COMPUTER NETWORKS LAB

NAME: ARITRA DATTA

ROLL NO: 002010501054

CLASS: BCSE – III

GROUP: A2

ASSIGNMENT: 3

DEADLINE: 16th September, 2022

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.

Date of Submission: 12th October, 2022

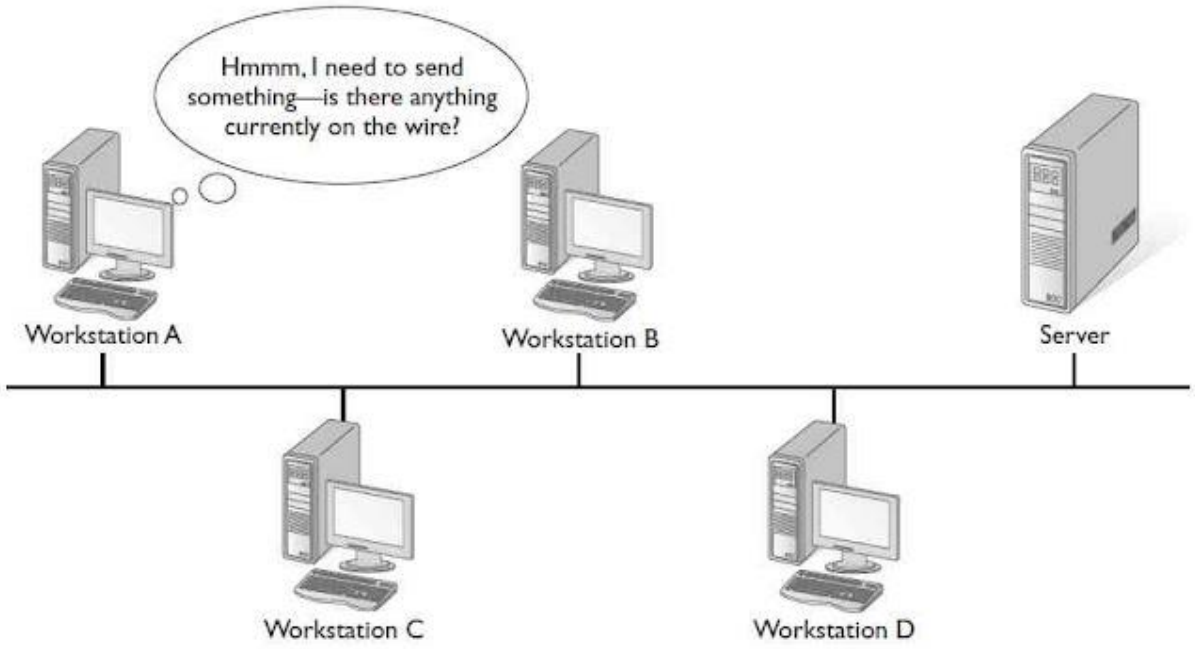
DESIGN

Station:

- Connect with the Channel.
- Get total clients and current node number
- Build Frame and send it to the channel and follow methods as per protocol

Channel:

- Get station request (Node[Station] - request).
- Maintain a physical channel in the form of an array.
- Block each index if a packet exists at that unit or unblock it
- Run a thread fro every node that connects to the channel



Flow Diagram of the Generalised Network

The packets have been framed as per the Ethernet IEEE 802.3 Frame Format:

7 byte	1 byte	6 byte	6 byte	2 byte	46 to 1500 byte	4 byte
Preamble	Start Frame Delimiter	Destination Address	Source Address	Length	Data	Frame Check Sequence (CRC)

IMPLEMENTATION

CHANNEL.py

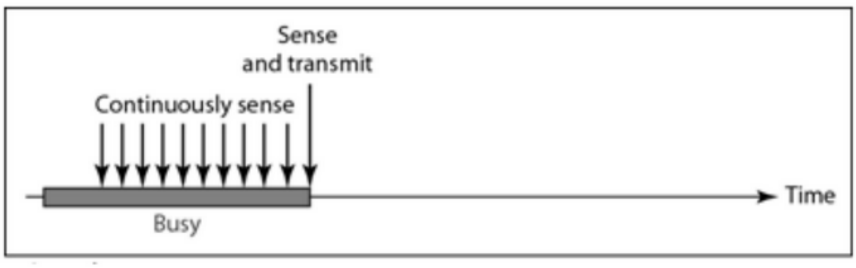
```python
import socket
import threading
import time
import random

IP = socket.gethostbyname(socket.gethostname())
PORT = 4456
ADDR = (IP, PORT)
SIZE = 1024
FORMAT = "utf-8"
DIST = 5
HEADERSIZE = 10
dist = [False for _ in range(31)]

def receive_message(client_socket):
    try:
        msg_header = client_socket.recv(HEADERSIZE)

        if not len(msg_header):
            return False

        msg_len = int(msg_header.decode(FORMAT).strip())

        data = client_socket.recv(msg_len).decode(FORMAT)
        return data

    except:
        return False

def createFrame(message):
    return  f"{len(message):<{HEADERSIZE}}" + message

def Isbusy(i):
    global dist
    return dist[i]

def block(i):
    global dist
    dist[i] = True

def unblock(i):
    global dist
    dist[i] = False

def unpack(data):
    return {
        'dst' : int(data[0]),
        'res' : data[1:-1],
        'status' : data[-1]
    }
```

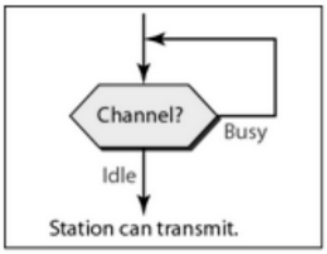
```python
def handle_client(conn, addr,n,client,lock):
    print(f"[NEW CONNECTION] {addr}  Station {client + 1}")
    conn.send(createFrame(str(n)+"$"+str(client)).encode(FORMAT))
    # station_r[client].send(createFrame(str(client)).encode(FORMAT))
    time.sleep(2)
    print("Ready for Communication")

    while True:
        flag = False
        cur = max(0,client*DIST - 1)
        dst = -1
        line_end = (n-1)*DIST - 1

        res = receive_message(conn)
        if not res:
            break

        lock.acquire()
        packet = unpack(res)
        if Isbusy(cur):
            print("Channel is Busy\n")
            conn.send(createFrame("0").encode(FORMAT))
        else :
            if packet['status'] == "1":
                block(cur)
                dst = max(0,packet['dst']*DIST - 1)
                flag = True
            conn.send(createFrame("1").encode(FORMAT))
        lock.release()


        if flag:
            left,right = True,True
            i,j = cur,cur

            while left or right:
                lock.acquire()
                if left:
                    if not Isbusy(i):
                        print(f"Collision Occured at Unit {i+1}")
                        print(f"Signal from Station - {client + 1}\n")
                        left = False
                    else:
                        unblock(i)
                    if left:
                        i -= 1
                        if i >= 0:
                            if Isbusy(i) :
                                print(f"Collision Occured at Unit {i+1}")
                                print(f"Signal from Station - {client + 1}\n")
                                left = False
```

```python
                            if Isbusy(i) :
                                print(f"Collision Occured at Unit {i+1}")
                                print(f"Signal from Station - {client + 1}\n")
                                left = False
                                unblock(i)
                            else:
                                block(i)
                                if i == dst:
                                    # station_r[packet['dst']].send(createFrame(packet['res']).encode(FORMAT))
                                    print(f"Data received by Station {packet['dst']+1} : {packet['res']}")
                        else:
                            left = False

                if j == i:
                    right = False
                if right:
                    if (j != i+1) and (not Isbusy(j)):
                        print(f"Collision Occured at Unit {j+1}")
                        print(f"Signal from Station - {client + 1}\n")
                        right = False
                    else:
                        unblock(j)
                    if right:
                        j += 1
                        if j <= line_end :
                            if Isbusy(j) :
                                print(f"Collision Occured at Unit {j+1}")
                                print(f"Signal from Station - {client + 1}\n")
                                right = False
                                unblock(j)
                            else:
                                block(j)
                                if j == dst:
                                    # station_r[packet['dst']].send(createFrame(packet['res']).encode(FORMAT))
                                    print(f"Data received by Station {packet['dst']+1} : {packet['res']}")
                        else:
                            right = False
                lock.release()
                time.sleep(2)

    print(f"[DISCONNECTED] {addr} Station - {client + 1}")
    conn.close()
```

```
def main():
    N = int(input("Enter the number of Stations : "))
    print("[STARTING] SHARED CHANNEL is starting")
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(ADDR)
    server.listen()
    print(f"[LISTENING] Server is listening on {IP}:{PORT}.")

    lock = threading.Lock()
    for i in range(N):
        conn_c, addr_c = server.accept()
        # conn_r, addr_r = server.accept()
        # station_c[i] = conn_c
        # station_r[i] = conn_r
        thread = threading.Thread(target=handle_client, args=(conn_c, addr_c,N,i,lock))
        thread.start()

if __name__ == "__main__":
    main()
```

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.



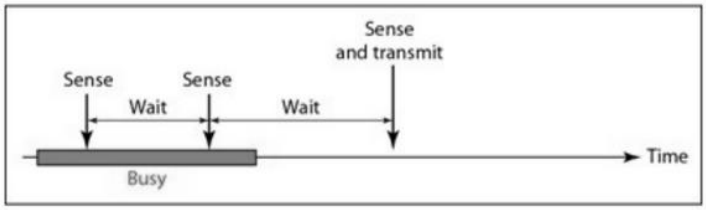


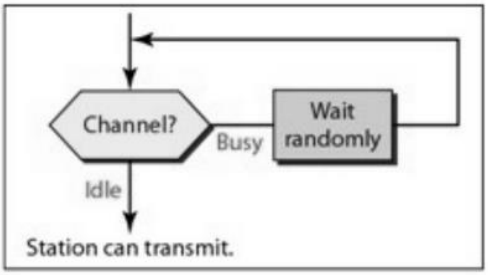
STATION.py

```
import socket
import time

IP = socket.gethostbyname(socket.gethostname())
PORT = 4456
ADDR = (IP, PORT)
HEADERSIZE = 10
FORMAT = "utf-8"

def receive_message(client_socket):
    try:
        msg_header = client_socket.recv(HEADERSIZE)

        if not len(msg_header):
            return False

        msg_len = int(msg_header.decode('utf-8').strip())

        data = client_socket.recv(msg_len).decode(FORMAT)
        return data

    except:
        return False

def createFrame(message):
    return  f"{len(message):<{HEADERSIZE}}" + message

def main():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(ADDR)
    print(f"[CONNECTED] Station connected to SHARED CHANNEL at {IP}:{PORT}")

    res = receive_message(client).split("$")
    n = int(res[0])
    client_num = int(res[1])
    print(f"There are {n} stations in the Shared Channel")
    print(f"You are Station  {client_num + 1}\n")

    while True:
        dest_client = int(input("Enter Station to send data : "))
        if dest_client - 1  == client_num :
            print("Invalid Destination")
            continue
        data = input(f"\nEnter Data to be sent to Station {dest_client} : ")
        if data == "exit":
            break
        res = "0"
        exit = False
```

```
            while res == "0":
                print("Sensing Channel\n")
                client.send(createFrame(str(dest_client - 1) + data + "1").encode(FORMAT))
                res = receive_message(client)
                if not res:
                    print("Channel Destroyed")
                    exit  = True
                    break

                if res == "0" :
                    print("Channel is Busy")
                else:
                    print("Channel Idle, Packet Sent")

            if exit:
                break

            print("\n")


        client.close()


    if __name__ == "__main__":
        main()
```

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



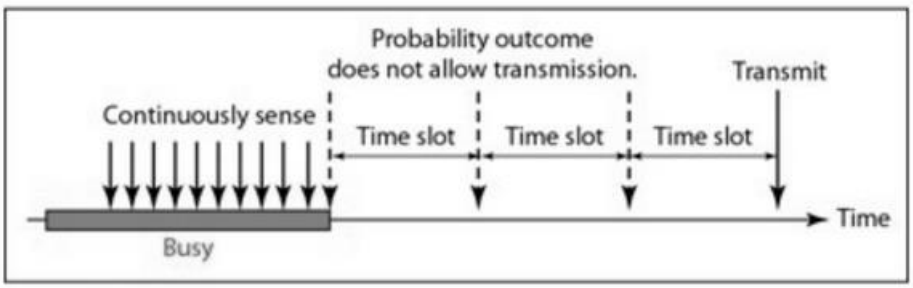


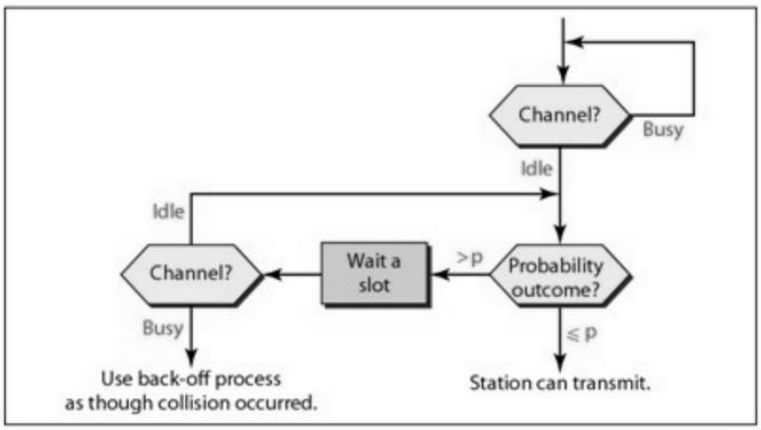
STATION.py

```
import socket
import random
import time

IP = socket.gethostbyname(socket.gethostname())
PORT = 4456
ADDR = (IP, PORT)
HEADERSIZE = 10
FORMAT = "utf-8"

def receive_message(client_socket):
    try:
        msg_header = client_socket.recv(HEADERSIZE)

        if not len(msg_header):
            return False

        msg_len = int(msg_header.decode('utf-8').strip())

        data = client_socket.recv(msg_len).decode(FORMAT)
        return data

    except:
        return False

def createFrame(message):
    return  f"{len(message):<{HEADERSIZE}}" + message

def main():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(ADDR)
    print(f"[CONNECTED] Station connected to SHARED CHANNEL at {IP}:{PORT}")

    res = receive_message(client).split("$")
    n = int(res[0])
    client_num = int(res[1])
    print(f"There are {n} stations in the Shared Channel")
    print(f"You are Station  {client_num + 1}\n")

    while True:
        dest_client = int(input("Enter Station to send data : "))
        if dest_client - 1  == client_num :
            print("Invalid Destination")
            continue
        data = input(f"\nEnter Data to be sent to Station {dest_client} : ")
        if data == "exit":
            break
        res = "0"
        exit = False
```

```
        while res == "0":
            print("Sensing Channel\n")
            client.send(createFrame(str(dest_client - 1) + data + "1").encode(FORMAT))
            res = receive_message(client)
            if not res:
                print("Channel Destroyed")
                exit  = True
                break

            if res == "0" :
                print("Channel is Busy")
                print("Waiting for Re Sensing of Channel...\n")
                time.sleep(random.randint(2,4))
            else:
                print("Channel Idle, Packet Sent")

        if exit:
            break

        print("\n")


    client.close()


if __name__ == "__main__":
    main()
```

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.





STATION.py

```python
import socket
import random
import time

IP = socket.gethostbyname(socket.gethostname())
PORT = 4456
ADDR = (IP, PORT)
HEADERSIZE = 10
FORMAT = "utf-8"
BACKOFF = 3

def receive_message(client_socket):
    try:
        msg_header = client_socket.recv(HEADERSIZE)

        if not len(msg_header):
            return False

        msg_len = int(msg_header.decode('utf-8').strip())

        data = client_socket.recv(msg_len).decode(FORMAT)
        return data

    except:
        return False

def createFrame(message):
    return  f"{len(message):<{HEADERSIZE}}" + message

def main():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(ADDR)
    print(f"[CONNECTED] Station connected to SHARED CHANNEL at {IP}:{PORT}")

    res = receive_message(client).split("$")
    n = int(res[0])
    client_num = int(res[1])
    print(f"There are {n} stations in the Shared Channel")
    print(f"You are Station  {client_num + 1}\n")

    PROBABILITY = 1/n

    while True:
        dest_client = int(input("Enter Station to send data : "))
        if dest_client - 1  == client_num :
            print("Invalid Destination")
            continue
        data = input(f"\nEnter Data to be sent to Station {dest_client} : ")
        if data == "exit":
            break
        res = "0"
        exit = False
        while res == "0":
            print("Sensing Channel\n")
            client.send(createFrame(str(dest_client - 1) + data + "0").encode(FORMAT))
            res = receive_message(client)
            if not res:
                print("Channel Destroyed")
                exit  = True
                break

            if res == "0" :
                print("Channel is Busy")
                time.sleep(1)
            else:
                sent = False
                while not sent:
                    print("Channel Idle")
                    prob = random.uniform(0,1)

                    if prob <= PROBABILITY :
                        client.send(createFrame(str(dest_client - 1) + data + "1").encode(FORMAT))
                        print("Packet Sent\n")
                        sent  = True
                    else:
                        print("Packet not Sent")
                        print("Waiting for Time Slot..\n")
                        time.sleep(2)
                        print("Sensing Channel\n")
                        client.send(createFrame(str(dest_client - 1) + data + "0").encode(FORMAT))
                        recv = receive_message(client);

                        if recv == "0":
                            print("Channel is Busy")
                            print("Waiting for BackOff Time...")
                            time.sleep(BACKOFF)
                            sent = True
                            res = "0"
        if exit:
            break

        print("\n")


    client.close()

if __name__ == "__main__":
    main()
```

OUTPUT - p-PERSISTENT

CHANNEL

```
E:\network\CSMA>python channel.py
Enter the number of Stations : 3
[STARTING] SHARED CHANNEL is starting
[LISTENING] Server is listening on 192.168.1.2:4456.
[NEW CONNECTION] ('192.168.1.2', 64113)  Station - 1
[NEW CONNECTION] ('192.168.1.2', 64114)  Station - 2
Ready for Communication
[NEW CONNECTION] ('192.168.1.2', 64115)  Station - 3
Ready for Communication
Ready for Communication
Channel is Busy

Collision Occured at Unit 6
Signal from Station - 1

Collision Occured at Unit 6
Signal from Station - 3

Data received by Station 1 : hi from s2
[DISCONNECTED] ('192.168.1.2', 64113) Station - 1
[DISCONNECTED] ('192.168.1.2', 64115) Station - 3
[DISCONNECTED] ('192.168.1.2', 64114) Station - 2

E:\network\CSMA>
```

STATION 1

```
E:\network\CSMA\p-Persistent>python station.py
[CONNECTED] Station connected to SHARED CHANNEL at 192.168.1.2:4456
There are 3 stations in the Shared Channel
You are Station - 1

Enter Station to send data : 3

Enter Data to be sent to Station 3 : hi from s1
Sensing to Channel

Channel Idle
Packet Sent
```

STATION 3

```
E:\network\CSMA\p-Persistent>python station.py
[CONNECTED] Station connected to SHARED CHANNEL at 192.168.1.2:4456
There are 3 stations in the Shared Channel
You are Station - 3

Enter Station to send data : 2

Enter Data to be sent to Station 2 : hi from s3
Sensing to Channel

Channel Idle
Packet Sent
```

STATION 2

```
E:\network\CSMA\p-Persistent>python station.py
[CONNECTED] Station connected to SHARED CHANNEL at 192.168.1.2:4456
There are 3 stations in the Shared Channel
You are Station - 2

Enter Station to send data : 1

Enter Data to be sent to Station 1 : hi from s2
Sensing to Channel

Channel Idle
Packet not Sent
Waiting for Time Slot..

Sensing to Channel

Channel Idle
Packet not Sent
Waiting for Time Slot..

Sensing to Channel

Collision has Occured
Waiting for BackOff Time...
Sensing to Channel

Channel Idle
Packet not Sent
Waiting for Time Slot..

Sensing to Channel

Channel Idle
Packet not Sent
Waiting for Time Slot..

Sensing to Channel

Channel Idle
Packet not Sent
Waiting for Time Slot..
```

```
Sensing to Channel

Collision has Occured
Waiting for BackOff Time...
Sensing to Channel

Channel Idle
Packet not Sent
Waiting for Time Slot..

Sensing to Channel

Channel Idle
Packet not Sent
Waiting for Time Slot..

Sensing to Channel

Channel Idle
Packet not Sent
Waiting for Time Slot..

Sensing to Channel

Channel Idle
Packet not Sent
Waiting for Time Slot..

Sensing to Channel

Channel Idle
Packet Sent
```

OUTPUT - NON-PERSISTENT / 1-PERSISTENT

CHANNEL

```
E:\NETWORK\Carrier Sense\CSMA>python channel.py
Enter the number of Stations : 3
[STARTING] SHARED CHANNEL is starting
[LISTENING] Server is listening on 192.168.1.7:4456.
[NEW CONNECTION] ('192.168.1.7', 55565)  Station 1
Ready for Communication
[NEW CONNECTION] ('192.168.1.7', 55566)  Station 2
[NEW CONNECTION] ('192.168.1.7', 55567)  Station 3
Ready for Communication
Ready for Communication
Collision Occured at Unit 3
Signal from Station - 2

Collision Occured at Unit 3
Signal from Station - 1

Collision Occured at Unit 8
Signal from Station - 2

Collision Occured at Unit 8
Signal from Station - 3

Collision Occured at Unit 6
Signal from Station - 3

Collision Occured at Unit 6
Signal from Station - 1

[DISCONNECTED] ('192.168.1.7', 55567) Station - 3
[DISCONNECTED] ('192.168.1.7', 55566) Station - 2
[DISCONNECTED] ('192.168.1.7', 55565) Station - 1

E:\NETWORK\Carrier Sense\CSMA>
```

STATION 1

```
E:\NETWORK\Carrier Sense\CSMA\Non-Persistent>python station.py
[CONNECTED] Station connected to SHARED CHANNEL at 192.168.1.7:4456
There are 3 stations in the Shared Channel
You are Station  1

Enter Station to send data : 2

Enter Data to be sent to Station 2 : hi from s1
Sensing Channel

Channel Idle, Packet Sent


Enter Station to send data : 3

Enter Data to be sent to Station 3 : hi how are you
Sensing Channel

Channel Idle, Packet Sent
```

STATION 2

```
E:\NETWORK\Carrier Sense\CSMA\Non-Persistent>python station.py
[CONNECTED] Station connected to SHARED CHANNEL at 192.168.1.7:4456
There are 3 stations in the Shared Channel
You are Station  2

Enter Station to send data : 3

Enter Data to be sent to Station 3 : from s2
Sensing Channel

Channel Idle, Packet Sent
```

STATION 3

```
E:\NETWORK\Carrier Sense\CSMA\Non-Persistent>python station.py
[CONNECTED] Station connected to SHARED CHANNEL at 192.168.1.7:4456
There are 3 stations in the Shared Channel
You are Station  3

Enter Station to send data : 1

Enter Data to be sent to Station 1 : hi from s3
Sensing Channel

Channel Idle, Packet Sent


Enter Station to send data : 1

Enter Data to be sent to Station 1 : i am fine
Sensing Channel

Channel Idle, Packet Sent
```

RESULTS

1-Persistent

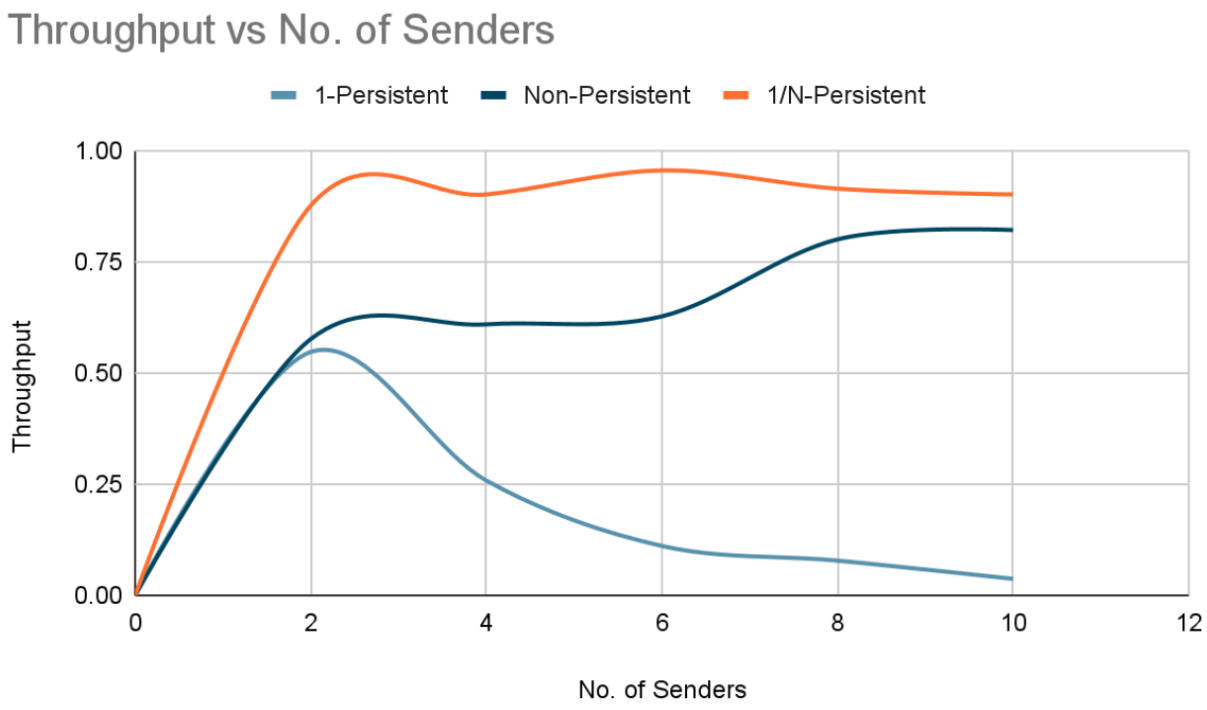
No of senders	Total packets sent	Effective packets sent	No. of collisions	Total time (in min)	Throughput	Delay per packet (in sec)
2	117	64	53	0.337	0.547	0.316
4	248	64	184	0.929	0.258	0.871
6	575	64	511	2.444	0.111	2.292
8	812	64	748	3.804	0.078	3.566
10	1692	64	1628	7.870	0.037	7.378

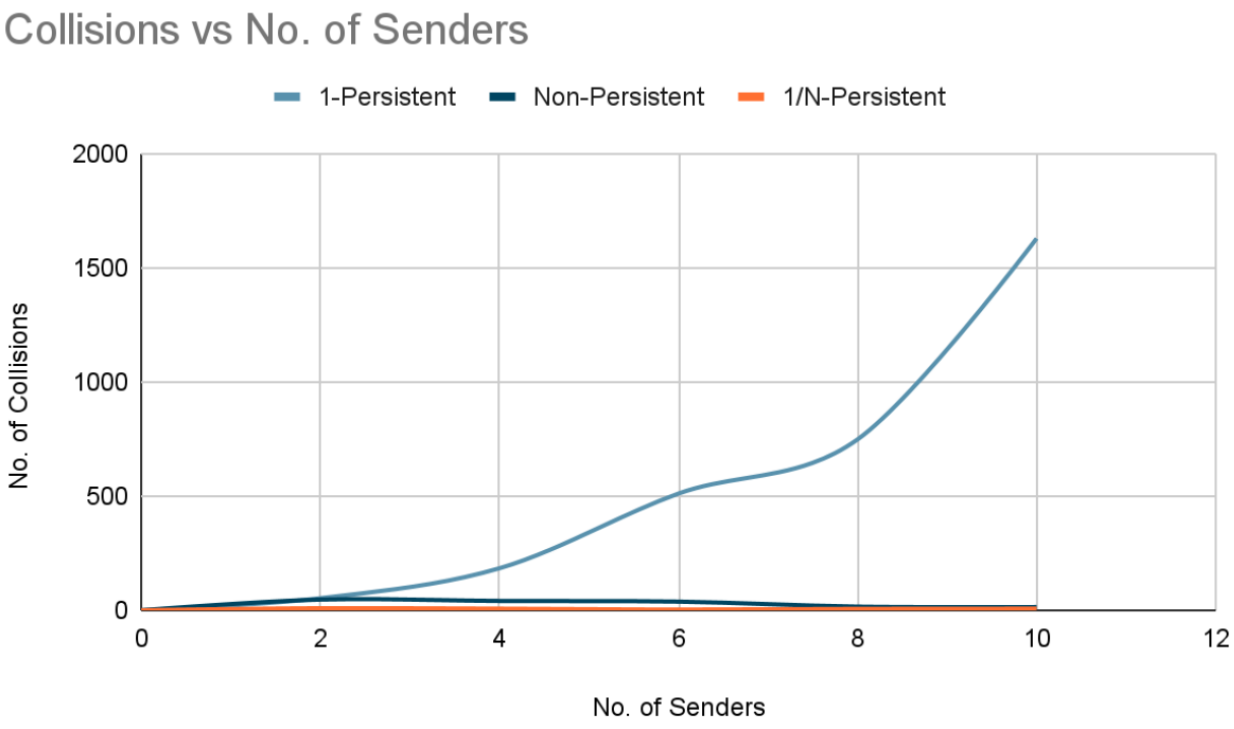
Non-Persistent

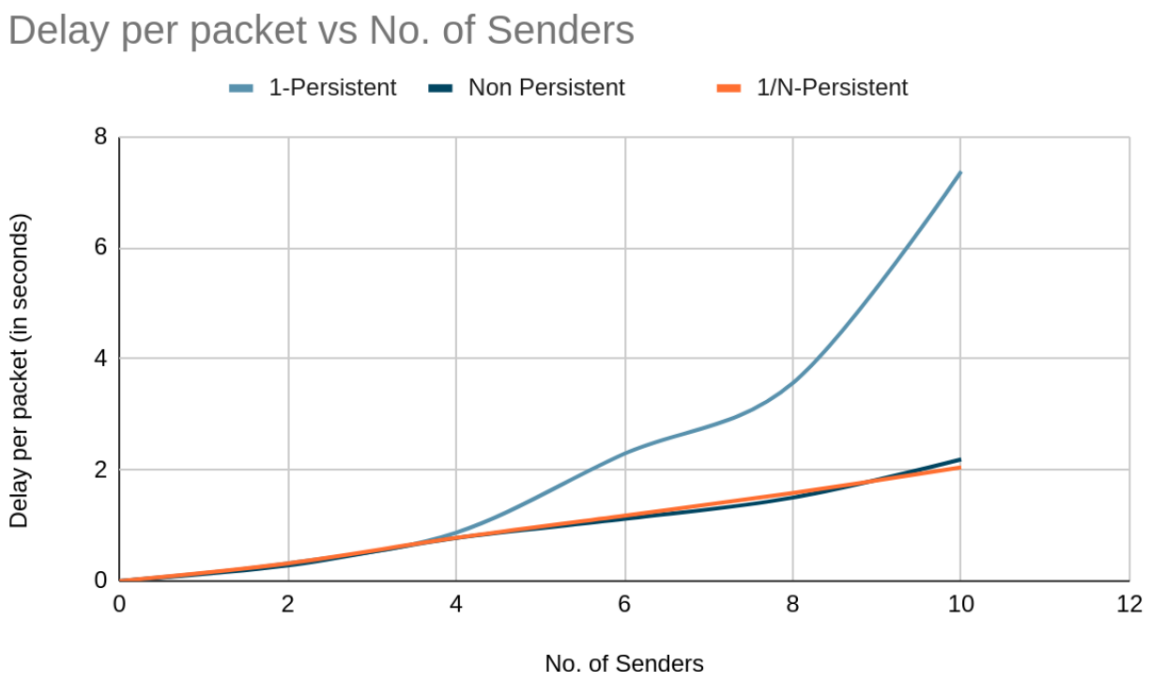
No of senders	Total packets sent	Effective packets sent	No. of collisions	Total time (in min)	Throughput	Delay per packet (in sec)
2	111	64	47	0.297	0.576	0.279
4	105	64	41	0.826	0.609	0.774
6	102	64	38	1.191	0.627	1.116
8	80	64	16	1.602	0.800	1.502
10	78	64	14	2.189	0.821	2.189

P-Persistent(1/N)

No of senders	Total packets sent	Effective packets sent	No. of collisions	Total time (in min)	Throughput	Delay per packet (in sec)
2	73	64	9	0.338	0.876	0.317
4	71	64	7	0.829	0.901	0.777
6	67	64	3	1.256	0.955	1.178
8	70	64	6	1.690	0.914	1.585
10	71	64	7	2.181	0.901	2.045


Throughput vs No. of Senders
1-Persistent
Non-Persistent
1/N-Persistent
1.00
0.75
0.50
0.25
0.00
Throughput
0
2
4
6
8
10
12
No. of Senders


Collisions vs No. of Senders
1-Persistent
Non-Persistent
1/N-Persistent
2000
1500
1000
500
0
No. of Collisions
0
2
4
6
8
10
12
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.