

Simplex Stop-and-Wait Protocol

One solution is to build the receiver to be powerful enough to process a continuous stream of back-to-back frames (or, equivalently, define the link layer to be slow enough that the receiver can keep up). It must have sufficient buffering and processing abilities to run at the line rate and must be able to pass the frames that are received to the network layer quickly enough. However, this is a worst-case solution.

A more general solution to this problem is to have the receiver provide feedback to the sender. After having passed a packet to its network layer, the receiver sends a little dummy frame back to the sender which, in effect, gives the sender permission to transmit the next frame. After having sent a frame, the sender is required by the protocol to bide its time until the little dummy (i.e., acknowledgment) frame arrives.

Protocols in which the sender sends one frame and then waits for an acknowledgment before proceeding are called **stop-and-wait**.

Although data traffic in this example is simplex, going only from the sender to the receiver, frames do travel in both directions. Consequently, the communication channel between the two data link layers needs to be capable of bidirectional information transfer. However, this protocol entails a strict alternation of flow : first the sender sends a frame, then the receiver sends a frame, then the sender sends another frame, then the receiver sends another one, and so on. A half-duplex physical channel would suffice here.

Code of Stop-and-Wait Protocol

```
#include<iostream>
#include<cstdlib>
#define Max_No_Frames 15
using namespace std;

struct Frame
{
    int index;
    int data;
    int Fault;
};          // Frame Structure

Frame frame;
bool MODE;          // Mode 0 denotes SENDER mode and 1 denotes
RECEIVER mode
int Connection=1,Time_Out=3,i=0;
int
Data_Set[Max_No_Frames+1]={54,46,67,61,546,44,93,53,65,78,94,24,200,332,6
5,10};

void SEND_from_network_layer(int *buffer)
{
    // SEND_ in function name denotes sender side
    *buffer= Data_Set[i++];          // Data is fetched from network layer
}

void REC_to_network_layer(int *buffer)
{
    cout<<"Frame Data "<<*buffer<<" is recieved. And Acknowledgment is Sent.
[Receiver]\n";
    if(i>Max_No_Frames)          //if all frames received then disconnect
        Connection=0, cout<<"Connection Closed.\n";
}

void REC_from_physical_layer(Frame *buffer)
{

```

```

*buffer = frame; // receiver takes data from physical layer
}

void SEND_to_physical_layer(Frame *s)
{
s->Fault = rand()%8; //Fault=0 means error and non zero means no error
frame = *s;          //probability of error = 1/8
}

typedef enum
{
    FRAME_SIGNAL, // frame is there
    ERROR,         // frame is damaged when frame.fault=0
    TIMEOUT,       // acknowledgement not received
    NO_EVENT       // nothing happens
}Event_Type;

void wait_for_event_sender(Event_Type * e)
{
static int Counter=0; // to count the timeout
if(MODE==0) //when sender mode
{
    Counter++;
    if(Counter==Time_Out) //TIMEOUT happened
    {
        *e = TIMEOUT, Counter=0;;
        cout<<"Acknowledgment is not received, TIMEOUT. [Sender]\n";
        return;
    }

    if(!frame.Fault) // when frame is damaged ERROR
        *e = ERROR;
    else
        *e = FRAME_SIGNAL, Counter=0; // otherwise frame
}
}

void wait_for_event_receiver(Event_Type * e)
{
if(MODE==1) //when receiver mode

```

```

{
    if(!frame.Fault)
        *e = ERROR;
    else
        *e = FRAME_SIGNAL;
}
}
void SENDER()
{
    static int Frame_No_to_Send=0; //next frame no to send
    static Frame s;
    int buffer;
    Event_Type event;           //event holder
    static int flag=0;

    if(flag==0)// only for first frame (since we started from sender mode)
    {
        SEND_from_network_layer(&buffer);           //data fetching from network
layer(sender mode)
        s.data = buffer;
        s.index = Frame_No_to_Send;    // next frame index
        cout<<"Frame No "<<s.index<<" and Data "<<s.data<<" is sent. [Sender]\n";
        MODE=1;           // receiver mode
        SEND_to_physical_layer(&s);
        flag = 1;
    }
    wait_for_event_sender(&event);

    if(MODE==0)    // when sender mode
    {
        if(event==FRAME_SIGNAL)
        {
            SEND_from_network_layer(&buffer);           //data fetching from network
layer(sender mode)
            s.index = ++Frame_No_to_Send;
            s.data = buffer;
            cout<<"Frame No "<<s.index<<" and Data "<<s.data<<" is sent. [Sender]\n";
            MODE=1;
            SEND_to_physical_layer(&s); //data sent to physical layer

```

```

    }

    if(event==TIMEOUT)
    {
        cout<<"Resending the Frame. [Sender]\n";
        MODE=1;
        SEND_to_physical_layer(&s); //data sent to physical layer
    }
}

void RECEIVER()
{
    static int frame_in=0;
    Frame A,B;
    Event_Type E;
    wait_for_event_receiver(&E);

    if(MODE==1)
    {
        if(E==FRAME_SIGNAL)
        {
            REC_from_physical_layer(&A);    // data received from physical layer
            (receiver mode)
            if(A.index==frame_in)
            {
                REC_to_network_layer(&A.data); // data sent to network layer (receiver side)
                ++frame_in;
            }
            else
            {
                cout<<"Reacknowledgement is sent. [Receiver]\n";
                MODE=0, SEND_to_physical_layer(&B); // Acknowledgment is received send
                next frame
            }
            if(E==ERROR)
            {
                cout<<"Frame is Damaged. [Receiver]\n";
                MODE=0;    //frame is damaged Sender should send it again.
            }

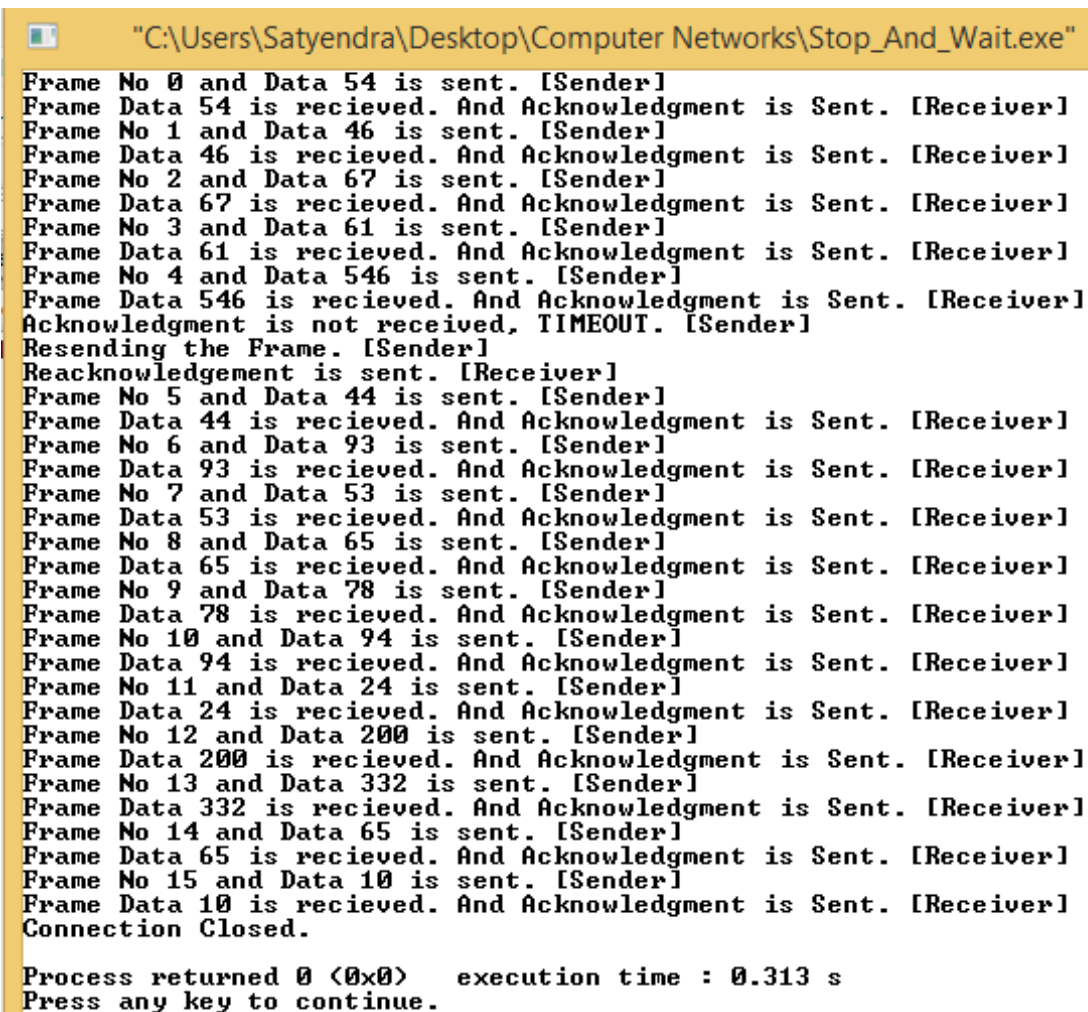
```

```

    }
}
int main()
{
    while(Connection)
    {
        SENDER();
        RECEIVER();
    }
}

```

RESULT :-



```

"C:\Users\Satyendra\Desktop\Computer Networks\Stop_And_Wait.exe"
Frame No 0 and Data 54 is sent. [Sender]
Frame Data 54 is recieved. And Acknowledgment is Sent. [Receiver]
Frame No 1 and Data 46 is sent. [Sender]
Frame Data 46 is recieved. And Acknowledgment is Sent. [Receiver]
Frame No 2 and Data 67 is sent. [Sender]
Frame Data 67 is recieved. And Acknowledgment is Sent. [Receiver]
Frame No 3 and Data 61 is sent. [Sender]
Frame Data 61 is recieved. And Acknowledgment is Sent. [Receiver]
Frame No 4 and Data 546 is sent. [Sender]
Frame Data 546 is recieved. And Acknowledgment is Sent. [Receiver]
Acknowledgment is not received, TIMEOUT. [Sender]
Resending the Frame. [Sender]
Reacknowledgement is sent. [Receiver]
Frame No 5 and Data 44 is sent. [Sender]
Frame Data 44 is recieved. And Acknowledgment is Sent. [Receiver]
Frame No 6 and Data 93 is sent. [Sender]
Frame Data 93 is recieved. And Acknowledgment is Sent. [Receiver]
Frame No 7 and Data 53 is sent. [Sender]
Frame Data 53 is recieved. And Acknowledgment is Sent. [Receiver]
Frame No 8 and Data 65 is sent. [Sender]
Frame Data 65 is recieved. And Acknowledgment is Sent. [Receiver]
Frame No 9 and Data 78 is sent. [Sender]
Frame Data 78 is recieved. And Acknowledgment is Sent. [Receiver]
Frame No 10 and Data 94 is sent. [Sender]
Frame Data 94 is recieved. And Acknowledgment is Sent. [Receiver]
Frame No 11 and Data 24 is sent. [Sender]
Frame Data 24 is recieved. And Acknowledgment is Sent. [Receiver]
Frame No 12 and Data 200 is sent. [Sender]
Frame Data 200 is recieved. And Acknowledgment is Sent. [Receiver]
Frame No 13 and Data 332 is sent. [Sender]
Frame Data 332 is recieved. And Acknowledgment is Sent. [Receiver]
Frame No 14 and Data 65 is sent. [Sender]
Frame Data 65 is recieved. And Acknowledgment is Sent. [Receiver]
Frame No 15 and Data 10 is sent. [Sender]
Frame Data 10 is recieved. And Acknowledgment is Sent. [Receiver]
Connection Closed.

Process returned 0 (0x0)   execution time : 0.313 s
Press any key to continue.

```