NETWORK LAB REPORT

NAME : Dibyendu Mukhopadhyay
CLASS : BCSE-III
ROLL NO. : 001710501077
GROUP : A3
ASSIGNMENT NO. : 3

PROBLEM STATEMENT:

In this assignment, you have to implement 1-persistent, non-persistent and p-persistent CSMA techniques. Measure the performance parameters like throughput (i.e., average amount of data bits successfully transmitted per unit time) and forwarding delay (i.e., average end-to-end delay, including the queuing delay and the transmission delay) experienced by the CSMA frames (IEEE 802.3). Plot the comparison graphs for throughput and forwarding delay by varying p. State your observations on the impact of performance of different CSMA techniques.

SUBMISSION DUE : 17TH FEBRUARY, 2020
REPORT SUBMITTED : 19TH SEPTEMBER,2020

This report has broadly 3 classifications :

- ➔ 1-persistent
- ➔ Non-persistent
- ➔ P-persisent

- # DESIGN

The protocol is designed using 4 files (1-persistent/Non-persistent/p-persistent) , channel.py, common.py and receiver.py.

- The different classification of CSMA are explained in later with detailed explanation along with code snippet. Here, they are the sender file.
- common.py : This file handles the inter process communication by the use of socket programming. It takes the input data and then converted into the list of frames. It then creates the socket connection for sending the frame through the channel file.
- Channel.py : This file simply build for the intermediate connection between sender and receiver in the form port address which was already build by common file. Its tasks are to receive frame from sender, send this frame to the receiver and a parallel thread that sends a continuous signal to the sender stating whether the channel is free or not.
- Receiver.py : This file is nothing but only receive the frame via channel process.

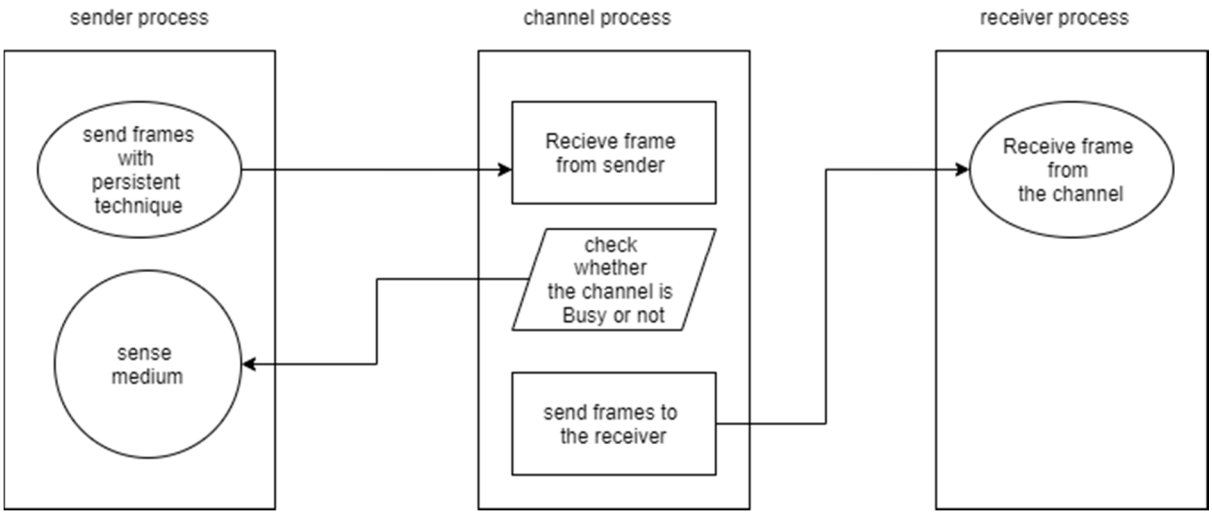

Fig.1 : Schematic diagram of the programming design of CSMA

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. This frame is then sent to the channel process.
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 contains a streams of only 0's and 1's.
Output format: The program output simulates the CSMA protocol.

- **IMPLEMENTATION**

The entire code is written in Python and the detailed method description is written below with code snippet.

➔ common.py

This module contains commonly used functions.
The port specified in global.

```
port_sendrec=11001
port_sendsignal=11002
port_recsend=11003
frame_size=4
shared_buffer=[]
```

- This function reads the file and returns the list of frame.

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

- 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.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    s.bind(('127.0.0.1', port))
    return s
```

- 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
```

- 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(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect(('',port))
    return sock
```

- 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())
```

➔ Channel.py

This module implements the channel process. First and foremost, it creates the sockets and then it goes into an infinite loop waiting for the sender to send. It receives the frame from the sender and then keeps it in a buffer array. The frames from the buffer is then sent to the receiver. The channel has actually 4 parallel threads running. One main thread continuously checks if any new sender has connected to the channel. Whenever a new sender connects to the channel it starts a new thread to serve that sender. In a parallel thread, the frames are then sent to the receiver.

- This method checks if a new connection is accepted.

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

- This function sends the frames to the receiver.

```
# Function to send to receiver
def send_to_receiver(receive):

    while(True):
        # If buffer not empty send the frame and clear buffer
        if(len(co.shared_buffer)>0):
            # Send the frame
            print('Sending frame to receiver')
            time.sleep(4)
            co.send_frame(co.shared_buffer[0], receive)
            del co.shared_buffer[0]
        else:
            continue
```

- This function receives data from sender, at each sender in a separate threads.

```
# Function to receive data from sender
def receive_from_sender(c, addr):
    # Receive data from the sender and keep it in stored buffer
    print('Started new connection to '+str(addr))

    while(True):
        # Receive data from sender
        frame=c.recv(1024).decode()
        time.sleep(2)
        co.shared_buffer.append(frame)
        print(co.shared_buffer)
```

- This function continuously sends every sender the signal whether the channel is busy or not.

```
def send_signal(s, saddr):
    signal=0

    while(True):
        if(len(co.shared_buffer)>=1): # Channel is busy
            signal=1
        else: # Channel not busy
            signal=0

        # Send the signal via socket
        co.send_frame(str(signal), s)

new_con()
```

➔ Persistent Technique

- 1-persistent CSMA : The sender medium_sense() continuously senses the channel to check its state i.e. idle or busy so that it can transfer data or not. In case when the channel is busy, the sender will wait for the channel to become idle. When sender found idle channel, it transmits the frame to the channel without any delay. It transmits the frame with probability 1.
- non-persistent CSMA : when a transmitting the station has a frame to send and it senses a busy channel, it waits for a random period of time without sensing the channel in the interim, and repeats the algorithm again.
- p-persistent : when a transmitting station has a frame to send and it senses a busy channel, it waits for the end of the transmission, and then transmits with a probability p. Since, it sends with a probability p.

➔ *1-persistent CSMA*

- From 1_sender.py, this method takes as its parameter a list of frames and sends all the frames to the channel. It checks whether the channel is busy by checking the busy_channel flag and follows according to the 1-persistent technique to send the frames.

```
# Function to send frames
def send_frame(frames_list):

    i = 0
    while(True):
        # Sense the channel and check if flag is 1 then dont send
        time.sleep(2)
        if(busy_channel == 0):  # Channel is free
            print('Sending frame '+str(i))
            co.send_frame(frames_list[i], s_send)
            if(frames_list[i] != '_'):
                i = i+1
            else:
                i = 0

        else:  # Channel is busy
            print('Channel : Busy State')
            continue
```

➔ Non-persistent CSMA

- From non-sender.py, this method takes as its parameter a list of frames and sends all the frames to the channel. It checks whether the channel is busy by checking the busy_channel flag and gives an ideal time and so of course, it follows according to the non-persistent technique to send the frames.

```
# Function to send frames
def send_frame(frames_list):

    i = 0
    while(True):
        # Sense the channel and check if flag is 1 then dont send
        time.sleep(2)
        if(busy_channel == 0):  # Channel is free
            print('Sending frame :'+str(i))
            co.send_frame(frames_list[i], s_send)
            if(frames_list[i] != '_'):
                i = i+1
            else:
                i = 0

        else:  # Channel is busy
            idealTime = random.randint(1, 5)
            print('Channel busy: Sleeping for {} seconds'.format(idealTime))
            time.sleep(idealTime)
            continue
```

➔ P-persistent CSMA

- From p-sender.py, this method takes as its parameter a list of frames and sends all the frames to the channel. It checks whether the channel is busy by checking the busy_channel flag and gives the time slot for that. However, it follows according to the p-persistent technique to send the frames.

```
# Function to send frames
def send_frame(frames_list):

    i=0
    while(True):
        # Sense the channel and check if flag is 1 then dont send
        if(busy_channel==0): # Channel is free
            # Send the frame with a probability p
            pr=random.randint(0,p_prob)
            if(pr<=p):
                # Send the frame
                print('Sending frame '+str(i))
                co.send_frame(frames_list[i], s_send)
                if(frames_list[i]!='_'):
                    i=i+1
                else:
                    i = 0
                time.sleep(3)
            else:
                print('Waiting for '+str(timeSlot))
                time.sleep(timeSlot)
                continue

        else: # Channel is busy
            print('Channel : Busy')
            time.sleep(2)
            continue
```

- This function continuously receives a signal from the channel process which is an indication of whether the channel is busy or not and accordingly sets the busy_channel flag. This function is run as a separate thread in the program.

```
# Function to sense the medium
def sense_medium():
    global busy_channel

    while(True):
        if(s_signal.recv(1024).decode() == '1'):
            busy_channel = 1 #busy channel
        else:
            busy_channel = 0 #not busy
```

- Main Thread

```
frames_list = co.readfile('input.txt', co.frame_size)
frames_list.append('_')

 # create the sending thread
sendThread = threading.Thread(target=send_frame, args=(frames_list,))

# create the sending thread
senseThread = threading.Thread(target=sense_medium)

sendThread.start()
senseThread.start()

sendThread.join()
senseThread.join()
```

➔ Receiver.py

- This function receives frames from the channel process via a socket.

```
# Function to receive a frame
def receive_frame():

    while(True):
        # Receive the frame
        frame=s_rec.recv(1024).decode()
        print('Frame received '+frame)

receive_frame()
```

- **RESULTS**

  The throughput here was measured in terms of the attempts it took to send the entire data. With increase in p the number of collisions increased as more station tried to send data simultaneously. However it also sometimes decreased due to immediate sending of data.

- **ANALYSIS**

  It may have some possible bugs 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 0's so that it is a multiple of the frame size. Currently the program works only for multiple sender and single receiver processes but the program may be modified to work with multiple sender and multiple receiver processes.

- **COMMENTS**

  The lab assignment was a great learning experience as I have walkthrough the implement of one of the most well-known CSMA protocol ourselves. The assignment can be rated as moderately difficult to extremely high.