ASSIGNMENT 2

NAME:SHASWATA SAHA
CLASS: BCSE III
GROUP: A1
ROLL: 001610501010
SUBJECT: COMPUTER NETWORKS LAB

<u>TITLE</u>

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

Sender, Receiver and Channel all are independent processes. There may be multiple Transmitter and Receiver processes, but only one Channel process. The channel process introduces random delay and/or bit error while transferring frames. Define your own frame format or you may use IEEE 802.3 Ethernet frame format.

Hints: Some points you may consider in your design.

Following functions may be required in Sender.

Send: This function, invoked every time slot at the sender, decides if the sender should (1) do nothing, (2) retransmit the previous data frame due to a timeout, or (3) send a new data frame. Also, you have to consider current network time measure in time slots.

Recv_Ack: This function is invoked whenever an ACK packet is received. Need to consider network time

when the ACK was received, ack_num and timestamp are the sender's sequence number and timestamp

that were echoed in the ACK. This function must call the timeout function.

Timeout: This function should be called by ACK method to compute the most recent data packet's round-trip time and then recompute the value of timeout.

Following functions may be required in Receiver.

Recv: This function at the receiver is invoked upon receiving a data frame from the sender.

Send_Ack: This function is required to build the ACK and transmit.

Sliding window: The sliding window protocols (Go-Back-N and Selective Repeat) extend the stop-and-wait protocol by allowing the sender to have multiple frames outstanding (i.e., unacknowledged) at any given time. The maximum number of unacknowledged frames at the sender cannot exceed its "window size". Upon receiving a frame, the receiver sends an ACK for the frame's sequence number. The receiver then buffers the received frames and delivers them in sequence number order to the application.

Performance metrics: Receiver Throughput (packets per time slot), RTT, bandwidth-delay product, utilization percentage.

Code Submission Deadline: 14/02/2019
Code Submitted On: 14/02/2019

DESIGN

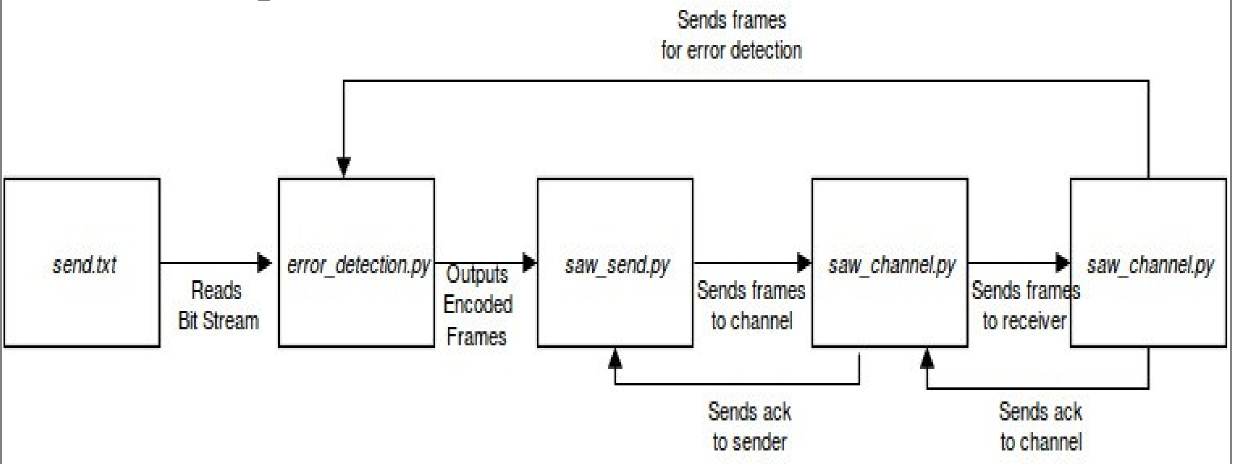
Stop-and-Wait ARQ

Purpose of the program:

This program aims at implementing Stop-and-Wait ARQ protocol in Computer Networking. This has been achieved by 3 programs:

- ***saw_send.py*** : This program defines various functions which read bit stream from a text file named *send.txt* and converts them into frames of codewords of desired lengths, and sends the frames to the receiver using a socket.
- ***saw_channel.py*** : This program uses 2 sockets, one connected to *saw_send.py* and the other connected to *saw_rcv.py*. It reads the data frames from *saw_send.py*, inserts error or delay in them using the *error_detection.py* program as shown in the previous assignment and then sends them to *saw_rcv.py*.
- ***saw_rcv.py*** : This program defines various functions which read bit stream from the socket connected to *saw_channel.py*, converts them into frames of datawords and prints the final message on screen.

Structure Diagram:



Input/Output Format:

The input is taken from a text file named *send.txt* . It contains a stream of 0s and 1s. These characters are treated as bits and are used to create datawords.

The message that is sent is printed on the terminal.

IMPLEMENTATION

Code:

- ***saw_send.py***

```python
import socket
import time
import os
import error_detection as ed

#host and port for socket connection
host = ""
port = 12345

#key for crc
key = '1011'
l=len(key)

#encoding frames by crc
crc_frames = ed.crc_encode(l,key)
print(crc_frames)

#setting up socket
s = socket.socket()
s.bind((host,port))
s.listen(5)

sn = 1

#f2 will store the frames that have been sent
f2=''

canSend = True

#gets address of receiver who is listening to it
c,addr = s.accept()

#functioning for resending
def send_loop(c,f,sn):
        c.sendall(f.encode('utf-8'))
        c.settimeout(5)
        try:
                ack = int(c.recv(1))
                print("\nsn = "+str(sn)+" ack = "+str(ack)+"\n")
                if ack == sn:
                        c.settimeout(5)
                        return
        except socket.timeout:
                print("Sending again "+f)
                send_loop(c,f,sn)

def send():
        i=0
        for f in crc_frames:
                #adding sn to frame
                f = str(sn)+f
                f2 = f
                c.sendall(f.encode('utf-8'))
                print('Sent '+f+'\nWaiting for ack')
                #setting time out of 5 secs
                c.settimeout(5)
                sn = (sn+1)%2
                i+=1
                try:
                        #receiving acknowledgement
                        ack = int(c.recv(1))
                        print("\nsn = "+str(sn)+" ack = "+str(ack)+"\n")

                        #checking acknowledgement
                        if ack==sn:
                                #timer is stopped
                                c.settimeout(None)
                except socket.timeout :
                        #if time out occurs saved frame is resent
                        print("Sending again")
                        send_loop(c,f2,sn)
                time.sleep(2)


        f = '000000'
        c.sendall(f.encode('utf-8'))
        print('Final frame sent')
        ack = int(c.recv(1))
        c.close()
        s.close()

send()
```

- ***saw_channel.py***

```
import socket
import time
import os
import error_detection as ed

host = ""
port_snd = 12345
port_rcv = 56780
key = '1011'
l=len(key)

ss = socket.socket()
ss.connect((host,port_snd))

s = socket.socket()
s.bind((host,port_rcv))
s.listen(5)

sr,addr = s.accept()

i = 0

while True:

        i+=1
        print('------------------------')
        try:
                f = (ss.recv(8).decode('utf-8'))
        except Exception as e:
                print('Timeout')

        #inserting error in 2nd, 5th frames only. hard coded for 
demonstration purpose only
        if i in [2,5]:
                print('Injecting error')
                f3 = ed.insert_error([f],1,[3,5])
                f = f3[0]
        if i != 7:
                print('Sending frame')
                sr.sendall(f.encode('utf-8'))
        else:
                print('Not sending frame')

        sr.settimeout(5)
        try:
                f2 = sr.recv(1)
                if i != 1:
                        ss.sendall(f2)
                else:
                        print('Not sending ack')
        except Exception as e:
                print('Timeout')
```

- ***saw_rcv.py***

```
import socket
import time
import os
import error_detection as ed

#crc key
key = '1011'
l=len(key)

#host and port of socket
port = 56780
host = ""

#stores the entire text received
text = ""

#setting up socket
s = socket.socket()
s.connect((host,port))

rn = 1

def rcv():
        while True:
                try:
                        f = (s.recv(8).decode('utf-8'))
                except:
                        break
        #if f is not null
        if f:
            if len(f)<8:
                print('Final frame received')
                s.send('1'.encode('utf-8'))
                break
            if ed.crc_check([f[-7:]],7,key):
                print(" Correct ")
                if f[0]==str(rn):
                    text += f[1:5]
                    rn = (rn+1)%2
                print('Received: seq num = '+f[0]+' frame = 
'+f[1:5]+' sending ack = '+str(rn))
                s.send(str(rn).encode('utf-8'))
                # print(text)
            else:
                print("Error")

    print(text)
    s.close()

rcv()
```

Function description:

The purpose and implementation of the functions in the above programs is given below:

saw_send.py:

- ***send():*** Iterates through the *crc_frames* array, adds the sequence number *sn* to each frame, and then sends it to the channel using the socket which will in turn send it to the receiver. Then it increases the *sn*, stores the sent frame in *f2*, starts a timer and waits for the acknowledgement *ack* from the receiver. If it gets the *ack* within the time, then it checks if the *ack* and the *sn* match or not. If yes, then the timer is stopped, the stored frame is purged and the next frame is sent. Otherwise, *send_loop()* function is called with *f2* and *sn* as arguments to send the stored frame.
- ***send_loop(c, f, sn):*** Sends the frame *f*, starts the timer and then waits for the acknowledgement *ack*. If it gets the correct *ack* in time then it returns to the *send()* functio. Otherwise it calls itself recursively with the same arguments.

saw_channel.py:

This program has 2 sockets. One connects it to the sender and the other to the receiver. It receives the frame from *saw_send.py*, sends it to *saw_rcv.py*, then receives the *ack* from it and sends it to *saw_send.py*. But it also introduces certain anomalies resembling the real world channel. For example, it introduces errors in some frames or may also not send the frame or the *ack* to its recipient.

saw_send.py:

- ***rcv():*** This function receives message from channel (*saw_channel.py*). Then it checks the *sn* of frame with its own sequence number *rn*. If they match then error checking is done of the frame. If no error is present *rn* is sent as acknowledgement.

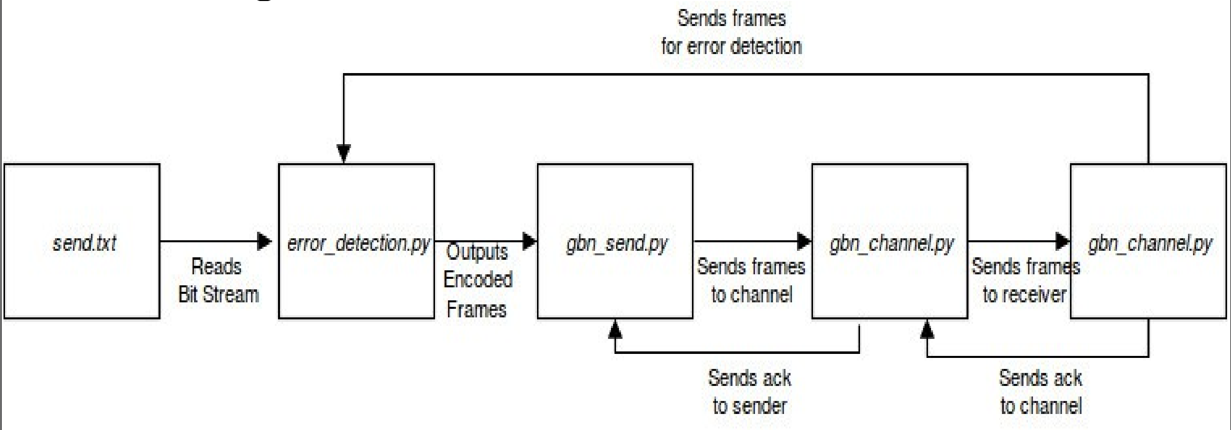
Go-Back-N ARQ

Purpose of the program:

This program aims at implementing Go-Back-N ARQ protocol in Computer Networking. To achieve this the following 3 programs have been designed:

- ***gbn_send.py*** : Defines functions which read bit stream from a text file named *send.txt,* converts them into frames, and sends them to the receiver using a socket.
- ***gbn_channel.py*** : This program emulates a real world channel. It uses 2 sockets, one connected to *gbn_send.py* and the another to *gbn_rcv.py*. It reads frames from *gbn_send.py*, inserts error or delay in them using the *error_detection.py* program as shown in the previous assignment and then sends them to *gbn_rcv.py*.
- ***gbn_rcv.py*** : Defines functions which read bit streams sent by *gbn_channel.py,* converts them into datawords and prints the final message on screen.

Structure Diagram:



Input/Output Format:

The input is taken from a text file named *send.txt* . It contains a stream of 0s and 1s. These characters are treated as bits and are used to create datawords.

The message that is sent is printed on the terminal.

IMPLEMENTATION

Code:

- *gbn_send.py*

```
import socket
import time
import os
import error_detection as ed
import threading

host = ""
port = 12346
key = '1011'
l=len(key)

crc_frames = ed.crc_encode(l,key)
print(crc_frames)

s = socket.socket()
s.bind((host,port))
s.listen(5)
c,addr = s.accept()

m = 2
sw = 2**m-1
sf = 0
sn = 0
isFinished = False

store_f = crc_frames

l = len(crc_frames)
t = -1

def rcv_ack():
        global sn,sf,sw
        while True:
                print('Receiving ack')
                if t == -1:
                        c.settimeout(5)
                #sn = sn + 1
                try:
                        ack = int(c.recv(1).decode('utf-8'))
                        print("\nsf = "+str(sf)+"sn = "+str(sn)+" ack = 
"+str(ack)+"\n")
                        if ack > (sf) and ack <= (sn):
                                while (sf)<ack:
                                        print('Deleting frame '+str(sf))
                                        sf += 1
                                c.settimeout(None)
                except:
                        print("Sending again")
                        send_loop(c)


def send_loop(c):
        global sf,sn,sw
        #print('Sending again')
        temp = sf
        while temp < sn:
                f = str(temp)+store_f[temp]
                c.sendall(f.encode('utf-8'))
                print('Sending again '+f+'\nWaiting for ack')
                temp += 1

def send():
        global sf,sn,sw
        i = 0
        while True:
                time.sleep(2)
                if (sn-sf) < sw:
                        if i < l:
                                #print(str(sn)+'\t'+str(sw))
                                f = str(sn)+crc_frames[i]
                                sn += 1
                                i += 1
                                c.sendall(f.encode('utf-8'))
                                print('Sent '+f+'\tWaiting for ack')


if __name__ == "__main__":
        t1 = threading.Thread(target = send)
        t2 = threading.Thread(target = send_loop)
        t3 = threading.Thread(target = rcv_ack)

        t1.start()

        t3.start()

        if not t1.isAlive():
                t1.join()
                t3.join()
```

- ***gbn_channel.py***

```
import socket
import time
import os
import error_detection as ed

host = ""
port_snd = 12346
port_rcv = 56781
key = '1011'
l=len(key)

ss = socket.socket()
ss.connect((host,port_snd))

s = socket.socket()
s.bind((host,port_rcv))
s.listen(5)

sr,addr = s.accept()
```

```
i = 0
while True:

        i+=1
        print('------------------------')
        try:
                f = (ss.recv(8).decode('utf-8'))
        except Exception as e:
                print('Timeout')

        #inserting error in 2nd, 5th frames only. hard coded for
demonstration purpose only
        if i in [2,5]:
                print('Injecting error')
                f3 = ed.insert_error([f],1,[3,5])
                f = f3[0]
        if i != 7:
                print('Sending frame')
                sr.sendall(f.encode('utf-8'))
        else:
                print('Not sending frame')

        sr.settimeout(5)
        try:
                f2 = sr.recv(1)
                if i != 1:
                        ss.sendall(f2)
                else:
                        print('Not sending ack')
        except Exception as e:
                print('Timeout')
```

- ***gbn_rcv.py***

```
import socket
import time
import os
import error_detection as ed

#crc key
key = '1011'
l=len(key)

#host and port of socket
port = 56781
host = ""

#stores the entire text received
text = ""

#setting up socket
s = socket.socket()
s.connect((host,port))

rn = 0
m = 2
sw = 2**m-1
```

```
def rcv():
        while True:
                try:
                        f = (s.recv(8).decode('utf-8'))
                except:
                        break

                #if f is not null
                if f:
                        if len(f)<8:
                                print('Final frame received')
                                s.send('1'.encode('utf-8'))
                                break
                        if ed.crc_check([f[-7:]],7,key):
                                print(" Correct ")
                                if f[0]==str(rn):
                                        text += f[1:5]
                                        rn = (rn+1)
                                print('Received: seq num = '+f[0]+' frame = 
'+f+' sending ack = '+str(rn)+' '+text)
                                s.send(str(rn).encode('utf-8'))
                                # print(text)
                        else:
                                print("Error")

        print(text)
        s.close()

rcv()
```

Function description:

The purpose and implementation of the functions in the above programs is given below:

gbn_send.py:

- ***send():*** This function iterates though the *crc_frames* array. It keeps on sending frames as long as the sliding window has empty places. If empty, then it adds the sequence number *sn* to the frame, sends it to *gbn_channel.py* and stores the frame in the window. Then it starts the timer and increases *sn*. Otherwise it waits for the sliding window to be empty.
- ***rcv_ack():*** Waits for the acknowledgment *ack* from the channel that is supposed to be sent by *gbn_rcv.py*. If it gets the *ack* and it is between *sf* and *sn*, then it stops the timer and purges all the frames from *sf* to *ack* and increases *sf* to *ack*. Otherwise it calls *send_loop()* to resend all the frames in the window.
- ***send_loop(c, f, sn):*** Sends all the frames from *sf* to *sn* in the sliding window.

saw_channel.py:

This program has 2 sockets. One connects it to the *gbn_send.py* and the other to the *gbn_rcv.py*. It receives the frame from the sender, sends it to the receiver, then receives the *ack* from it and sends it to the sender again. Apart from these it also inserts some anomalies like introducing errors in some frames or may be not sending the frame or the *ack* to the corresponding recipient.

saw_send.py:

- ***rcv():*** This function receives message from channel (*gbn_channel.py*). Then it checks the *sn* of frame with its own sequence number *rn*. If they match then error checking is done of the frame. If no error is present *rn* is sent as acknowledgement. Otherwise it waits for the sender to send the correct frame.

OUTPUT

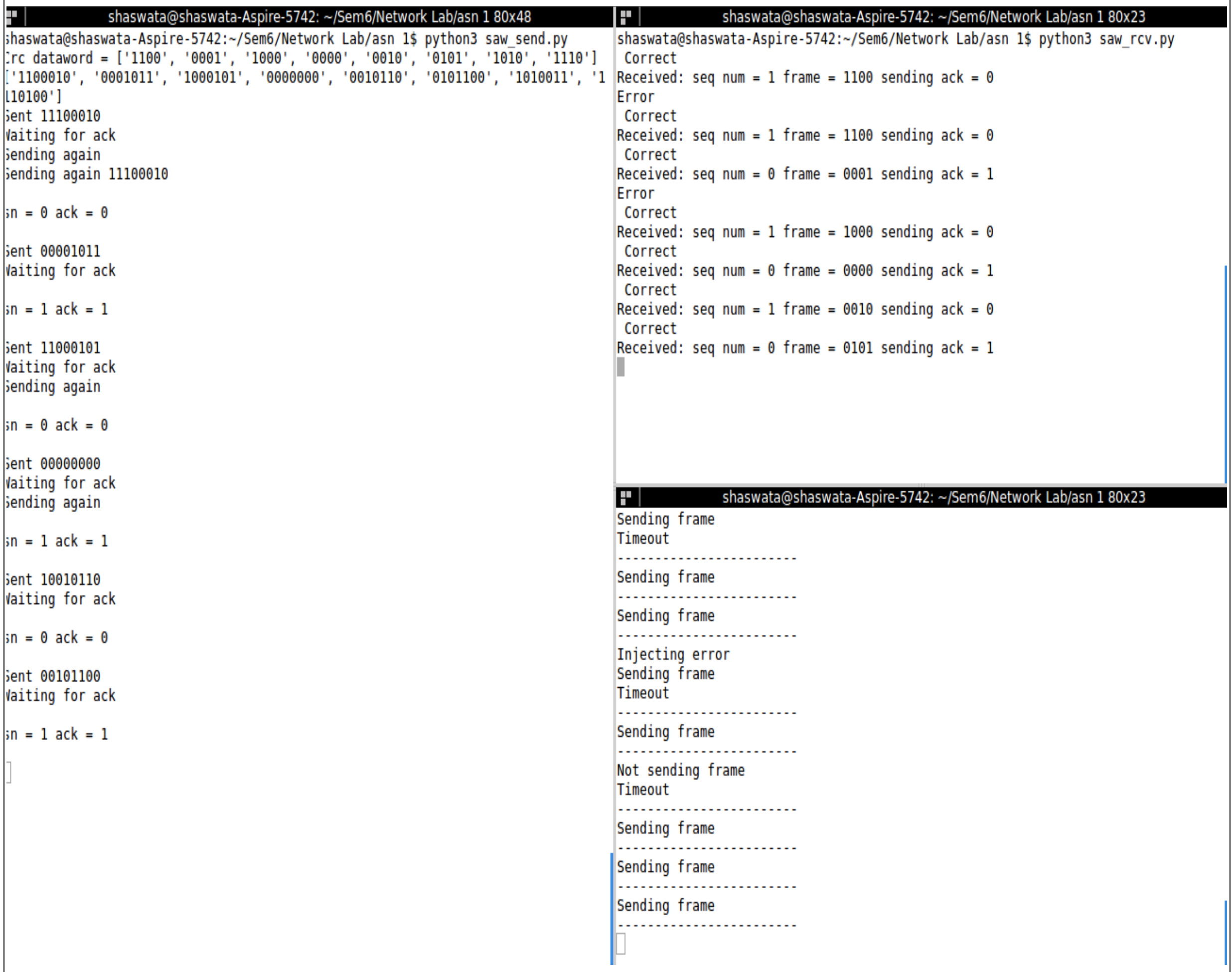
To demonstrate the 2 network protocols given in the problem statement, a *send.txt* file has been loaded with the following stream of 0s and 1s.

1100000110000000000100101101011110

Frames are read from this stream, sequence number is added and then they are sent into the channel to reach the revceiver. The channel then introduces errors and delays. Finally the frame reaches the receiver which then sends acknowledgement in the reverse path. In both of the outputs, the terminal at the left is the sender, the bottom right has the channel running and the one at the top right is the receiver.

- **Stop-and-Wait ARQ**

  Errors introduced at 3rd and 6th frames, 8th frame is dropped and ack of 2nd is dropped.



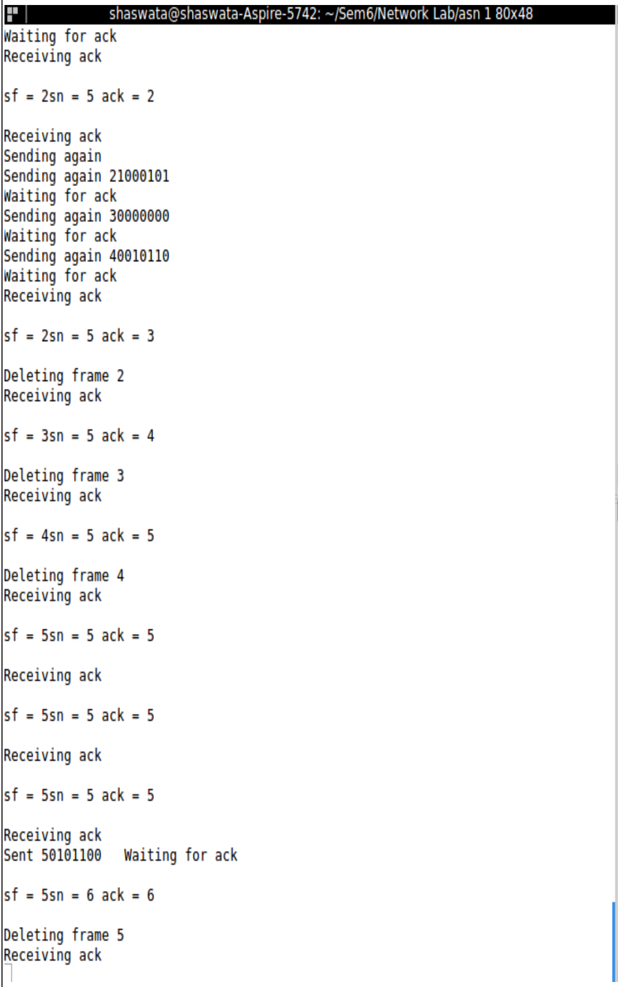
```
shaswata@shaswata-Aspire-5742: ~/Sem6/Network Lab/asn 1 80x48
;haswata@shaswata-Aspire-5742:~/Sem6/Network Lab/asn 1$ python3 saw_send.py
:rc dataword = ['1100', '0001', '1000', '0000', '0010', '0101', '1010', '1110']
['1100010', '0001011', '1000101', '0000000', '0010110', '0101100', '1010011', '1
110100']
;ent 11100010
Vaiting for ack
;ending again
;ending again 11100010

;n = 0 ack = 0

;ent 00001011
Vaiting for ack

;n = 1 ack = 1

;ent 11000101
Vaiting for ack
;ending again

;n = 0 ack = 0

;ent 00000000
Vaiting for ack
;ending again

;n = 1 ack = 1

;ent 10010110
Vaiting for ack

;n = 0 ack = 0

;ent 00101100
Vaiting for ack

;n = 1 ack = 1
```

```
shaswata@shaswata-Aspire-5742: ~/Sem6/Network Lab/asn 1 80x23
shaswata@shaswata-Aspire-5742:~/Sem6/Network Lab/asn 1$ python3 saw_rcv.py
 Correct
Received: seq num = 1 frame = 1100 sending ack = 0
Error
 Correct
Received: seq num = 1 frame = 1100 sending ack = 0
 Correct
Received: seq num = 0 frame = 0001 sending ack = 1
Error
 Correct
Received: seq num = 1 frame = 1000 sending ack = 0
 Correct
Received: seq num = 0 frame = 0000 sending ack = 1
 Correct
Received: seq num = 1 frame = 0010 sending ack = 0
 Correct
Received: seq num = 0 frame = 0101 sending ack = 1
```

```
shaswata@shaswata-Aspire-5742: ~/Sem6/Network Lab/asn 1 80x23
Sending frame
Timeout
-----------------------
Sending frame
-----------------------
Sending frame
-----------------------
Injecting error
Sending frame
Timeout
-----------------------
Sending frame
-----------------------
Not sending frame
Timeout
-----------------------
Sending frame
-----------------------
Sending frame
-----------------------
Sending frame
-----------------------
```

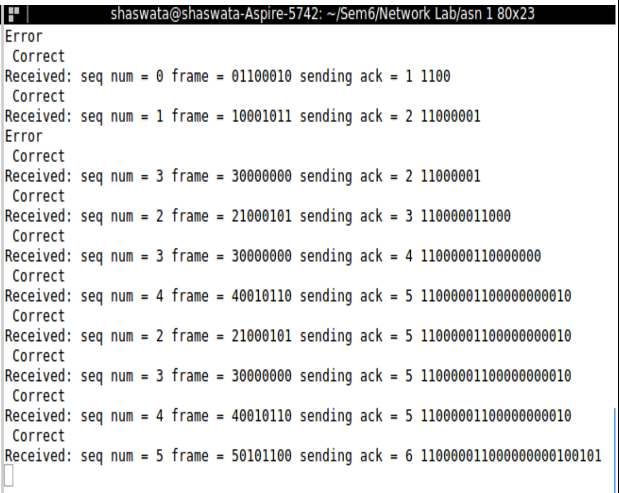
- **Go-Back-N ARQ**

  Errors introduced at 3rd and 6th frames, 8th frame is dropped and ack of 2nd is dropped.



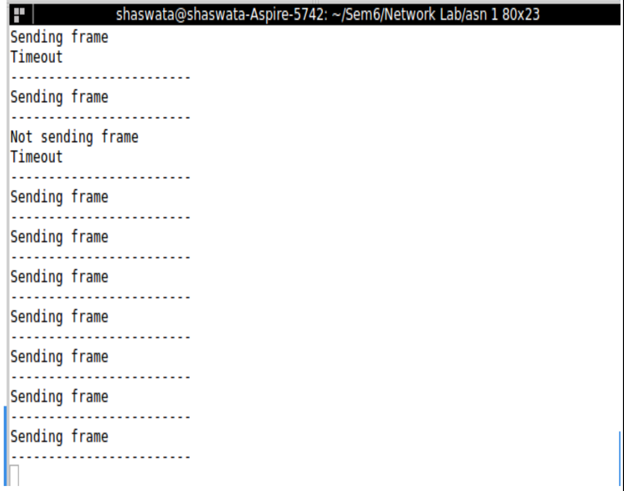
shaswata@shaswata-Aspire-5742: ~/Sem6/Network Lab/asn 1 80x48

```
Waiting for ack
Receiving ack

sf = 2sn = 5 ack = 2

Receiving ack
Sending again
Sending again 21000101
Waiting for ack
Sending again 30000000
Waiting for ack
Sending again 40010110
Waiting for ack
Receiving ack

sf = 2sn = 5 ack = 3

Deleting frame 2
Receiving ack

sf = 3sn = 5 ack = 4

Deleting frame 3
Receiving ack

sf = 4sn = 5 ack = 5

Deleting frame 4
Receiving ack

sf = 5sn = 5 ack = 5

Receiving ack

sf = 5sn = 5 ack = 5

Receiving ack

sf = 5sn = 5 ack = 5

Receiving ack
Sent 50101100   Waiting for ack

sf = 5sn = 6 ack = 6

Deleting frame 5
Receiving ack
```



shaswata@shaswata-Aspire-5742: ~/Sem6/Network Lab/asn 1 80x23

```
Error
 Correct
Received: seq num = 0 frame = 01100010 sending ack = 1 1100
 Correct
Received: seq num = 1 frame = 10001011 sending ack = 2 11000001
Error
 Correct
Received: seq num = 3 frame = 30000000 sending ack = 2 11000001
 Correct
Received: seq num = 2 frame = 21000101 sending ack = 3 110000011000
 Correct
Received: seq num = 3 frame = 30000000 sending ack = 4 1100000110000000
 Correct
Received: seq num = 4 frame = 40010110 sending ack = 5 11000001100000000010
 Correct
Received: seq num = 2 frame = 21000101 sending ack = 5 11000001100000000010
 Correct
Received: seq num = 3 frame = 30000000 sending ack = 5 11000001100000000010
 Correct
Received: seq num = 4 frame = 40010110 sending ack = 5 11000001100000000010
 Correct
Received: seq num = 5 frame = 50101100 sending ack = 6 11000001100000000100101
```



shaswata@shaswata-Aspire-5742: ~/Sem6/Network Lab/asn 1 80x23

```
Sending frame
Timeout
-----------------------
Sending frame
-----------------------
Not sending frame
Timeout
-----------------------
Sending frame
-----------------------
Sending frame
-----------------------
Sending frame
-----------------------
Sending frame
-----------------------
Sending frame
-----------------------
Sending frame
-----------------------
Sending frame
-----------------------
```

RESULTS

Both the protocols have been demonstrated using the same message bits. Each protocol has to send 8 frames. While Stop-and-Wait ARQ sends 12 frames to complete the job, Go-back-N ARQ takes 15 frame sendings. However, in Go-Back-N ARQ, all the frames in the sliding window are sent almost simultaneously without waiting for the acknowledgement of each other. Thus though, Go-back-N has to send 15 frames, but it takes time that is equivalent to sending 11 frames in Stop-And-Wait ARQ.

ANALYSIS

The programs have been designed with the intention to make them as complete and sound as possible. Just like IEEE formats, the program also uses its own format that is kept constant throughout all the programs. The protocols can deal with messages of varying lengths and windows of varying sizes. In accordance to the success of CRC error detection of the previous assignment, the module has been used in this assignment too.

However one drawback of both the protocols is that the introduction of errors and dropping of frames by the channel have been hard-coded for demonstration purpose and not randomized.

COMMENTS

This assignment helps us understand to how the Stop-And-Wait ARQ and Go-Back-N ARQ protocols are implemented in real world. This has been a great learning experience. The difficulty of this assignment according to me is moderate.