SLIDING WINDOW

The Sliding Window Protocol is a key [computer networking](https://www.geeksforgeeks.org/basics-computer-networking/) technique for controlling the flow of data between two devices. It guarantees that data is sent consistently and effectively, allowing many packets to be sent before requiring an acknowledgment for the first, maximizing the use of available bandwidth.

**Terminologies Related to Sliding Window Protocol**

**Transmission Delay (Tt)** – Time to transmit the packet from the host to the outgoing link. If B is the [Bandwidth](https://www.geeksforgeeks.org/what-is-bandwidth-definition-working-importance-uses/) of the link and D is the Data Size to transmit

Tt = D/B

**Propagation Delay (Tp)** – It is the time taken by the first bit transferred by the host onto the outgoing link to reach the destination. It depends on the distance d and the wave propagation speed s (depends on the characteristics of the medium).

Tp = d/s

**Efficiency** – It is defined as the ratio of total useful time to the total cycle time of a packet. For stop and wait protocol,

Total time(TT) = Tt(data) + Tp(data) +   
 Tt(acknowledgement) + Tp(acknowledgement)  
 = Tt(data) + Tp(data) + Tp(acknowledgement)  
 = Tt + 2\*Tp

Since acknowledgements are very less in size, their transmission delay can be neglected.

Efficiency = Useful Time / Total Cycle Time   
 = Tt/(Tt + 2\*Tp) (For Stop and Wait)  
 = 1/(1+2a) [ Using a = Tp/Tt ]

**Effective Bandwidth(EB) or Throughput** – Number of bits sent per second.

EB = Data Size(D) / Total Cycle time(Tt + 2\*Tp)  
Multiplying and dividing by Bandwidth (B),  
 = (1/(1+2a)) \* B [ Using a = Tp/Tt ]  
 = Efficiency \* Bandwidth

**Capacity of link** – If a channel is [Full Duplex](https://www.geeksforgeeks.org/transmission-modes-computer-networks/), then bits can be transferred in both the directions and without any collisions. Number of bits a channel/Link can hold at maximum is its capacity.

Capacity = Bandwidth(B) \* Propagation(Tp)  
   
 For Full Duplex channels,

## 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 2𝑛−1. Consequently, the size of the sending window is 2𝑛−1. Thus in order to accommodate a sending window size of 2𝑛−1, a

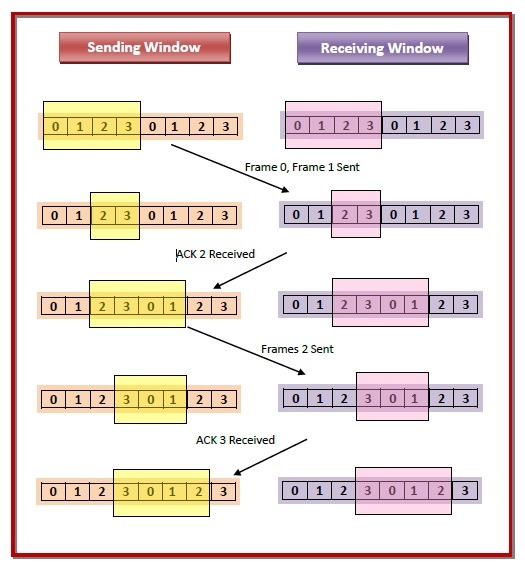
n-bit sequence number is chosen.

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.



Capacity = 2\*Bandwidth(B) \* Propagation(Tp)

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <time.h>

#define WINDOW\_SIZE 4

#define TIMEOUT 2

typedef struct {

int sequenceNumber;

int ack;

char data[20];

} Packet;

void simulateTransmission(Packet packets[], int numPackets) {

srand(time(0));

int currentIndex = 0;

int nextSequenceNumber = 0;

while (currentIndex < numPackets) {

if (nextSequenceNumber < currentIndex + WINDOW\_SIZE) {

printf("Sending packet with sequence number %d\n", nextSequenceNumber);

packets[nextSequenceNumber].sequenceNumber = nextSequenceNumber;

if (nextSequenceNumber == numPackets - 1) {

packets[nextSequenceNumber].ack = 1; // Set ack flag for last packet

} else {

packets[nextSequenceNumber].ack = 0;

}

// Simulating packet loss

if (rand() % 4 != 0) {

// Simulating packet corruption

if (rand() % 3 != 0) {

strcpy(packets[nextSequenceNumber].data, "DATA");

} else {

strcpy(packets[nextSequenceNumber].data, "CORRUPT");

}

} else {

strcpy(packets[nextSequenceNumber].data, "LOST");

}

nextSequenceNumber++;

}

int ackPacketNumber = -1;

printf("Acknowledgment received for packets: ");

for (int i = currentIndex; i < nextSequenceNumber; i++) {

if (packets[i].ack == 1) {

ackPacketNumber = i;

printf("%d ", ackPacketNumber);

}

}

printf("\n");

if (ackPacketNumber != -1) {

currentIndex = ackPacketNumber + 1;

}

}

}

int main() {

int numPackets;

printf("Enter the number of packets to be transmitted: ");

scanf("%d", &numPackets);

Packet packets[numPackets];

printf("Simulating transmission...\n");

simulateTransmission(packets, numPackets);

printf("Transmission complete.\n");

return 0;

}