Unit-4

PPT-1 - Transport Layer Protocols

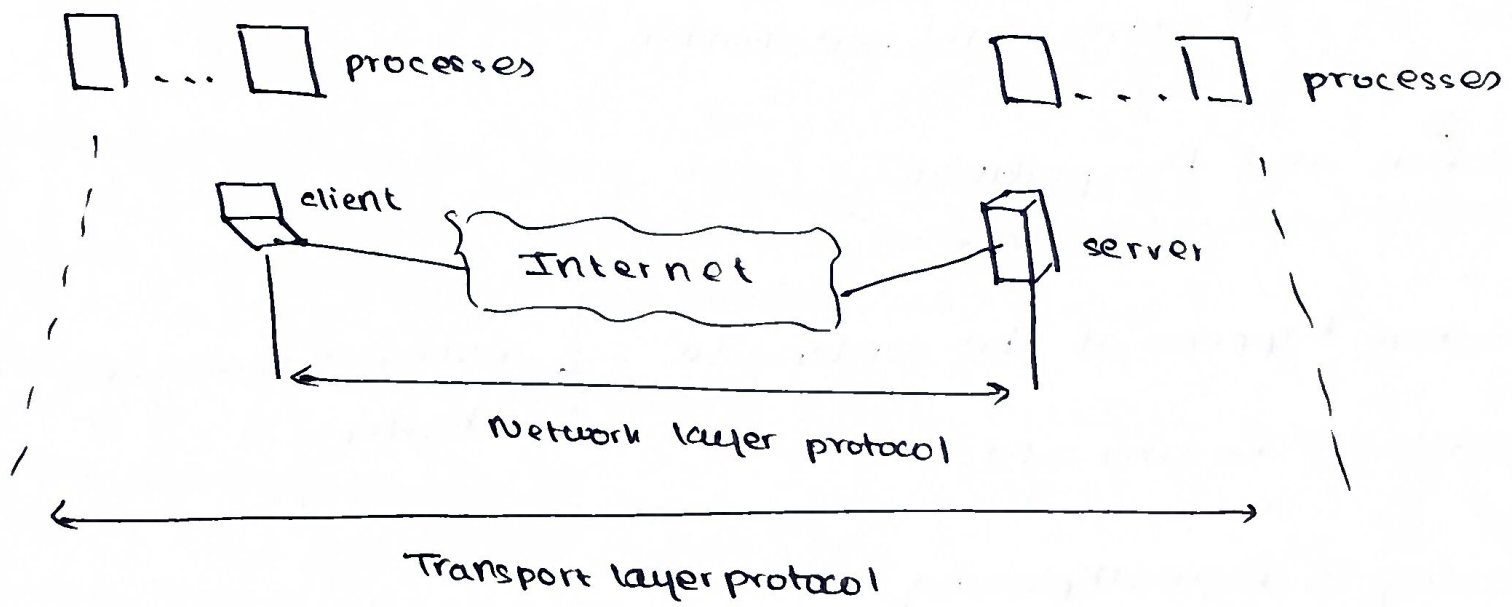
Introduction → TL is between the application and the NL

→ provides process to process communication between two application layers - one at the local host, and the other at the remote host.

* Transport Layer Services

(1) Process to process communication

→ TL protocol is responsible for the delivery of a message to the appropriate process (the NL helps only in host-to-host communication)



(2) Addressing - port numbers

→ The most common way to achieve process to process communication is using the client-server paradigm

→ A process on the localhost (client) needs services from a process on the remote host (server).

→ The local and remote hosts are identified with their IP addresses.

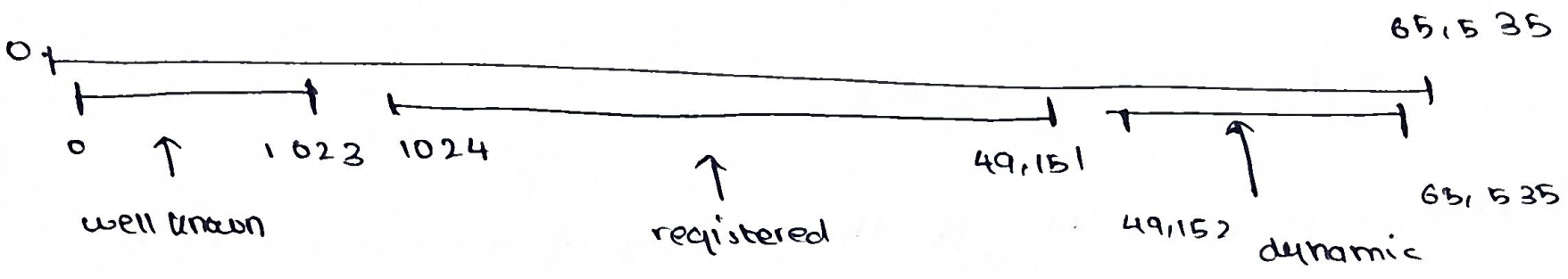
→ To define the processes, use port numbers (16 bits : 0 - 65,535)

Port nos. of clients = ephemeral port no (short-lived)

anything greater than 1023

Port nos. of servers = universal port numbers, called well known port numbers

ICANN ranges of port nos



Socket Address

combination of IP address and port number.

(3) Encapsulation and Decapsulation

→ Encapsulation happens at the sender site. } adding & removing header

→ Decapsulation - receiver site

(4) Multiplexing & Demultiplexing

→ Whenever an entity accepts items from more than one source, it is called multiplexing (many to one)

→ Whenever an entity delivers items to more than one source, it is called demultiplexing (one to many)

→ TL source = multiplexing
TL destination = demultiplexing

(5) Flow Control

→ to maintain a balance between production and consumption rates.

Pushing & pulling

pulling → delivery of items after request

pushing → sender delivers items whenever produced, even without a request.

→ Flow control is needed only when there is pushing, as the consumer may become overwhelmed.

→ In the transport layer: there are 4 entities

(i) sender process ⟶ producer

(ii) sender TL ⟶ consumer + producer: encapsulate + push to (iii)

(iii) receiver TL ⟶ consumer + producer: receive from (ii) + decapsulate

(iv) receiver process → ~~pushing~~ pulling deliver - consumer

→ Flow control can be done with buffers (a set of memory locations that can hold packets at the sender and receiver)

→ When the buffer of the TL is full, it informs the application layer to stop passing chunks of messages.

(6) Error Control

→ TL is responsible for:

(i) detecting and discarding corrupted packets

(ii) keeping track of lost & discarded packets and resending them

(iii) recognizing duplicate packets and discarding them

(iv) buffering out-of-order packets until the missing packet arrive.

Sequence Number - error control requires knowing which packet should be resent (is a duplicate / out of order ⇒ use sequence numbers.

→ If the header of a packet allows m bits, the sequence no ranges from 0 to $2^m - 1$.

Acknowledgement

→ send an Acx for packets that have arrived safely

→ discard corrupted packets & duplicate packets

→ out-of-order ⇒ discard / stash

⑦ Combination of Flow and Error Control

Sequence numbers are modulo 2^m

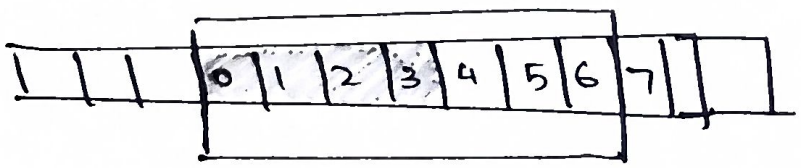
buffer = a set of slices.

→ When a packet is sent, mark the slice

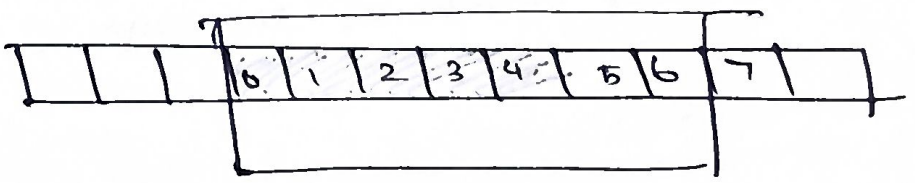
→ If all the slices are marked ⇒ full buffer & no more messages can be accepted.

→ when an ack arrives - slide the window

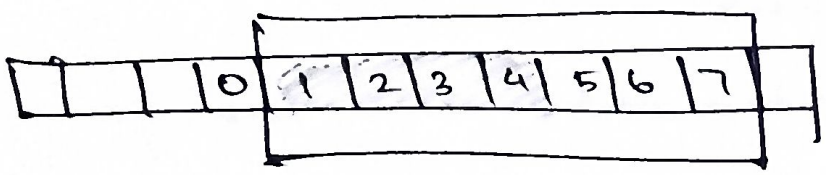
eg.



4 sent packets



5 sent packets



0 has been ack, move window

⑧ Congestion Control

→ congestion occurs if the load on the network, is greater than the capacity of the network.

→ congestion occurs because routers and switches have queues - buffers that hold packets before & after processing

→ Congestion control refers to the mechanism that control the congestion & keep the load below the capacity.

* Connectionless and Connection-Oriented Protocols

independency between packets	dependency between packets

A. Connectionless Service

→ divide a message into chunks and deliver to TL one by one.

→ There is no dependency between packets - packets may arrive out of order (even worse, if all chunks belong to the same msg)

→ receiver does not know when the packet will com / when all have arrived.

→ no flow / error / cong. control

B Connection-Oriented Service

→ Client and server first need to establish a logical connection

→ Data exchange happens after the connection is set

→ After the data exchange, the connection needs to be tore down

→ can implement flow / error / congestion control

Transport Layer Protocols

① Simple Protocol

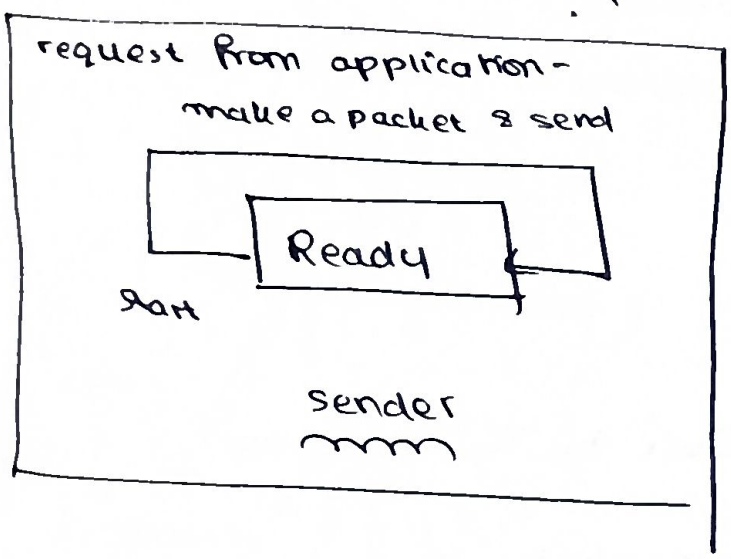
→ connectionless, no flow / error control

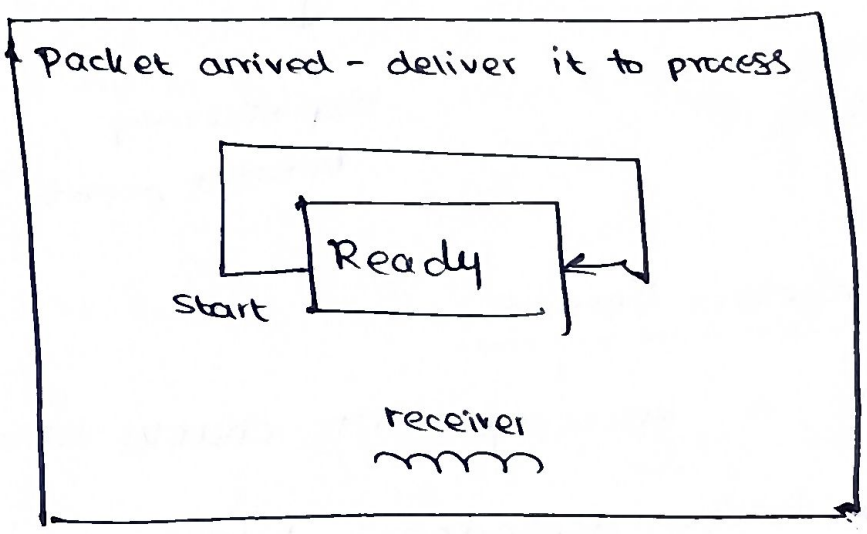
→ assume that the receiver can handle any packet it receives, i'e the receiver can never be overwhelmed with incoming packets.

FSM

sender - should not send until application layer has a msg. to send

receiver - cannot deliver a message to the application layer until a packet arrives.





② Stop-and-Wait Protocol

→ a connection-oriented protocol, has flow & error control

→ Both the sender and receiver use a sliding window of size 1.

→ The sender sends one packet at a time & waits for an acknowledgement before sending the next one.

→ A checksum is used to check if the packet is corrupted.

→ Each time the sender sends a packet, it starts a timer. If an ack arrives before the timer expires - stop & send next pkt

→ If the timer expires - resend (keep a copy of the packet till an ack is received)

Sequence Numbers

→ used to prevent duplicate packets.

→ Use x, $x+1$. There are 3 cases

(i) packet sent - received correctly - ack received ⇒ send next packet = $x+1$

(ii) packet sent, received correctly - ack is lost. Send the same packet (x) again. It is a duplicate - the receiver expects $x+1$

(iii) packet is corrupted & never arrives - resend the packet numbered x.

→ The only sequence needed is 0,1,0,1 ....

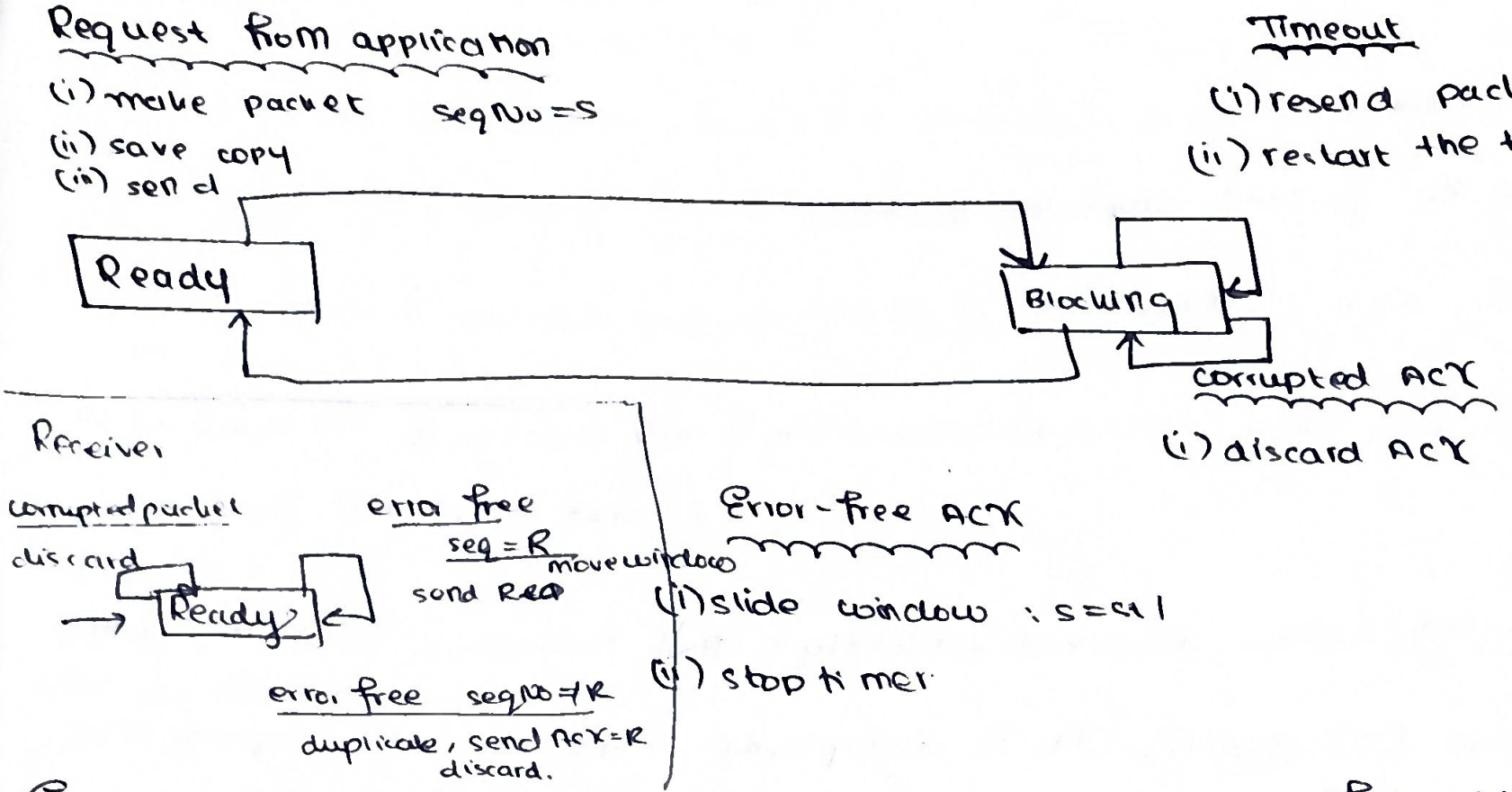
☞ Acknowledgment - announce the sequence no. of the next expected packet.

FSM : Sender → ready - waits for req. from application layer

→ blocking - (i) error-free ACK
(ii) corrupted ACK | error free act with ackNo ≠ (s+1) mod 2
(iii) timeout occurs.

Receiver → ready - (i) error free - R = seq No
(ii) error free - R ≠ seq No
(iii) corrupted packet

Sender

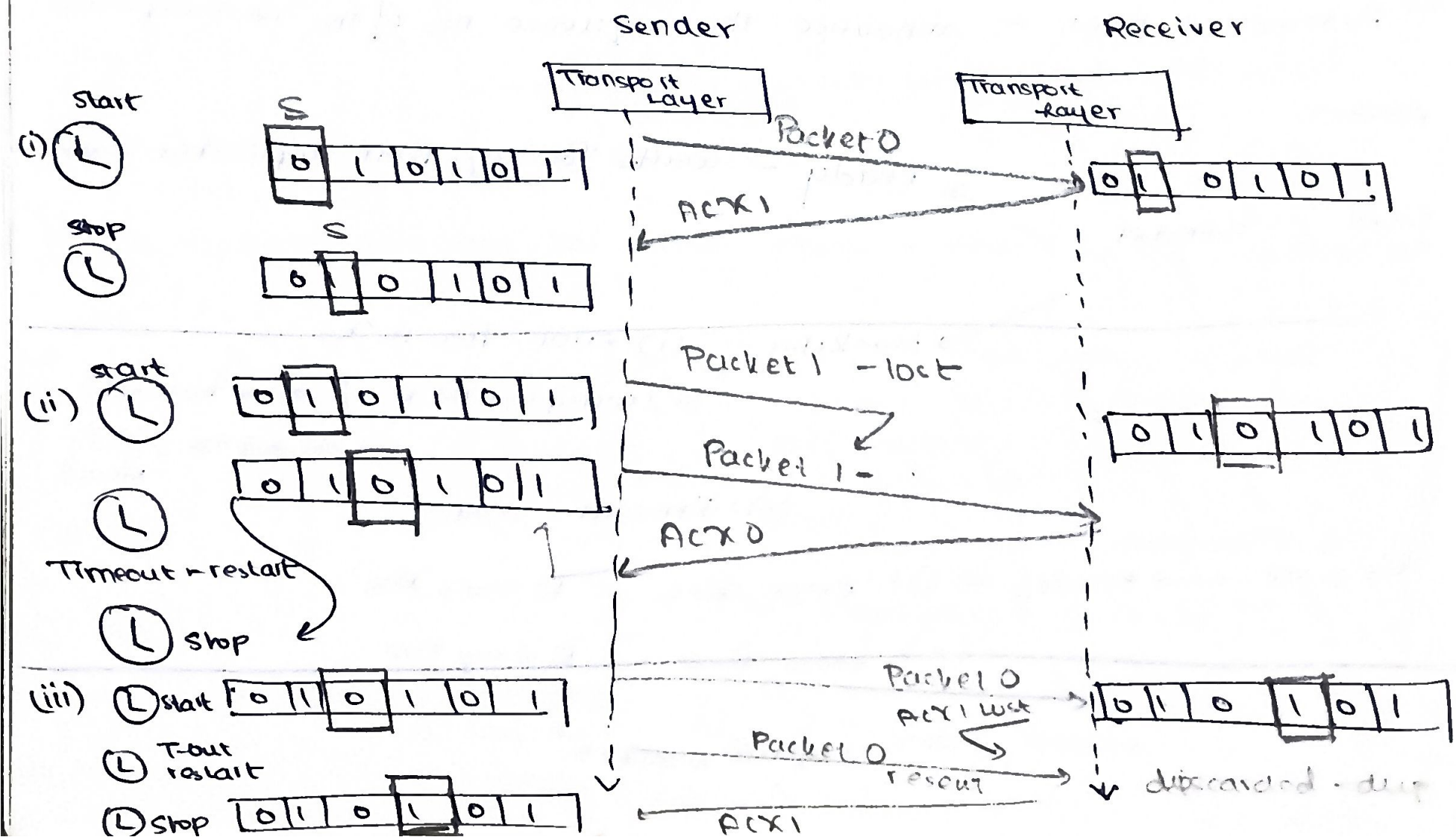


Example Using the stop-and-wait protocol, depict the following actions.

(i) Packet 0 is sent and acknowledged.

(ii) Packet 1 is lost and resent after the time-out, and acknowledged

(iii) Packet 0 is sent & ack, but the ack is lost.



See eq. 23.5 pg 712

(2) Go-Back-N Protocol (GBN)

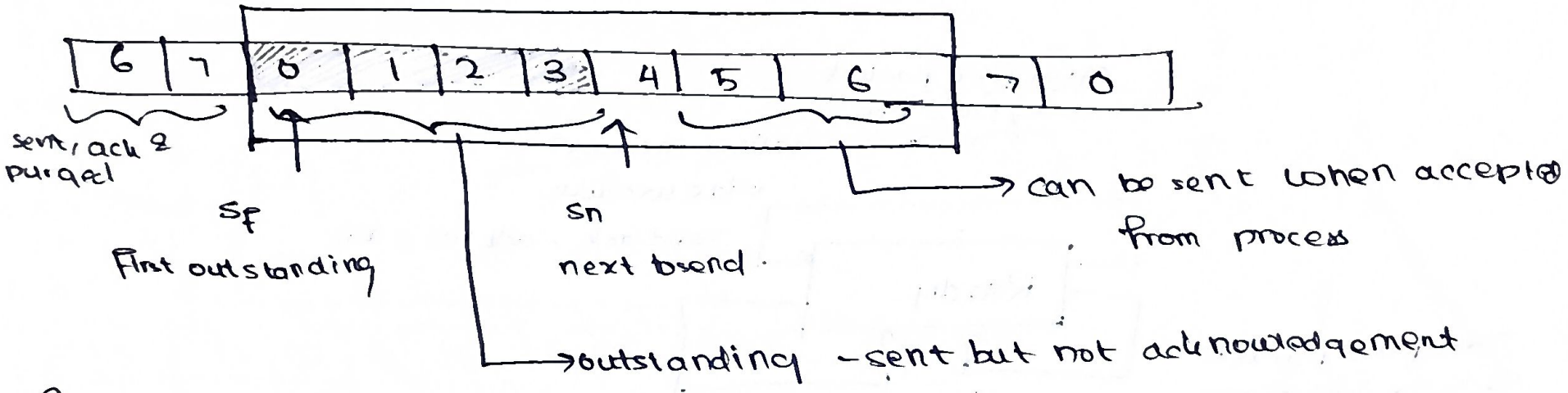
→ To improve efficiency, multiple packets must be in transition while the sender is waiting for an acknowledgement.

→ i.e can send several packets before receiving an ack, but the receiver can buffer only 1 packet

Sequence Numbers: modulo 2^m

Acknowledgment No: ack is cumulative, defines the seq. no. of the next packet expected.

Send window - of size 2^{m-1}



Receive window - always of size 1.

Resending packet - when the timer expires - resend all the outstanding packets

FSM Sender

Ready - (i) req. from application layer
(ii) error-free ACK Sf = ack no
(iii) timeout
(iv) corrupted

Blocking - (ii), (iii) & (iv)

Receiver: ready - (i) error free - seq = Rn - slide window
(ii) error free - seq ≠ Rn (outside window)
(iii) corrupted ack discard.

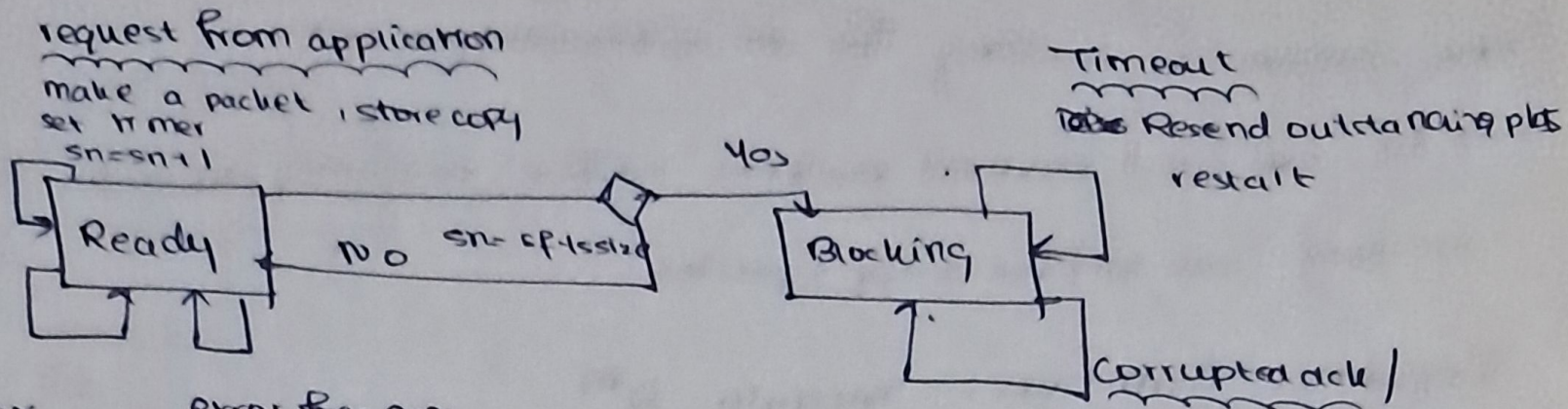
FSN is same

for sender

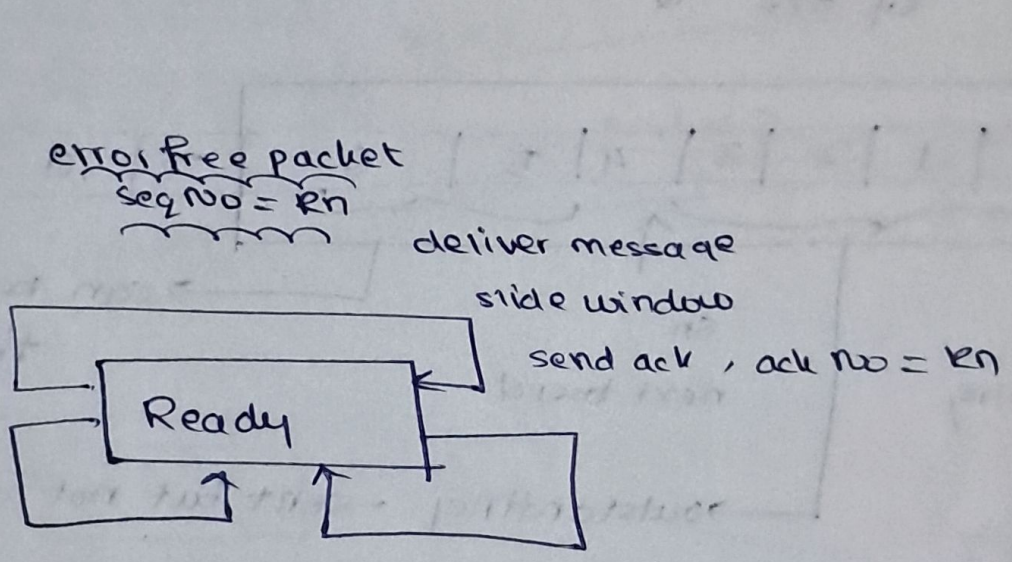
↓

include timer

timeout & window size check



receiver is the same.



Example: Depict the following scenario, with the Go-Back-N protocol.

(i) Packets 0, 1, 2, 3 are sent

(ii) Packet 1 is lost.

move Sn
Send
if ack - slide, move Sf

move Sn no matter what

move Sf only if successful

Sender

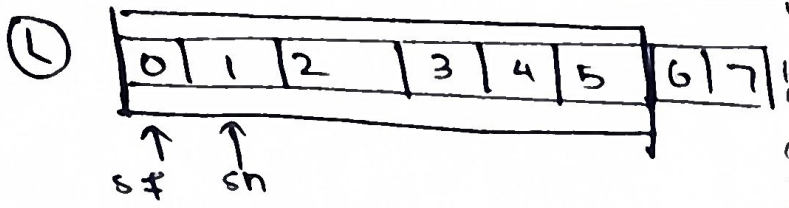
Receiver

(1)

Transport Layer

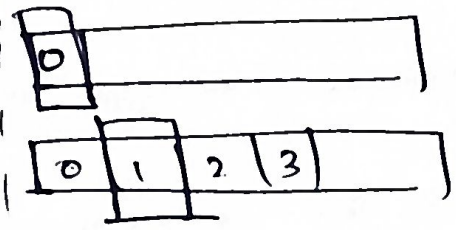
Transport Layer

start



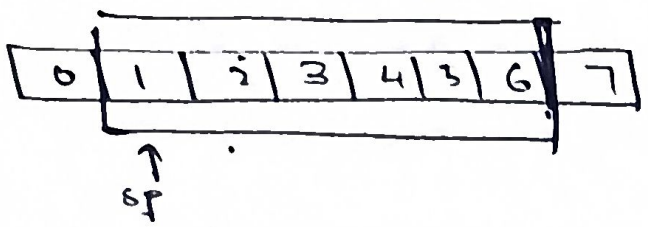
packet 0

ACK 1



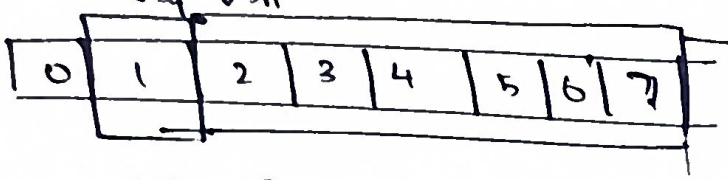
Stop

move window



move sn.

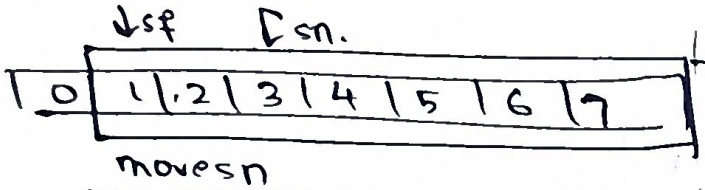
start



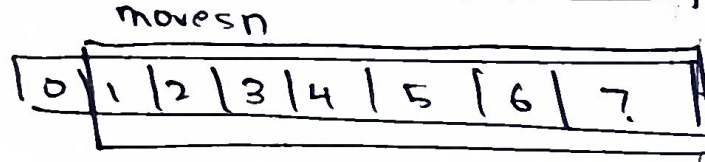
packet 1

Lost

move sn



packet 2



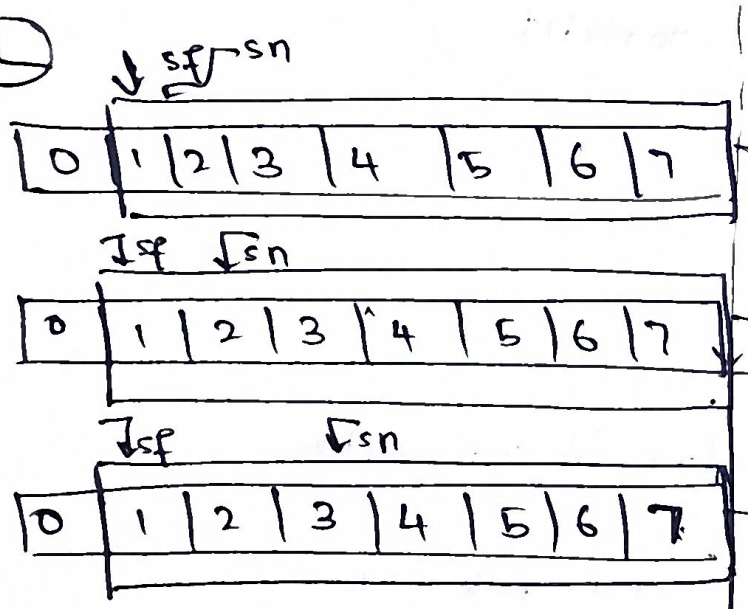
ACK 1

packet 3

ACK 1

discarded

Timeout



pkt 1

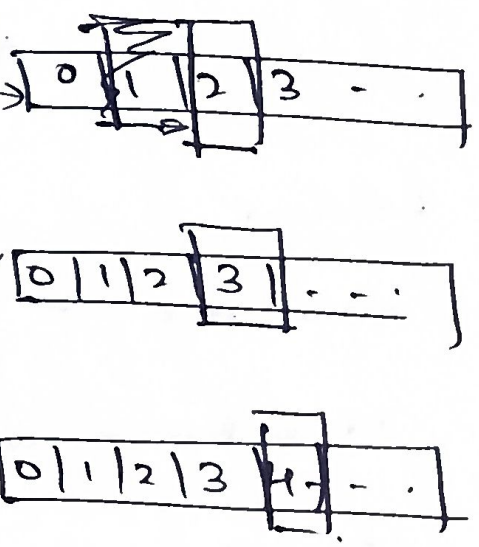
pkt 2

pkt 3

ack

ack

ack.



restart move window

restart move window

restart move window.

(3) Selective Repeat Protocol

size = 2^{m-1}

sender & receive window = 2^{m-1}

→ The SR Protocol allows as many packets as the size of the receive window to arrive out of order and be kept there until there is a set of consecutive packets to be delivered to the application layer.

FSM

Sender

Ready — (i) request from application layer

(ii) error free ACK (ack No = sf)

(iii) corrupted ACK - discard

(iv) time-out - resend

Blocking - error free - slide window to right

corrupt - discard

timeout - resend

Receiver

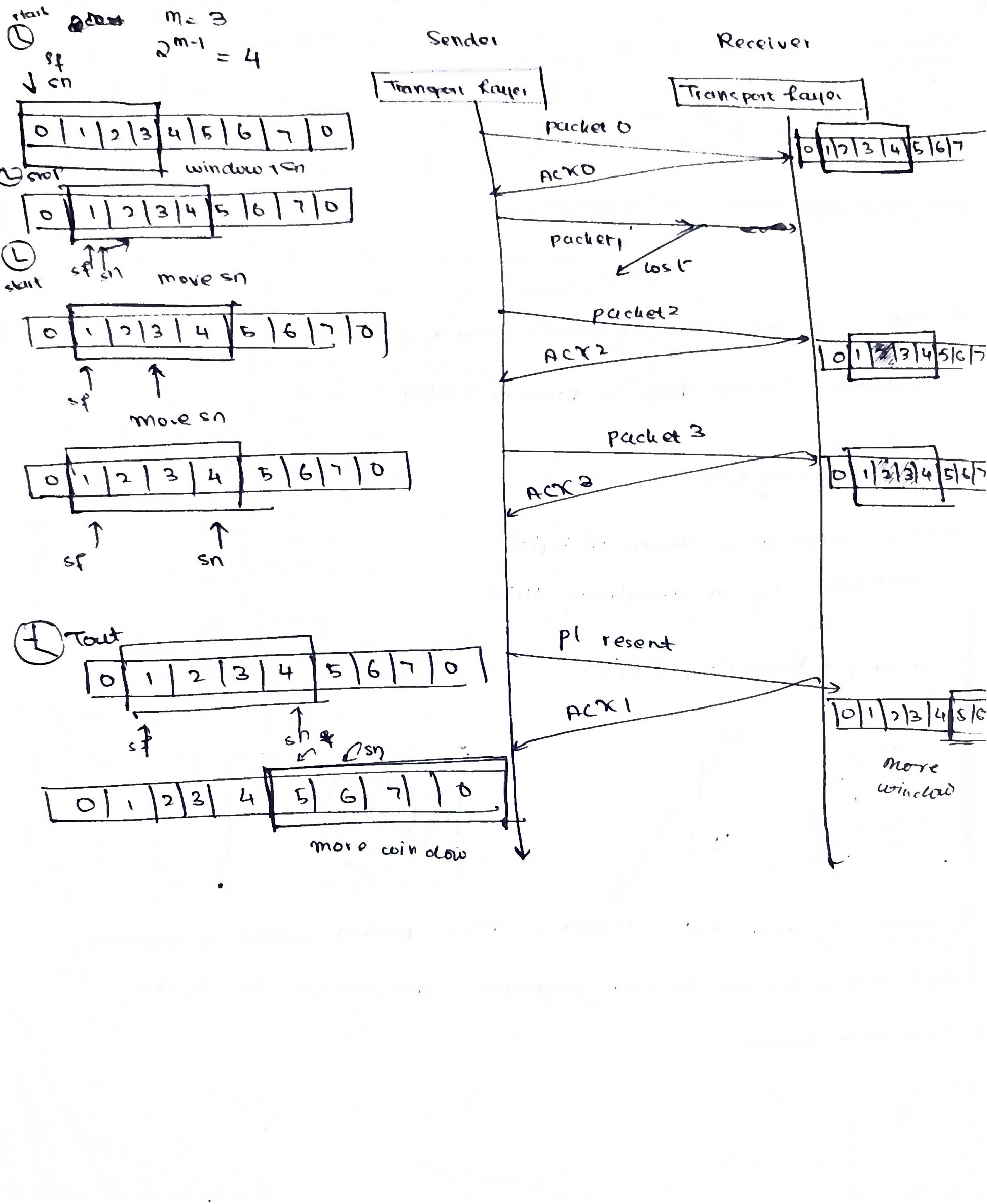
error free - ack = seq No , slide window if consecutive

error free - outside window - discard; send Ack = Rn

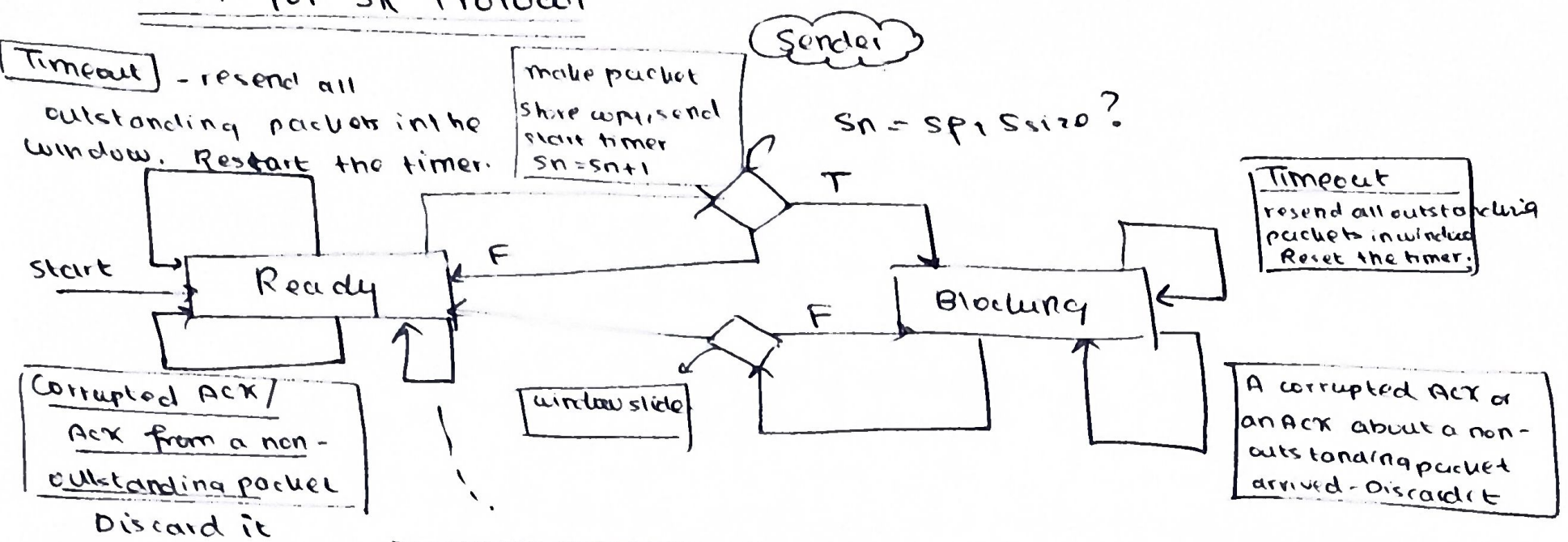
corrupt - discard

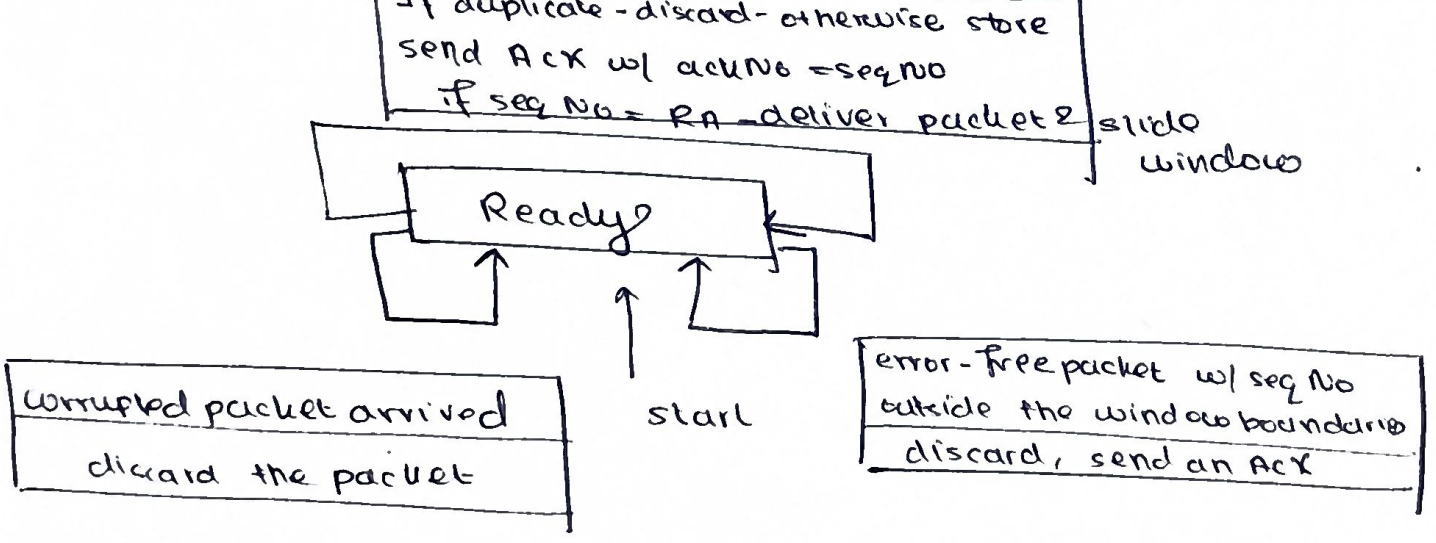
same as GBN - change writing alone.

Example Consider the same example as on pg.10



FSM for SR Protocol





TCP and UDP

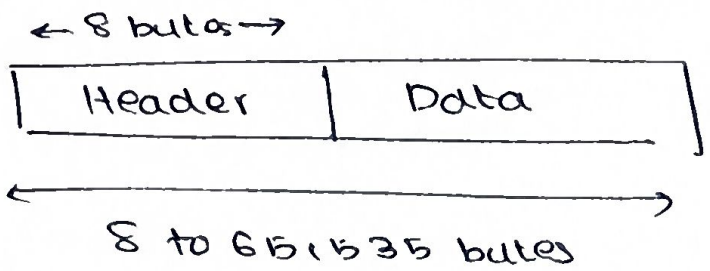
(A) **UDP** - stands for user datagram protocol

- a connectionless, unreliable transport protocol

- very simple, minimum overhead.

UDP Packets - UDP packets - called user datagram.

Format



Header Format

0	16	31
source port number	destination port no	
Total length	checksum	

Applications of UDP - TFTP

- multicasting
- SNMP
- routing table updates - RIP
- DNS

(B) **TCP** - a connection-oriented reliable protocol

- explicitly defines connections for connection establishment, data transfer & connection teardown phases

* **TCP Features**

1. process to process communication - communication w/ port nos.

2. Stream delivery service – 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

- connection in the form of a tube
- there are sending and receiving buffers, as transmission and reception do not happen at the same rate.
- The sending buffer has: sent data
  written data but not sent
  next byte to write
- The receiving buffer has - next byte to read
  received but not read
  next byte to receive
- Though buffering handles the disparity between the speed of production and consumption, the network layer sends data as packets and not as a stream.
- At the TL, TCP groupes a no. of bytes together into a packet called a segment and delivers it to the NL, with a header added.

3. Full duplex communication

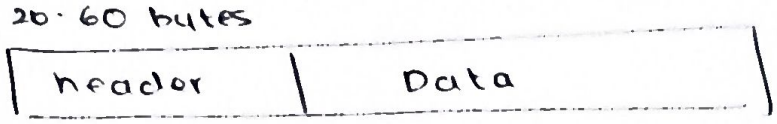
4. Multiplexing and demultiplexing

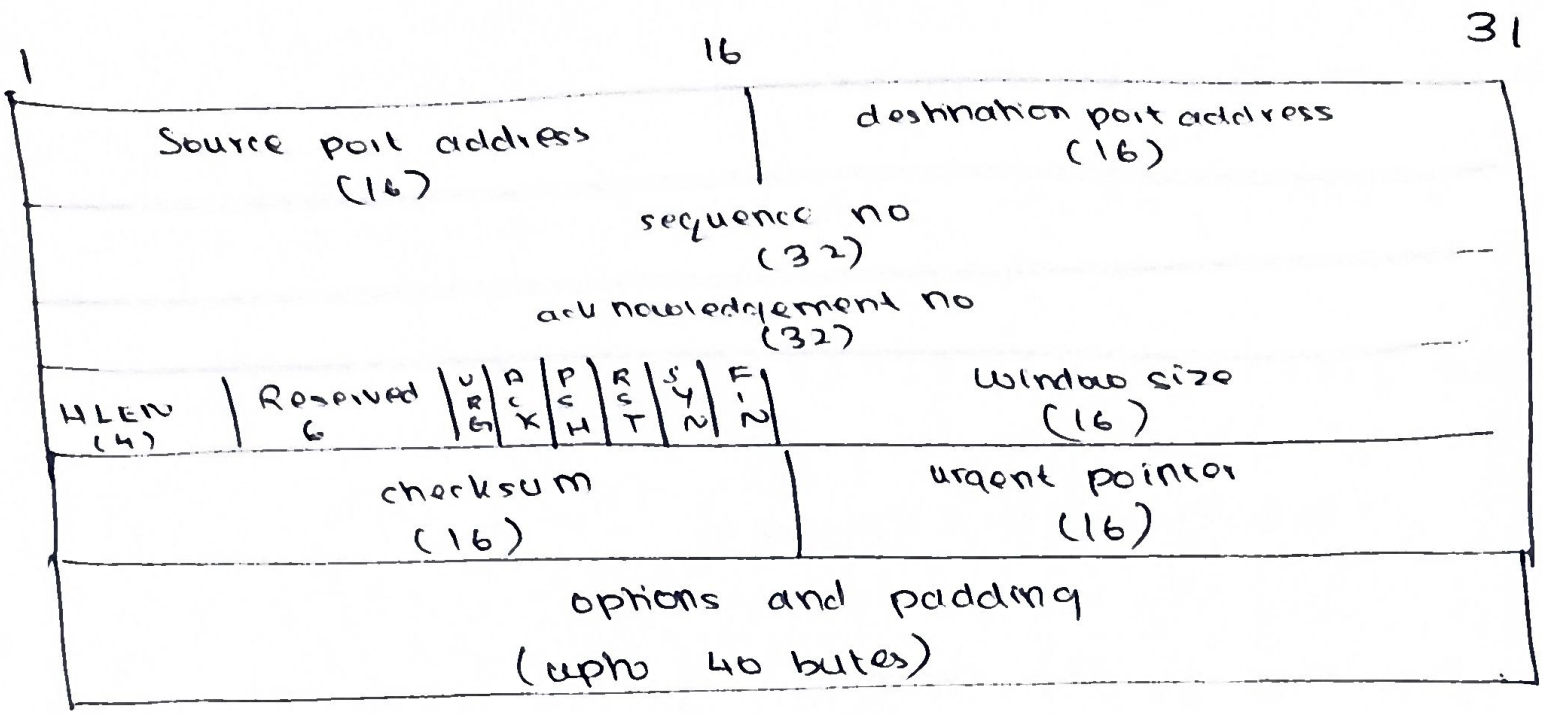
5. Connection-Oriented

6. Reliable Service

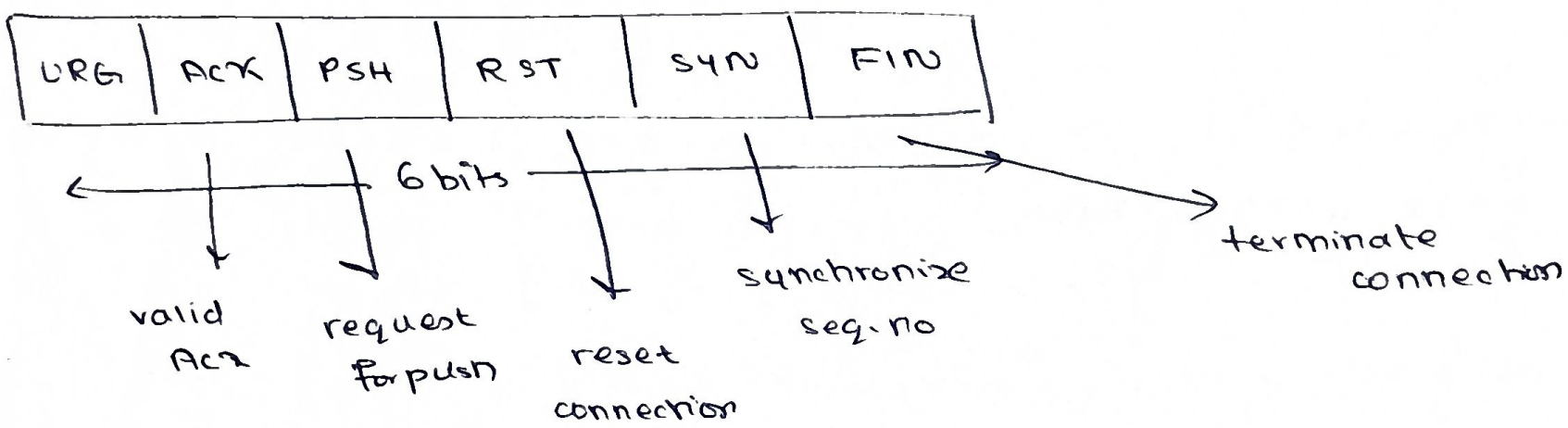
* TCP Segment Format

20-60 bytes





control field



Example – The following is a dump of a TCP header in hexadecimal format.

0532 0017 00000001 00000000 5002 07FF 00000000

(i) What is the source port number?

Each no. corresponds to 4 bits

First 16 bits = 0532 = 1330 (convert hex to decimal)

(ii) What is the destination port number?

second 16 bits = 0017 = 23

(iii) What is the sequence number?

32 bits = 0000 0001 = 1

(iv) What is the acknowledgement number?

next 32 bits = 0000 0000 = 0

(v) What is the header length?

4bits = 5 length = 5 x 4 = 20 bytes

(vi) What is the type of the segment?

-- 0000 0010 - syn packet

(vii) window size?

07FF = 2047 bytes

* TCP Connection - 3 way handshake mechanism

→ In TCP, connection oriented transmission has 3 phases:

(i) connection establishment

(ii) data transfer

(iii) connection termination

- called a 3-way handshake

A. **Connection Establishment**

(i) server tells the TCP that it is ready to accept a connection

- called a passive open

(ii) client sends SYN segment - only the SYN flag is set.

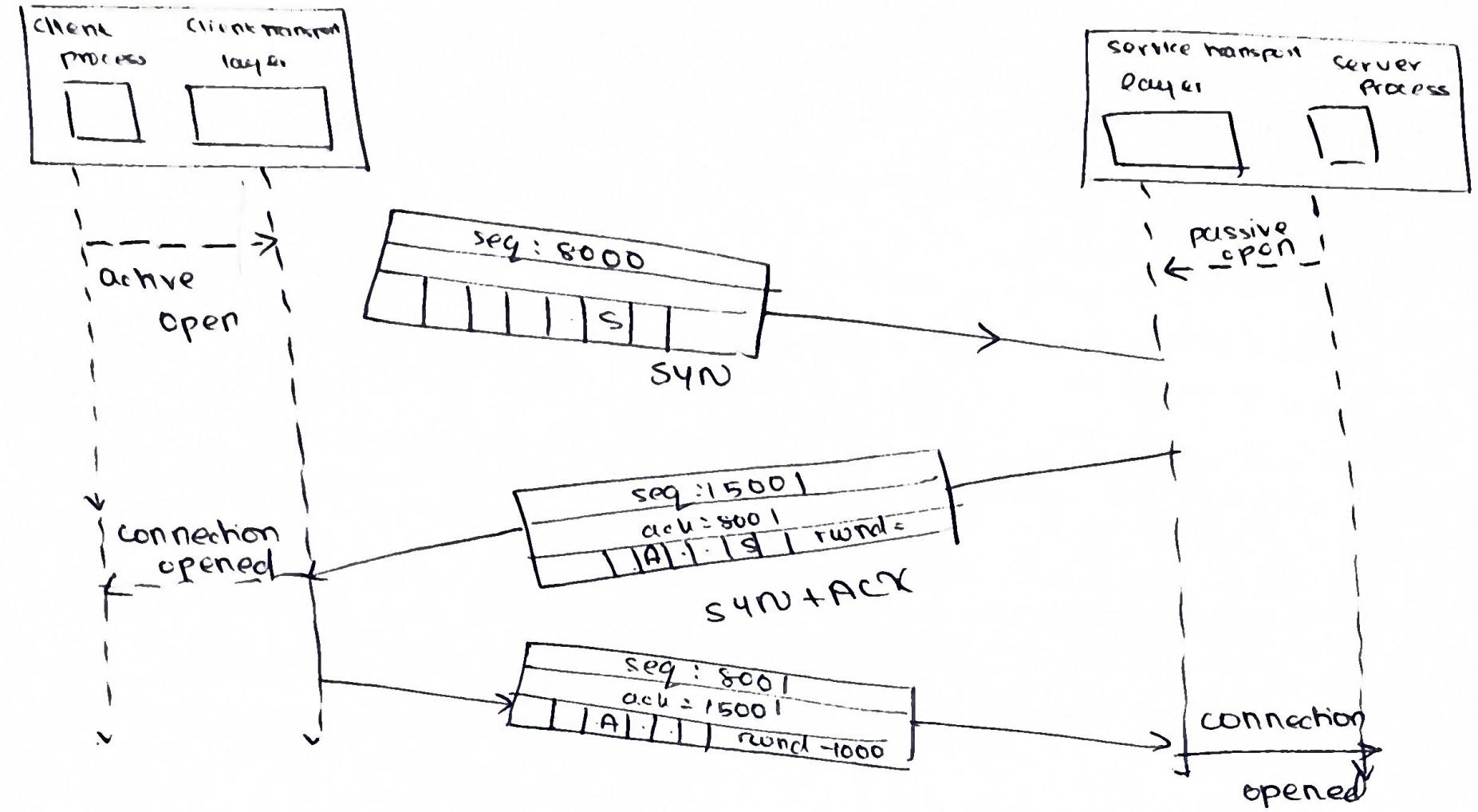
↳ called the ISN - initial sequence number

(iii) server sends a SYN + ACK segment - with the SYN and ACK flag bits set. It has a dual purpose

- SYN segment = for communication in the other direction
- ACK segment - as acknowledgement to the SYN segment from the client - also displays the next sequence no. it expects from the client.

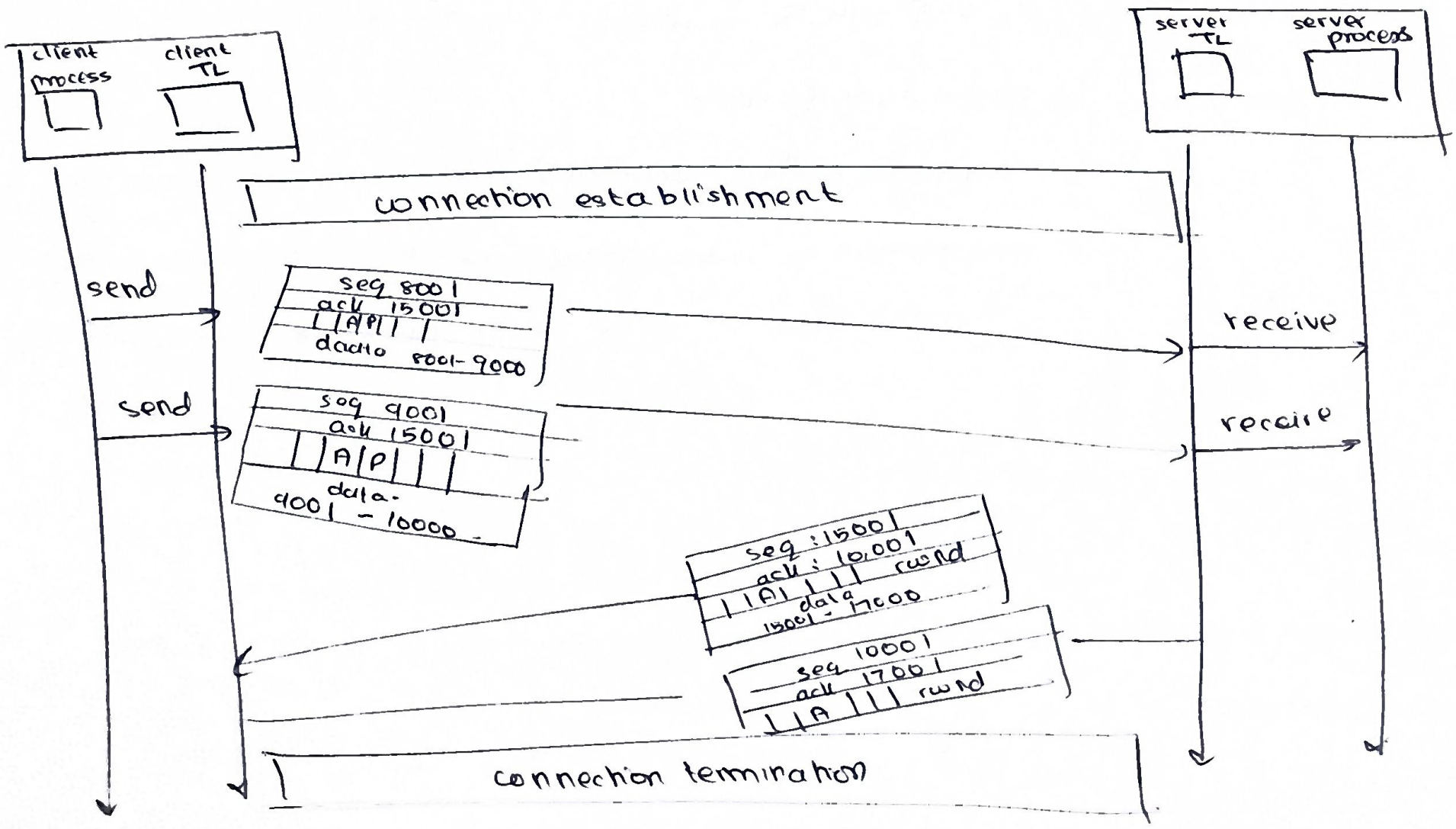
- also set the rwnd

(iv) client sends an ACK - acknowledges the receipt of the second segment.



B. **Data Transfer**

- client and server can send data and acknowledgements in both direction
- The acknowledgment is piggybacked w/ the data.

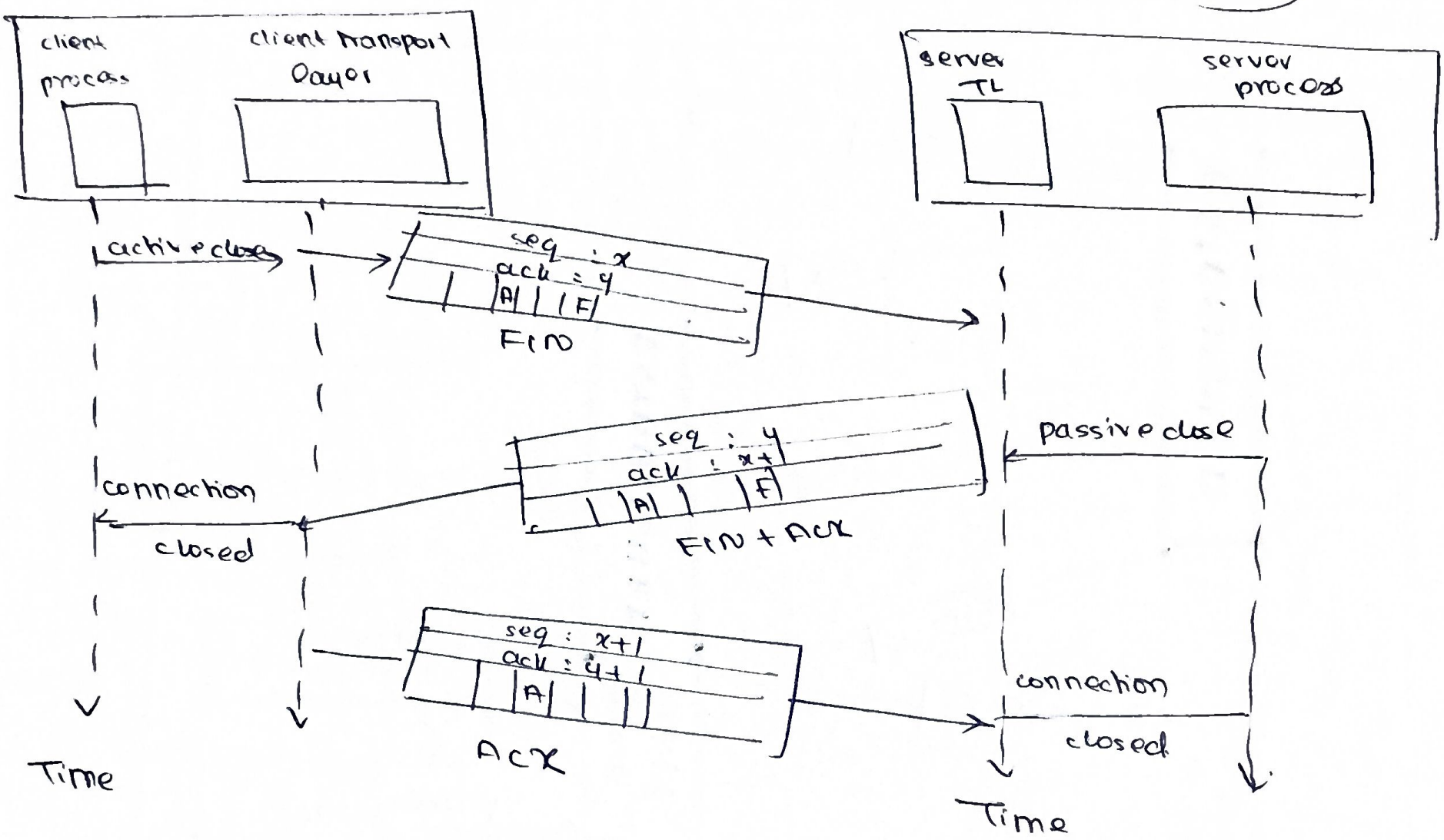


c. Connection Termination

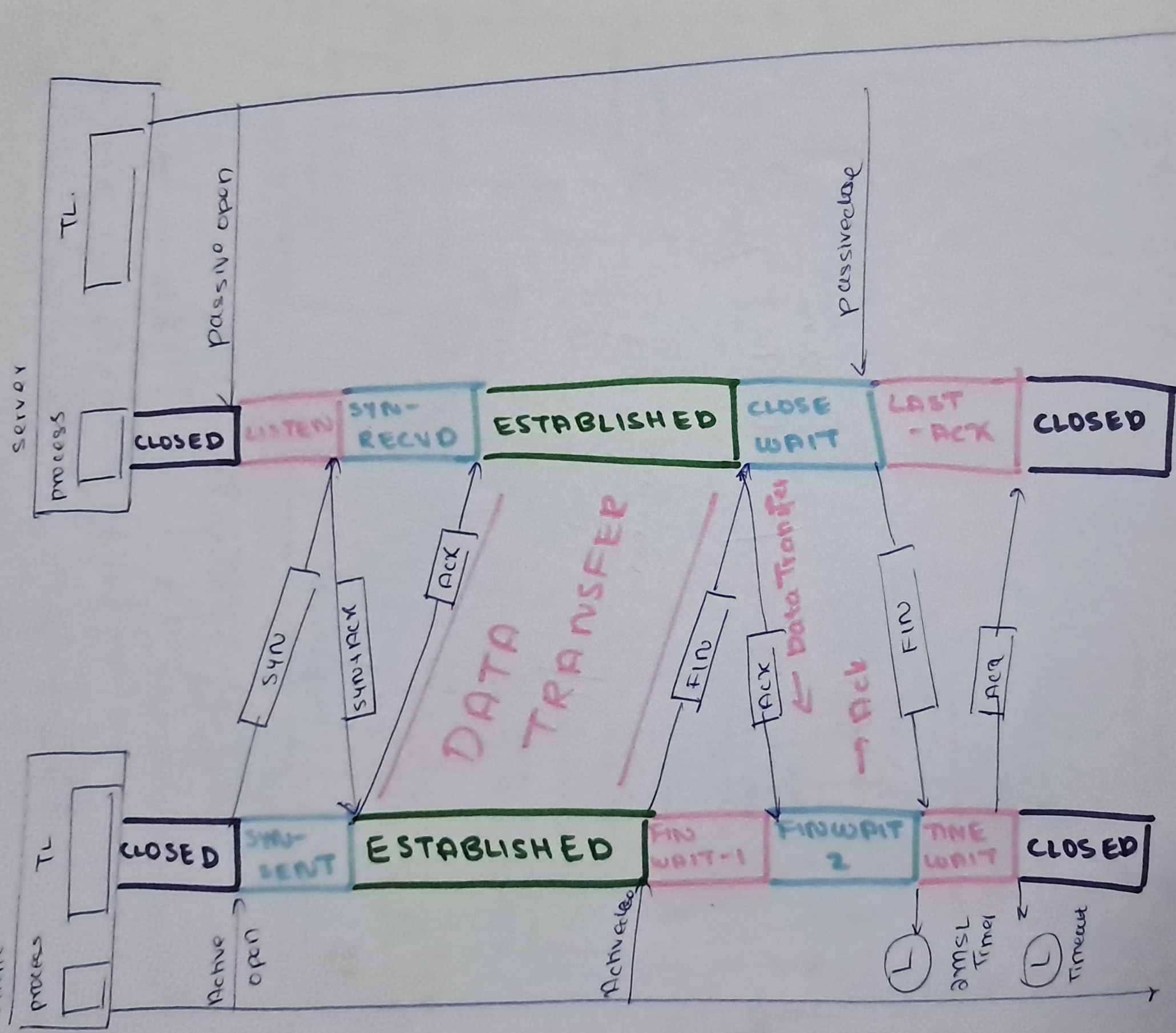
- client sends a FIN segment
- server sends FIN + ACK
- client sends an ACK

Half-close

one-end can stop sending data, while still receiving data



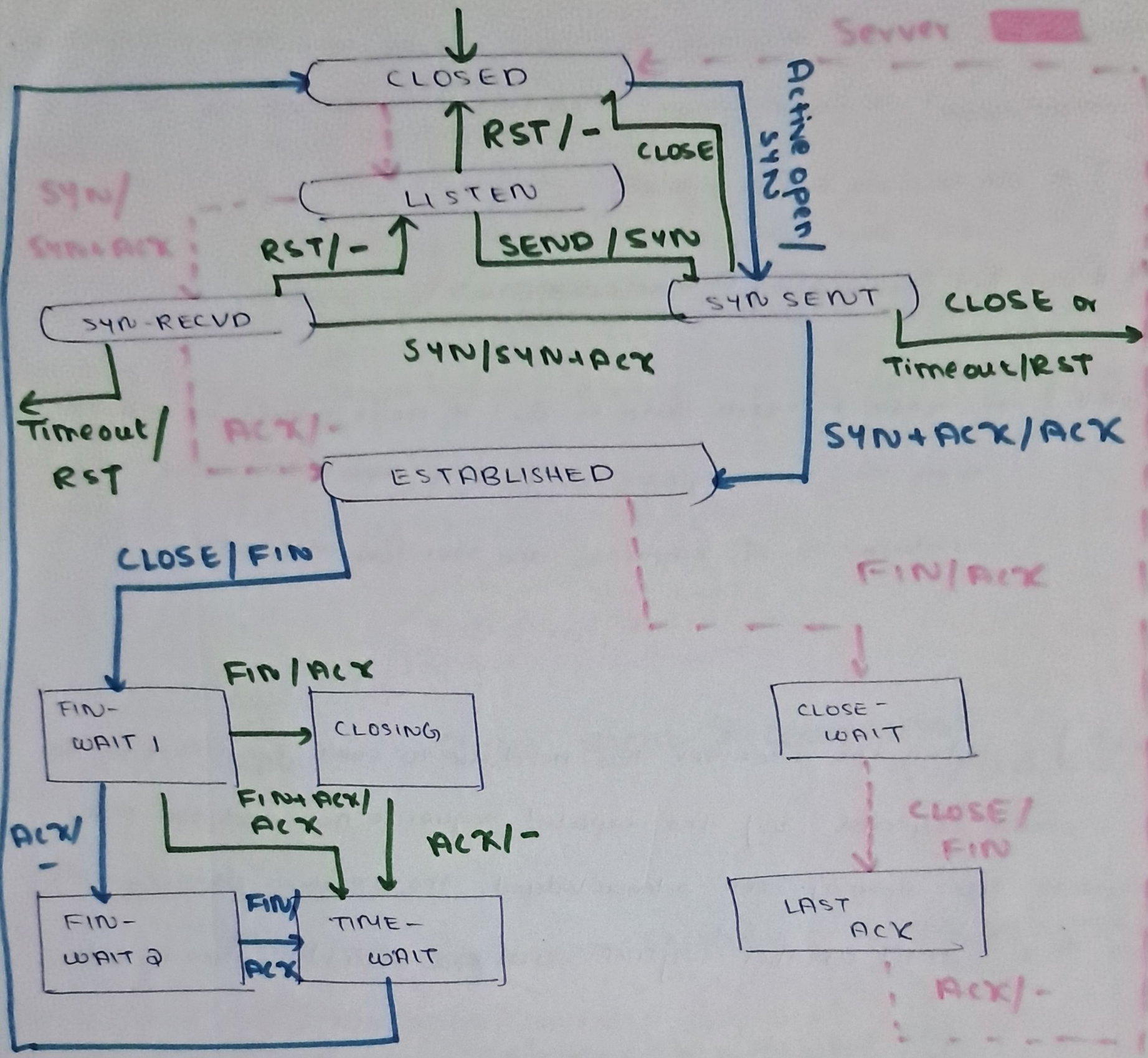
* State Transition Diagram



Client or Server

Client

Server



Time-out

2MSL

* Silly window Syndrome - problem w/ the sliding window operation
- when the application program creates data slowly or the receiving application program consumes data slowly - or both

Solutions

Sender - Nagle's Algorithm - delay sending of small packets - accumulate messages to fill a max-size segment

Receiver - Clark's solution - send an acknowledgement as soon as the data arrives - but to announce a window size of zero until either there is enough space to accommodate a segment of maximum size or until at least half of the receiver buffer is empty.

* Rules for Generating Acknowledgements

Rule1 - when A sends data to B, it must include an ack that gives the next sequence no. that it expects to receive.

- reduces no. of segments, and therefore the traffic

Piggybacking

Rule2 - When the receiver has no data to send & it receives an in-order segment w/ the expected sequence no - and the prev segment has already been acknowledged, the receiver delays sending an ACK until another segment arrives - or a time has passed.

delayed acknowledgement

Rule 3 - When a segment arrives w/ a sequence no. that is expected by the receiver, and a previous segment has not been acknowledged, send an ACK immediately

- there should be no more than 2 unacknowledged segments at given time
- prevents retransmission that may cause congestion

no more than 2 unacknowledged segs. at a time

Rule 4 - Ack for out of order

send expected seq. no

Rule 5 Ack for missing segs

reports that the missing seq has been received by sending the next expected ACK

Rule 6 - If a duplicate arrives - discard seq, but send ACK w/ the expected sequence.

duplicate segments

* TCP Congestion Control

TCP Taho - slow start & congestion avoidance

set cwnd = 1

if ACK arrives - cwnd + 1 = exponential

at timeout ssthresh = cwnd / 2

cwnd = 1

at any point if cwnd ≥ ssthresh - go into

congestion avoidance mode

cwnd = cwnd + 1/cwnd

If 3 dup acks arrive - ssthresh = cwnd/2 (inc by 1)

cwnd = 1

TCP Reno -

Set cwnd = 1

- at timeout - ssthresh = cwnd/2
  cwnd = 1
- at duplicate ACKs - ssthresh = cwnd/2
  cwnd = ssthresh + 3
- at new Ack - cwnd = ssthresh - regular increase
  regular increase
  ssthresh = cwnd/2

See Flow diagrams

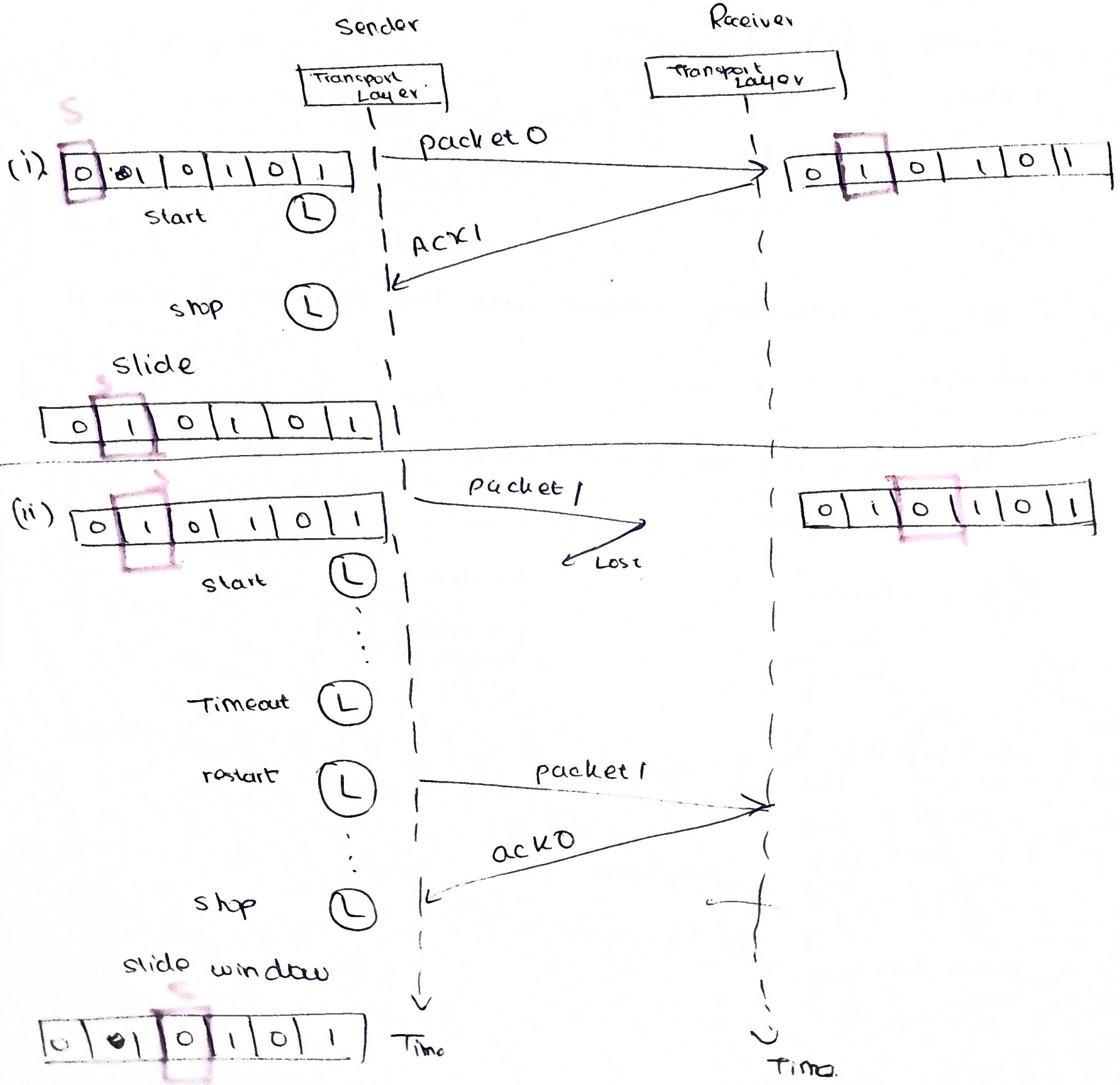
Unit 4 Numericals

(1) Using the stop and wait protocol, depict the following actions

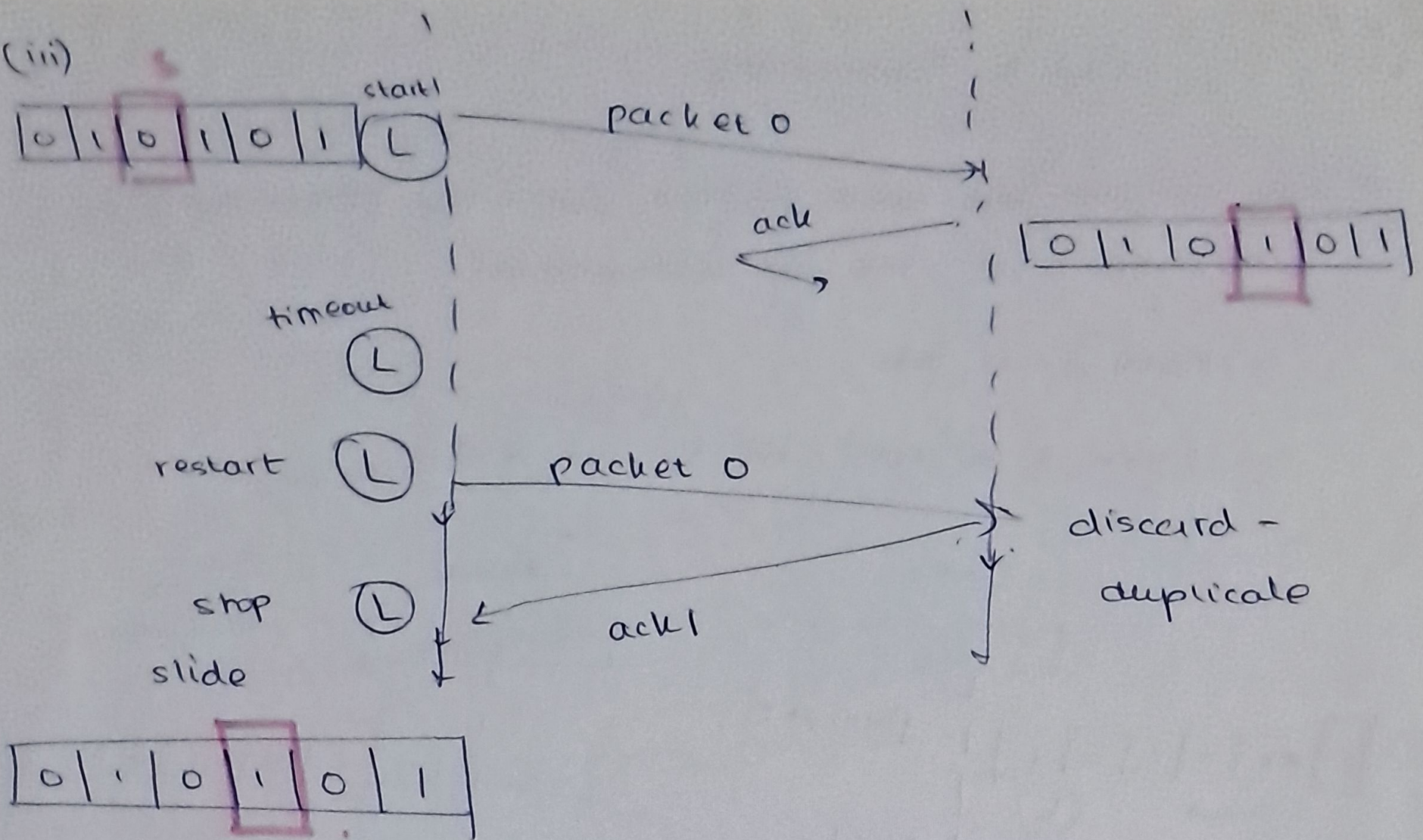
(i) Packet 0 is sent and acknowledged

(ii) Packet 1 is lost

(iii) Packet 0 is sent, but the ack is lost



(iii)



② Depict the following scenario with the Go-Back N protocol,

assume m = 3

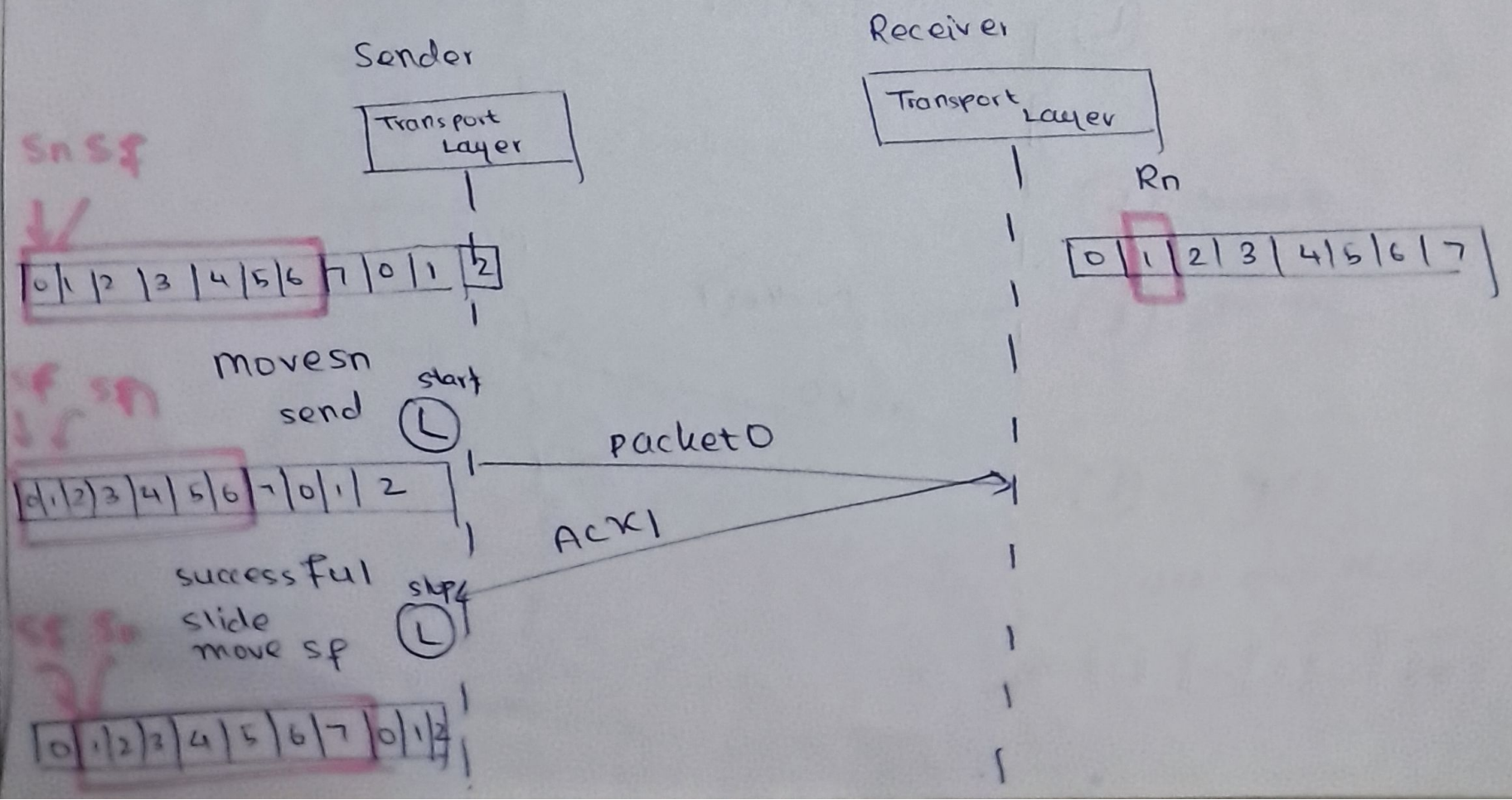
(i) Packets 0, 1, 2, 3 are sent

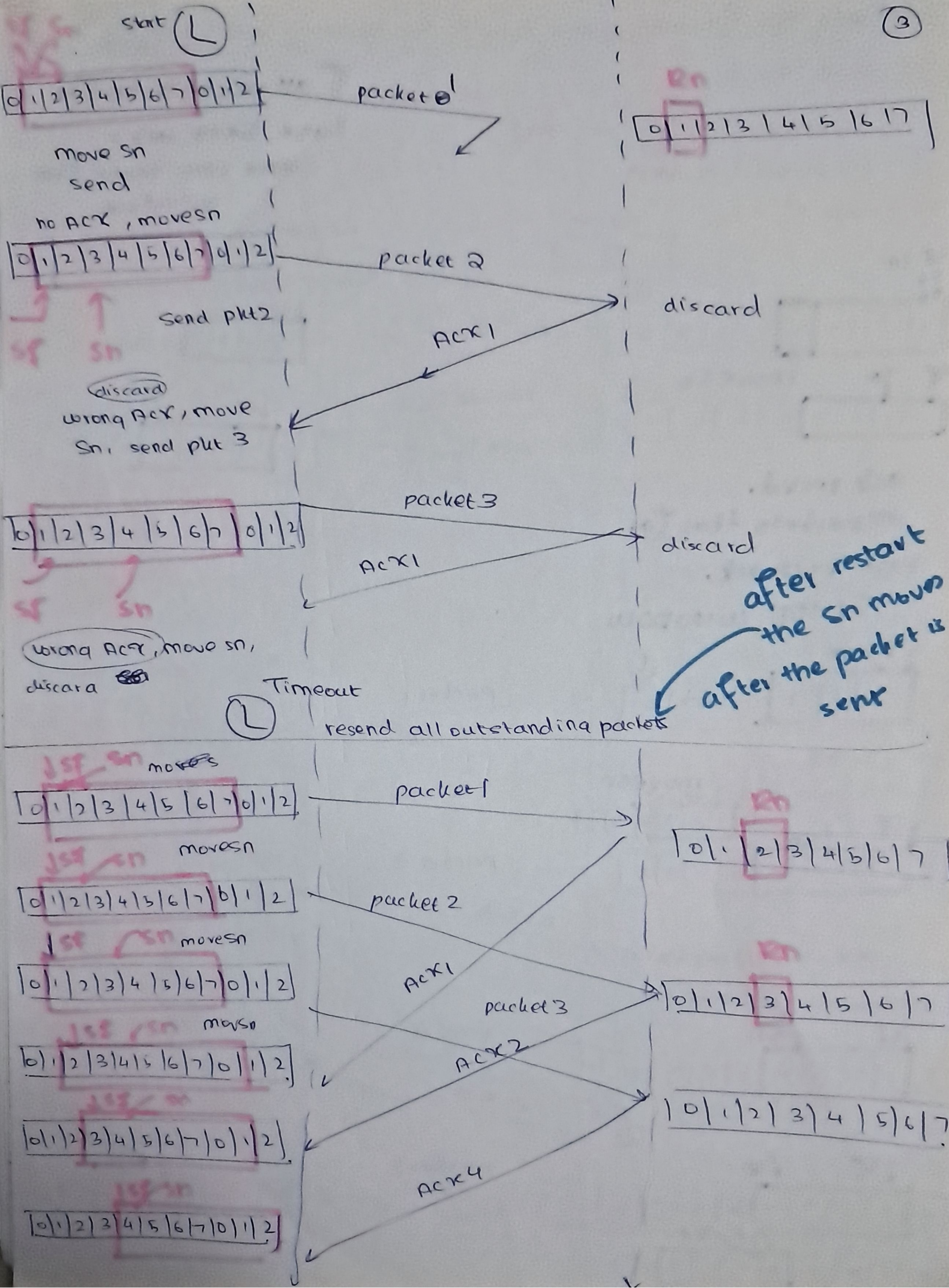
(ii) Packet 1 is lost

m = 3

window = $2^m - 1$

= 7




start
packet 0
Rn
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
move Sn
send
no ACK, move sn
packet 2
discard
send pkt2
ACK1
Sf
Sn
discard
wrong ACK, move
Sn, send pkt 3
packet3
discard
ACK1
Sf
Sn
wrong ACK, move sn,
discard
Timeout
after restart
the Sn moves
after the packet is
sent
resend all outstanding packets
Sf Sn moves
packet1
Rn
movesn
packet 2
Sf Sn movesn
ACK1
packet3
moves
ACK2
ACK4

(3) Depict the following scenario with the selective repeat protocol with $m = 3$

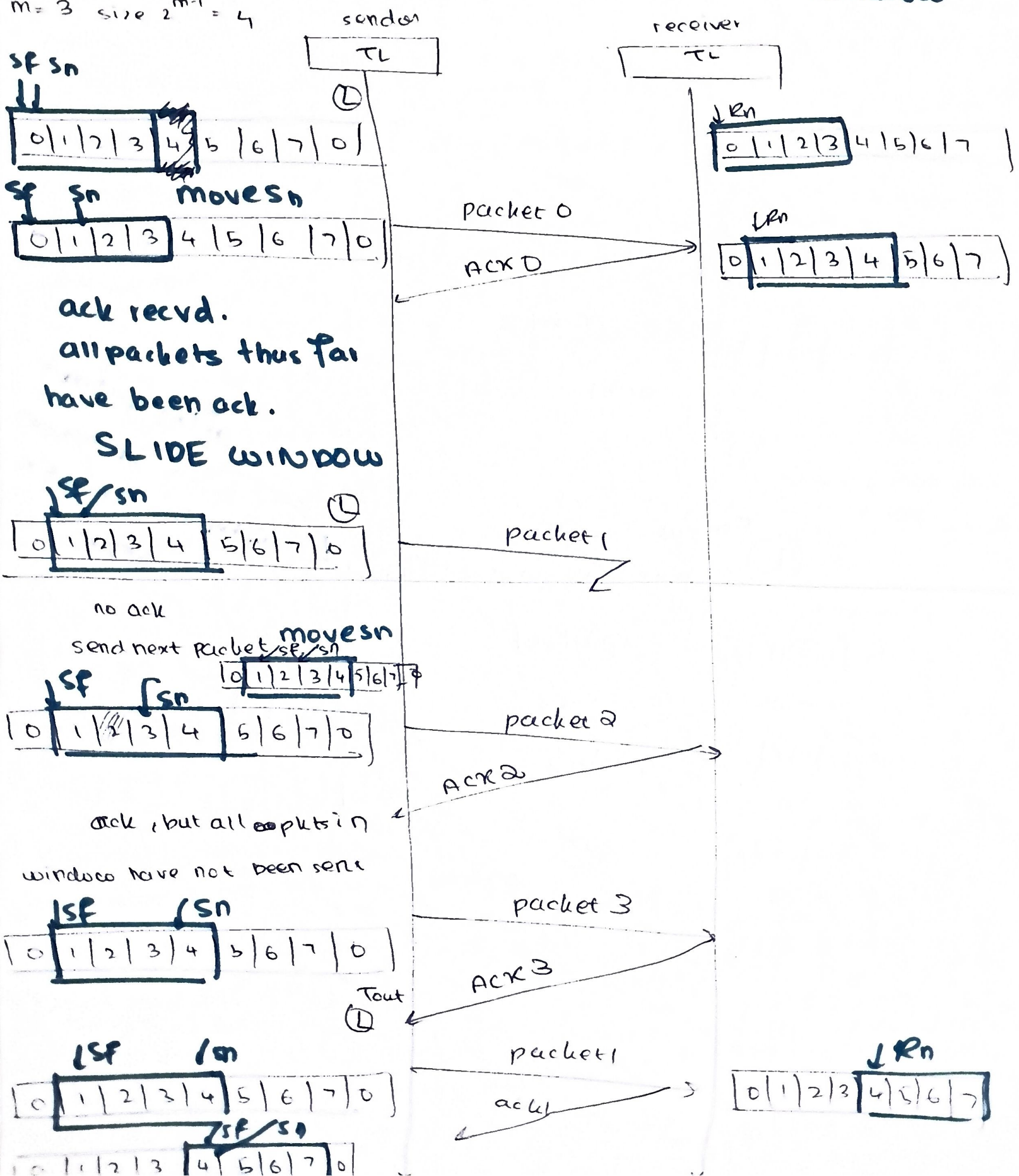
(i) packets 0, 1, 2, 3 are sent

(ii) packet 1 is lost

(3)

If all packets so far have been ack - you can move the window

$m = 3$ size $2^{m-1} = 4$



(4) Apply the Go back N and selective repeat flow control (5)

protocols for the following scenario

Packets 0, 1, 2, 3, 4 are sent

packet 3 is lost

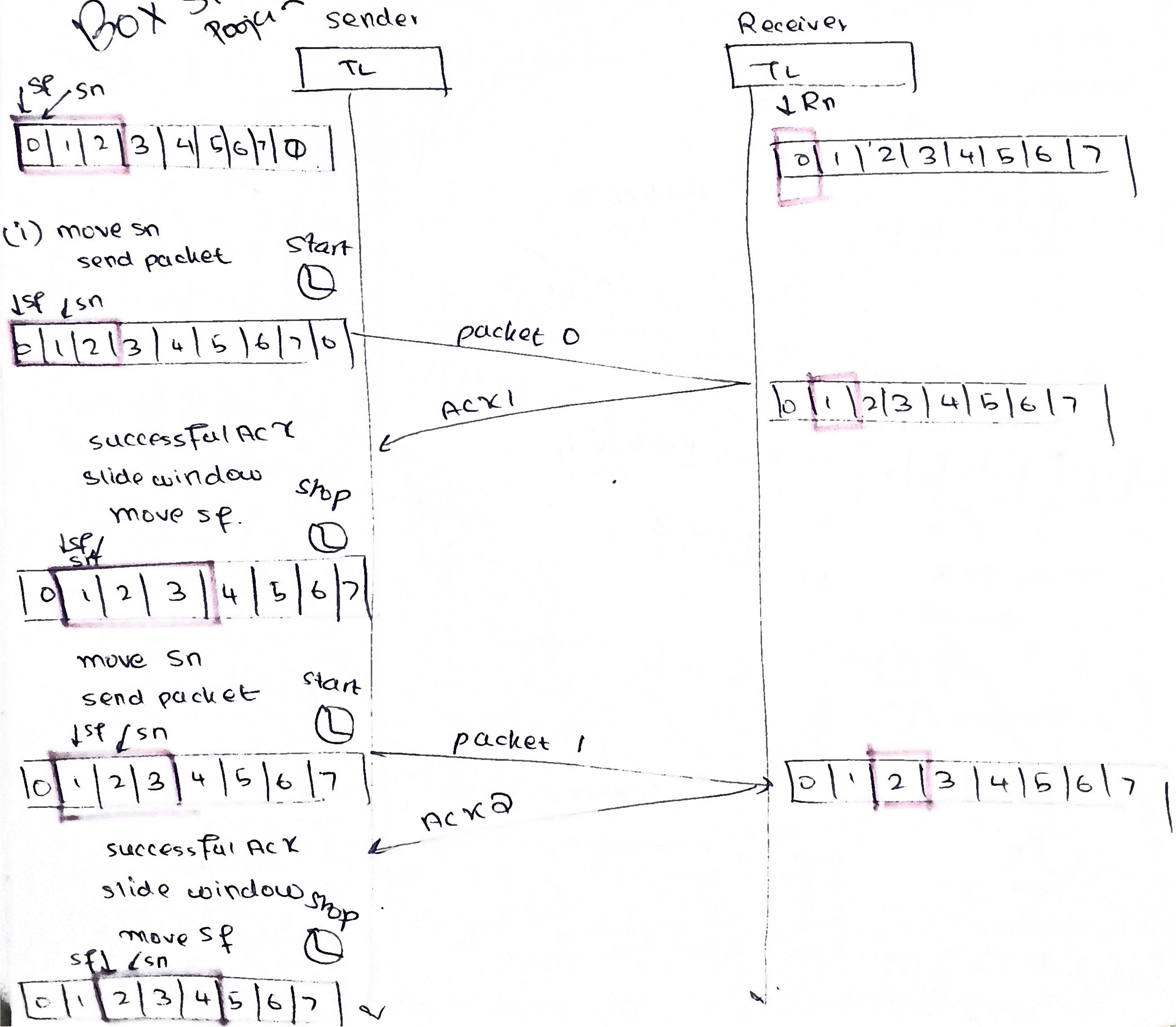
The receiver get packets 0, 1, 2, 4.

The maximum window size is . 8

Go Back N $2^m - 1 \le 8$

$m = 3$

Box Size = 7



sender

receiver

move sn

sf sn

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

TL

TL

Packet 0

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

start

ACK 0

Successful ACK

slide window

move sf

sf sn

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

stop

move sn

send packet

start

sf sn

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

packet 3

lost

no acknowledgement

move sn, send packet

packet 4

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

sf

sn

ACK 3

wrong ACK

discard

sn at the end

Timeout

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

packet 3

ACK

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

sf sn

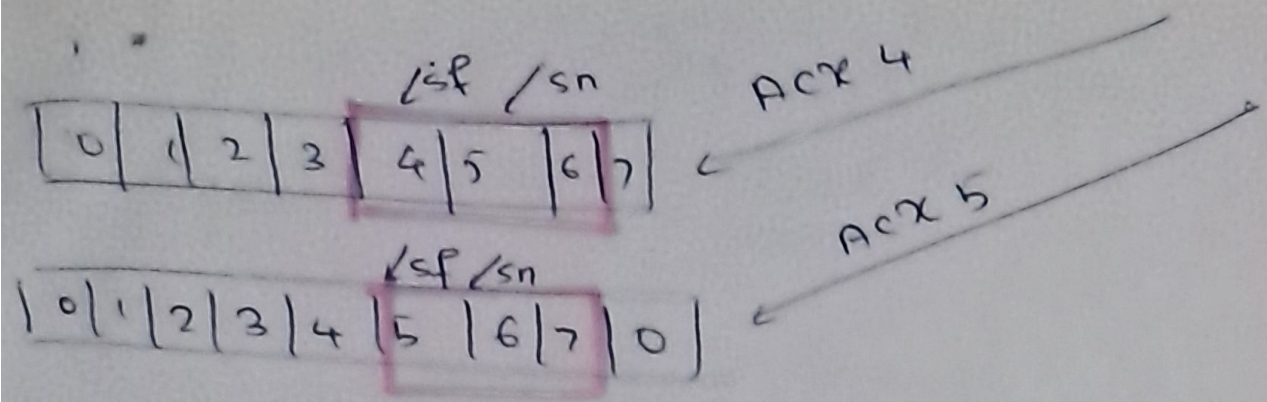
move sn

packet 4

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

sf sn



Selective - Repeat

$2^{m-1} \leq 8$

$m = 4$

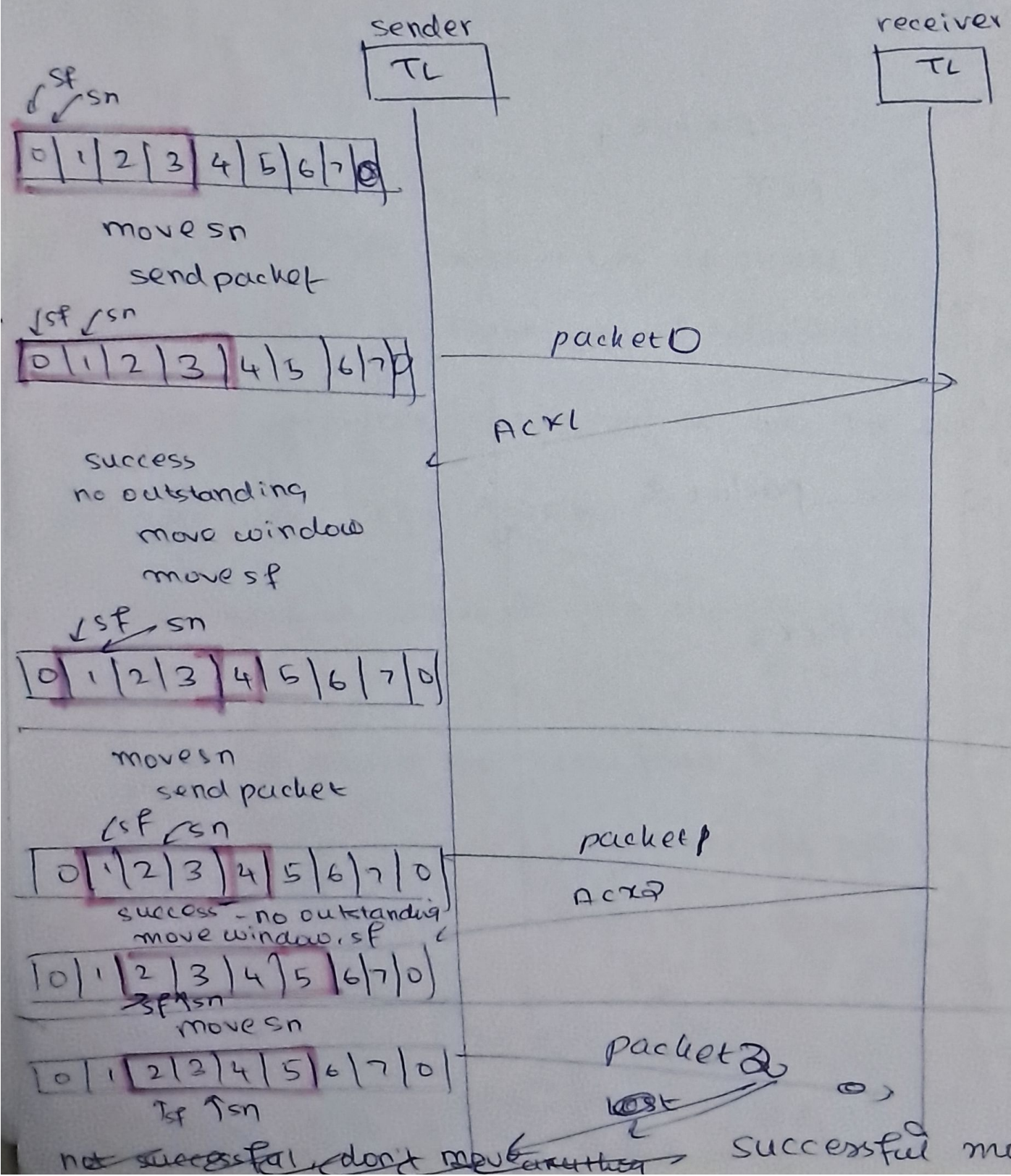
$\Rightarrow 2^3 = 8$ ~~send window size = 8~~

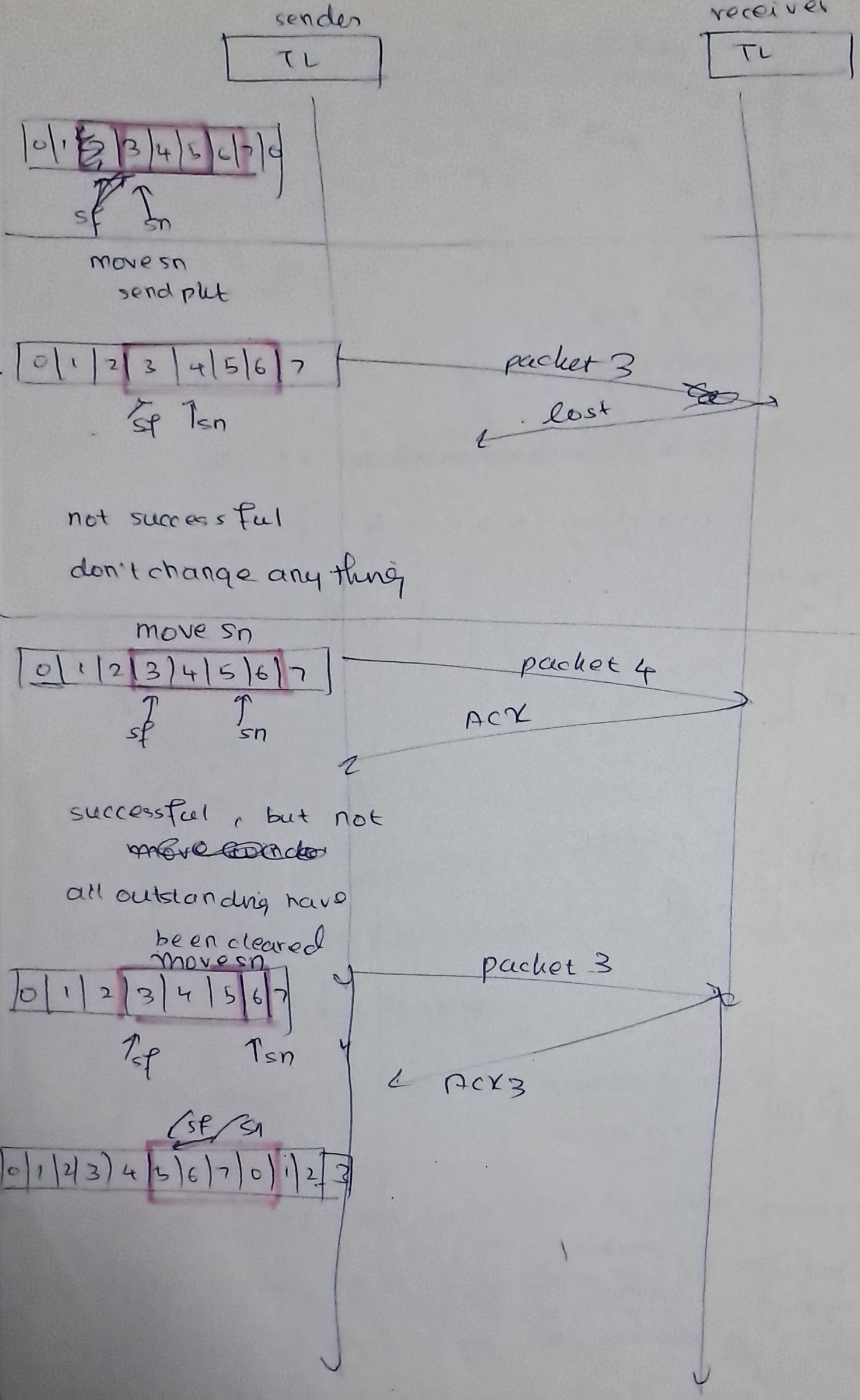
window size = 4

$2^m - 1$

$\frac{2^m}{2}$

$m = 3$




sender
receiver
TL
TL
0 1 2 3 4 5 6 7
sf
sn
move sn
send pkt
packet 3
lost
not successful
don't change anything
move sn
packet 4
ACK
successful, but not
all outstanding have
been cleared
move sn
packet 3
ACK3
sf sn