Computer Networks Lab Report – Assignment 3

# TITLE

**Name – Aditya Ganguly**

**Roll – 002010501019**

**Class – BCSE 3rd year**

**Group – A1**

**Number – 3**

**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.**

**Design**

The system has been modelled as a number of threads or nodes trying to access a single shared resource, the channel. The channel is a lock

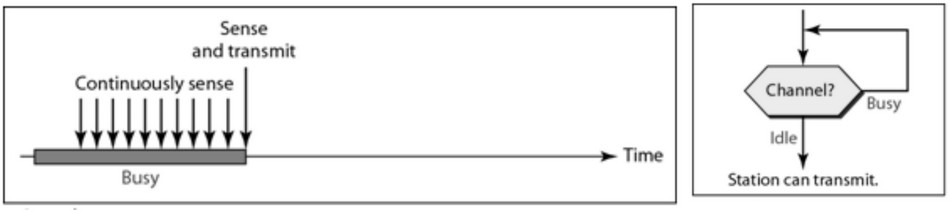
There are three files.

1. 1\_persistent.py – Implementation of 1 persistent CSMA
2. non\_persistent.py – Implementation of non persistent CSMA
3. p\_persistent.py – Implementation of p-persistent CSMA

**IMPLEMENTATION**

1-Persistent: In 1-persistent CSMA, the station 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 station will wait for the channel to become idle. When the station found an idle channel, it transmits the frame to the channel without any delay. It transmits the frame with probability

1. Due to probability 1, it is called 1-persistent CSMA.



import threading

import random

import time

frameTime = 3

interFrameTime =1

numFrames = 10

totalFrames = 0

def metrics(lock, total):

global totalFrames

tot = 0

used = 0

while totalFrames < total:

if lock.locked():

used += 1

tot += 1

print(f"Channel utilization is {used/tot}")

print()

class CSMA(threading.Thread):

def \_\_init\_\_(self,lock,k,index):

super().\_\_init\_\_()

self.lock=lock

self.k=k

self.index=index

def run(self):

cnt=1

global numFrames

global totalFrames

while cnt <= numFrames:

print(f"Attempting to send frame {cnt} of node {self.index}")

print()

while self.lock.locked():

pass

self.lock.acquire()

time.sleep(frameTime)

print(f"Successfully sent frame {cnt} of node {self.index}")

print()

self.lock.release()

totalFrames += 1

time.sleep(interFrameTime)

cnt+=1

return

if \_\_name\_\_=='\_\_main\_\_':

numberNodes = int(input("Enter number of nodes: "))

lock = threading.Lock()

met = threading.Thread(target=metrics, args=[lock, numberNodes\*numFrames])

Nodes = [CSMA(lock,4,i+1) for i in range(0,numberNodes)]

met.start()

for node in Nodes:

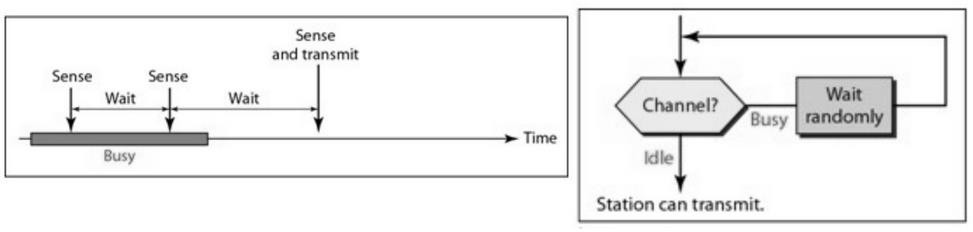
node.start()

for node in Nodes:

node.join()

met.join()

**Non-persistent:** In this method, the station that has frames to send, only that station senses for the channel. In case of an idle channel, it will send a frame immediately to that channel. In case when the channel is found busy, it will wait for the random time and again sense for the state of the station whether idle or busy. In this method, the station does not immediately sense the channel for only the purpose of capturing it when it detects the end of the previous transmission. The main advantage of using this method is that it reduces the chances of collision. The problem with this is that it reduces the efficiency of the network



import threading

import random

import time

import threading

import random

import time

frameTime = 3

interFrameTime =1

numFrames=10

totalFrames=0

def metrics(lock,total):

global totalFrames

tot=0;used=0;

while totalFrames<total:

if lock.locked():

used+=1

tot+=1

print(f"Channel utilization is {used/tot}")

print()

class CSMA(threading.Thread):

def \_\_init\_\_(self,lock,k,index):

super().\_\_init\_\_()

self.lock=lock

self.k=k

self.index=index

def run(self):

global numFrames

global totalFrames

cnt=1

while cnt<=numFrames:

print(f"Attempting to send frame {cnt} of node {self.index}")

print()

while self.lock.locked():

backOffTime=random.randint(2,5)

print(f"node {self.index} waiting for time {backOffTime}")

print()

time.sleep(backOffTime)

self.lock.acquire()

time.sleep(frameTime)

print(f"Successfully sent frame {cnt} of node {self.index}")

print()

self.lock.release()

totalFrames+=1

time.sleep(interFrameTime)

cnt+=1

return

if \_\_name\_\_=='\_\_main\_\_':

numberNodes = int(input("Enter number of nodes: "))

lock = threading.Lock()

met = threading.Thread(target=metrics,args=[lock,numberNodes\*numFrames])

Nodes = [CSMA(lock,4,i+1) for i in range(0,numberNodes)]

met.start()

for node in Nodes:

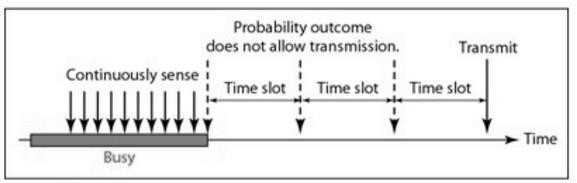
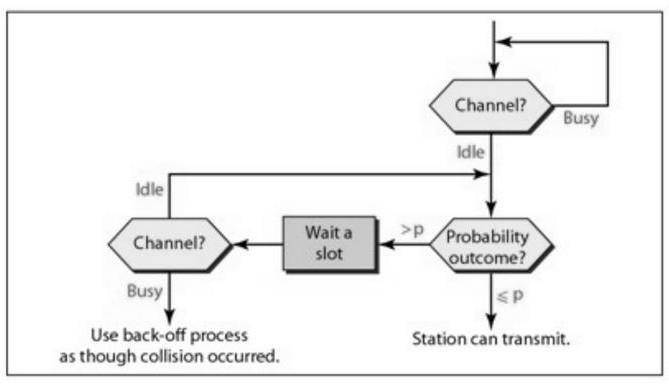
node.start()

for node in Nodes:

node.join()

met.join()

**p-Persistent**: This is the method that is used when the channel has time-slots and that time- slot duration is equal to or greater than the maximum propagation delay time. When the station is ready to send the frames, it will sense the channel. If the channel is found to be busy, the channel will wait for the next slot. If the channel is found to be idle, it transmits the frame with probability p, thus for the left probability i.e. q which is equal to 1-p the station will wait for the beginning of the next time slot. In case, when the next slot is also found idle it will transmit or wait again with the probabilities p and q. This process is repeated until either the frame gets transmitted or another station has started transmitting.



import threading

import random

import time

frameTime = 3

interFrameTime = 1

backOffTime=2

numFrames = 10

totalFrames = 0

def metrics(lock, total):

global totalFrames

tot = 0

used = 0

while totalFrames < total:

if lock.locked():

used += 1

tot += 1

print(f"Channel utilization is {used/tot}")

print()

class CSMA(threading.Thread):

def \_\_init\_\_(self, lock, k, index,prob):

super().\_\_init\_\_()

self.lock = lock

self.k = k

self.index = index

self.prob=prob

def run(self):

global numFrames

global totalFrames

cnt = 1

while cnt <= numFrames:

print(f"Attempting to send frame {cnt} of node {self.index}")

print()

while self.lock.locked():

pass

decision = random.random()

while decision>self.prob:

print(f"Node {self.index} backing off")

print()

time.sleep(backOffTime)

while self.lock.locked():

pass

decision=random.random()

self.lock.acquire()

time.sleep(frameTime)

print(f"Successfully sent frame {cnt} of node {self.index}")

print()

self.lock.release()

totalFrames += 1

time.sleep(interFrameTime)

cnt += 1

return

if \_\_name\_\_ == '\_\_main\_\_':

numberNodes = int(input("Enter number of nodes: "))

lock = threading.Lock()

met = threading.Thread(target=metrics, args=[lock, numberNodes\*numFrames])

Nodes = [CSMA(lock, 4, i+1,0.95) for i in range(0, numberNodes)]

met.start()

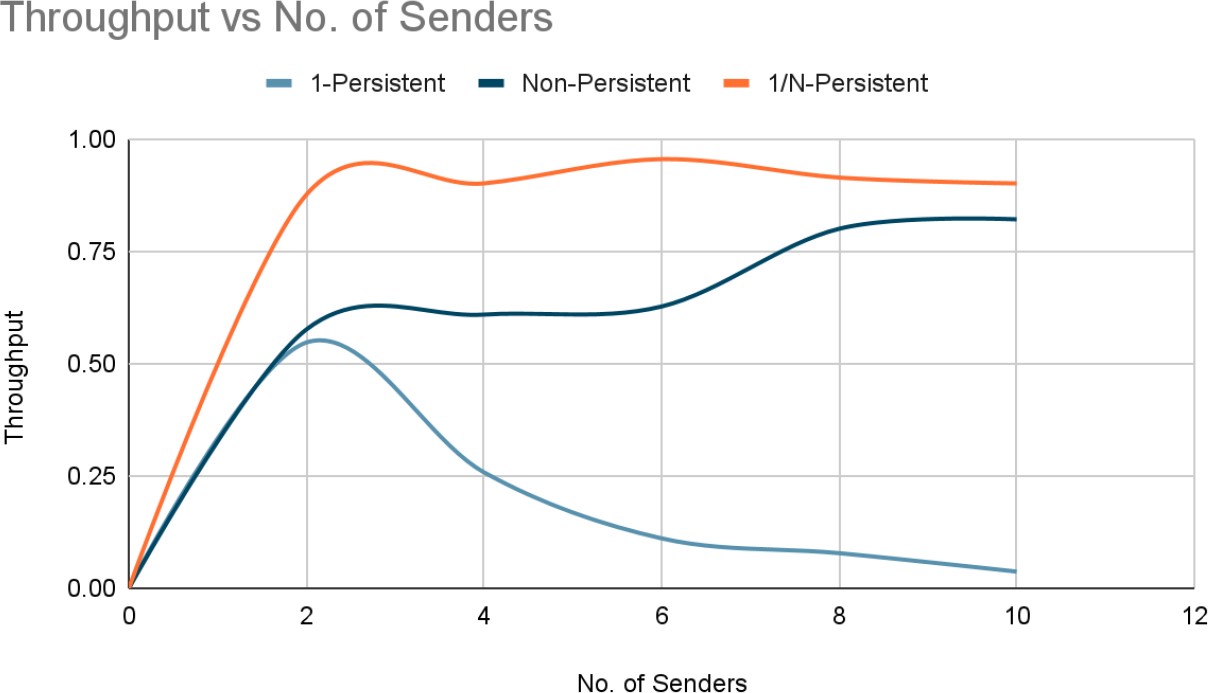
for node in Nodes:

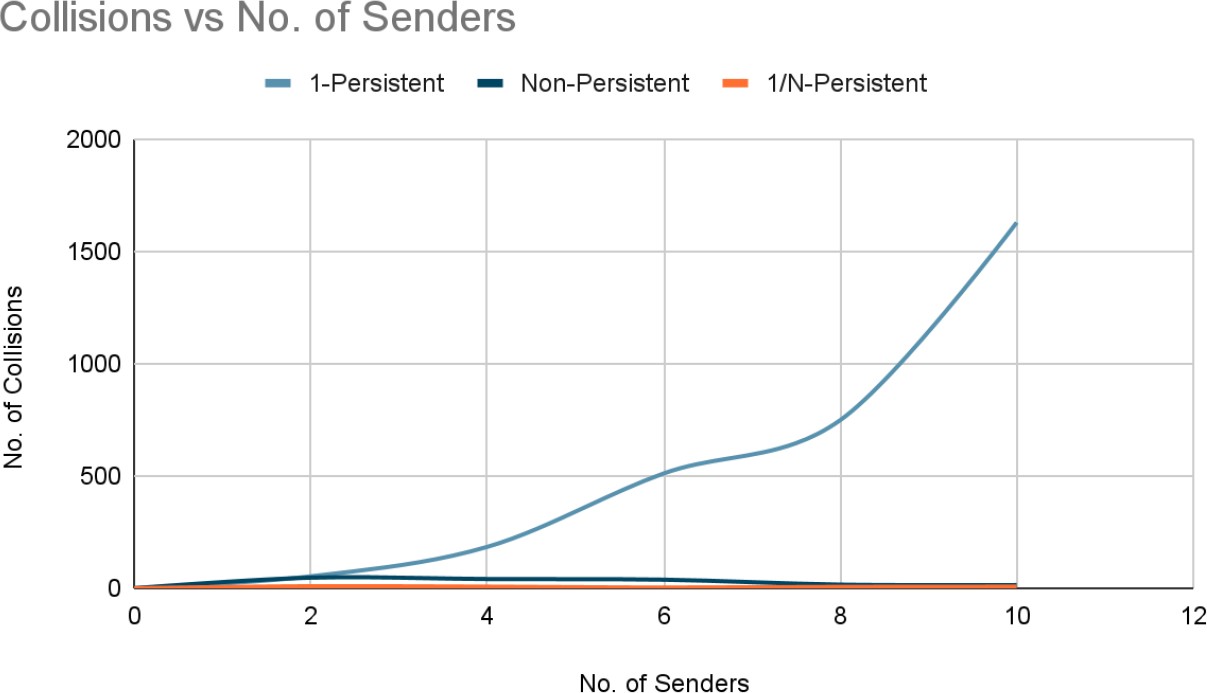
node.start()

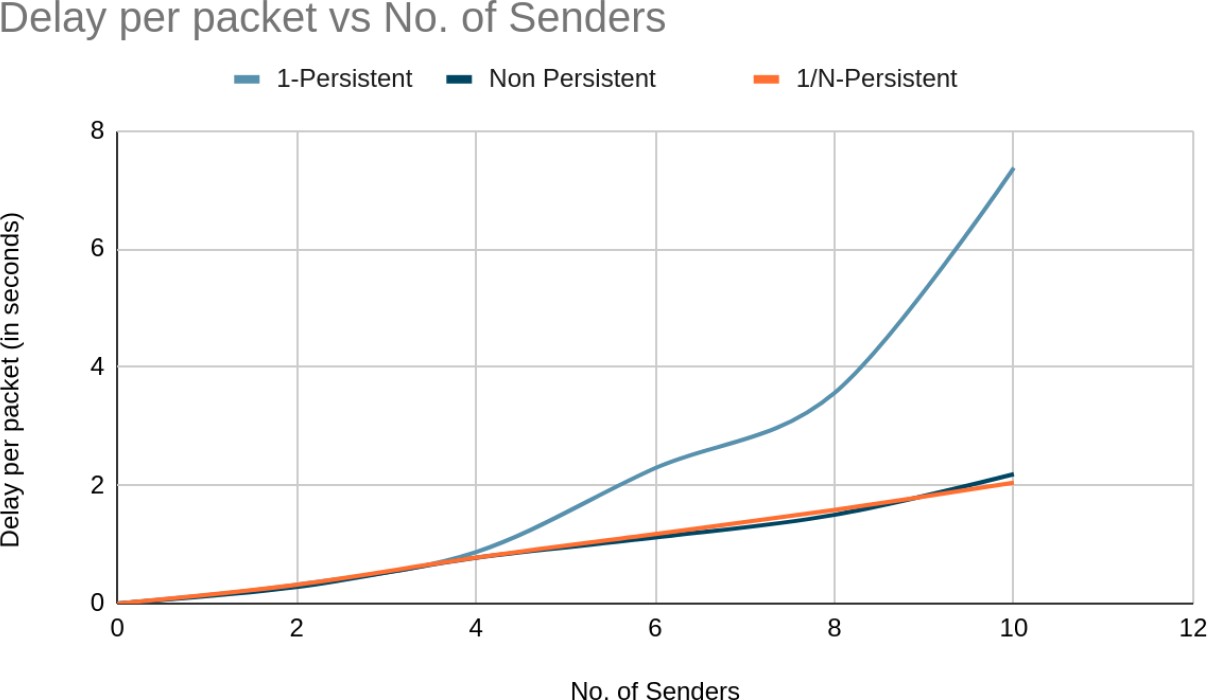
for node in Nodes:

node.join()

met.join()







**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.