COMPUTER NETWORKS LAB

REPORT ASSIGNMENT-4

NAME: SOHAM LAHIRI

CLASS: BCSE UG-III 5TH SEMESTER

ROLL NO: 002210501107

GROUP: A3

SUBMISSION DATE: 18/11/2024

Problem Statement:

Implement CDMA with Walsh code.

In this assignment you have to implement CDMA for multiple access of a common channel by n stations. Each sender uses a unique code word, given by the Walsh set, to encode its data, send it across the channel, and then perfectly reconstruct the data at n stations.

DESIGN

Theory:

Introduction to CDMA:

Code Division Multiple Access (CDMA) is a communication technique that allows multiple users to access a single communication channel simultaneously. Each user is assigned a unique code, allowing their signals to overlap in time and frequency without interference. CDMA ensures secure and efficient communication by using orthogonal codes.

Walsh Code:

Walsh codes are a set of orthogonal binary sequences used in CDMA to encode and decode data. These codes enable multiple users to share the same communication channel without interference, as their orthogonality ensures that the cross-correlation between any two codes is zero.

Properties of Walsh Codes:

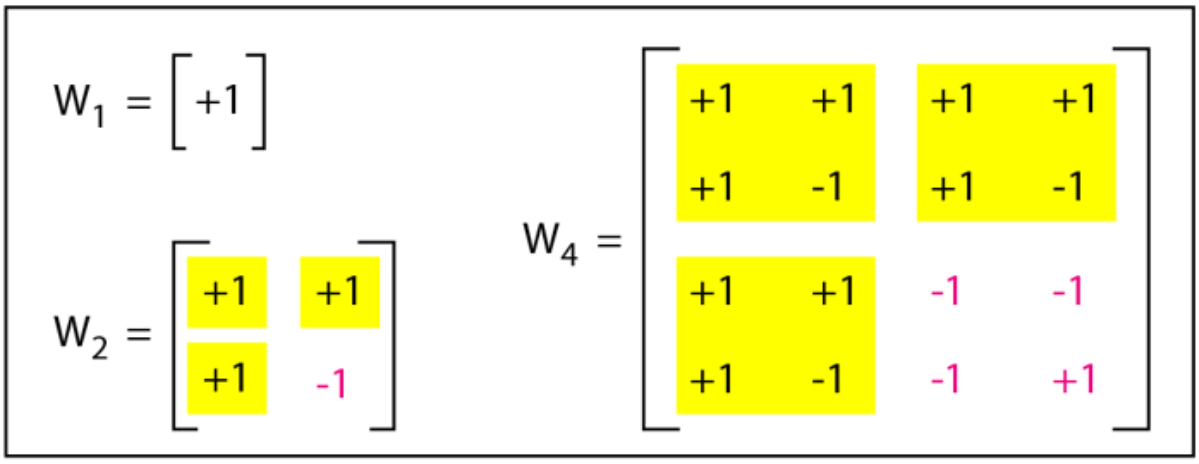
1. Orthogonality: The dot product of any two different Walsh codes is zero, meaning they are mutually exclusive.
2. Length and Number of Codes: For n users, Walsh codes of length 2^m are used, where m≥log(n)/log2.
3. Construction: Walsh codes are constructed recursively using the Hadamard matrix.

Hadamard Matrix Construction:

The Walsh code set is derived from the Hadamard matrix, which is defined as follows:

$$W_1 = \begin{bmatrix} +1 \end{bmatrix} \qquad W_{2N} = \begin{bmatrix} W_N & W_N \\ W_N & \overline{W_N} \end{bmatrix}$$

a. Two basic rules



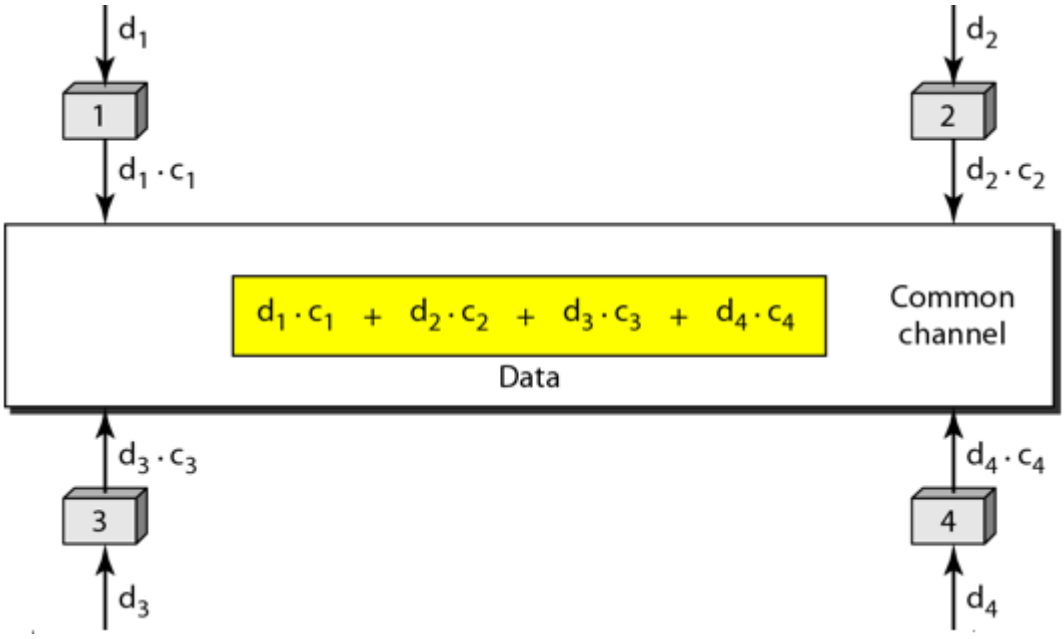
b. Generation of W_1, W_2, and W_4

Each row of the Hadamard matrix represents a Walsh code.

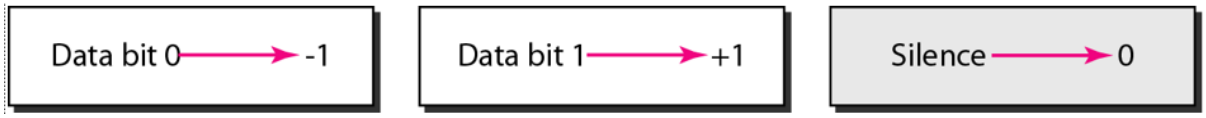
Encoding and Decoding in CDMA

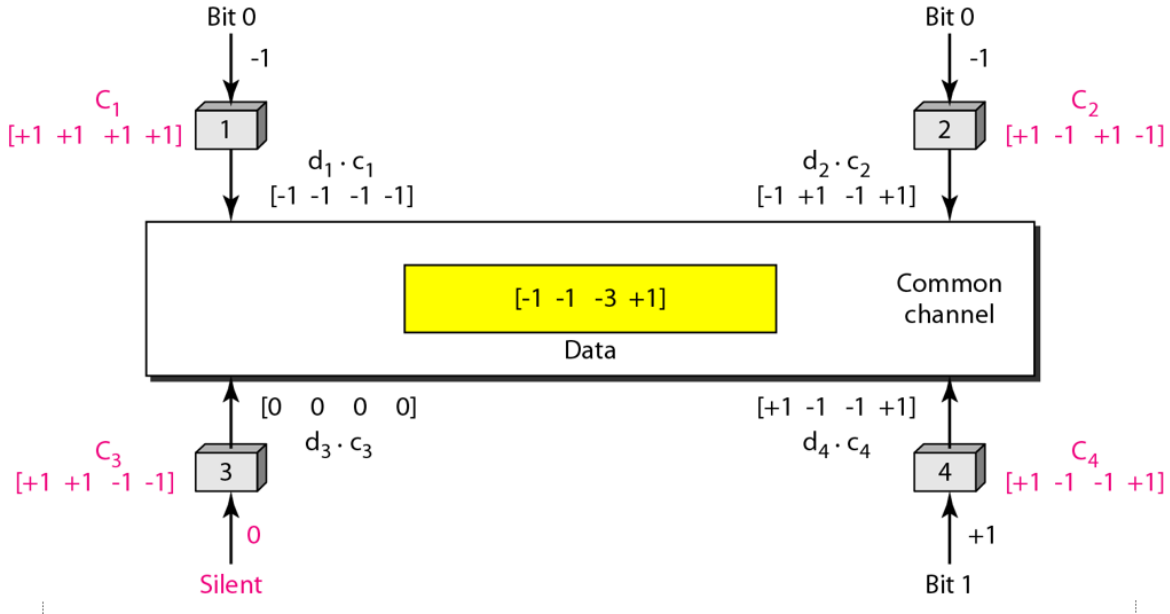
1. Encoding: Each sender multiplies its data with a unique Walsh code. This spreads the signal over the channel using the code.
   Encoded Signal=Data Bit×Walsh Code
2. Transmission: The encoded signals from all users are summed together and transmitted over the shared channel.
3. Decoding: Each receiver multiplies the received signal with the Walsh code assigned to the sender it wants to decode. Orthogonality ensures that only the intended signal is reconstructed.

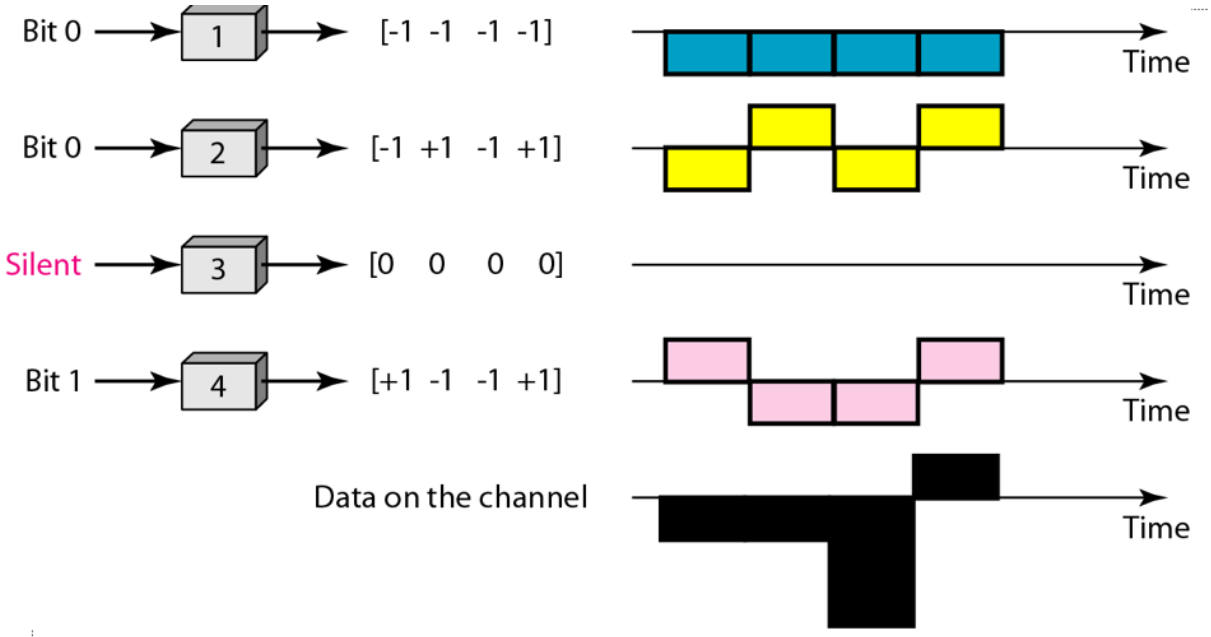
Working Diagram:

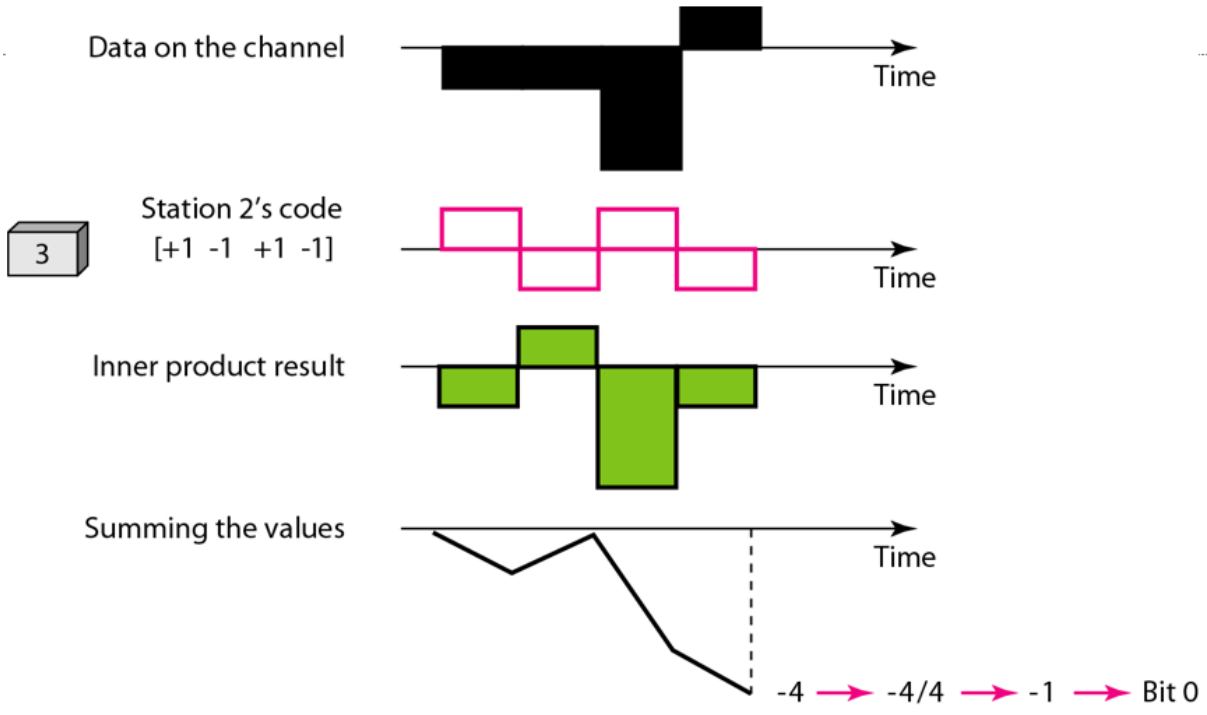


Example:


Data bit 0 → -1
Data bit 1 → +1
Silence → 0


Bit 0
-1
C1
[+1 +1 +1 +1]
1
d1 · c1
[-1 -1 -1 -1]
Bit 0
-1
C2
[+1 -1 +1 -1]
2
d2 · c2
[-1 +1 -1 +1]
[-1 -1 -3 +1]
Data
Common channel
[0 0 0 0]
d3 · c3
C3
[+1 +1 -1 -1]
3
0
Silent
[+1 -1 -1 +1]
d4 · c4
4
C4
[+1 -1 -1 +1]
+1
Bit 1


Bit 0
1
[-1 -1 -1 -1]
Time
Bit 0
2
[-1 +1 -1 +1]
Time
Silent
3
[0 0 0 0]
Time
Bit 1
4
[+1 -1 -1 +1]
Time
Data on the channel
Time


Data on the channel
Time
Station 2's code
[+1 -1 +1 -1]
3
Time
Inner product result
Time
Summing the values
Time
-4
-4/4
-1
Bit 0

Implementation:

Sender Program:

```
import numpy as np
import math
import pickle
def code_size(n):
    N=2**math.ceil(math.log(n)/math.log(2))
    return N
def create_code(n):
    r=range(code_size(n))
    return np.array([[int(bin(x&y),13)%2or-1for x in r]for y in r])
def code_data(d):
    code=create_code(len(d))
    size=code_size(len(d))
    coded=np.zeros((size,len(d)))
    coded=(code*d).T
    return np.sum(coded,axis=0)
def get_data(num_senders):
    data=[]
    for i in range(num_senders):
        inp=input('Enter bit')
        if len(inp)==0:
            data.append(0)
        elif int(inp)==1:
            data.append(1)
        else:
            data.append(-1)
    return np.array(data)
import socket
def send(num_senders,host,port):
    data=get_data(num_senders)
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        # Connect to the receiver
        s.connect((host, port))
        data=code_data(data)
        serialized_data = pickle.dumps(data)
        # Send the serialized data
        s.sendall(serialized_data)
host='localhost'
port=64000
num_senders=8
send(num_senders,host,port)
```

Explanation:

1. code_size(n)
Purpose:

- Determines the size of the Walsh code matrix based on the number of senders (n).

How it works:

- Computes the smallest power of 2 greater than or equal to the number of senders, N=2^⌈log2(n)⌉.
- This ensures the Walsh code matrix has sufficient orthogonal rows to assign a unique code to each sender.

2. create_code(n)
Purpose:

- Generates the Walsh code matrix of size N×N, where N=2^⌈log2(n)⌉.

How it works:

- Uses binary operations to create a Walsh code matrix. Each element of the matrix is computed as:

$$\text{code}[i][j] = \begin{cases} 1, & \text{if (binary AND of i and j)} = 0 \\ -1, & \text{otherwise} \end{cases}$$

- Converts the result to either 1 or -1 for encoding.

3. code_data(d)
Purpose:

- Encodes the input data from multiple senders using their respective Walsh codes.

How it works:

1. Calls create_code(n) to generate a Walsh code matrix.
2. Multiplies each sender's data bit with its corresponding Walsh code (row-wise).
3. Sums the encoded values across all senders to generate the composite signal.

4. get_data(num_senders)
Purpose:

- Collects input data from users (one bit per sender) and converts it into an array for encoding.

How it works:

1. Loops through the number of senders.
2. Accepts a bit (1, 0, or -1) from the user for each sender.
    - If no input is provided, defaults to 0.
    - Converts input into the range [−1,1][-1, 1][−1,1] (e.g., 1 for '1', -1 for '0').

5. send(num_senders, host, port)

Purpose:

- Encodes data from multiple senders and sends the composite signal to a receiver over a network socket.

How it works:

1. Calls get_data(num_senders) to collect sender data.
2. Encodes the data using code_data(d).
3. Serializes the encoded signal using pickle for transmission.
4. Establishes a TCP connection to the receiver at the specified host and port.
5. Sends the serialized signal over the socket.

Receiver Program:

```
import numpy as np
import math
import pickle
def code_size(n):
    N=2**math.ceil(math.log(n)/math.log(2))
    return N
def create_code(n):
    r=range(code_size(n))
    return np.array([[int(bin(x&y),13)%2or-1for x in r]for y in r])
def decode_data(data):
    code=create_code(len(data))
    stations=len(data)
    return np.sum(code*data,axis=1)/stations
import socket
def receive(host, port):
    # Create a socket object
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        # Bind the socket to the address and port
        s.bind((host, port))
        # Listen for incoming connections
        s.listen()

        # Accept a connection
        conn, addr = s.accept()
        with conn:
            print(f'Connected by {addr}')
            # Receive the serialized data
            data = b''
            while True:
                packet = conn.recv(4096)
                if not packet:
                    break
                data += packet
    data=pickle.loads(data)
    data=decode_data(data)
    for i in range(len(data)):
        if data[i]==0:
            data[i]=None
        elif data[i]==-1:
            data[i]=0
    print(data)
host='localhost'
port=64000
receive(host,port)
```

Explanation:

1. code_size(n)
Purpose:
- Determines the size of the Walsh code matrix based on the number of senders (n).

How it works:
- Computes the smallest power of 2 greater than or equal to the number of senders, N=2^⌈log2(n)⌉.
- This ensures the Walsh code matrix has sufficient orthogonal rows to assign a unique code to each sender.

2. create_code(n)
Purpose:
- Generates the Walsh code matrix of size N×N, where N=2^⌈log2(n)⌉.

How it works:
- Uses binary operations to create a Walsh code matrix. Each element of the matrix is computed as:

$$\text{code}[i][j] = \begin{cases} 1, & \text{if (binary AND of i and j)} = 0 \\ -1, & \text{otherwise} \end{cases}$$

- Converts the result to either 1 or -1 for encoding.

3. decode_data(data)
Purpose:
- Decodes the received composite signal to reconstruct the original data sent by each station.

How it works:
1. Calls create_code(len(data)) to generate the Walsh code matrix.
2. Multiplies the Walsh code matrix by the received composite signal element-wise.
3. Sums the results across rows to reconstruct the original data of each station.
4. Divides the result by the number of stations to normalize the data.

Post-processing:
- Replaces specific values in the decoded data to match the original input:
  - If the decoded value is 000, replaces it with None.
  - If the decoded value is −1-1−1, replaces it with 000.
  -

4. receive(host, port)
Purpose:
- Establishes a TCP server to receive the composite signal from the sender.

How it works:
1. Creates a socket object and binds it to the specified host and port.
2. Listens for incoming client connections.
3. Accepts a connection and retrieves the sender's address.

4. Receives serialized data in chunks of 4096 bytes until no more data is available.
5. Deserializes the received data using pickle.loads.
6. Calls decode_data(data) to reconstruct the original sender data.
7. Prints the decoded data for each sender.

TCP Communication Workflow

Sender-Side:

- Collects data from multiple senders.
- Encodes data using Walsh codes.
- Serializes and sends the composite signal over a TCP connection.

Receiver-Side:

- Listens for incoming connections.
- Receives the serialized signal and deserializes it.
- Decodes the composite signal using the Walsh code matrix.
- Reconstructs and prints the original sender data.

TEST CASES

Sender Output:

```
Enter bit1
Enter bit0
Enter bit1
Enter bit0
Enter bit1
Enter bit0
Enter bit1
Enter bit0
```

Receiver Output:

```
Connected by ('127.0.0.1', 50749)
[1. 0. 1. 0. 1. 0. 1. 0.]
```

COMMENTS:

This assignment has significantly deepened my understanding of CDMA (Code Division Multiple Access) using Walsh codes through both research and practical implementation. By exploring and applying Walsh code-based encoding and decoding techniques, I gained valuable insights into their strengths, such as orthogonality and interference minimization, as well as their limitations, including scalability challenges for large numbers of senders.

Moreover, I learned how the fundamental properties of Walsh codes enable secure and efficient communication in multi-access systems, addressing issues of data overlap and signal interference effectively.

I would like to extend my sincere gratitude to our teacher for guiding us through this learning process. Their support and encouragement have been instrumental in helping me grasp the intricacies of Walsh codes and their applications more thoroughly.