Computer Networks Lab Report — Assignment 4

TITLE

Name — Nikhil Badyal

Roll — 001810501069

Class — BCSE 3’d year Group — A2 Assignment Number — 4

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.

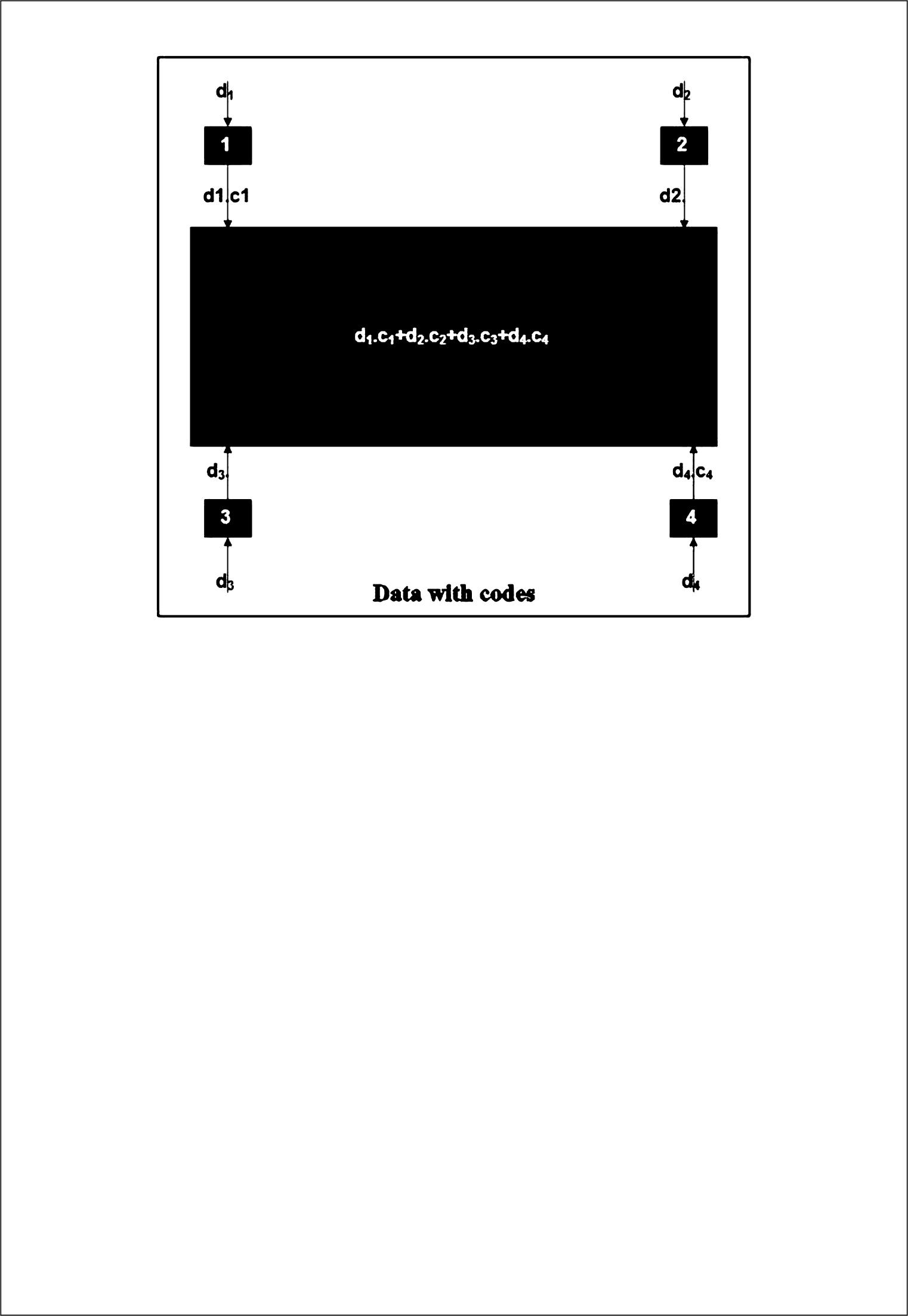
# DESIGN

I have implemented the assignment in total of 3 main files.

* channel.java (Program for channel)
* sender.java (Program for sender )
* **receiver.java** (Program for receiver)
* **CDMA.java** (for CDMA related functions)

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

1. channel.java — It doesn't do much work other than transferring data from one place to other.
2. Sender.java – In nutshell its task is to send data to channel in order to make it to receiver.
3. It first asks for the number of stations.
4. Then it runs a loop to prepare the data in order to send them.
5. In every loop. It reads some bit from the file and add them to an array. It does same for every station in the system.
6. It then takes that preapared array and forward It to CDMA function which prepares walsh table and do the rest of procedure.
7. This loop goes on till there is no more data left to send.
8. Receiver.java – In nutshell its task is to receive data from the channel in order to make it to receiver.
9. It first asks for the number of receivers in the system.
10. It has a while loop in its end which keeps on receiving data until it receives an EOT frame which ask it to stop waiting and go to sleep.
11. In every run of the loop it accepts a packet from the channel and pass it to CMDA function which is responsible for the interpretation of the data.
12. **CDMA.java—** It is the main function which serves the main purpose of assignment. It has following functions in it.
    * + 1. **SETUP** – It task it to prepare the data so that it can be sent over the channel. It build walsh table and multiply the bits with the row of the table.
        2. **RECEIVER** – It receives the data and extract the original data sent by the sender.



c2

Station 3

Station 2

Station 1

Channel

Sender

CDMA Sender Setup

Channel

Receiver

CDMA Message Retriever

IMPLEMENTATION

**Sender.java**

*import* java.io.\*;  
  
*import* java.net.Socket;  
*import* java.util.ArrayList;  
*import* java.util.Scanner;  
  
*import static* java.lang.System.exit;  
  
  
*public class* Sender {  
  
 *public static void* main(String[] args) {  
  
 Scanner sc = *new* Scanner(System.in);  
 System.out.println("Enter number of stations");  
 *int* numberOfStations = sc.nextInt();  
  
 ArrayList<String> fileContent = *new* ArrayList<>(numberOfStations);  
  
 *int* maxDataLength = Integer.MIN\_VALUE;  
  
 *//reading all data from all files and creating a 2D array wrt to their number  
 for*(*int* i=1;i<=numberOfStations;i++){  
 *try* {  
 File file = *new* File("Sender" + i +".txt");  
*// File file = new File("E:\\BCSE 3RD YEAR\\SARBANI MAM\\LAB\\Assignment4\\code\\src\\Data\\Sender" + i +".txt");* Scanner myReader = *new* Scanner(file);  
 StringBuilder s = *new* StringBuilder();  
 *while* (myReader.hasNextLine()) {  
 s.append(myReader.nextLine());  
  
 }  
 *if*(s.length()>maxDataLength){  
 maxDataLength = s.length();  
 }  
 fileContent.add(s.toString());  
 System.out.println(fileContent.get(i-1));  
  
  
 myReader.close();  
 } *catch* (FileNotFoundException e) {  
 System.out.println("An error occurred.");  
 e.printStackTrace();  
 exit(1);  
 }  
 }  
  
 System.out.println("Content is :");  
 *for*(String x : fileContent){  
 System.out.println(x);  
 }  
 System.out.println("End");  
  
 *int*[] dataInEachStation = *new int*[numberOfStations];  
 *int* port = 5000;  
 *try*{  
 Socket s = *new* Socket("localhost", port);  
 ObjectOutputStream objectOutputStream = *new* ObjectOutputStream(s.getOutputStream());  
 ObjectInputStream objectInputStream = *new* ObjectInputStream(s.getInputStream());  
  
 *for*(*int* i=0;i<maxDataLength;i++){  
 *//preparing data frame i.e. extracting data from all sender and creating a final data  
 for*(*int* j=0;j<numberOfStations;j++){  
 *if*(i >= fileContent.get(j).length()){  
 dataInEachStation[j] = 0;  
 }*else*{  
 *if*(Integer.parseInt(String.valueOf(fileContent.get(j).charAt(i))) == 1){  
 dataInEachStation[j] = Integer.parseInt(String.valueOf(fileContent.get(j).charAt(i)));  
 }*else*{  
 dataInEachStation[j] = -1;  
 }  
 }  
 System.out.print(dataInEachStation[j]);  
 }  
  
  
 CDMA cdma = *new* CDMA();  
  
 String data = cdma.setUp(dataInEachStation,numberOfStations); *// this will return the data which is to be transferred* System.out.println("String data is "+data);  
  
 System.out.println("Frame before error insertion : " + i);  
  
  
 *if*(i == maxDataLength-1){  
 data = "e" + data; *//e for last frame* }*else*{  
 data = "n" + data; *//n for normal frame* }  
 Channel.send(data, objectOutputStream); *//sending data  
  
 while*(Channel.getAck(objectInputStream, s)){  
 Channel.send(data, objectOutputStream); *//receiving acknowledgement* }  
  
 Thread.sleep(1000);  
 System.out.println();  
 }  
  
 }*catch*(Exception e){  
 System.out.println("exception");  
 e.printStackTrace();  
 }  
 }  
}

Receiver

*import* java.io.IOException;  
*import* java.io.ObjectInputStream;  
*import* java.io.ObjectOutputStream;  
*import* java.net.\*;  
  
*import* java.util.Scanner;  
  
*public class* Receiver {  
  
 *public static void* sendAck(String ack, ObjectOutputStream objectOutputStream){  
 *try* {  
 objectOutputStream.writeObject(ack);  
 } *catch* (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 *public static void* main(String[] args) {  
 *int* port = (5000);  
 *try*{  
 *int* noOfRecievers;  
 Scanner scanner = *new* Scanner(System.in);  
 System.out.println("Enter numbeer of reccievers : ");  
 noOfRecievers = scanner.nextInt();  
  
 ServerSocket serverSocket = *new* ServerSocket(port);  
 System.out.println("Waiting for connection");  
 Socket s2 = serverSocket.accept();  
 System.out.println("Connection Established");  
 ObjectInputStream objectInputStream = *new* ObjectInputStream(s2.getInputStream());  
 ObjectOutputStream objectOutputStream = *new* ObjectOutputStream(s2.getOutputStream());  
  
 *boolean* goOn = *true*;  
  
  
 String[] allReceiverData = *new* String[noOfRecievers];  
 CDMA cdma = *new* CDMA();  
 cdma.setUp(noOfRecievers);  
 *while*(goOn){  
 String frame = (String)objectInputStream.readObject();  
 System.out.println(frame);  
  
 String data = frame.substring(1);  
  
 cdma.receiver(noOfRecievers, data, allReceiverData);  
 System.out.println("Writing next");  
  
 sendAck("next", objectOutputStream);*//Sending ack  
  
 if*(frame.charAt(0) == 'e'){  
 System.out.println("hello");  
 goOn = *false*;  
 }  
 }  
 System.out.println("RECEIVED DATA IS : ");  
 *for*(*int* i=0;i<allReceiverData.length;i++){  
 System.out.println("Sender " + (i+1) + " sends : " + allReceiverData[i]);  
 }  
 }*catch*(Exception e){  
 System.out.println("Exception");  
 e.printStackTrace();  
 }  
  
 }  
}

CDMA

*import* java.util.Arrays;  
  
*public class* CDMA {  
 *int*[][] wtable;  
 *int*[][] copy;  
 *int*[] channel\_seq;  
  
 *public* String setUp(*int*[] data, *int* num\_stations) {  
 StringBuilder dataStr = *new* StringBuilder();  
 wtable = *new int*[num\_stations][num\_stations];  
 copy = *new int*[num\_stations][num\_stations];  
  
 *//creating walsh table* buildWalshTable(num\_stations, 0, num\_stations - 1, 0,  
 num\_stations - 1, *false*);  
  
 showWalshTable(num\_stations);  
  
 *//create copy of walsh table  
 for* (*int* i = 0; i < num\_stations; i++) {  
  
 *for* (*int* j = 0; j < num\_stations; j++) {  
 copy[i][j] = wtable[i][j];  
 wtable[i][j] \*= data[i];  
 }  
 }  
  
 channel\_seq = *new int*[num\_stations];  
  
 *for* (*int* i = 0; i < num\_stations; i++) {  
  
 *for* (*int* j = 0; j < num\_stations; j++) {  
 *// Adding all sequences to get channel sequence* channel\_seq[i] += wtable[j][i];  
 }  
 }  
 System.out.println("Data is " + Arrays.toString(channel\_seq));  
 *for* (*int* i : channel\_seq) {  
 dataStr.append(i);  
 }  
 *return* dataStr.toString();  
  
 }  
  
 *public void* buildWalshTable(*int* len, *int* i1, *int* i2, *int* j1,  
 *int* j2, *boolean* isBar) {  
  
 *if* (len == 2) {  
  
 *if* (!isBar) {  
  
 wtable[i1][j1] = 1;  
 wtable[i1][j2] = 1;  
 wtable[i2][j1] = 1;  
 wtable[i2][j2] = -1;  
 } *else* {  
  
 wtable[i1][j1] = -1;  
 wtable[i1][j2] = -1;  
 wtable[i2][j1] = -1;  
 wtable[i2][j2] = +1;  
 }  
  
 *return*;  
 }  
  
 *int* midi = (i1 + i2) / 2;  
 *int* midj = (j1 + j2) / 2;  
  
 buildWalshTable(len / 2, i1, midi, j1, midj, isBar);  
 buildWalshTable(len / 2, i1, midi, midj + 1, j2, isBar);  
 buildWalshTable(len / 2, midi + 1, i2, j1, midj, isBar);  
 buildWalshTable(len / 2, midi + 1, i2, midj + 1, j2, !isBar);  
  
 }  
  
 *public void* showWalshTable(*int* num\_stations) {  
  
 System.out.print("\n");  
  
 *for* (*int* i = 0; i < num\_stations; i++) {  
 *for* (*int* j = 0; j < num\_stations; j++) {  
 System.out.print(wtable[i][j] + " ");  
 }  
 System.out.print("\n");  
 }  
 System.out.println("-------------------------");  
 System.out.print("\n");  
 }  
  
 *public void* receiver(*int* noOfStations, String dataStr, String[] allRecieverData) {  
  
  
 *for*(*int* x=0;x<noOfStations;x++){  
 *int* product = 0;  
 *int* cnt = 0;  
  
 *if*(allRecieverData[x] == *null*){  
 allRecieverData[x] = "";  
 }  
  
 *for* (*int* j = 0; j < dataStr.length(); ++j) {  
 *if* (dataStr.charAt(j) != '-') {  
 cnt++;  
 }  
 }  
 System.out.println("count is " + cnt);  
 System.out.println("Data is " + dataStr);  
 *int*[] channel\_seq = *new int*[cnt];  
 *int* j = 0;  
 *int* i = 0;  
  
 *//converting string data to int  
 while* (j < cnt) {  
 *if* (dataStr.charAt(i) == '-') {  
 String str = dataStr.substring(i, i + 2);  
 System.out.println("TO convert string is" + str);  
 i += 2;  
 channel\_seq[j] = Integer.parseInt(str);  
 } *else* {  
 System.out.println(dataStr.charAt(i));  
 channel\_seq[j] = Integer.parseInt(dataStr.charAt(i) + "");  
 i++;  
 }  
 j++;  
 }  
  
 System.out.println(Arrays.toString(channel\_seq));  
 System.out.println();  
  
 *//  
 for* (*int* k = 0; k < noOfStations; ++k) {  
 product += wtable[x][k] \* channel\_seq[k];  
 }  
 *int* data = product / noOfStations;  
 System.out.println("Dat is "+data);  
 *if* (data == 1) {  
 allRecieverData[x] = allRecieverData[x] + data;  
 System.out.println("The data received is from station" + (x + 1) + " is " + data);  
 } *else if* (data == -1) {  
 allRecieverData[x] += 0;  
 System.out.println("The data received is from station" + (x + 1) + " is " + (0));  
 } *else* {  
 System.out.println("Channel didn't sent any data.");  
 }  
 }  
  
 }  
  
 *public void* setUp(*int* num\_stations) {  
 wtable = *new int*[num\_stations][num\_stations];  
 copy = *new int*[num\_stations][num\_stations];  
 buildWalshTable(num\_stations, 0, num\_stations - 1, 0,  
 num\_stations - 1, *false*);  
  
 }  
}

Channel

*import* java.io.IOException;  
*import* java.io.ObjectInputStream;  
*import* java.io.ObjectOutputStream;  
*import* java.net.Socket;  
*import* java.net.SocketTimeoutException;  
  
*public class* Channel {  
  
 *public static void* send(String Frame, ObjectOutputStream objectOutputStream) {  
  
  
 *try*{  
 System.*out*.println("Sending frame from channel");  
 objectOutputStream.writeObject((String)Frame);  
 System.*out*.println(Frame);  
  
 }*catch*(IOException e){  
 System.*out*.println("IOException " + e);  
 }  
 }  
  
 *public static void* sendAck(String ack, ObjectOutputStream objectOutputStream){  
 *try* {  
 objectOutputStream.writeObject(ack);  
 } *catch* (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 *public static boolean* getAck(ObjectInputStream objectInputStream, Socket s) *throws* IOException, ClassNotFoundException {  
 s.setSoTimeout(1000);  
  
 *try*{  
 System.out.println("Waiting for acknowledgement");  
 String ack = (String)objectInputStream.readObject();  
 *return* ack.equals("resend");  
 }*catch*(SocketTimeoutException socketTimeoutException){  
 System.out.println("SomeThing bad happened, data lost (no ack, Timeout)\n\n\n");  
 *return true*;  
 }  
 }  
  
}

# RESULTS & ANALYSIS

* Unlike other , in CDMA all stations can transmit data simultaneously, there is no timesharing.
* CDMA allows each station to transmit over the entire frequency spectrum all the time.
* Multiple simultaneous transmissions are separated using coding theory.
* In CDMA each user is given a unique code sequence.
* The basic idea of CDMA is explained below:

1. Let us assume that we have four stations 1, 2, 3 and 4 that are connected to same channel. The data from station 1 are dl, from station 2 are d2 and so on.
2. The code assigned to first station is Co, to the second is Cz and so on.
3. These assigned codes have two properties:
   1. If we multiply each code by another, we get 0.
   2. If we multiply each code by itself, we get 4. (No. of stations).
4. When these four stations are sending data on the same channel, station 1 multiplies its data by its code *i.e.* di.ca), station 2 multiplies its data by its code *i.e.* d2 .C2 and so on.
5. The data that go on channel are the sum of all these terms as shown in Fig.
6. Any station that wants to receive data from one of the other three stations multiplies the data on channel by the code of the sender. For example, suppose station 1 and 2 are talking to each other. Station 2 wants to hear what station 1 is saying. It multiples the data on the channel by CI (the code of station 1).
7. Because (Co. Co) is 4, but (Cz. Co), (C/. Co), and (C4. Co) are all zeroes, station 2 divides the result by 4 to get the data from station 1.

data = (di . C + d2 • Cz+ da. C3\* d . C<) • C

= d|. Ct . Ct+ d . Ct. Ct+ d3 . !3• Ct + d4 • !4• Ct= 4 x d/

* The code assigned to each station is a sequence of numbers called chips. These chips are called orthogonal sequences. This sequence has following properties:

1. Each sequence is made of N elements, where N is the number of stations as shown in fig.



C1

1. If we multiple a sequence by a number, every element in the sequence is multiplied by that element. This is called multiplication of a sequence by a scalar.

For example:

[+1 +1-1 -1] = [+2 +2 -2 -2]

1. If we multiply two equal sequences, element by element and add the results, we get N, where N is the number of elements in each sequence. This is called inner product of two equal sequences. For example:

[+1 +1-1 -1] . [+1 +1-1-1] = 1+ 1+ 1+ 1 = 4

1. If we multiply two different sequences, element by element and add the results, we get 0. This is called inner product of two different sequences. For example:

[+1 +1-1-1]. [+1 +1 +1 +1] = 1+1-1 -1= 0

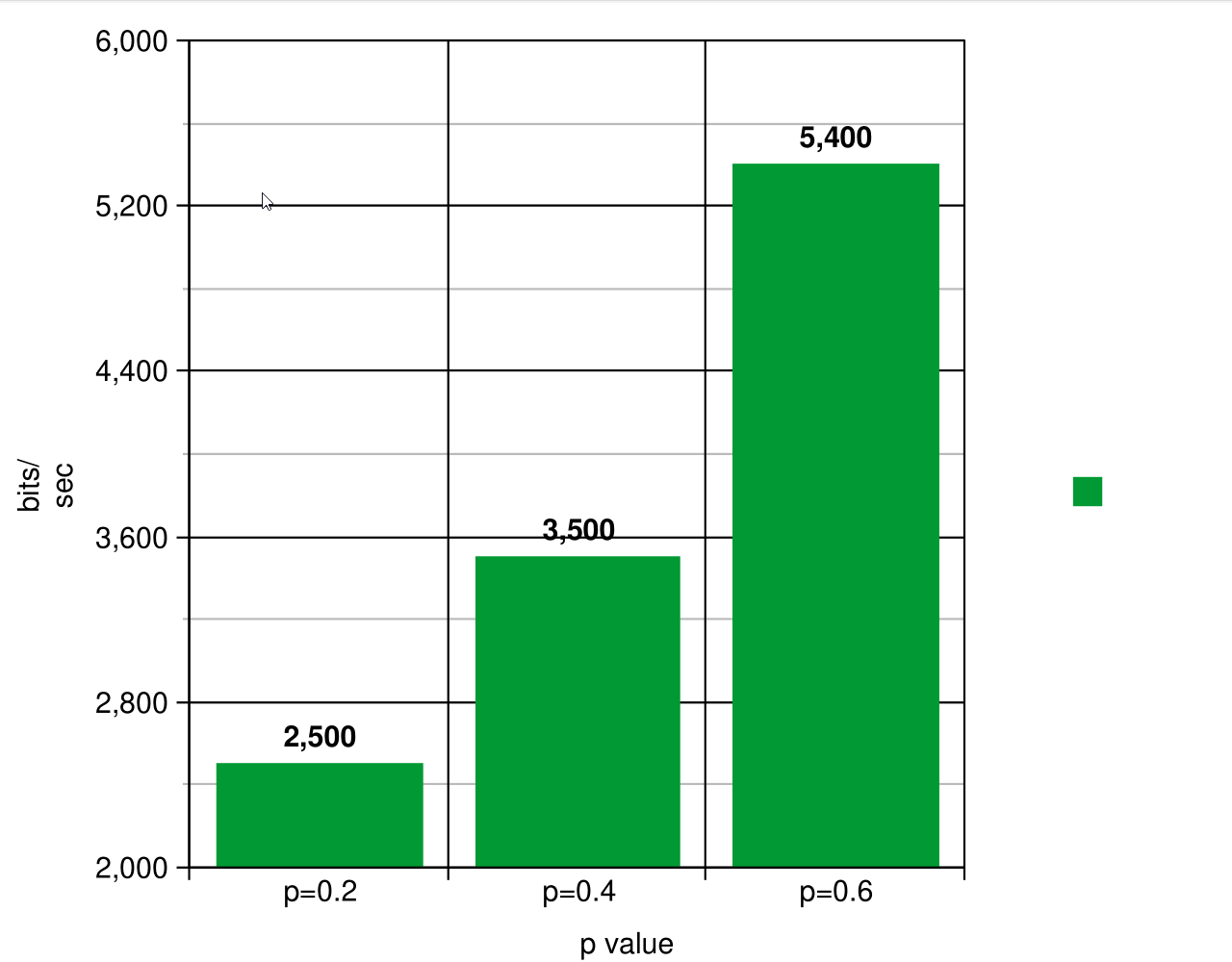
1. Adding two sequences means adding the corresponding elements. The result is another sequence. For example:

[+1 +1-1 -1] + [+1 +1+1 +1] = [+2 +2 0 0]

* The data representation and encoding is done by different stations in following manner:

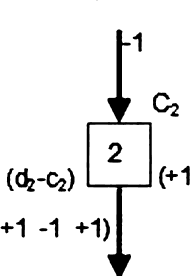
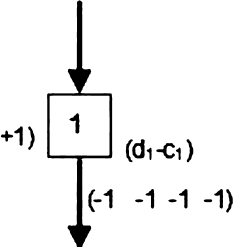
1. If a station needs to send a 0 bit, it encodes it as -1.
2. If it needs to send a 1 bit, it encodes it as + 1.
3. When station is idle, it sends no signal, which is interpreted as a 0.

* For example, If station 1 and station 2 are sending a 0 bit, station 3 is silent and station 4 is sending a 1 bit; the data at sender site are represented as -1, - 1,0 and +1 respectively.
* Each station multiplies the corresponding number by its chip, which is unique for each station.
* Each station send this sequence to the channel ; The sequence of channel is the sum of all four sequence as shown in fig.

****

4 6 8

|  |  |  |
| --- | --- | --- |
| Try | No. of Station (total no. of frames) | Total Time Taken |
| 1 | 5(15) | 21 Seconds |
| 2 | 4(15) | 15 Seconds |
| 3 | 4(10) | 12 Seconds |
| 4 | 3(15) | 9 Seconds |



Sharing channel in CDMA

Bit 0

Bit 0

C1

(+1 \*1 \*1

-1 +1 -1)

(-1

(0 0 0 0)

(+1

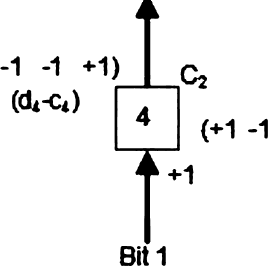
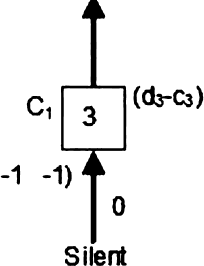
-1 •1)

( 1 +1

Chame

oaa

(.1 .1 .3 +1)

If station 3, which was silent, is listening to station 2. Station 3 multiplies the total data on the channel by the code for station 2, which is [+ 1 -1+1 -1], to get

[-1-1 -3+1] . [+1 -1+1 -1]= -4/4 = -1 --> bit 0

COMMENTS

This assignment has helped me to understand the how Walsh Table is built for a given number of stations, and how CDMA channelization protocol encodes and decodes the data bits sent by all stations simultaneously.