

DATA COMMUNICATION  
AND  
COMPUTER NETWORKS  
LAB RECORD

NAME: PRAKRATI SINGH

SAP: 500082638

ROLL: R214220840

BATCH: B6(HONS)

# INDEX

Sno.	Assessment name	Teacher's sign
1.	Lab Task 1	
2.	Lab Task 2	
3.	Lab Task 3	
4.	Lab Task 4	
5.	Lab Task 5	
6.	Lab Task 6	
7.	Lab Task 7	

# LAB TASK 1

## Signal rate calculation

Bandwidth - Bandwidth is characterized as the capability of the information that will be moved in a particular timeframe. It is the information conveying limit of the organization or transmission medium. In basic words, the greatest measure of information can be moved each second on a connection. It is by and large estimated in bits per second(bps), Megabits per second (Mbps) or Giga bits per second (Gbps)

Signal rate - The signal rate of an advanced sign is characterized as the proportional of the piece width (1/bit width). The signaling rate is utilized to decide the recurrence scope of electrical confinement.

CODE:

---

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
int main()
{
    float c = 0.5;
    float N = 1000, temp;
    float ratio, SR;
    for (int i = 0; i < 10; i++)
    {
        ratio = rand() % (8 + 1 - 1) + 1;
        temp = (float)(1 / ratio);
        SR = (c * temp * N);
        printf("for value r= %f\n", ratio);
        printf("Signal rate: %f bauds\n", SR);
    }
}
int main()
{
    float c;
    float N;
    int r;
    float S;
    printf("Enter case factor: \n");
    scanf("%f", &c);
    printf("Enter data rate: \n");
    scanf("%f", &N);
    printf("Enter signal element: \n");
    scanf("%d", &r);
```

```
PS C:\Users\91766\Desktop\ \500082638-PRAKRATI SINGH-DCCN\Lab> gcc Lab1.c
PS C:\Users\91766\Desktop\ \500082638-PRAKRATI SINGH-DCCN\Lab> ./a
for value r= 2.000000
Signal rate: 250.000000 bauds
for value r= 4.000000
Signal rate: 125.000000 bauds
for value r= 7.000000
Signal rate: 71.428574 bauds
for value r= 5.000000
Signal rate: 100.000000 bauds
for value r= 2.000000
Signal rate: 250.000000 bauds
for value r= 5.000000
Signal rate: 100.000000 bauds
for value r= 7.000000
Signal rate: 71.428574 bauds
for value r= 7.000000
Signal rate: 71.428574 bauds
for value r= 3.000000
Signal rate: 166.666672 bauds
for value r= 1.000000
Signal rate: 500.000000 bauds
```

### Calculating channel capacity

Data rate - Data Rate is characterized as how much data communicated during a predefined time-frame over an organization. It is the speed at which data is moved starting with one gadget then onto the next or between a fringe gadget and the PC. It is for the most part estimated in Megabits per second(Mbps) or Megabytes per second(Mbps)

CODE:

```
#include <stdio.h>
#include<math.h>
#include<stdlib.h>
int main()
{
    float c = 0.5;
    float N = 1000, temp;
    float ratio, SR;
    for (int i = 0; i < 10; i++)
    {
        ratio = rand() % (8 + 1 - 1) + 1;
        temp = (float)(1 / ratio);
        SR = (c * temp * N);
        printf("for value r= %f\n", ratio);
        printf("Signal rate: %f bauds\n", SR);
        int BW;
        double Lvl;
        float r;
        printf("Enter Bandwidth: \n");
        scanf("%d", &BW);
        printf("Enter number of Levels: \n");
        scanf("%lf", &Lvl);
        r = (2 * BW * (log2(Lvl)));
        printf("Channel Capacity: %f bps\n", r);
    }
}
```

```
PS C:\Users\91766\Desktop\ \500082638-PRAKRATI SINGH-DCCN\Lab> gcc Lab1.c
PS C:\Users\91766\Desktop\ \500082638-PRAKRATI SINGH-DCCN\Lab> ./a
for value r= 2.000000
Signal rate: 250.000000 bauds
Enter Bandwidth:
300
Enter number of Levels:
4
Channel Capacity: 1200.000000 bps
for value r= 7.000000
Signal rate: 71.428574 bauds
```

## LAB TASK 2

## BIT STUFFING AND DESTUFFING

Bit Stuffing - It is the process of adding an extra bit 0 in the bit stream whenever there are five consecutive 1's in it.

Destuffing - The process of removing the 0 after five consecutive 1's in the received signal is called destuffing.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// Bit Stuffing - It is the process of adding an extra 0 after five 1's so that
the receiver does not mistake the pattern '11111' for a flag.
int main()
{
    int num, count = 0;
    printf("Enter length of bit stream: ");
    // bit stream: all data shown in the form of binary
    scanf("%d", &num);
    int k = num + (num / 5);
    int *arr = (int *)malloc(k * sizeof(int));
    printf("Enter the bit stream: \n");
    // Taking the bit stream as input
    for (int i = 0; i < num; i++)
    {
        scanf("%d", &arr[i]);
    }
    for (int i = 0; i < num; i++)
    {
        if (arr[i] == 1)
        {
            count++;
            // counter variable to count the 1's
        }
        else
        {
            count = 0;
        }
        if (count == 5)
        // checking if count is equal to 5
        {
```

```
num = num + 1;
// increasing the size of array by 1 to insert the new bit int j = num - 1;
while (j > i + 1)
// loop from the end of the array till the fifth 1.
{
    arr[j] = arr[j - 1];
    j--;
}
arr[j] = 0;
// inserting 0 in the place after five 1's
i++;
count = 0;
}
}
printf("Array after stuffing: \n");
for (int i = 0; i < num; i++)
{
    printf("%d", arr[i]);
    // printing the stuffed array after bit stuffing
}
// Destuffing - The process of removing the 0 after five consecutive 1's in the
received signal is called destuffing.
for (int i = 0; i < num; i++)
{
    if (arr[i] == 1)
    {
        count++;
    }
    else
    {
        count = 0;
    }
    if (count == 5)
    {
        num = num - 1;
        // decreasing the array size int j = i+1;
        while (j < num)
        {
            arr[j] = arr[j + 1];
            j++;
        }
        count = 0;
    }
}
```

```
}  
printf("\nArray after de-stuffing: \n");  
for (int i = 0; i < num; i++)  
{  
    printf("%d", arr[i]);  
}  
}
```

OUTPUT:

```
PS C:\Users\91766\Desktop\ \500082638-PRAKRATI SINGH-DCCN\Lab> cd "c:\Users\91766\Desktop\ \500082638-PRAKRATI SINGH-DCCN\Lab\LAB WORK" & .\bitstuffing } ; if ($?) { .\bitstuffing }  
Enter length of bit stream: 10  
Enter the bit stream:  
1  
1  
1  
1  
1  
1  
1  
1  
1  
1  
1  
Array after stuffing:  
111110111110  
Array after de-stuffing:  
1111111111  
PS C:\Users\91766\Desktop\ \500082638-PRAKRATI SINGH-DCCN\Lab\LAB WORK> |
```



## LAB TASK 3

To check if data frame contains noise.

Noise - Noise in a correspondence framework is fundamentally bothersome or unwanted signals that get arbitrarily added to the real data conveying signal. Resultantly, causes aggravations in the first sign being communicated from a finish to another.

The presence of noise in the framework causes impedance in the sign being sent and this at last causes mistakes in the correspondence framework.

CODE:

```
#include <stdio.h>
#include<stdlib.h>
#include<time.h>
int Compare(int a1[], int a2, int n)
{
    int f = 1, i = 0;
    for (; i < n; i++)
    {
        if (a1[i] != a2)
        {
            f = 0;
            break;
        }
    }
    return f;
    if (f == 1)
    {
        printf("Data frame contains noise\n");
    }
    else
    {
        printf("Data frame does not contain noise\n");
    }
}

void display(int arr[], int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%d", arr[i]);
    }
}

int main()
{
    int n, noise;
    printf("Enter size of the data frame: \n");
    scanf("%d", &n);
```

---

```
float Tp = 0, Tb = 0, Ftime = 0, R;
int *df = (int *)malloc(n * sizeof(int));
int *result = (int *)malloc(n * sizeof(int));
printf("Enter the noise: ");
scanf("%d", &noise);
printf("Enter data frame: \n");
for (int i = 0; i < n; i++)
{
    scanf("%d", &df[i]);
}
// We will add the noise and dataframe arrays.
int rem = 0;
for (int i = n - 1; i >= 0; i--)
{
    if (df[i] + noise + rem == 2)
    {
        noise = 1;
        rem = 0;
    }
    else
    {
        noise = 0;
        rem = 0;
    }
}
printf("received df : \n");
display(df, n);
Compare(df, noise, n);
noise = 1;
return 0;
}
```

OUTPUT:

```
PS C:\Users\91766\Desktop\ \500082638-PRAKRATI SINGH-DCCN\Lab> gcc noise.c
PS C:\Users\91766\Desktop\ \500082638-PRAKRATI SINGH-DCCN\Lab> ./a
Enter size of the data frame:
5
Enter the noise: 1
Enter data frame:
1 0 0 1 0
received df :
10010
Data frame contains noise
```

## LAB TASK 4 - PURE ALOHA AND SLOTTED ALOHA

CODE:

```
#include <stdio.h> #include<stdlib.h> #include<time.h>
int Compare(int a1[], int a2[], int n)
{
    int f = 1, i = 0;
    for (; i < n; i++)
    {
        if (a1[i] != a2[i])
        {
            f = 0;
            break;
        }
    }
    return f;
    if (f == 1)
    {
        printf("Data frame contains noise\n");
    }
    else
    {
        printf("Data frame does not contain noise\n");
    }
}

void display(int arr[], int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%d", arr[i]);
    }
}

int main()
{
    int n, noise;
    printf("Enter size of the data frame: \n");
    scanf("%d", &n);
    float Tp = 0, Tb = 0, Ftime = 0, R;
    int *df = (int *)malloc(n * sizeof(int));
    int *result = (int *)malloc(n * sizeof(int));
    printf("Enter the Propagation time : ");
    scanf("%f", &Tp);
    printf("Enter the noise: ");
```

```

scanf("%d", &noise);
Ftime = Ftime + (2 * Tp);
printf("\nTime Taken: %.3f", Ftime);
printf("Enter data frame: \n");
for (int i = 0; i < n; i++)
{
    scanf("%d", &df[i]);
}
// We will add the noise and dataframe arrays. int rem=0;
for (int i = n - 1; i >= 0; i--)
{
    if (df[i] + noise + rem == 2)
    {
        noise = 1;
        rem = 0;
    }
    else
    {
        noise = 0;
        rem = 0;
    }
}
printf("received df : \n");
display(df, n);
Compare(df, noise, n);
srand(time(0));
int temp = (rand() % ((n - 1) - 0 + 1));
noise = 1;
display(noise, n);
Compare(df, noise, n);
Tb = R * Tp;
printf("Backoff time : %.3f", Tb);
return 0;
}

```

Pure Aloha - In Pure Aloha, Stations send at whatever point information is accessible at erratic times and it are obliterated to Collide outlines. While In Slotted aloha, A station is expected to hang tight for the start of the following opening to communicate. weak period is divided instead of pure Aloha.

Slotted Aloha - There is a high chance of edge hitting in pure aloha, so slotted aloha is intended to conquer it. Not at all like pure aloha, slotted aloha doesn't permit the transmission of information at whatever point the station needs to send it. In slotted Aloha, the common channel is isolated into a decent time span called openings. So that, to send a casing to a common channel, the edge must be sent toward the start of the space, and just a single casing is permitted to be shipped off each opening. In the event that the station is neglected to send the information, it needs to hold on until the following opening.

## OUTPUT:

```
PS C:\Users\91766\Desktop\ \500082638-PRAKRATI SINGH-DCCN\Lab> gcc aloha.c
PS C:\Users\91766\Desktop\ \500082638-PRAKRATI SINGH-DCCN\Lab> ./a
Enter the size of the data frame : 5
Enter the data frame :
1
0
0
1
1
Enter the Propagation time : 4

0
1
1
Time Taken: 10.000
Data frame contains noise
Backoff time : 10.000
Enter the noise: 0

Original data frame : 1 0 0 1 1
Noise : 0
Recieved Data frame : 1 0 0 1 1
Time Taken: 20.000
Data frame has no noise
```

#CODE - CSMA / CD

#include &lt;stdlib.h&gt; #include &lt;stdio.h&gt; #include &lt;math.h&gt;

void display(int arr[], int n)

```
{
    for (int i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
}
```

int noisefunc(int arr1[], int arr2[], int n)

```
{
    int flag = 1, i = 0;
    for (; i < n; i++)
    {
        if (arr1[i] != arr2[i])
        {
            flag = 0;
            break;
        }
    }
    return flag;
}
```

int main()

```
{
    int k = 0, kmax = 10;
    int n;
    int d = 0;
    int noise = 1;
    float slot = 0.0000015;
    float Tp = 200.0, Tb;
    float Ftime;
    printf("Enter size of data frame : ");
    scanf("%d", &n);
    int *df = (int *)malloc(n * sizeof(int));
    int *recdf = (int *)malloc(n * sizeof(int));
    printf("Enter the data frame : \n");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &df[i]);
    }
    while (d < 1000)
    {
        while (k < kmax)
```

```
{

    // printf("\nOriginal data frame : ");
    // display(df, n);

    for (int i = 0; i < n; i++)
    {
        recdf[i] = df[i];
    }
    int rem = 0;
    if (recdf[n - 1] == 1 && noise == 1)
    {
        recdf[n - 1] = 0;
        rem = 1;
        recdf[n - 2] += rem;
    }
    else
    {
        recdf[n - 1] = recdf[n - 1] + noise;
    }

    // printf("\nRecieved Data frame : ");
    // display(recdf,n);
    Ftime = Ftime + (2 * Tp) + Tb;

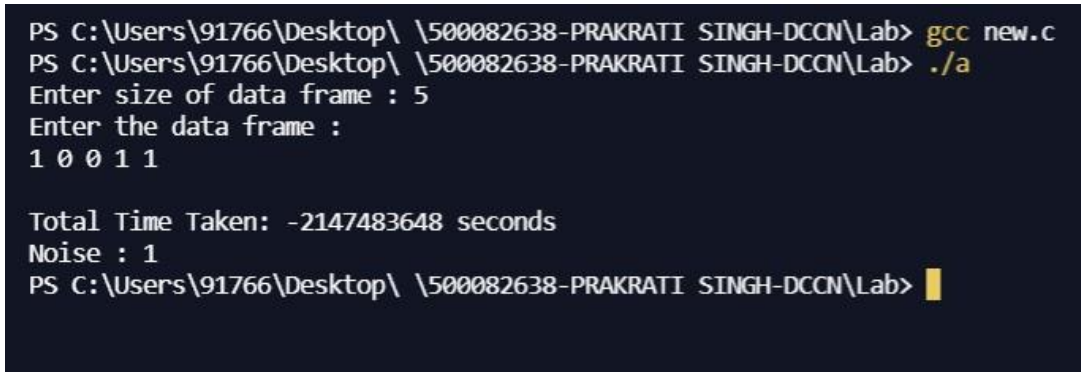
    // check for noise
    int f = noisefunc(df, recdf, n);
    if (f == 1)
    {
        printf("\nData frame has no noise");
        break;
    }
    else
    {
        // printf("\nData frame contains noise\n");
        k = k + 1;
        int R = (pow(2, k) - 1);

        Tb = (rand() % ((int)R - 0 + 1)) * slot;
    }
}
if (k > kmax)
{
    printf("ABORT");
}
```



```
d++;  
}  
printf("\nTotal Time Taken: %d seconds", Ftime);  
printf("\nNoise : %d", noise);  
}
```

OUTPUT:



```
PS C:\Users\91766\Desktop\ \500082638-PRAKRATI SINGH-DCCN\Lab> gcc new.c  
PS C:\Users\91766\Desktop\ \500082638-PRAKRATI SINGH-DCCN\Lab> ./a  
Enter size of data frame : 5  
Enter the data frame :  
1 0 0 1 1  
  
Total Time Taken: -2147483648 seconds  
Noise : 1  
PS C:\Users\91766\Desktop\ \500082638-PRAKRATI SINGH-DCCN\Lab> 
```

## CSMA/CD

How CSMA/CD works?

Stage 1: Check if the source is prepared for transmitting data parcels.

Stage 2: Check if the transmission connect is inactive?

Source needs to continue to check assuming the transmission interface/medium is inactive. For this, it consistently faculties transmissions from other hubs. Source sends faker data on the connection.

Stage 3: Transmit the data and check for collisions.

Shipper transmits its data on the connection. CSMA/CD doesn't utilize an 'affirmation' framework. It checks for effective and ineffective transmissions through crash signals.

Stage 4: If no impact was identified in proliferation, the source finishes its edge transmission and resets the counters.

# LAB TASK 6

## CSMA/CA

### CODE:

```
#include <stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>
int n = 5, df[] = {1, 1, 0, 0, 1};
void display(int arr[], int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
}
float cal(float ifs)
{
    int noise;
    int *df = (int *)malloc(n * sizeof(int));
    int *recdf = (int *)malloc(n * sizeof(int));
    float Tp = 0.00012, Tb = 0, Ftime = 0, R, tSlot = 0.000009;
    int k = 0, kMax = 15, id = 1;
    while (k < kMax)
    {
        if (id == 0)
        {
            continue;
        }
        R = pow(2, k) - 1;
        Tb = R * Tp;
        Ftime = Ftime + (2 * Tp) + ifs;
        // acknowledgement int ack=0; if(ack==1)
        {
            printf("\nAcknowledgement received!");
        }
        else
        {
            k = k + 1;
        }
    }
}
```

```

return Ftime;
}
int main()
{
    // printf("Enter the Propagation time : ");
    // scanf("%f",&Tp);
    float DIFS = 0.000034, SIFS = 0.000016, FTime;
    float time1 = 0.0, time2 = 0.0;
    time1 = cal(DIFS);
    printf("\nDifs time: %f", time1);
    time2 = cal(SIFS);
    printf("\nSifs time: %f", time2);
    FTime = time1 + time2;
    printf("\nTotal time: %f", FTime);
    return 0;
}

```

## OUTPUT:

```

PS C:\Users\91766\Desktop\ \500082638-PRAKRATI SINGH-DCCN\Lab> gcc csmaca.c
PS C:\Users\91766\Desktop\ \500082638-PRAKRATI SINGH-DCCN\Lab> ./a

Difs time: 0.004110
Sifs time: 0.003840
Total time: 0.007950
PS C:\Users\91766\Desktop\ \500082638-PRAKRATI SINGH-DCCN\Lab> 

```

## THEORY/COMMENTS:

CSMA/CA - Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) is an organization convention for transporter transmission that works in the Medium Access Control (MAC) layer. As opposed to CSMA/CD (Carrier Sense Multiple Access/Collision Detection) that arrangements with collisions after their event, CSMA/CA forestalls collisions preceding their event.

The calculation of CSMA/CA is:

1. Whenever a casing is prepared, the transmitting station checks whether the channel is inactive or occupied.
2. Assuming the channel is occupied, the station holds on until the channel becomes inactive.
3. Assuming that the channel is inactive, the station sits tight for an Inter-outline hole (IFG) measure of time and afterward sends the edge.

4. Subsequent to sending the edge, it sets a clock.

5. The station then hangs tight for affirmation from the collector. On the off chance that it gets the affirmation before expiry of clock, it denotes an effective transmission.

Otherwise, it sits tight for a back-off time-frame and restarts the calculation.

# LAB TASK 7

## Peer to Peer Network

In peer-to-peer (P2P) networking, a group of computers are linked together with equal permissions and responsibilities for processing data. The primary goal of peer-to-peer networks is to share resources and help computers and devices work collaboratively, provide specific services, or execute specific tasks. For example, some online gaming platforms use P2P for downloading games between users. P2P networks have some characteristics that make them useful:

- It's hard to take them down. Even if one of the peers is shut down, the others are still operating and communicating.
- Peer-to-peer networks are incredibly scalable. Adding new peers is easy as you don't need to do any central configuration on a central server.

Steps to set up peer to peer network:

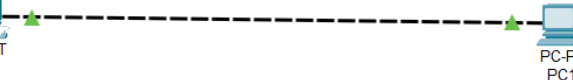
1. Open Cisco Packet Tracer.
2. Click End Devices icon and drag PC icon to the workspace.

Logical Physical x 1051, y 348



3. Click Connections icon and then select the Copper Cross-over.

Logical Physical x 371, y 194



4. Click PC0 then Click PC1.

5. Assign IP addresses to both the PCs by double clicking on the PCs.

The first screenshot shows the 'IP Configuration' window for a device with interface 'FastEthernet0'. The 'Static' radio button is selected. The IP Address is 10.10.10.11, Subnet Mask is 255.0.0.0, Default Gateway is 0.0.0.0, and DNS Server is 0.0.0.0.

The second screenshot shows the same window for another device. The 'Static' radio button is selected, but a warning message 'This address is already used in the network.' is displayed. The IP Address is 10.10.10.12, Subnet Mask is 255.0.0.0, Default Gateway is 0.0.0.0, and DNS Server is 0.0.0.0.

6. Now we double click on any Pc and open the command prompt. We will type the command ping IP-add and put IP address of the other PC.

PC1

Physical Config Desktop Programming Attributes

Command Prompt

```
Packet Tracer PC Command Line 1.0
C:\>ping 10.10.10.11

Pinging 10.10.10.11 with 32 bytes of data:

Reply from 10.10.10.11: bytes=32 time<1ms TTL=128
Reply from 10.10.10.11: bytes=32 time<1ms TTL=128
Reply from 10.10.10.11: bytes=32 time<1ms TTL=128
Reply from 10.10.10.11: bytes=32 time<1ms TTL=128

Ping statistics for 10.10.10.11:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\>
```

☐ Top

7. The Pcs are now successfully connected.