NETWORK LAB REPORT

NAME: ANURAN CHAKRABORTY
ROLL NO.: 20
CLASS: BCSE-III
SECTION: A1

ASSIGNMENT NUMBER: 2

PROBLEM STATEMENT:
Implement two data link layer protocols, Stop and Wait, Go Back N Sliding Window for flow control.

DEADLINE: 14TH FEBRUARY, 2019
SUBMITTED ON: 14TH FEBRUARY, 2019
REPORT SUBMITTED ON: 21ST FEBRUARY, 2019

The report has two sections one for the stop-and-wait protocol and the other for the sliding window go-back-n protocol.

STOP AND WAIT ARQ:

DESIGN

The program implements the stop and wait ARQ flow control protocol. The program consists of 5 modules.

1. **errorchecker.py**
   This file contains the implementation of all the error detection algorithm from assignment1.

2. **sender.py**
   This file contains the code to perform the work of the sender. Read from the input file, create the frame to be sent to the receiver and the send the frame to the channel process. This process also receives the acknowledgement sent by the receiver and accordingly resends the frame if ack is not received or is corrupt.

3. **channel.py**
   This is the channel process whose task are the following
   a) Receive frame from sender
   b) Inject error into the frame with a random probability
   c) Send the frame to the receiver with a random probability else discard it and wait for the sender to resend, also inject a random probability.
   d) Receive the acknowledgement frame from the receiver and send it to the sender with a random probability and also inject delay.

4. **rec.py**
   This file contains the code to perform the work of the receiver. It receives the frame sent by the sender from the channel process, generates an acknowledgement and sends it to the channel process.

5. **common.py**
   This file contains some common function to be used by all processes.

All inter-process communication has been carried out by making use of the Python3 **socket**.

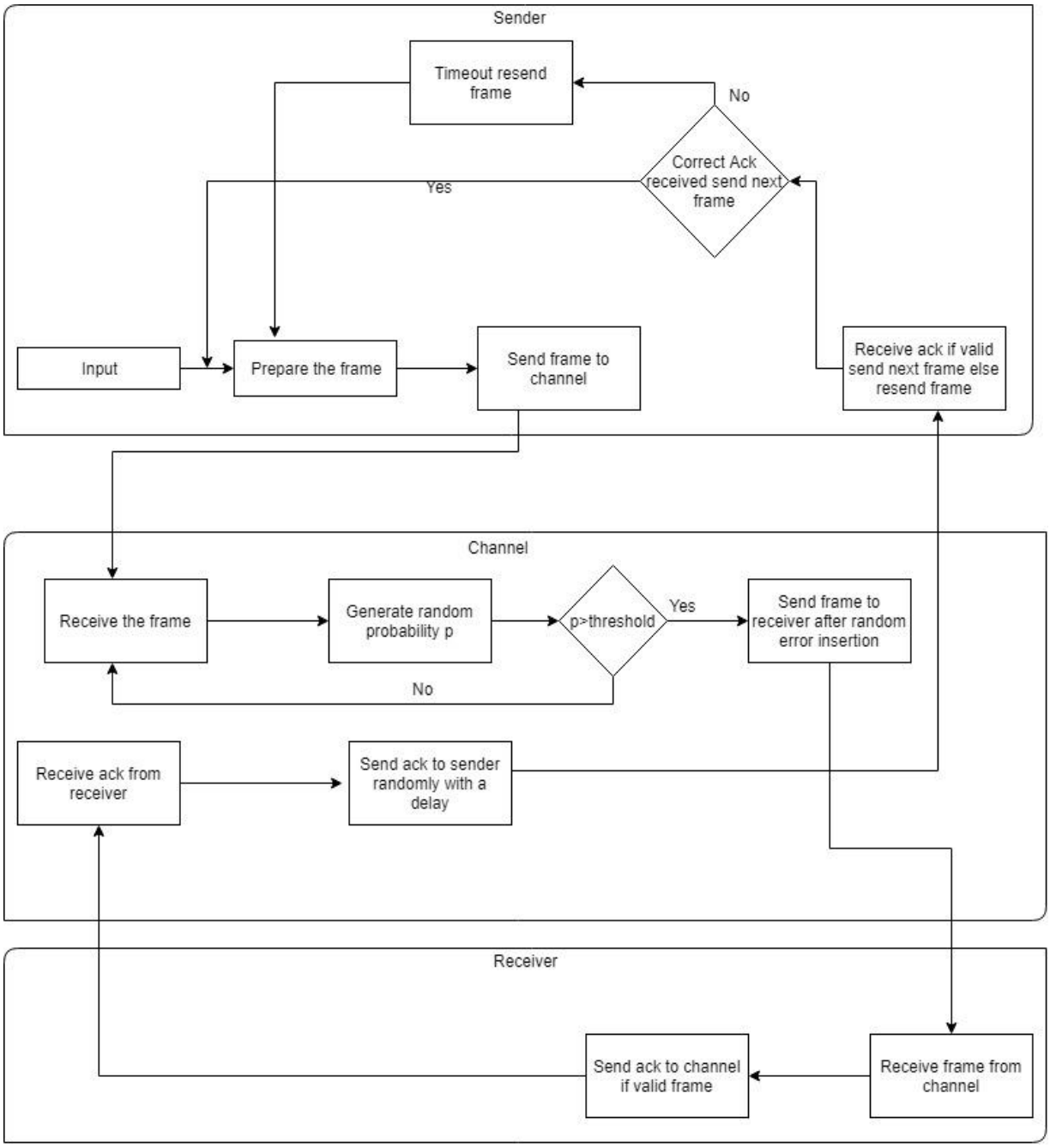

Fig. 1. *A brief outline of the program design of stop and wait ARQ*

Fig. 1 gives a brief outline of the program.

Some important parameters for the design of the program are:

Frame format: The frame format used in the sender process is described as follows. The input data is split into frames of 4 bits each. Then for each frame the frame number (0 or 1) is

appended to the beginning of the frame. The entire 5 bit data is then encoded using CRC-3. Thus the resulting frame size is 8 bits. This frame is then sent to the channel process.

Acknowledgement format: The acknowledgement frame which is sent by the receiver consists of a bit stating the frame number (0 or 1) which is then CRC encoded using CRC-3 thus making 4 bit acknowledgement.

Assumption: During the design one assumption that has been made is that the number of bits in the input file is a multiple of 4.

Input format: The input for the program is a text file consisting of a string of only 0s and 1s.

Output format: The program output simulates the flow control protocol.

IMPLEMENTATION

The assignment has been implemented in Python3. The detailed description is given below.

errorchecker.py: This is the same error checking module as present in Assignment 1.

common.py:

This module contains some commonly used function in the modules.

The port specifications are given below

```
portSenderReceive=11001
portSenderSend=11002
portReceiverReceive=11003
portReceiverSend=11004
frame_size=4
```

createSocket(port):

This function creates a socket and binds it to a port

```
# Function to create a socket and bind it to a port
def createSocket(port):
        s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.bind(('', port))
        return s
```

allowConn(port):

Function to establish a connection with the port.

```
# Function to receive a connection
def allowConn(s):
        s.listen(5)
        c, addr=s.accept()
        return c, addr
```

createConn(port):

Function to create a socket with a port and connect to it.

```
# Function to create a socket and connect to it
def createConn(port):
        sock=socket.socket()
        sock.connect(('',port))
        return sock
```

send_frame(frame,c):

Function to send a frame through a particular socket.

```
# Function to send a frame
def send_frame(frame, c):
        # Send the frame to the other process
        c.send(frame.encode())
```

prepare_frame(frame,c):

Function to prepare the frame for the sender given the frame number by converting it to binary and applying crc..

```
# Function to prepare a frame
def prepare_frame(frame,sn):
        frame=str(sn)+frame
        # CRC application
        crcframe=err.crc([frame], err.generator_poly, frame_size)
        return crcframe[0]
```

generateAck(rn):

Function to generate acknowledgement for the receiver given the frame number by converting it to binary and applying crc.

```
# Function to generate ack
def generateAck(rn):
        # Generate crc appended code
        ack=bin(rn)[2:]
        crcframe=err.crc([ack], err.generator_poly, frame_size)
        return crcframe[0]
```

readFile(filename, frame_size):

Function to read the input file and split into frames.

```
# Function to read the file and split into frames
def readfile(filename, frame_size):
        # Open the file for reading
        f=open(filename,'r')
        data=f.read()

        # Now split the data into frames
        list_of_frames=[data[i:i+frame_size] for i in range(0, len(data),
frame_size)]
        return list_of_frames
```

ins_error(frame, list_of_bits):

Function to insert error at certain bit positions in the frame.

```
# Function to introduce error
def ins_error(frame, list_of_bit):

        new=list(frame)

        # Inserting error in the given bit position here
        for i in range(len(list_of_bit)):
                if(new[list_of_bit[i]]=='0'):
                        new[list_of_bit[i]]='1'
                elif (new[list_of_bit[i]]=='1'):
                        new[list_of_bit[i]]='0'
        new=''.join(new)
        return new
```

sender.py:

This is the code for the sender process.

```
timeoutTime=5
frame_size=4
```

isValid(ack,sn):

This function checks if the acknowledgement received is valid by performing a CRC check and also matching with the expected sequence number.

```
# Check if ack is valid using crc
def isValid(ack,sn):

        if(ack[0]!=str((sn+1)%2)):
                return False
        # Now check CRC
        if(int(err.modulo2div(ack,err.generator_poly),2)!=0):
                return False
        return True
```

send_all(list_of_frames):

Function to send all the frames to the channel process via the sockets. It first connects to the sockets and sends the frames and waits. If it encounters a timeout it resends the frame.

```
# Function to send all the frames
def send_all(list_of_frames):
```

```
        sockSend=co.createSocket(co.portSenderSend) # sender socket to send
data to channel
        c, addr=co.allowConn(sockSend)

        sockRec=co.createConn(co.portSenderReceive)   # Socket to receive data
from channel
        sockRec.settimeout(timeoutTime)

        print('Connected to channel')
        # implementing stop and wait arq
        sn=0
        i=0
        while(i<(len(list_of_frames))): # While there are frames send
                print(15*'-')
                canSend=True
                sn=(i)%2
                stored_frame=list_of_frames[i]
                if(stored_frame!='#'):
                        stored_frame=co.prepare_frame(list_of_frames[i],sn)
                print('Sending frame '+str(i)+' '+stored_frame)
                co.send_frame(stored_frame, c)
                canSend=False

                try:
                        ack=sockRec.recv(1024).decode()
                except Exception as e:
                        # Resend so repeat this iteration of loop
                        print('Timeout.. Resending')
                        continue

                if(ack=='#'):
                        break
                print('Ack received '+ack)
                if(ack and isValid(ack,sn)): # Wrong acknowledgement
                        print('Correct ack received')
                        canSend=True
                        i=i+1
                elif(not isValid(ack,sn)):
                        # invalid ack so resend
                        print('Wrong ack.. resending')

                print(15*'-')

        # Close the sockets
        sockSend.close()
```

```
        sockRec.close()
```

channel.py:

This module implements the channel process. It first creates the appropriate sockets and then it goes into an infinite loop waiting for the sender to send. It receives the frame from the sender, may inject error and injects delay and the sends it to the receiver and also does the same for the receiver. It may also not send a frame to the sender or receiver (this is to simulate a lost acknowledgement or a lost frame).

```
# This is the channel process
# First create the required sockets
import common as co
import random
import time

probas=10
randSendF=2 # Random probability of sending frame or not
randSendAck=2
randErrF=1
randErrAck=1

# *************** SENDER ****************************
sockSenderReceive=co.createConn(co.portSenderSend) # Socket to receive data
from sender

sockSenderSend=co.createSocket(co.portSenderReceive)  # Socket to send data to
sender
senderSend, addr=co.allowConn(sockSenderSend)
print('Channel connected to sender')
#*****************************************************

#***************** RECEIVER *************************
sockReceiverReceive=co.createConn(co.portReceiverSend) # Socket to receive
data from receiver
sockReceiverReceive.settimeout(3)

sockReceiverSend=co.createSocket(co.portReceiverReceive)     # Socket to send
data to receiver
receiverSend, addr=co.allowConn(sockReceiverSend)
print('Channel connected to receiver')
#*****************************************************

while True:
        print(15*'-')
```

```
        # Receive the frame from the sender
        stored_frame=sockSenderReceive.recv(1024).decode()
        print('Frame received from sender '+stored_frame)
        # Insert error and other stuffs here

        # Send frame with a probability p
        p=random.randint(0,probas)
        print(p)
        if(p>=randSendF or len(stored_frame)<8): # Ending marker should always
be sent
                # Introduce error here with a probability
                p2=random.randint(0,probas)
                if(p2<=randErrF and len(stored_frame)>=8):
                        print("Introducing error")
                        stored_frame=co.ins_error(stored_frame,[1])

                # Send the frame to the receiver
                print('Sending frame to receiver '+stored_frame)
                # Add sleep here
                time.sleep(2)
                co.send_frame(stored_frame, receiverSend)
                print('Sent frame '+stored_frame)


        # Dont Send the frame
        else:
                print('Not sending frame')
                continue

        try:
                # Wait ack from receiver
                ack=sockReceiverReceive.recv(1024).decode()
                print('Ack received from receiver '+ack)
        except Exception as e:
                # Resend so repeat this iteration of loop
                print('Timeout.. waiting for sender to send')
                continue
        # Insert error and other stuffs here

        # send the frame with a probability
        p=random.randint(0,probas)
        print(p)
        if(p>=randSendAck or stored_frame=='#'):
                # Introduce error here with a probability
                p2=random.randint(0,probas)
```

```
                if(p2<=randErrAck and stored_frame=='#'):
                        print("Introducing error")
                        ack=co.ins_error(ack,[1])

                print('Sending ack to sender '+ack)
                # Add sleep here
                time.sleep(2)
                # Send the ack to the sender
                co.send_frame(ack, senderSend)
                if(stored_frame=='#'):
                        break
        else:
                print('Not sending acknowledgement')
                continue

        print(15*'-')



sockSenderReceive.close()
sockSenderSend.close()
sockReceiverReceive.close()
sockReceiverSend.close()
```

rec.py

This module is the receiver process.

isValid(ack,sn):

This function checks if the frame received is valid by performing a CRC check and also matching with the expected sequence number on the receiver side.

```
# Function to check if frame is valid
def isValid(frame, rn):
        if(frame[0]!=str(rn)):
                return 0
        # Now check CRC
        if(int(err.modulo2div(frame,err.generator_poly),2)!=0):
                return 1
        return 2
```

receive():

Function to receive a frame from the channel and send the ack.

```
def receive():
        # Establish connection
```

```
        sockRec=co.createSocket(co.portReceiverSend)
        c, addr=co.allowConn(sockRec)

        sockSend=co.createConn(co.portReceiverReceive)
        print('Connected to channel')

        # Connection established
        rn=0
        while True:
                # Wait till frame received
                print(15*'-')
                frame=sockSend.recv(1024).decode()

                print('Frame received '+frame)

                if(frame=='#'):
                        ack='#'
                else:
                        if(isValid(frame, rn)==0): # wrong frame no received
send ack for prev
                                print('Invalid frame')
                                print('Sending ack for previous frame')
                                ackno=(rn)%2

                        elif(isValid(frame, rn)==1):
                                print('Error in frame')
                                continue

                        else: # Valid frame
                                ackno=(rn+1)%2

                        # For valid data frame
                        rn=(rn+1)%2
                        # Send an acknowledgement
                        ack=co.generateAck(ackno)
                # Send the ack
                print('Sending ack '+ack)
                co.send_frame(ack,c)

                if(frame=='#'): # Means end frame
                        break
                print(15*'-')

        # Close the sockets
        sockSend.close()
```

```
        sockRec.close()

print('Demonstrating STOP AND WAIT ARQ')
receive()
```

OUTPUTS

RESULTS

The throughput here was measured in terms of the attempts it took to send the entire data. With random frame loss and random error insertion it took an average of 20 attempts by the sender to send 10 frames of data. Average propagation time was 2 seconds and with delay inserted it was average of 3 seconds.

ANALYSIS

Overall the implementation of the assignment is more or less correct. Some possible bugs can arise due to the assumption that the input size is a multiple of the frame size. However, this can easily be overcome by padding the last frame of the input data with 0s so that it is a multiple of the frame size. Currently the program works only for single sender and receiver processes but the program may be modified to work with multiple sender and multiple receiver processes.

COMMENTS

Overall the lab assignment was a great learning experience as we got to implement the well-known flow control protocols ourselves. The assignment can be rated as moderately difficult.

GO BACK N SLIDING WINDOW:

DESIGN

The program implements the go back N flow control protocol. The program consists of 5 modules.

1. **errorchecker.py**
   This file contains the implementation of all the error detection algorithm from assignment1.

2. **gbn_sender.py**
   This file contains the code to perform the work of the sender. Read from the input file, create the frame to be sent to the receiver and the send the frame to the channel process. This process also receives the acknowledgement sent by the receiver and accordingly resends the frame if ack is not received or is corrupt. Here as the sending and receiving can be done simultaneously they are run as two separate threads one sending the frame till window size is full and the other receiving the acknowledgement and deleting the required frames. A third thread in the sender process handles the timeout and resending operation.

3. **channel_gbn.py**
   This is the channel process where two threads are created. One thread receives a frame from the sender injects delay and error and sends it to the receiver. The second thread receives an acknowledgement frame from the receiver and sends it to the sender.

4. **gbn_receiver.py**
   This file contains the code to perform the work of the receiver. It receives the frame sent by the sender from the channel process, generates an acknowledgement and sends it to the channel process.

5. **common.py**
   This file contains some common function to be used by all processes.

All inter-process communication has been carried out by making use of the Python3 **socket**.

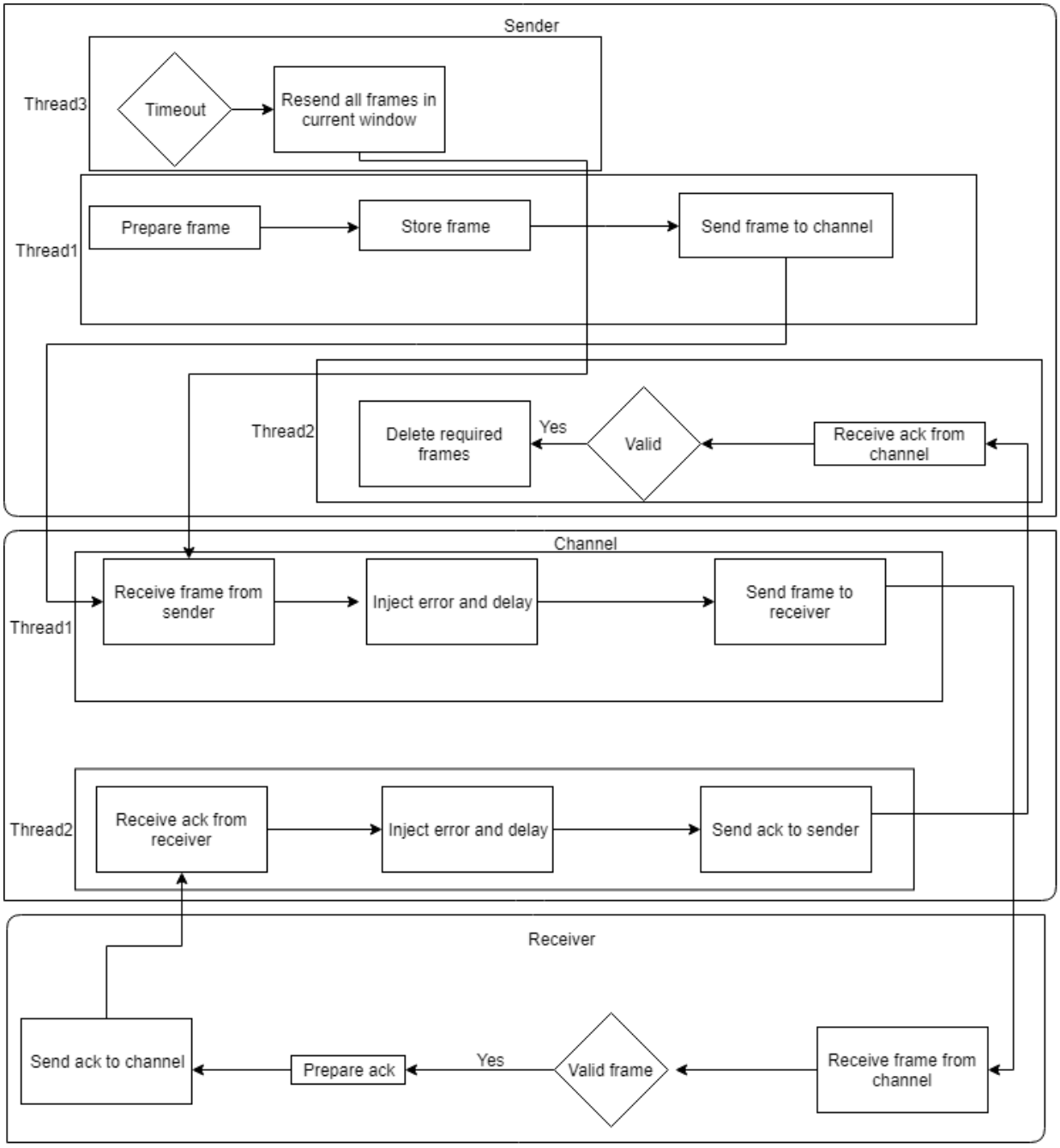

Fig. 2. *A brief outline of the program design of go back n*

Fig. 2 gives a brief outline of the program.

Some important parameters for the design of the program are:

Window size: The window size assumed is 3 (2^2-1).

Frame format: The frame format used in the sender process is described as follows. The input data is split into frames of 4 bits each. Then for each frame the frame number converted to binary (00 to 11) is appended to the beginning of the frame. The entire 6 bit data is then encoded using CRC-3. Thus the resulting frame size is 9 bits. This frame is then sent to the channel process.

Acknowledgement format: The acknowledgement frame which is sent by the receiver consists of a bit stating the frame number in binary which is then CRC encoded using CRC-3 thus making 5 bit acknowledgement.

Assumption: During the design one assumption that has been made is that the number of bits in the input file is a multiple of 4.

Input format: The input for the program is a text file consisting of a string of only 0s and 1s.

Output format: The program output simulates the flow control protocol.

IMPLEMENTATION

The assignment has been implemented in Python3. The detailed description is given below.

errorchecker.py: This is the same error checking module as present in Assignment 1.

common.py:

This module contains some commonly used function in the modules.

The port specifications are given below

```
portSenderReceive=11001
portSenderSend=11002
portReceiverReceive=11003
portReceiverSend=11004
frame_size=4
m=2
window_size=2**m-1
```

createSocket(port):

This function creates a socket and binds it to a port

```
# Function to create a socket and bind it to a port
def createSocket(port):
        s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.bind(('', port))
        return s
```

allowConn(port):

Function to establish a connection with the port.

```
# Function to receive a connection
def allowConn(s):
        s.listen(5)
        c, addr=s.accept()
        return c, addr
```

createConn(port):

Function to create a socket with a port and connect to it.

```
# Function to create a socket and connect to it
def createConn(port):
        sock=socket.socket()
        sock.connect(('',port))
        return sock
```

send_frame(frame,c):

Function to send a frame through a particular socket.

```
# Function to send a frame
def send_frame(frame, c):
        # Send the frame to the other process
        c.send(frame.encode())
```

prepare_frame_gbn(frame,c):

Function to prepare the frame for the sender given the frame number by converting it to binary and applying crc.

```
def prepare_frame_gbn(frame,sn):
        sn=sn%(window_size+1)
        sn=bin(sn)
        sn=sn[2:].zfill(3)
        frame=str(sn)+frame
        # CRC application
        crcframe=err.crc([frame], err.generator_poly, frame_size)
        return crcframe[0]
```

generateAck_gbn(rn):

Function to generate acknowledgement for the receiver given the frame number by converting it to binary and applying crc.

```
# Function to generate ack
def generateAck(rn):
        # Generate crc appended code
        ack=bin(rn)[2:]
        crcframe=err.crc([ack], err.generator_poly, frame_size)
        return crcframe[0]
```

readFile(filename, frame_size):

Function to read the input file and split into frames.

```
# Function to read the file and split into frames
def readfile(filename, frame_size):
        # Open the file for reading
        f=open(filename,'r')
        data=f.read()

        # Now split the data into frames
```

```
        list_of_frames=[data[i:i+frame_size] for i in range(0, len(data),
frame_size)]
        return list_of_frames
```

ins_error(frame, list_of_bits):

Function to insert error at certain bit positions in the frame.

```
# Function to introduce error
def ins_error(frame, list_of_bit):

        new=list(frame)

        # Inserting error in the given bit position here
        for i in range(len(list_of_bit)):
                if(new[list_of_bit[i]]=='0'):
                        new[list_of_bit[i]]='1'
                elif (new[list_of_bit[i]]=='1'):
                        new[list_of_bit[i]]='0'
        new=''.join(new)
        return new
```

sender_gbn.py:

This is the code for the sender process. The global variables sn and sf are pointers to the end and beginning and of the window respectively and sw is the window size defined in common.py.

```
timeoutTime=17
frame_size=4
sw=co.window_size
sf=0
sn=0
stored_buffer={i:'' for i in range(sw)}

print('Demonstrating Go Back N ARQ')
list_of_frames=co.readfile('input.txt', frame_size)
print(list_of_frames)
list_of_frames.append('#') # Attach a blank frame

sockSend=co.createSocket(co.portSenderSend) # sender socket to send data to
channel
sender, addr=co.allowConn(sockSend)

sockRec=co.createConn(co.portSenderReceive)   # Socket to receive data from
channel
sockRec.settimeout(timeoutTime)
```

```
print('Connected to channel')

send_all(list_of_frames)
```

isValid(ack,sn):

This function checks if the acknowledgement received is valid by performing a CRC check.

```
# Check if ack is valid using crc
def isValid(ack):
        # Now check CRC
        if(int(err.modulo2div(ack,err.generator_poly),2)!=0):
                return False
        return True
```

send_all(list_of_frames):

Function to send all the frames to the channel process via the sockets. It creates the required threads for sending and receiving.

```
# Function to send all the frames
def send_all(list_of_frames):

        sendThread=threading.Thread(target=sendFrame, args=(list_of_frames,))
# create the sending thread
        receiveThread=threading.Thread(target=receiveFrame) # create the
receiving thread

        sendThread.start()
        receiveThread.start()



        sendThread.join()
        print('Send thread end')
        receiveThread.join()

        # Close the sockets
        sockSend.close()
        sockRec.close()
```

sendFrame():

This function is run as a separate thread which sends all the frames to the channel until window size is full.

```
# Function to send list of frames
def sendFrame(list_of_frames):
```

```
        global sn
        global stored_buffer
        global sw
        i=0
        while True:
                # Store_frame(sn)
                stored_frame=list_of_frames[i]
                print('sn: '+str(sn)+' sf: '+str(sf)+' sw: '+str(sw))
                if(stored_frame!='#'):

        stored_frame=co.prepare_frame_gbn(list_of_frames[i],(i%(co.window_size
+1)))
                # Else send the blank frame
                # If window size reached wait
                if((sn-sf)<sw):
                        # Sendframe(sn)
                        stored_buffer[sn%sw]=stored_frame
                        print('Sending frame '+str(i)+' '+stored_frame)
                        co.send_frame(stored_frame, sender)
                        sn=(sn+1)

                        if (i<len(list_of_frames)-1):
                                i=i+1
                else:
                        print('Window Size full')
                print(stored_buffer)
                time.sleep(5)
```

receiverFrame():

This function is run as a separate thread which receives ack from the channel process, checcks if it is a valid ack and accordingly deletes the required frames. In case of a timeout this function also starts the resend thread which resends all the frames in the range sf and sn.

```
# Function to receive ack
def receiveFrame():
        global sf
        global sn
        global sw
        global stored_buffer

        while True:

                try:
                        ack=sockRec.recv(1024).decode()
                except Exception as e:
```

```
                        # Resend so repeat this iteration of loop
                        print('Timeout.. Resending')

        resendThread=threading.Thread(target=resendFrameAfterTimeout,
args=(stored_buffer,)) # create the resending thread
                        resendThread.start()
                        resendThread.join()
                        continue

                print('Ack received '+str((ack[0:3])))
                if(ack !='#' and isValid(ack)): # Correct acknowledgement
                        print('Correct ack received')
                        ackno=int(ack[0:3],2)
                        print('Ackno '+str(ackno))
                        # Purge required frames
                        if(sn%(sw+1)<sf%(sw+1) and ackno<sf%(sw+1)):
                                while((sf%(sw+1))>ackno):
                                        print('Deleting frame '+str(sf%sw))
                                        stored_buffer[sf%sw]=''
                                        sf=(sf+1)
                                stored_buffer[sf%sw]=''

                        if(ackno>=(sf%(sw+1)) and ackno<=(sn%(sw+1))):
                                while((sf%(sw+1))<ackno):
                                        print('Deleting frame '+str(sf%sw))
                                        stored_buffer[sf%sw]=''
                                        sf=(sf+1)
                                sf=sf+1

                elif(ack !='#' and not isValid(ack)): # Wrong ack
                        # invalid ack so resend
                        print('Wrong ack.. resending')
```

<u>resendFrameAfterTimeout(list_of_frames):</u>

This function is also run as a separate thread for resending frames after timeout.

```
# Function to resend frame after timeout
def resendFrameAfterTimeout(list_of_frames):
        global sn
        global sf
        global stored_buffer

        # Resend frame
        temp=sf
        print('Resending '+str(sf))
```

```
        while(temp<sn):
                if(stored_buffer[temp%sw]!=''):
                        print('Resending frame '+str(temp%(sw))+'
'+stored_buffer[temp%sw])
                        co.send_frame(stored_buffer[temp%sw], sender)
                        time.sleep(5)
                temp=(temp+1)
```

channel.py:

This module implements the channel process. It first creates the appropriate sockets. It then starts two threads one to receive data from sender and send to receiver and other to receive data from receiver and send to sender. It may also not send a frame to the sender or receiver (this is to simulate a lost acknowledgement or a lost frame).

```
# This is the channel process
# First create the required sockets
import common as co
import random
import time
import threading

probas=10
randSendF=2 # Random probability of sending frame or not
randSendAck=2
randErrF=1
randErrAck=-1

# *************** SENDER *****************************
sockSenderReceive=co.createConn(co.portSenderSend) # Socket to receive data
from sender

sockSenderSend=co.createSocket(co.portSenderReceive)  # Socket to send data to
sender
senderSend, addr=co.allowConn(sockSenderSend)
print('Channel connected to sender')
#****************************************************

#***************** RECEIVER *************************
sockReceiverReceive=co.createConn(co.portReceiverSend) # Socket to receive
data from receiver
# sockReceiverReceive.settimeout(11)

sockReceiverSend=co.createSocket(co.portReceiverReceive)      # Socket to send
data to receiver
receiverSend, addr=co.allowConn(sockReceiverSend)
```

```
print('Channel connected to receiver')
#*****************************************************
```

receiveFromSender():

This function receives the frame from the sender and sends it to the receiver after injecting delay and error in the frame.

```
def receiveFromSender():

        while True:
                print(15*'-')
                # Receive the frame from the sender
                stored_frame=sockSenderReceive.recv(1024).decode()
                print('Frame received from sender '+stored_frame)
                # Insert error and other stuffs here

                # Send frame with a probability p
                p=random.randint(0,probas)
                print(p)
                if(p>=randSendF or len(stored_frame)<8): # Ending marker
should always be sent
                        # Introduce error here with a probability
                        p2=random.randint(0,probas)
                        if(p2<=randErrF and len(stored_frame)>=8):
                                print("Introducing error")
                                stored_frame=co.ins_error(stored_frame,[1])

                        # Send the frame to the receiver
                        print('Sending frame to receiver '+stored_frame)
                        # Add sleep here
                        # time.sleep(7)
                        co.send_frame(stored_frame, receiverSend)
                        print('Sent frame '+stored_frame)

                # Dont Send the frame
                else:
                        print('Not sending frame')
                print(15*'-')
```

receiveAckFromReceiver():

This function receives the ack from the receiver and sends it to the sender.

```
def receiveAckFromReceiver():
```

```
        while True:
                print(15*'=')
                # Wait ack from receiver
                ack=sockReceiverReceive.recv(1024).decode()
                print('Ack received from receiver '+ack)

                # Insert error and other stuffs here

                # send the frame with a probability
                p=random.randint(0,probas)
                print(p)
                if(p>=randSendAck):
                        # Introduce error here with a probability
                        p2=random.randint(0,probas)
                        if(p2<=randErrAck):
                                print("Introducing error")
                                ack=co.ins_error(ack,[1])

                        print('Sending ack to sender '+ack)
                        # Add sleep here
                        # time.sleep(7)
                        # Send the ack to the sender
                        co.send_frame(ack, senderSend)
                else:
                        print('Not sending acknowledgement')
                print(15*'=')
```

mainChannel():

It creates the required threads and starts the threads one for receiving data from sender and the other for receiving from receiver.

```
def mainChannel():

        sendThread=threading.Thread(target=receiveFromSender) # create the
sending thread
        receiveThread=threading.Thread(target=receiveAckFromReceiver) # create
the receiving thread

        receiveThread.start()
        sendThread.start()

        sendThread.join()
        receiveThread.join()

mainChannel()
```

```
sockSenderReceive.close()
sockSenderSend.close()
sockReceiverReceive.close()
sockReceiverSend.close()
```

gbn_receiver.py

This module is the receiver process.

isValid(ack,sn):

This function checks if the frame received is valid by performing a CRC check and also matching with the expected sequence number on the receiver side.

```
# Function to check if frame is valid
def isValid(frame, rn):
        if(int(err.modulo2div(frame,err.generator_poly),2)!=0):
                return False
        elif(int(frame[0:3],2)%(co.window_size+1)!=rn):
                return False
        return True
```

receive():

Function to receive a frame from the channel and send the ack.

```
def receive():
        # Establish connection
        sockRec=co.createSocket(co.portReceiverSend)
        c, addr=co.allowConn(sockRec)

        sockSend=co.createConn(co.portReceiverReceive)
        print('Connected to channel')

        # Connection established
        rn=0
        while True:
                # Wait till frame received
                print(15*'-')
                frame=sockSend.recv(1024).decode()
                print('Expecting: '+str(rn))
                print('Frame received '+frame)

                if(frame!='#' and not isValid(frame, rn)): # wrong frame no
received send ack for prev
```

```
                        print('Invalid frame..Not sending ack')
                elif(frame!='#'):
                        ackno=frame[0:3]
                        # Send an acknowledgement
                        ack=co.generateAck_gbn(ackno)
                        time.sleep(5)
                        # Send the ack
                        print('Sending ack '+ack)
                        co.send_frame(ack,c)
                        rn=(rn+1)%(co.window_size+1)
                else:
                        # For valid frame
                        ack='#'
                        time.sleep(5)
                        # Send the ack
                        print('Sending ack '+ack)
                        co.send_frame(ack,c)



                print(15*'-')

        # Close the sockets
        sockSend.close()
        sockRec.close()

print('Demonstrating Go Back N ARQ')
receive()
```

OUTPUTS


anuran:/media/anuran/WORK/JUCSE WORK/3rd Year 2nd Sem/Networks/Assignment2 (master ?) $ python3 gbn_sender.py
Demonstrating Go Back N ARQ
['1001', '0001', '1100', '1010']
Connected to channel
sn: 0 sf: 0 sw: 3
Sending frame 0 0001001000
{0: '0001001000', 1: '', 2: ''}
sn: 1 sf: 0 sw: 3
Sending frame 1 0010001011
{0: '0001001000', 1: '0010001011', 2: ''}
sn: 2 sf: 0 sw: 3
Sending frame 2 0101100001
{0: '0001001000', 1: '0010001011', 2: '0101100001'}
sn: 3 sf: 0 sw: 3
Window Size full
{0: '0001001000', 1: '0010001011', 2: '0101100001'}
Timeout.. Resending
Resending 0
Resending frame 0 0001001000
sn: 3 sf: 0 sw: 3
Window Size full
{0: '0001001000', 1: '0010001011', 2: '0101100001'}
Resending frame 1 0010001011
sn: 3 sf: 0 sw: 3
Window Size full
{0: '0001001000', 1: '0010001011', 2: '0101100001'}
Resending frame 2 0101100001
sn: 3 sf: 0 sw: 3
Window Size full
{0: '0001001000', 1: '0010001011', 2: '0101100001'}
Ack received 001
Correct ack received
Ackno 1
Deleting frame 0
sn: 3 sf: 2 sw: 3
Sending frame 3 0111010101
{0: '0111010101', 1: '0010001011', 2: '0101100001'}
sn: 4 sf: 2 sw: 3
Sending frame 4 #
{0: '0111010101', 1: '#', 2: '0101100001'}
sn: 5 sf: 2 sw: 3
Window Size full
{0: '0111010101', 1: '#', 2: '0101100001'}
Ack received #
sn: 5 sf: 2 sw: 3
Window Size full
{0: '0111010101', 1: '#', 2: '0101100001'}
anuran:/media/anuran/WORK/JUCSE WORK/3rd Year 2nd Sem/Networks/Assignment2 (master ?) $ python3 gbn_receiver.py
Demonstrating Go Back N ARQ
Connected to channel
Expecting: 0
Frame received 0001001000
Sending ack 000000
Expecting: 1
Frame received 0101100001
Invalid frame..Not sending ack
Expecting: 1
Frame received 0001001000
Invalid frame..Not sending ack
Expecting: 1
Frame received 0010001011
Sending ack 001001
Expecting: 2
Frame received 0101100001
Sending ack 010010
Expecting: 3
Frame received 0111010101
Sending ack 011011
/media/anuran/WORK/JUCSE WORK/3rd Year 2nd Sem/Networks/Assignment2 [anuran] 115x33
Channel connected to sender
Channel connected to receiver
Frame received from sender 0001001000
7
Sending frame to receiver 0001001000
Sent frame 0001001000
Frame received from sender 0010001011
1
Not sending frame
Ack received from receiver 000000
1
Not sending acknowledgement
Frame received from sender 0101100001
9
Sending frame to receiver 0101100001
Sent frame 0101100001
Frame received from sender 0001001000
8
Sending frame to receiver 0001001000
Sent frame 0001001000
Frame received from sender 0010001011


anuran:/media/anuran/WORK/JUCSE WORK/3rd Year 2nd Sem/Networks/Assignment2 (master ?) $ python3 gbn_sender.py
Demonstrating Go Back N ARQ
['1001', '0001', '1100', '1010']
Connected to channel
sn: 0 sf: 0 sw: 3
Sending frame 0 0001001000
{0: '0001001000', 1: '', 2: ''}
sn: 1 sf: 0 sw: 3
Sending frame 1 0010001011
{0: '0001001000', 1: '0010001011', 2: ''}
sn: 2 sf: 0 sw: 3
Sending frame 2 0101100001
{0: '0001001000', 1: '0010001011', 2: '0101100001'}
sn: 3 sf: 0 sw: 3
Window Size full
{0: '0001001000', 1: '0010001011', 2: '0101100001'}
Timeout.. Resending
Resending 0
Resending frame 0 0001001000
sn: 3 sf: 0 sw: 3
Window Size full
{0: '0001001000', 1: '0010001011', 2: '0101100001'}
Resending frame 1 0010001011
sn: 3 sf: 0 sw: 3
Window Size full
{0: '0001001000', 1: '0010001011', 2: '0101100001'}
Resending frame 2 0101100001
sn: 3 sf: 0 sw: 3
Window Size full
{0: '0001001000', 1: '0010001011', 2: '0101100001'}
Ack received 001
Correct ack received
Ackno 1
Deleting frame 0
sn: 3 sf: 2 sw: 3
Sending frame 3 0111010101
{0: '0111010101', 1: '0010001011', 2: '0101100001'}
sn: 4 sf: 2 sw: 3
Sending frame 4 #
{0: '0111010101', 1: '#', 2: '0101100001'}
sn: 5 sf: 2 sw: 3
Window Size full
{0: '0111010101', 1: '#', 2: '0101100001'}
Ack received #
sn: 5 sf: 2 sw: 3
Window Size full
{0: '0111010101', 1: '#', 2: '0101100001'}
sn: 5 sf: 2 sw: 3
Window Size full
{0: '0111010101', 1: '#', 2: '0101100001'}
sn: 5 sf: 2 sw: 3
Window Size full
{0: '0111010101', 1: '#', 2: '0101100001'}
Timeout.. Resending
Resending 2
Resending frame 2 0101100001
Expecting: 1
Frame received 0001001000
Invalid frame..Not sending ack
Expecting: 1
Frame received 0010001011
Sending ack 001001
Expecting: 2
Frame received 0101100001
Sending ack 010010
Expecting: 3
Frame received 0111010101
Sending ack 011011
Expecting: 0
Frame received #
Sending ack #
Expecting: 0
Frame received 0101100001
Invalid frame..Not sending ack
/media/anuran/WORK/JUCSE WORK/3rd Year 2nd Sem/Networks/Assignment2 [anuran]
1
Not sending acknowledgement
Frame received from sender 0111010101
4
Sending frame to receiver 0111010101
Sent frame 0111010101
Frame received from sender #
2
Sending frame to receiver #
Sent frame #
Ack received from receiver 011011
1
Not sending acknowledgement
Ack received from receiver #
9
Sending ack to sender #
Frame received from sender 0101100001
7
Sending frame to receiver 0101100001
Sent frame 0101100001

RESULTS

The throughput here was measured in terms of the attempts it took to send the entire data. With random frame loss and random error insertion it took an average of 7 attempts by the sender to send 10 frames of data with a window of size 3. Average propagation time for a frame was 2 seconds and with delay inserted it was average of 3 seconds per frame. So total delay for the window is 9 seconds.

ANALYSIS

Overall the implementation of the assignment is more or less correct. Some possible bugs can arise due to the assumption that the input size is a multiple of the frame size. However, this can easily be overcome by padding the last frame of the input data with 0s so that it is a multiple of the frame size. Currently the program works only for single sender and receiver processes but the program may be modified to work with multiple sender and multiple receiver processes.

COMMENTS

Overall the lab assignment was a great learning experience as we got to implement the well-known flow control protocols ourselves. The assignment can be rated as difficult.