Computer Networks Lab Report - Assignment 3

Name: Bikash Sah

Class: BCSE-3

Group: A1

Assignment Number: 3

Problem Statement:

Assignment 3: Implement 1-persistent, non-persistent and p-persistent CSMA techniques.

Submission due: 12th -16th *September* 2022

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 Date: **21 Nov, 2022**

Deadline: 16th September, 2022

Introduction

1. 1-Persistent
2. Non-persistent
3. p-persistent

1-Persistent

The 1-persistent method is simple and straightforward. In this method, after the station finds the line idle, it sends its frame immediately (with probability 1). This method has the highest
chance of collision because two or more stations may find the line idle and send their frames immediately.

Non-Persistent

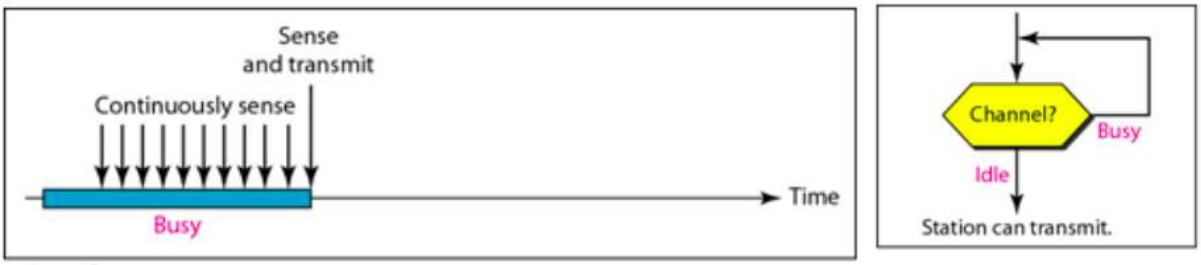
In the nonpersistent method, a station that has a frame to send senses the line. If the line is idle, it sends immediately. If the line is not idle, it waits for a random amount of time and then senses the line again. The non persistent approach reduces the chance of collision because it is
unlikely that two or more stations will wait the same amount of time and retry to send simultaneously.

P-Persistent

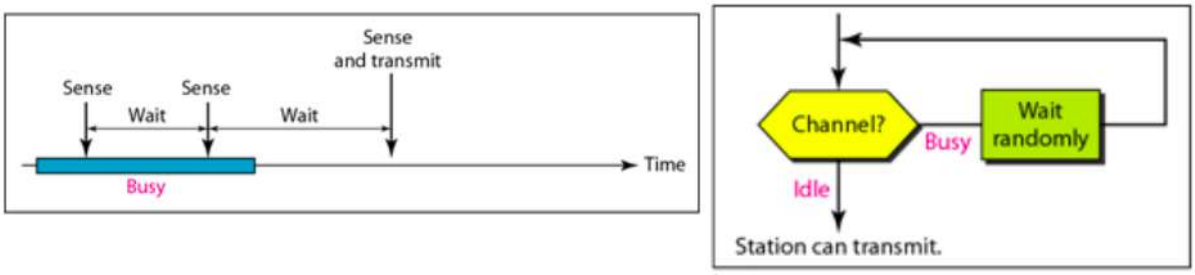
The p-persistent method uses advantages of both other strategies. It reduces the collision and improves the efficiency.

Design

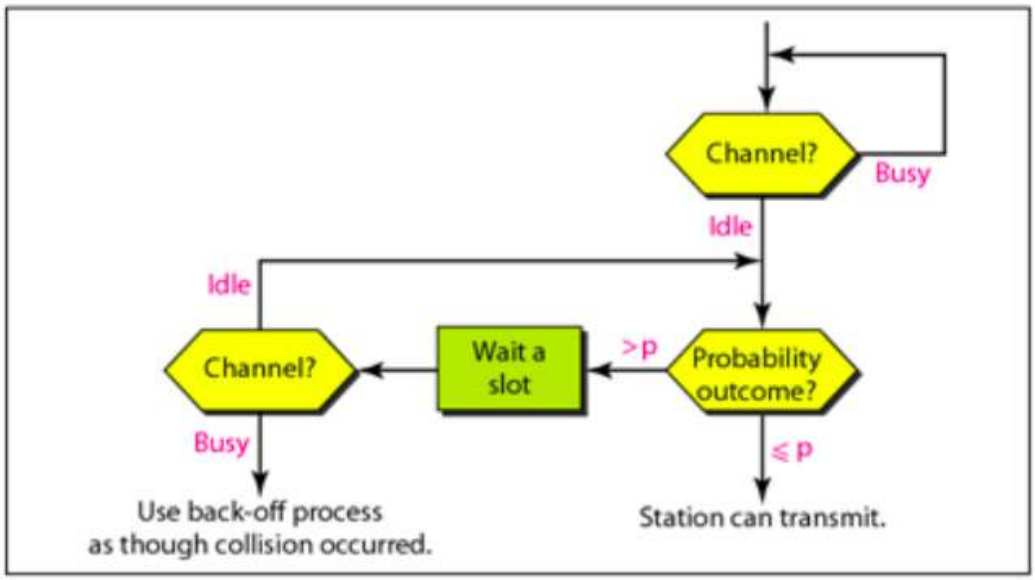
1-Persistent



Non-Persistent



P-Persistent



Implementation

P-Persistent:

```
import threading
import time

frameTime = 3
interFrameTime = 1

numFrames = 2
totalFrames = 0


# function to determine channel utilization of 1p CSMA
# Channel utilization is the percentage of time that the channel is busy (i.e. the per
centage of time that the channel is not idle)
def utility(lock, total): # It is the fraction of time used to transmit data packets.
    global totalFrames
    tot = 0
    used = 0
    while totalFrames < total:
        if lock.locked():
            used += 1
        tot += 1
    channelUtil = used / tot
    # Channel Utilization to 2 decimal places
    print("-----------------------------------------------------")
    print("Channel Utilization: " + str(round(channelUtil, 4)))
    print("-----------------------------------------------------")

    print()


# CSMA class using threading
class CSMA(threading.Thread):
    def __init__(self, lock, k, index): # initialize class with lock, k, and index val
ues
        super().__init__()
        self.lock = lock
        self.k = k
        self.index = index

    def run(self):
        cnt = 1
        global numFrames
        global totalFrames
        while cnt <= numFrames:
            print("-----------------------------------------------------")
            print("Thread " + str(self.index) + " is trying to acquire lock")
            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: ")) # Take number of nodes as inpu
t

    lock = threading.Lock() # Create lock object which will be used by all threads
    met = threading.Thread(target=utility, args=[lock, numberNodes * numFrames]) # Cre
ate thread to calculate channel utilization

    Nodes = [CSMA(lock, 4, i + 1) for i in range(0, numberNodes)] # Create list of thr
eads for each node in the network
    met.start() # Start thread to calculate channel utilization
    for node in Nodes: # Start all threads
        node.start()

    for node in Nodes: # Wait for all threads to finish
        node.join()
    met.join() # Wait for channel utilization thread to finish
```

Non-Persistent:

```
import threading
import random
import time

frameTime = 3
interFrameTime = 1
numFrames = 2
totalFrames = 0


def metrics(lock, total):
    global totalFrames
    tot = 0
    used = 0
    while totalFrames < total:
        if lock.locked():
            used += 1
        tot += 1
    print("-------------------------------------------------")
    # Channel Utilization to 4 decimal places
    print("Channel Utilization: " + str(round(used / tot, 4)))
    print()


class CSMA(threading.Thread):
    def __init__(self, lock, index):
        super().__init__()
        self.lock = lock
```

```
        self.index = index

    def run(self):
        global numFrames
        global totalFrames

        cnt = 1
        while cnt <= numFrames:
            print("--------------------------------------------------")
            print(f"Attempting to send frame {cnt} of node {self.index}")
            print("--------------------------------------------------")
            print()

            while self.lock.locked():
                backOffTime = random.randint(2, 5)
                print("--------------------------------------------------")
                print(f"Node {self.index} waiting for time : {backOffTime}")
                print("--------------------------------------------------")
                print()
                time.sleep(backOffTime)

            self.lock.acquire()
            time.sleep(frameTime)
            print("--------------------------------------------------")
            print(f"Successfully sent frame {cnt} of node {self.index}")
            print("--------------------------------------------------")
            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, 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:

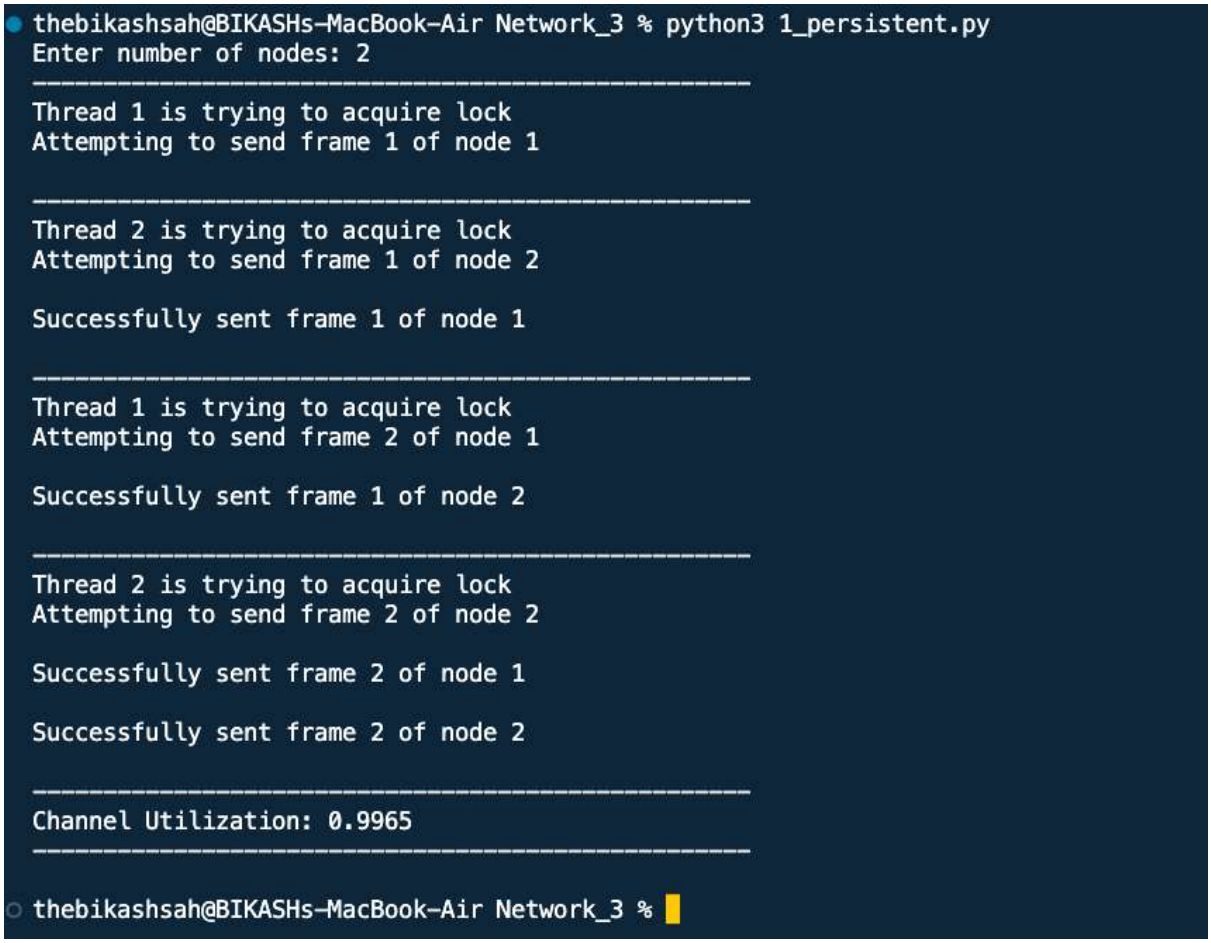
```
import threading
import random
import time
```

```
frameTime = 3
interFrameTime = 1
backOffTime = 2
numFrames = 2
totalFrames = 0


def metrics(lock, total):
    global totalFrames
    tot = 0
    used = 0
    while totalFrames < total:
        if lock.locked():
            used += 1
        tot += 1
    print("--------------------------------------------------")
    print("Channel Utilization: " + str(round(used / tot, 4)))



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

            while self.lock.locked():
                pass

            decision = random.randint(0, 1);
            while decision > self.prob:
                print("--------------------------------------------------")
                print(f"Node {self.index} backing off")
                print("--------------------------------------------------")


                time.sleep(backOffTime)

                while self.lock.locked():
                    pass

                decision = random.randint(0, 1);

            self.lock.acquire()
            time.sleep(frameTime)
            print(f"Successfully sent frame {cnt} of node {self.index}")
```

```
            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, 1 / numberNodes) for i in range(0, numberNodes)]
    met.start()
    for node in Nodes:
        node.start()

    for node in Nodes:
        node.join()
    met.join()
```

Test Cases

1-persistent:



Non-persistent:

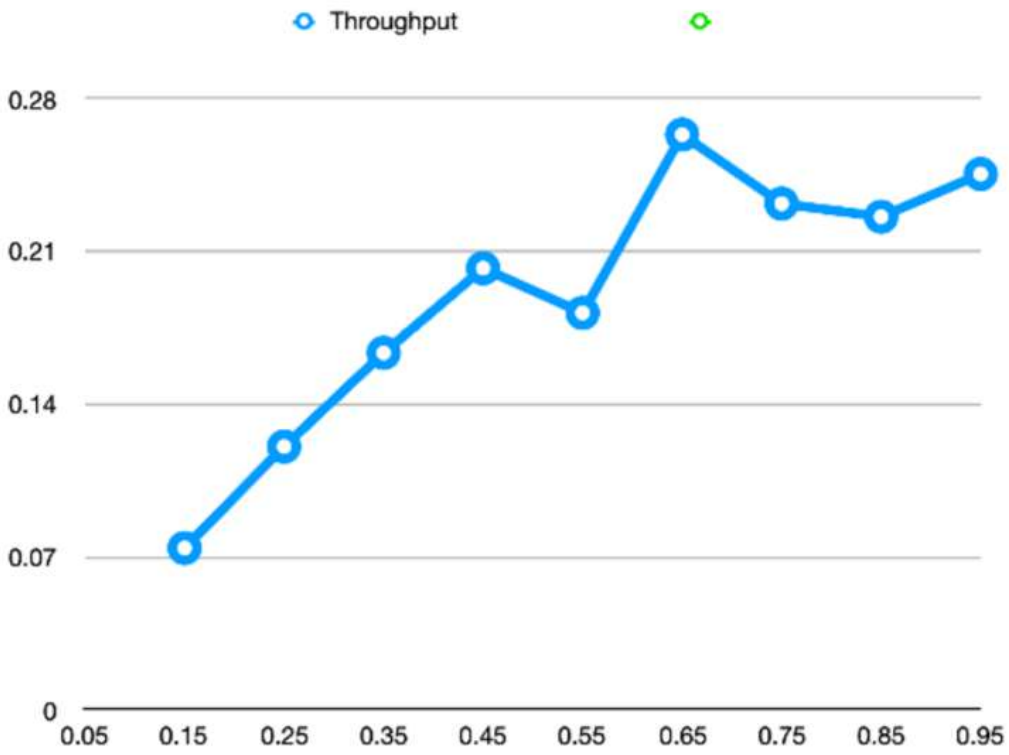
```
thebikashsah@BIKASHs-MacBook-Air Network_3 % python3 non_persistent.py
Enter number of nodes: 2
--------------------------------------------------
Attempting to send frame 1 of node 1
--------------------------------------------------
--------------------------------------------------
Attempting to send frame 1 of node 2
--------------------------------------------------
--------------------------------------------------
Node 2 waiting for time : 4
--------------------------------------------------
--------------------------------------------------
Successfully sent frame 1 of node 1
--------------------------------------------------
--------------------------------------------------
Attempting to send frame 2 of node 1
--------------------------------------------------
--------------------------------------------------
Node 2 waiting for time : 4
--------------------------------------------------
--------------------------------------------------
Successfully sent frame 2 of node 1
--------------------------------------------------
--------------------------------------------------
Successfully sent frame 1 of node 2
--------------------------------------------------
--------------------------------------------------
Attempting to send frame 2 of node 2
--------------------------------------------------
--------------------------------------------------
Successfully sent frame 2 of node 2
--------------------------------------------------
--------------------------------------------------
Channel Utilization: 0.7474
thebikashsah@BIKASHs-MacBook-Air Network_3 %
```

P-persistent:

```
thebikashsah@BIKASHs-MacBook-Air Network_3 % python3 p_persistent.py
Enter number of nodes: 2
------------------------------------------------------
Attempting to send frame 1 of node 1
------------------------------------------------------
------------------------------------------------------
Node 1 backing off
------------------------------------------------------
------------------------------------------------------
Attempting to send frame 1 of node 2
------------------------------------------------------
Successfully sent frame 1 of node 2
------------------------------------------------------
Node 1 backing off
------------------------------------------------------
------------------------------------------------------
Attempting to send frame 2 of node 2
------------------------------------------------------
------------------------------------------------------
Node 2 backing off
------------------------------------------------------
------------------------------------------------------
Node 1 backing off
------------------------------------------------------
------------------------------------------------------
Node 2 backing off
------------------------------------------------------
Successfully sent frame 1 of node 1
------------------------------------------------------
Attempting to send frame 2 of node 1
------------------------------------------------------
Successfully sent frame 2 of node 2
Successfully sent frame 2 of node 1
------------------------------------------------------
Channel Utilization: 0.6476
thebikashsah@BIKASHs-MacBook-Air Network_3 %
```

Results

Comparing the throughput by varing p of the p-persistent CSMA approach. The p is varied according to the X-axis and the throughput is along the Y-axis. The number of nodes considered for this simulation is 12.



Discussion:

> This assignment was like a real life project, it had a problem statement and I had to come up with a solution, I have learnt a lot, I learnt a lot of python in this assignment and also learnt how to implement big problems by dividing it into subproblems.