NETWORK LAB REPORT

NAME : Dibyendu Mukhopadhyay
CLASS : BCSE-III
ROLL NO. : 001710501077
GROUP : A3
ASSIGNMENT NO. : 2

PROBLEM STATEMENT:

Implement three data link layer protocols, Stop and Wait, Go Back N Sliding Window and Selective Repeat Sliding Window for flow control.

SUBMISSION DUE : 10TH FEBRUARY,2020

REPORT SUBMITTED : 24TH FEBRUARY,2020

This report has three sections: Section 1 contains Stop and Wait protocol, section 2 contains Go Back n protocol and section 3 contains Selective Repeat Sliding windows protocol.

SECTION 1: Stop and wait arq

- ## Design

The protocol is designed using two files- `snw_sender.py` and `snw_receiver.py`.

a. **`snw_sender.py`**: This file handles the sender side. It takes data input from file, converts the data into number of codewords of length of given frame size using the VRC algorithm. Then the program sends the frames one by one to the receiver. While sending the frame, it also introduces the error to the frame in a few cases. It also receives the acknowledgements from the receiver side and in case of error it resends the frame.
b. **`snw_receiver.py`**: This file handles the receiver side. It accepts the frames sent from the sender side, checks for error (if any), and if it is the expected frame, it sends the acknowledgment to the sender end.

The design of the implementation is shown in the diagram (1)

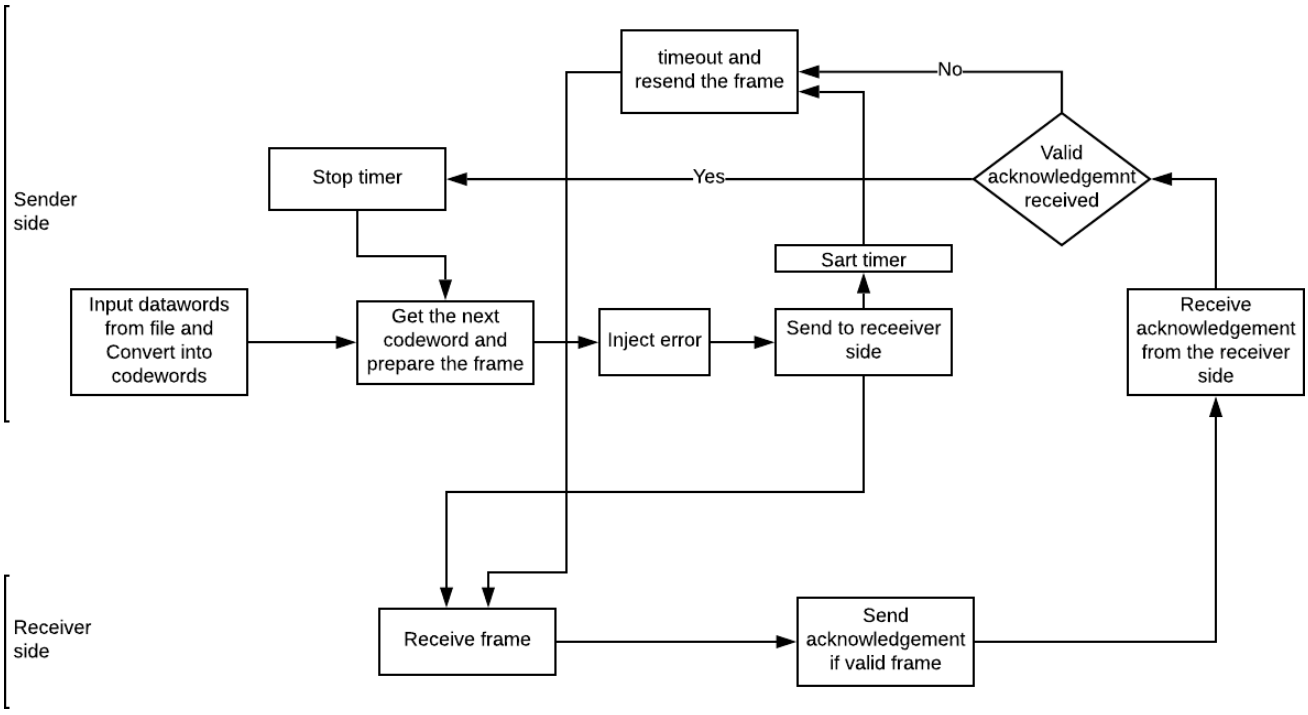


Figure 1: design of the implementation of stop and wait arq algorithm

Some important parameters used in the implementation are described below:

a. **Frame size**: 16 bits
b. **Dataword size**: 8 bits

c. **Frame format**: sequence number of frame followed by the stuffed zero and the codeword at the end. The 8 bit data word is converted into 9 bit codeword using the vrc algorithm. Then 6 0's are stuffed before it and finally the sequence number (0 or 1) is added before it to generate the 16 bit frame.
d. **Error detection algorithm used**: vrc or simple pairity checker.
e. **Acknowledgement format**: 1 bit acknowledgement specifying the next expected frame.
f. **Input file format**: bit stream of 0's and 1's.

- ## Implementation

In this problem statement, the entire program is implemented in Python3.
The detailed method description is written below with a suitable code snippets along with comments for better understanding code overview**.**

- *snw_sender.py*

  These are the global variables which is declared globally.

```
framesize=16 #size of frame
dwsize=8 #data word size
x_time=0 #time passed
codeword=[]
index=0
s_n=0 #send next frame
```

  Socket programming are implemented globally also which helps to interact between sender and receiver. Here, the codewords are send via port.

```
s_send = socket.socket()

print( "Socket successfully created")

port_send = 12345
port_receive = 12346

s_send.bind(('', port_send))
print( "socket binded to %s" %(port_send) )

s_send.listen(5)
print( "socket is listening\n")

cansend=True
generate_codewords()
count=1
```

Function Definitions:

- This method generates the frame from the 9 bit codeword.

```
#Generates the frame from the 9 bit codeword
def gen_frame(codeword):
    new_cw=str(s_n)+'0'*(framesize-len(codeword)-1)+codeword
    return new_cw
```

- This method adds the a parity bit after the 8 bit dataword to generate the 9bit codeword

```
#Adds the a parity bit after the 8bit dataword to generate the 9bit codeword
def make_pairity(num):
    count=0
    for s in num:
        if s=='1':
            count=count+1
    if count%2==0:
        num=num+'0'
    else:
        num=num+'1'
    return num
```

- This function reads the input file and generates datawords of length 8 and then return it .

```
#Reads the input file and generates datawords of length 8
def generate_codewords():
    dataword=[]
    with open("data1.txt","rt") as fp:
        for line in fp:
            newdataword=[''.join(x) for x in zip(*[list(line[z::dwsize])
             for z in range(dwsize)])]

    dataword.extend(newdataword)

    for dw in dataword:
        codeword.append(make_pairity(dw))
```

- This module checks if the sender is prepare for sending the new frames.

```
#checks if the sender side is prepared for sending a new frame.
def request_to_send():
    if(index<len(codeword)):
        return True

    return False
```

- This method actually converts the frames from string to bytearray.

```
#Converts the frame from string to bytearray
def makeframe():
    global index
    new_s=gen_frame(codeword[index])
    index=index+1
    fr=bytearray()
    fr.extend(map(ord,new_s))

    return fr
```

- This module introduces the error in the desired frame.

```
#Introduces error in the desired frame.
def introduce_error(frame):
    frame1=frame
    if count%2==0:
        if frame1[len(frame1)-1]==49:
            frame1[len(frame1)-1]=48
        else:
            frame1[len(frame1)-1]=49
    return frame1
```

- This method helps to resend the frame in case of timeout in a new thread.

```
#Resends the frame in case of timeout in a new thread.
def timeout():
    print("timeout !!")
    x_time=0
    try:
        global count,frame
        c,addr=s_send.accept()
        print("Got connection from",addr)
        # print(introduce_error(frame))
        print("Resending frame "+str(count%2))

        c.send(introduce_error(frame))
        print(frame.decode("utf-8")+" Sent")
        c.close()
        print("**********************************\n")
        # timer=threading.Timer(5.0,timeout)
        # timer.start()
    except InterruptedError as err:
        print(err)
#
```

- **Main Thread :**

The **while()** loop runs infinitely to run the send and receive operations whenever frame is ready and acknowledgment is received. When the sender is ready to send and there is a request to send, the sending socket checks for a connection with receiver port. If the connection is found sender creates a new frame from the dataword and introduces error (if any) to it. Then the sender sends the frame to the receiver port using the sending socket and the timer starts and `cansend` is turned **`false`**.

The receiver socket searches for a connection from the sending socket in the receiver side. If there is any connection, the socket accepts the acknowledgment.

If the acknowledgement is not corrupted and it is same as the next ready frame to be sent, then the timer is stopped and `cansend` is turned `true`.

```
while True:

    if (request_to_send() and cansend==True):
        try:
            c, addr = s_send.accept()
            print('Got connection from', addr )
        except InterruptedError as err:
            print(err)
        # getdata();
        frame=makeframe()
        count=count+1

        frame_e=introduce_error(frame)
        print("Sending frame "+str(count%2))
        print(frame_e.decode("utf-8")+" Sent")
        c.send(frame_e)
        # Close the connection with the client
        c.close()
        print("*********************************\n")
        x_time=1
        timer=threading.Timer(5.0,timeout)
        timer.start()
        s_n=(s_n+1)%2
        cansend= False

    try:
        s_receive = socket.socket()
        s_receive.connect(('127.0.0.1', port_receive))
        ackno=s_receive.recv(1024)
        print("Acknowledgemnt for "+ackno.decode("utf-8")+" received")
        print("*********************************\n")
```

```
        s_receive.close()
        if(int(ackno)==s_n):
            timer.cancel()
            frame=bytearray()
            cansend=True
    except ConnectionRefusedError as err:
        print(err)
```

- *snw_receiver.py*

  Global variables declaration and socket modules

```
framesize=16
dwsize=8
s_send = socket.socket()

print( "Socket successfully created")

port_send = 12346
port_receive = 12345

s_send.bind(('', port_send))
print( "socket binded to %s" %(port_send) )

s_send.listen(5)
print( "socket is listening\n")

r_n=0
```

- This method checks the parity of the given string. If the odd number of 1’s, it returns False else True.

```
#checks the parity of the given string.
#If odd number of 1’s it returns False else True
def check_pairity(num):
    count=0
    for s in num:
        if s=='1':
            count=count+1
    if count%2==0:
        return True
    return False
```

- This method checks whether there is an error in an incoming frame.

```
#checks if there is error in the incoming frame.
def corrupted(frame):
    s1=frame.decode('ASCII')
    s2=s1[framesize-dwsize-1:]
    if check_pairity(s2):
        return False
    return True
```

- This method creates the ACK for the next expected frame.

```
#creates acknowledgment for the next expected frame.
def acknowledgement(r_n):
    ack=str(r_n)
    b_ack=bytearray()
    b_ack.extend(map(ord,ack))
    return b_ack
```

- This function defines the sequence number of the incoming frame.

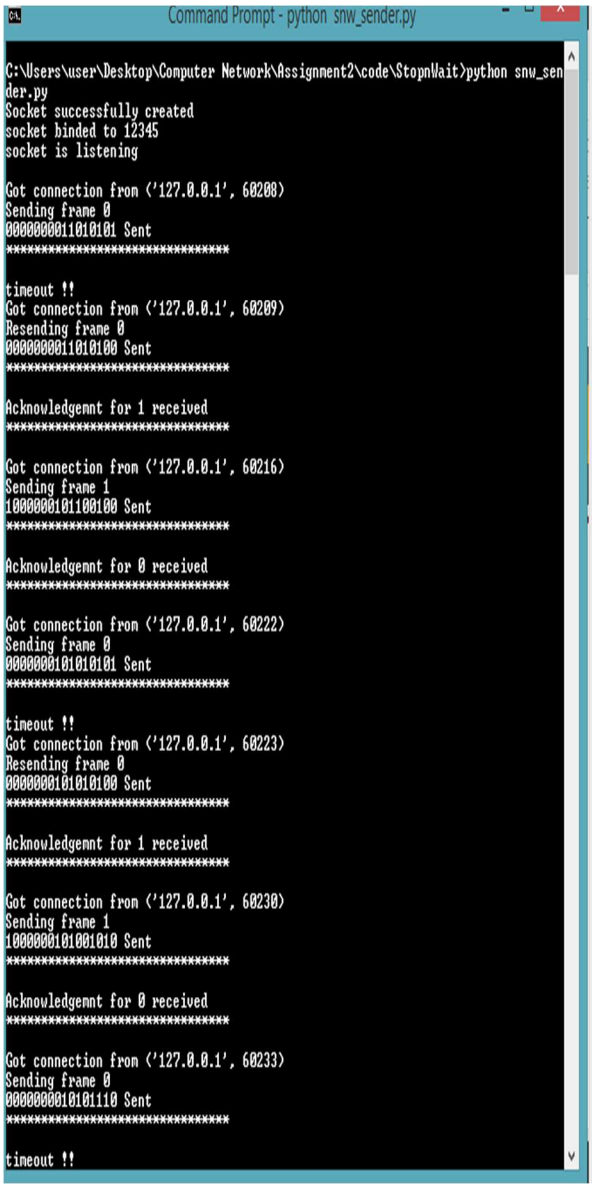
```
#returns the sequence number of the incoming frame.
def seqno(frame):
    if frame[0]==49:
        return 1
    else:
        return 0
```

- **Main Thread :**
  The receive socket in the receiver side searches for connection from port in the sender side. If a connection is found, it receives the frame, checks if corrupted and if the sequence number is same as expected. Then a sender socket is created that sends the acknowledgement to the sender side.

```
while True:
    try:
        s_receive = socket.socket()
        s_receive.connect(('127.0.0.1', port_receive))
        frame=s_receive.recv(1024)
        s_receive.close()
        print("****")
        print("Received Frame = "+frame.decode("utf-8"))
        if corrupted(frame)==False:
            if seqno(frame)==r_n:
                try:
                    c, addr = s_send.accept()
                    print('Got connection from', addr )
                    r_n=(r_n+1)%2
                    time.sleep(3)
                    c.send(acknowledgement(r_n))
                    print("acknowledgement sent for "+acknowledgement(r_n).decode("utf-8")+"\n")
                    c.close()
                except InterruptedError as err:
                    print(err)
            else:
                print("Expected frame = "+str(r_n)+", Arrived Frame = "+str(seqno(frame))+"\n")
            # extract()
            # deliver()
        else:
            print("Frame Corrupted\n")

    except ConnectionRefusedError as err:
        print(err)
    except OSError as err2:
        pass
```

- **OUTPUT**



- **Results**:
  The performance of the above algorithm is measured in terms of throughput. The entire data file consists of 20 datawords. So, for sending 20 frames in average 40 attempts were taken considering insertion of random errors. Considering timeout after 5 seconds average propagation delay is 2.5 seconds.

- **Analysis**:
  The program may have some possible bugs due to the lack of randomness in injecting the error. Again, the program is implemented for one sender and receiver that can be extended up to multiple senders and receivers. The last few bits from the input file is discarded to make the dataword size same. This bug can be overcome by padding of 0's or 1's.

- **Comment**:
  The assignment was very helpful in better understanding of the implementation of one of the most popular flow control algorithms in noisy channel. It's level can be rated as moderately high.

SECTION 2 : Go back n arq

- **Design**

The protocol is designed using two files- `gbn_sender.py` and `gbn_receiver.py`.

a. **`gbn_sender.py`**: This file handles the sender side. It takes data input from file, converts the data into number of codewords of length of given frame size using the vrc algorithm. Then the program sends the frames one by one to the receiver. While sending the frame it also introduces error to the frame in a few cases. It also receives the acknowledgements from the receiver side and in case of error it resends the frame.
b. **`gbn_receiver.py`**: This file handles the receiver side. It accepts the frames sent from the sender side, checks for error (if any), and if it is the expected frame, it sends the acknowledgment to the sender end.

The design of the implementation is shown in the diagram (2)

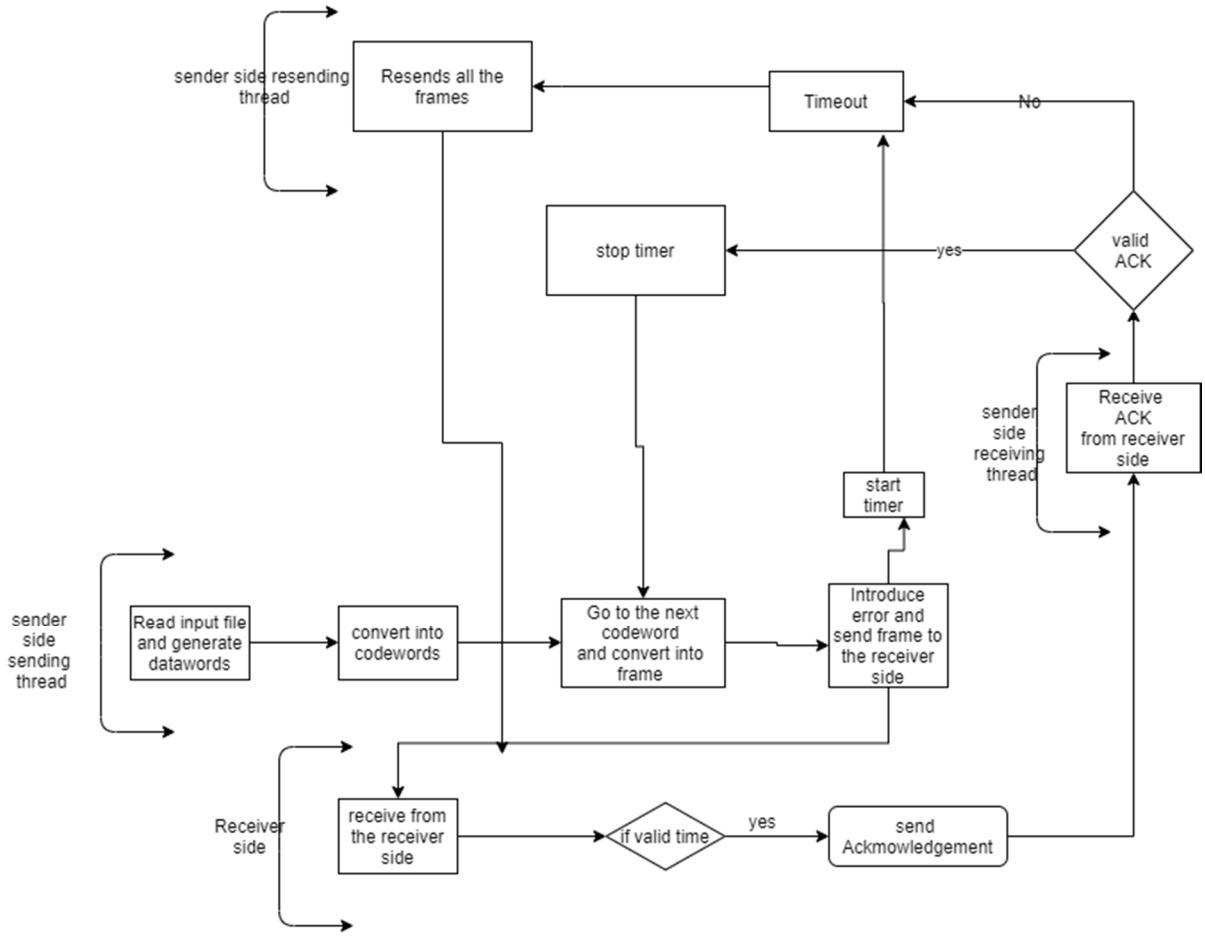


Figure 2: design of the implementation of go back n arq algorithm

Some important parameters used in the implementation are described below:

a. **Frame size**: 16 bits
b. **Dataword size**: 8 bits
c. **Frame format**: sequence number of frame following the stuffed zeros and the codeword at the end. The 8 bit data word is converted into 9 bit codeword usong vrc algorithm. Then the sequence number of the frame is added before it and finally zeros are stuffed to make the frame size 16.
d. **Error detection algorithm used**: vrc or simple pairity checker.
e. **Acknowledgement format**: 1 bit acknowledgement specifying the next expected frame.
f. **Input file format**: bit stream of 0's and 1's.
g. **Window size**: in this case m=3. Hence window size $s\_w = 2^m - 1 = 7$

- ## Implementation

The algorithm has been implemented using python3. The details are given below.

- *gbn_sender.py*

  Global variables and modules

```
framesize=16
dwsize=8
x_time=0
codeword=[]
index=0

m=3
s_n=0
s_w=2**m-1
s_f=0
```

  Socket functionality

```
lock=threading.Lock()

window=dict()
s_send = socket.socket()

print( "Socket successfully created")

port_send = 12345
port_receive = 12346

s_send.bind(('', port_send))
print( "socket binded to %s" %(port_send) )

s_send.listen(5)
print( "socket is listening")

cansend=True
```

- This method generates the frame from the 9 bit codeword.

```
#Generates the frame from the 9 bit codeword following the method described above
def gen_frame(codeword):
    new_cw=str(s_n)+'0'*(framesize-len(codeword)-1)+codeword
    return new_cw
```

- This function adds the parity bit after the 8 bit data word to generate 9 bit codeword.

```
#Adds the a parity bit after the 8bit dataword to generate the 9bit codeword
def make_pairity(num):
    count=0
    for s in num:
        if s=='1':
            count=count+1
    if count%2==0:
        num=num+'0'
    else:
        num=num+'1'
    return num
```

- This function reads the input data file and generates the dataword of length 8.

```
#Reads the input file and generates datawords of length 8
def generate_codewords():
    dataword=[]
    with open("data1.txt","rt") as fp:
        for line in fp:
            newdataword=[''.join(x) for x in zip(*[list(line[z::dwsize]) for z in range(dwsize)])]

    dataword.extend(newdataword)

    for dw in dataword:
        codeword.append(make_pairity(dw))
```

- It converts the frame from string to bytearray.

```
#Converts the frame from string to bytearray
def makeframe():
    global index
    new_s=gen_frame(codeword[index])
    index=index+1
    fr=bytearray()
    fr.extend(map(ord,new_s))

    return fr
```

- This module introduces the error in the desired frame.

```
#Introduces error in the desired frame.
def introduce_error(frame):
    frame1=frame
    if count%2==0:
        if frame1[len(frame1)-1]==49:
            frame1[len(frame1)-1]=48
        else:
            frame1[len(frame1)-1]=49
    return frame1
```

- Resends all the frames from s_f to s_n in case of timeout in a new thread.

```
#Resends all the frames from s_f to s_n in case of timeout in a new thread.
def timeout():
    print("timeout")
    x_time=0
    try:
        temp=s_f
        while temp!=s_n:
            c,addr=s_send.accept()
            print("Got connection from",addr)
            # frame_e=introduce_error(frame)
            if temp==2:
                window[str(temp)][len(window[str(temp)])-1]=48

            print("[Resending "+str(temp)+" ]")
            print(window[str(temp)].decode("utf-8")+" Sent")
            c.send(window[str(temp)])

            print("*****************************\n")
            time.sleep(2)
            c.close()
            temp=(temp+1)%(2**m)

    except InterruptedError as err:
        print(err)
```

- **Threading class**

➔ **`class sending_thread(threading.Thread):`**

This class handles the sending of the data frames from the sender side to the receiver side. The **`while()`** loop runs infinitely to run the send operation whenever frames are ready and acknowledgment is received. When the sender is ready to send and there is a request to send, the sending socket checks for a connection with receiver port. If the connection is found sender creates a new frame from the dataword and introduces error (if any) to it. Then the sender sends the frame to the receiver port using the sending socket and the timer starts and `cansend` is turned `false`.

```
class sending_thread(threading.Thread):
    def _init_(self):
        print("Thread for sending frames")

    def run(self):
        global s_n,s_f,s_w,cansend,x_time,window,frame,timer
        while True:
            # print(s_n,s_f,s_w)
            if (s_n>=s_f and s_n-s_f<s_w) or (s_n<s_f):
            # if s_n-s_f<s_w:

                try:
                    c, addr = s_send.accept()
                    print('Got connection from', addr )
                except InterruptedError as err:
                    print(err)
                # getdata();
                frame=makeframe()
                window[str(s_n)]=frame
                # print(window.keys())

                frame_e=introduce_error(frame)
                print("[Sending "+str(s_n)+" ]")
                print(window[str(s_n)].decode("utf-8")+" sent")
                c.send(frame_e)

                print("**************************\n")
                time.sleep(2)

                c.close()
```

➔ **class receiver_thread(threading.Thread)**

This class handles the receiving of the acknowledgement from the receiver side. The receiver socket searches for a connection from the sending socket in the receiver side. If there is any connection, the socket accepts the acknowledgment. If the acknowledgement is not corrupted and it is same or greater than `s_f`, then the timer is stopped and `cansend` is turned `true`. Again, all the frames from `s_f` to the acknowledgement number is purged.

```
class receiver_thread(threading.Thread):
    def _init_(self):
        print("Thread for receiving Acknowledgements")

    def run(self):
        global s_n,s_f,s_w,cansend,x_time,window,frame,timer
        while True:
            try:
                s_receive = socket.socket()
                s_receive.connect(('127.0.0.1', port_receive))
                ackno=s_receive.recv(1024)
                print("acknowledgement received for frame [ "+ackno.decode("utf-8")+" ]")
                s_receive.close()
                if (s_n>=s_f and int(ackno)<=s_n and int(ackno)>s_f) or (s_n<s_f and int(ackno)>s_f
                    print("no acknowledgement error\n")
                    lock.acquire()
                    timer.cancel()
                    x_time=0
                    lock.release()
                    while (s_n>=s_f and s_f<int(ackno)) or (s_n<s_f and s_n>=int(ackno)):
                        x=window.pop(str(s_f))
                        s_f=(s_f+1)%(2**m)

                    # print(window.keys())

                    frame=bytearray()
                    cansend=True
            except ConnectionRefusedError as err:
                pass
            except ConnectionResetError as err2:
                pass
```

- **Main Thread**

```
thread1=sending_thread()
thread2=receiver_thread()

thread1.start()
thread2.start()
```

- *gbn_receiver.py*
  Global variables and modules

```
framesize=16
dwsize=8
m=3
s_send = socket.socket()

print( "Socket successfully created")

port_send = 12346
port_receive = 12345

s_send.bind(('', port_send))
print( "socket binded to %s" %(port_send) )

s_send.listen(5)
print( "socket is listening")

r_n=0
```

- This method checks the parity of the given string. If the odd number of 1’s, it returns False else True.

```
#checks the parity of the given string.
#If odd number of 1’s it returns False else True
def check_pairity(num):
    count=0
    for s in num:
        if s=='1':
            count=count+1
    if count%2==0:
        return True
    return False
```

- This method checks whether there is an error in an incoming frame.

```
#checks if there is error in the incoming frame.
def corrupted(frame):
    s1=frame.decode('ASCII')
    s2=s1[framesize-dwsize-1:]
    if check_pairity(s2):
        return False
    return True
```

- This method creates the ACK for the next expected frame.

```
#creates acknowledgment for the next expected frame.
def acknowledgement(r_n):
    ack=str(r_n)
    b_ack=bytearray()
    b_ack.extend(map(ord,ack))
    return b_ack
```

- This function defines the sequence number of the incoming frame.

```
#returns the sequence number of the incoming frame.
def seqno(frame):
    if frame[0]==49:
        return 1
    else:
        return 0
```

- **Main Thread**

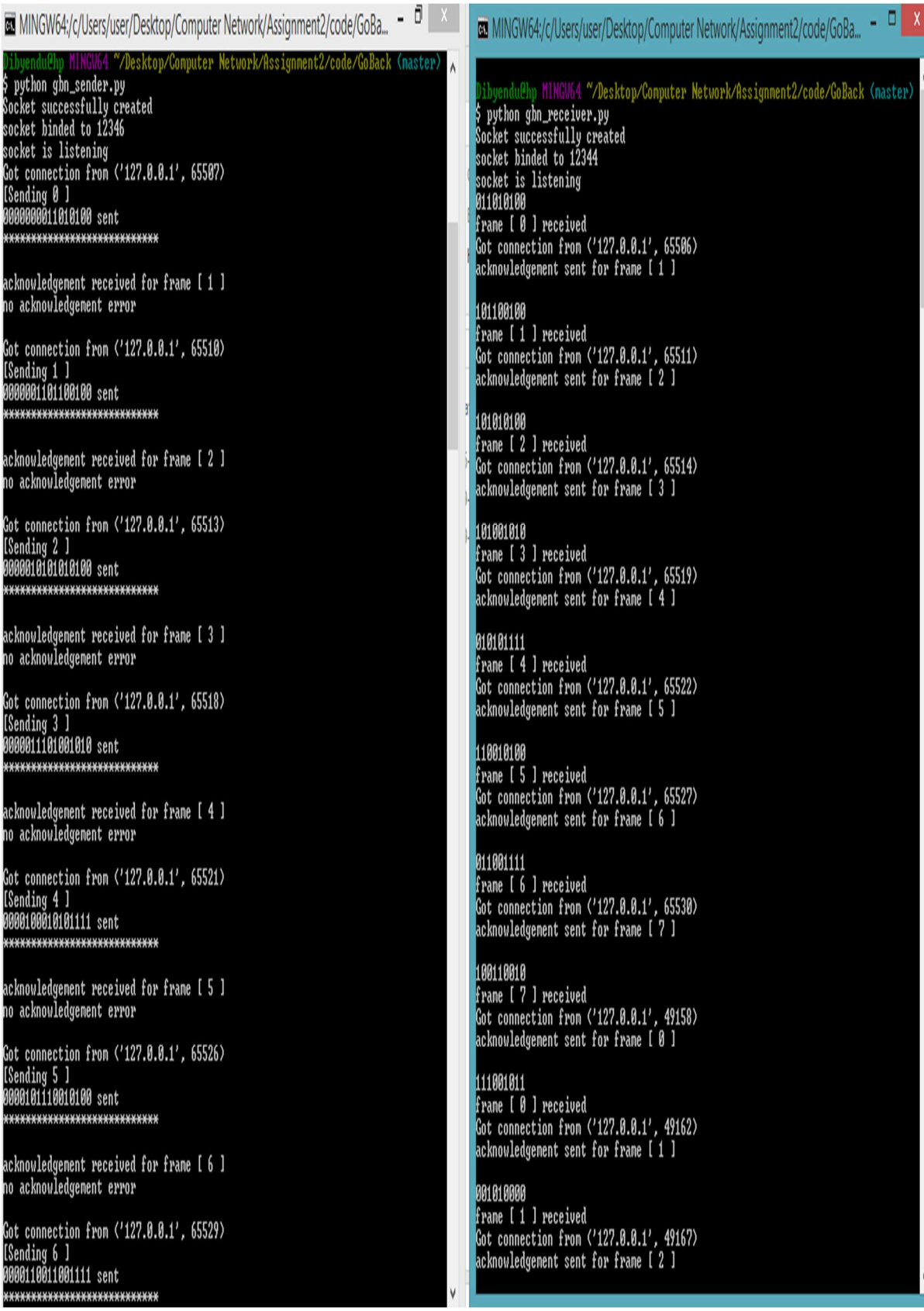
  The receive socket in the receiver side searches for connection from a port in the sender side. If a connection is found, it receives the frames, checks if corrupted and if the sequence number is same or more than as expected. Then a sender socket is created that sends the ACK for the next expected frame to the sender side.

```
while True:
    try:
        s_receive = socket.socket()
        s_receive.connect(('127.0.0.1', port_receive))
        frame=s_receive.recv(1024)
        s_receive.close()
        print("****")
        if corrupted(frame)==False:
            if seqno(frame)==r_n:
                try:
                    print("frame [ "+str(r_n)+" ] receiverd")
                    c, addr = s_send.accept()
                    print('Got connection from', addr )

                    r_n=(r_n+1)%(2**m)
                    time.sleep(3)

                    c.send(acknowledgement(r_n))
                    print("acknowledgement sent for frame [ "+str(r_n)+" ]\n")
                    c.close()
                except InterruptedError as err:
                    print(err)
            else:
                print("Expected Frame: "+str(r_n)+" Arrived Frame: "+str(seqno(frame))+"\n")
            # extract()
            # deliver()
        else:
            print("Arrived Frame is Corrupted\n")
    except ConnectionRefusedError as err:
        print(err)
    except OSError as err2:
        pass
```

- **OUTPUT**


MINGW64:/c/Users/user/Desktop/Computer Network/Assignment2/code/GoBa...
Dibyendu@hp MINGW64 ~/Desktop/Computer Network/Assignment2/code/GoBack (master)
$ python gbn_sender.py
Socket successfully created
socket binded to 12346
socket is listening
Got connection from ('127.0.0.1', 65507)
[Sending 0]
00000000110101100 sent
****
acknowledgement received for frame [1]
no acknowledgement error
Got connection from ('127.0.0.1', 65510)
[Sending 1]
00000011011001100 sent
****
acknowledgement received for frame [2]
no acknowledgement error
Got connection from ('127.0.0.1', 65513)
[Sending 2]
00000101010101100 sent
****
acknowledgement received for frame [3]
no acknowledgement error
Got connection from ('127.0.0.1', 65518)
[Sending 3]
00000111010010100 sent
****
acknowledgement received for frame [4]
no acknowledgement error
Got connection from ('127.0.0.1', 65521)
[Sending 4]
00001000101011111 sent
****
acknowledgement received for frame [5]
no acknowledgement error
Got connection from ('127.0.0.1', 65526)
[Sending 5]
00001011100101100 sent
****
acknowledgement received for frame [6]
no acknowledgement error
Got connection from ('127.0.0.1', 65529)
[Sending 6]
00001100110011111 sent
****
MINGW64:/c/Users/user/Desktop/Computer Network/Assignment2/code/GoBa...
Dibyendu@hp MINGW64 ~/Desktop/Computer Network/Assignment2/code/GoBack (master)
$ python gbn_receiver.py
Socket successfully created
socket binded to 12344
socket is listening
01101010
frame [0] received
Got connection from ('127.0.0.1', 65506)
acknowledgement sent for frame [1]
10110010
frame [1] received
Got connection from ('127.0.0.1', 65511)
acknowledgement sent for frame [2]
10101010
frame [2] received
Got connection from ('127.0.0.1', 65514)
acknowledgement sent for frame [3]
10100101
frame [3] received
Got connection from ('127.0.0.1', 65519)
acknowledgement sent for frame [4]
01010111
frame [4] received
Got connection from ('127.0.0.1', 65522)
acknowledgement sent for frame [5]
11001010
frame [5] received
Got connection from ('127.0.0.1', 65527)
acknowledgement sent for frame [6]
01100111
frame [6] received
Got connection from ('127.0.0.1', 65530)
acknowledgement sent for frame [7]
10011001
frame [7] received
Got connection from ('127.0.0.1', 49158)
acknowledgement sent for frame [0]
11100101
frame [0] received
Got connection from ('127.0.0.1', 49162)
acknowledgement sent for frame [1]
00101000
frame [1] received
Got connection from ('127.0.0.1', 49167)
acknowledgement sent for frame [2]

- **Result** :
  The performance of the above algorithm is measured in terms of throughput. The entire data file consists of 20 datawords. So for sending 20 frames in average of 65 attempts were taken considering insertion of random errors. Considering timeout after 5 seconds average propagation delay is 2 seconds.

- **Analysis**:
  The program may have some possible bugs due to the lack of randomness in injecting the error. Again, the program is implemented for one sender and receiver that can be extended up to multiple senders and receivers. The last few bits from the input file is discarded to make the dataword size same. This bug can be overcome by padding of 0's or 1's.

- **Comment**:
  The assignment was very helpful in better understanding of the implementation of one of the most popular flow control algorithms in noisy channel. Its difficulty level can be rated as extremely high.

SECTION 3 : Selective Repeat Sliding Window Design

The protocol is designed using two files- `gbn_sender.py` and `gbn_receiver.py`.

a. **`srp_sender.py`**: This file handles the sender side. It takes data input from file, converts the data into number of codewords of length of given frame size using the vrc algorithm. Then the program sends the frames one by one to the receiver. While sending the frame it also introduces error to the frame in a few cases. It also receives the acknowledgements from the receiver side and in case of NAK, it resends the frame at the end after sending only the ACK frame in the sorting way.
b. **`srp_receiver.py`**: This file handles the receiver side. It accepts the frames sent from the sender side, checks for error (if any), and if it is the expected frame, it sends the acknowledgment to the sender end.

The design of the implementation is shown in the diagram (3)

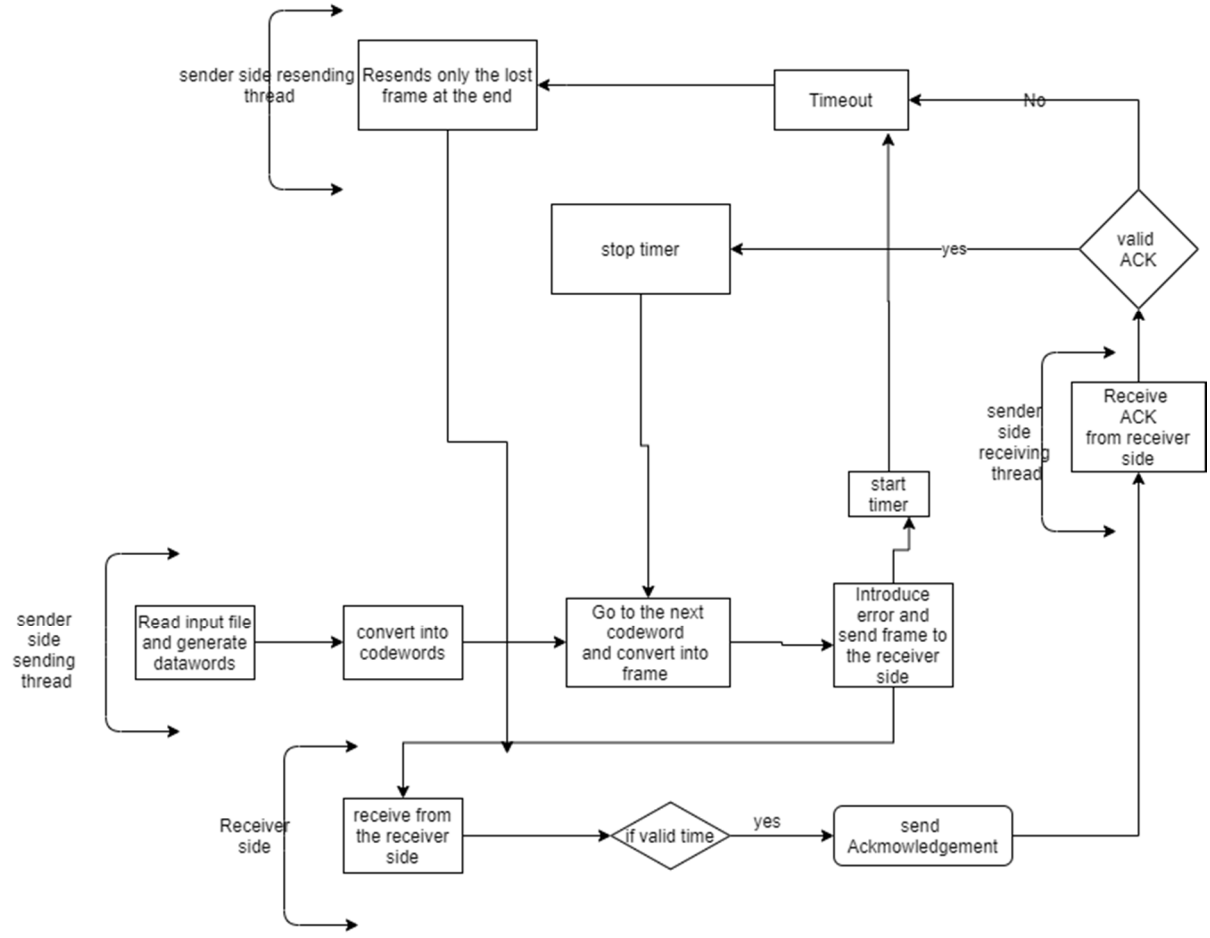


Figure 3: design of the implementation of selective repeat algorithm

a. **Frame size**: 16 bits
b. **Dataword size**: 8 bits
c. **Frame format**: sequence number of frame following the stuffed zeros

and the codeword at the end. The 8 bit data word is converted into 9 bit codeword usong vrc algorithm. Then the sequence number of the frame is added before it and finally zeros are stuffed to make the frame size 16.

d. **Error detection algorithm used**: vrc or simple pairity checker.
e. **Acknowledgement format**: 1 bit acknowledgement specifying the next expected frame.
f. **Input file format**: bit stream of 0's and 1's.
g. **Window size**: in this case m=3. Hence window size $s\_w = 2^m - 1 = 7$

• Implementation

The algorithm has been implemented using python3. The details are given below.

- *Srp_sender..py*

  Global variables and modules

```
port_send = 12346
port_receive = 12344

framesize=16
dwsize=8
x_time=0
codeword=[]
index=0

m=3
s_n=0
s_w=2**m-1
s_f=0
times=0

lock=threading.Lock()

window=dict()
```

- This method generates the frame from the 9 bit codeword.

```
#Generates the frame from the 9 bit codeword following the method described above
def gen_frame(codeword):
    new_cw=str(s_n)+'0'*(framesize-len(codeword)-1)+codeword
    return new_cw
```

- This function adds the parity bit after the 8 bit data word to generate 9 bit codeword.

```
#Adds the a parity bit after the 8bit dataword to generate the 9bit codeword
def make_pairity(num):
    count=0
    for s in num:
        if s=='1':
            count=count+1
    if count%2==0:
        num=num+'0'
    else:
        num=num+'1'
    return num
```

- This function reads the input data file and generates the dataword of length 8.

```
#Reads the input file and generates datawords of length 8
def generate_codewords():
    dataword=[]
    with open("data1.txt","rt") as fp:
        for line in fp:
            newdataword=[''.join(x) for x in zip(*[list(line[z::dwsize]) for z in range(dwsize)])]

    dataword.extend(newdataword)

    for dw in dataword:
        codeword.append(make_pairity(dw))
```

- It converts the frame from string to bytearray.

```
#Converts the frame from string to bytearray
def makeframe():
    global index
    new_s=gen_frame(codeword[index])
    index=index+1
    fr=bytearray()
    fr.extend(map(ord,new_s))

    return fr
```

- This module introduces the error in the desired frame.

```
#Introduces error in the desired frame.
def introduce_error(frame):
    frame1=frame
    if count%2==0:
        if frame1[len(frame1)-1]==49:
            frame1[len(frame1)-1]=48
        else:
            frame1[len(frame1)-1]=49
    return frame1
```

- Resends only the damaged /NAK/ lost frame from s_f to s_n in case of timeout in a new thread at the end.

```
def timeout():  #defines timeout protocol
    print("timeout!!")
    x_time=0
    try:
        temp=s_f
        while temp!=s_n:
            c,addr=s_send.accept()
            #print("Received signal from : ",addr)
            print("Got connection from",addr)
            if temp==2:
                window[str(temp)][len(window[str(temp)])-1]=48


            temp=(temp+1)%(2**m)

        #Resending the damaged or lost frame only at the end
        print("[Resending "+str(temp)+" ]")
        print(window[str(temp)].decode("utf-8")+" Sent")
        c.send(window[str(temp)])

        print("*******************************\n")
        time.sleep(2)
        c.close()

    except InterruptedError as err:
        print(err)
```

- **Threading class**

➔ `class sending_thread(threading.Thread):`

This class handles the sending of the data frames from the sender side to the receiver side. The **`while()`** loop runs infinitely to run the send operation whenever frames are ready and acknowledgment is received. When the sender is ready to send and there is a request to send, the sending socket checks for a connection with receiver port. If the connection is found sender creates a new frame from the dataword and introduces error (if any) to it. Then the sender sends the frame to the receiver port using the sending socket and the timer starts and `cansend` is turned `false`.

```
class sendThread(threading.Thread):
    def _init_(self):
        print("Thread for sending frames")

    def run(self):
        global s_n,s_f,s_w,cansend,x_time,window,frame,timer,times
        while True:
            if (s_n>=s_f and s_n-s_f<s_w) or (s_n<s_f):
                try:
                    c, addr = s_send.accept()
                    print('Got connection from', addr )
                except InterruptedError as err:
                    print(err)

                frame=makeframe()
                if(frame==-1):
                    break
                window[str(s_n)]=frame
                frame_e=introduce_error(frame)

                #will send only the damaged frame
                print("[Sending "+str(s_n)+" ]")
                print(window[str(s_n)].decode("utf-8")+" sent")
                c.send(frame_e)

                print("*****************************\n")
                time.sleep(2)
                c.close()

                lock.acquire()
```

➔ **class receiver_thread(threading.Thread)**

This class handles the receiving of the acknowledgement from the receiver side. The receiver socket searches for a connection from the sending socket in the receiver side. If there is any connection, the socket accepts the acknowledgment. If the acknowledgement is not corrupted and it is same or greater than `s_f`, then the timer is stopped and `cansend` is turned `true`. Again, all the frames from `s_f` to the acknowledgement number is purged.

```
class receiveThread(threading.Thread):
    def _init_(self):
        print("Thread for receiving Acknowledgements")

    def run(self):
        global s_n,s_f,s_w,cansend,x_time,window,frame,timer
        while True:
            try:
                s_receive = socket.socket()
                s_receive.connect(('127.0.0.1', port_receive))
                ackno=s_receive.recv(1024)
                print("acknowledgement received for frame [ "+ackno.decode("utf-8")+" ]")
                s_receive.close()
                if (s_n>=s_f and int(ackno)<=s_n and int(ackno)>s_f) or (s_n<s_f and int(ackn
                    print("No Acknowledgement error\n")
                    lock.acquire()
                    timer.cancel()
                    x_time=0
                    lock.release()
                    while (s_n>=s_f and s_f<int(ackno)) or (s_n<s_f and s_n>=int(ackno)):
                        x=window.pop(str(s_f))
                        s_f=(s_f+1)%(2**m)
                    frame=bytearray()
                    cansend=True
            except ConnectionRefusedError as err:
                pass
            except ConnectionResetError as err2:
                pass
```

- **Main Thread**

```
thread1=sending_thread()
thread2=receiver_thread()

thread1.start()
thread2.start()
```

- *gbn_receiver.py*
  Global variables and modules

```
framesize=16
dwsize=8
m=3
s_send = socket.socket()

print( "Socket successfully created")

port_send = 12346
port_receive = 12345

s_send.bind(('', port_send))
print( "socket binded to %s" %(port_send) )

s_send.listen(5)
print( "socket is listening")

r_n=0
```

- This method checks the parity of the given string. If the odd number of 1’s, it returns False else True.

```
#checks the parity of the given string.
#If odd number of 1’s it returns False else True
def check_pairity(num):
    count=0
    for s in num:
        if s=='1':
            count=count+1
    if count%2==0:
        return True
    return False
```

- This method checks whether there is an error in an incoming frame.

```
#checks if there is error in the incoming frame.
def corrupted(frame):
    s1=frame.decode('ASCII')
    s2=s1[framesize-dwsize-1:]
    if check_pairity(s2):
        return False
    return True
```

- This method creates the ACK for the next expected frame.

```
#creates acknowledgment for the next expected frame.
def acknowledgement(r_n):
    ack=str(r_n)
    b_ack=bytearray()
    b_ack.extend(map(ord,ack))
    return b_ack
```

- This function defines the sequence number of the incoming frame.

```
#returns the sequence number of the incoming frame.
def seqno(frame):
    if frame[0]==49:
        return 1
    else:
        return 0
```

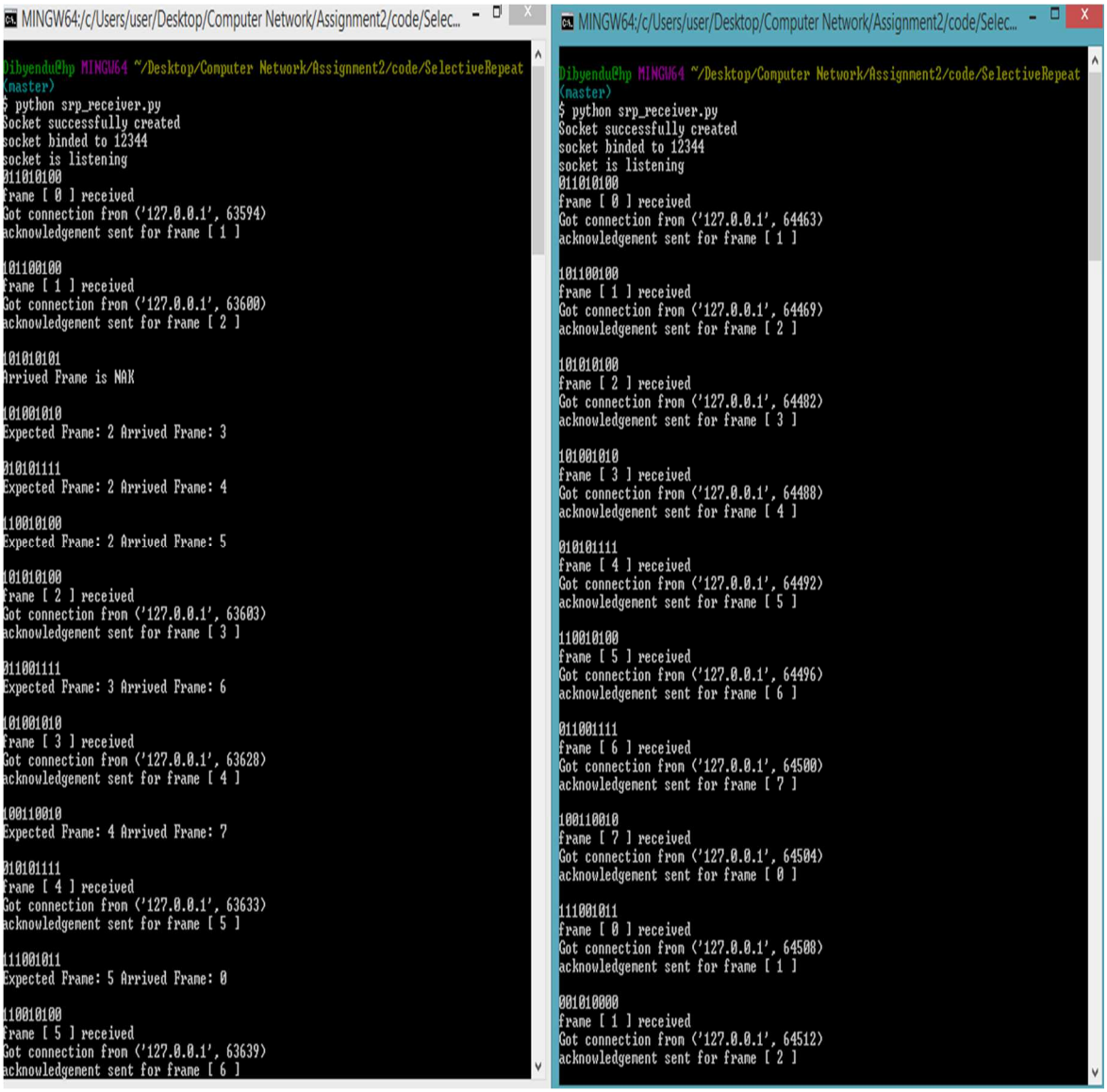
- Main Thread

```
while True:
    try:
        #print("cur=",cur," numDataword=",numDataword)
        if cur==numDataword:
            break
        s_receive = socket.socket()
        s_receive.connect(('127.0.0.1', port_receive))
        frame=s_receive.recv(1024)
        s_receive.close()
        if isEnd(frame):
            break
        if corrupted(frame)==False:
            if seqno(frame)==r_n:
                cur=cur+1
                try:
                    print("frame [ "+str(r_n)+" ] received")
                    c, addr = s_send.accept()
                    print('Got connection from', addr )

                    r_n=(r_n+1)%(2**m)
                    time.sleep(3)

                    c.send(acknowledgement(r_n))
                    print("acknowledgement sent for frame [ "+str(r_n)+" ]\n")
                    c.close()
                except InterruptedError as err:
                    print(err)
            else:
                print("Expected Frame: "+str(r_n)+" Arrived Frame: "+str(seqno(frame))+"\n")
```

- **OUTPUT**



- **Result** :
  The performance of the above algorithm is measured in terms of throughput. The entire data file consists of 20 datawords. So for sending 20 frames in average of 65 attempts were taken considering insertion of random errors. Considering timeout after 5 seconds average propagation delay is 2 seconds.

- **Analysis**:
  The program may have some possible bugs due to the lack of randomness in injecting the error. Again, the program is implemented for one sender and receiver that can be extended up to multiple senders and receivers. The last few bits from the input file is discarded to make the dataword size same. This bug can be overcome by padding of 0's or 1's.

- **Comment**:
  The assignment was very helpful in better understanding of the implementation of one of the most popular flow control algorithms in noisy channel. Its difficulty level

can be rated as extremely high.