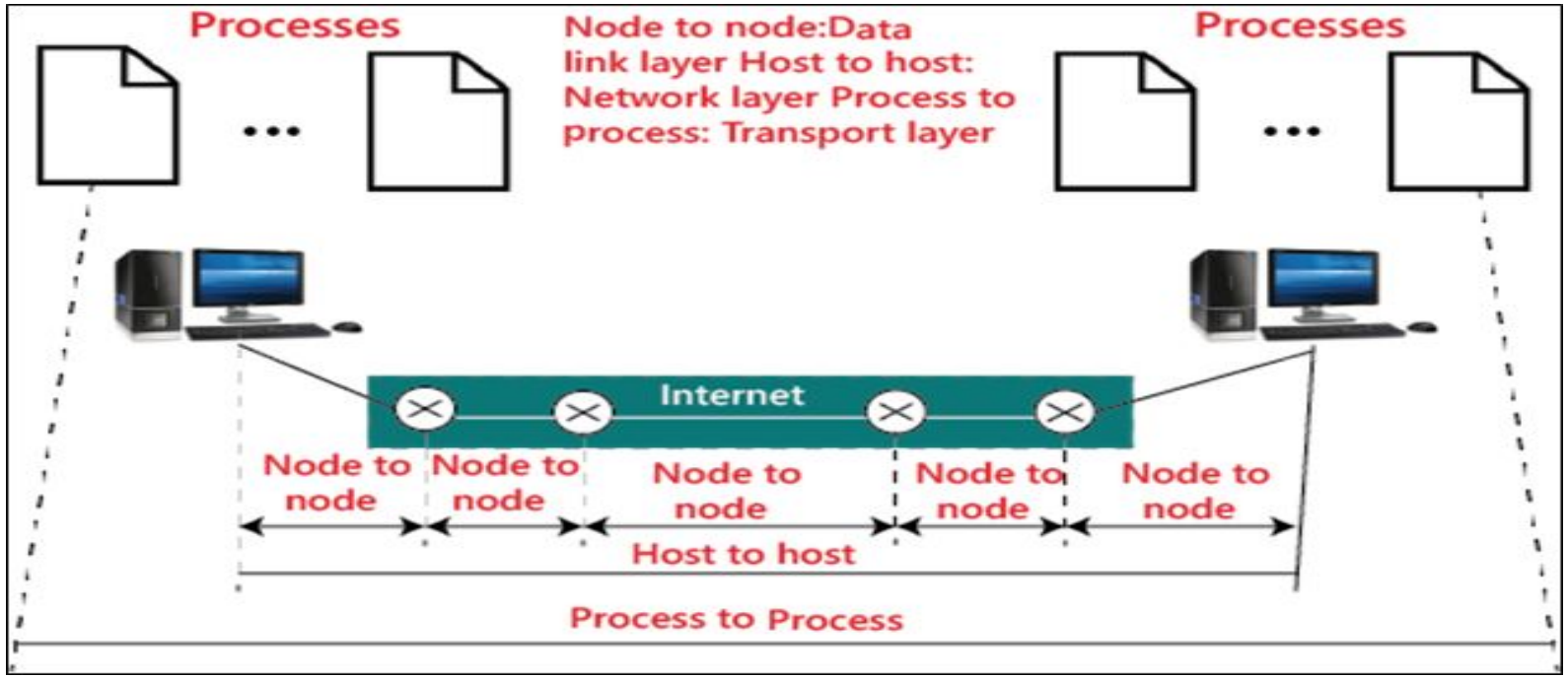
- Transport Layer provides the following services

1. Process-to-Process Communication
2. Addressing: Port Numbers
3. Encapsulation and Decapsulation
4. Multiplexing and Demultiplexing
5. Flow control
6. Error Control
7. Congestion Control

Transport Layer Services (contd..)

1. Process-to-Process Communication

- Transport layer is responsible for process-to-process communication (ie) delivering the data to the correct process in the destination system.



- Above diagram shows,

Transport Layer	does	Process-to-process delivery
Network Layer	does	Host-to-host delivery
Data-link layer	does	Node-to-Node delivery

Transport Layer Services (contd..)

2. Addressing: Port Numbers

- Since transport layer is responsible for process-to-process delivery, the process running in the machine is identified with the help of Port Numbers
- 16-bits are used to represent port numbers (ie) Port numbers range from 0 to 65535
- The client process (program) defines itself with a port number called ephemeral port number which will be valid until the lifetime of that process. Ephemeral port numbers is greater than 1023
- Server process cannot be assigned any port numbers as like client process. It should run only on defined port numbers. These port numbers are called well-known port numbers

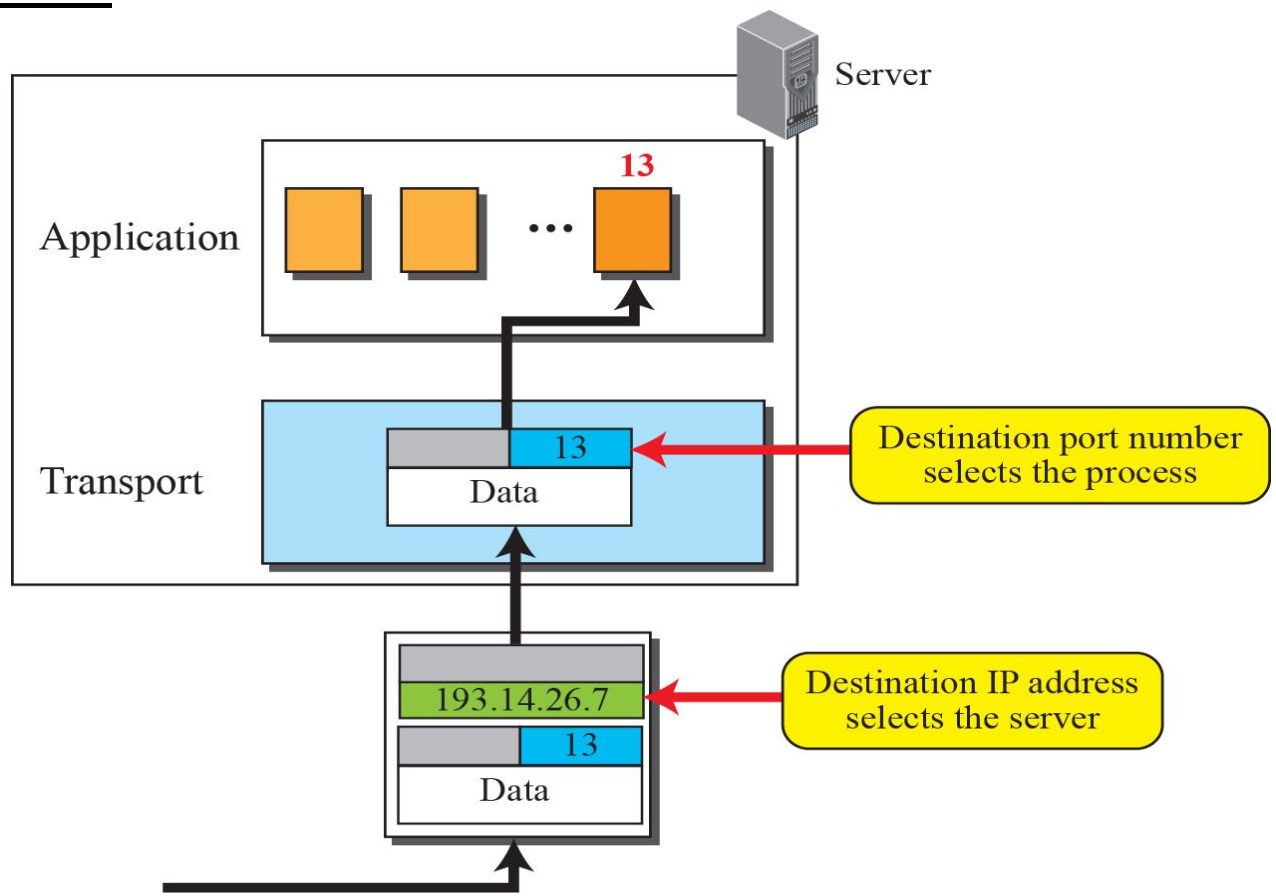
Transport Layer Services (contd..)

2. Addressing: Port Numbers

- Above diagram shows, client process run on port number 52,000 and server process run on port number 13 (Port number for Daytime server is always 13)

Transport Layer Services (contd..)

IP Address vs Port Number



- IP address is used for identifying machine and port number is used identifying process in that machine

Transport Layer Services (contd..)

ICANN Ranges

ICANN divides port numbers into 3 ranges:

1. **Well-known ports:** These ports ranges from 0 to 1023. They are assigned and controlled by ICANN
2. **Registered ports:** These ports ranges from 1024 to 49,151. They are not assigned or controlled by ICANN. They are registered with ICANN to prevent duplication
3. **Dynamic ports:** These ports ranges from 49,152 to 65,535. They are neither controlled nor registered. They can be used for temporary connections

Transport Layer Services (contd..)

Socket Address:

- Combination of IP address and a port number is called a socket address.
- Client socket address defines the client process uniquely
- Server socket address defines the server address uniquely

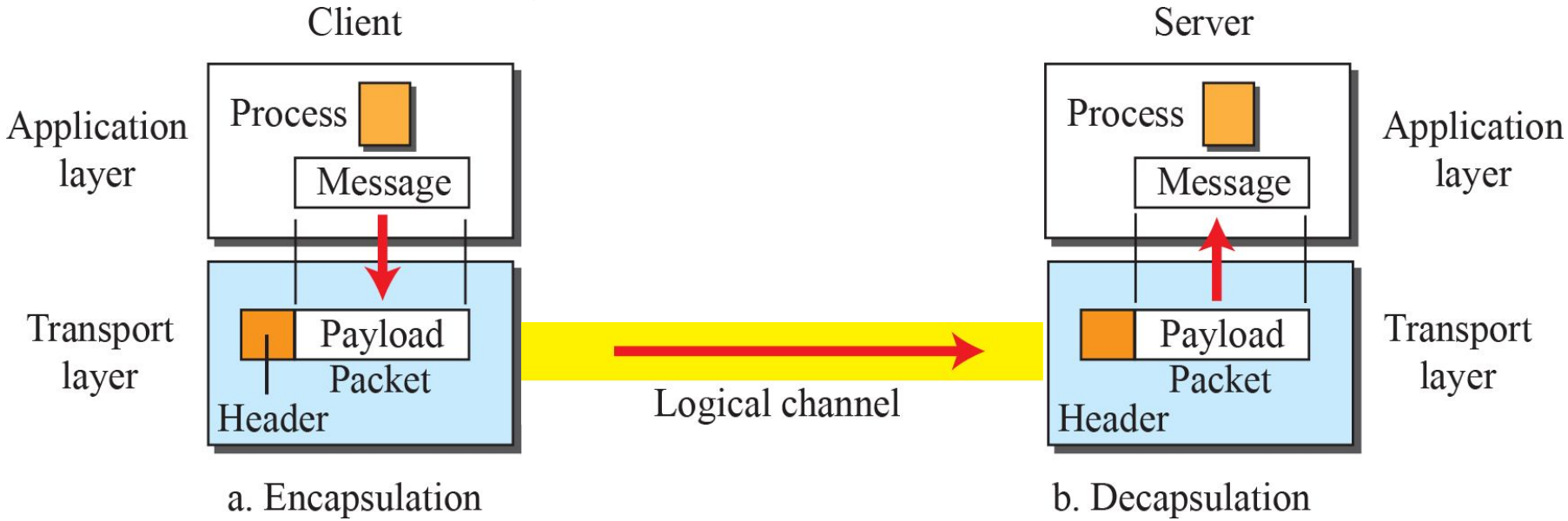
Transport Layer Services (contd..)

3. Encapsulation and Decapsulation

- Encapsulation happens at sender site (ie) transport layer receives data from application layer and adds its header which is called encapsulation
- After adding header, data in transport layer is called user datagrams, segments or packets
- Decapsulation happens at receiver side (ie) removing the header and giving the data to the process at application layer

Encapsulation : Adding Header of that layer to data

Decapsulation : Removing Header



Transport Layer Services (contd..)

4. Multiplexing and Demultiplexing

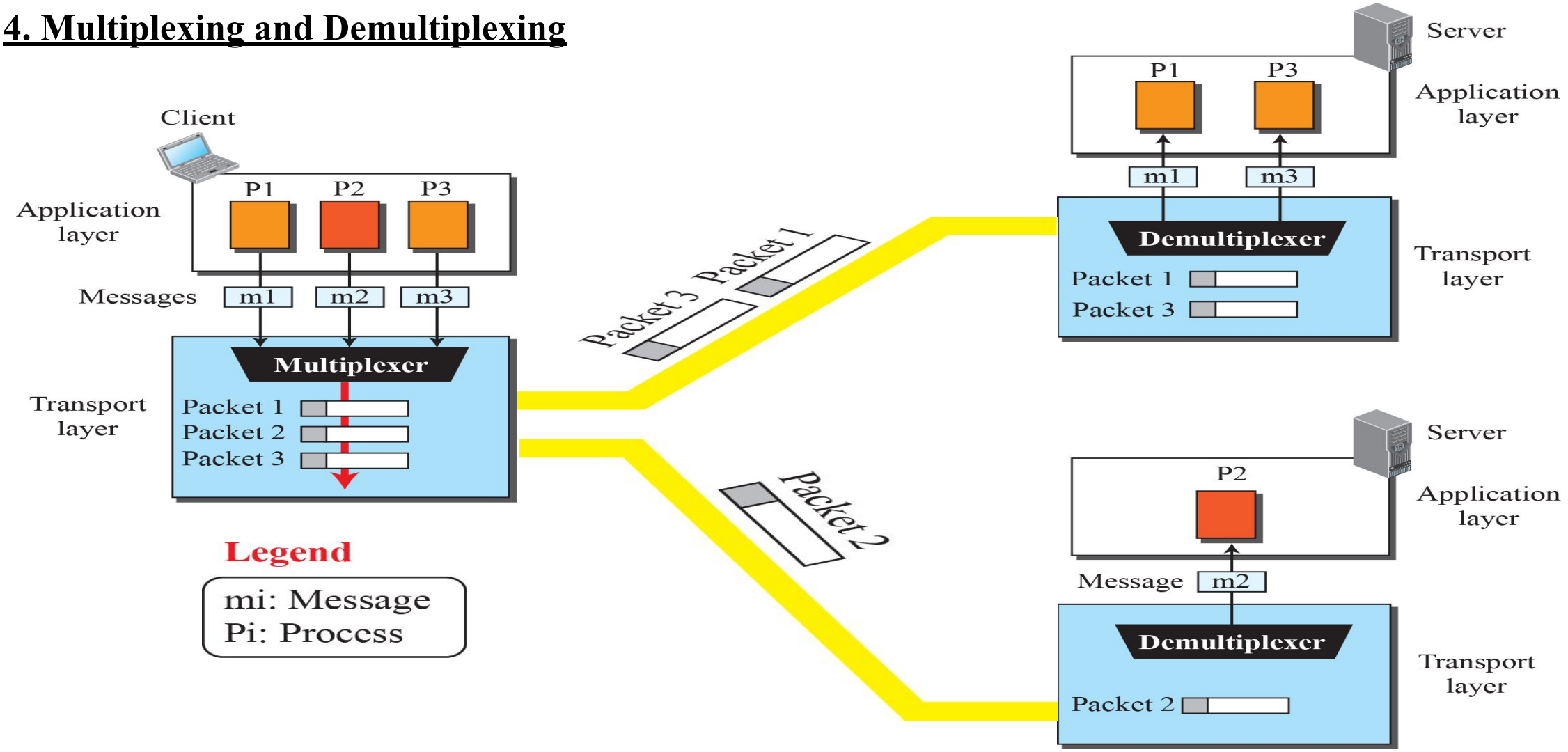
Multiplexing – Accepting data from more than one entity

Demultiplexing - Delivering the data to more than one entity

- Transport layer at source performs multiplexing and Transport layer at destination performs demultiplexing

Transport Layer Services (contd..)

4. Multiplexing and Demultiplexing



Transport Layer Services (contd..)

5. Flow Control

- If the data are produced faster than they can be consumed at receiver side, then receiver may discard some data. This is controlled using some mechanism called Flow control
- Flow control is used to prevent lose of data at consumer site
- One solution for flow control is to use buffers at both sender and receiver side

Transport Layer Services (contd..)

6. Error Control

- Error control in transport layer is responsible for,
  1. Detecting and discarding corrupted packets
  2. Keeping track of lost and discarded packets and resending them
  3. Recognizing duplicate packets and discarding them
  4. Buffering out-of-order packets until the missing packets arrive

Sequence Number:

- Error control is provided by transport layer with the help of sequence numbers
- When packet has to be resent or when duplicate packets arrives at receiver side or packets arriving in out-of-order all these things can be done with the help of sequence number only
- Sequence number range is 0 to 2^m-1, where m is number of bits in sequence number field of header

Acknowledgement Number:

- Along with sequence number, acknowledgement number also is useful for error control

Transport Layer Services (contd..)

7. Congestion Control

- Congestion means traffic in the network (ie) number of packets sent to the network is greater than the capacity of the network makes network congested
- Congestion control refers to techniques that control the congestion and keep the load below the capacity

Transport Layer Services (contd..)

Connectionless and Connection oriented service

- A transport-layer protocol provide two types of service
  1.connectionless service
  2. connection-oriented service

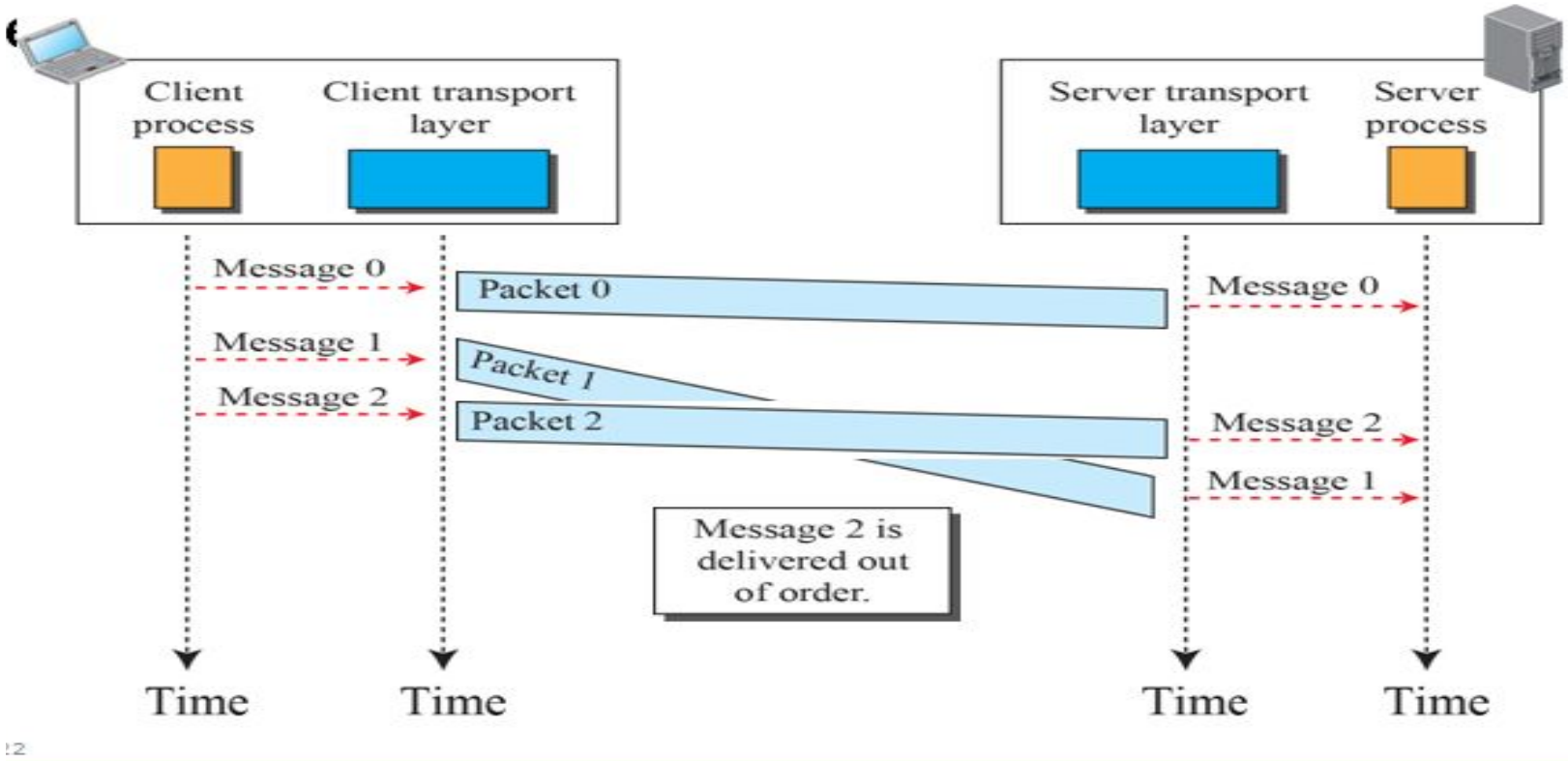
- Connection-oriented service is provided with the help of TCP protocol

- Connectionless service is provided with the help of UDP protocol

Transport Layer Services (contd..)

Connectionless service

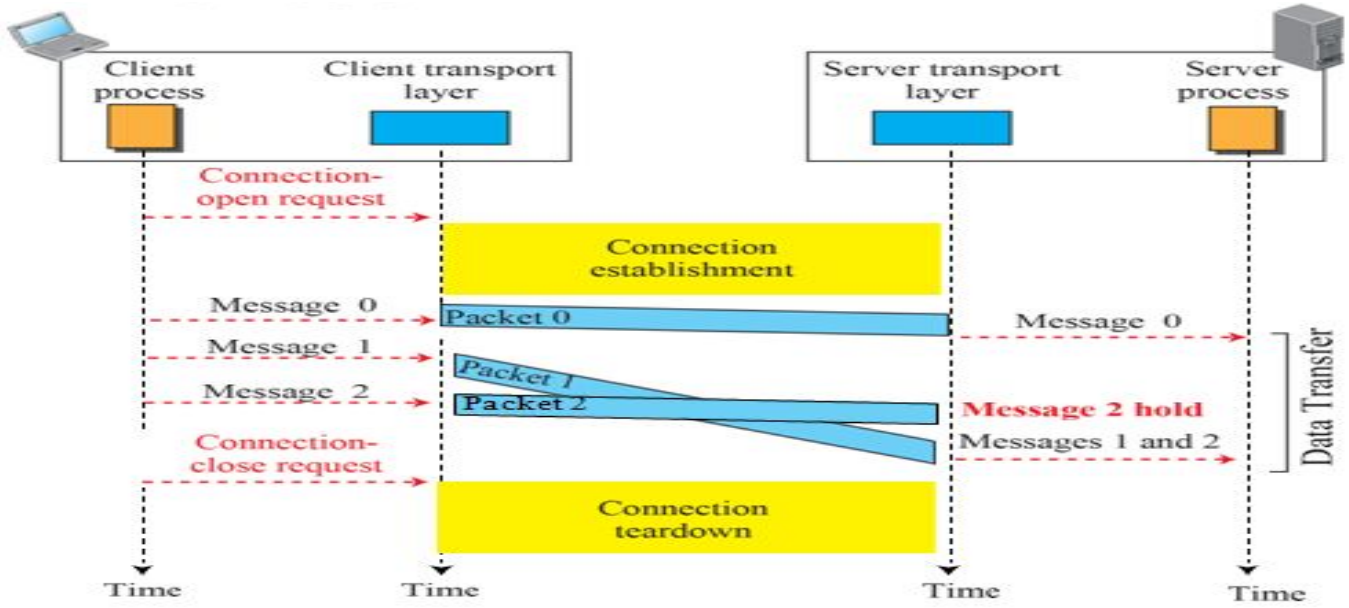
- In connectionless service, there is no dependency between packets (ie) no sequence numbers are assigned to packets
- If packets arrive in out-of-order at receiver side, receiver can't able to arrange the packets correctly



Transport Layer Services (contd..)

Connection-oriented service

- In connection-oriented service, first a connection is created then only data are transmitted. Finally after data transmission connection is terminated
- In case of connection-oriented service, packets are numbered using sequence number with the help of which packets can be arranged correctly at receiver side even though if it is arrived in out-of-order



User Datagram Protocol (UDP)

- The User Datagram Protocol (UDP) is
    1. a connectionless, unreliable transport protocol
    2. UDP is a very simple protocol using a minimum of overhead

Services provided by UDP

1. Process-to-process communication
2. Connectionless service
3. Encapsulation and Decapsulation
4. Multiplexing and Demultiplexing

Services not provided by UDP

1. Flow control
2. Error control except checksum in header
3. Congestion control

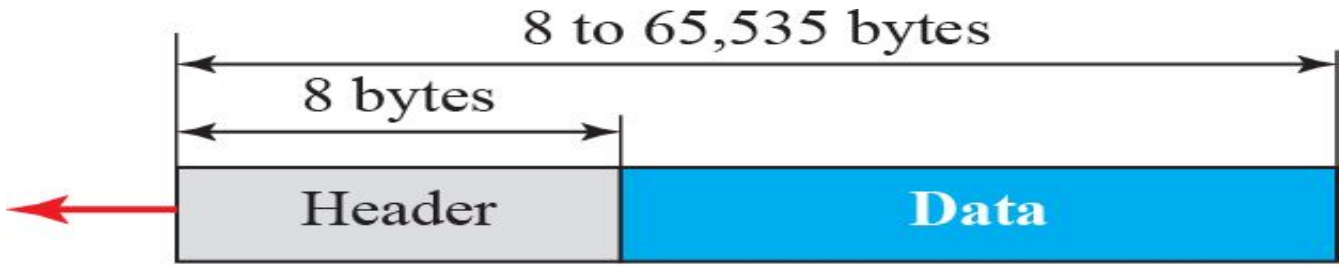
UDP Applications

1. UDP is suitable for a process that requires simple request-response communication. It is not suitable for process that sends bulk of data
2. UDP is suitable for a process with internal flow- and error-control mechanisms. Example, the Trivial File Transfer Protocol (TFTP) process includes flow and error control.
3. UDP is a suitable transport protocol for multicasting. Multicasting capability is embedded in the UDP software but not in the TCP software.
4. UDP is used for management processes such as SNMP
5. UDP is used for some route updating protocols such as Routing Information Protocol (RIP)
6. UDP is normally used for interactive real-time applications (like live streaming)

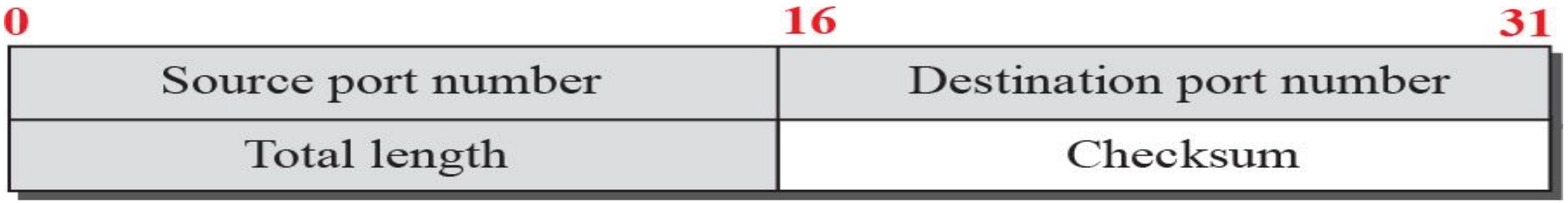
UDP (contd…)

User Datagram

- UDP packets are called user datagrams
- User Datagram is nothing but data from application layer + header of UDP



a. UDP user datagram



b. Header format

UDP (contd…)

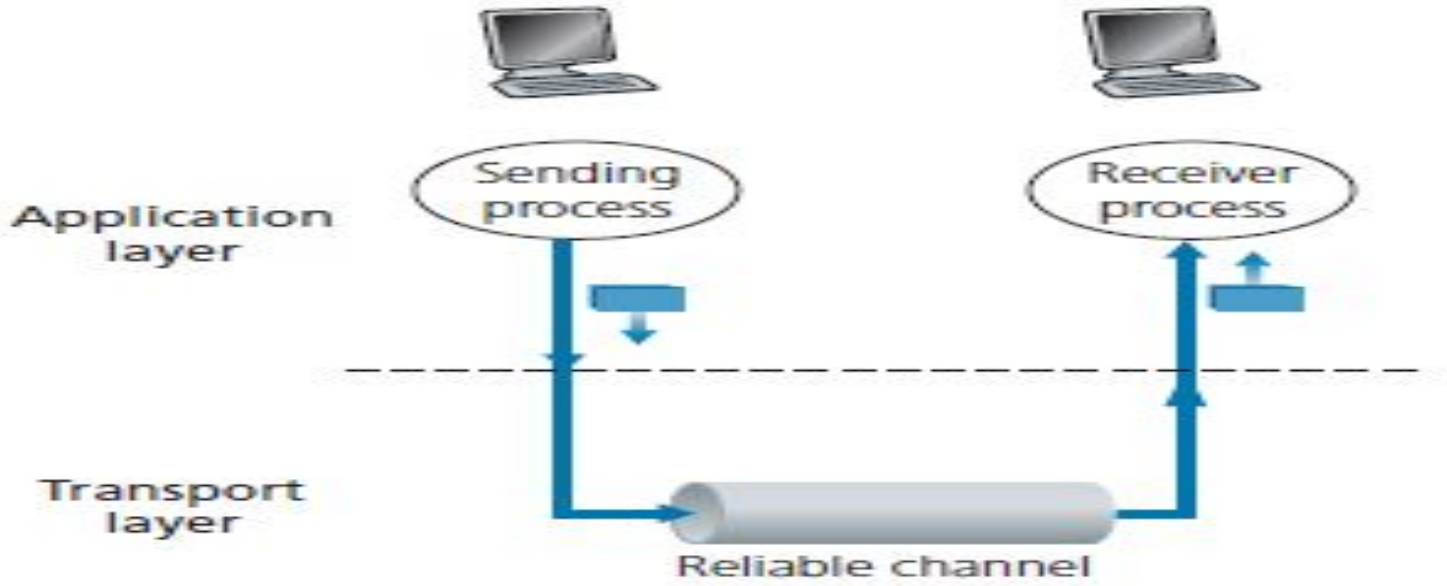
UDP header

- Header length in UDP is of 8 bytes which is fixed
  - The header in the UDP packet has 4 fields:
    1. Source Port Number
    2. Destination Port Number
    3. Total Length
    4. Checksum

1. **Source Port Number (16 bit)** – Port number of process running in sender which send the data to receiver
2. **Destination Port Number (16 bit)** –Port number of process running in receiver to which data has to be delivered
3. **Total length (16 bit) -** header + data. Maximum total length should be 65,535 bytes
4. **Checksum (16 bit) -** for error detection

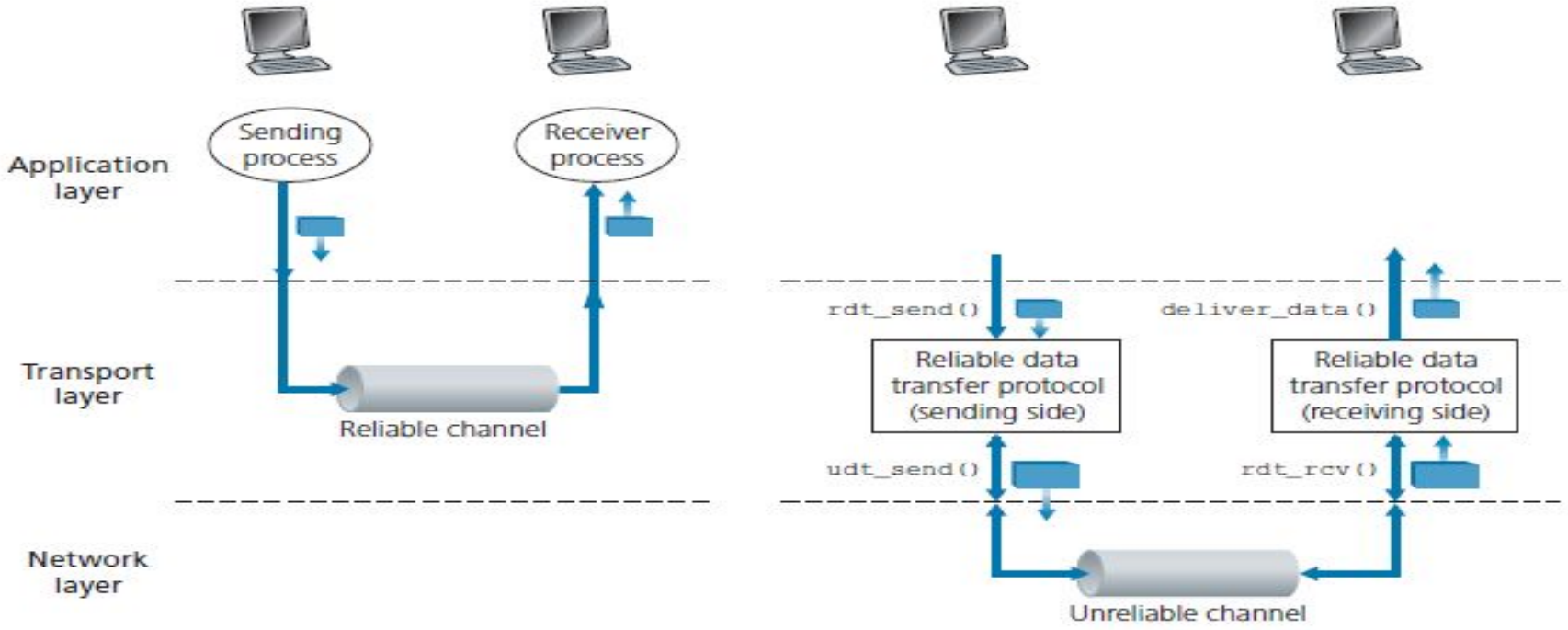
Principles of Reliable Data Transfer

- For a reliable data transfer, a reliable channel is required
- With a reliable channel, no transferred data bits are corrupted or lost, and all are delivered in the order in which they were sent.
- TCP offers a reliable data transfer to the Internet applications that invoke it.



Principles of Reliable Data Transfer (contd..)

- TCP is a reliable data transfer protocol
- But Network layer provides only unreliable data transfer
- TCP is a reliable data transfer protocol that is implemented on top of an unreliable (IP) network layer



Principles of Reliable Data Transfer (contd..)

- rdt_send() - Sending side of the data transfer protocol will be invoked from above by a call to this function (Here rdt stands for reliable data transfer protocol and _send indicates that the sending side of rdt is being called)
- rdt_rcv() - will be called when a packet arrives from the receiving side of the channel
- deliver_data() - the rdt protocol delivers data to the upper layer by calling this function
- udt_send() - Both the send and receive sides of rdt send packets to the other side by a call to udt_send() (udt: unreliable data transfer).

Principles of Reliable Data Transfer (contd..)

Building a Reliable Data Transfer Protocol

- Reliable data transfer protocol is built step by step by considering the following cases one by one

3 cases:

1.Reliable data transfer over a perfectly reliable channel: rdt1.0

2.Reliable data transfer over a channel with bit errors

(i) rdt2.0

(ii) rdt2.1

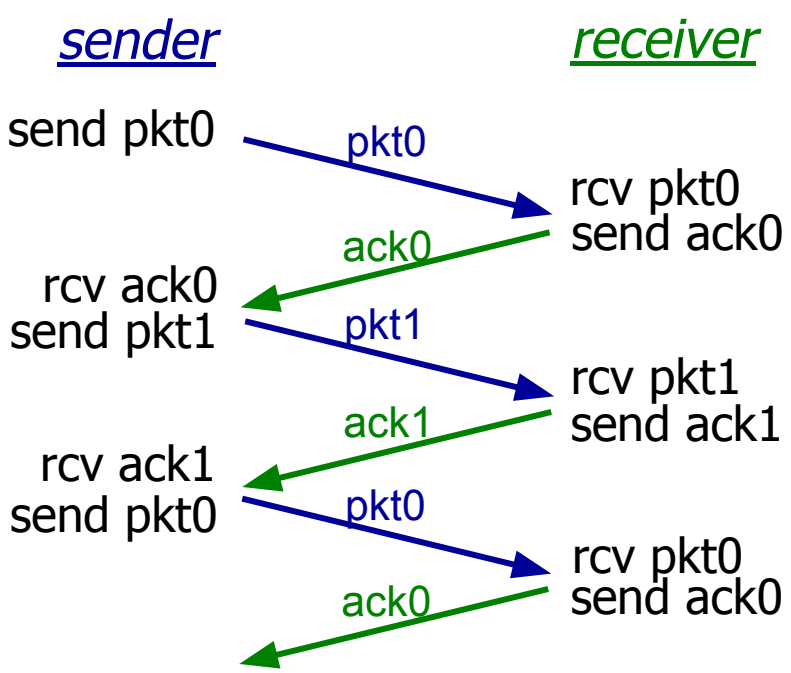
(iii) rdt 2.2

3.Reliable data transfer over a lossy channel with bit errors: rdt3.0

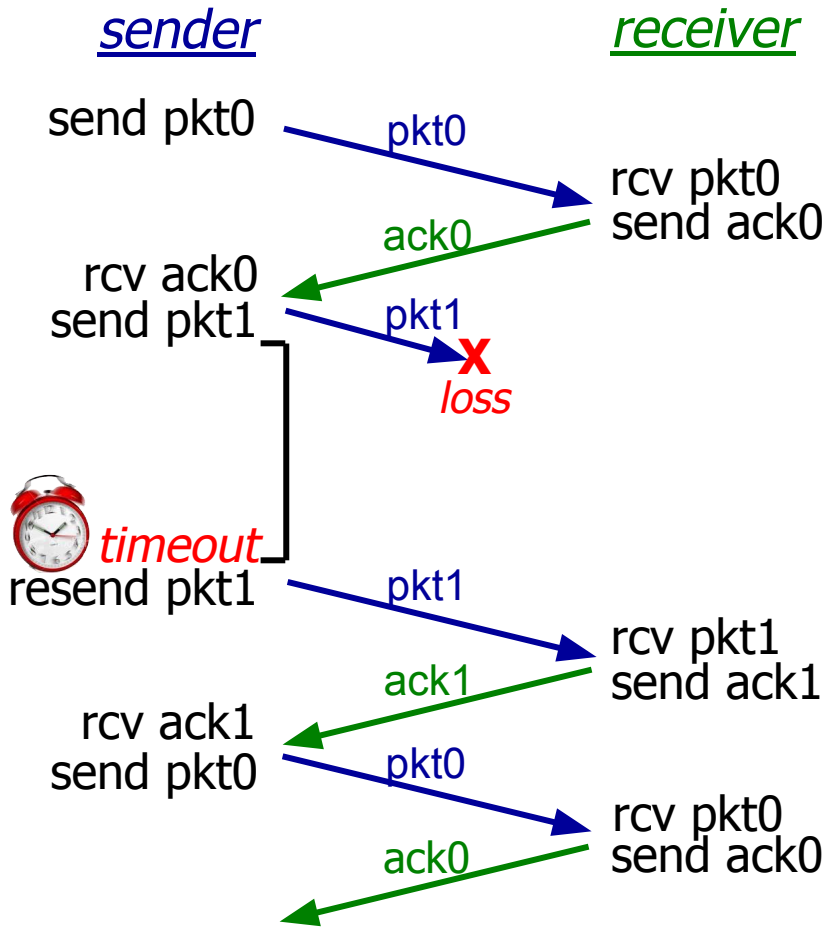
3.Reliable data transfer over a lossy channel with bit errors: rdt3.0

- In rdt3.0 the channel is considered as a lossy channel where data or ACK can get lost and also can get corrupted
- Data loss problem is addressed here by retransmitting the data after timeout
- rdt3.0 is also called as alternating-bit protocol since sequence number in the packet alternate between 0 and 1

3.Reliable data transfer over a lossy channel with bit errors: rdt3.0 (contd..)

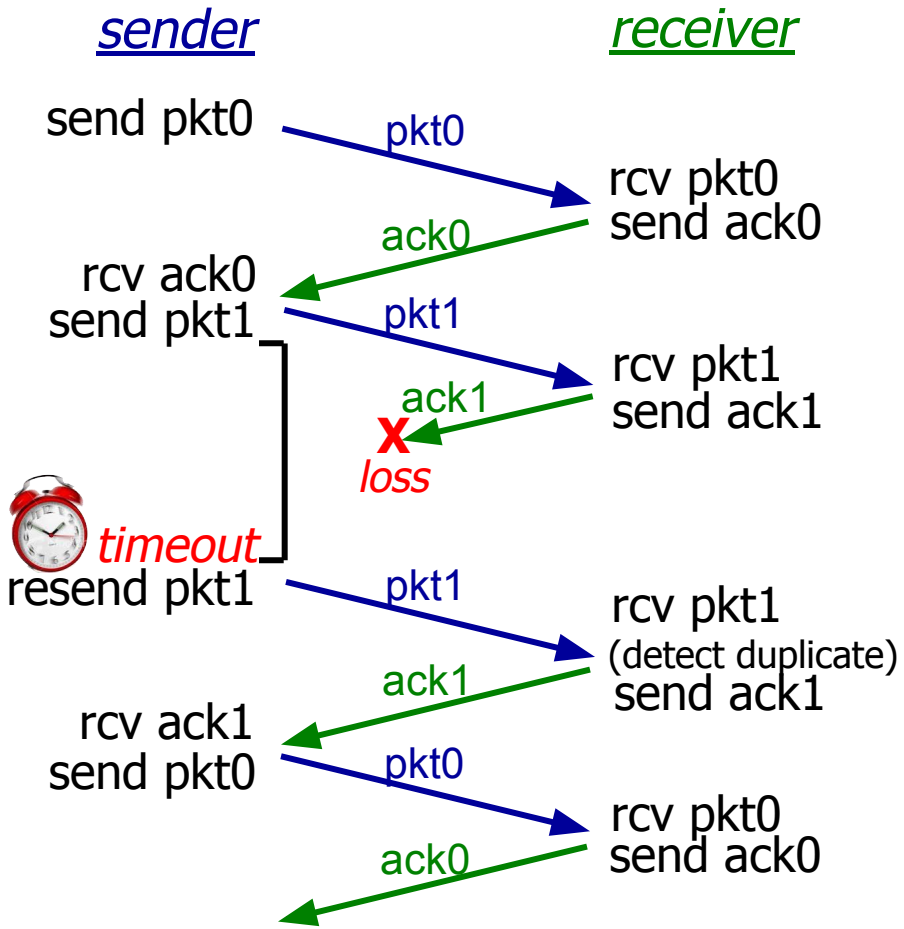


(a) no loss

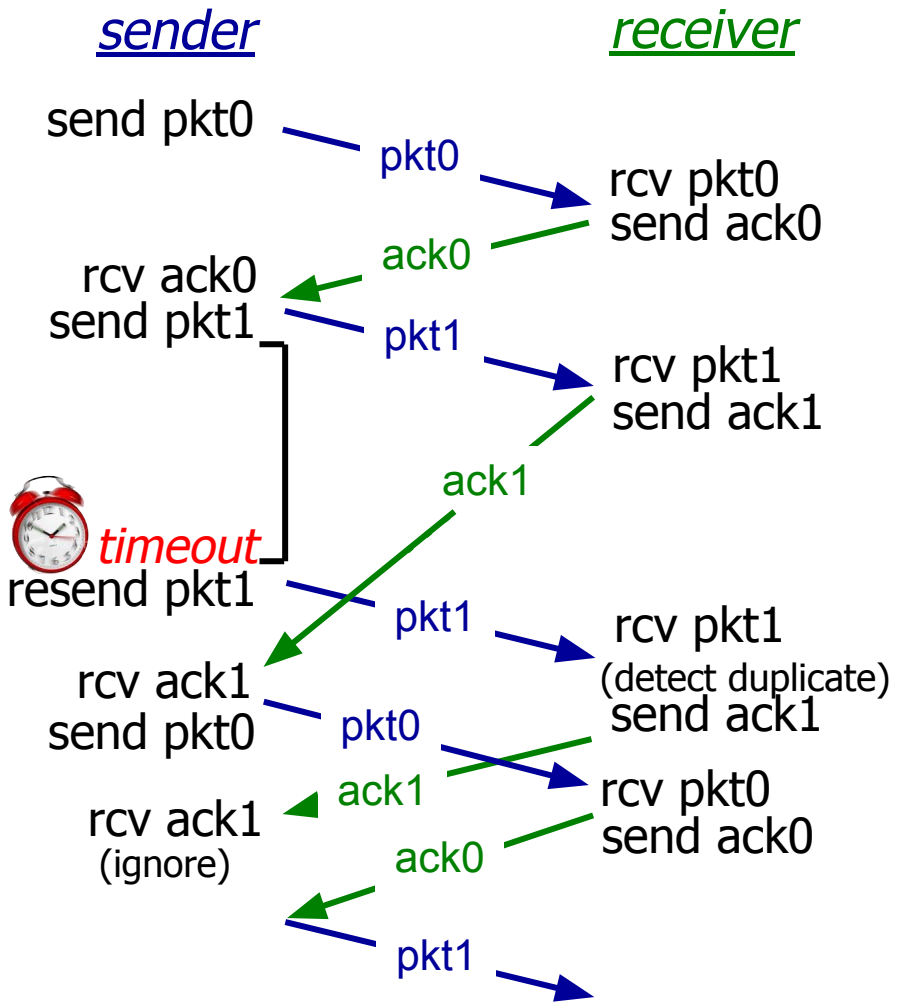


(b) packet loss

3.Reliable data transfer over a lossy channel with bit errors: rdt3.0 (contd..)



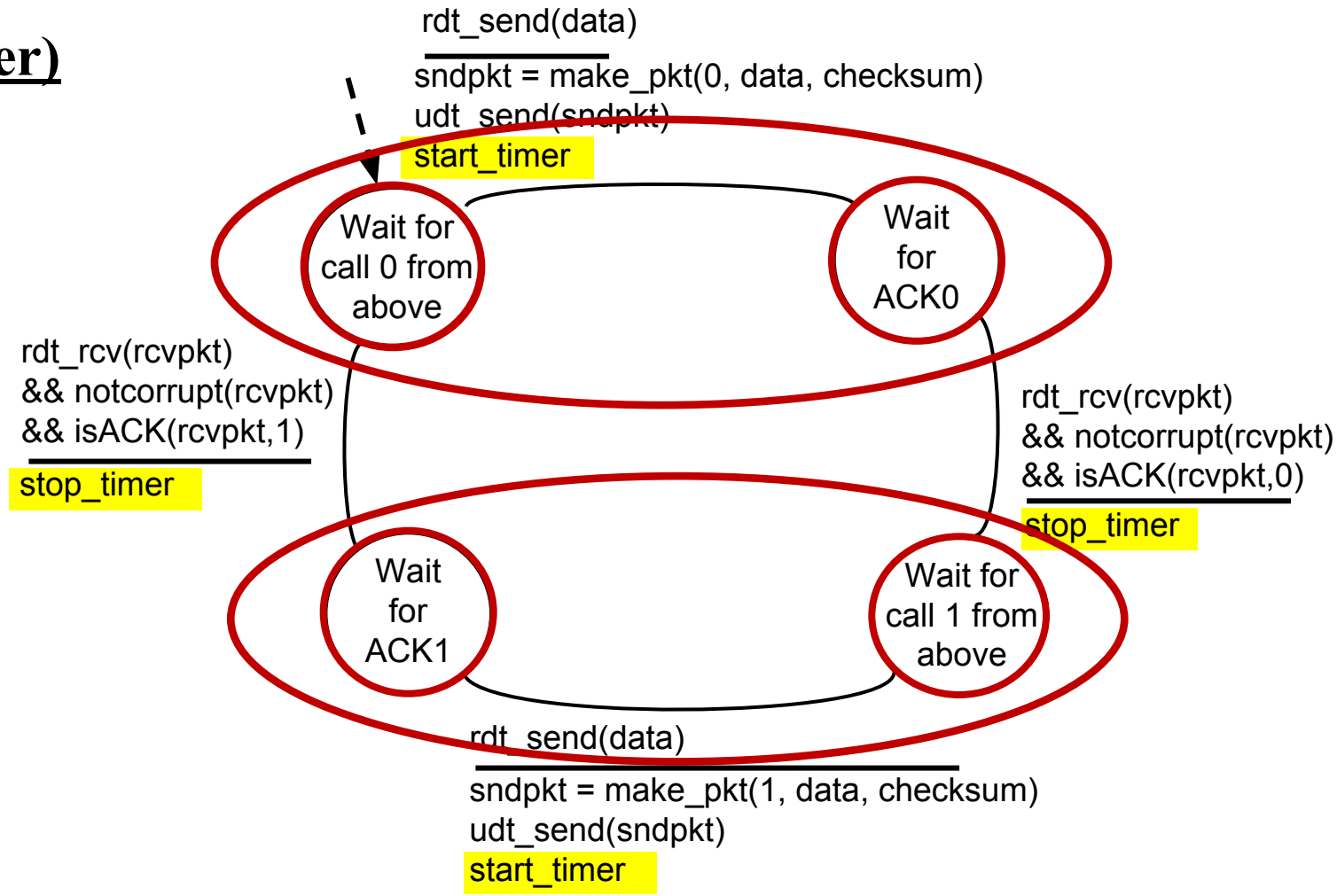
(c) ACK loss



(d) premature timeout/ delayed ACK

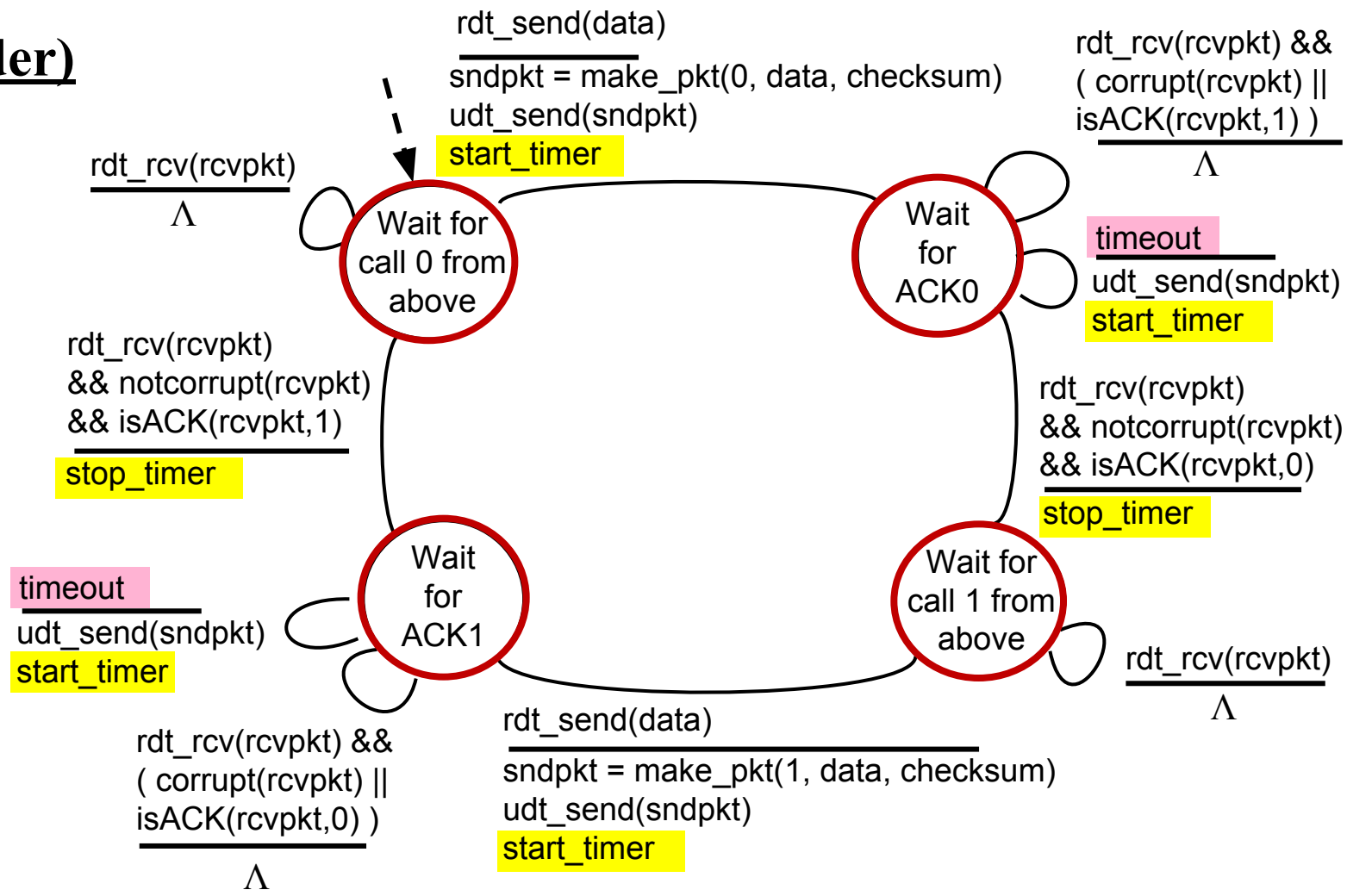
3.Reliable data transfer over a lossy channel with bit errors: rdt3.0 (contd..)

rdt3.0 (sender)



3.Reliable data transfer over a lossy channel with bit errors: rdt3.0 (contd..)

rdt3.0(sender)



3.Reliable data transfer over a lossy channel with bit errors: rdt3.0 (contd..)

rdt3.0 (sender)

- FSM Diagram in the previous slide shows the rdt3.0 sender side
- In rdt3.0, sender initially sends data with sequence number 0 to receiver and starts the timer and waits for ACK0
- If it receives ACK0 and it is not corrupted means it stops the timer and moves to next state wait for call 1 from above
- If it receives ACK1 (ACK with sequence number 1) or if received data is corrupted means it does nothing and remains in the same state and once time-out occurs sender retransmits the data and start the timer again
- Above steps are repeated for data with sequence number 1 also

2.Reliable data transfer over a channel with bit errors (contd..)

(iii) rdt3.0 (receiver)

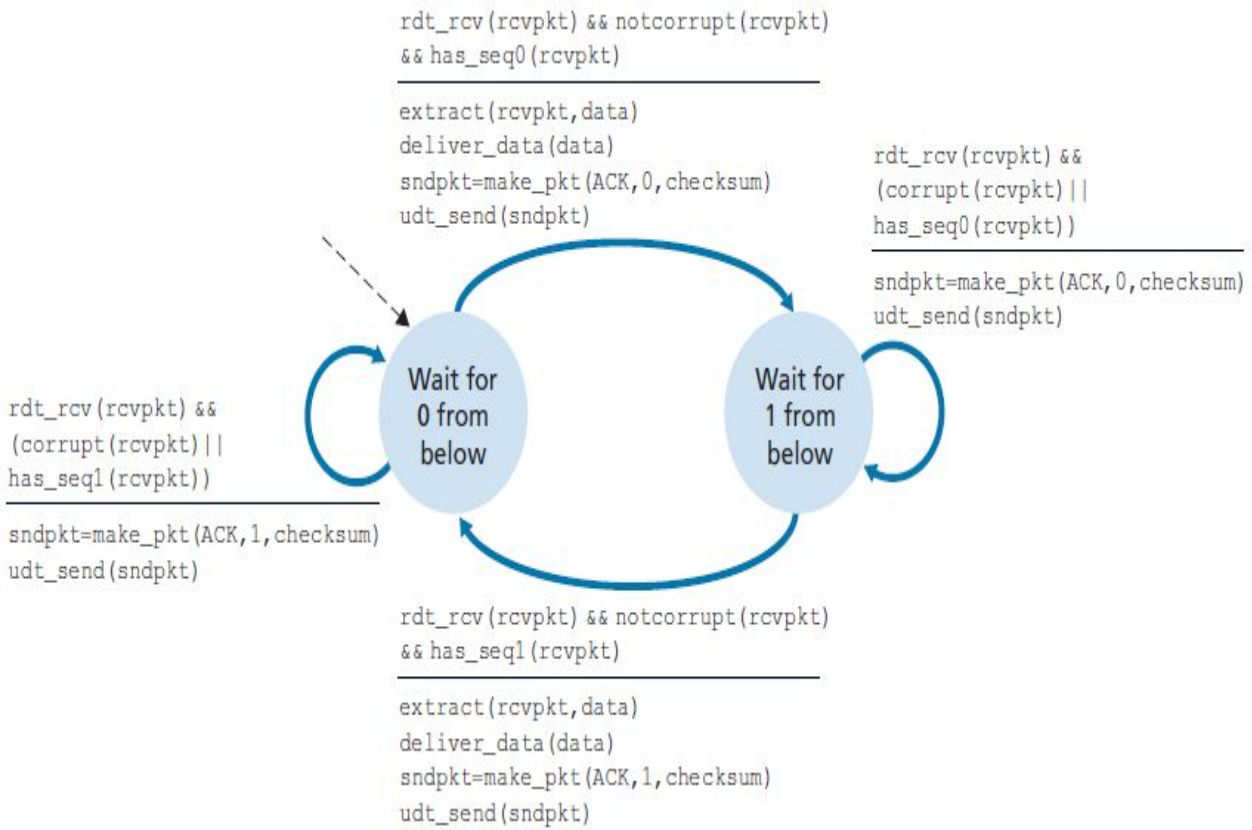


Figure 3.14 ♦ rdt2.2 receiver

2.Reliable data transfer over a channel with bit errors (contd..)

(iii) rdt2.2 (receiver)

- FSM Diagram in the previous slide shows the rdt2.2 receiver side
- Receiver initially waits for data with sequence number 0
- Receiver upon receiving data from sender checks whether it is not corrupted and has sequence number 0. If it is true means it delivers data to upper layer and sends ACK with sequence number 0 and checksum to sender and moves to next state
- If received data is corrupted or has sequence number 1 means it sends ACK with sequence number 1 to sender
- Above steps are repeated for data with sequence number 1 also

Pipelined Reliable Data Transfer Protocols

Stop-and-wait protocol vs Pipelined protocols

- All the previously seen rdt protocols (rdt1.0,rdt2.0,rdt2.1,rdt2.2, rdt3.0) are stop-and-wait protocols only
- In stop-and-wait protocol the utilization of sender is minimum (ie) after sending data the sender has to wait for ACK from receiver. During this time the sender is idle though it has some more data to send to the receiver
- In order to reduce this idle time of sender and to improve sender utilization, Pipelined protocols are proposed
- In case of pipelined protocols sender sends more data to receiver without waiting for ACK of previously send data.

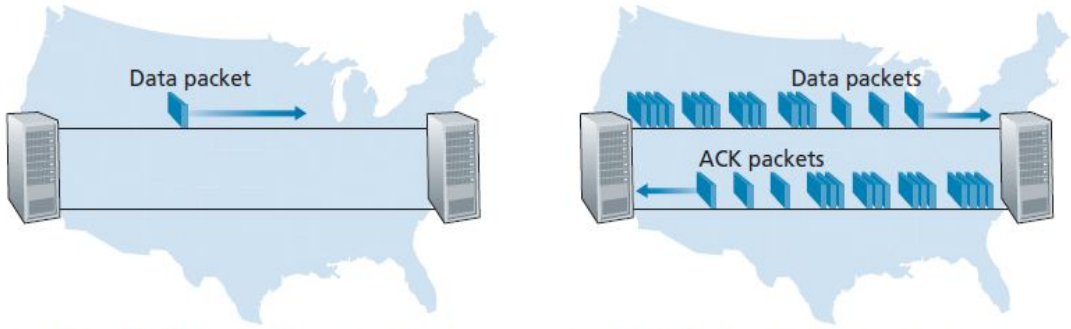


Figure 3.17 ♦ Stop-and-wait versus pipelined protocol

Pipelined Reliable Data Transfer Protocols

- Two types of Pipelined protocols are there:
    1. Go-Back-N protocol (GBN)
    2. Selective Repeat protocol (SR)

Go-Back-N protocol (GBN)

- In a Go-Back-N (GBN) protocol, the sender is allowed to transmit multiple packets without waiting for an acknowledgment
- But the number packets the sender can send is constrained using send window
- GBN is also called as sliding window protocol since both sender and receiver slides its window after receiving data/ACK
- GBN is working based on cumulative ACK concept

Go-Back-N protocol (GBN) (contd..)

Sequence Numbers

- Every packet is numbered which is called as sequence number
- Range for Sequence number depends on number of bits for sequence number field Range for sequence number is 0 to 2^m- 1 where m is the number of bits in sequence number field

Acknowledgement Numbers

- Acknowledgement number in this protocol is cumulative and defines the sequence number of the packet correctly received
- If acknowledgement number is 6 then all packets with sequence number upto 6 have arrived safely

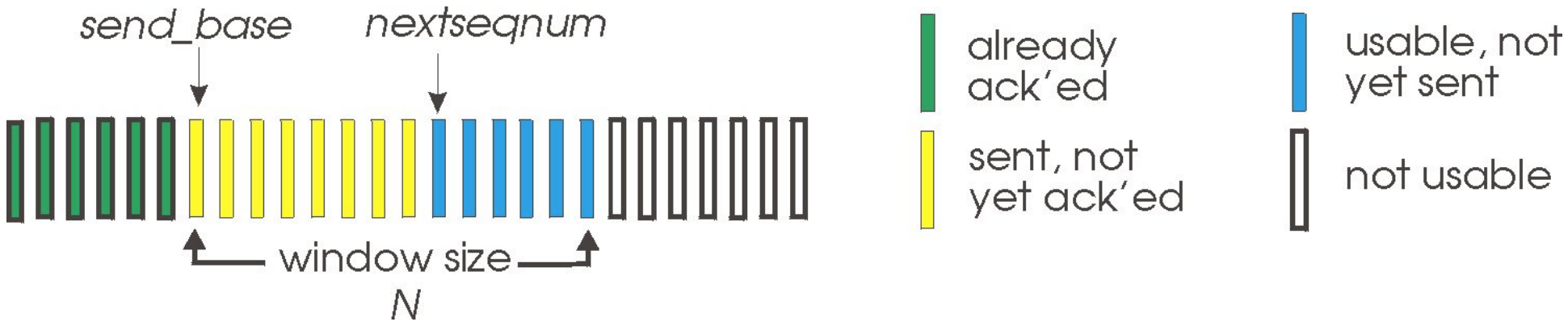
Send window

- Send window is an imaginary box covering the sequence number of data packets.
- In GBN, at sender side send window is considered which represents number of packets the sender can send without waiting for ACK
- Maximum size of send window is 2^m-1, m is the number of bits in the sequence number field

Go-Back-N protocol (GBN) (contd..)

send window (contd..)

- Below diagram shows the send window of size N



- There are 4 regions in the above diagram,

1. First region(Green color), left of the send window, defines sequence number of packets that are sent and acknowledged. No copies of them are maintained

Go-Back-N protocol (GBN) (contd..)

send window (contd..)

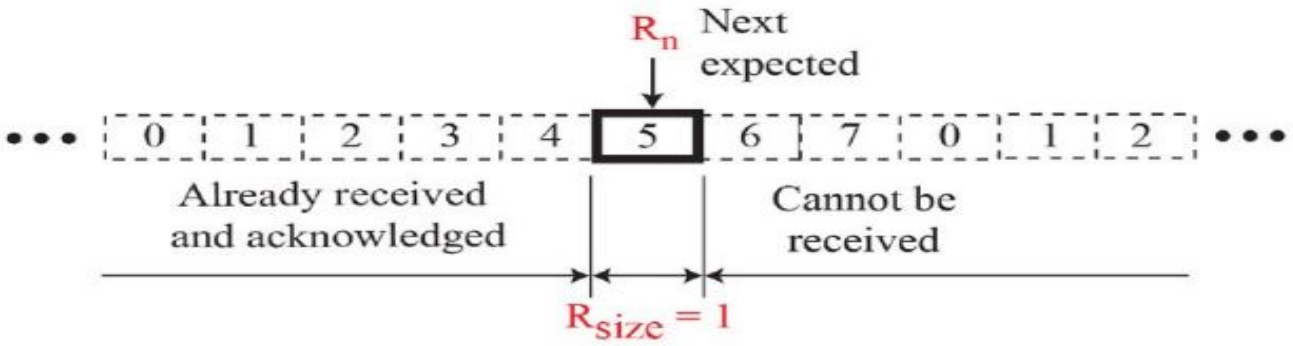
2. Second region(yellow color), defines range of sequence number of packets that are sent but not received acknowledgement. These packets are called outstanding packets
3. Third region(blue color) defines range of sequence numbers of packets that can be send when data is ready
4. Fourth region (white color) defines sequence numbers that cannot be used

- Sender slides it send window one step ahead after it have received correct ACK

Go-Back-N protocol (GBN) (contd..)

Receive window

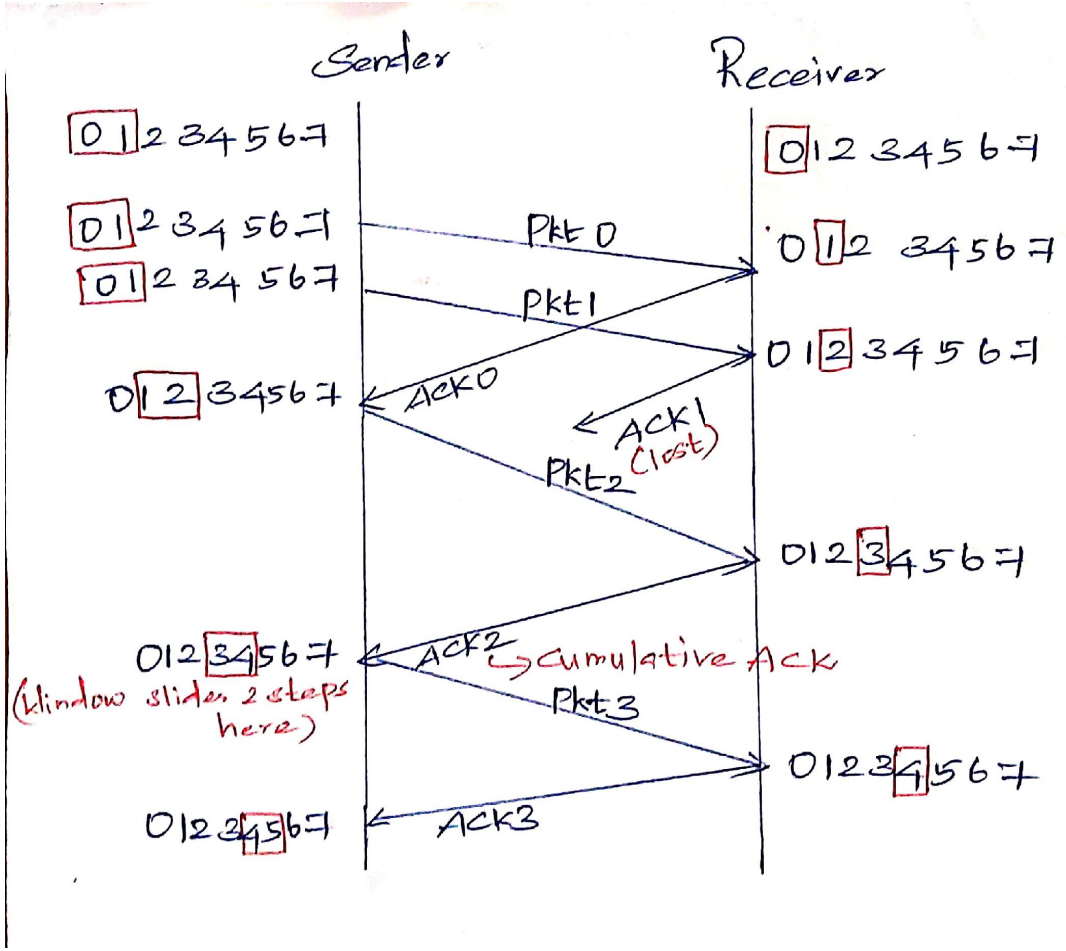
- Receive window is used to ensure correct data are received and the correct acknowledgements are sent
- In Go-Back-N size of receiver window is always 1
- Receiver is expecting for packet what is present in the receive window. If some packets arrived it is discarded and needs to be resent



- Receiver slides it receive window one step ahead after it have received correct data

Go-Back-N protocol (GBN) (contd..)

Example 1: (ACK is lost)



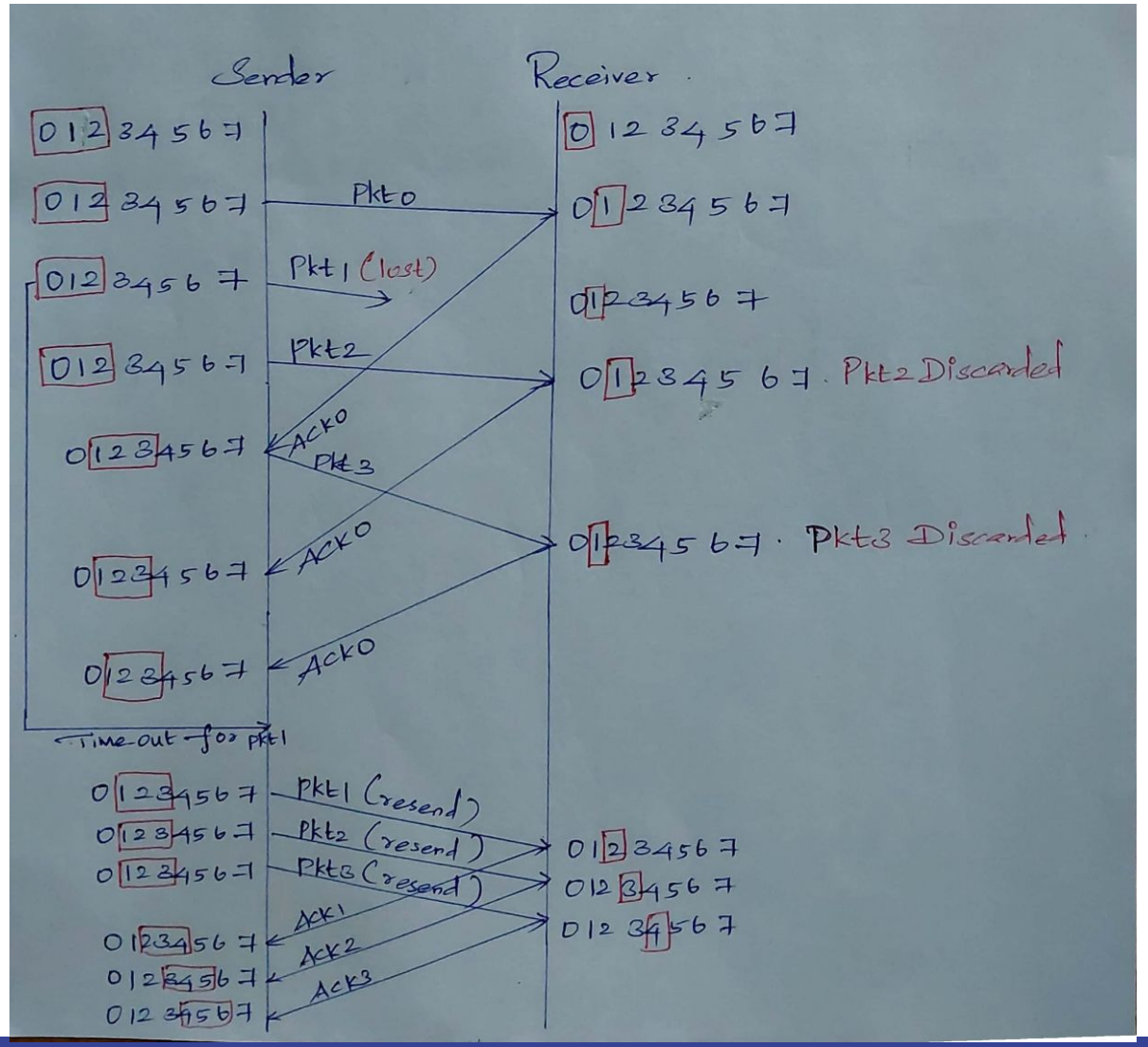
Go-Back-N protocol (GBN) (contd..)

Example 1: (ACK is lost)

- In the example shown in the previous slide, ACK 1 is lost.
- But sender after receiving ACK2 it understood packet 1 also received by receiver successfully and slides send window 2 steps ahead
- ACK 2 is called Cumulative ACK

Go-Back-N protocol (GBN) (contd..)

Example 2: (Data is lost)



Go-Back-N protocol (GBN) (contd..)

Example 2: (Data is lost)

- In the example shown in the previous slide, pkt1 is lost.
- Receiver does not receives pkt1. While it waiting for pkt1 from sender it receives pkt2 and pkt3 only. So receiver is going to discard pkt2 and pkt3 and send ACK0 (ACK number is the sequence number of last correctly received packet)
- Sender while receiving ACK0 for 1st time slides the send window. When it receives ACK0 for 2nd and 3rd time it does nothing
- Once time-out occurs for pkt1 sender is going to retransmit all packets from 1(ie) sender it will again retransmit packet1, 2 and 3
- This why we are saying this protocol as Go-Back-N. If a packet gets lost, sender is going to re transmit all packets from lost packet

Selective Repeat protocol (SR)

- In Go-Back-N protocol, when a single packet is lost or corrupted sender resends all packets from the lost packet even though some of these packets may be received safely by the receiver.
- Resending of all packets from the lost packet makes the network congested
- To overcome this problem, Selective Repeat protocol is proposed
- As the name itself suggests this protocol is used to resend only selected packets that are actually lost
- When packets come in incorrect order to the receiver, receiver buffer those data
- There is no cumulative ACK in SR. The number in the ACK represents that packet alone it is been received by the receiver successfully and it does not mean about the previous packets

Selective Repeat protocol (SR) (contd..)

Send Window:

- Maximum size of send window is smaller than compared to GBN
- In this protocol maximum size of send window is 2^{m-1}
- For example if m=4, the sequence number go from 0 to 15 but maximum send window size is 8 (2^{4-1})

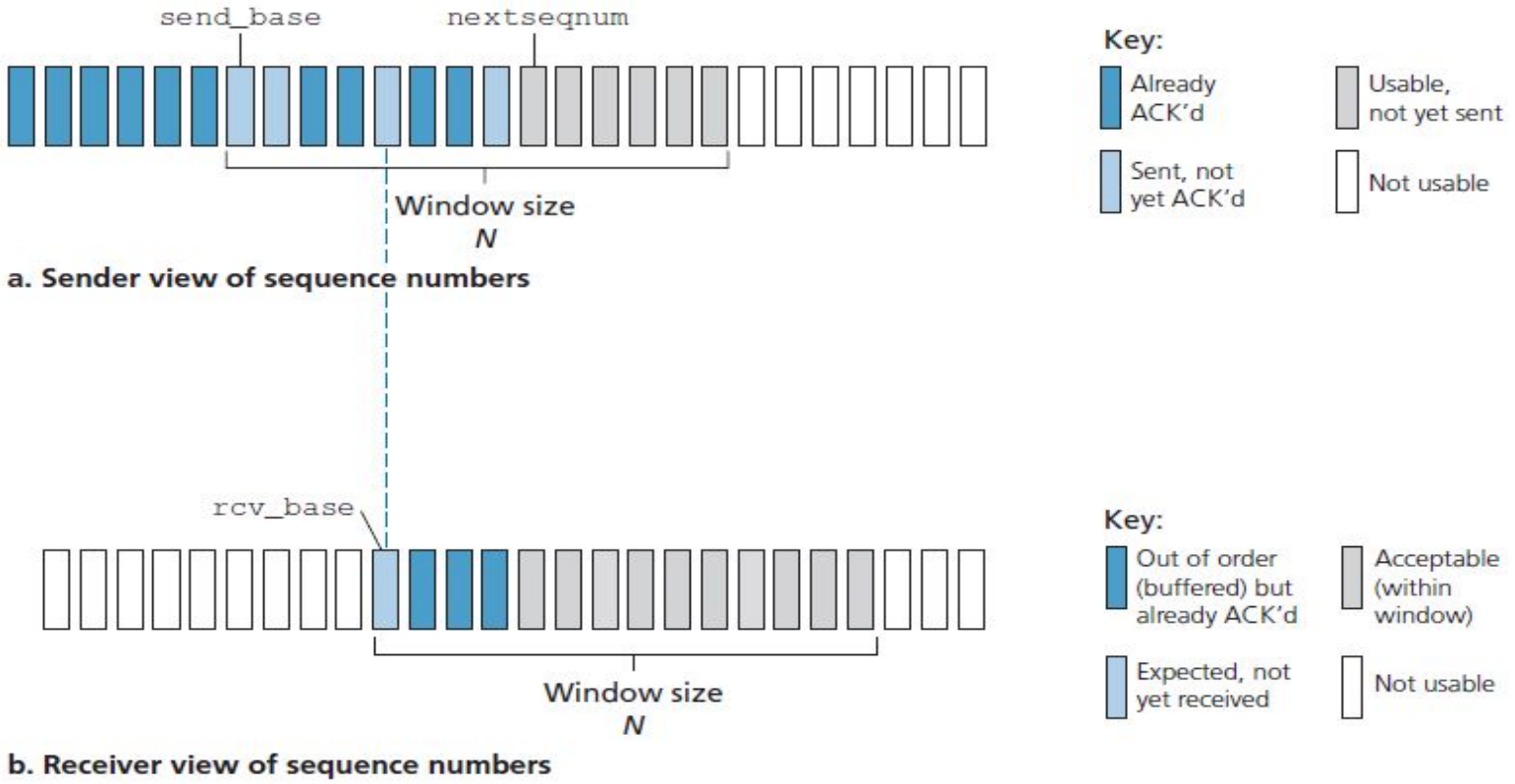
Receive Window:

- Size of receive window is same as that of the size of send window (ie) 2^{m-1}
- In this protocol size of receive window is same as that of send window because to allow as many packets as the size of the receive window to arrive out of order and be kept until there is a set of consecutive packets to be delivered to the application layer.

Selective Repeat protocol (SR) (contd..)

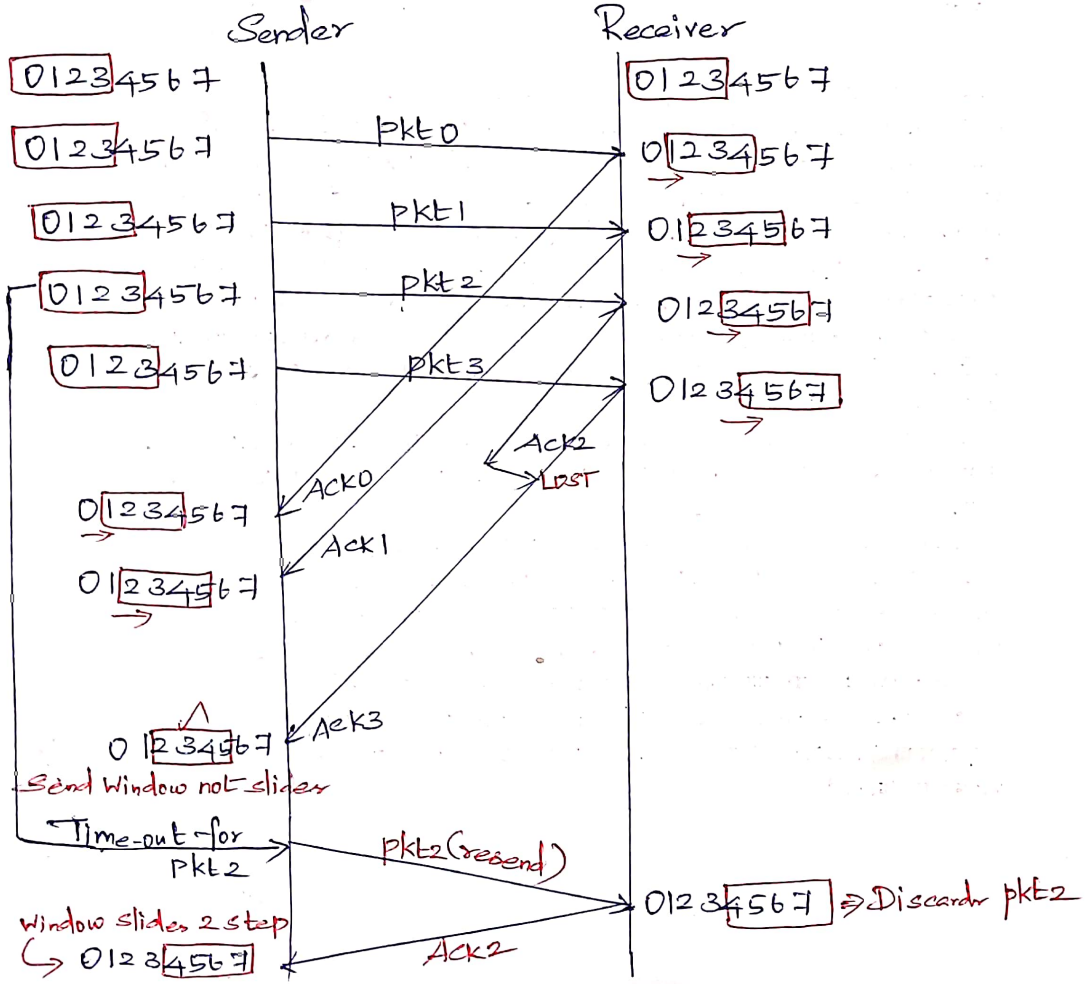
Send window and Receive window

Below diagram shows the send and receive window



Selective Repeat protocol (SR) (contd..)

Example 1: (ACK is lost)



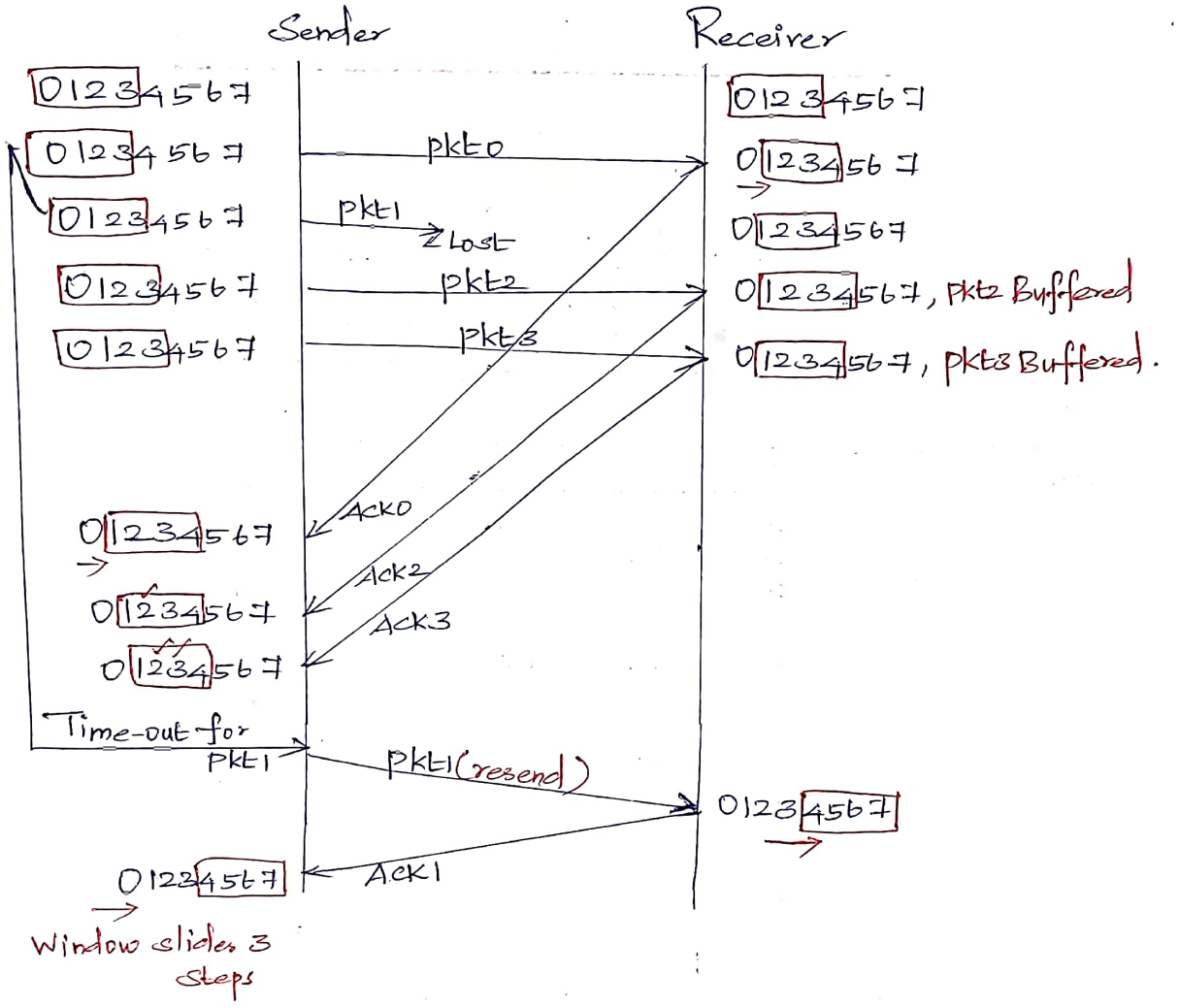
Selective Repeat protocol (SR) (contd..)

Example 1: (ACK is lost)

- In the example shown in the previous slide, ACK 2 is lost.
- Receiver after receiving pkt3, it is going to give ACK3. ACK3 represents packet3 is successfully received by the receiver . ACK3 does not mean about previous packets
- Since sender does not receives ACK for pkt2 it is going to resend pkt2. After receiving ACK2 send window slides 2 steps ahead

Selective Repeat protocol (SR) (contd..)

Example 2: (Data is lost)



Selective Repeat protocol (SR) (contd..)

Example 2: (Data is lost)

- In the example shown in the previous slide, pkt1 is lost.
- Receiver after receiving pkt2 and pkt3, it put these packets in the buffer since pkt1 still not received at receiver side
- Sender won't slide its window for ACK2 and ACK3, since ACK1 is not received by the sender
- After timeout, sender resends pkt1
- After receiving ACK1, sender slides its window three steps ahead
- In case of packet loss, sender is going to retransmit only the lost packet (ie) it is selectively repeating the packet that is actually lost and not all packets as in GBN

Connection Oriented Transport: TCP

- TCP(Transmission Control Protocol) is the transport layer connection-oriented, reliable protocol
- Since TCP provides error detection, retransmissions, cumulative acknowledgement, timers and header fields for sequence number and acknowledgment number, we are saying TCP as a reliable data transfer protocol
- TCP is working by combining the features of both Go-Back-N and Selective Repeat

TCP Segment and Maximum Segment Size(MSS)

- In TCP, the term called as segment is used to represent the data. The maximum size of a segment is fixed using a parameter called as Maximum Segment Size(MSS).
- Actually the MSS is set using Maximum Transmission Unit(MTU). MTU is the parameter that represents the maximum size of Data link layer frame. Based on this MTU value only MSS value can be set

TCP (contd..)

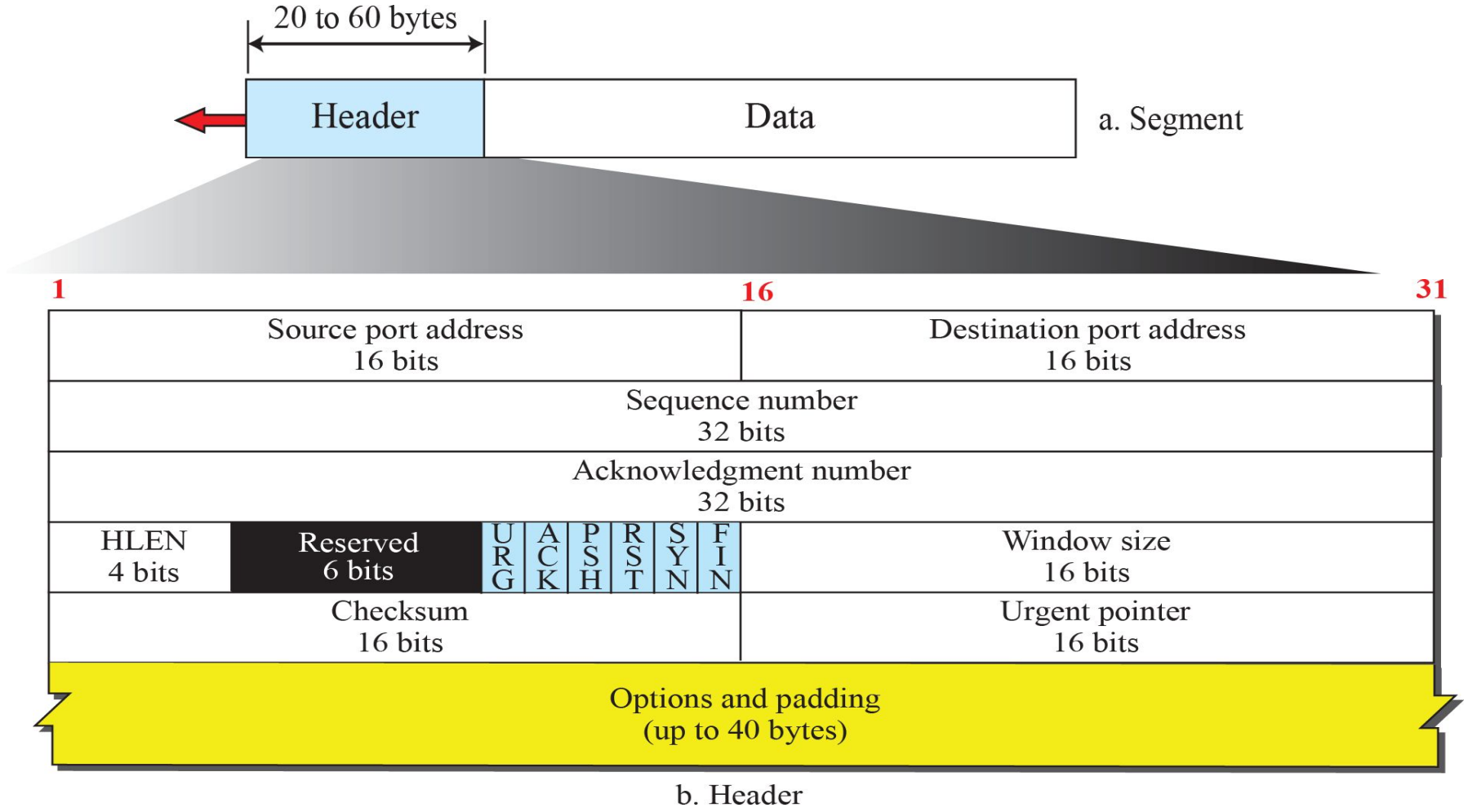
TCP Segment Structure

- TCP segment consists of header field and data field
- Header field ranges from 20 to 60 bytes
- The maximum size of data field in the TCP segment is set by Maximum Segment Size(MSS). If data size is greater than MSS then data is split into multiple segments

TCP (contd..)

TCP Segment Structure

The following diagram shows the different fields present in TCP header



b. Header

TCP (contd..)

TCP Segment Structure

Various fields in the TCP header are as follows:

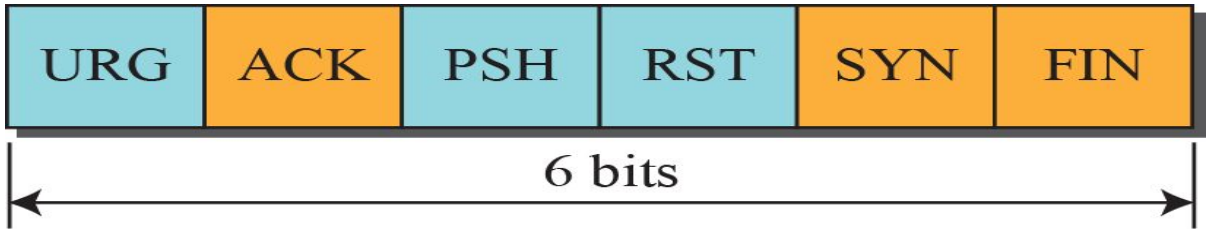
1. **Source port number (16 bits):** This field defines the port number of process running at source side
2. **Destination port number (16 bits):** This field defines the port number of process running at destination side
3. **Sequence number (32 bits):** This field defines the number assigned to each segment
4. **Acknowledgment number (32 bits):** This field defines the number assigned to acknowledgment segment
5. **Header length (4 bits):** This field contains the length of header. The value in this field should be multiplied by 4 to get actual header length

TCP (contd..)

TCP Segment Structure

Various fields in the TCP header are as follows:

6. **Control field/ flag field (6 bits):** The 6 bits control field says what type of segment it is.



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

TCP (contd..)

TCP Segment Structure

Various fields in the TCP header are as follows:

7. **Window size (16 bits):** This field is used to announce the window size of each side to the other side. This window size is also called as receive window size
8. **Checksum (16 bits):** This field is used for error detection
9. **Urgent Pointer (16 bits):** This field is used when URG(Urgent bit) bit is set in the control field. The field contains the byte number where the urgent data ends in the data field
10. **Options (0 to 40 bytes):** This field can be used to include some extra information if needed

TCP (contd..)

TCP Segment Structure

Urgent Data in TCP

- When part of the data is to be handled in a special way by the receiver, then that part of the data can be marked as Urgent data
- When segment contains urgent data, then URG(urgent) bit is set to 1.
- Urgent data will present in the beginning of the segment
- Urgent pointer field in the header contains end of the urgent data

Example:

- If segment sequence number is 15000 and the value in the urgent pointer field is 200 means, bytes from 15000 to15200 is the urgent data and remaining are the normal data

TCP (contd..)

Sequence Number and Acknowledgement number in TCP

Byte Number

- TCP receives bytes of data from application layer
- Each byte have a number called byte number

Sequence Number

- Each segment is assigned a number called Sequence number
- Sequence number of every segment is the first byte number in that segment
- The sequence number of the first segment is called Initial Sequence Number(ISN)
- The sequence number of the next segment is the sequence number of previous segment + number of bytes carried by previous segment
- In case of TCP sequence number field is of 32 bits (ie) range of sequence number is from 0 to $2^{32} - 1$
- Sequence number will start from any value in that range

TCP (contd..)

Sequence Number Example:

- Suppose a TCP connection is transferring a file of 5000 bytes. The first byte is numbered 1. What are the sequence numbers for each segment if data are sent in five segments, each carrying 1000 bytes?

Solution:

The following are the sequence number for each segment

Segment 1 → Sequence Number: 1, Byte Range: 1 to 1000

Segment 2 → Sequence Number: 1001 , Byte Range: 1001 to 2000

Segment 3 → Sequence Number: 2001 , Byte Range: 2001 to 13000

Segment 4 → Sequence Number: 3001 , Byte Range: 3001 to 14000

Segment 5 → Sequence Number: 4001 , Byte Range: 4001 to 15000

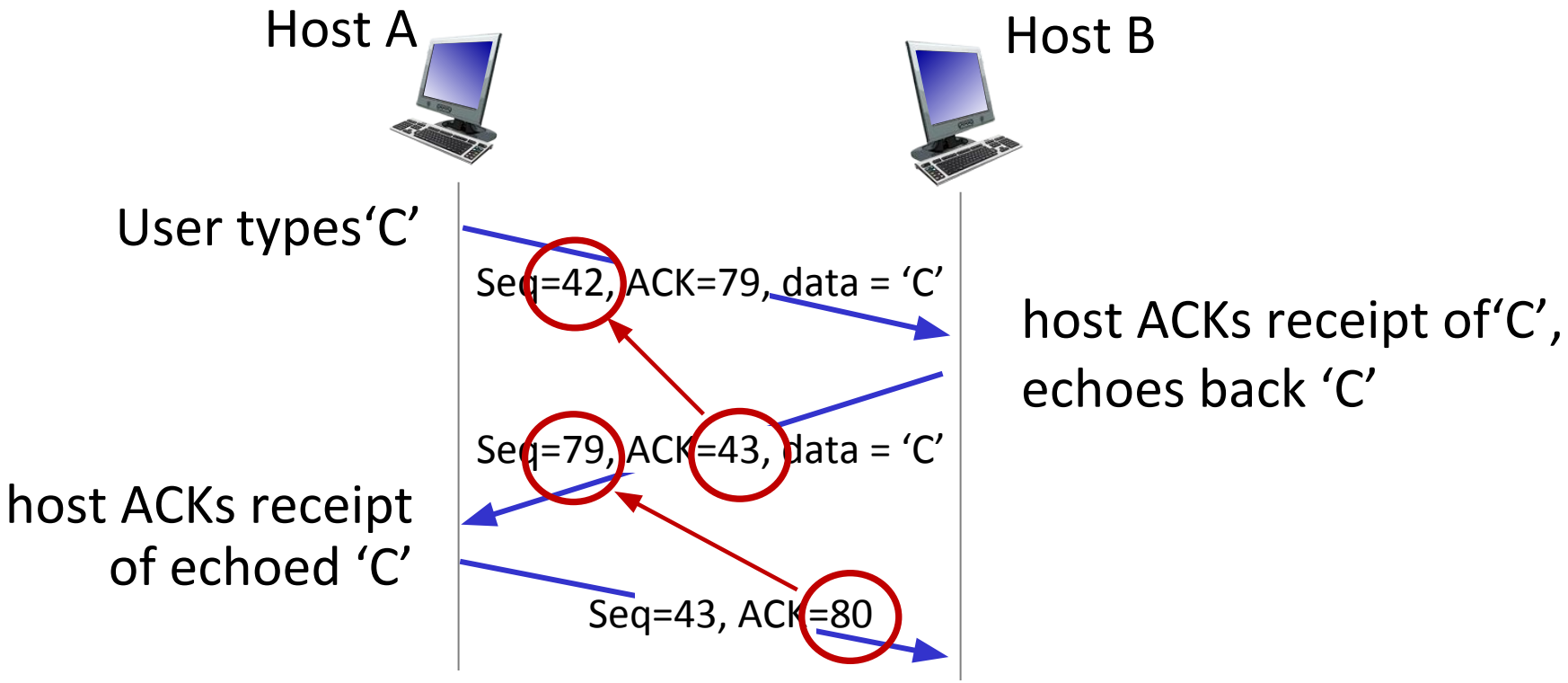
TCP (contd..)

Acknowledgment Number

- Along with sequence number, acknowledgment number also used to confirm whether a data has been received successfully or not
- Acknowledgment number is cumulative
- And also acknowledgment number indicates the number of next byte the receiver expects

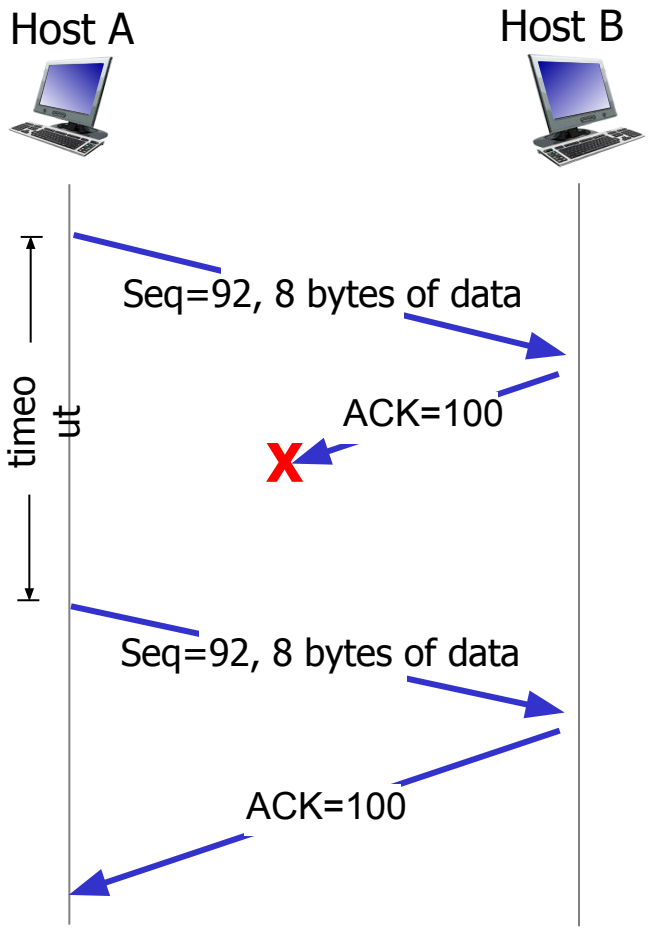
TCP (contd..)

Example scenario for sequence no. and acknowledgement no. in TCP

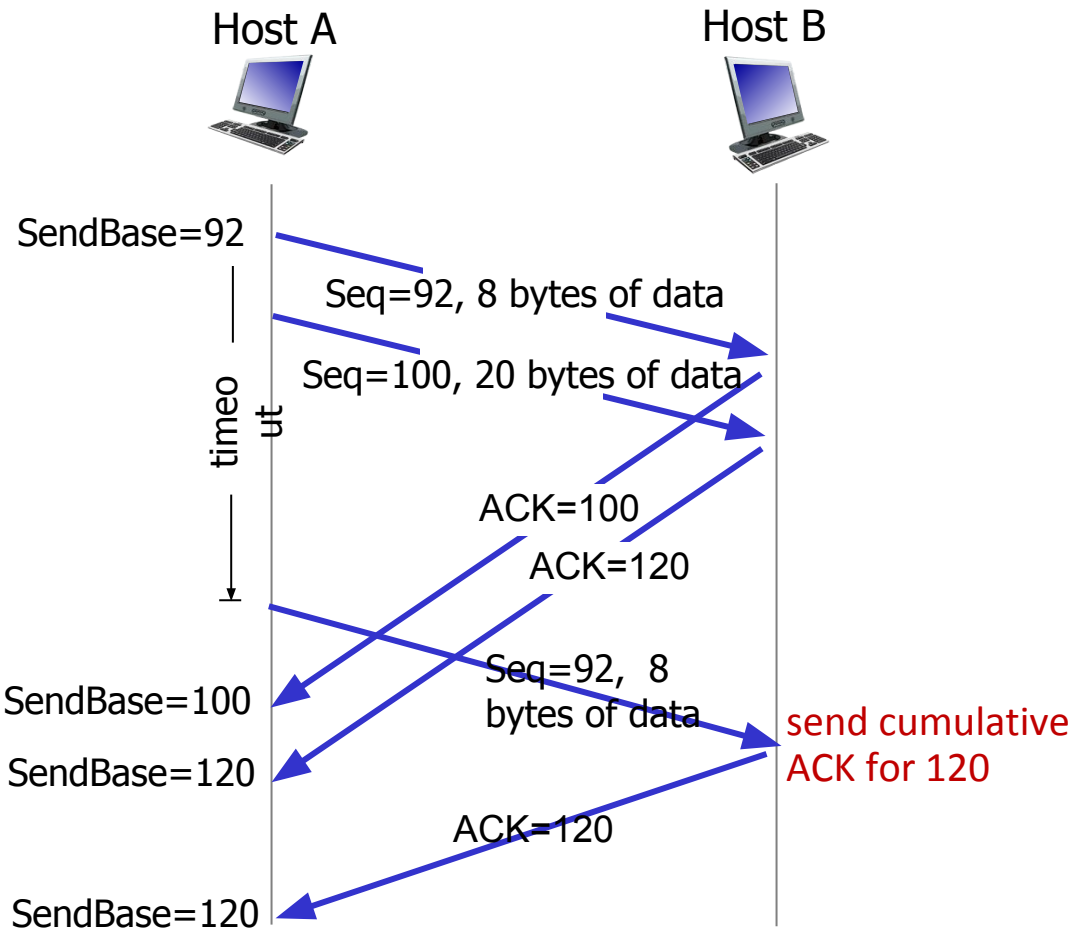


TCP (contd..)

Example scenario for Retransmission



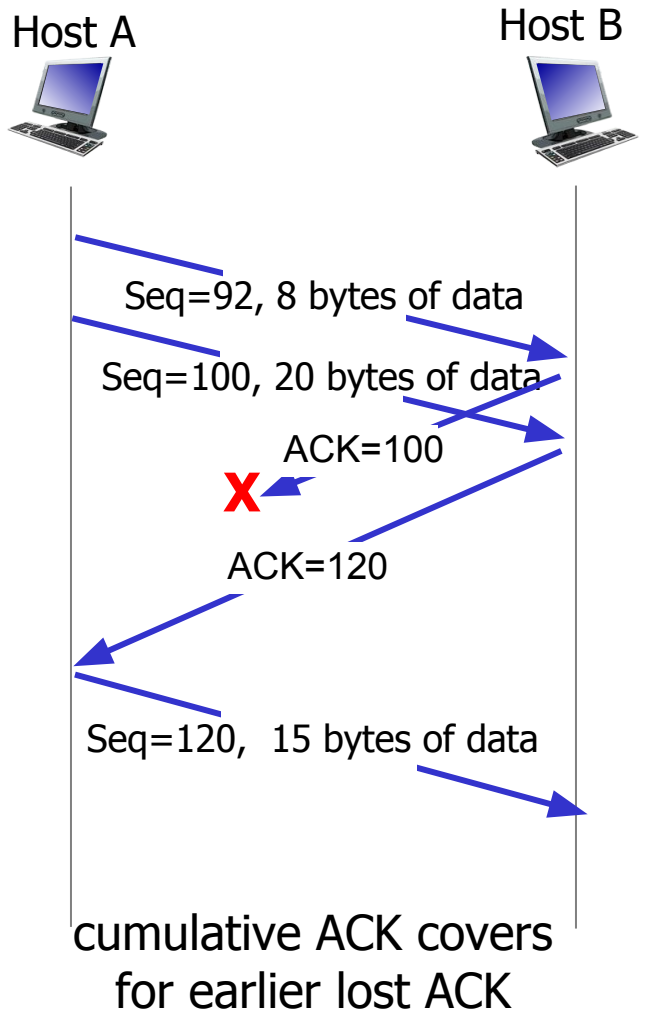
lost ACK scenario



premature timeout

TCP (contd..)

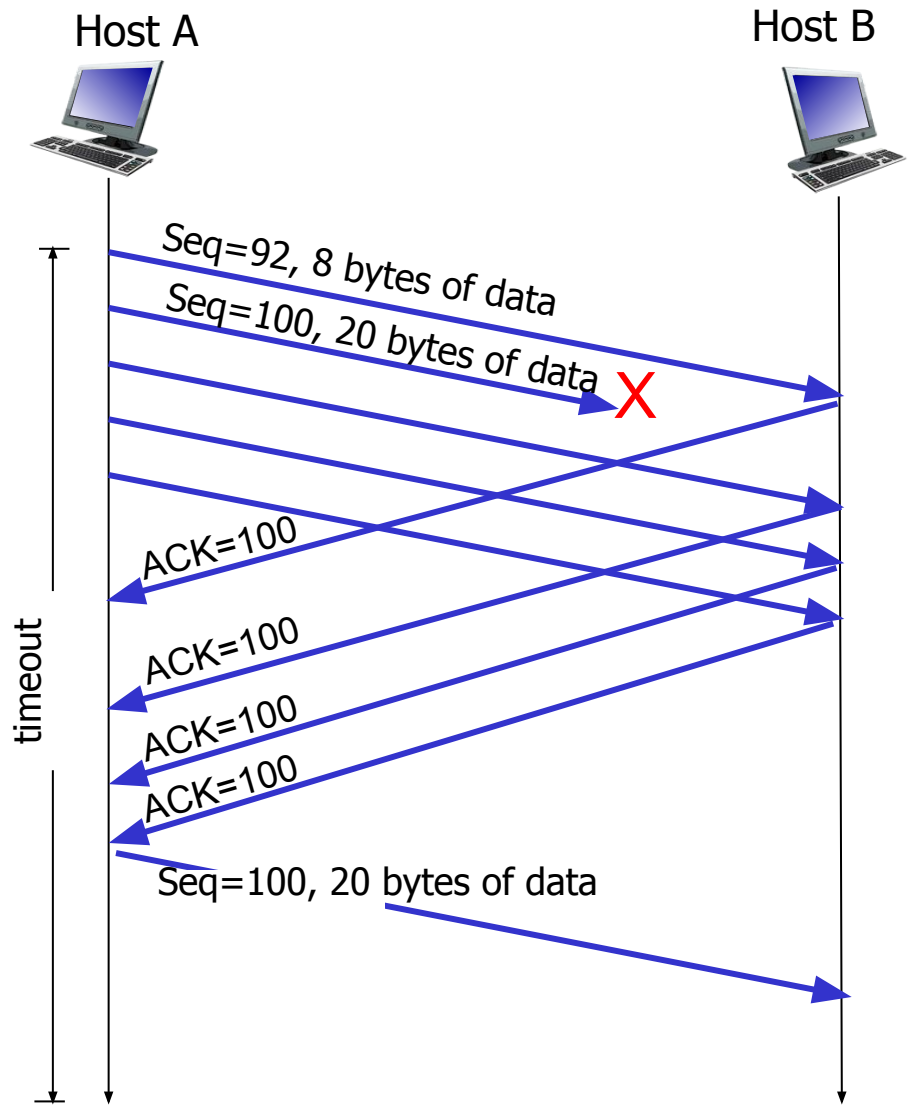
Example scenario for Retransmission



TCP (contd..)

TCP Fast Retransmit

- In case of TCP if sender receives 3 duplicate ACKs, then before timeout the sender is going to retransmit the data. This concept is called as Fast Retransmit (shown in the diagram at right).
- In the diagram, segment with seq no. 100 is lost. Receiver gives ACK 100 only for the next 3 segments. Sender upon receiving 3 duplicate ACK 100 it retransmits segment with seq no. 100 before timeout. This is called as Fast Retransmit



TCP (contd..)

TCP Retransmission Policy

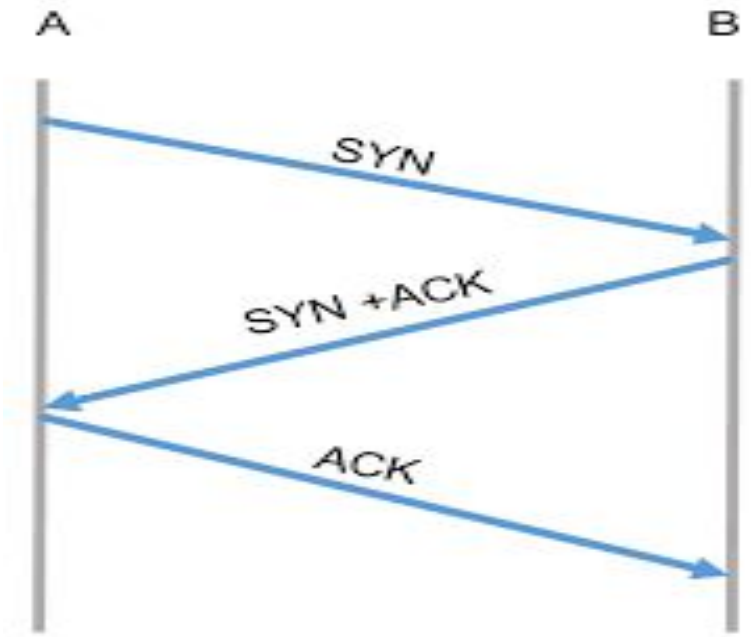
- In TCP if an ACK gets lost means upon receiving next ACK(cumulative ACK), the sender will identify that the data has been received successfully by the receiver
- Whereas if data get lost means, receiver will continuously give ACK of last correctly received data for the upcoming data also. Once timeout occurs sender retransmits the data
- This is how TCP handles ACK lost and Packet lost concept

TCP (contd..)

TCP Connection Establishment and Termination

TCP Connection Establishment

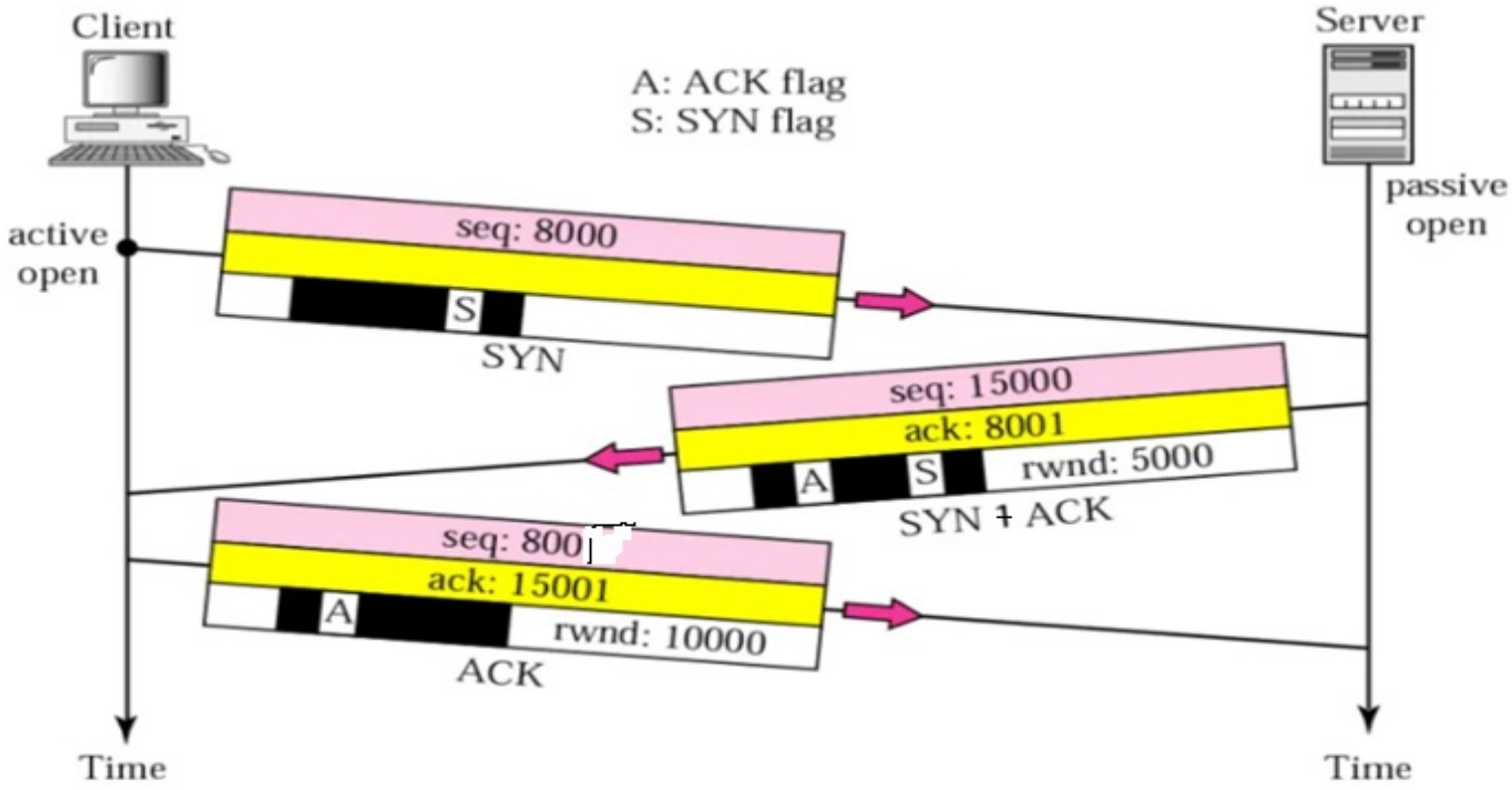
- Connection establishment in TCP is done using three-way handshaking as shown in diagram
- During connection establishment SYN (Synchronize) segment is send from sender to receiver. Receiver gives ACK for this SYN plus it also sends the SYN to sender. Sender gives ACK finally. In these three segments except last segment(ACK) remaining segments carries no data



Three way handshake

TCP (contd..)

TCP Connection Establishment (contd..)



TCP (contd..)

TCP Connection Establishment (contd..)

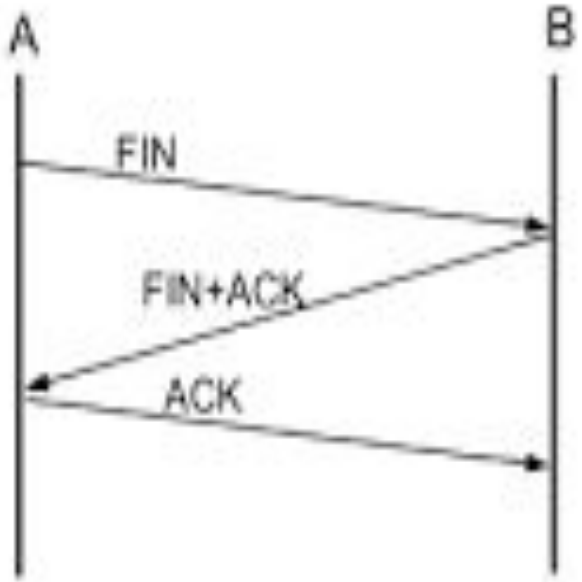
- The client first send SYN segment with its initial sequence number 8000. A SYN segment cannot carry data, but it consumes one sequence number. SYN bit is set in control field
- After that server also sends its SYN segment along with acknowledgment(ACK) for SYN send by client. Server starts its sequence number from 15000. Acknowledgment number is going to be next sequence number (ie) 8001. A SYN + ACK segment cannot carry data, but it does consume one sequence number. SYN and ACK bit is set in control field. Window size of server(rwnd:5000) is also specified in this segment
- Finally client sends ACK for SYN sent by server with ACK number 15001 and sequence number 8001. ACK bit only set in control field

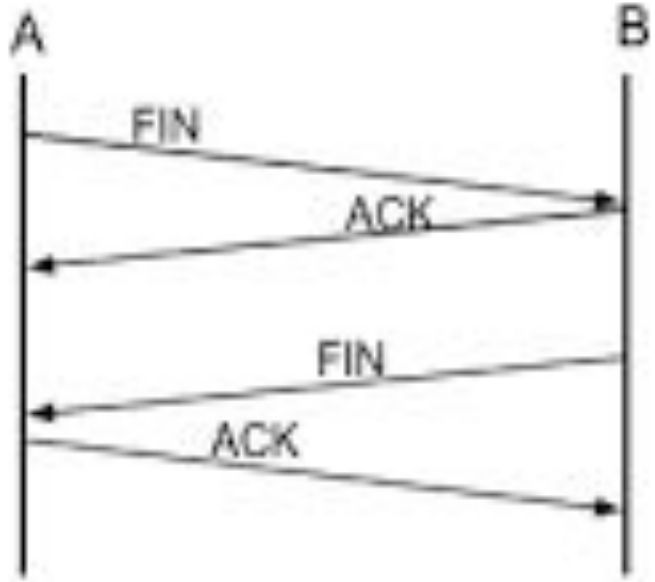
TCP (contd..)

TCP Connection Termination

- Connection Termination in TCP is done using three-way handshaking or four-way handshaking as shown in diagram
- During connection termination FIN (Finish) segment is send from sender to receiver. Receiver gives ACK for this FIN plus it also sends the FIN to sender. Sender gives ACK finally. (3-way handshake)
- Connection Termination can be done in Four-way handshaking also where server gives ACK only and after sometimes it gives FIN to client. Finally client gives ACK

3-way Handshake



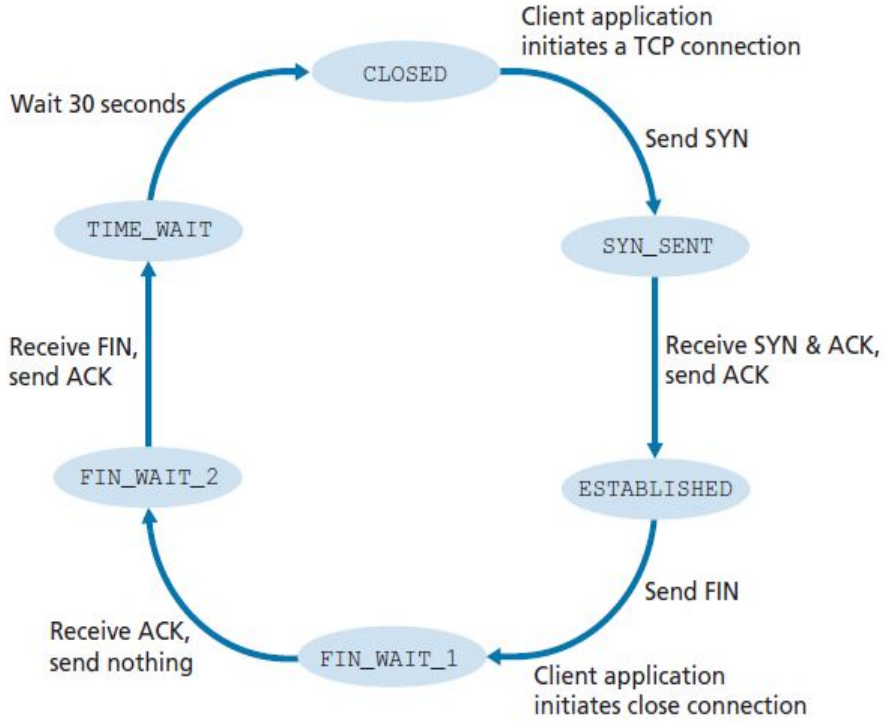


4-way Handshake

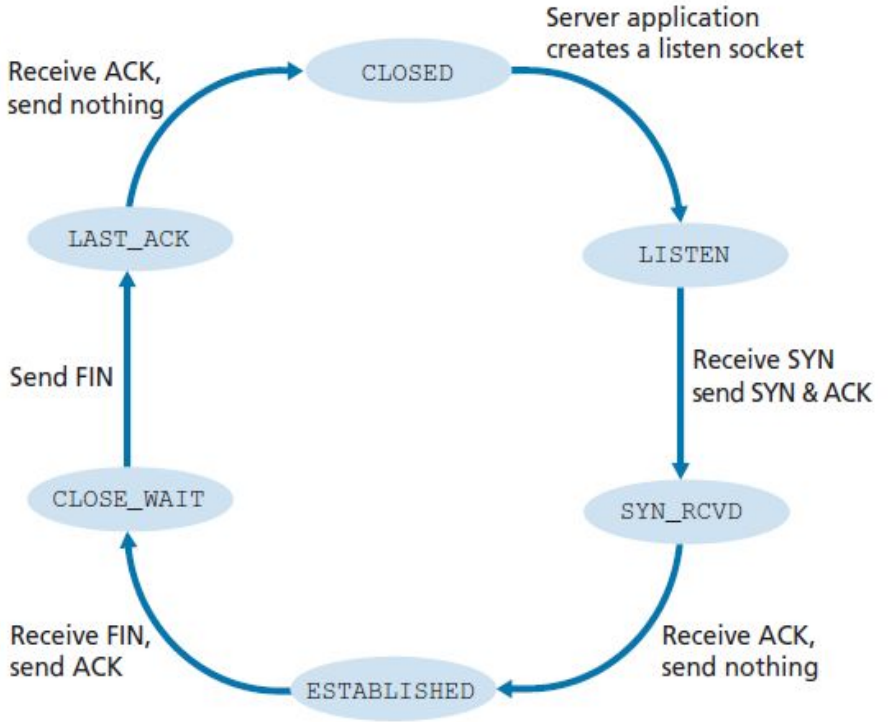
TCP (contd..)

TCP Connection Establishment and Termination(contd..)

- Below FSM diagram shows the client and server state during connection establishment and termination



client



server

TCP (contd..)

Round-Trip Time Estimation and Timeout

- In case of TCP, when sender sends data it starts a timer and wait for ACK. If within that time ACK is received for that data means no problem. No ACK is not received by sender then sender retransmits the data. How long does the sender waits for ACK (or) when timeout should occur for a data
- This time is calculated using Round-Trip Time (RTT)
- Consider sampleRTT as RTT for one segment. Since RTT vary from one segment to another segment it is necessary to take average of all segments RTT. This average RTT is called Estimated RTT and is calculated as follows:

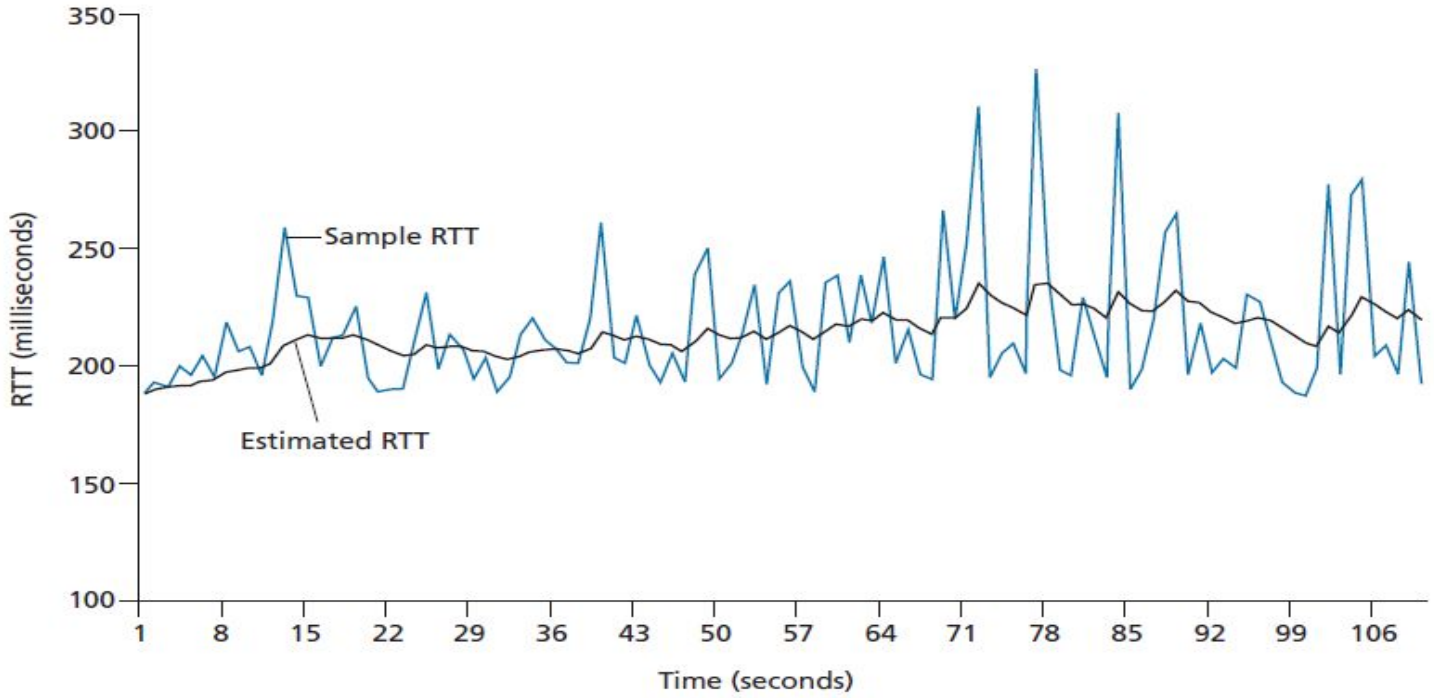
$$EstimatedRTT = (1 - \alpha) \cdot EstimatedRTT + \alpha \cdot SampleRTT$$

- The recommended value for α=0.125

TCP (contd..)

Round-Trip Time Estimation and Timeout (contd..)

- Once RTT is calculated for every segment(sample RTT), the estimated RTT is calculated using formula in previous slide



- From above graph we can infer, Estimated RTT smoothes out the actual sampleRTT value

TCP (contd..)

Round-Trip Time Estimation and Timeout (contd..)

- How much sample RTT deviates from Estimated RTT is represented as DevRTT and is calculated as below:

$$DevRTT = (1 - \beta) \cdot DevRTT + \beta \cdot | SampleRTT - EstimatedRTT |$$

- The recommended value of β is 0.25
- Based on this Estimated RTT and DevRTT the timeout at sender side is calculated as below:

$$TimeoutInterval = EstimatedRTT + 4 \cdot DevRTT$$

TCP (contd..)

TCP Flow Control

- TCP provides a flow-control service to its applications to eliminate sender overflowing the receiver's buffer.
- Flow control is called as speed matching service: matching the rate at which the sender is sending against the rate at which the receiving application is reading.
- TCP provides flow control by maintaining a variable called the receive window
- At receiver side a buffer is maintained for holding packets called as Receive Buffer. How much free space is available at receive buffer is indicated by receive window and this receive window value is given to sender using the field window size in TCP header

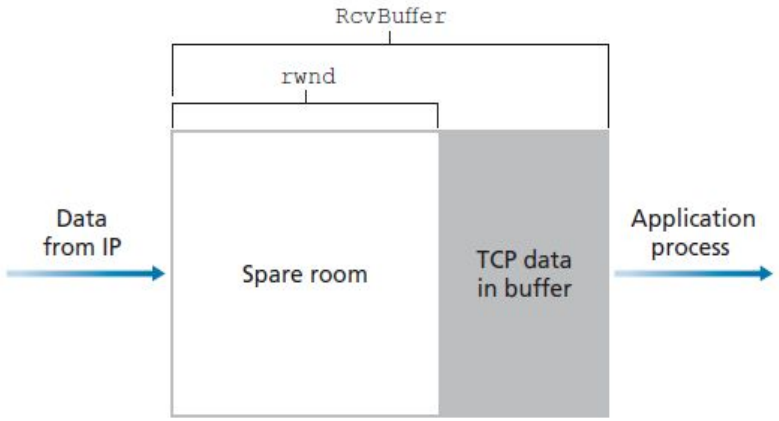


Figure 3.38 ♦ The receive window (rwnd) and the receive buffer (RcvBuffer)

TCP (contd..)

TCP Flow Control (contd..)

- Once sender gets the receive window size (free space in receive buffer), sender sets its send window size
- By setting send window size that equals to receive window, TCP is ensuring, sender is not going to overflow the receive buffer thereby providing flow control service

TCP (contd..)

TCP Flow Control (contd..)

- At receiver side in order to ensure not to overflow the allocated buffer, we must have

$$\text{LastByteRcvd} - \text{LastByteRead} <= \text{RcvBuffer}$$

- The size of receive window is calculated as,

$$\text{rwnd} = \text{RcvBuffer} - [\text{LastByteRcvd} - \text{LastByteRead}]$$

- At sender side the number of bytes sent should follow the below formula in order to ensure flow control

$$\text{LastByteSent} - \text{LastByteAcked} <= \text{rwnd}$$

TCP (contd..)

Principles of Congestion Control

- Congestion means traffic in the network
- In order to reduce congestion in the network TCP uses congestion control
- Whenever a network is identified as congested, the sender is going to slow down its transmission rate
- Both in flow control and congestion control, sender is only responsible to slow down its transmission rate
- In congestion control, number of segments to transmit is controlled using variable called congestion window(cwnd)
- cwnd is changeable according to congestion in the network
- cwnd and rwnd (receive window used for flow control) together defines the size of send window

Send window size = minimum (rwnd, cwnd)

TCP (contd..)

Flow control vs Congestion Control

flow control

one sender too fast for one receiver

congestion control

too many senders, sending too fast

TCP (contd..)

Causes and cost of Congestion

Three scenarios it is been used here to identify the causes and cost 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

TCP (contd..)

Causes and cost of Congestion (contd..)

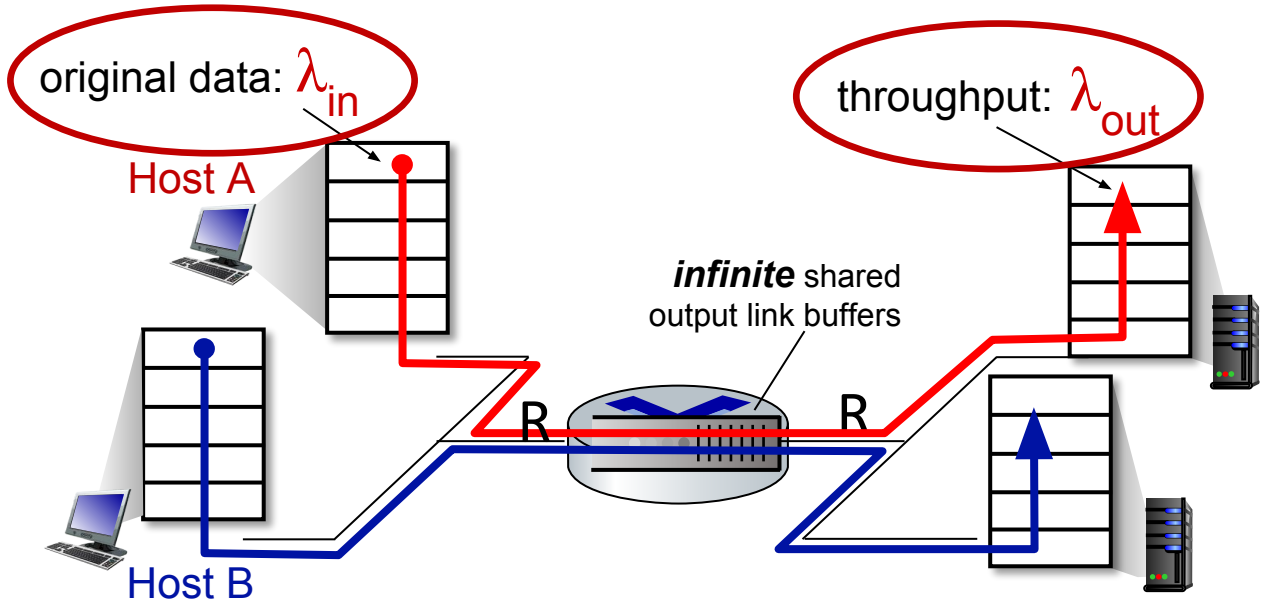
Scenario 1: Two Senders, a Router with Infinite Buffers

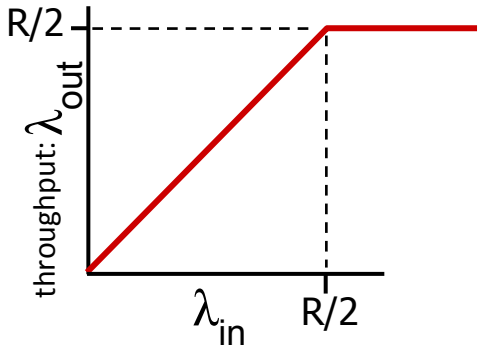
- In the first scenario the data is send via router which is having infinite capacity
- Sending rate is represented as : λ_{in} and receiving rate is represented as λ_{out}
- Since incoming and outgoing capacity of router is R and two systems A and B will share this link capacity will have a maximum throughput of R/2

TCP (contd..)

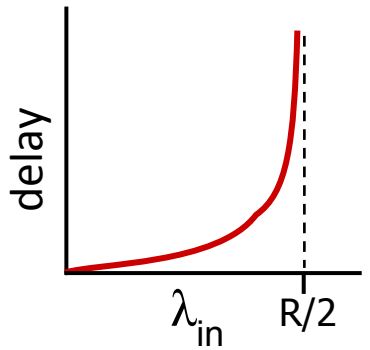
Scenario 1

- When sending rate λ_{in} is less than or equal to R/2 the throughput will be same as like that of sending rate. When sending rate is greater than R/2 then throughput will remain same
- Delay also increases when sending rate equals to R/2





maximum per-connection throughput: R/2



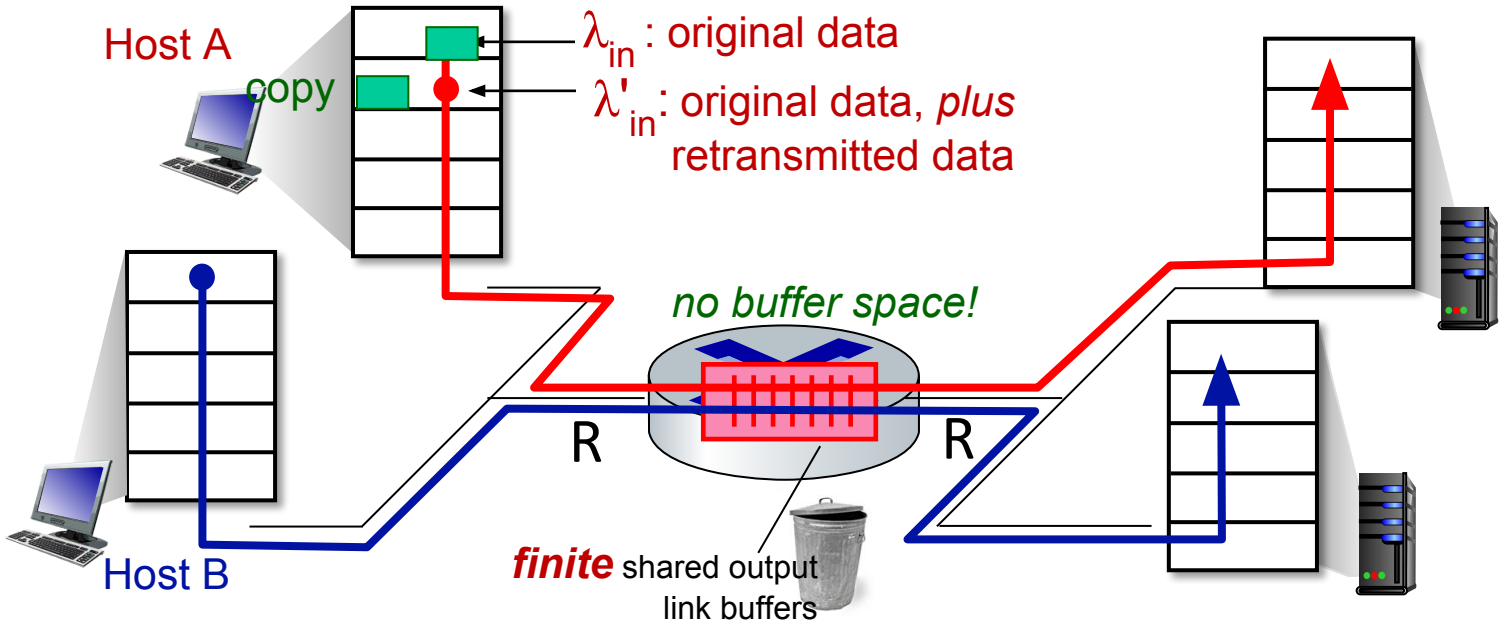
large delays as arrival rate λ_{in} approaches capacity

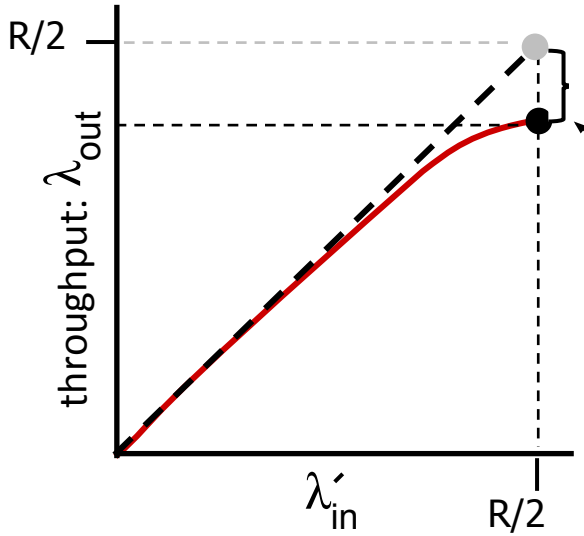
TCP (contd..)

Causes and cost of Congestion (contd..)

Scenario 2: Two Senders and a Router with Finite Buffers

- packets can be lost (dropped at router) due to full buffer
- Sender resends if packet known to be lost





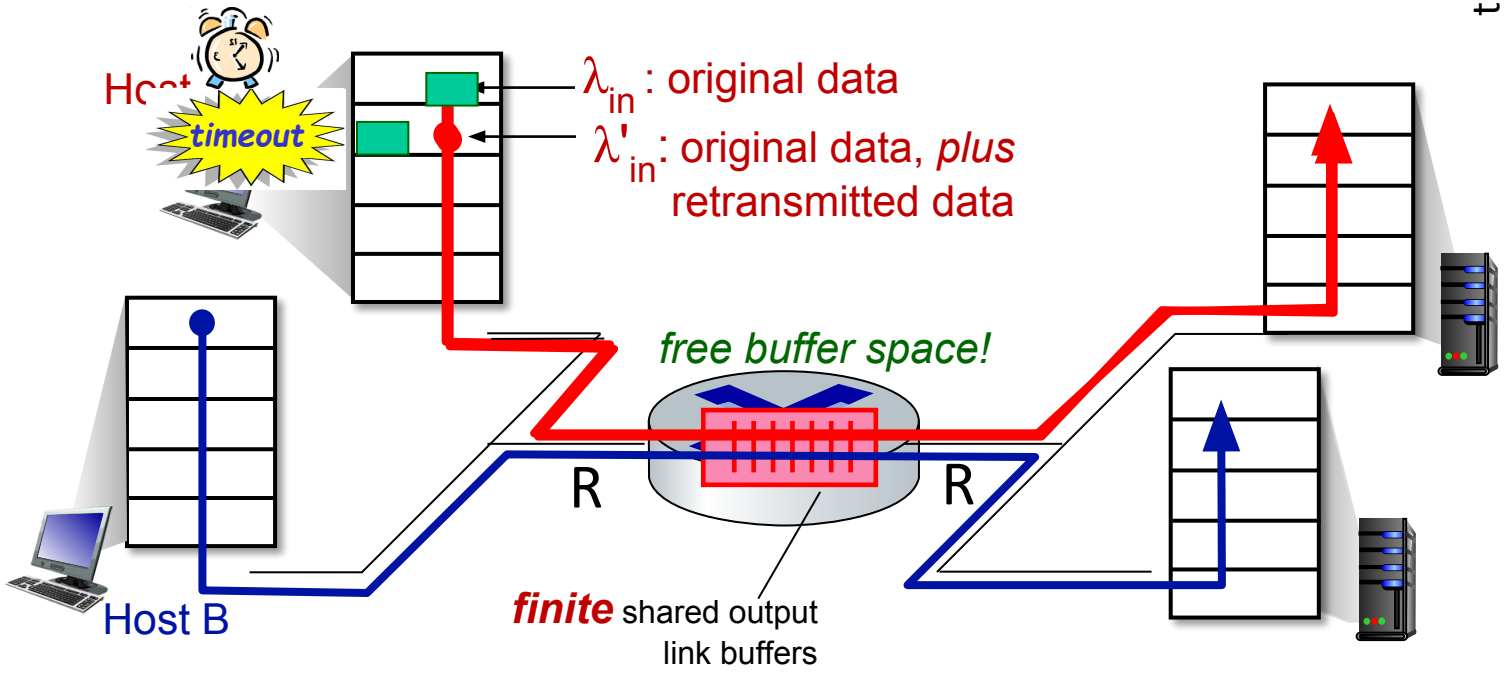
Retransmission wastes actual throughput (see above graph)

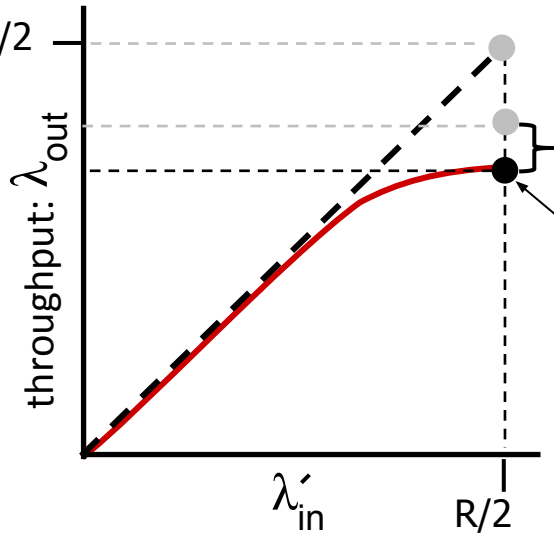
TCP (contd..)

Causes and cost of Congestion (contd..)

Scenario 2: Two Senders and a Router with Finite Buffers

- packets can be lost(dropped at router)due to full buffer
- but sender times can time out prematurely, sending two copies, both of which are delivered (duplicate packets)





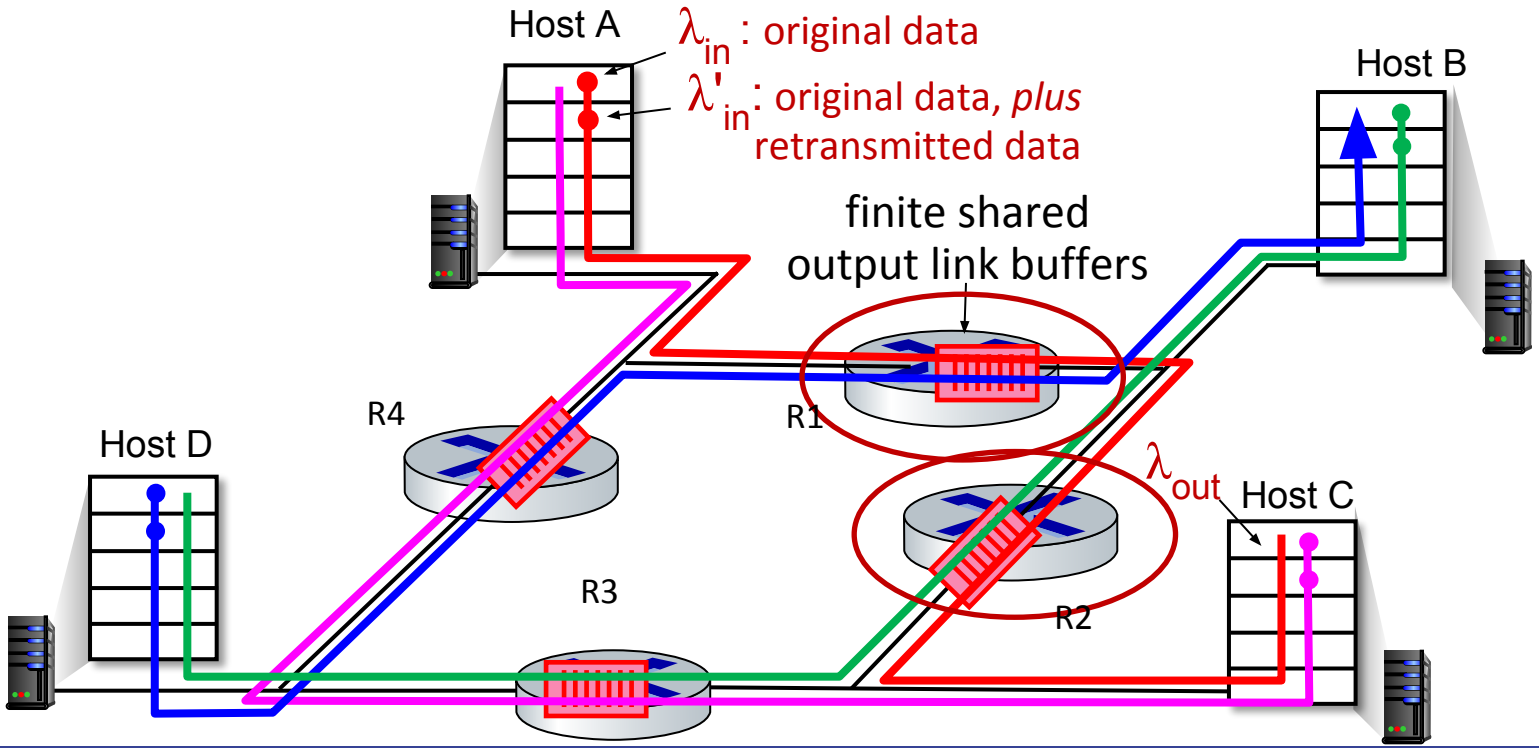
Duplicate transmission also wastes actual throughput further (see above graph)

TCP (contd..)

Causes and cost of Congestion (contd..)

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

- D sends data to B via R3 and R2
- A also sends data to C via R1 and R2. While A's data reaches R2 since R2 is busy in sending D's data to B, it drops data by A. Because of this, the transmission capacity that was used at each of the upstream links to forward that data to the point at which it is dropped ends up having been wasted.



TCP (contd..)

Causes and cost of Congestion (contd..)

The following things are inferred from the three scenarios

Scenario 1:

- large queuing delays are experienced as the sending rate equals to R/2

Scenario 2:

- Packet loss and retransmission due to it decreases effective throughput
- Un-needed duplicates further decreases effective throughput

Scenario 3:

- upstream transmission capacity / buffering wasted for packets lost in the downstream

All these as a whole will account for the cost of congestion

TCP (contd..)

Approaches to Congestion Control

- 2 approaches to congestion control,
  1. End-to-end congestion control
  2. Network-assisted congestion control

1. End-to-end congestion control

- In an end-to-end approach to congestion control, the network layer provides no explicit support to the transport layer for congestion-control purposes.
- The presence of network congestion must be inferred by the end systems based only on observed network behavior
- TCP congestion control is based on this approach

2. Network-assisted congestion control

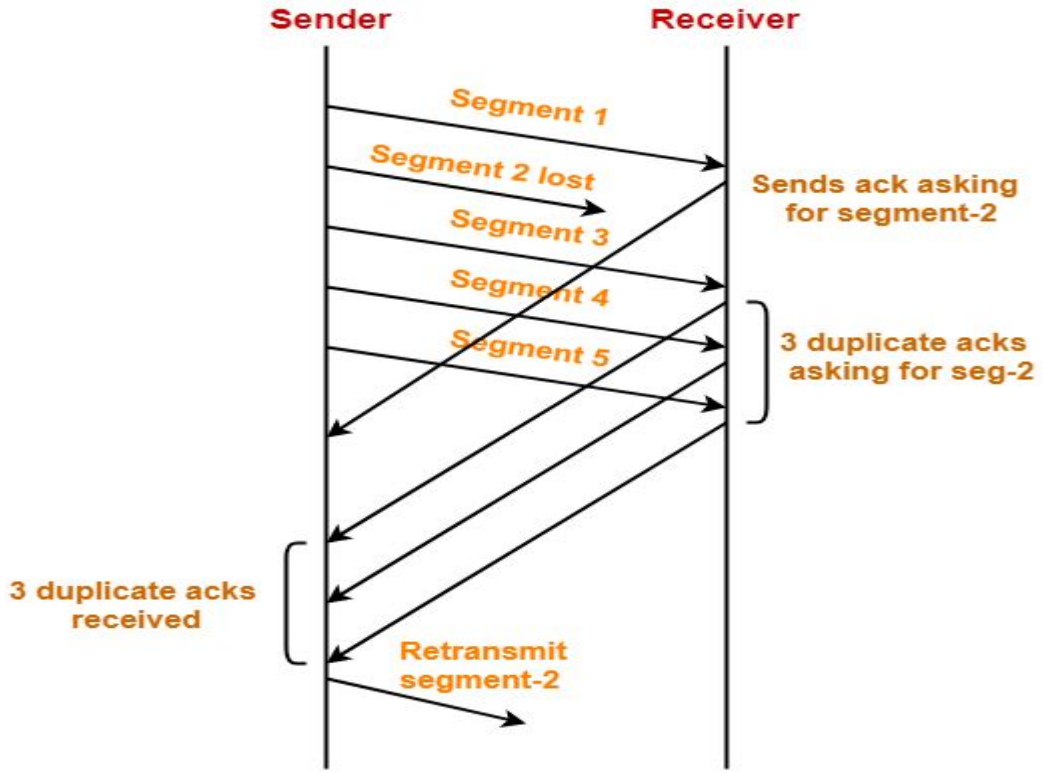
- With network-assisted congestion control, routers provide explicit feedback to the sender and/or receiver regarding the congestion state of the network.

TCP (contd..)

TCP Congestion Control

Congestion Detection:

- We can say a network is congested if anyone of the following events occur:
  1. Time-out - heavy congestion
  2. Receiving three duplicate ACKs - light congestion



Retransmission after receiving 3 duplicate acks

TCP (contd..)

TCP Congestion Control (contd..)

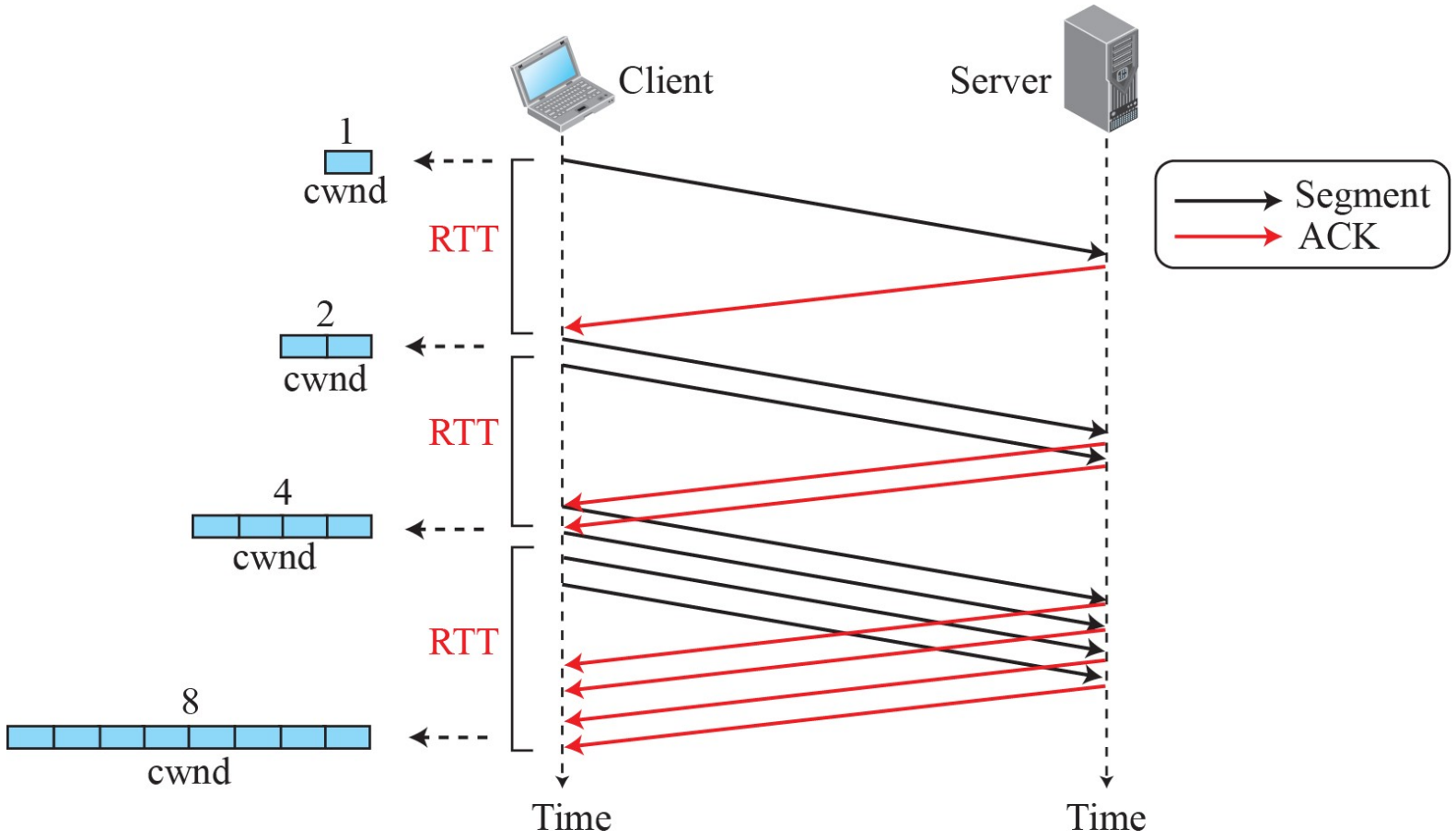
Congestion control algorithms:

- Handling congestion is based on three algorithms:
  1. Slow start: Exponential Increase
  2. Congestion avoidance: Additive increase
  3. Fast Recovery

TCP (contd..)

TCP Congestion Control (contd..)

1. Slow start: Exponential Increase



TCP (contd..)

TCP Congestion Control (contd..)

1. Slow start: Exponential Increase

- If an ACK arrives cwnd=cwnd+1

Start → cwnd = 1 → 2^0

cwnd = cwnd + 1 = 1 + 1 = 2→ 2^1

cwnd = cwnd + 2 = 2 + 2 = 4→ 2^2

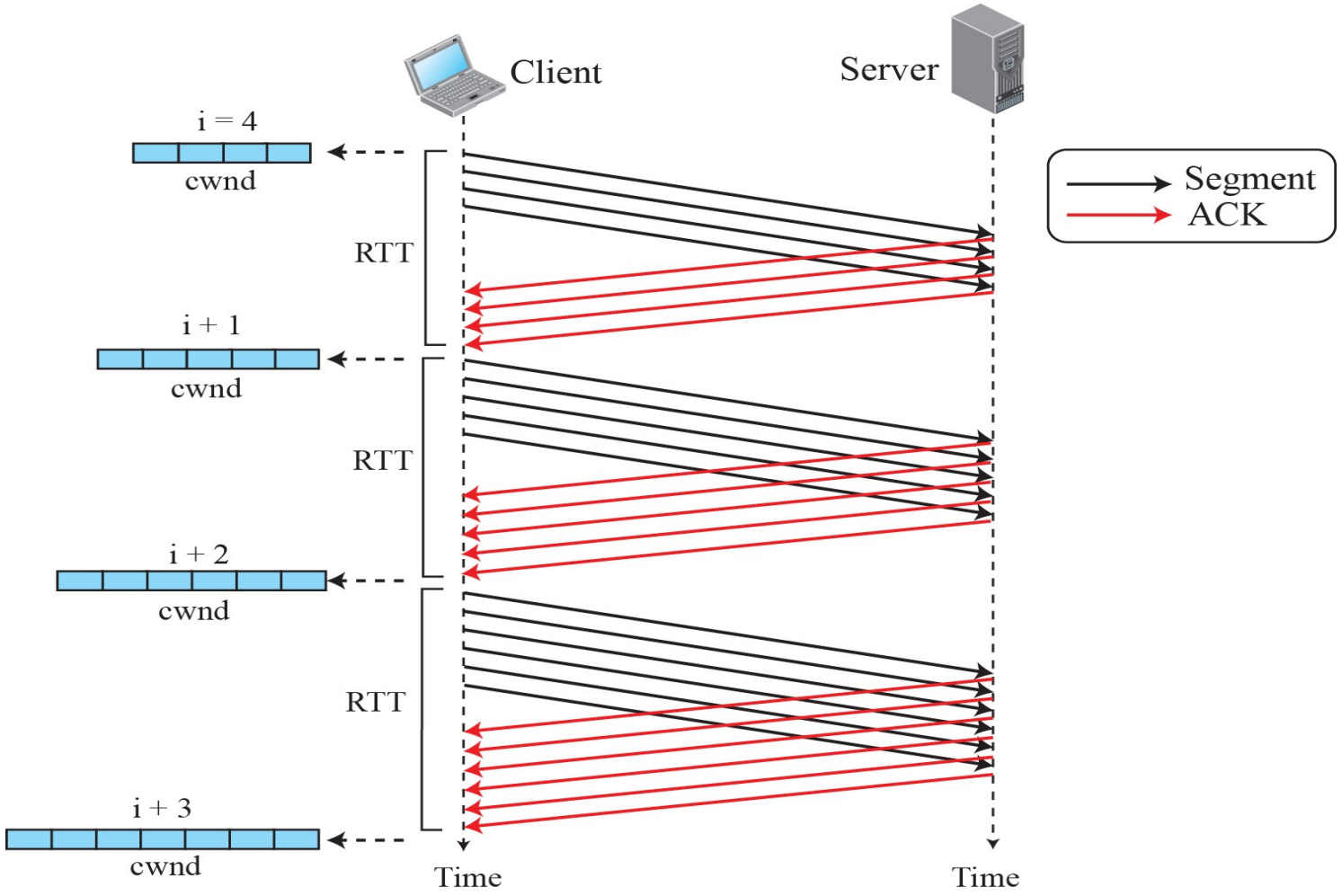
cwnd = cwnd + 4 = 4 + 4 = 8→ 2^3

- In slow-start the size of congestion window increases exponentially until it reaches a threshold value called ssthresh (slow start threshold value)
- When it reaches threshold value slow start stops and next phase starts

TCP (contd..)

TCP Congestion Control (contd..)

2. Congestion avoidance, Additive Increase



TCP (contd..)

TCP Congestion Control (contd..)

2. Congestion avoidance, Additive Increase

- If an ACK arrives cwnd = cwnd+(1/cwnd)

Start → cwnd = i

cwnd = i + 1

cwnd = i + 2

cwnd = i + 3

- In this algorithm cwnd increases additively

TCP (contd..)

TCP Congestion Control (contd..)

3. Fast Recovery

- This fast recovery policy is used when 3 duplicate ACKs received.
- Here cwnd is increased in exponential manner only when duplicate ACK is received
- If an duplicate ACK arrives cwnd=cwnd+1
- Used in new version of TCP

TCP (contd..)

TCP Congestion Control (contd..)

Versions of TCP:

2 versions:

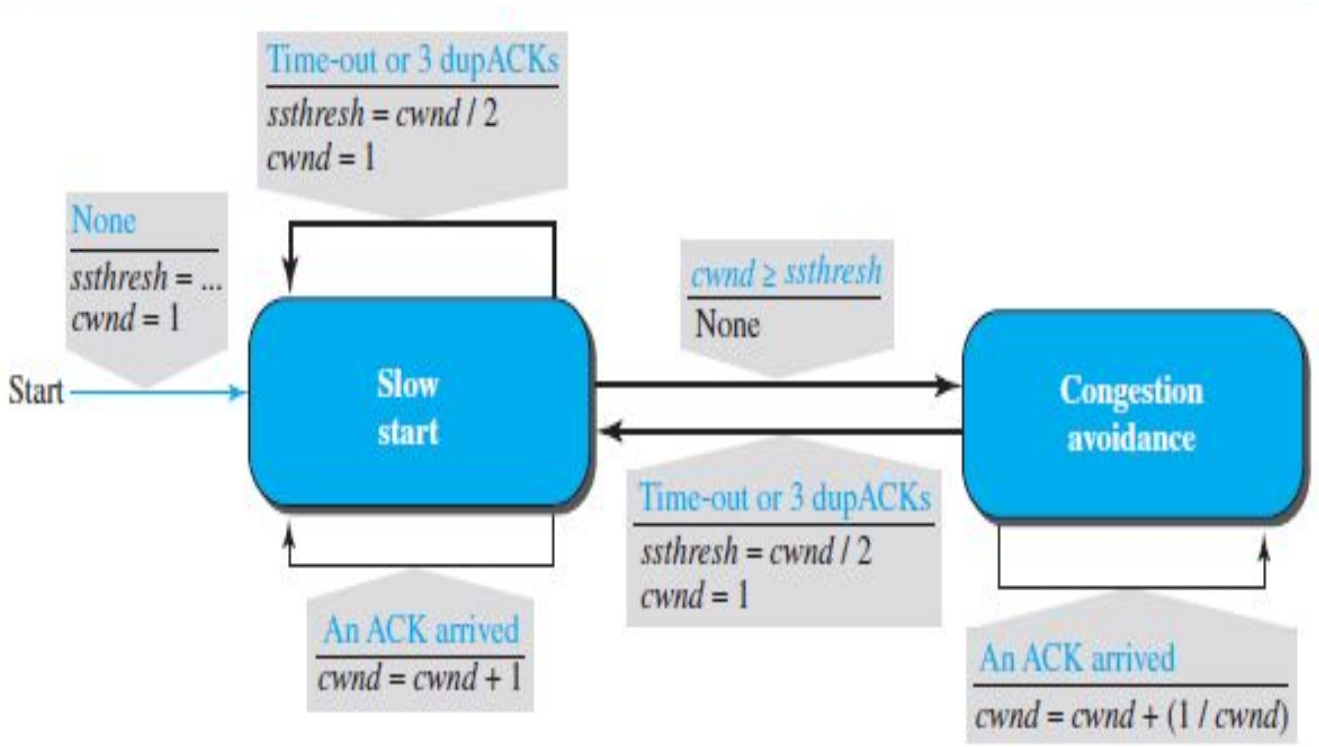
1. Taho TCP
2. Reno TCP

TCP (contd..)

TCP Congestion Control (contd..)

1. Taho TCP:

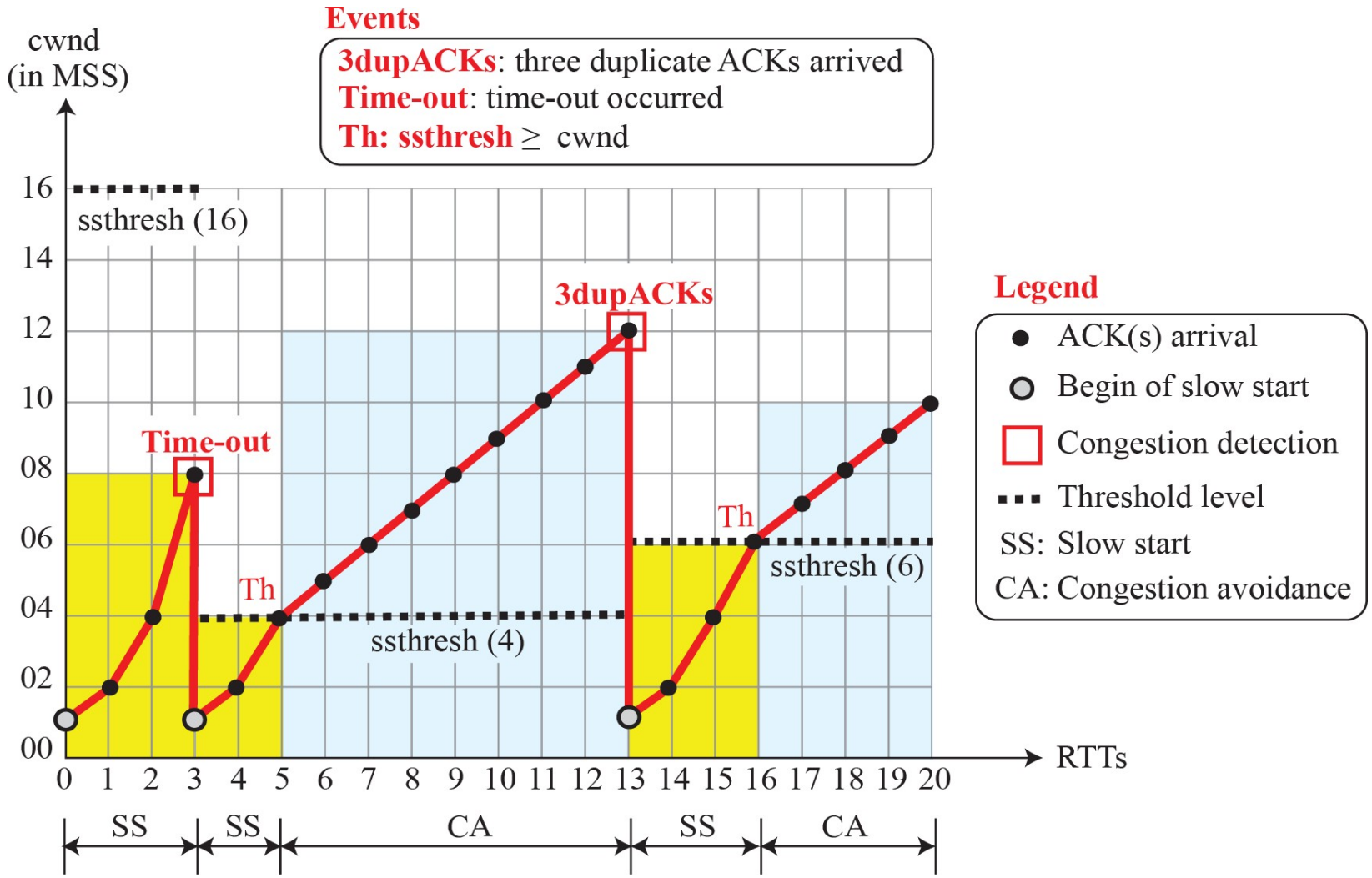
- Below diagram is the FSM for Taho TCP
- Treats both time-out and three duplicate ACKs condition in same manner



TCP (contd..)

TCP Congestion Control (contd..)

Example of Taho TCP

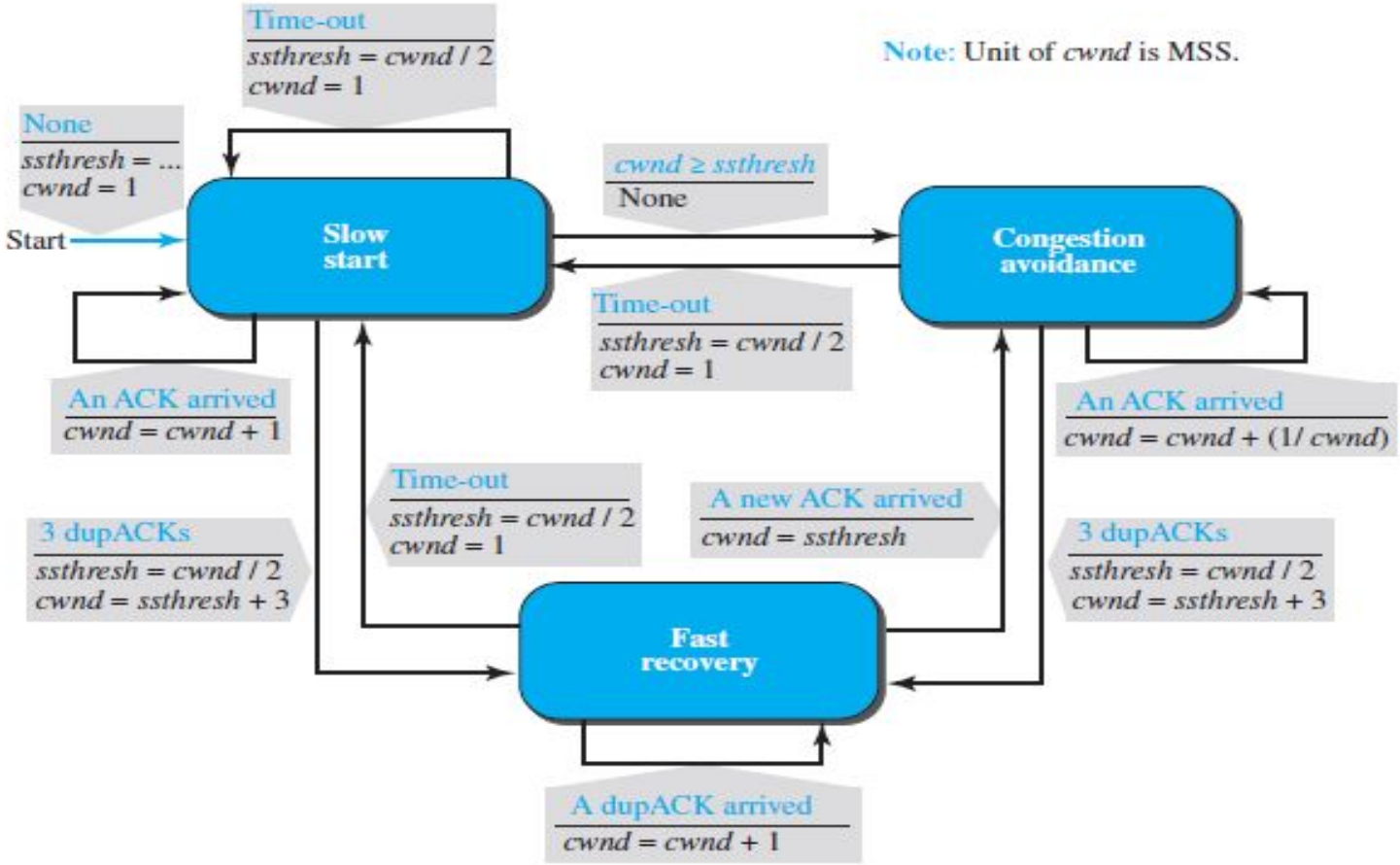


TCP (contd..)

TCP Congestion Control (contd..)

2. Reno TCP:

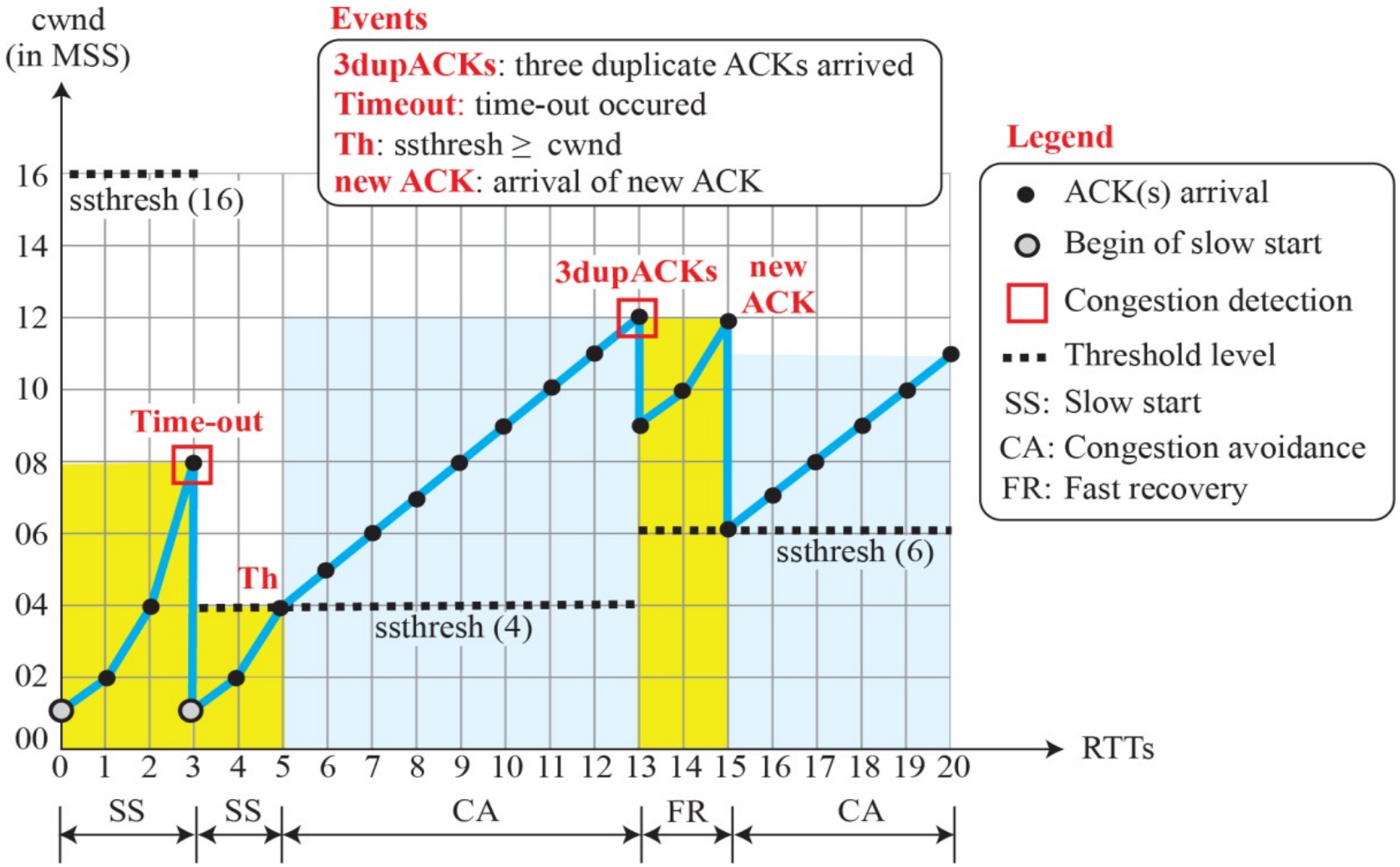
- Below diagram is the FSM for Reno TCP
- Treats time-out and three duplicate ACKs condition in different manner



TCP (contd..)

TCP Congestion Control (contd..)

Example of a Reno TCP



TCP (contd..)

TCP Congestion Control (contd..)

Additive increase, multiplicative decrease (AIMD)

- Sometimes TCP Congestion control is called AIMD
- When ACK arrives, additive increase is followed, when congestion is detected multiplicative decrease is followed