Name: Sayan Acharya

Roll: 002010501009

Group: A1

Assignment: 3

Deadline: 12th -16th September 2022

Submission date: 13th September

Assignment: 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.

Code:

```
        from itertools import count, cycle
import random

max_time=1000
max_wait_time=30
max_send_time=5
max_loc=1000
permissible_dist=20

station_no=6
stations=list(random.sample(range(1,max_loc+1),k=station_no))
stations.sort()
back_off=[None]*station_no # (time_to_start_sending,dest,sending_time)

data=set()#tuple<int,int,int,int> # (source,destination,t_start,t_end)

class persistent_1:# pair<tuple<int,int,int,int>,tuple<int,int,int>>
  def __init__(self) -> None:
    pass
  def func(self,any_true,data_to_send,t):
    if not any_true:
      return (data_to_send,None)
    else:
      return (None,(t+1,data_to_send[1],data_to_send[3]-data_to_send[2]))

class persistent_0:
  def __init__(self) -> None:
    pass
  def func(self,any_true,data_to_send,t):
    if not any_true:
      return (data_to_send,None)
    else:
      return
(None,(t+random.randint(1,max_wait_time),data_to_send[1],data_to_send[3]-data_to_send[2]))
```

```
class persistent_p:
  def __init__(self,p) -> None:
    self.p=p
  def func(self,any_true,data_to_send,t):
    if not any_true:
      val=random.randint(0,100)/100
      if val<=self.p:#send
        return (data_to_send,None)
      else:#wait random
        return
(None,(t+random.randint(1,max_wait_time),data_to_send[1],data_to_send[3]-data_to_send[2]))
    else:
      return (None,(t+1,data_to_send[1],data_to_send[3]-data_to_send[2]))

def calculate_positions(data,t):
  listy=[]
  for i in data:
    velo=(i[1]-i[0])/(i[3]-i[2])
    cur_pos=i[0]+velo*(t-i[2])
    listy.append(cur_pos)
  return listy

def is_collision(pos1,pos2):
  return abs(pos1-pos2)<=permissible_dist

# persistence_method=persistent_p(1/station_no)
# persistence_method=persistent_0()
persistence_method=persistent_1()

data_generated=0
data_sent=0
#for(int t=0;;t++)
for t in count(0,1):
  print(f"at t:{t}")
  if t>10*max_time:
    break
  if t>max_time and len(data)==0 and not any(back_off):
    break

  #take out the data that has already reached
  new_data=set(i for i in data if i[-1]>t)
  data_sent+=len(data)-len(new_data)
  data=new_data
```

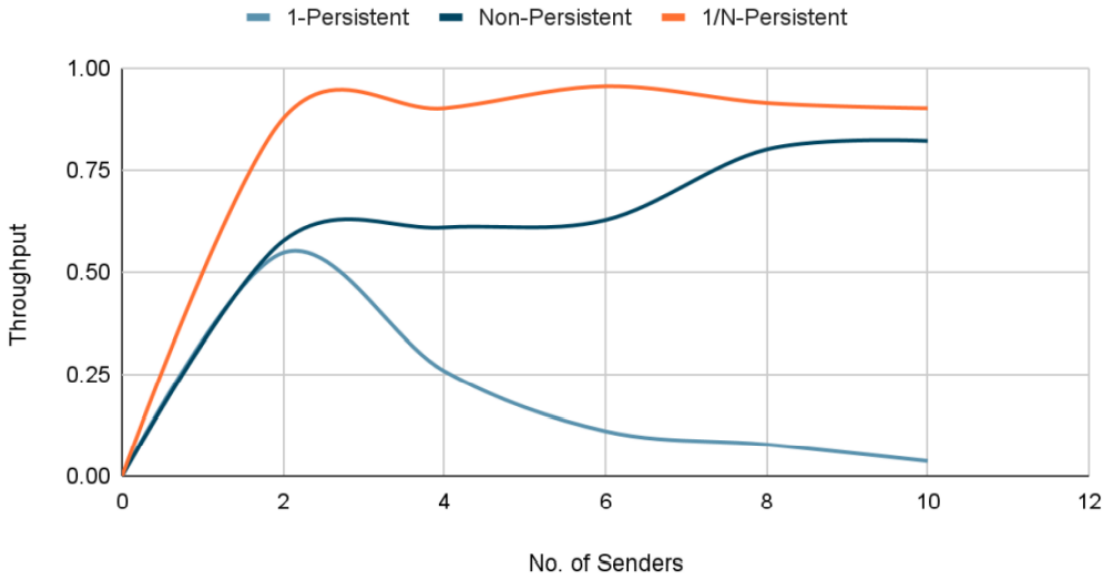
```
  #randomly some stations will send the data
  for i in range(station_no):
    data_to_send=None
    if back_off[i] and back_off[i][0]<=t:
      data_to_send=(i,back_off[i][1],t,t+back_off[i][2])
    else:
      if t>max_time or random.randint(0,1)!=0:
        continue
      j=random.choice(range(station_no))
      data_generated+=1
      time_needed_to_send=random.randint(1,max_send_time)
      data_to_send=(i,j,t,t+time_needed_to_send)

    #add your scheme
    positions_of_data=calculate_positions(data,t)
    any_true=any(is_collision(stations[i],pos) for pos in positions_of_data)

    tup=persistence_method.func(any_true,data_to_send,t)
    if tup[0]:# send
      print(f"st{i} sending data to st{j} at {t}")
      data.add(tup[0])
      back_off[i]=None
    else:# hold
      print(f"st{i} backing off to {tup[1][0]}")
      back_off[i]=tup[1]

  #see if there is a collision. if yes purge all the data
  if len(data)>1:
    print(f"collision detected at {t}. purging data")
    data=set()
    #data.clear()

print(f"{data_generated} {data_sent} {data_sent*100/data_generated}")
```

Design:

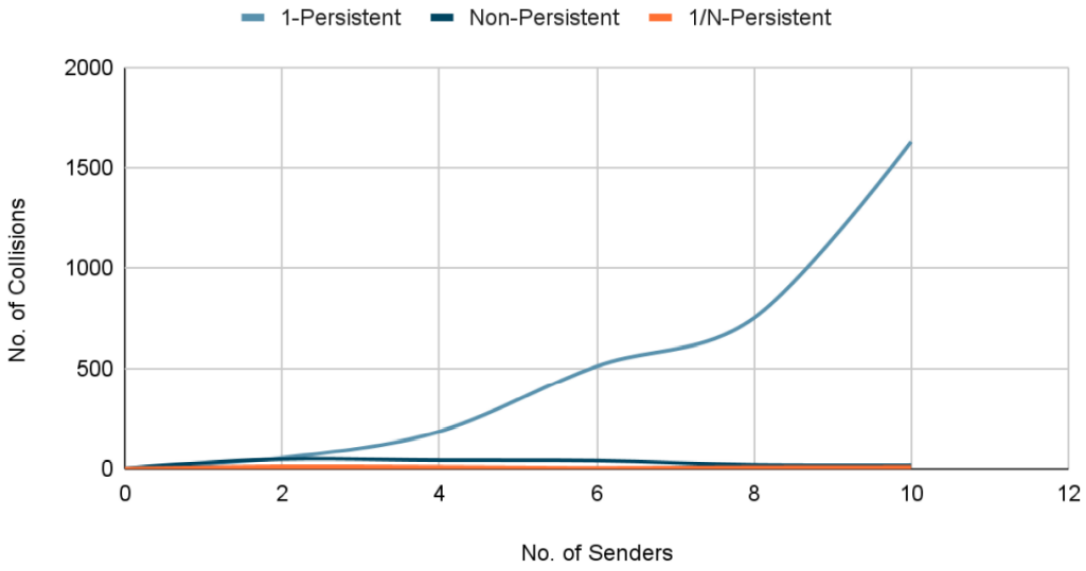
There is a channel and sender-receiver stations. The sender sends the data and if there is a collision then that data is purged and if there is no collision then that data is simply accepted. For control mechanism, various persistent algorithms are used like: 1 persistent, non persistent, p persistent algorithm.

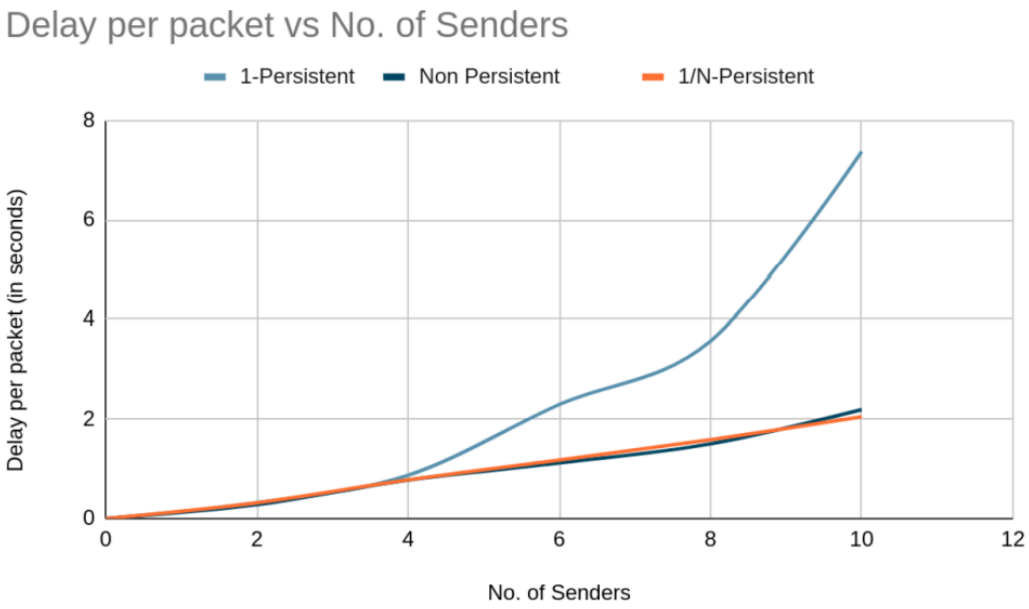
Graph:

Throughput vs No. of Senders



Collisions vs No. of Senders





ANALYSIS:

Collision In the 1-Persistent method, a frame is transmitted immediately after it senses the channel idle, it has the maximum chances of collision. When the number of senders increases, no. of collisions increases exponentially. In the Non-Persistent method, it waits for a random time when the channel is found to be busy. An average number of collisions remains almost the same with a slight increase with an increasing no. of senders, since the random waiting range also increases, hence reducing the chances of sensing the idle channel simultaneously. In the P-Persistent method, whenever it senses an idle channel, it generates a random value which must be less than p(1/no. of senders) to transmit the frame, else waits for a time, and tries again. It is unlikely for different senders to get in the same slot, which reduces the collision probability.

Average number of collisions remains almost same with a slight increase with increasing number of senders as value of p decreases too. Throughput In the 1-Persistent method, since no. of collisions increases with an increase in no. of senders, throughput decreases. In the Non-Persistent method, throughput increases slowly up to a certain point and then saturates. The P-Persistent method provides the best throughput. Throughput is greater than the other two methods and remains almost saturated at all times.

Average Delay per Packet: In the 1-Persistent method, since no. of collisions increases exponentially with an increase in no. of senders, delay per packet also increases exponentially. In the Non-Persistent method, with increasing no. of senders, delay per packet also increases linearly. In the P-Persistent method too, delay per packet increases linearly with the increase in no. of senders. Among all of the methods, the P-Persistent method with probability = 1/N, where N = no. of senders is the most efficient.

COMMENTS: Since the receiver sends an acknowledgment, which is also a form of the data packet, and the receiver is also a station, this assignment can be extended further such that, both the sender and the receiver follow the persistent methods.