SENDING & RECEIVING BUFFERS

One implementation → use circular array of 1-byte locations
Normally buffers are hundreds or thousands of bytes.

At sending side 3 types of chambers

a: Empty chambers that can be filled by sender process
b: Bytes have been sent but not yet acknowledged.
TCP keeps these bytes in buffer until ack is received.
c: Bytes to be sent by the sender TCP.

At receiver side, circular buffer has 2 areas

a: Empty chambers to be filled by bytes from the N/W.
b: Chambers containing received bytes that can be read by receiving process.

When a byte is read by receiving process, chamber is recycled & added to pool of empty chambers.

SEGMENTS

TCP groups a no of bytes together into a packet called a segment. TCP adds header to each segment. Segments are encapsulated in IP datagram & transmitted.

Segments are not necessarily of same size & are of the order of hundreds of bytes; if not thousand of bytes.

TCP offers FULL DUPLEX service.

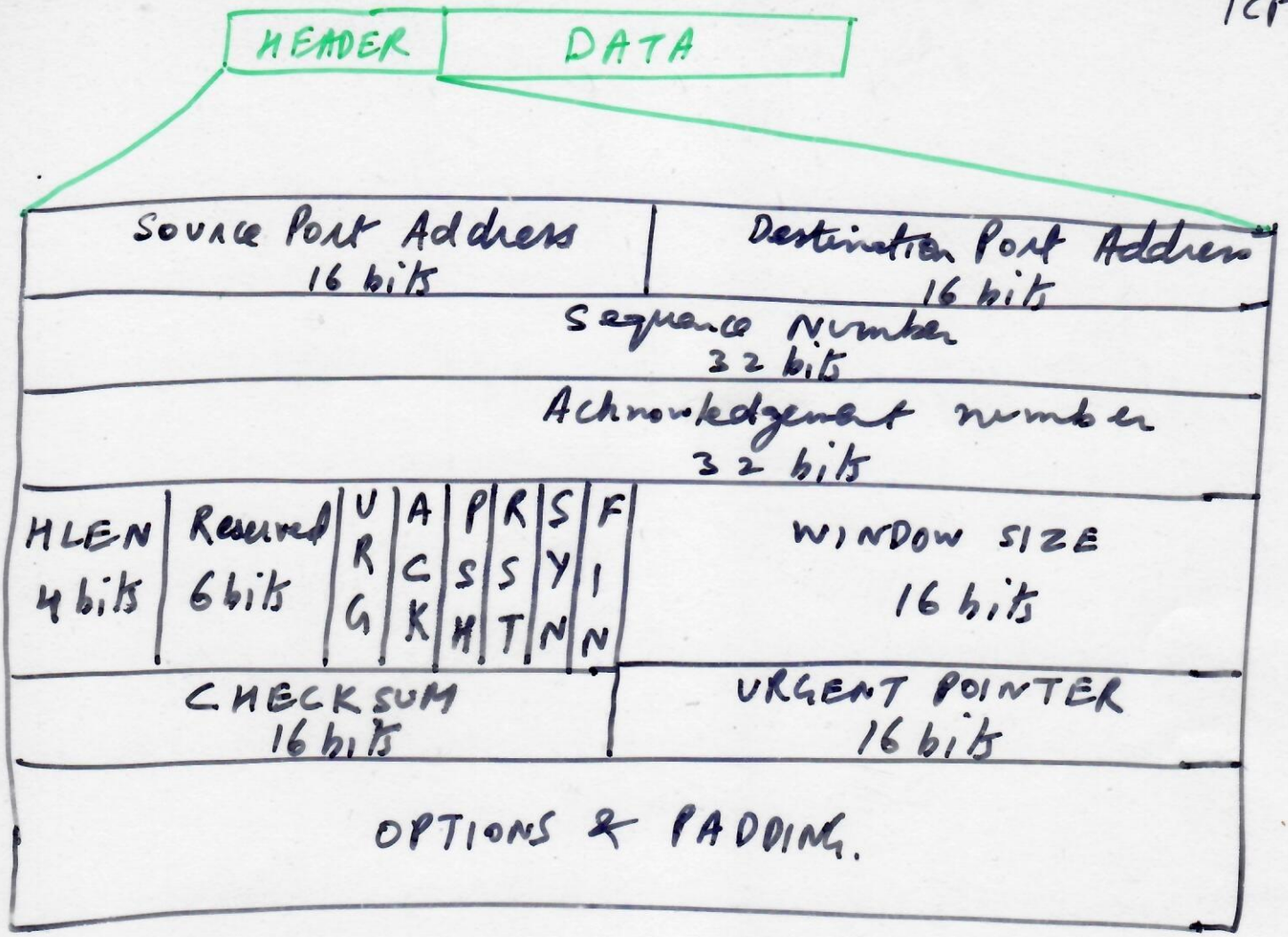
TCP FEATURES

TCP uses sequence no & acknowledgement no to keep track of bytes numbers in the segments

TCP nos the bytes stored in buffers with random nos generated between $0 - 2^{32} - 1$.

Sequence no is given to each segment that is being sent. Seg. NO = No of 1st byte in that segment.

SEGMENT



Header = 20–60 bytes

Header = 20 bytes, then no options.

Source Port NO, Dest Port NO, Sequence NO = 1st byte in the segment i.e data segment.

Ack NO, Header Length is between 20–60 bytes.

Value of this field between 5 (5×4 = 20) & 15 (15×4 = 60).

Reserved — for future use.

Control defines 6 different control bits or flags.
One or more bits can be set at a time.

URG : Urgent pointer is valid : used for urgent data.

ACK : Ack is valid

PSH : Push the data

RST : Connection must be reset.

SYN Synchronize seq nos during connection TCP 4 TL

FIN Terminate the connection

WINDOW SIZE Max size 65535 bytes. Window size is generally called receiving window (rwnd) & is determined by the receiver.

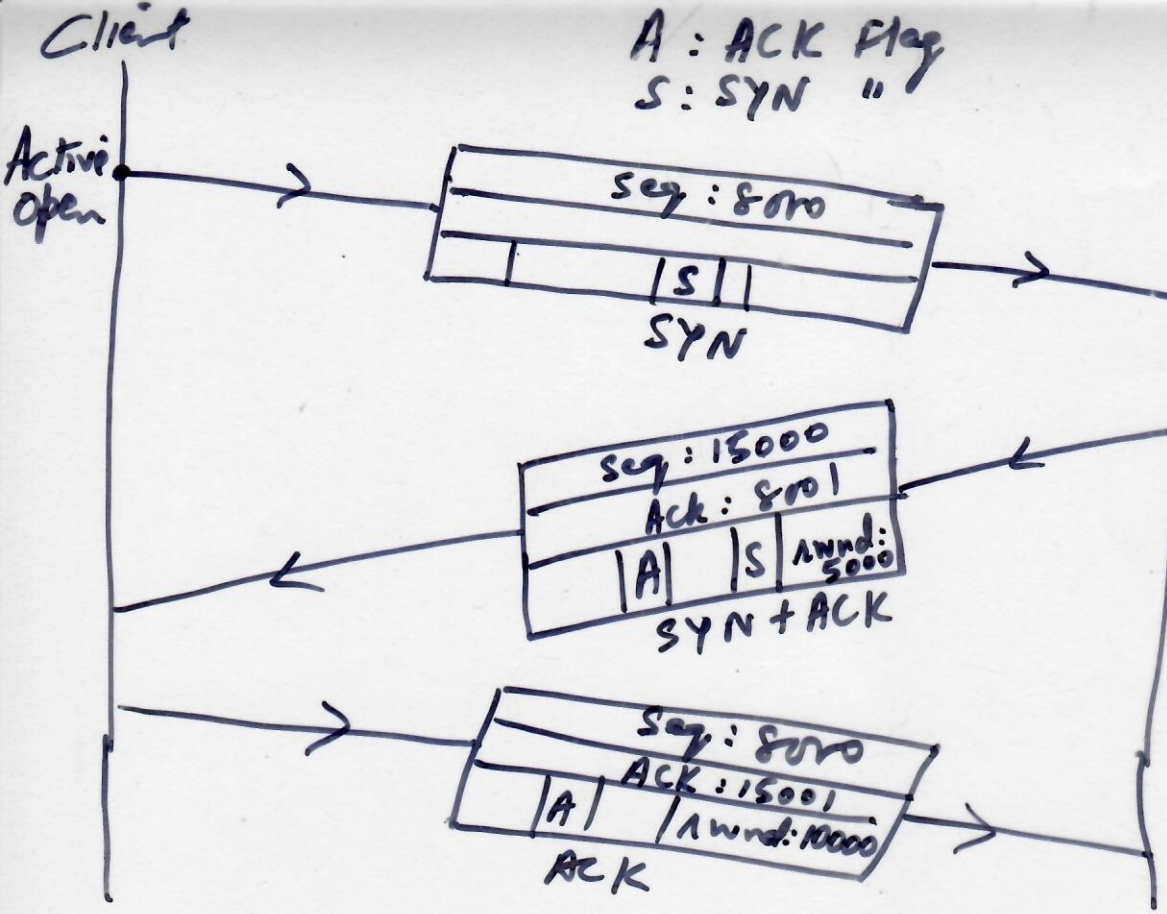
CHECKSUM

OPTIONS can be upto 40 bytes of information.

TCP CONNECTION establishes a virtual connection using single virtual pathway for entire message facilitates ack process as also retransmission of damaged or lost frames. TCP controls the connection.

Connection Establishment (Three Way Handshaking)

a. Process starts with server.
b. Server program tells its TCP that it is ready to accept a connection. This is request for passive open.
c. Client program issues a request for an active open.
1. Client sends 1st segment, SYN segment, in which only SYN flag is set. Client seeds Initial sequence No ~~(ISN)~~ (ISN). This segment doesn't have ACK no.
SYN segment = control segment, so no data.
SYN " has a sequence no
At Data transfer sequence no is one more than SYN segment seq no.



2. Server sends SYN+ACK segment. Serves 2 purposes
It is SYN segment for communication in this direction.
Server uses it to initialize a seq no for numbering the bytes sent from server to client.
Server ACKnowledges.

3. Client send 3rd segment, just an ACK segment.
Doesn't consume any seq no.
Some implementations carry user data, where seq no is consumed.

Issues Involved in Connection Establishment

(a) **Simultaneous Open** when both processes issue active open.
Both TCPs send SYN+ACK segment to each other & one single connection is established between them.

(b) **SYN Flooding Attack** when a malicious attacker sends large SYN segments to server pretending each coming from different IP addresses.

Detecting corrupted segments, lost segments, out of order segments & duplicated segments.

Error correction & detection by checksum, ACK, Time-out.

CHECKSUM

Corrupted segment dropped by dest TCP & considered lost. TCP uses 16 bit checksum.

ACKNOWLEDGEMENT

Some Imp rules

1. Piggybacking, reduces no of segments.
2. Receiver needs to delay sending an ACK segment if there is only one outstanding in-order segment.
3. When a segment arrives with a seq no that is expected by the receiver & the previous in-order segment has not been Acknowledged, the receiver immediately sends an ACK segment.
4. When a segment arrives with an out-of-order seq no that is higher than expected, receiver immediately sends ACK segment announcing the seq no of the next expected segment.
5. When a missing segment arrives, the receiver sends an ACK segment to announce the next seq no expected. This informs receiver that segments reported missing have been recd.
6. If a duplicate segment arrives, receiver sends ACK immediately. This solves some problems when an ACK itself is lost.

ACK Type

1. Accumulative ACK Receiver advertises next byte it expects to receive, ignoring all segments recd out of order. Discarded, lost or duplicate segments are not reported. 32 bit ACK field in TCP header is used for accumulative ACK & its value is valid only when ACK flag bit = 1

2. Selective ACK doesn't replace ACK, but reports additional info to sender. Out of order & duplicated segments are reported. SACK is implemented as option at end of TCP header.

RETRANSMISSION (a) occurs, if retransmission timer expires.

(b) sender receives 3 duplicate ACKs.

Retransmission After RTO (Retransmission Time-Out)

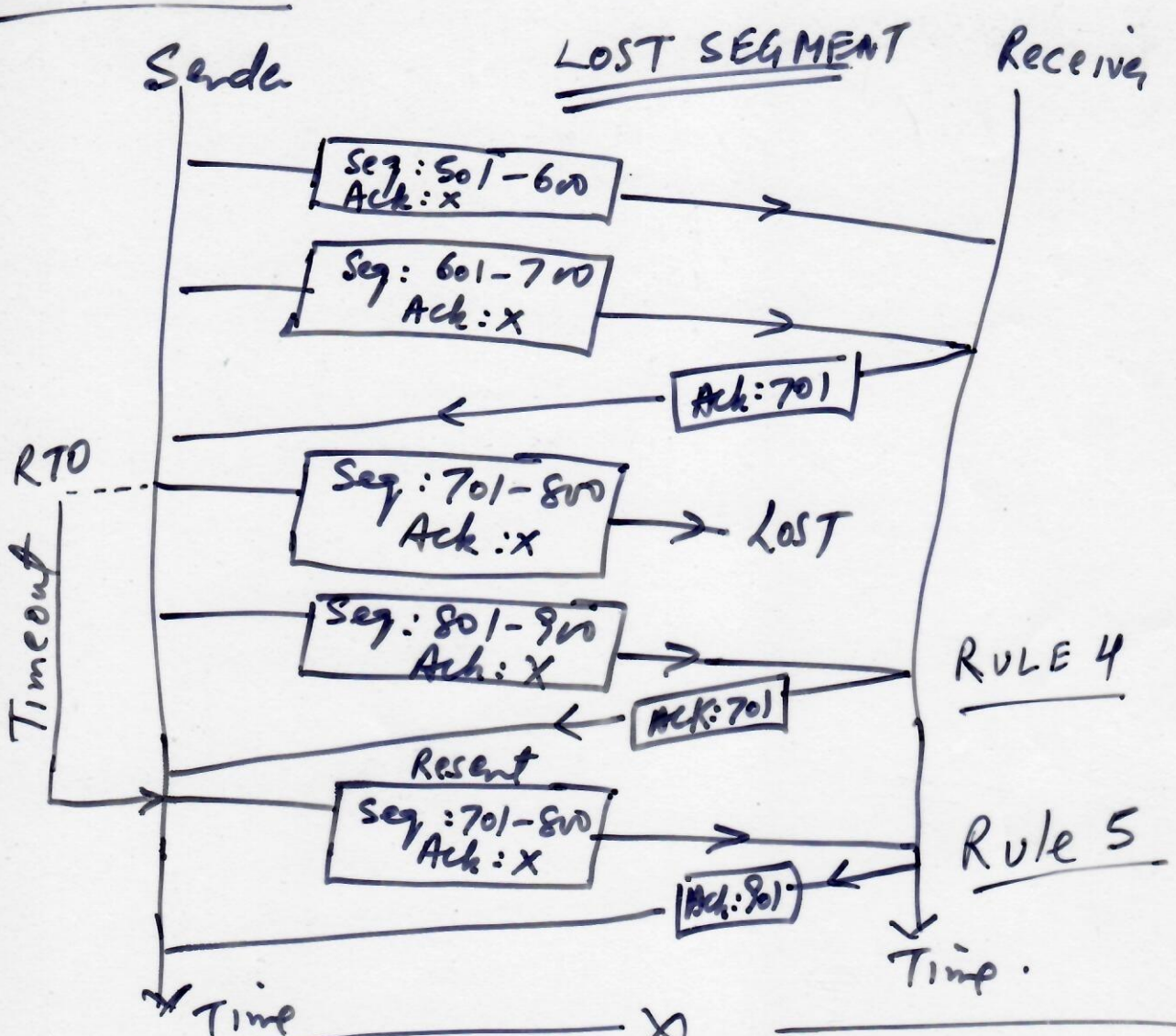
Source TCP starts one retransmission time out (RTO) timer for each segment sent.

RTO is dynamic & is updated based on round trip time (RTT) of segments.

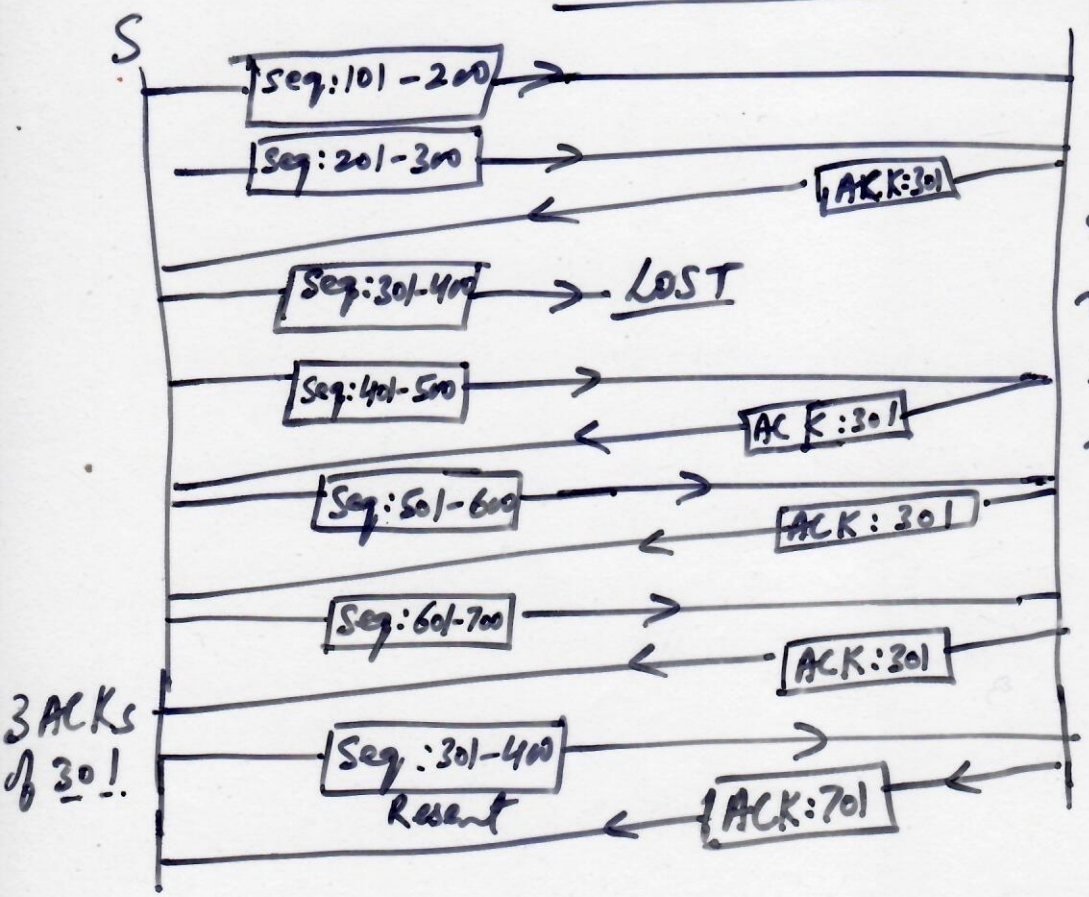
Out of Order Segments Originally TCP discarded all out of order segments. Now, out of order segments are not discarded but stored temporarily till missing segments arrive, then given to process (in order).

Some Scenarios

LOST SEGMENT



FAST RETRANSMISSION

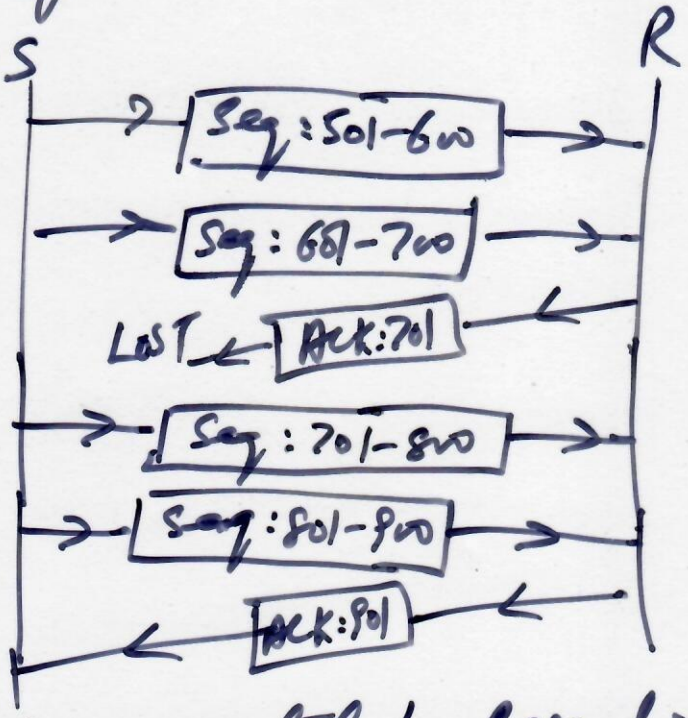


Although Timer for seq 3 has not matured yet, the 3rd segment is sent immediately because of 3 ACKs of 301 received by sender.

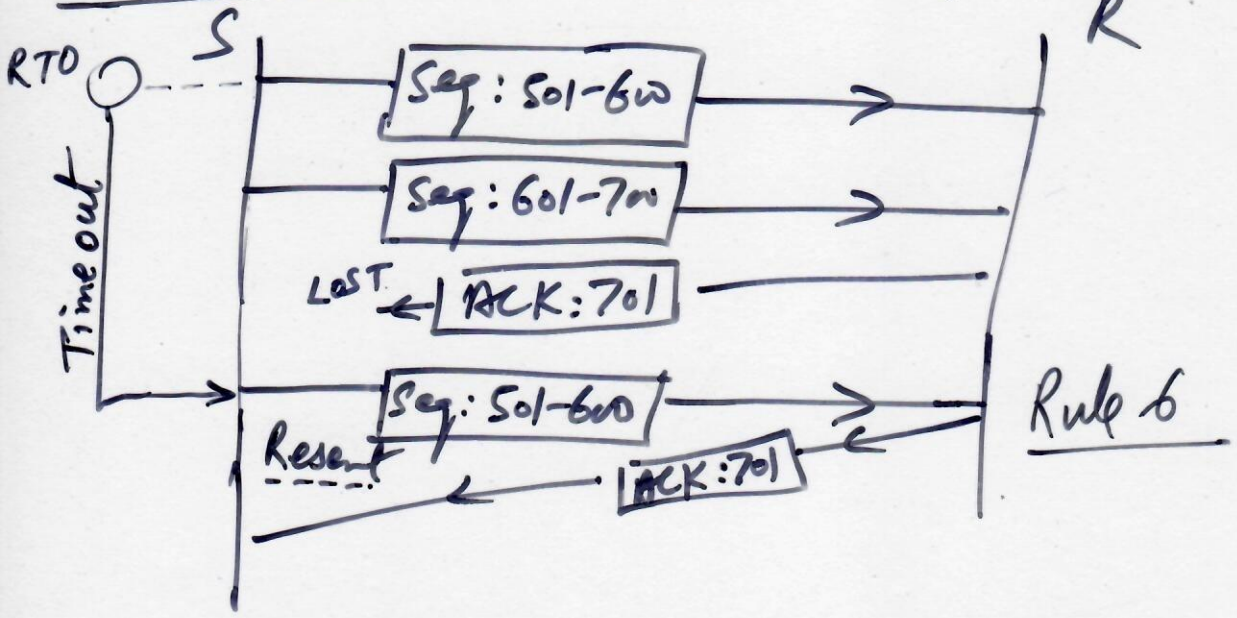
Delayed Segment Diff route leads to delayed segments.
" " may arrive after it has been resent. (Duplicate segment).

Duplicate Segment Duplicate seg is handled as follows:
when a seg arrives that contains a seq no less than the previously Acknowledged bytes, it is discarded.

Automatically corrected Lost ACK

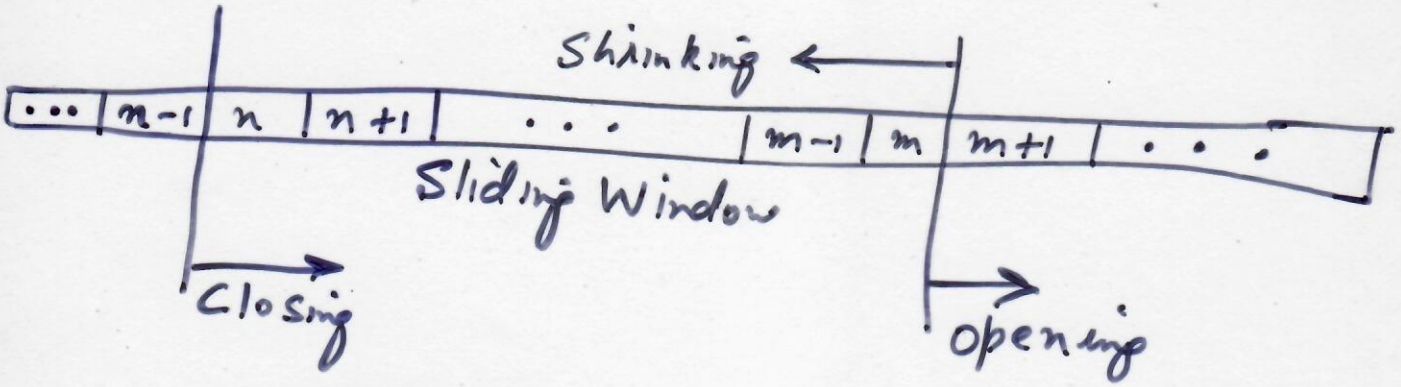


Lost ACK corrected by Resending a seg



FLOW CONTROL regulates the amount of data a source can send before receiving an acknowledgement from the destination. TCP defines a window that is imposed on the buffer of data delievered from the application program & ready to be sent. TCP sends an amount of data defined by the sliding window protocol.

Sliding Window Protocol TCP uses sliding window protocol for flow control. In this method, host uses a window for outbound communication. (sending data). window is portion of the buffer containing bytes received from the process. Bytes inside the window are the bytes that can be in transit. The imaginary window has 2 walls, one left & one right. The window is called sliding window because the right & left walls can slide as shown below:-



window size = minimum (rwnd, cwnd)

Window is opened, closed or shrunk. These are in the control of the receiver, not the sender.

Opening window implies moving the right wall to the right. This allows new bytes in the buffer that are eligible for sending. Closing window is moving left wall to the right which means some bytes have been acknowledged. Shrinking is moving ~~right~~ left wall ~~to the right~~ left. This is strongly discouraged & not allowed in some implementations, because it means revoking the eligibility of some bytes for sending. If sender has already sent these bytes, then previous ACKs have to be revoked, which is a problematic area.

However there is an exception i.e the receiver can temporarily shut down the window by sending rwnd=0 This implies receiver doesn't want data from sender for a while.

Some facts about TCP's sliding windows

1. Size of the window is the lesser of rwnd & cwnd.
2. Source doesn't have to send a full window's worth of data.
3. Window can be opened or closed by the receiver but should not be shrunk.
4. Destination can send an ACK at any time as long as it doesn't result in a shrinking window.
5. The receiver can temporarily shut down the window the sender, however, can always send a segment of one byte after the window is shut down.

Silly Window Syndrome Problems can occur in sliding window operation when sender creates data slowly or receiver consumes data slowly or both. Any of these can result in reduced efficiency. For e.g. if TCP sends segments of 1 byte of data, it implies 41 byte datagram (20 bytes of TCP header + 20 bytes of IP header). This problem is called silly window syndrome.

Syndrome created by Sender The process writes 1 byte at a time into the buffer of the sending TCP.

Solution is to prevent the sending TCP from sending data byte by byte. TCP must be forced to wait for more data. How long should TCP wait? If longer is wait it delays the process. If wait is small, it may result in small segments. Nagle found elegant solution.

Nagle's Algorithm

1. The sending TCP sends the 1st piece of data it receives from the sending application program even if it is 1 byte.
2. After sending 1st segment, data is gathered in the O/P buffer & waits until ACK is received or enough data is received to fill a maximum-size segment. Now TCP can send the segment.
3. Step 2 is repeated for the rest of transmission.

Nagle's Algo takes into account speed of application program & the speed of N/w that transports the data.

If application program is faster than the N/w, segments are larger.
" " " " slower " " " , " " smaller.

Syndrome Created by the Receiver

Receiving TCP may create a silly window syndrome if it is serving an application program that consumes data slowly, for e.g, 1 byte at a time. Suppose sender creates data in blocks of 1 kbyte, but receiver consumes data 1 byte at a time. Suppose IP buffer of receiver is 4 kbytes. sender sends the 1st 4 kbytes of data. Receiver stores it in its buffer, which is full. Receiver advertises window size of zero, which implies sender should stop sending. Receiver reads the 1st byte of data from IP buffer. Receiver advertises window size of 1 byte & sender sends 1 byte. The procedure will continue.

Two solutions have been proposed

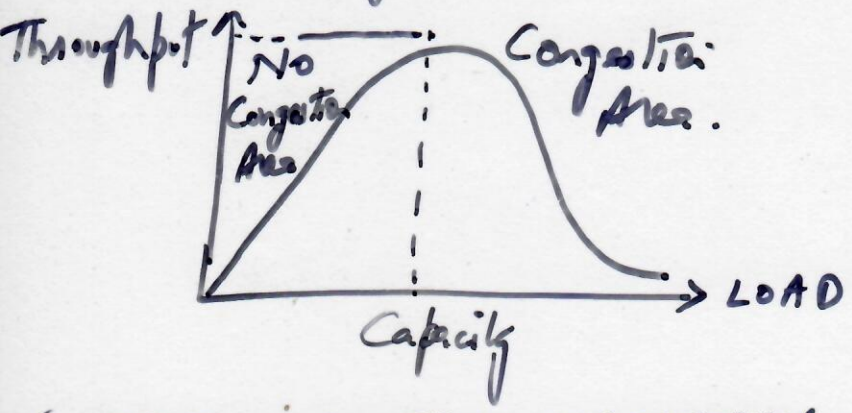
Clark's Solution: To send an ACK as soon as data arrives, but to announce a window size of zero until there is enough space to accomodate a segment of maximum size or until half of the buffer is empty.

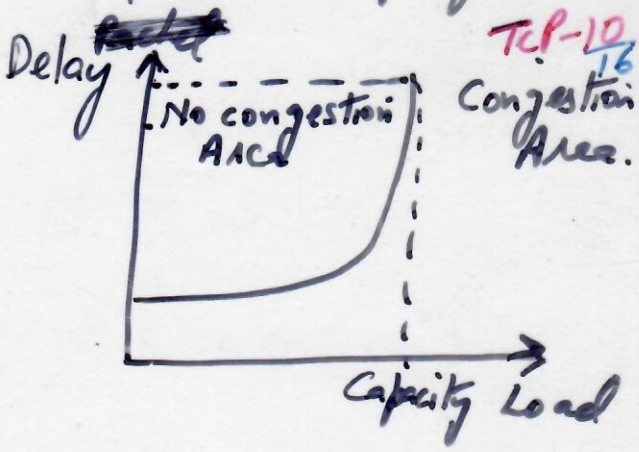
Delayed ACK: when segment arrives, it is not ACK immediately. Receiver waits until there is a decent amount of space in its incoming buffer before ACKing the arrived segments. Delayed ACK prevents the sending TCP from sliding its window. After it has sent the data in the window, it stops. This kills the syndrome.

Delayed ACK has the advantage of reduced traffic. Receiver doesn't have to acknowledge each segment. Disadvantage of delayed ACK is, it may force the sender to retransmit the unacknowledged segments.

The protocol balances the advantages & disadvantages. It now defines that the ACK should not be delayed by more than 500 ms.

CONGESTION CONTROL involves 2 factors that measure performance of a N/W: delay & throughput.





CONGESTION CONTROL MECHANISMS

(a) Open Loop Congestion Control policies are applied to prevent congestion. Congestion control by — source or dest.

Brief List of Policies for congestion prevention are:-

(i) Retransmission Policy Good retransmission policy can prevent congestion.

(ii) Acknowledgment Policy imposed by receiver may also affect congestion.

(iii) Discard Policy Good discard policy possessed by routers may prevent congestion & at same time will not harm the integrity of transmission.

(b) Closed Loop Congestion Control policies are applied to remove congestion after it happens. Few mechanisms are

(i) Back Pressure when router is congested, it can inform the previous router to reduce rate of outgoing packets. This action can be recursive upto router just prior to source.

(ii) Choke Point packet sent by router to source to inform congestion

(iii) Implicit Signaling Source can detect implicit signal warning of congestion & slow down its sending rate.

(iv) Explicit Signaling Routers that experience congestion can send an explicit signal, setting of a bit in packet to inform sender/receiver of congestion.

CONGESTION CONTROL IN TCP

cwnd = congestion window size

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

Congestion Policy TCP general policy for handling congestion is based on 3 phases :- slow start, congestion avoidance, congestion detection. In slowstart, sender starts with very slow rate, but increases rate rapidly to reach threshold. After reaching threshold data rate is reduced to avoid congestion. Finally if congestion is detected, sender goes back to slow start or congestion avoidance phase based upon how congestion is detected.

Slow Start starts with cwnd = 1, then increased exponentially upto slowstart threshold. Then slow start stops & congestion avoidance phase starts. cwnd = 1 implies 1 MSS (Max segment Size)

Slowstart threshold = 65535 bytes (usually)

Congestion Avoidance When threshold is reached, then window size is increased additively, until congestion is detected.

Congestion Detection (a) Retransmission Timer expires, then
Threshold ~~window size~~ = $\frac{1}{2}$ x Current window size
& Slow start phase starts where cwnd = 1

(b) When 3 ACKs are received, then
Threshold ~~window size~~ = $\frac{1}{2}$ x Current window size
cwnd = slow start threshold
& Congestion Avoidance phase starts.

TCP Timers

(a) Retransmission
(b) Persistence
(c) Keepalive
(d) TIME-WAIT

Retransmission Timer To retransmit a lost segment, TCP employs a " " that handles Retransmission Time-out (RTO) " waiting time for an ACK of a segment.

For every segment Retransmission Timer is created.

2 situations may occur

(a) If ACK is recd for a particular segment before timer expires, timer is destroyed.

(b) If Timer expires, before ACK arrives, segment is transmitted & timer reset.

ROUND TRIP TIME (RTT) To calculate RTO, we require RTT.

Measured RTT (RTT_M) · Only one RTT_M at any time.

Smoothed RTT Fluctuation in RTT_M is so high in today's internet, that it can't be used for retransmission time out purposes. Most implementations used smoothed RTT called RTT_S which is weighted average of RTT_M & previous RTT_S.

Original → No value

After 1st measurement → $RTT_S = RTT_M$

After any other " → $RTT_S = (1-\alpha)RTT_S + \alpha \cdot RTT_M$

Value of α is implementation dependent, but normally $= \frac{1}{8}$.

RTT Deviation (RTT_D) Most implementations use this.

Original → No value

After 1st measurement → $RTT_D = RTT_M / 2$

After any other " → $RTT_D = (1-\beta)RTT_D + \beta|RTT_S - RTT_M|$

β is implementation dependent, but usually $= \frac{1}{4}$.

Retransmission Time Out (RTO) based upon RTTs & its deviation:

Original → Initial value

After any measurement ⇒ $RTO = RTT_S + 4RTT_D$

KARN'S ALGORITHM A segment not acknowledged during retransmission period is retransmitted. Sender TCP receives ACK for this segment, it doesn't know if ACK is for original segment or for the retransmitted one. The value of new RTT is based upon departure of segment. However, if original segment was lost & ACK is for the retransmitted one, the value of current RTT must be calculated from the time the segment was retransmitted. This is a dilemma which was solved by Karn. Karn solution is simple. Don't consider RTT of a retransmitted in calculation of new RTT. Do not update value of RTT until you send a segment & receive an ACK w/o the need for retransmission.

Exponential Backoff What is value of RTO if retransmission occurs. Value of RTO is doubled for each retransmission. If segment is retransmitted once, value is 2 times the RTO, if it transmitted twice, value is four times the RTO & so on.

Persistence Timer Receiving TCP announces window size = 0. Sending TCP stops transmitting until receive announces non-zero window size. This ACK can be lost leading to deadlock. There is no retransmission timer for a segment containing only an ACK.

TCP 14
16

To resolve deadlock TCP uses Persistence Timer.
Sender TCP starts persistence timer when it receives an ACK with window size = 0.

When timer elapses, sending TCP sends special segment called a probe, containing just one byte of data. It has a seq no, but this seq no is never acknowledged. Probe alerts receiver that ACK was lost & must be resent.

Value of Persistence Timer = value of Retransmission Time.

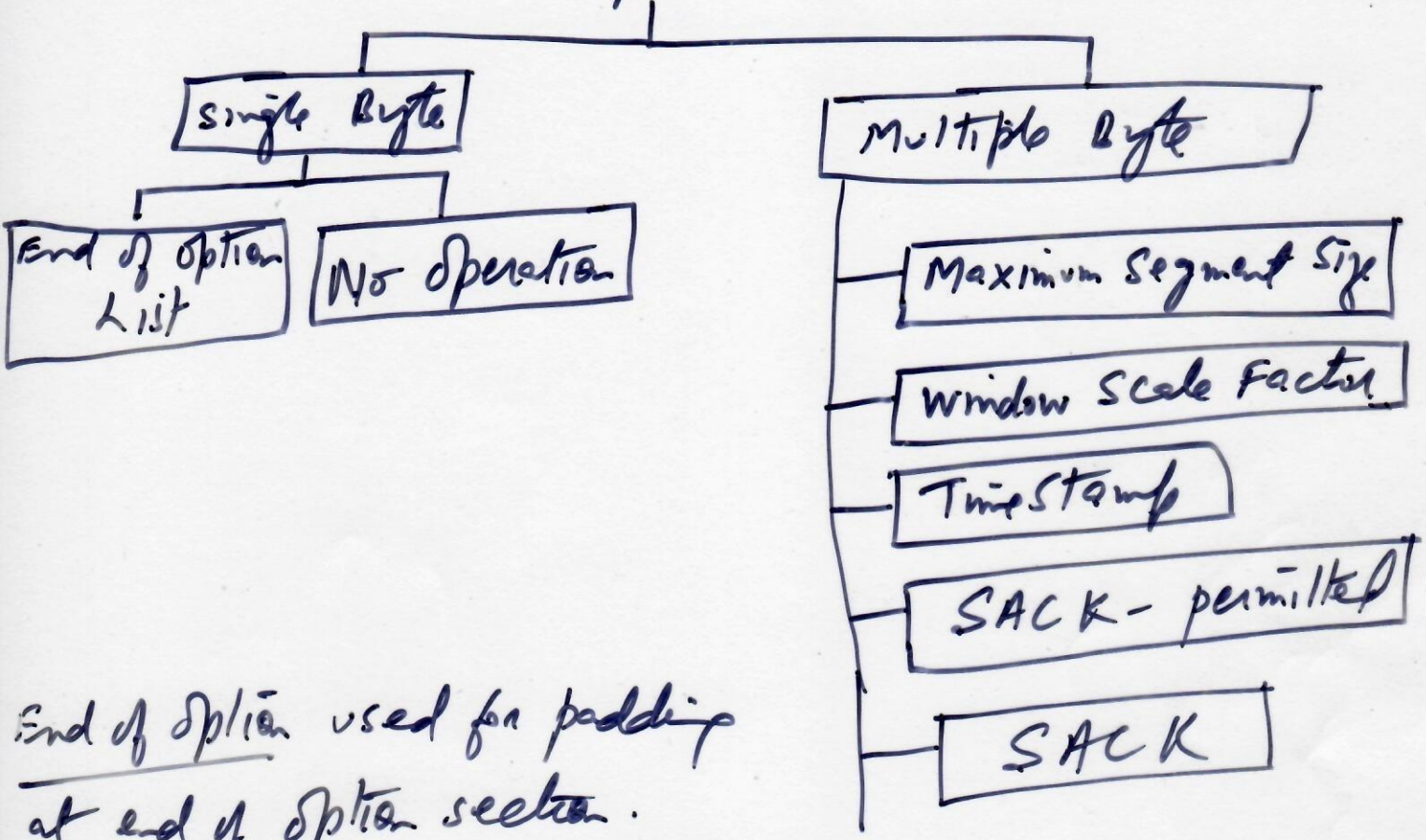
If no response from receiver, another probe segment & value of persistence timer is doubled & reset. This process of sending probe segment is repeated till value of persistence timer reaches threshold (usually 60s) After that sender sends one probe segment every 60 s until window is opened.

Keepalive Timer To prevent long idle connection between 2 TCPs. Suppose client opens a TCP connection & becomes silent, connection opens forever. Most implementations equip server with a keepalive timer. Each time server hears from client, it resets timer. Time out is usually 2 hours. If server doesn't hear from client after 2 hrs, it sends probe segment. If no response after 10 probes, each of which is 75s apart, it assumes client is down & terminates connection.

Time Wait Timer used during connection termination.

OPTIONS convey additional info to destination or align other options.

TCP IS
16



End of option used for padding at end of option section.
After this option receiver looks for payload data.

NO Operation used as a filler. However, it usually comes before another option to make an option fit in a 4 word slot.

Maximum Segment Size (MSS) define largest unit of data that can be received by dest TCP. It defines max data size & not max size of segment. MSS = 16 bit field.
Default value of MSS = 536 bytes.

Window Scale Factor 16 bit field, defines size of sliding window, which can range from 0 – 65535 bytes. This size may not be sufficient. To increase window size, window scale factor is used.

New window size = window size in header $\times 2^{\text{window scale factor}}$.

TCP 16/16

Although scale factor could be as large as 255, the largest value allowed by TCP/IP is 14 i.e max window size is $2^{16} \times 2^{14} = 2^{30}$ which is less than max value of seq no.

Last Slide TCP

Timestamp 10 byte option. Timestamp has 2 applications (a) Measures RTT (b) Prevents wrap around seq nos.

Identification of segment = combination of seq no + timestamp

SACK Permitted and SACK Options

SACK Permitted → 2 bytes. SACK permitted used only during connection establishment. Host that sends SYN segment adds this option to show that it can support SACK option.

SACK option, of variable length used during data Xfer only if both end agree. The option includes list of blocks arriving out-of-order.