NAME: Debadrita Roy

CLASS: BCSE-III

GROUP: A1

ASSIGNMENT NUMBER: 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.

DEADLINE: 14th September, 2021

DATE OF SUBMISSION: 19th November, 2021

DESIGN

The purpose of the program is to simulate the real-world network environment and design and implement 1-persistent, non-persistent and p-persistent CSMA (Carrier Sense Multiple Access) techniques. The required programs are written in Python 3.

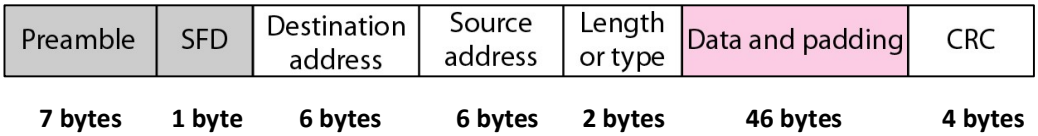
The sender program (**sender_csma.py**) contains codes for the different CSMA techniques. It contains functionality for checking whether the channel is busy or not and proceeds depending upon the CSMA technique chosen. It sends the packets using one of the techniques. If a collision happens, the packet is dropped. The channel program (**channel_csma.py**) contains a list with the connections to the different senders (connection is implemented in the program as pipe) and sends whether it is busy or not to the sender when asked. The channel sends the packets from the sender to the receiver, introducing delay for channel propagation time. In this program, I have used multiple senders and one receiver. The receiver (**receiver_csma.py**) gets the packet from the channel and displays the sequence number of the packet received and stores the packet in a file. Modules imported in the programs include **errordetect.py** (from Assignment 1, for the CRC codes), **constants.py** (to store the different constants common to all the programs, such as number of senders, propagation time, default packet size, vulnerable time, etc), **packet.py** (contains functionality for creating a packet in IEEE 802.3 format and extracts data or other parts like destination address, source address, sequence number, etc or whether there is error in the packet or not) and **collisions.py** (this contains a class storing the number of senders trying to send data at the same time, used for collision detection). The **csma.py** program contains main() for calling the different threads and running them.

1-persistent method: The sender senses the channel **continuously**, if it finds the channel idle, it sends data immediately. We check if a collision happens, if so, the packet is dropped and the collision count is incremented.

Non-persistent method: The sender senses the channel after **random intervals of time**, if it finds the channel idle, it then sends the data immediately. If not, it senses after a random amount of time. We check if a collision happens, if so, the packet is dropped and the collision count is incremented.

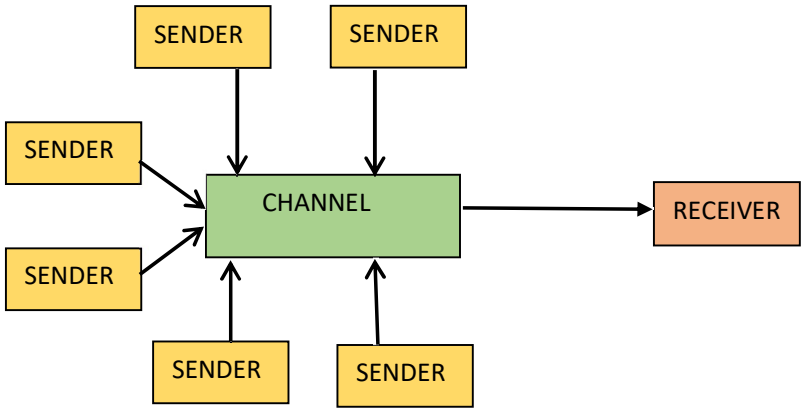
P-persistent method: The sender senses the channel **continuously**, if it finds the channel idle, it sends data with a probability of p, or waits for the next time slot with a probability of q=1-p. We check if a collision happens, if so, the packet is dropped and the collision count is incremented.

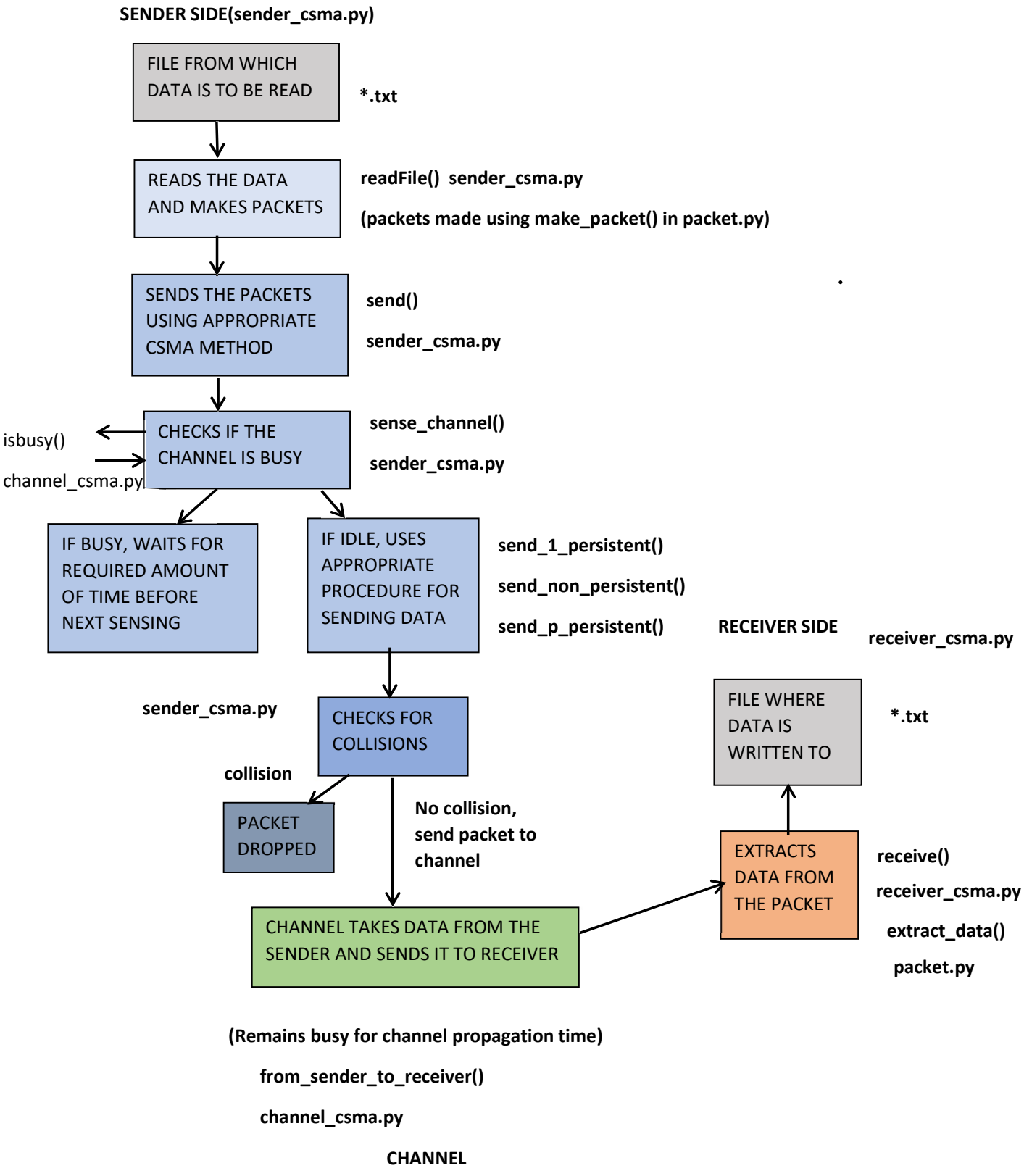
Packet Structure:



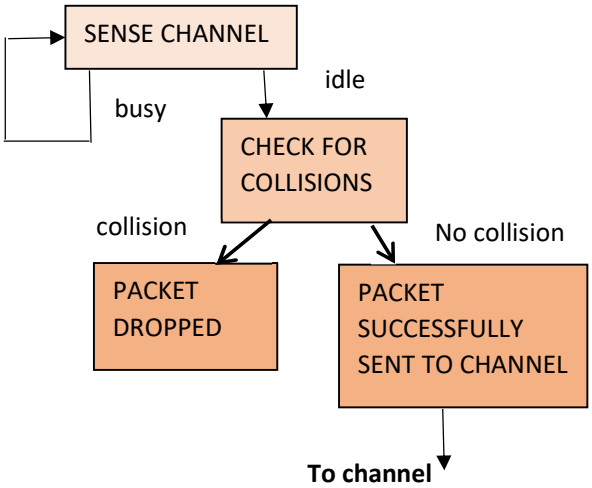
We use the IEEE 802.3 frame format for the packets. Preamble consists of 56 bits of alternating 0's and 1's. SFD is the start frame delimiter, containing 10101011. CRC-32 is used for error detection. Length or type contains 1 byte of length and 1 byte for the sequence number of the packet.

Structure Diagram:

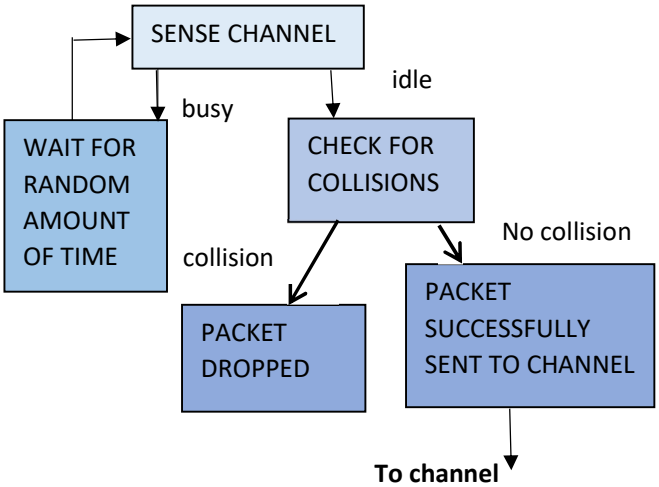



SENDER SIDE(sender_csma.py)
FILE FROM WHICH DATA IS TO BE READ
*.txt
READS THE DATA AND MAKES PACKETS
readFile() sender_csma.py
(packets made using make_packet() in packet.py)
SENDS THE PACKETS USING APPROPRIATE CSMA METHOD
send()
sender_csma.py
CHECKS IF THE CHANNEL IS BUSY
isbusy()
channel_csma.py
sense_channel()
sender_csma.py
IF BUSY, WAITS FOR REQUIRED AMOUNT OF TIME BEFORE NEXT SENSING
IF IDLE, USES APPROPRIATE PROCEDURE FOR SENDING DATA
send_1_persistent()
send_non_persistent()
send_p_persistent()
RECEIVER SIDE
receiver_csma.py
sender_csma.py
CHECKS FOR COLLISIONS
FILE WHERE DATA IS WRITTEN TO
*.txt
collision
PACKET DROPPED
No collision, send packet to channel
EXTRACTS DATA FROM THE PACKET
receive()
receiver_csma.py
extract_data()
packet.py
CHANNEL TAKES DATA FROM THE SENDER AND SENDS IT TO RECEIVER
(Remains busy for channel propagation time)
from_sender_to_receiver()
channel_csma.py
CHANNEL

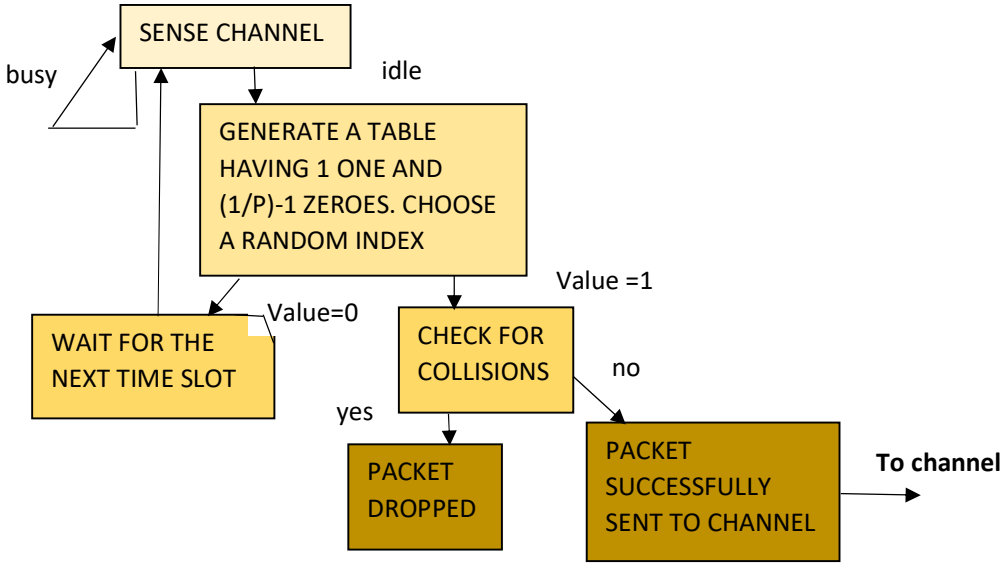
1-PERSISTENT METHOD: **send_1_persistent() in sender_csma.py**



NON-PERSISTENT METHOD: **send_non_persistent() in sender_csma.py**



P-PERSISTENT METHOD: **send_p_persistent(p) in sender_csma.py**



Input Format:

The collision technique to be used is given as input on the terminal. The csma.py sends the input file(s) from which the data is to be read to the sender threads.

Output Format:

The various significant stages in the different threads (packets sent/received/collided) are displayed in the terminal to keep track of the running of the program. The performance of the different senders is stored in a text file performance.txt.

IMPLEMENTATION

Sender_csma.py

The Sender class contains a constructor to initialize the sender details. It contains methods for changing the receiver where the data is to be sent, getting the current receiver, methods for sending a packet using different CSMA techniques, function to read a file and return its contents, function to sense the channel, send() function to send the packets using chosen method and to write the performance details in performance.txt after transmission is complete and communicate() method to start the transmission thread and the channel sensing thread.

def __init__(self, num, filename, tochannel, fromchannel, choice,coll)

Method Description: The constructor initialises the sender number, the file from which data is to be read, pipes for reading from channel and for writing to channel, choice of CSMA technique used and Collision class object for detecting collisions. It initialises the other members of the class with default values.

Code Snippet:

```
def __init__(self, num, filename, tochannel, fromchannel, choice,coll):
    self.num=num
    self.file=filename
    self.receiver=0  # default
    self.tochannel=tochannel
    self.fromchannel=fromchannel
    self.endTransmitting = False
    self.start = 0
    self.seqNo = 0
    self.pktCount = 0
    self.choice = choice
    self.busy = False
    self.collisionCount = 0
    self.collision=coll
    self.succpktCount=0
```

def set_receiver(self,num)

Method Description: This function sets the receiver to something other than the default value.

Code Snippet:

```
def set_receiver(self,num):
    self.receiver=num
```

def get_receiver(self)

Method Description: This function returns the receiver to which data is to be sent.

Code Snippet:

```
def get_receiver(self):
    return self.receiver
```

def readFile(self,filen)

Method Description: It reads the contents of a file and returns them.

Code Snippet:

```
def readFile(self,filen):
    f=open(filen,'r')
    s=f.read()
    f.close()
    return s
```

def send_1_persistent(self,pkt)

Method Description: It sends the packet pkt to the channel using 1-persistent technique.

Code Snippet:

```
def send_1_persistent(self,pkt):
    while True:
        if not self.busy:
            coll=self.readFile('collision.txt')
            if coll=='1':
                self.collisionCount+=1
                print('Sender ',self.num,': Collision happened')
                time.sleep(constants.collisionWaitTime)
                break

            else:
                print('Packet ',(self.pktCount+1),' sent by the sender ',self.num)
                f=open('collision.txt','w')
                f.write('1')
                f.close()
                self.collision.increment()
                time.sleep(constants.vulnerableTime)
                if self.collision.no_senders_currently_sending()>1:
                    print('collision happened')
                    self.collisionCount+=1
                    time.sleep(constants.collisionWaitTime)
                    self.collision.decrement()
                    f = open('collision.txt', 'w')
```

```
                f.write('0')
                f.close()
                break
            self.collision.decrement()
            f = open('collision.txt', 'w')
            f.write('0')
            f.close()

            self.tochannel.send(pkt)
            time.sleep(constants.propagationTime)
            self.succpktCount+=1
            break

        else:
            print('Sender ',self.num,' found the channel busy')
            time.sleep(0.5)
```

def send_non_persistent(self,pkt)

Method Description: It sends the packet pkt to the channel using non-persistent technique.

Code Snippet:

```
    def send_non_persistent(self,pkt):
        while True:
            if not self.busy:
                coll=self.readFile('collision.txt')
                if coll=='1':
                    self.collisionCount+=1
                    print('Sender ',self.num,': Collision happened')
                    time.sleep(constants.collisionWaitTime)
                    break

                else:
                    print('Packet ',(self.pktCount+1),' sent by the sender ',self.num)
                    f=open('collision.txt','w')
                    f.write('1')
                    f.close()
                    self.collision.increment()
                    time.sleep(constants.vulnerableTime)
                    if self.collision.no_senders_currently_sending() > 1:
                        print('collision happened')
                        self.collisionCount += 1
                        time.sleep(constants.collisionWaitTime)
                        self.collision.decrement()
                        f = open('collision.txt', 'w')
                        f.write('0')
                        f.close()
                        break
                    self.collision.decrement()
```

```
                f = open('collision.txt', 'w')
                f.write('0')
                f.close()

                self.tochannel.send(pkt)
                time.sleep(constants.propagationTime)
                self.succpktCount+=1
                break

            else:
                print('Sender ',self.num,' found the channel busy')
                time.sleep(1+int(4*random.random()))   # waits for a random amount of time
```

def send_p_persistent(self,pkt,p=(1/constants.noofsenders))

Method Description: It sends the packet pkt to the channel using p-persistent technique.

Code Snippet:

```
        def send_p_persistent(self,pkt,p=(1/constants.noofsenders)):
            while True:
                if not self.busy:
                    x=int(1/p)
                    list=[0 for _ in range(x)]
                    list[0]=1
                    pr=int(x*random.random())    # chooses an index randomly
                    if pr==1:   # if to be sent
                        coll=self.readFile('collision.txt')
                        if coll=='1':
                            self.collisionCount+=1
                            print('Sender ',self.num,': Collision happened')
                            time.sleep(constants.collisionWaitTime)
                            break

                        else:
                            print('Packet ',(self.pktCount+1),' sent by the sender ',self.num)
                            f=open('collision.txt','w')
                            f.write('1')
                            f.close()
                            self.collision.increment()
                            time.sleep(constants.vulnerableTime)
                            if self.collision.no_senders_currently_sending() > 1:
                                print('collision happened')
                                self.collisionCount += 1
                                time.sleep(constants.collisionWaitTime)
                                self.collision.decrement()
                                f = open('collision.txt', 'w')
                                f.write('0')
                                f.close()
                                break
```

```
                    self.collision.decrement()
                    f = open('collision.txt', 'w')
                    f.write('0')
                    f.close()
                    self.tochannel.send(pkt)
                    time.sleep(constants.propagationTime)
                    self.succpktCount+=1
                    break
                else:
                  print('Sender ',self.num,' waiting for the next timeslot')
                  time.sleep(constants.timeSlot)

            else:
              print('Sender ',self.num,' found the channel busy')
              time.sleep(0.5)
```

def send(self)

Method Description: It sends the data to the channel by making packets and calling the respective CSMA technique. It then stores the results.

Code Snippet:

```
        def send(self):
          self.start=time.time()
          fi=self.readFile(self.file)
          self.seqNo=0
          if len(fi)%constants.defaultDataPacketSize!=0:
            fi=fi+('0'*(constants.defaultDataPacketSize-
      (len(fi)%constants.defaultDataPacketSize)))

          data=fi[0:constants.defaultDataPacketSize]
          k=constants.defaultDataPacketSize
          fi=fi[0:(15*constants.defaultDataPacketSize)]   # for 15 packets
          while data:
            p=packet.make_packet(self.seqNo,data,self.num,self.receiver)
            if self.choice==1:
              self.send_1_persistent(p)
            elif self.choice==2:
              self.send_non_persistent(p)
            else:
              self.send_p_persistent(p)
            self.pktCount+=1
            self.seqNo+=1
            data=fi[k:(k+constants.defaultDataPacketSize)]
            k+=constants.defaultDataPacketSize
          self.endTransmitting=True
          delay=time.time()-self.start
          per=open('performance.txt','a')
          per.write('----------Sender '+str(self.num)+' Performance----------\n')
```

```
per.write('Total number of packets sent: '+str(self.pktCount)+'\n')
per.write('Total delay: '+str(delay)+'\n')
per.write('No. of collisions: '+str(self.collisionCount)+'\n')
per.write('No. of packets successfully sent:'+str(self.succpktCount)+'\n')
per.write('Throughput: '+str(self.succpktCount/delay)+'\n')
```

def sense_channel(self)

Method Description: It checks whether the channel is busy or not.

Code Snippet:

```
def sense_channel(self):
  while True:
    if self.fromchannel.recv()=='1':
      self.busy=True
    else:
      self.busy=False
```

def communicate(self)

Method Description: It starts the two threads for each sender—sending thread and sensing thread.

Code Snippet:

```
def communicate(self):
  sendt=threading.Thread(name="sending",target=self.send)
  cst=threading.Thread(name="sensing",target=self.sense_channel)
  sendt.start()
  cst.start()
  sendt.join()
  cst.join()
```

channel_csma.py

The Channel class contains a constructor to initialize the channel details. It contains functions to send packets from the sender to the receiver, to send signal to the sender whether the channel is busy or not and to start the channel threads.

def __init__(self, fromsender,tosender,fromreceiver,toreceiver)

Method Description: It initializes the channel details.

Code Snippet:

```
def __init__(self,fromsender,tosender,fromreceiver,toreceiver):
  self.fromsender=fromsender
  self.fromreceiver=fromreceiver
  self.tosender=tosender
  self.toreceiver=toreceiver
  self.busy=False
```

def from_sender_to_receiver(self)

Method Description: It sends packets from sender to receiver, introducing delay for channel propagation time. The receiver is selected based on the destination address in the packet(s).

Code Snippet:

```
def from_sender_to_receiver(self):
while True:
   pkt= self.fromsender.recv()
   self.busy=True
   time.sleep(constants.ch_propagationTime)
   self.busy = False
   receiver=packet.get_dest_address(pkt)
   self.toreceiver[receiver].send(pkt)
```

def isbusy(self,sender)

Method Description: It sends 1 to the sender if the channel is busy, 0 otherwise.

Code Snippet:

```
def isbusy(self,sender):
   while True:
      if self.busy:
         self.tosender[sender].send('1')
      else:
         self.tosender[sender].send('0')
```

def startChannel(self)

Method Description: It starts the channel threads—sending packet from sender to receiver and sending busy signal to the sender.

Code Snippet:

```
def startChannel(self):
toreceiverthreads=[]
tosenderthreads=[]
t=threading.Thread(name="Transmission",target=self.from_sender_to_receiver)
toreceiverthreads.append(t)
for i in range(constants.noofsenders):
   s=threading.Thread(name="Sensed"+str(i),target=self.isbusy,args=(i,))
   tosenderthreads.append(s)
for th in tosenderthreads:
   th.start()
for th in toreceiverthreads:
   th.start()
for th in tosenderthreads:
   th.join()
for th in toreceiverthreads:
   th.join()
```

constants.py

It contains the different constants for running the program(s) simultaneously, such as the propagation time, vulnerable time, collision wait time, and so on.

errordetect.py

This module is same as the one done for Assignment 1. It is imported to provide support for making the CRC codeword for the packet as well as detecting whether there is an error in the packet or not.

packet.py

This module contains functionality for making a packet in the IEEE 802.3 frame format, as well as for performing operations on a packet in the IEEE 802.3 frame format. It imports errordetect.py.

def make_packet(seqNo,data,sender,destination)

Method Description: It creates a packet using the sequence number, data, sender and destination addresses.

Code Snippet:

```
def make_packet(seqNo,data,sender,destination):
    preamble = '01' * 28
    sfd = '10101011'
    sequence_bits = '{0:08b}'.format(int(seqNo))
    destination_address = '{0:048b}'.format(int(destination))
    length = '{0:008b}'.format(len(data))
    source_address = '{0:048b}'.format(int(sender))
    if len(data)<(46*8):
        data=data+'0'*(46*8-len(data))
    pkt = preamble + sfd + destination_address + source_address + sequence_bits + length +
data
    pkt = errordetect.crc(pkt,'100000100110000010001110110110111')
    return pkt
```

def extract_data(p)

Method Description: It returns the data from the packet, p, passed as argument.

Code Snippet:

```
def extract_data(p):
    data = p[176:544]
    l=int(p[168:176],2)
    return data[0:l]
```

def get_dest_address(p)

Method Description: It returns the destination address of the packet, p, passed as argument.

Code Snippet:

```
def get_dest_address(p):
    d = p[64:112]
```

```
    return int(d,2)
```

def get_src_address(p)

Method Description: It returns the source address of the packet, p, passed as argument.

Code Snippet:

```
def get_src_address(p):
    s = p[112:160]
    return int(s,2)
```

def is_error_free(p)

Method Description: It returns True if the packet does not contain any error, else returns False.

Code Snippet:

```
def is_error_free(p):
    if errordetect.detect_error_crc(p,'10000010011000001000111011011011 1'):
        return False
    else:
        return True
```

def get_seq_no(p)

Method Description: It returns the sequence number of the packet, p, passed as argument.

Code Snippet:

```
def get_seq_no(p):
    sno = p[160:168]
    return int(sno, 2)
```

collisions.py

This class enables us to find out whether collisions of the type where more than one sender tries to engage the channel at the same time after finding the channel not busy. It increments the number of senders to write in collision.txt when the sender tries to send and decrements the number of senders when the sender is done sending.

Code Snippet:

```
class Collision:
    def __init__(self):
        self.count=0

    def increment(self):
        self.count+=1

    def decrement(self):
        self.count-=1

    def no_senders_currently_sending(self):
        return self.count
```

receiver_csma.py

The Receiver class initializes the receiver details and receives packet (s) from the channel, extracts data and writes it in the respective file for each sender.

def __init__(self,num,fromchannel)

Method Description: It initializes the receiver details.

Code Snippet:

```
def __init__(self,num,fromchannel):
  self.num=num
  self.fromchannel=fromchannel
```

Method Description: It receives packet(s) from the channel, extracts data and writes it into the respective file.

Code Snippet:

```
def receive(self):
  while True:
    pkt=self.fromchannel.recv()
    s=packet.get_src_address(pkt)
    f=open('received'+str(s)+'.txt','w')
    f.write(packet.extract_data(pkt))
    f.close()
    print('Received packet ',packet.get_seq_no(pkt)+1,' from sender ',s)
```

csma.py

It contains the main() function which is used to create objects for the different classes and call the respective functions to run the program. This is run in the terminal for checking the program and getting output(s). It takes input from the user as to the CSMA technique to be used in the program and sets up the connections between the sender(s), channel, and receiver and starts the threads.

TEST CASES

Sample Test 1: To check the working of the pipes (whether packets are being sent or not)

For this, we take a single sender and receiver and see whether all packets are being sent and received (no collisions as of yet) using 1-persistent technique.

```
C:\Users\USER19\PycharmProjects\python
1. 1-persistent method
2. Non-persistent method
3. p-persistent method
Enter your choice(1-3):1
Packet  1  sent by the sender  0
Received packet  1  from sender  0
Packet  2  sent by the sender  0
Received packet  2  from sender  0
Packet  3  sent by the sender  0
Received packet  3  from sender  0
Packet  4  sent by the sender  0
Received packet  4  from sender  0
Packet  5  sent by the sender  0
Received packet  5  from sender  0
Packet  6  sent by the sender  0
Received packet  6  from sender  0
Packet  7  sent by the sender  0
Received packet  7  from sender  0
Packet  8  sent by the sender  0
Received packet  8  from sender  0
Packet  9  sent by the sender  0
Received packet  9  from sender  0
Packet  10  sent by the sender  0
Received packet  10  from sender  0
Packet  11  sent by the sender  0
Received packet  11  from sender  0
Packet  12  sent by the sender  0
Received packet  12  from sender  0
Packet  13  sent by the sender  0
```

```
Packet  12  sent by the sender  0
Received packet  12  from sender  0
Packet  13  sent by the sender  0
Received packet  13  from sender  0
Packet  14  sent by the sender  0
Received packet  14  from sender  0
Packet  15  sent by the sender  0
Received packet  15  from sender  0
```

Sample Test 2: To check the working of p-persistent technique (whether waiting for number of slots or not)

```
C:\Users\USER19\PycharmProjects\python_assignr
1. 1-persistent method
2. Non-persistent method
3. p-persistent method
Enter your choice(1-3):3
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Packet  1  sent by the sender  0
Received packet  1  from sender  0
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Packet  2  sent by the sender  0
Received packet  2  from sender  0
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
```

```
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Packet  3  sent by the sender  0
Received packet  3  from sender  0
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Packet  4  sent by the sender  0
Received packet  4  from sender  0
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Packet  5  sent by the sender  0
Received packet  5  from sender  0
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
```

```
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Packet  6  sent by the sender  0
Received packet  6  from sender  0
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Packet  7  sent by the sender  0
Received packet  7  from sender  0
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
```

```
Received packet  7  from sender  0
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Packet  8  sent by the sender  0
Received packet  8  from sender  0
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Sender  0  waiting for the next timeslot
Packet  9  sent by the sender  0
Received packet  9  from sender  0
```

RESULTS

We have to evaluate the performance of the CSMA techniques in comparison to one another. For this, we take a dataset consisting of random 0's and 1's for using as the data to be sent and vary the number of senders in constants.py before running as well as the value of p in p-persistent technique.

Performance metrics for evaluation: All the techniques are utilized one by one, randomly, with different number of senders, at different times (so that system variables are also random). We measure the total number of packets sent, delay, number of collisions, successful number of packets sent for each sender and find the average. We also find the throughput as (number of packets successfully sent/delay) for each sender and consider the average for analysis purposes.

Observation Tables:

Each observation percentage is correct to two decimal places.

TOTAL NUMBER OF PACKETS: 15 PER SENDER

NUMBER OF SENDERS (N): 2

CSMA TECHNIQUE	AVERAGE DELAY (in s)	AVERAGE NO. OF COLLISIONS	AVERAGE THROUGHPUT (packets/s)
NON-PERSISTENT	16.88	0	0.89
0.1-PERSISTENT	43.24	0	0.35
0.25-PERSISTENT	25.61	1	0.56
0.5-PERSISTENT	20.14	1	0.7
1/N-PERSISTENT	20.14	1	0.7
1-PERSISTENT	10.18	7.5	0.44

NUMBER OF SENDERS (N): 4

CSMA TECHNIQUE	AVERAGE DELAY (in s)	AVERAGE NO. OF COLLISIONS	AVERAGE THROUGHPUT (packets/s)
NON-PERSISTENT	34.25	0.5	0.43
0.1-PERSISTENT	51.55	1	0.29
0.25-PERSISTENT	27.72	3	0.45
0.5-PERSISTENT	25.61	3	0.46
1/N-PERSISTENT	27.72	3	0.45
1-PERSISTENT	10.18	7.5	0.16

NUMBER OF SENDERS (N): 6

CSMA TECHNIQUE	AVERAGE DELAY (in s)	AVERAGE NO. OF COLLISIONS	AVERAGE THROUGHPUT (packets/s)
NON-PERSISTENT	53.56	0.67	0.27
0.1-PERSISTENT	51.12	2.33	0.25
0.25-PERSISTENT	40.96	3	0.30
0.5-PERSISTENT	50.59	2.33	0.27
1/N-PERSISTENT	50.48	1.5	0.27
1-PERSISTENT	43.44	1	0.27

NUMBER OF SENDERS (N): 8

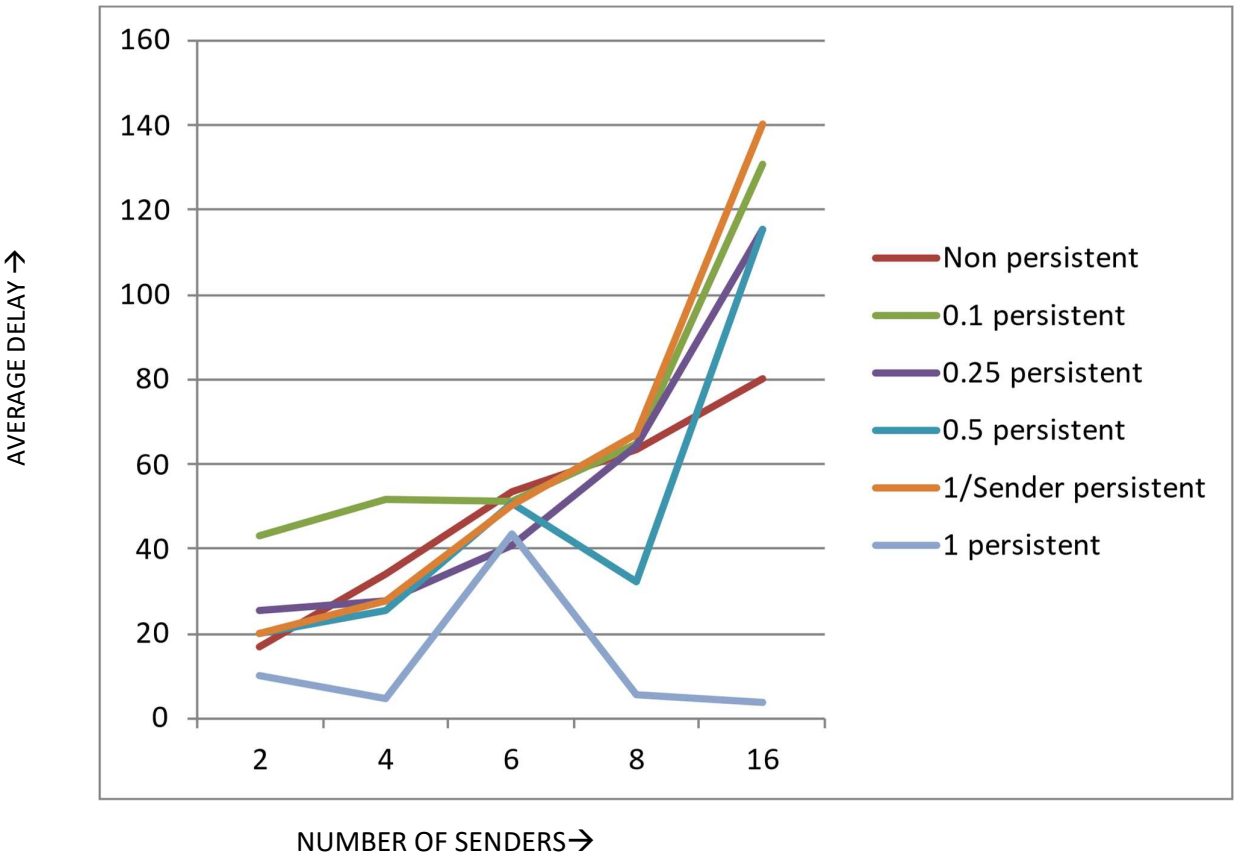
CSMA TECHNIQUE	AVERAGE DELAY (in s)	AVERAGE NO. OF COLLISIONS	AVERAGE THROUGHPUT (packets/s)
NON-PERSISTENT	63.55	2	0.20
0.1-PERSISTENT	64.86	2	0.19
0.25-PERSISTENT	64.55	2.62	0.19
0.5-PERSISTENT	32.37	6.5	0.25
1/N-PERSISTENT	67.23	1.75	0.19
1-PERSISTENT	5.37	12.12	0.52

NUMBER OF SENDERS (N): 16

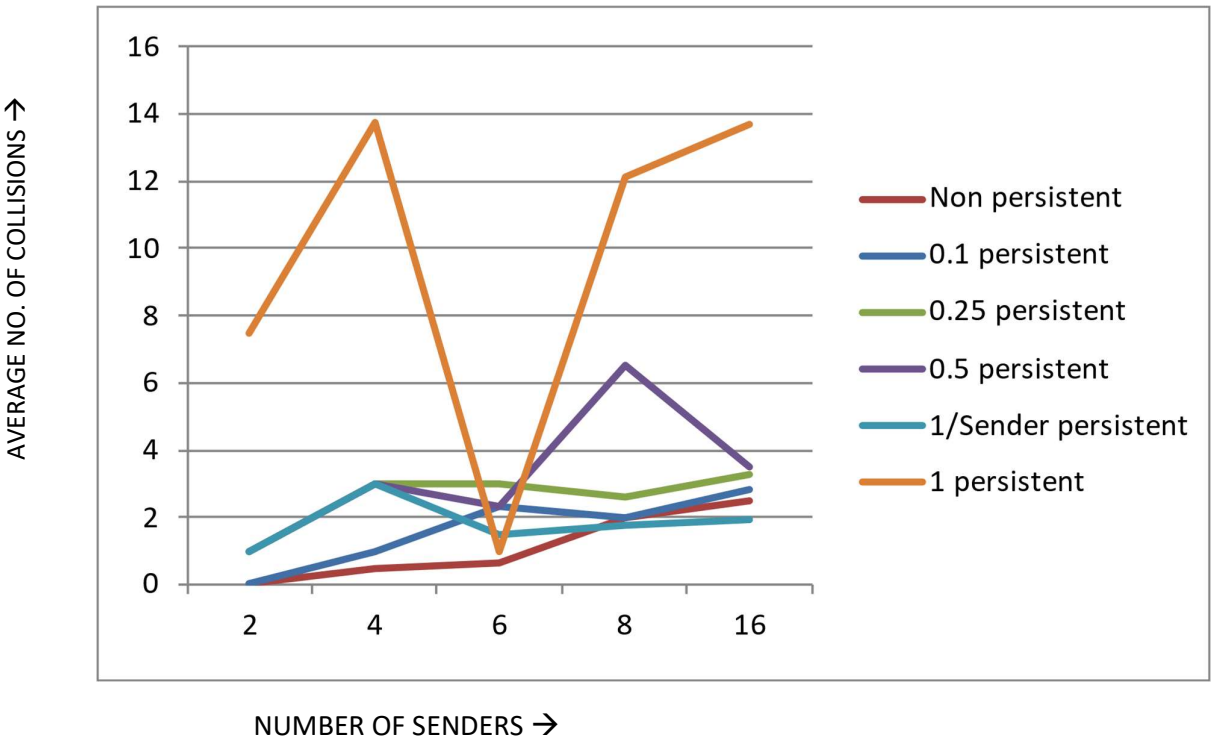
CSMA TECHNIQUE	AVERAGE DELAY (in s)	AVERAGE NO. OF COLLISIONS	AVERAGE THROUGHPUT (packets/s)
NON-PERSISTENT	80.34	2.5	0.14
0.1-PERSISTENT	130.61	2.81	0.09
0.25-PERSISTENT	115.48	3.25	0.1
0.5-PERSISTENT	115.50	3.5	0.1
1/N-PERSISTENT	140.24	1.93	0.09
1-PERSISTENT	3.90	13.69	0.25

Graphs:

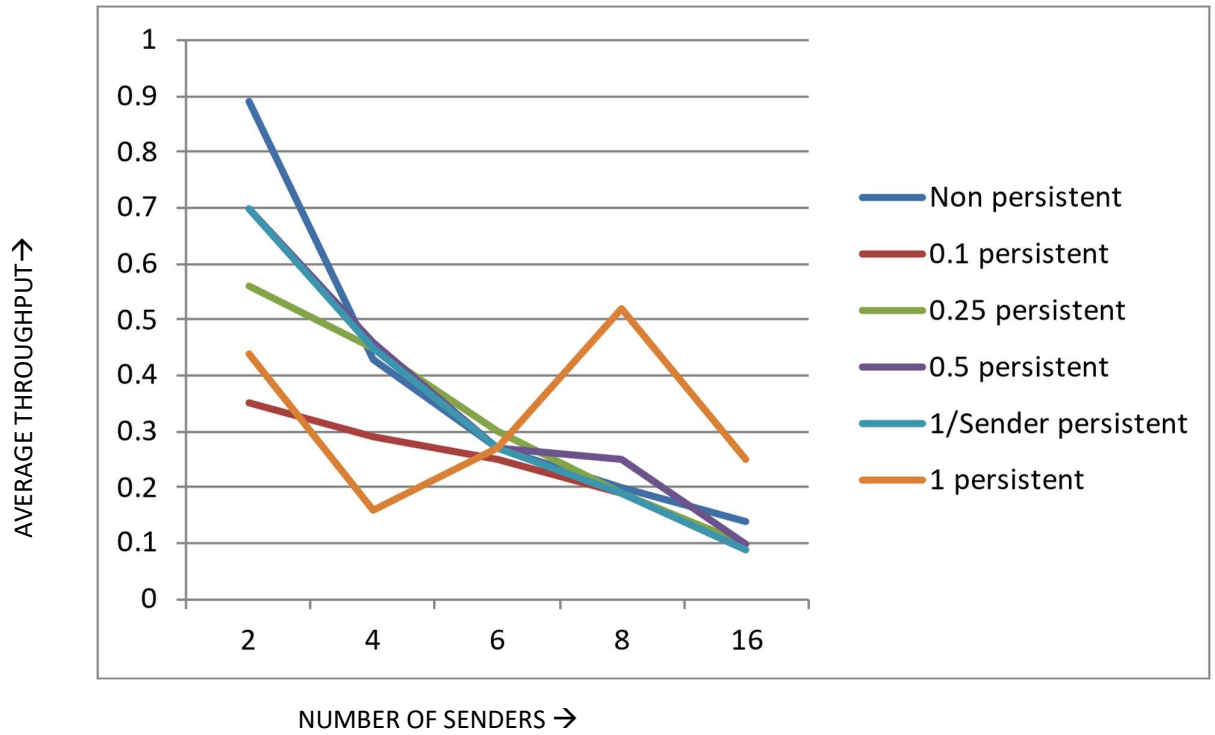
Average Delay vs Number of Senders



Average number of collisions vs Number of senders graph



Average Throughput vs Number of senders graph



ANALYSIS

From the test cases and observations, we find out that (1/sender)-persistent technique is the most efficient algorithm generally, although in specific cases it is outstripped by others, and is the most reliable due to uniformly low number of collisions. Non-persistent technique also results in fewer collisions; however, its channel utilization is low. Delay is greater in non-persistent technique, so throughput is less. 1-persistent technique results in the least amount of delay and highest channel utilization, however it also results in higher number of collisions. So, it is fast but not a reliable technique as there is higher probability of collision of the packets. If we have to use 1-persistent technique reliably, we have to supplement it with a proper flow control mechanism so that the dropped packets can be retransmitted. Non-persistent technique can be used when the delay is not an important consideration as it has low number of collisions even when the number of senders increases. When the value of p is less, the number of collisions decreases but the delay increases, so channel utilization also decreases. When the number of senders is large, (1/sender)-persistent technique can result in huge delay so might not be the best option. For those cases, 0.1-persistent technique can be used. 0.25 or 0.5-persistent techniques with flow control mechanism may also be considered for reliable transmission in those cases. P-persistent techniques in general have better channel utilization than non-persistent technique but have more delay.

Possible Improvements: With the program as it is now, there is no flow control mechanism for retransmission if packets collide. Collision only results in the packet getting dropped. The packets that were dropped can be retransmitted if ACK is not received from the receiver end. Also the channel as of now has been considered noiseless, but that is not the case for real life. Error detection must be included at the receiver end. Also, I have used only one receiver in the program, but multiple receivers can easily be connected if there is a list of receivers not unlike the list of senders. Also, collision detection or collision avoidance algorithms may also be implemented.

COMMENTS

The lab assignment was interesting as it allowed us to see the different advantages and limitations of each CSMA technique in a practical manner and allowed us to verify the results we have read about in theory. I found the analysis of the different observations very engaging. Writing the programs was difficult after figuring out the design and the algorithms, as there were issues regarding thread handling as well as the detection of collisions. More than often a scenario would arise where two senders were executing the same piece of code at the same time leading to non-detection of collisions (so collisions.py came into being). Comparing the performance of different p-persistent techniques (different values of p) is a possibility. Collision Detection or collision avoidance can also be considered.