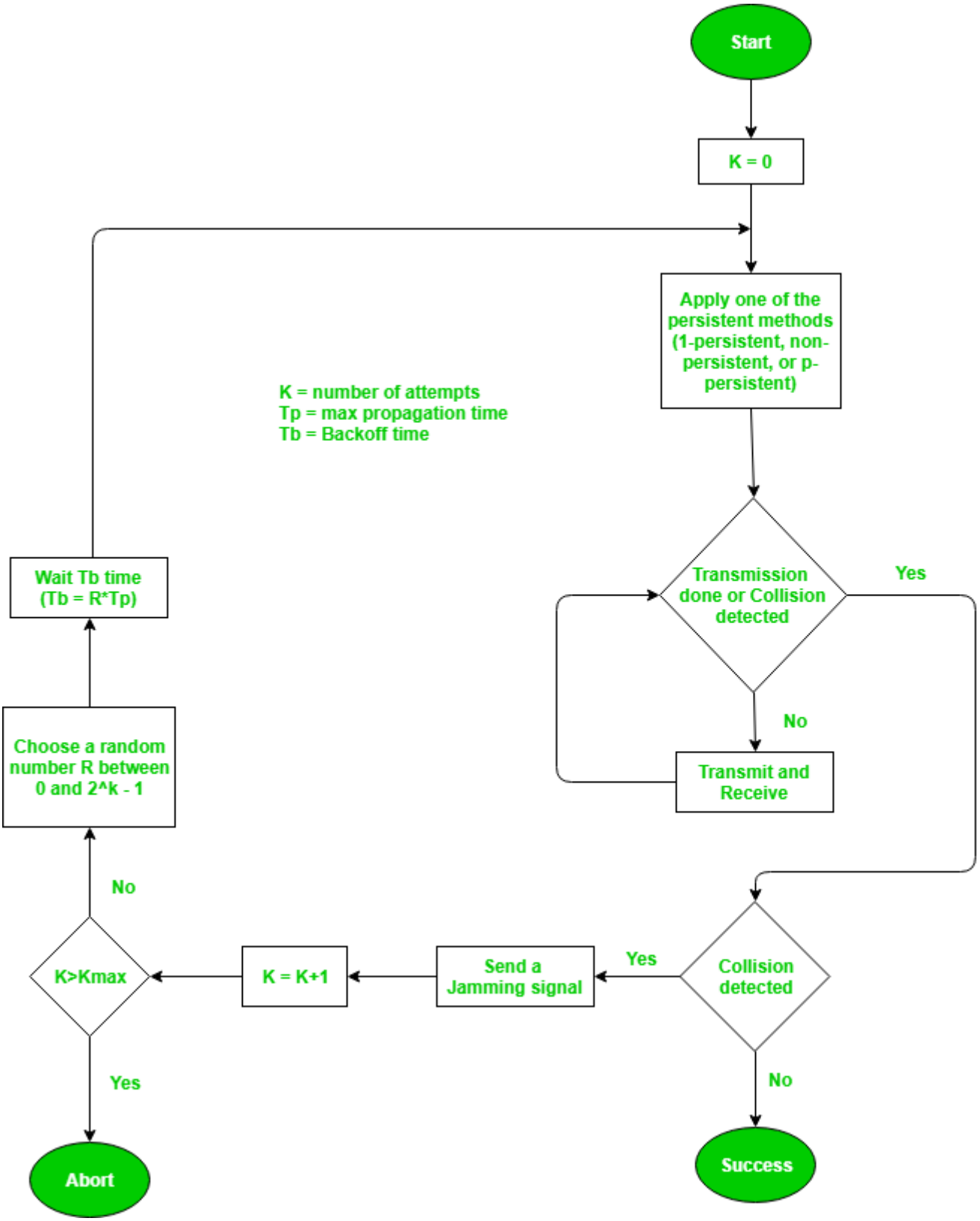
Name: Tanmoy Sarkar
Roll No: 002010501020
Class: BCSE III
Assignment No: 3
Subject: Computer Network
Group: A1

Problem Statement: Implement 1-persistent, non-persistent and p-persistent CSMA techniques.

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 the performance of different CSMA techniques.



For CSMA, In our implementation, we have created each thread for each node and to identify whether the channel is busy or idle we used a thread lock to simulate them. Instead of sending a frame from one client to another, we just wait for a static time that is required to send the frame from one client to another client.

IMPLEMENTATION

Code for 1-persistent CSMA

```
import threading
import time


frameTime = 3
interFrameTime = 1

# Total frames currently sent
totalFrames = 0


def log(lock, totalFramesWillBeSent):
    global totalFrames
    tmp = 0
    used = 0
    while totalFrames < totalFramesWillBeSent:
        if lock.locked():
            used += 1
        tmp += 1
    print(f"Channel utilization -> {used/tmp}")

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

    def run(self):
        global numFrames
        global totalFrames
        count=1

        while count <= numFrames:
            print(f"[Attempting] Frame {count} | Node {self.index}")
            # If locked wait for milliseconds
            while self.lock.locked():
                pass

            # Acquire lock
            self.lock.acquire()
            # Sleep for frame time -- simulation that the frame sending on the log
            time.sleep(frameTime)
            totalFrames += 1
            print(f"[Successful] Frame {count} | Node {self.index}")
            # Release lock
```

```
            self.lock.release()
            # Sleep for interframe time
            time.sleep(interFrameTime)
            count+=1

if __name__=='__main__':
    numberNodes = int(input("Enter number of nodes: "))
    numFrames = int(input("Enter number of frames will be sent by frame : "))

    lock = threading.Lock() # Lock object
    nodes = [CSMA(lock,i+1) for i in range(0,numberNodes)] # List of nodes
    # Create log thread
    logThread = threading.Thread(target=log, args=[lock, numberNodes*numFrames])
    logThread.start()
    # Start all threads
    for node in nodes:
        node.start()
    # Wait for all threads to finish
    for node in nodes:
        node.join()
    # Wait for log thread to finish
    logThread.join()
```

Code for non-persistent CSMA

```
import random
import threading
import time

frameTime = 3
interFrameTime = 1

# Total frames currently sent
totalFrames = 0

def channel(lock, totalFramesWillBeSent):
    global totalFrames
    tmp = 0
    used = 0
    while totalFrames < totalFramesWillBeSent:
        if lock.locked():
            used += 1
        tmp += 1
    print(f"Channel utilization -> {used/tmp}")

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

```
        self.lock=lock
        self.index=index

    def run(self):
        global numFrames
        global totalFrames
        count=1

        while count <= numFrames:
            print(f"[Attempting] Frame {count} | Node {self.index}")
            # If locked wait for milliseconds
            while self.lock.locked():
                backOffTime=random.randint(2,5)
                print(f"[Waiting] Fram {count} | Node {self.index} | Backoff
Time {backOffTime}")
                time.sleep(backOffTime)

            # Acquire lock
            self.lock.acquire()
            # Sleep for frame time -- simulation that the frame sending on the
channel
            time.sleep(frameTime)
            totalFrames += 1
            print(f"[Successful] Frame {count} | Node {self.index}")
            # Release lock
            self.lock.release()
            # Sleep for interframe time
            time.sleep(interFrameTime)
            count+=1

if __name__=='__main__':
    numberNodes = int(input("Enter number of nodes: "))
    numFrames = int(input("Enter number of frames will be sent by frame : "))

    lock = threading.Lock() # Lock object
    nodes = [CSMA(lock,i+1) for i in range(0,numberNodes)] # List of nodes
    # Create channel thread
    channelThread = threading.Thread(target=channel, args=[lock,
numberNodes*numFrames])
    channelThread.start()
    # Start all threads
    for node in nodes:
        node.start()
    # Wait for all threads to finish
    for node in nodes:
```

```
        node.join()
    # Wait for channel thread to finish
    channelThread.join()
```

Code for p-persistent CSMA

```
import random
import threading
import time

frameTime = 3
interFrameTime = 1

# Total frames currently sent
totalFrames = 0


def channel(lock, totalFramesWillBeSent):
    global totalFrames
    tmp = 0
    used = 0
    while totalFrames < totalFramesWillBeSent:
        if lock.locked():
            used += 1
        tmp += 1
    print(f"Channel utilization -> {used/tmp}")

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

    def run(self):
        global numFrames
        global totalFrames
        global backOffTime
        global probability

        count=1

        while count <= numFrames:
            print(f"[Attempting] Frame {count} | Node {self.index}")
            # If locked wait for milliseconds
            while self.lock.locked():
                pass
```

```
            decision = random.random()
            while decision>probability:
                print(f"[Waiting] Fram {count} | Node {self.index} | Backoff
Time {backOffTime} | Decision {decision}")
                time.sleep(backOffTime)
                while self.lock.locked():
                    pass
                decision=random.random()

            # Acquire lock
            self.lock.acquire()
            # Sleep for frame time -- simulation that the frame sending on the
channel
            time.sleep(frameTime)
            totalFrames += 1
            print(f"[Successful] Frame {count} | Node {self.index}")
            # Release lock
            self.lock.release()
            # Sleep for interframe time
            time.sleep(interFrameTime)
            count+=1

if __name__=='__main__':
    numberNodes = int(input("Enter number of nodes: "))
    numFrames = int(input("Enter number of frames will be sent by frame : "))
    backOffTime = int(input("Enter backoff time: "))
    probability = 1/numberNodes

    lock = threading.Lock() # Lock object
    nodes = [CSMA(lock,i+1) for i in range(0,numberNodes)] # List of nodes
    # Create channel thread
    channelThread = threading.Thread(target=channel, args=[lock,
numberNodes*numFrames])
    channelThread.start()
    # Start all threads
    for node in nodes:
        node.start()
    # Wait for all threads to finish
    for node in nodes:
        node.join()
    # Wait for channel thread to finish
    channelThread.join()
```

OUTPUT

Output for 1-persistent -

```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass3$ python 1_persistent.py
Enter number of nodes: 3
Enter number of frames will be sent by frame : 5
[Attempting] Frame 1 | Node 1
[Attempting] Frame 1 | Node 2
[Attempting] Frame 1 | Node 3
[Successful] Frame 1 | Node 1
[Attempting] Frame 2 | Node 1
[Successful] Frame 1 | Node 2
[Attempting] Frame 2 | Node 2
[Successful] Frame 1 | Node 3
[Attempting] Frame 2 | Node 3
[Successful] Frame 2 | Node 1
[Attempting] Frame 3 | Node 1
[Successful] Frame 2 | Node 2
[Attempting] Frame 3 | Node 2
[Successful] Frame 2 | Node 3
[Attempting] Frame 3 | Node 3
[Successful] Frame 3 | Node 1
[Attempting] Frame 4 | Node 1
[Successful] Frame 3 | Node 2
[Attempting] Frame 4 | Node 2
[Successful] Frame 3 | Node 3
[Attempting] Frame 4 | Node 3
[Successful] Frame 4 | Node 2
[Attempting] Frame 5 | Node 2
[Successful] Frame 4 | Node 1
[Attempting] Frame 5 | Node 1
[Successful] Frame 4 | Node 3
[Attempting] Frame 5 | Node 3
[Successful] Frame 5 | Node 1
[Successful] Frame 5 | Node 2
[Successful] Frame 5 | Node 3
Channel utilization -> 0.9972703974209695
```

Output for non-persistent -

```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass3$ python non_persistent.py
Enter number of nodes: 3
Enter number of frames will be sent by frame : 5
[Attempting] Frame 1 | Node 1
[Attempting] Frame 1 | Node 2
[Waiting] Fram 1 | Node 2 | Backoff Time 4
[Attempting] Frame 1 | Node 3
[Waiting] Fram 1 | Node 3 | Backoff Time 3
[Successful] Frame 1 | Node 1
[Attempting] Frame 2 | Node 1
[Waiting] Fram 2 | Node 1 | Backoff Time 4
[Waiting] Fram 1 | Node 2 | Backoff Time 5
[Successful] Frame 1 | Node 3
[Attempting] Frame 2 | Node 3
[Waiting] Fram 2 | Node 1 | Backoff Time 2
[Waiting] Fram 1 | Node 2 | Backoff Time 2
[Waiting] Fram 2 | Node 1 | Backoff Time 2
[Successful] Frame 2 | Node 3
[Attempting] Frame 3 | Node 3
[Waiting] Fram 3 | Node 3 | Backoff Time 4
[Waiting] Fram 2 | Node 1 | Backoff Time 3
[Successful] Frame 1 | Node 2
[Waiting] Fram 3 | Node 3 | Backoff Time 4
[Attempting] Frame 2 | Node 2
[Waiting] Fram 2 | Node 2 | Backoff Time 3
[Successful] Frame 2 | Node 1
[Attempting] Frame 3 | Node 1
[Waiting] Fram 3 | Node 1 | Backoff Time 3
[Waiting] Fram 3 | Node 3 | Backoff Time 5
[Successful] Frame 2 | Node 2
[Attempting] Frame 3 | Node 2
[Waiting] Fram 3 | Node 2 | Backoff Time 3
[Waiting] Fram 3 | Node 3 | Backoff Time 2
[Successful] Frame 3 | Node 1
[Attempting] Frame 4 | Node 1
[Waiting] Fram 4 | Node 1 | Backoff Time 2
[Waiting] Fram 3 | Node 3 | Backoff Time 3
[Successful] Frame 3 | Node 2
[Attempting] Frame 4 | Node 2
[Waiting] Fram 4 | Node 2 | Backoff Time 2
[Waiting] Fram 3 | Node 3 | Backoff Time 3
[Successful] Frame 4 | Node 1
[Attempting] Frame 5 | Node 1
[Waiting] Fram 5 | Node 1 | Backoff Time 2
[Waiting] Fram 3 | Node 3 | Backoff Time 4
[Waiting] Fram 5 | Node 1 | Backoff Time 4
[Successful] Frame 4 | Node 2
[Attempting] Frame 5 | Node 2
[Waiting] Fram 3 | Node 3 | Backoff Time 5
[Waiting] Fram 5 | Node 1 | Backoff Time 2
[Successful] Frame 5 | Node 2
[Waiting] Fram 3 | Node 3 | Backoff Time 4
[Successful] Frame 5 | Node 1
[Successful] Frame 3 | Node 3
[Attempting] Frame 4 | Node 3
[Successful] Frame 4 | Node 3
[Attempting] Frame 5 | Node 3
[Successful] Frame 5 | Node 3
Channel utilization -> 0.761094890756491
```

Output for p-persistent -

```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass3$ python p_persistent.py
Enter number of nodes: 3
Enter number of frames will be sent by frame : 5
Enter backoff time: 1
[Attempting] Frame 1 | Node 1
[Waiting] Fram 1 | Node 1 | Backoff Time 1 | Decision 0.44661324889794696
[Attempting] Frame 1 | Node 2
[Attempting] Frame 1 | Node 3
[Successful] Frame 1 | Node 2
[Waiting] Fram 1 | Node 1 | Backoff Time 1 | Decision 0.8762436572912545
[Waiting] Fram 1 | Node 3 | Backoff Time 1 | Decision 0.6219134203679332
[Attempting] Frame 2 | Node 2
[Successful] Frame 1 | Node 3
[Attempting] Frame 2 | Node 3
[Successful] Frame 2 | Node 2
[Waiting] Fram 2 | Node 3 | Backoff Time 1 | Decision 0.39491054673103576
[Attempting] Frame 3 | Node 2
[Successful] Frame 1 | Node 1
[Waiting] Fram 3 | Node 2 | Backoff Time 1 | Decision 0.6877240626941749
[Waiting] Fram 2 | Node 3 | Backoff Time 1 | Decision 0.804523328874772
[Waiting] Fram 3 | Node 2 | Backoff Time 1 | Decision 0.7538629728803926
[Attempting] Frame 2 | Node 1
[Waiting] Fram 2 | Node 1 | Backoff Time 1 | Decision 0.3790190034986691
[Waiting] Fram 2 | Node 3 | Backoff Time 1 | Decision 0.4851412069250256
[Waiting] Fram 2 | Node 1 | Backoff Time 1 | Decision 0.6837603032060176
[Waiting] Fram 3 | Node 2 | Backoff Time 1 | Decision 0.9343598397899764
[Successful] Frame 2 | Node 3
[Attempting] Frame 3 | Node 3
[Successful] Frame 2 | Node 1
[Attempting] Frame 3 | Node 1
[Successful] Frame 3 | Node 2
[Attempting] Frame 4 | Node 2
[Successful] Frame 3 | Node 1
[Waiting] Fram 3 | Node 3 | Backoff Time 1 | Decision 0.555302821463085
[Waiting] Fram 4 | Node 2 | Backoff Time 1 | Decision 0.4205059484333239
[Attempting] Frame 4 | Node 1
[Successful] Frame 4 | Node 1
[Waiting] Fram 3 | Node 3 | Backoff Time 1 | Decision 0.6889952073915103
[Attempting] Frame 5 | Node 1
[Successful] Frame 4 | Node 2
[Attempting] Frame 5 | Node 2
[Successful] Frame 3 | Node 3
[Attempting] Frame 4 | Node 3
[Successful] Frame 5 | Node 2

[Waiting] Fram 4 | Node 3 | Backoff Time 1 | Decision 0.6235615738429497
[Waiting] Fram 5 | Node 1 | Backoff Time 1 | Decision 0.9021538466265436
[Waiting] Fram 4 | Node 3 | Backoff Time 1 | Decision 0.5907892551227651
[Successful] Frame 5 | Node 1
[Successful] Frame 4 | Node 3
[Attempting] Frame 5 | Node 3
[Waiting] Fram 5 | Node 3 | Backoff Time 1 | Decision 0.860453998302961
[Waiting] Fram 5 | Node 3 | Backoff Time 1 | Decision 0.5010107987806726
[Successful] Frame 5 | Node 3
Channel utilization -> 0.6788664549915409
```

RESULTS & ANALYSIS

p-persistent CSMA:

- This method is used when the channel has time slots such that the time slot duration is equal to or greater than the maximum propagation delay time.
- Whenever a station becomes ready to send, it senses the channel.
- If the channel is busy, the station waits until the next slot.
- If the channel is idle, it transmits with a probability p.
- With the probability q=l-p, the station then waits for the beginning of the next time slot.
- If the next slot is also idle, it either transmits or waits again with probabilities p and q.
- This process is repeated till either frame has been transmitted or another station has begun transmitting.
- In case of the transmission by another station, the station acts as though a collision has occurred and it waits a random amount of time and starts again.

Advantages of p-persistent CSMA:

- It reduces the chance of collision and improves the efficiency of the network.

CSMA/CD :