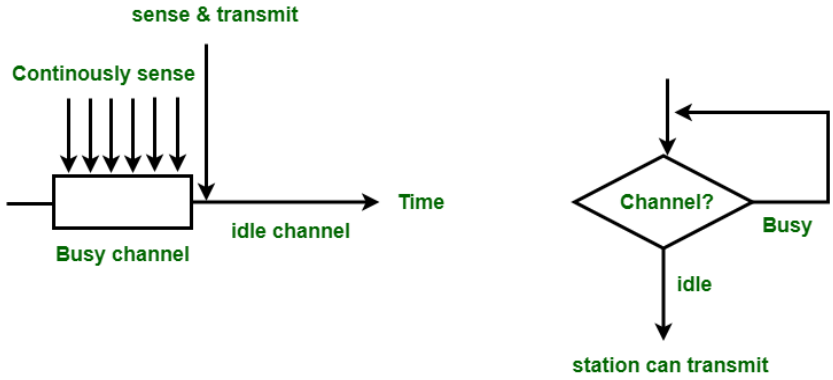
Assignment 3 : Computer Networks Lab

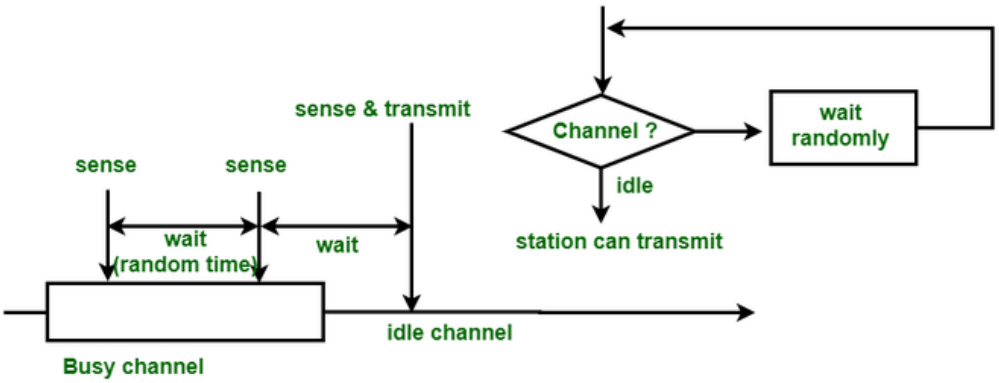
Tonmoy Biswas 002110501133

Theory Research

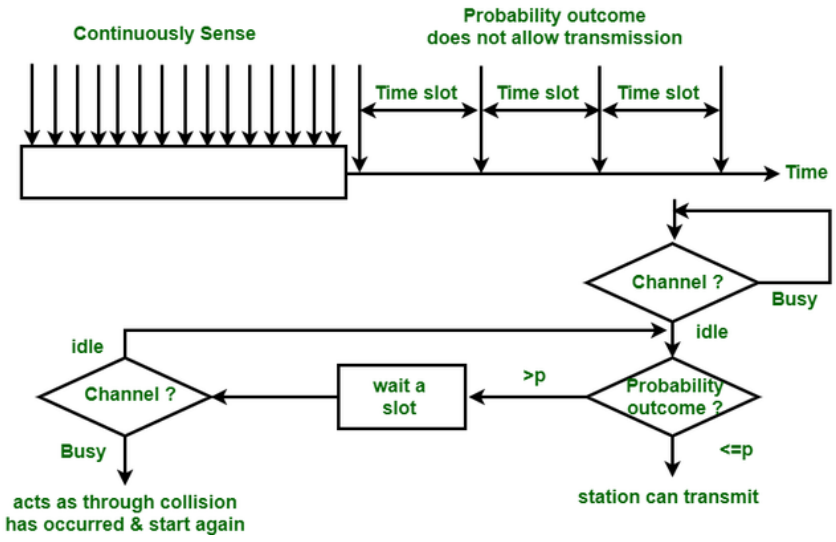
1. 1-Persistent CSMA (Carrier Sense Multiple Access) : 1-Persistent CSMA is a network access technique used in Ethernet and similar systems. When a station wants to transmit data, it first listens to the channel. If the channel is busy, it continues to sense it persistently until it becomes idle. Once the channel is free, the station transmits its data. This approach ensures immediate transmission when the channel becomes available, but it can lead to collisions if multiple stations sense the channel as idle simultaneously.



2. Non-Persistent CSMA: Non-Persistent CSMA is another variant of CSMA used in network protocols. In this method, when a station wants to transmit, it first checks the channel. If the channel is busy, it waits for a random period before retrying. This random backoff helps reduce the likelihood of multiple stations repeatedly trying to transmit simultaneously, which can lead to fewer collisions compared to 1-Persistent CSMA.



3. P-Persistent CSMA: P-Persistent CSMA is a modification of CSMA used in wireless networks like IEEE 802.11 (Wi-Fi). In this approach, when a station wants to transmit, it checks the channel. If it's busy, it waits for a random amount of time based on a probability distribution. The probability of transmission is "p," and if p=1, it behaves like 1-Persistent CSMA. If p=0, it behaves like Non-Persistent CSMA. By adjusting the probability "p," the network can balance the trade-off between channel efficiency and collision avoidance.



Code Approach

I have implemented the error detection module in three program files.

- sender.py (Sender program)
- receiver.py (Receiver program)
- channel.py (Channel program)

The individual files fulfils different assignment purposes, following which have been explained in details :

1. sender.py – The following are the tasks performed in this Sender program :
    a. The data is entered by the user.
    b. It follows the above design to transmit the data.
2. receiver.py – The following are the tasks performed in this Receiver program :
    a. The data is received from one of the sender processes.
3. channel.py – The following are the tasks performed in this Channel program :
    a. Asks for number of sender processes and receiver processes.
    b. Initiates all sender and receiver processes.
    c. Receives data from one sender at a time.
    d. When the channel receives data from a sender, it changes its state to Busy. (The duration is set by the sender process).
    e. Sends the received data to one of the receiver (chosen randomly).

Implementation

Code Snippet of channel.py:

```
import socket
import time
import subprocess
import random
import os


class Channel():
   def __init__(self, totalsender, totalreceiver):
       self.totalsender = totalsender
       self.senderhost = '127.0.0.1'
       self.senderport = 8080
       self.senderconn = []
       self.totalreceiver = totalreceiver
       self.receiverhost = '127.0.0.2'
       self.receiverport = 9090
       self.receiverconn = []

   def initSenders(self):
       senderSocket = socket.socket()
       senderSocket.bind((self.senderhost, self.senderport))
       senderSocket.listen(self.totalsender)
       for i in range(1, self.totalsender+1):
           conn = senderSocket.accept()
           self.senderconn.append(conn)
       print('Initiated all sender connections')

   def closeSenders(self):
       for conn in self.senderconn:
           conn[0].close()
       print('Closed all sender connections')

   def initReceivers(self):
       receiverSocket = socket.socket()
       receiverSocket.bind((self.receiverhost, self.receiverport))
       receiverSocket.listen(self.totalreceiver)
       for i in range(1, self.totalreceiver+1):
```

```
            conn = receiverSocket.accept()
            self.receiverconn.append(conn)
        print('Initiated all receiver connections')

    def closeReceivers(self):
        for conn in self.receiverconn:
            conn[0].close()
        print('Closed all receiver connections')

    def processData(self):
        fileout = open('status.txt', "w")
        fileout.write(str(0))
        fileout.close()
        while True:
            for i in range(len(self.senderconn)):
                print()
                conn = self.senderconn[i]
                fileout = open('status.txt', "w")
                fileout.write(str(0))
                fileout.close()
                data = conn[0].recv(1024).decode()
                fileout = open('status.txt', "w")
                fileout.write(str(1))
                fileout.close()
                if not data:
                    break
                if data == 'q0':
                    break
                print('Received from Sender', i+1, ':', str(data))
                recvno = random.randint(0, len(self.receiverconn)-1)
                print('Sending to Receiver', recvno+1)
                rconn = self.receiverconn[recvno]
                rconn[0].sendto(data.encode(), rconn[1])
                if data == 'q0':
                    break
            return


if __name__ == '__main__':
    totalsen = int(input('Enter number of senders: '))
    totalrecv = int(input('Enter number of receivers: '))
    ch = Channel(totalsen, totalrecv)
```

```
    ch.initSenders()
    ch.initReceivers()
    ch.processData()
    ch.closeSenders()
    ch.closeReceivers()
```

Code Snippet of sender.py:

```
import socket
import sys
import time
import random

def Main(senderno):
    print('Initiating Receiver #', senderno)
    host = '127.0.0.2'
    port = 9090
    mySocket = socket.socket()
    mySocket.connect((host, port))
    while True:
        print()
        data = mySocket.recv(1024).decode()
        if not data:
            break
        if data == 'q':
            break
        print('Received from channel:', str(data))
    mySocket.close()

if __name__ == '__main__':
    if len(sys.argv) > 1:
        senderno = int(sys.argv[1])
    else:
        senderno = 1
    Main(senderno)
```

Code Snippet of receiver.py:

```
import socket
import sys
import time
import random


def Main(senderno):
   print('Initiating Sender #', senderno)
   host = '127.0.0.1'
   port = 8080
   mySocket = socket.socket()
   mySocket.connect((host, port))
   prevtime = time.time()
   success = 0

   while True:
       print()
       data = input("Enter $ ")
       k = 0
       kmax = 15

       while True:
           print('ATTEMPT NUMBER', str(k))
           print('Checking channel status ...')
           filein = open('status.txt', "r")
           status = int(filein.read())

           if status == 0:
               print("Channel is IDLE!")
               prob = random.uniform(0, 1)
               print('Probability value is:', str(prob))
               print()

               if prob <= 0.5:
                   fileout = open('status.txt', "w")
                   fileout.write(str(1))
                   fileout.close()
                   waittime = random.randint(3, 7)
                   print('Channel has been captured. It will take ' + str(waittime) +
```

```
                    ' seconds to send!')
                print('Sending to channel:', str(data))
                time.sleep(waittime)
                mySocket.send(data.encode())
                success += 1
                break
            else:
                print('Waiting for time-slot 2 seconds')
                time.sleep(2)
                filein = open('status.txt', "r")
                status = int(filein.read())
                print('After waiting for 2 seconds, the channel is', end=' ')

                if status == 0:
                    print('IDLE')
                else:
                    print('BUSY')

                k += 1

                if k > kmax:
                    print("Transmission aborted!")
                    break

                if status == 0:
                    continue
                else:
                    r = random.randint(0, pow(2, k) - 1)
                    print("Waiting for back off period")

        elif status == 1:
            time.sleep(0.1)
            print("Channel is BUSY!")
            print()

    if not data:
        break

    if data == 'q':
        break

    print("------------------------------")
```

```
        curtime = time.time()
        totaltime = curtime - prevtime
        throughput = success / totaltime
        print("Throughput:", str(throughput))
        mySocket.close()


if __name__ == '__main__':
    if len(sys.argv) > 1:
        senderno = int(sys.argv[1])
    else:
        senderno = 1
    Main(senderno)
```

Test Cases

```
Initiating Sender # 1
Enter $ 1001
ATTEMPT NUMBER 0
Checking channel status ...
Channel is IDLE!
Probability value is: 0.14882951956621937

Channel has been captured. It will take 3 seconds to send!
Sending to channel: 1001
```

```
Initiating Sender # 2
Enter $ 0001
ATTEMPT NUMBER 0
Checking channel status ...
Channel is BUSY!
ATTEMPT NUMBER 0
Checking channel status ...
Channel is BUSY!
ATTEMPT NUMBER 0
Checking channel status ...
Channel is BUSY!
ATTEMPT NUMBER 0
Checking channel status ...
Channel is BUSY!
ATTEMPT NUMBER 0
Checking channel status ...
Channel is BUSY!
ATTEMPT NUMBER 0
Checking channel status ...
Channel is BUSY!
ATTEMPT NUMBER 0
Checking channel status ...
Channel is BUSY!
ATTEMPT NUMBER 0
Checking channel status ...
Channel is BUSY!
ATTEMPT NUMBER 0
Checking channel status ...
Channel is BUSY!
ATTEMPT NUMBER 0
Checking channel status ...
Channel is BUS
```

Result and Analysis

p-persistent CSMA:

- This approach is employed when the channel has time slots, and the duration of each time slot is equal to or greater than the maximum propagation delay time.
- When a station is ready to send, it first senses the channel.
- If the channel is busy, the station waits until the next time slot.
- If the channel is idle, the station decides to transmit with a probability denoted as "p."
- With a complementary probability, denoted as "q" (where q = 1 - p), the station waits for the beginning of the next time slot.
- If the next slot is idle, the station repeats the decision process, choosing to either transmit or wait again based on probabilities p and q.
- This process continues until either the frame is successfully transmitted or another station begins transmitting.
- In the event of another station's transmission, the station treats it as a collision, waits for a random duration, and then reinitiates the process.

Advantages of p-persistent CSMA: It reduces the likelihood of collision, thereby enhancing network efficiency.

Feedback

Completing this assignment has deepened my comprehension of CSMA and CSMA/CD, as it required hands-on implementation of these concepts.