COMPUTER NETWORKS

LAB REPORT

ASSIGNMENT 4

DEBJIT DHAR

BCSE UG 3

ROLL:002210501106

GROUP: A3

SUBMISSION: 18/11/2024

Problem Statement: Implement CDMA with Walsh Code.

Use the same sender-receiver design as previous assignments. 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:

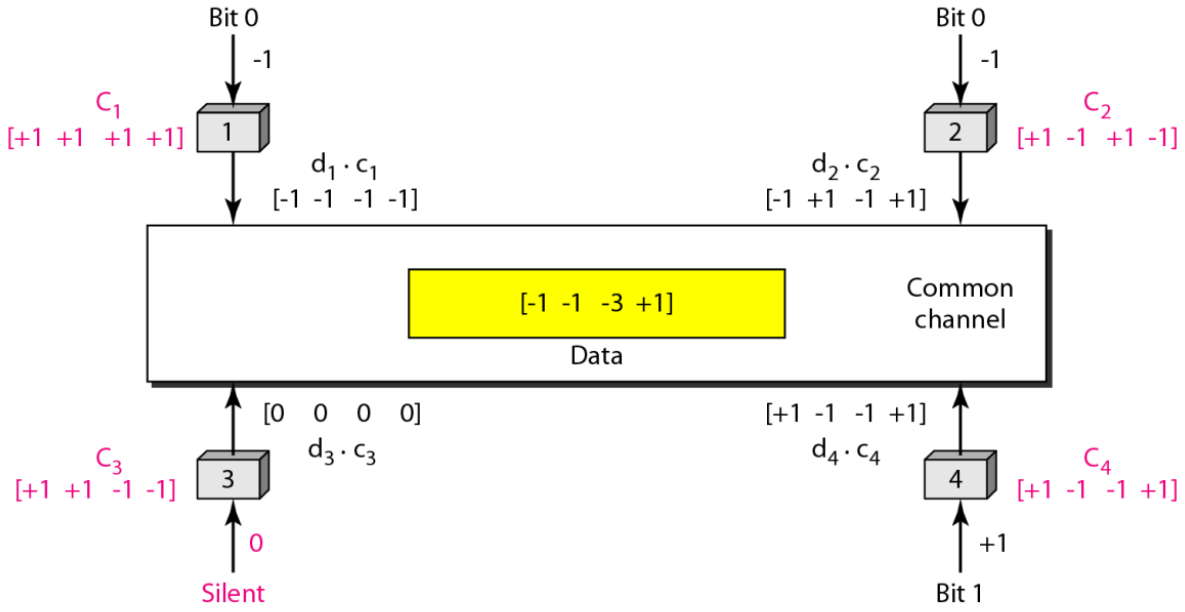
Code Division Multiple Access (CDMA) is a channel access method widely used in wireless communication. It allows multiple users to share the same frequency band simultaneously by assigning unique codes to each user. These codes help distinguish users' data even if their signals overlap in time and frequency.

Walsh Codes in CDMA

- **Walsh codes** are a set of orthogonal binary sequences used in CDMA to separate user signals.
- These codes are generated using a Hadamard matrix, which ensures that the dot product of any two different Walsh codes is zero (orthogonality property).
- Each user in a CDMA system is assigned a unique Walsh code. The codes are used to encode the user's data before transmission.

How Walsh Codes Work:

1. **Encoding:**
    - Each user’s data is multiplied with its assigned Walsh code.
    - This spreads the data across a wider frequency band, ensuring robustness against interference.
2. **Decoding:**
    - At the receiver, the signal is correlated with the assigned Walsh code.
    - Due to orthogonality, only the desired user’s data is retrieved, while signals from other users cancel out.



Explanation of Implementation

1. Encoding (Sender Side):

- **Input Data:**
  - Each sender provides its data as either 1 (bit high), -1 (bit low), or 0 (no bit sent).
- **Walsh Codes:**
  - A set of orthogonal codes (Walsh codes) is generated based on the number of senders. Each sender is assigned a unique Walsh code.
  - These codes ensure that each sender's signal can be uniquely identified, even when combined with others.
- **Encoding Process:**
  - Each sender's data is multiplied with its Walsh code, spreading it across a wider frequency spectrum.

- The encoded signals from all senders are summed together into a single composite signal for transmission.

---

2. Transmission:

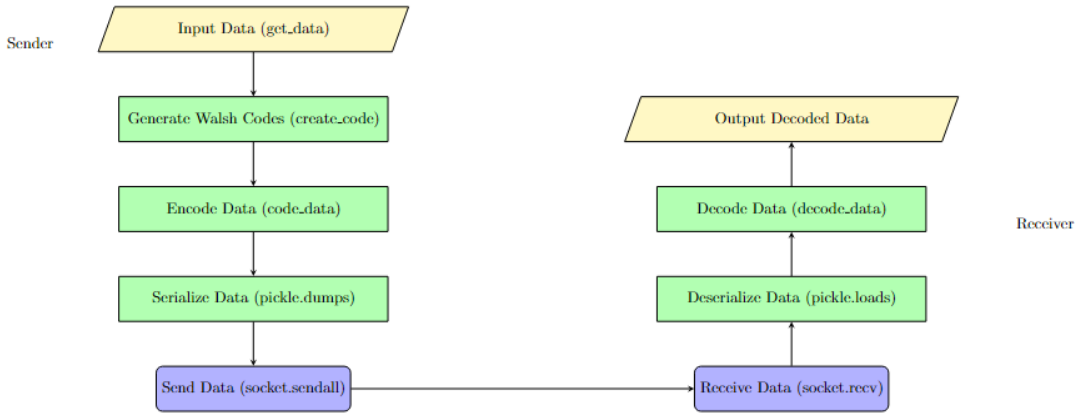
- The encoded composite signal, representing the combined data from all senders, is serialized using **Python's pickle** library.
- The signal is then sent over a TCP network connection to the receiver.

---

3. Decoding (Receiver Side):

- **Receiving the Signal:**
  - The receiver collects the transmitted composite signal and deserializes it to retrieve the encoded data.
- **Regenerating Walsh Codes:**
  - The receiver regenerates the same set of Walsh codes that were used for encoding at the sender's end.
- **Decoding Process:**
  - The composite signal is multiplied with each Walsh code and summed.
  - Using the orthogonality property of Walsh codes, the receiver isolates each sender's original data from the combined signal.

Structural Diagram



Implementation

Implementable code at: https://github.com/Debjit-Dhar/Networks

The sender and receiver are same as the previous design of Assignment 3 except with the presence of Welsh Code CDMA instead of p persistent CSMA.

Sender:

```
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)

data=np.array([-1,-1,0,1])
code_data(data)

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=63000
num_senders=8

send(num_senders,host,port)
```

Receiver:

```
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

data=np.array([-1, -1, -3,  1])
decode_data(data)

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=63000

receive(host,port)
```

Input-Output Format

Sender:

1. **Input (User Data):**
    - The sender collects data for multiple users through console input.
    - Each user inputs a bit value:
        - 1: Represents a high bit.
        - -1: Represents a low bit.
        - 0: Represents no bit or silence.
    - If no input is given for a user, it defaults to 0.
2. **Output (Encoded Signal Sent to Receiver):**

- The sender encodes the input data using Walsh codes and sends the composite signal.
- This is not displayed but transmitted as a serialized signal.

Receiver:

1. **Input (Composite Signal):**
    - The receiver gets the composite encoded signal over a TCP connection.
2. **Output (Decoded Data):**
    - The receiver decodes the composite signal to retrieve the original data for each user.
    - Decoded data is displayed on the console, with the following rules:
        - 1: High bit.
        - 0: Low bit (originally -1 in input).
        - None: Represents no bit (originally 0 in input).

OBSERVATIONS

Sender:

```
Enter bit 1
Enter bit
Enter bit 0
Enter bit 1
Enter bit 1
Enter bit 0
Enter bit
Enter bit 1
```

Walsh Code:

```
array([[ 1,  1,  1,  1],
       [ 1, -1,  1, -1],
       [ 1,  1, -1, -1],
       [ 1, -1, -1,  1]])
```

Receiver:

```
Connected by ('127.0.0.1', 50633)
[ 1. nan  0.  1.  1.  0. nan  1.]
```

COMMENTS

This assignment has greatly enhanced my understanding of code division multiple access with walsh code. I have also learnt how to implement a real-time simulation for the same. Furthermore, this implementation also highlighted the advantages of cdma and how it is implemented.

I would like to express my heartfelt thanks to my teachers Dr Sarbani Roy and Dr Nandini Mukherjee for their guidance and support throughout this journey. Their encouragement has played a key role in helping me better comprehend these concepts.