

Assignment No. 3

Title:

Write a program for error detection and correction for 7/8 bits ASCII codes using Hamming Codes or CRC. Demonstrate the packets captured traces using Wireshark Packet Analyzer Tool for peer to peer mode.(50% students will perform Hamming Code and others will perform CRC) (Use C/C++)

1. Prerequisite:

1. Data Link Layer: Roles, Protocols(Ethernet)
2. C/C++ Programming Syntax
3. Wireshark Tool

2. Learning Objectives:

- Students will able to understand Data Link Layer of OSI Model
- Students will able to understand hamming code and CRC.

3. Theory

Computer Networks Error Detection and Correction

A condition when the receiver's information does not matches with the sender's information. During transmission, digital signals suffer from noise that can introduce errors in the binary bits traveling from sender to receiver. That means a 0 bit may change to 1 or a 1 bit may change to 0.

Some popular techniques for error detection are:

1. Simple Parity check
2. Two-dimensional Parity check
3. Checksum
4. Cyclic redundancy check

1. Hamming Codes

Hamming code is a set of error-correction codes that can be used to detect and correct bit errors that can occur when computer data is moved or stored.

Like other error-correction code, Hamming code makes use of the concept of parity and parity bits, which are bits that are added to data so that the validity of the data can be checked when it is read or after it has been received in a data transmission. Using more than one parity bit, an error-correction code can not only identify a single bit error in the data unit, but also its location in the data unit.

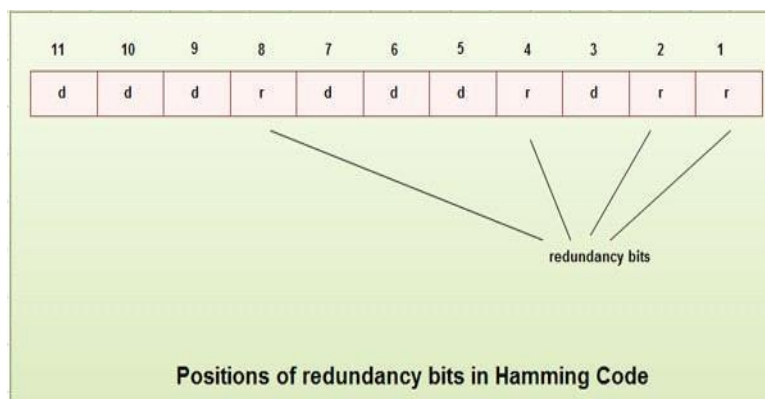
Calculating the Hamming Code

Determining the positions of redundancy bits

We know that to detect errors in a 7 bit code, 4 redundant bits are required.

Now, the next task is to determine the positions at which these redundancy bits will be placed within the data unit.

- These redundancy bits are placed at the positions which correspond to the power of 2.
- For example in case of 7 bit data, 4 redundancy bits are required, so making total number of bits as 11. The redundancy bits are placed in position 1, 2, 4 and 8 as shown in fig.



- In Hamming code, each r bit is the VRC for one combination of data bits. r_1 is the VRC bit for one combination of data bits, r_2 is the VRC for another combination of data bits and so on.
- Each data bit may be included in more than one VRC calculation.
- r_1 bit is calculated using all bits positions whose binary representation includes a 1 in the rightmost position.
- r_2 bit calculated using all the bit positions with a 1 in the second position and so on.
- Therefore the various r bits are parity bits for different combination of bits.

The various combinations are:

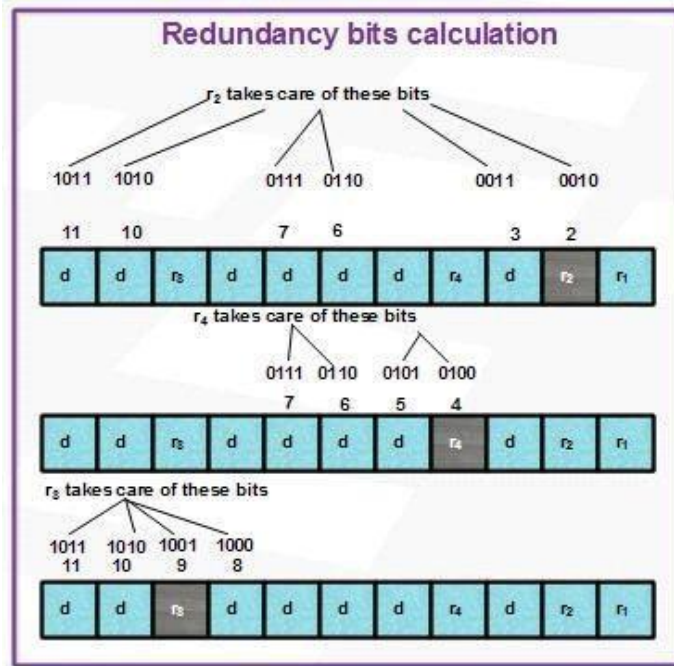
r_1 : bits 1,3,5, 7, 9, 11

r_2 : bits 2, 3, 6, 7, 10, 11

r_4 : bits 4, 5, 6, 7

r_8 : bits 8, 9, 10, 11

Example of Hamming Code Generation



Suppose a binary data 1001101 is to be transmitted. To implement hamming code for this, following steps are used:

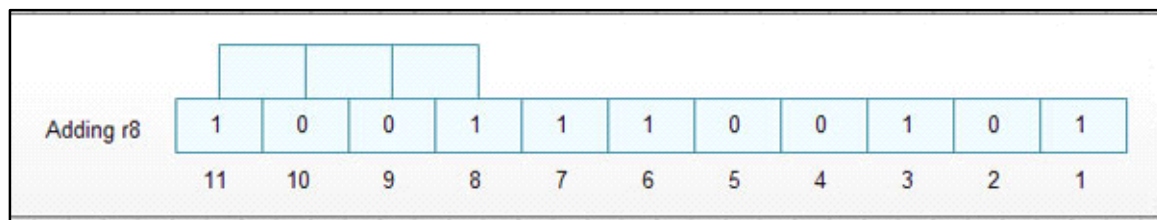
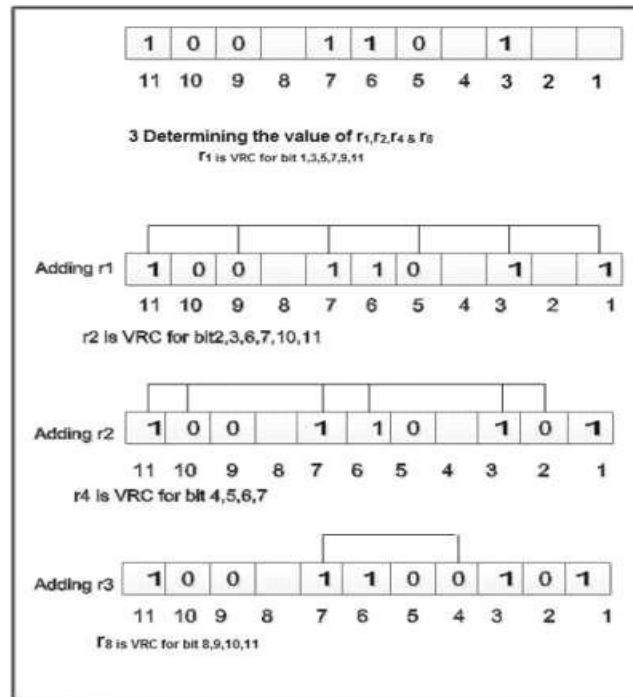
1. Calculating the number of redundancy bits required. Since number of data bits is 7, the value of r is calculated as

$$2^r \geq m + r + 1$$

$$2^4 \geq 7 + 4 + 1$$

Therefore no. of redundancy bits = 4

2. Determining the positions of various data bits and redundancy bits. The various r bits are placed at the position that corresponds to the power of 2 i.e. 1, 2, 4, 8.



4. Thus data 1 0 0 1 1 1 0 0 1 0 1 with be transmitted.

Error Detection & Correction

Data sent: 1 0 0 1 1 1 0 0 1 0 1

Data received: 1 0 0 1 0 1 0 0 1 0 1 (seventh bit changed)

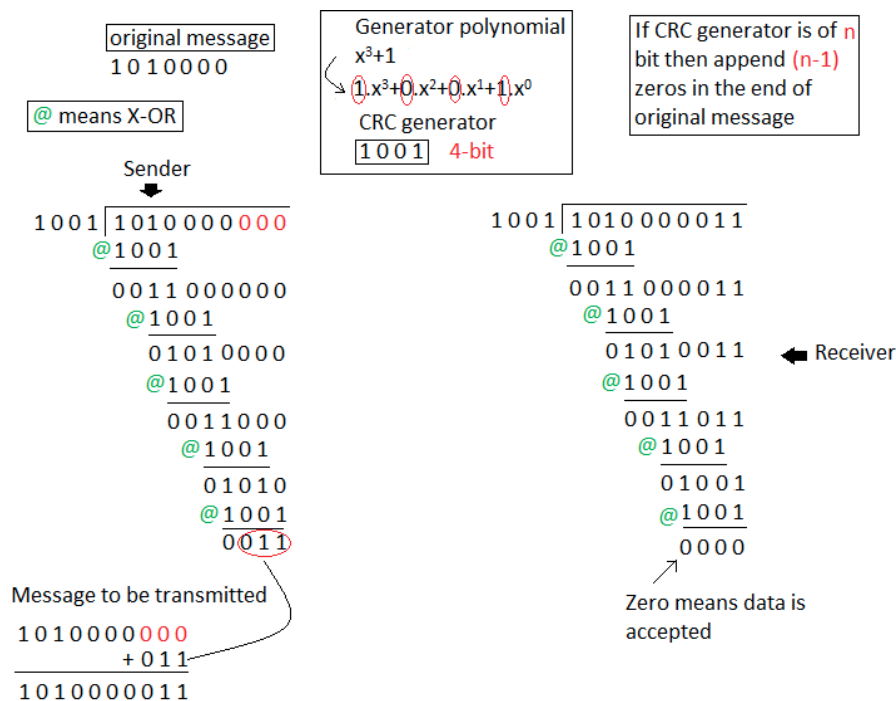
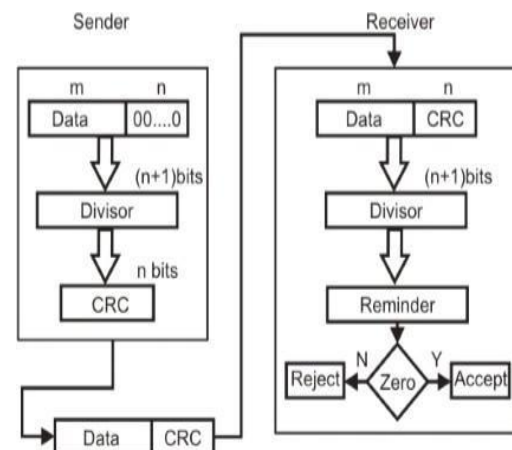
The receive takes the transmission and recalculates four new VRCs using the same set of bits used by sender plus the relevant parity (r) bit for each set as shown in fig.

Then it assembles the new parity values into a binary number in order of r position (r_8, r_4, r_2, r_1).

In this example, this step gives us the binary number 0111. This corresponds to decimal 7. Therefore bit number 7 contains an error. To correct this error, bit 7 is reversed from 0 to 1.

2. Cyclic redundancy check (CRC)

- CRC is based on binary division.
- In CRC, a sequence of redundant bits, called cyclic redundancy check bits, are appended to the end of data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number.
- At the destination, the incoming data unit is divided by the same number. If at this step there is no remainder, the data unit is assumed to be correct and is therefore accepted.
- A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.



Conclusion:

Hence we have studied error detection and correction using Hamming Codes and CRC.

Hamming code –

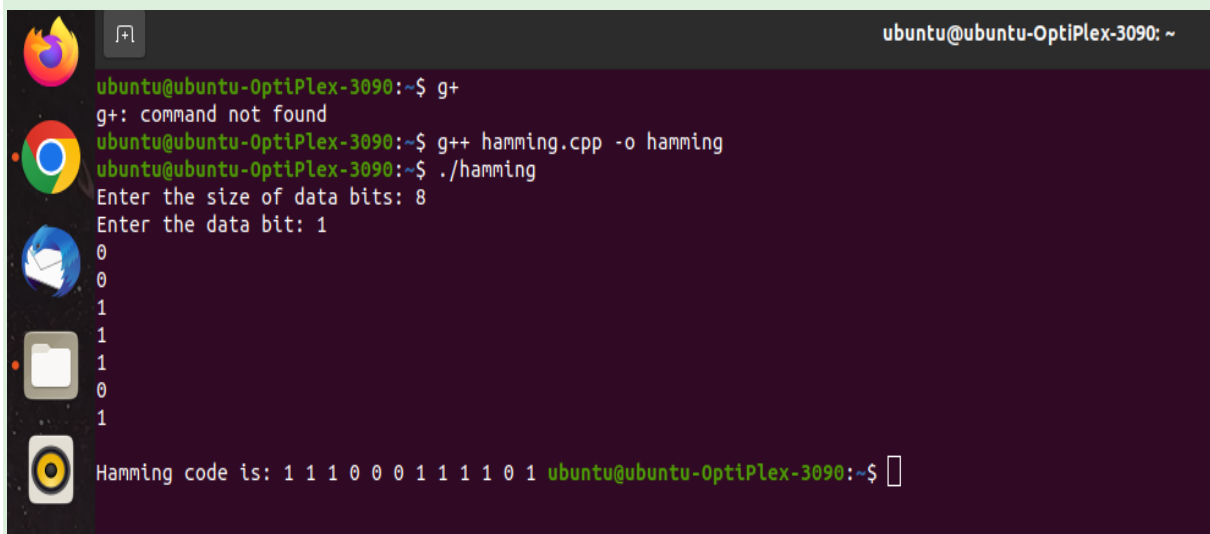
```
include <cstdlib>
#include <iostream>
#include <math.h>
using namespace std;
int main(int argc, char** argv) {
int data_bits[20],m,r = 0,parity;    //m = no. of data bits, r = no. of redundant bits
cout<<"Enter the size of data bits: ";
cin>>m;        //finding no. of redundant bits
while(pow (2,r) < m + r + 1){
r++;
}
cout<<"Enter the data bit: ";
for(int i = 1; i <= m; i++)
cin>>data_bits[i];
int hamming[m + r],j = 0,k = 1; //finding positions of redundant bits.
for(int i = 1; i <= m + r; i++){
if( i == pow( 2, j )){
hamming[i] = -1;        //-1 is initial value of redundant bits
j++;
}
else{
hamming[i] = data_bits[k];
k++;
}
}
k = 0;
int x, min, max = 0;        //finding parity bit
for (int i = 1; i <= m + r; i = pow (2, k)){
k++;
parity = 0;
j = i;
x = i;
```

```

min = 1;
max = i;
while (j <= m + r){
for (x = j; max >= min && x <= m + r; min++, x++){
if (hamming[x] == 1)
parity = parity + 1;;
}
j = x + i;
min = 1;
}

//checking for even parity
if (parity % 2 == 0){
hamming[i] = 0;
}
else{
hamming[i] = 1;
}
}
cout<<"\nHamming code is: ";
for(int i = 1; i <= m + r; i++)
cout<<hamming[i]<<" ";
return 0;
}

```



The screenshot shows a terminal window on an Ubuntu system. The user runs the following commands:

```

ubuntu@ubuntu-OptiPlex-3090:~$ g+
g+: command not found
ubuntu@ubuntu-OptiPlex-3090:~$ g++ hamming.cpp -o hamming
ubuntu@ubuntu-OptiPlex-3090:~$ ./hamming
Enter the size of data bits: 8
Enter the data bit: 1
0
0
1
1
1
0
0
1
Hamming code is: 1 1 1 0 0 0 1 1 1 1 0 1

```

The output of the program shows the Hamming code for the input data bits 11100011, which is 111000111101.

CRC code –

```
#include <iostream>
#include <math.h>
#include <cstring>
using namespace std;

char exor(char a,char b)                // perform exor operation
{
    if(a==b)
        return '0';
    else
        return '1';
}

void crc(char data[], char key[])
{
    int datalen = strlen(data);
    int keylen = strlen(key);

    for(int i=0;i<keylen-1;i++)          //appending n-1 zeroes to data
        data[datalen+i]='0';
    data[datalen+keylen-1]='\0';

    int codelen = datalen+keylen-1;      //lenght of appended data word

    char temp[20],rem[20];

    for(int i=0;i<keylen;i++)
        rem[i]=data[i];                //considering n bits of data for each step of binary division/EXOR

    for(int j=keylen;j<=codelen;j++)
    {
        for(int i=0;i<keylen;i++)
            temp[i]=rem[i];             // remainder of previous step is dividend of current step

        if(rem[0]=='0')                  //if 1's bit of remainder is 0 then shift the rem by 1 bit
        {
            for(int i=0;i<keylen-1;i++)
                rem[i]=temp[i+1];
        }
        else                            //else exor the dividend with generator key
        {
            for(int i=0;i<keylen-1;i++)
                rem[i]=exor(temp[i+1],key[i+1]);
        }

        if(j!=codelen)
            rem[keylen-1]=data[j];      //appending next bit of data to remainder obtain by division
        else
    }
```



```

        rem[keylen-1]='\0';
    }

    for(int i=0;i<keylen-1;i++)
        data[datalen+i]=rem[i];           //replace n-1 zeros with n-1 bit CRC
    data[codelen]='\0';
    cout<<"CRC="<<rem<<"\nDataword="<<data;

}

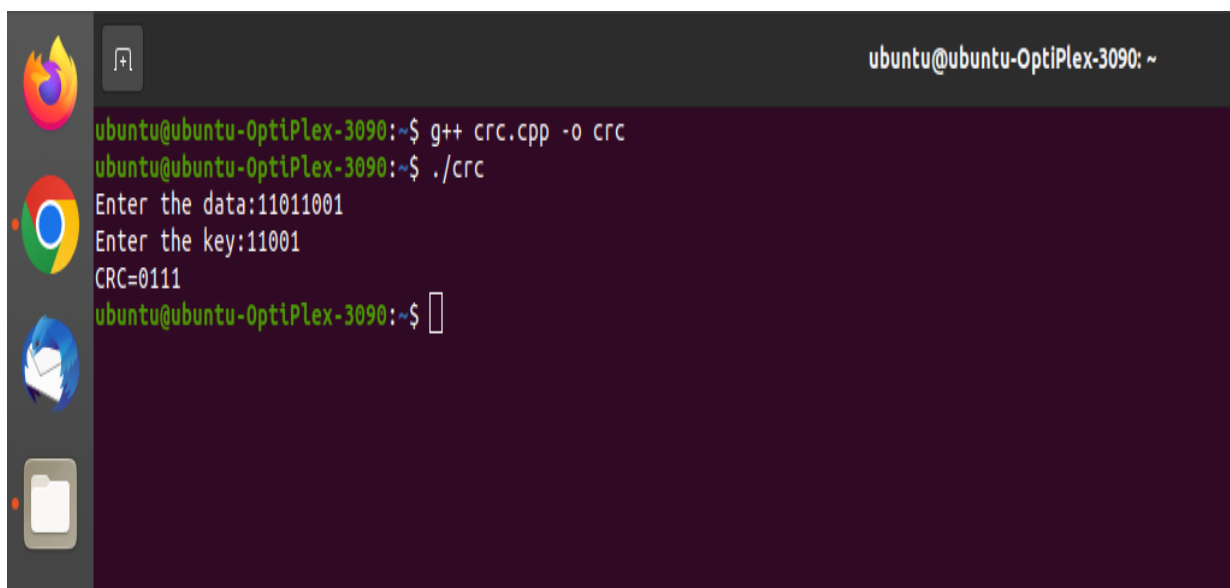
int main()
{
    char key[20],data[20];

    cout<<"Enter the data:";
    cin>>data;
    cout<<"Enter the key:";
    cin>>key;

    crc(data,key);           // generate crc

    return 0;
}

```



The screenshot shows a terminal window titled 'ubuntu@ubuntu-OptiPlex-3090: ~'. The user has compiled a C++ program named 'crc.cpp' into an executable named 'crc' using 'g++'. They then ran './crc', which prompted them to enter data and a key. The data entered was '11011001' and the key was '11001'. The program outputted 'CRC=0111'. The terminal window has a dark purple background and a sidebar on the left with icons for Firefox, Google Chrome, and a file manager.

```

ubuntu@ubuntu-OptiPlex-3090:~$ g++ crc.cpp -o crc
ubuntu@ubuntu-OptiPlex-3090:~$ ./crc
Enter the data:11011001
Enter the key:11001
CRC=0111
ubuntu@ubuntu-OptiPlex-3090:~$ 

```