Department Name: Computer Science
Course Name: M. Tech
Semester: 2nd
Paper Name: Advanced Computer Network (MTCS-201)
Topic: Berkley Sockets, Silly Window Syndrome, Sliding Window Protocol, Finite State Machine Model

1. BERKLEY SOCKETS

Socket:

1. Sockets are a service provided by transport layer.
2. A socket is one endpoint of a two-way communication link between two programs running on the network.

Berkeley Socket:

1. A Berkeley socket is an application programming interface (API) for Internet sockets and UNIX domain sockets.
2. It is used for inter-process communication (IPC).
3. It is commonly implemented as a library of linkable modules.
4. It originated with the 4.2BSD UNIX released in 1983.
5. The term **POSIX sockets** are essentially synonymous with Berkeley sockets, but they are also known as BSD sockets.

Primitive used in Berkeley Socket:

Primitives	Meaning
SOCKET	Create a New Communication Endpoint.
BIND	Attach a Local Address to a SOCKET.
LISTEN	Shows the Willingness to Accept Connections.
ACCEPT	Block the Caller until a Connection Attempts Arrives.
CONNECT	Actively Attempt to Establish a Connection.
SEND	Send Some Data over Connection.
RECEIVE	Receive Some Data from the Connection.
CLOSE	Release the Connection.

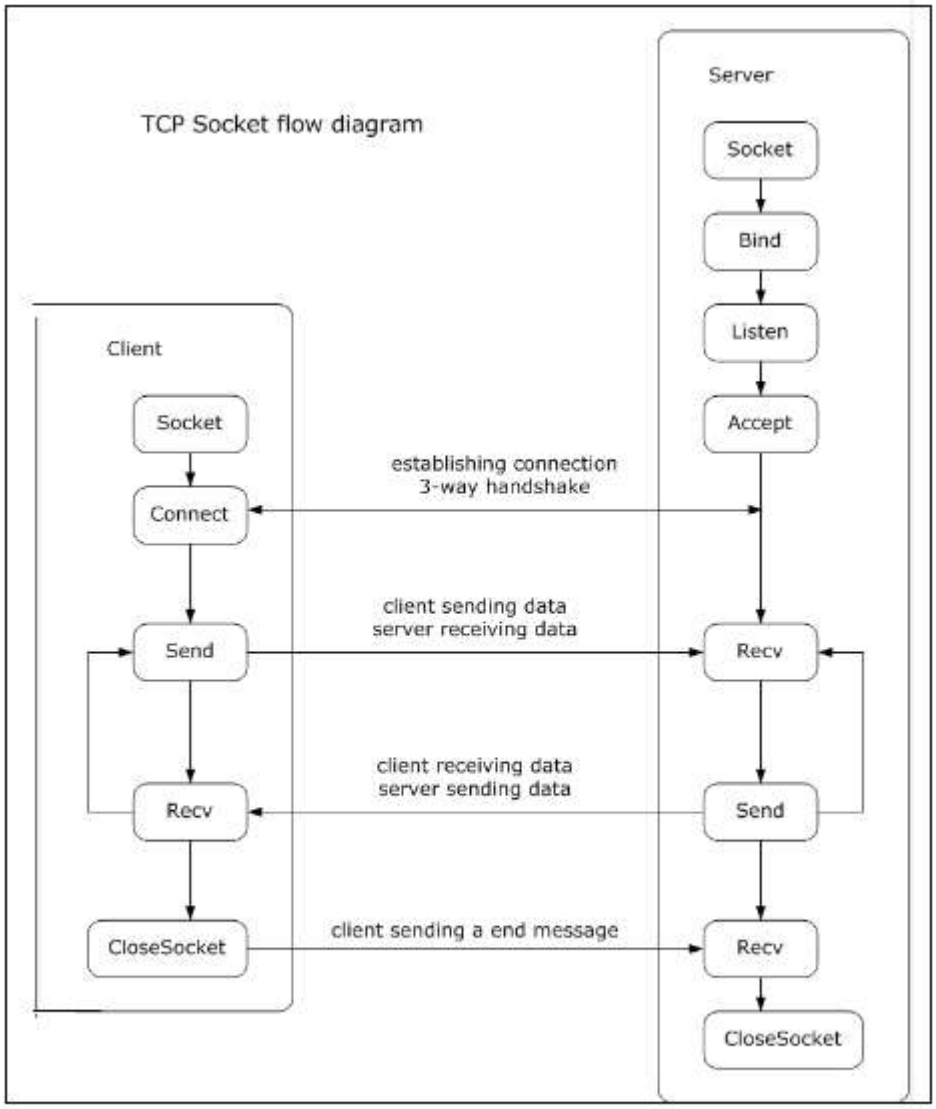
Socket Programming:

I) Server side:

- Server startup executes SOCKET, BIND¬ & LISTEN primitives.
- LISTEN primitive allocate queue for multiple simultaneous clients.
- Then it use ACCEPT to suspend server until request.
- When client request arrives: ACCEPT returns.
- Start new socket (thread or process) with same properties as original, this handles the request, server goes on waiting on original socket.
- If new request arrives while spawning thread for this one, it is queued.
- If queue full it is refused.

II) Client side:

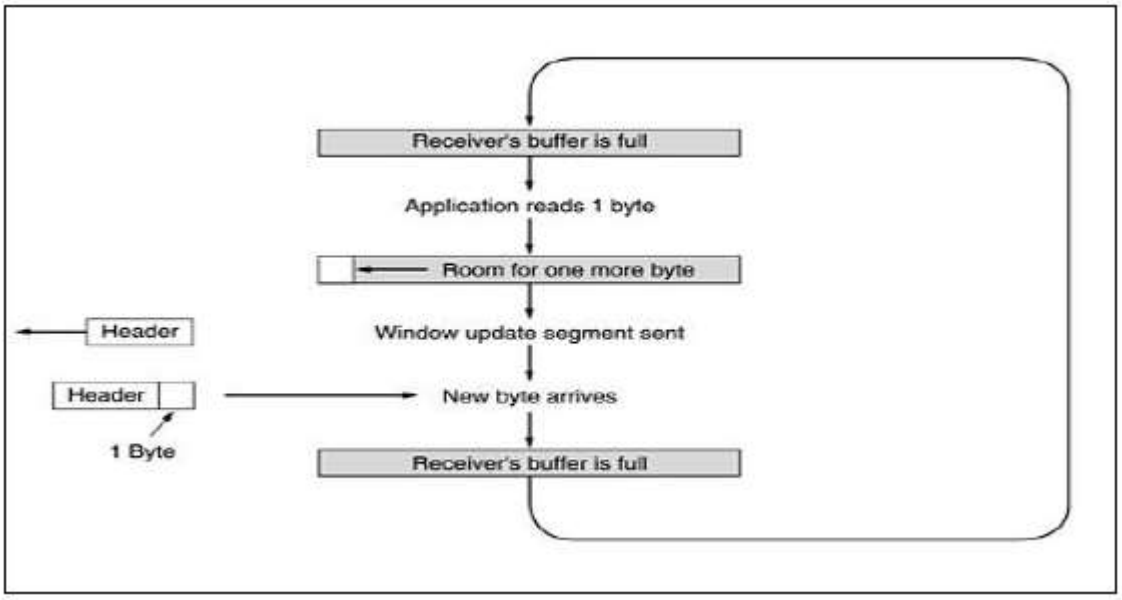
- It uses SOCKET primitives to create.
- Then use CONNECT to initiate connection process.
- When this returns the socket is open.
- Both sides can now SEND, RECEIVE.
- Connection not released until both sides do CLOSE.
- Typically client does it, server acknowledges.


TCP Socket flow diagram
Server
Socket
Bind
Listen
Accept
Client
Socket
Connect
establishing connection
3-way handshake
client sending data
server receiving data
Send
Recv
client receiving data
server sending data
Recv
Send
CloseSocket
client sending a end message
Recv
CloseSocket

2. SILLY WINDOW SYNDROME

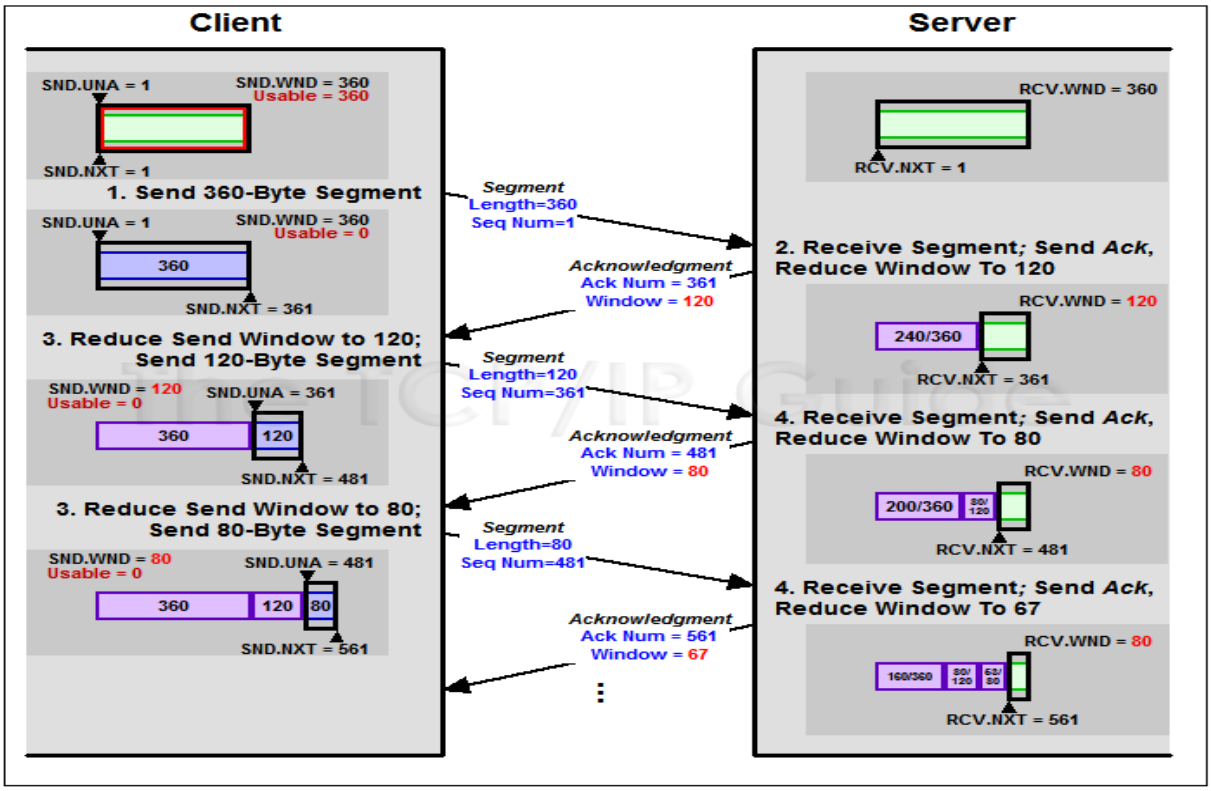
- This is one of the problems that ruin the TCP performance, which occurs when data are passed to the sending TCP entity in large blocks, but an interactive application on the receiving side reads 1 byte at a time.
- Silly Window Syndrome is a problem that arises due to the poor implementation of TCP.
- It degrades the TCP performance and makes the data transmission extremely inefficient.
- The problem is called so because-
  - It causes the sender window size to shrink to a silly value.
  - The window size shrinks to such an extent where the data being transmitted is smaller than TCP Header.

Example1



- Initially the TCP buffer on the receiving side is full and the sender knows this(win=0)
- Then the interactive application reads 1 character from TCP stream.
- Now, the receiving TCP sends a window update to the sender saying that it is all right to send 1 byte.
- The sender obligates and sends 1 byte.
- The buffer is now full, and so the receiver acknowledges the 1 byte segment but sets window to zero. This behavior can go on forever.

Example2



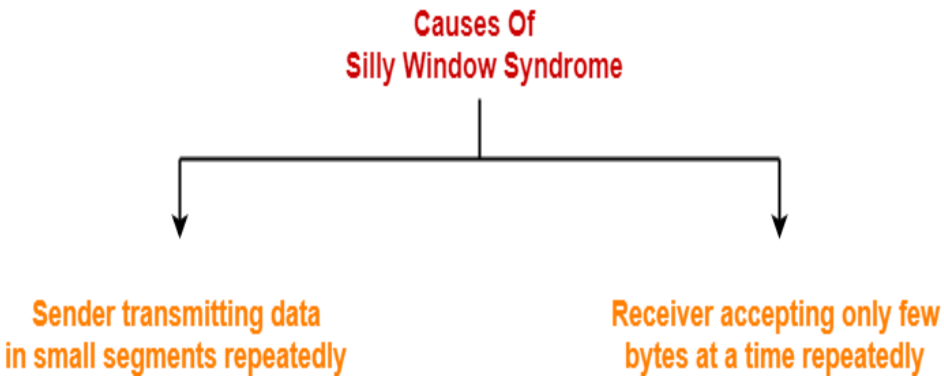
This diagram shows one example of how the phenomenon known as TCP *silly window syndrome* can arise. The client is trying to send data as fast as possible to the server, which is very busy and cannot clear its buffers promptly. Each time the client sends data the server reduces its receive window. The size of the messages the client sends shrinks until it is only sending very small, inefficient segments.

(Note that in this diagram I have shown the server's buffer fixed in position, rather than sliding to the right as in the other diagrams in this section. This way you can see the receive window decreasing in size more easily.

Causes-

The problem arises due to following causes-

- Sender transmitting data in small segments repeatedly
- Receiver accepting only few bytes at a time repeatedly



Cause-01: Sender Transmitting Data In Small Segments Repeatedly-

- Consider application generates one byte of data to send at a time.
- The poor implementation of TCP causes the sender to send each byte of data in an individual TCP segment.

This problem is solved using Nagle's Algorithm.

Nagle's Algorithm-

Nagle's Algorithm tries to solve the problem caused by the sender delivering 1 data byte at a time.

Nagle's algorithm suggests-

- Sender should send only the first byte on receiving one byte data from the application.
- Sender should buffer all the rest bytes until the outstanding byte gets acknowledged.
- In other words, sender should wait for 1 RTT.
- After receiving the acknowledgement, sender should send the buffered data in one TCP segment.
- Then, sender should buffer the data again until the previously sent data gets acknowledged.

Cause-02: Receiver Accepting Only Few Bytes Repeatedly-

- Consider the receiver continues to be unable to process all the incoming data.
- In such a case, its window size becomes smaller and smaller.
- A stage arrives when it repeatedly sends the window size of 1 byte to the sender.

This problem is solved using Clark's Solution.

Clark's Solution-

Clark's Solution tries to solve the problem caused by the receiver sucking up one data byte at a time.

Clark's solution suggests-

- Receiver should not send a window update for 1 byte.
- Receiver should wait until it has a decent amount of space available.
- Receiver should then advertise that window size to the sender.

Specifically, the receiver should not send a window update-

- Until it can handle the MSS it advertised during Three Way Handshake
- Or until its buffer is half empty, whichever is smaller.

Note-01:

Nagle's algorithm is turned off for the applications that require data to be sent immediately.

This is because-

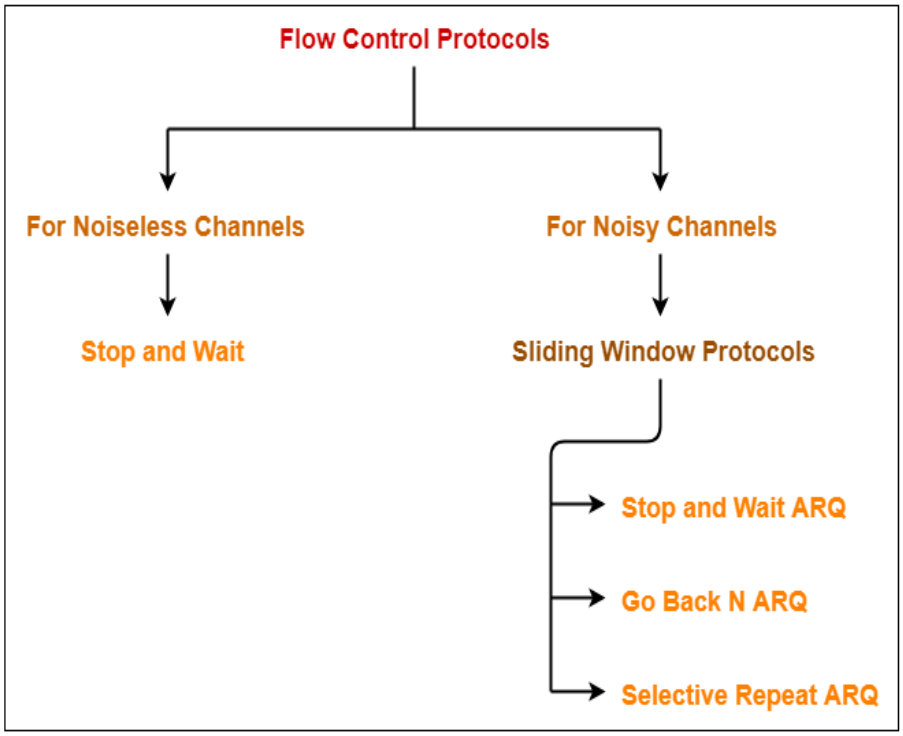
- Nagle's algorithm sends only one segment per round trip time.
- This impacts the latency by introducing a delay.

Note-02:

- Nagle's algorithm and Clark's solution are complementary.
- Both Nagle's solution and Clark's solution can work together.
- The ultimate goal is sender should not send the small segments and receiver should not ask for them.

3. SLIDING WINDOW PROTOCOL

- In computer networking, there are various flow control protocols-



- Sliding window protocols are data link layer and also used in transport layer for reliable and sequential delivery of data frames.
- In this protocol, multiple frames can be sent by a sender at a time before receiving an acknowledgment from the receiver. The term sliding window refers to the imaginary boxes to hold frames. Sliding window method is also known as windowing.
- Sliding window protocol is a flow control protocol.
- It allows the sender to send multiple frames before needing the acknowledgements.
- Sender slides its window on receiving the acknowledgements for the sent frames.
- This allows the sender to send more frames.
- It is called so because it involves sliding of sender's window.

Maximum number of frames that sender can send without acknowledgement = Sender window size

- **Optimal Window Size**

In a sliding window protocol, optimal sender window size = 1 + 2a

Where a= (propagation time/Transmission time)

Working Principle

In these protocols, the sender has a buffer called the sending window and the receiver has buffer called the receiving window.

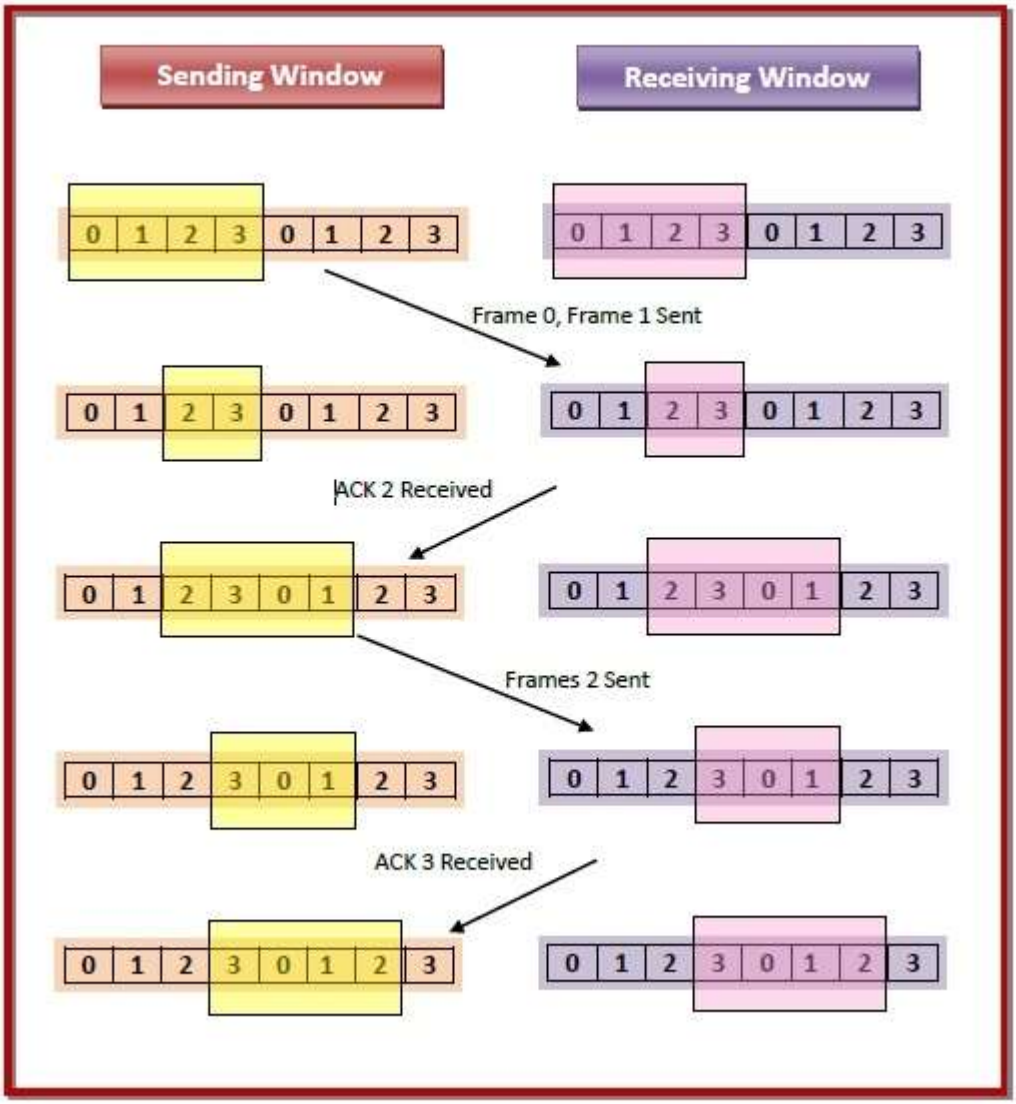
The size of the sending window determines the sequence number of the outbound frames. If the sequence number of the frames is an n-bit field, then the range of sequence numbers that can be assigned is 0 to 2^n-1. Consequently, the size of the sending window is 2^n-1. Thus in order to accommodate a sending window size of 2^n-1, a n-bit sequence number is chosen.

The sequence numbers are numbered as modulo-n. For example, if the sending window size is 4, then the sequence numbers will be 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, and so on. The number of bits in the sequence number is 2 to generate the binary sequence 00, 01, 10, 11.

The size of the receiving window is the maximum number of frames that the receiver can accept at a time. It determines the maximum number of frames that the sender can send before receiving acknowledgment.

Example

Suppose that we have sender window and receiver window each of size 4. So the sequence numbering of both the windows will be 0,1,2,3,0,1,2 and so on. The following diagram shows the positions of the windows after sending the frames and receiving acknowledgments.



Derivation-

We know,

$$\text{Efficiency } (\eta) = \frac{T_t}{T_t + 2 \times T_p}$$

To get 100% efficiency, we must have-

$$n = 1$$

$$T_t / (T_t + 2T_p) = 1$$

$$T_t = T_t + 2T_p$$

Thus,

- To get 100% efficiency, transmission time must be $T_t + 2T_p$ instead of T_t.
- This means sender must send the frames in waiting time too.
- Now, let us find the maximum number of frames that can be sent in time $T_t + 2T_p$.

We have

- In time T_t, sender sends one frame.
- Thus, In time $T_t + 2T_p$, sender can send $(T_t + 2T_p) / T_t$ frames i.e. 1+2a frame .
- Thus, to achieve 100% efficiency, window size of the sender must be 1+2a.

Required Sequence Numbers-

- Each sending frame has to be given a unique sequence number.
- Maximum number of frames that can be sent in a window = 1+2a.
- So, minimum number of sequence numbers required = 1+2a.

To have 1+2a sequence numbers,

Minimum number of bits required in sequence number field = $\lceil \log_2(1+2a) \rceil$

NOTE-

- When minimum number of bits is asked, we take the ceil.
- When maximum number of bits is asked, we take the floor.

Choosing a Window Size-

The size of the sender's window is bounded by-

1. **Receiver's Ability-**
   - Receiver's ability to process the data bounds the sender window size.
   - If receiver can not process the data fast, sender has to slow down and not transmit the frames too fast.

2. Sequence Number Field-

- Number of bits available in the sequence number field also bounds the sender window size.
- If sequence number field contains n bits, then 2^n sequence numbers are possible.
- Thus, maximum number of frames that can be sent in one window = 2^n.

For n bits in sequence number field, Sender Window Size = min (1+2a , 2^n)

Types of Sliding Window Protocols

The Sliding Window ARQ (Automatic Repeat reQuest) protocols are of two categories −

- **Go – Back – *N* ARQ**

  Go – Back – *N* ARQ provides for sending multiple frames before receiving the acknowledgment for the first frame. It uses the concept of sliding window, and so is also called sliding window protocol. The frames are sequentially numbered and a finite number of frames are sent. If the acknowledgment of a frame is not received within the time period, all frames starting from that frame are retransmitted.

- **Selective Repeat ARQ**

  This protocol also provides for sending multiple frames before receiving the acknowledgment for the first frame. However, here only the erroneous or lost frames are retransmitted, while the good frames are received and buffered.

Efficiency-

Efficiency of any flow control protocol may be expressed as-

$$\text{Efficiency } (\eta) = \frac{\text{Number of frames sent in one window}}{\text{Total number of frames that can be sent in one window}}$$

OR

$$\text{Efficiency } (\eta) = \frac{\text{Sender Window Size in the Protocol}}{\text{Optimal Sender Window Size}}$$

OR

$$\text{Efficiency } (\eta) = \frac{\text{Sender Window Size in the Protocol}}{1 + 2a}$$

In **Stop and Wait ARQ**, sender window size = 1.

Thus,

Efficiency of Stop and Wait ARQ = 1 / 1+2a

PRACTICE PROBLEMS BASED ON SLIDING WINDOW PROTOCOL-

Problem-01:

If transmission delay and propagation delay in a sliding window protocol are 1 msec and 49.5 msec respectively, then-

1) What should be the sender window size to get the maximum efficiency?
2) What is the minimum number of bits required in the sequence number field?
3) If only 6 bits are reserved for sequence numbers, then what will be the efficiency?

Solution-

Given-

- Transmission delay = 1 msec
- Propagation delay = 49.5 msec

Part-01:

To get the maximum efficiency, sender window size
= 1 + 2a
= 1 + 2 x (T_p / T_t)
= 1 + 2 x (49.5 msec / 1 msec)
= 1 + 2 x 49.5
= 100

Thus,
For maximum efficiency, sender window size = 100

Part-02:

Minimum number of bits required in the sequence number field
$= \lceil \log_2(1+2a) \rceil$
$= \lceil \log_2(100) \rceil$
$= \lceil 6.8 \rceil$
$= 7$
Thus,
Minimum number of bits required in the sequence number field = 7

Part-03:

If only 6 bits are reserved in the sequence number field, then-

Maximum sequence numbers possible = $2^6 = 64$

Now,
Efficiency
= Sender window size in the protocol / Optimal sender window size
= 64 / 100
= 0.64
= 64%

Problem-02:

If transmission delay and propagation delay in a sliding window protocol are 1 msec and 99.5 msec respectively, then-

1) What should be the sender window size to get the maximum efficiency?
2) What is the minimum number of bits required in the sequence number field?
3) If only 7 bits are reserved for sequence numbers, then what will be the efficiency?

Solution-

Given-

- Transmission delay = 1 msec
- Propagation delay = 99.5 msec

Part-01:

To get the maximum efficiency, sender window size
$= 1 + 2a$
$= 1 + 2 \times (T_p / T_t)$
$= 1 + 2 \times (99.5 \text{ msec} / 1 \text{ msec})$
$= 1 + 2 \times 99.5$
$= 200$
Thus,
For maximum efficiency, sender window size = 200

Part-02:

Minimum number of bits required in the sequence number field
$= \lceil \log_2(1+2a) \rceil$
$= \lceil \log_2(200) \rceil$
$= \lceil 7.64 \rceil$
$= 8$
Thus,
Minimum number of bits required in the sequence number field = 8

Part-03:

If only 6 bits are reserved in the sequence number field, then-
Maximum sequence numbers possible = $2^7 = 128$
Now,
Efficiency
= Sender window size in the protocol / Optimal sender window size
= 128 / 200
= 0.64
= 64%

Problem-03:

A 3000 km long trunk operates at 1.536 Mbps and is used to transmit 64 byte frames and uses sliding window protocol. If the propagation speed is 6 μsec / km, how many bits should the sequence number field be?

Solution-

Given-

- Distance = 3000 km
- Bandwidth = 1.536 Mbps
- Packet size = 64 bytes
- Propagation speed = 6 μsec / km

Calculating Transmission Delay-

Transmission delay (T_t)
= Packet size / Bandwidth
= 64 bytes / 1.536 Mbps
= (64 x 8 bits) / (1.536 x 10^6 bits per sec)
= 333.33 μsec

Calculating Propagation Delay-

For 1 km, propagation delay = 6 μsec
For 3000 km, propagation delay = 3000 x 6 μsec = 18000 μsec

Calculating Value of 'a'-

$a = T_p / T_t$

$a = 18000\ \mu sec / 333.33\ \mu sec$
$a = 54$

Calculating Bits Required in Sequence Number Field-

Bits required in sequence number field

$= \lceil \log_2(1+2a) \rceil$
$= \lceil \log_2(1 + 2 \times 54) \rceil$
$= \lceil \log_2(109) \rceil$
$= \lceil 6.76 \rceil$
$= 7$ bits

Thus,

- Minimum number of bits required in sequence number field = 7
- With 7 bits, number of sequence numbers possible = 128
- We use only (1+2a) = 109 sequence numbers and rest remains unused.

Problem-04:

Compute approximate optimal window size when packet size is 53 bytes, RTT is 60 msec and bottleneck bandwidth is 155 Mbps.

Solution-

Given-

- Packet size = 53 bytes
- RTT = 60 msec
- Bandwidth = 155 Mbps

Calculating Transmission Delay-

Transmission delay (T_t)
= Packet size / Bandwidth
= 53 bytes / 155 Mbps
= (53 x 8 bits) / (155 x 10^6 bits per sec)
= 2.735 μsec

Calculating Propagation Delay-

Propagation delay (T_p)
= Round Trip Time / 2
= 60 msec / 2
= 30 msec

Calculating Value of 'a'-

$a = T_p / T_t$
$a = 30\ msec / 2.735\ \mu sec$

a = 10968.921

Calculating Optimal Window Size-

Optimal window size
= 1 + 2a
= 1 + 2 x 10968.921
= 21938.84

Thus, approximate optimal window size = 21938 frames.

Problem-05:

A sliding window protocol is designed for a 1 Mbps point to point link to the moon which has a one way latency (delay) of 1.25 sec. Assuming that each frame carries 1 KB of data, what is the minimum number of bits needed for the sequence number?

Solution-

Given-

- Bandwidth = 1 Mbps
- Propagation delay (T_p) = 1.25 sec
- Packet size = 1 KB

Calculating Transmission Delay-

Transmission delay (T_t)
= Packet size / Bandwidth
= 1 KB / 1 Mbps
= (2^{10} x 8 bits) / (10^6 bits per sec)
= 8.192 msec

Calculating Value of 'a'-

$a = T_p / T_t$
a = 1.25 sec / 8.192 msec
a = 152.59

Calculating Bits Required in Sequence Number Field-

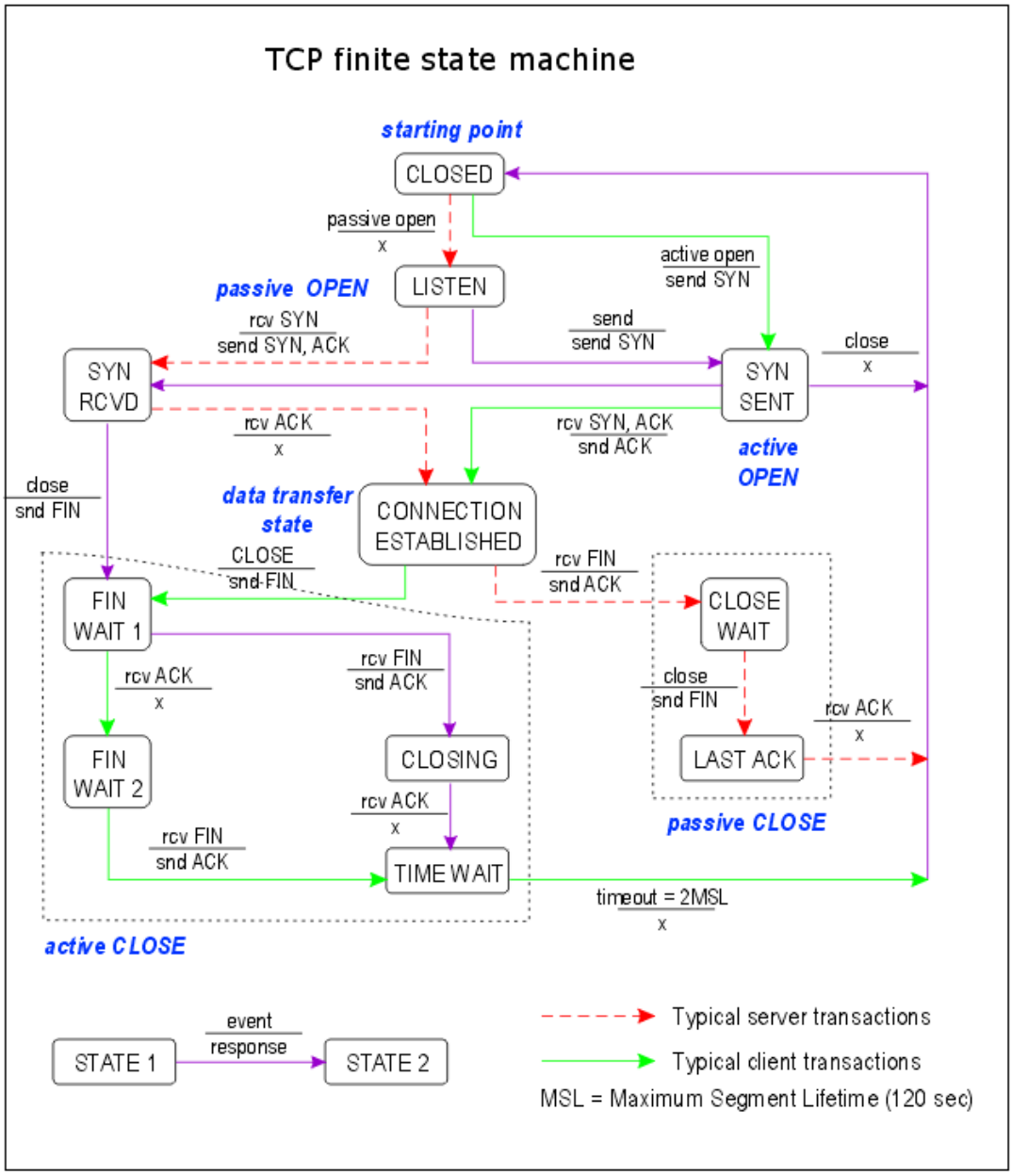
Bits required in sequence number field
= $\lceil \log_2(1+2a) \rceil$
= $\lceil \log_2(1 + 2 \times 152.59) \rceil$
= $\lceil \log_2(306.176) \rceil$
= $\lceil 8.25 \rceil$
= 9 bits

Thus,

- Minimum number of bits required in sequence number field = 9
- With 9 bits, number of sequence numbers possible = 512.
- We use only (1+2a) sequence numbers and rest remains unused.

4. FINITE STATE MACHINE MODEL

- In the case of TCP, the finite state machine can be considered to describe the “life stages” of a connection. Each connection between one TCP device and another begins in a null state where there is no connection, and then proceeds through a series of states until a connection is established. It remains in that state until something occurs to cause the connection to be closed again, at which point it proceeds through another sequence of transitional states and returns to the closed state.
- A Finite State Machine is a logical model for the behavior of a system whose internal state changes with the occurrence of specified events.
- The boxes represent the states of the machine and the arrows represent the transition from one state to another.
- The labels on the arrows represent the event that caused the transition (above the line) and the response of TCP to that event (below the line). An x as response indicates that no action is required or expected.



- ***SYN:*** A *synchronize* message, used to initiate and establish a connection. It is so named since one of its functions is to synchronizes sequence numbers between devices.
- ***FIN:*** A *finish* message, which is a TCP segment with the *FIN* bit set, indicating that a device wants to terminate the connection.
- ***ACK:*** An *acknowledgment*, indicating receipt of a message such as a *SYN* or a *FIN*.