Assignment 4 : Computer Networks Lab

Tonmoy Biswas 002110501133

Theory Research

CDMA, or Code Division Multiple Access, is a digital cellular technology that enables multiple users to share the same frequency band simultaneously. Each user is allocated a unique code to differentiate their signals, allowing for concurrent transmission and reception without interference. In the context of this assignment, CDMA is employed for multiple access of a common channel by 'n' stations.

Unique Code Words: In CDMA, each sender uses a unique code word for encoding its data. These code words are typically derived from mathematical structures known as Walsh codes, forming a set referred to as the Walsh set.

Walsh Set: The Walsh set consists of orthogonal binary sequences that are mutually exclusive. These sequences are used to encode and decode data, ensuring that each user's signal can be uniquely identified and reconstructed.

Orthogonal Codes: Walsh codes are orthogonal, meaning that their cross-correlation is zero. This orthogonality property is crucial in CDMA, as it allows for simultaneous transmission and reception without interference.

Code Construction: Walsh codes are constructed using Hadamard matrices. Each row of the Hadamard matrix provides a unique Walsh code, ensuring orthogonality among the codes.

Encoding: Each sender encodes its data using its assigned Walsh code. This process involves multiplying the data by the corresponding Walsh code, effectively modulating the signal.

Transmission: The modulated signals from all users are transmitted over the common channel simultaneously. The use of orthogonal codes minimizes the likelihood of interference.

Decoding: At the receiving end, each station uses its designated Walsh code to decode the incoming signal. The orthogonality ensures that the correct data is reconstructed at each station.

Design

I have implemented the error detection module across three program files: channel.py (Program for the channel) station.py (Program for a station process) These individual files serve distinct assignment purposes, detailed as follows:

channel.py

- Solicits the number of stations and initializes all stations.
- Generates a Walsh Table for the given number of stations, considering the highest power of 2.
- Accepts data bits from each station.
- Multiplies data bits with the corresponding Walsh Sequence of each station and sums them to obtain the final data.
- Requests the sender and receiver station numbers to determine the source and destination of the data bit.
- Computes the data bit by multiplying the final data with the Walsh sequence of the sender station, summing it up, and then dividing the result by the number of stations.

station.py

- Sends a stream of data bits to the channel process.
- If a station sends a stream with a length less than X (where X is the maximum length of data bits sent by a station), the remaining bits are assumed to be silent.
- Receives a data bit from the channel.

These programs collectively facilitate an error detection mechanism by interacting between the channel and station processes, ensuring the proper flow and processing of data bits.

Codes

channel.py

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

def multiplyTuples(t1, t2):
    tup = []
    for i in range(len(t1)):
        tup.append(t1[i] * t2[i])
    return tup

def multiplyScalar(t1, x):
    tup = []
    for i in range(len(t1)):
        tup.append(t1[i] * x)
    return tup

def addTuples(t1, t2):
    tup = []
    for i in range(len(t1)):
        tup.append(t1[i] + t2[i])
```

```
    return tup

class Channel():
    def __init__(self, totalstations):
        self.totalstation = totalstations
        self.stationhost = '127.0.0.1'
        self.stationport = 8080
        self.stationconn = []
        self.walshtable = [[]]

    def initStations(self):
        stationSocket = socket.socket()
        stationSocket.bind((self.stationhost, self.stationport))
        stationSocket.listen(self.totalstation)
        for i in range(1, self.totalstation + 1):
            conn = stationSocket.accept()
            self.stationconn.append(conn)
        print('Initiated all station connections')

    def closeStations(self):
        for conn in self.stationconn:
            conn[0].close()
        print('Closed all station connections')

    def generateWalshTable(self):
        print('Generating Walsh table ...')
        n = self.totalstation
        p = 1
        prevtable = [[1]]
        while p < n:
            p *= 2
            curtable = []
            for i in range(p):
                tup = []
                for j in range(p):
                    tup.append(0)
                curtable.append(tup)
            for i in range(0, p // 2):
                for j in range(0, p // 2):
                    curtable[i][j] = prevtable[i][j]
                    curtable[i + p // 2][j] = prevtable[i][j]
                    curtable[i][j + p // 2] = prevtable[i][j]
```

```
                curtable[i + p // 2][j + p // 2] = -1 * prevtable[i][j]
        prevtable = curtable
    self.walshtable = prevtable
    print('Printing Walsh table:')
    for i in range(p):
        for j in range(p):
            if self.walshtable[i][j] == 1:
                print(end=' ')
            print(self.walshtable[i][j], end=' ')
        print()

def process(self):
    data = []
    for i in range(self.totalstation):
        conn = self.stationconn[i]
        d = conn[0].recv(1024).decode()
        data.append(d)

    for i in range(self.totalstation):
        print('Station', i + 1, 'will send data:', end=' ')
        print(data[i])

    maxlen = 0
    for i in data:
        maxlen = max(maxlen, len(i))

    datavalue = []
    for d in data:
        tup = []
        for j in range(maxlen):
            if j < len(d):
                if d[j] == '0':
                    tup.append(-1)
                elif d[j] == '1':
                    tup.append(1)
                else:
                    tup.append(0)
        datavalue.append(tup)

    for i in range(maxlen):
        print('-----------------------------------------')
        print('Sending bit', i + 1, 'of each station\'s data')
```

```
            finaldata = []
            d = []
            c = []
            n = len(self.walshtable)

            for j in range(n):
                if j < self.totalstation:
                    d.append(datavalue[j][i])
                else:
                    d.append(1)
                c.append(self.walshtable[j])
                finaldata.append(0)

            for j in range(n):
                temp = multiplyScalar(c[j], d[j])
                finaldata = addTuples(finaldata, temp)

            print('Bit', i + 1, 'of each station is:', end=' ')
            print(d)
            print()
            print('After multiplying data bit with code bits of corresponding stations,
and adding them all,')
            print('Final data is:', end=' ')
            print(finaldata)
            print()

            choice = input('Does any station want to receive data? (y/n) ')

            while choice == 'y':
                stnum, renum = input('Enter the sender and receiver station number:
').split()
                stnum = int(stnum)
                renum = int(renum)

                if stnum > self.totalstation or stnum <= 0 or renum > self.totalstation
or renum <= 0:
                    print('Invalid station number')
                else:
                    temp = multiplyTuples(finaldata, c[stnum - 1])
                    print('Multiplying final data with Code bits of sender station',
stnum)
                    print('The result is:', end=' ')
```

```
                    print(temp)
                    summ = sum(temp)
                    print('The sum of result is:', summ)
                    databit = str(summ // n)
                    print('THE DATA BIT OF STATION', stnum, 'is:', databit)
                    conn = self.stationconn[renum - 1]
                    conn[0].sendto(databit.encode(), conn[1])
                    print('Data bit sent to receiver', renum, 'successfully!')
                    print()

                    choice = input('Does any station want to receive data? (y/n) ')

if __name__ == '__main__':
    totalstations = int(input('Enter number of stations: '))
    ch = Channel(totalstations)
    ch.generateWalshTable()
    ch.initStations()
    ch.process()
    ch.closeStations()
```

station.py

```
import socket
import sys
import time
import random

def Main(senderno):
    print('Initiating Station #', senderno)
    host = '127.0.0.1'
    port = 8080
    mySocket = socket.socket()
    mySocket.connect((host, port))

    data = input('Enter $ ')
    mySocket.send(data.encode())

    while True:
        data = mySocket.recv(1024).decode()

        if not data:
```

```
            break

        print('Received bit value from channel:', str(data))
        data = int(data)

        if data == -1:
            val = 0
        elif data == 1:
            val = 1
        else:
            val = "silent"

        print('VALUE OF RECEIVED BIT IS:', str(val))
        mySocket.close()

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

```
Enter number of stations: 4

Generating Walsh table ...
Printing Walsh table :
1  1  1  1
1 -1  1 -1
1  1 -1 -1
1 -1 -1  1

Initiated all station connections

Station 1 will send data: O1
Station 2 will send data: 1
Station 3 will send data: 1O
Station 4 will send data: 1

-----------------------------------------
Sending bit 1 of each station's data
Bit 1 of each station is: [-1, 1, 1, 1]
After multiplying data bit with code bits of corresponding stations, and adding them
Final data is: [2, -2, -2, -2]
```

```
Does any station want to receive data? (y/n) y
Enter the sender and receiver station number: 4 2
Multiplying final data with Code bits of sender station 4
The result is : [2, 2, 2, -2]
The sum of result is: 4
THE DATA BIT OF STATION 4 is: 1
Data bit sent to receiver 2 successfully!

Does any station want to receive data? (y/n) y
Enter the sender and receiver station number: 3 1
Multiplying final data with Code bits of sender station 3
The result is : [2, -2, 2, 2]
The sum of result is: 4
THE DATA BIT OF STATION 3 is: 1
Data bit sent to receiver 1 successfully!

Does any station want to receive data? (y/n) y
Enter the sender and receiver station number: 2 3
Multiplying final data with Code bits of sender station 2
The result is : [2, 2, -2, 2]
The sum of result is: 4
THE DATA BIT OF STATION 2 is: 1
Data bit sent to receiver 3 successfully!
```

```
Does any station want to receive data? (y/n) y
Enter the sender and receiver station number: 1 4
Multiplying final data with Code bits of sender station 1
The result is : [2, -2, -2, -2]
The sum of result is: -4
THE DATA BIT OF STATION 1 is: -1
Data bit sent to receiver 4 successfully!

Does any station want to receive data? (y/n) n

-------------------------------------------
Sending bit 2 of each station's data
Bit 2 of each station is: [1, 0, -1, 0]
After multiplying data bit with code bits of corresponding stations, and adding them
Final data is: [0, 0, 2, 2]

Does any station want to receive data? (y/n) y
Enter the sender and receiver station number: 2 3
Multiplying final data with Code bits of sender station 2
The result is : [0, 0, 2, -2]
The sum of result is: 0
THE DATA BIT OF STATION 2 is: 0
Data bit sent to receiver 3 successfully!

Does any station want to receive data? (y/n) n
```

Station.py (1st station):

```
Initiating Station # 1
Enter $ 01
Received bit value from channel: 1
VALUE OF RECEIVED BIT IS : 1
```

Station.py (2nd station):

```
Initiating Station # 2
Enter $ 1
Received bit value from channel: 1
VALUE OF RECEIVED BIT IS : 1
```

Station.py (3rd station):

```
Initiating Station # 3
Enter $ 10
Received bit value from channel: 1
VALUE OF RECEIVED BIT IS : 1
Received bit value from channel: 0
VALUE OF RECEIVED BIT IS : silent
```

Station.py (4th station):

```
Initiating Station # 4
Enter $ 1
Received bit value from channel: -1
VALUE OF RECEIVED BIT IS : 0
```

RESULTS & ANALYSIS

Unlike TDMA, CDMA allows all stations to transmit data simultaneously without timesharing.
CDMA enables each station to transmit over the entire frequency spectrum continuously.
Multiple simultaneous transmissions are differentiated using coding theory.
Each user in CDMA is assigned a unique code sequence.

The fundamental concept of CDMA is outlined below:

1. Consider four stations labeled 1, 2, 3, and 4, connected to the same channel. The data from station 1 is denoted as d_1, from station 2 as d_2, and so forth.
2. Each station is assigned a code (e.g., C_1, C_2), with the following properties:
   (a) The product of any two codes results in 0.
   (b) The product of a code with itself results in the number of stations (4 in this case).
3. When these stations send data simultaneously, each station multiplies its data by its corresponding code (e.g., $d_1 \cdot C_1$, $d_2 \cdot C_2$).
4. The data on the channel is the sum of these terms.
5. To receive data, a station multiplies the channel data by the code of the sender.
6. Because the product of a code with itself is 4, while the product of different codes is 0, the receiving station divides the result by the number of stations to obtain the sender's data.
   $data = (d_1 \cdot C_1 + d_2 \cdot C_2 + d_3 \cdot C_3 + d_4 \cdot C_4) \cdot C_1 = 4 \cdot d_1$
   • Each station's assigned code is a sequence of numbers called chips, known as orthogonal sequences, with the following properties:
7. Each sequence consists of N elements, where N is the number of stations.
8. Multiplying a sequence by a scalar multiplies each element in the sequence by that scalar.
9. The inner product of two equal sequences results in N.
10. The inner product of two different sequences results in 0.
11. Adding two sequences means adding corresponding elements to create another sequence.
    • Data representation and encoding by stations:
12. 0 bits are encoded as -1, and 1 bits are encoded as +1.
13. During idle periods, stations send no signal, interpreted as a 0.
    • For example, if stations 1 and 2 send 0 bits, station 3 is silent, and station 4 sends a 1 bit, the data at sender sites are represented as -1, -1, 0, and +1 respectively.
    • Each station multiplies its number by its unique chip, and the station sends this sequence to the channel. The channel sequence is the sum of all four sequences.
    • If station 3, previously silent, is listening to station 2, it multiplies the total channel data by the code for station 2, obtaining $[-1, -1, -3, +1] \cdot [1, -1, +1, -1] = -4/4 = -1$, representing bit 0.

FEEDBACK

This assignment has enhanced my comprehension of constructing the Walsh Table for a specified number of stations and elucidated the encoding and decoding

process of data bits by the CDMA channelization protocol, allowing simultaneous communication from all stations.