VIT Vidyalankar Institute of Technology
Accredited A+ by NAAC

DEPARTMENT OF COMPUTER ENGINEERING
Computer Network Lab

Semester	T.E. Semester V – Computer Engineering
Subject	Computer Network
Subject Professor In-charge	Prof. Amit K. Nerurkar
Assisting Teachers	Prof. Amit K. Nerurkar
Laboratory	M-313-A

Student Name	Deep Salunkhe
Roll Number	21102A0014
TE Division	A

Title : Error Handling

Theory:

Parity Check:

Parity is a simple error-checking method used in digital communication to detect errors in data transmission. It involves adding an extra bit, called a **parity bit**, to a group of data bits.

- **Even Parity:** In even parity, the total number of 1s in the data and parity bit combined should be even. If there's an odd number of 1s, an error is detected.
- **Odd Parity:** In odd parity, the total number of 1s in the data and parity bit combined should be odd. If there's an even number of 1s, an error is detected.

Parity check is useful for detecting single-bit errors. If a single bit is flipped during transmission, the parity check will detect it. However, it cannot correct errors, only identify them.

Checksum:

A **checksum** is a value calculated from a data set for the purpose of error checking. It is generated by summing up the binary values of data in a specific way and then appending the checksum to the data. When the data is received, the recipient can calculate the checksum again and compare it to the received checksum to check for errors.

Checksums are commonly used in networking protocols, such as the Internet Control Message Protocol (ICMP) and the Transmission Control Protocol (TCP), to ensure data integrity during transmission.

One of the most common checksum algorithms is the **Internet Checksum**, which calculates a 16-bit checksum for data. If the calculated checksum at the receiver's end doesn't match the received checksum, an error is detected.

CRC (Cyclic Redundancy Check):

A **Cyclic Redundancy Check (CRC)** is an advanced error-checking method used for data transmission. CRCs use polynomial division to detect errors. A specific polynomial, called the **generator polynomial**, is used for CRC calculations.

Here's how CRC works:

1. The sender and receiver agree on a generator polynomial (e.g., x^3 + x + 1).
2. The sender appends a certain number of zeros (equal to the degree of the generator polynomial) to the data.
3. The sender performs polynomial division on the combined data to create a **remainder**, which is appended to the data.
4. The data with the remainder is transmitted.
5. The receiver performs the same polynomial division using the received data, including the remainder.
6. If the remainder at the receiver's end is not zero, an error is detected.

CRCs are powerful error-detection methods and are widely used in network communication and storage systems. They can detect a wide range of errors, including burst errors, and are more robust than simple parity checks or checksums.

Implementation:

```cpp
#include <iostream>
#include <vector>
#include <cstdlib>
using namespace std;

// Parity check functions
void sender_code_parity_check(vector<int>& data) {
    cout<<"enter the data to transmit........"<<endl;
    int n;
    cout<<"enter the number of bits in data:";
    cin>>n;
    //taking input and calculating parity bit
    int n_one=0;
    cout<<"start entering data bitwise"<<endl;
    for(int i=0;i<n;i++)
    {
        int temp;
        cin>>temp;
        if(temp==1)n_one++;
        data.push_back(temp);
    }
    //now adding parity bit to last
    int parity_bit;
    if(n_one%2==0)
    {
        //if even
        parity_bit=0;
    }else{
        //if odd
        parity_bit=1;
    }
    //appending parity bit to last
    data.push_back(parity_bit);
}

// Transmitter function
void transmit_P(vector<int>& data) {
    int n=data.size();

    cout<<"want to transmit with error(1) or not(0):";
```

```
    int check=0;
    cin>>check;

    //processing according to instruction
    if(check==0)
    {
        //without error

        //not updating the data
    }else{
        //with error

        //now converting data with someerror
        //as this is parity bit it can handle only 1 bit error
        int posi=rand()%n;
        //fliping the bit (to generate error)
        if(data[posi]==1)
        {
            data[posi]=0;
        }else{
            data[posi]=1;
        }
    }
}

void reciever_code_parity_check(vector<int>& data) {
    //calculating no_of_ones
    int n_one=0;
    int n=data.size();

    for(int i=0;i<n;i++)
    {
        if(data[i]==1)n_one++;
    }
    //finding parity bit for reciever
    int parity_bit;
    if(n_one%2==0)
    {
        //if even
        parity_bit=0;
    }else{
        //if odd
        parity_bit=1;
```

```
    }
    if(parity_bit==0)
    {
        cout<<"data send.........(without error)";
    }else{
        cout<<"data send..........(with error)";
    }
}

//-------------------------------------------------------------------------------
-------------------------
// Check Sum functions
void accept_Bits(vector<int>& f) {
    cout<<"\nEnter the 8 bit binary: ";
    for(int i = 0; i<8; i++){
        cin>>f[i];
    }
    cout << "\nf: ";
    for(int i = 0; i<8; i++){
        cout<<f[i];
    }

}

void divide(vector<int>& f, vector<int>& f1, vector<int>& f2) {
    for (int i = 0; i < 4; i++) {
        f1.push_back(f[i]);
    }
    for (int i = 4; i < 8; i++) {
        f2.push_back(f[i]);
    }
    cout << "\nf1: ";
    for (int i = 0; i < f1.size(); i++) {
        cout << f1[i];
    }
    cout << "\nf2: ";
    for (int i = 0; i < f2.size(); i++) {
        cout << f2[i];
    }
}

void binaryAddition(vector<int>& f1, vector<int>& f2, vector<int>& result) {
    int carry = 0;
```

```
    for (int i = 3; i >= 0; i--) {
        int sum = f1[i] + f2[i] + carry;
        if (sum == 0) {
            result[i] = 0;
            carry = 0;
        } else if (sum == 1) {
            result[i] = 1;
            carry = 0;
        } else if (sum == 2) {
            result[i] = 0;
            carry = 1;
        } else { // sum == 3
            result[i] = 1;
            carry = 1;
        }
    }
}

void binaryComplement(vector<int>& Bin_result, vector<int>& Comp) {
    for(int i = 0; i<Bin_result.size(); i++){
        if(Bin_result[i] == 1){
            Bin_result[i] = 0;
        }
        else if(Bin_result[i] == 0){
            Bin_result[i] = 1;
        }
        Comp.push_back(Bin_result[i]);
    }
    cout << "\nComplement of Binary Addition: ";
    for (int i = 0; i < Comp.size(); i++) {
        cout << Comp[i];
    }
}

void CheckSumRecivers(vector<int>& f, vector<int>& Bin_result) {
    vector<int> Rf1;
    vector<int> Rf2;
    vector<int> Rf3;
    vector<int> RBin_result(4, 0);
    for (int i = 0; i < 4; i++) {
        Rf1.push_back(f[i]);
    }
    for (int i = 4; i < 8; i++) {
```

```
        Rf2.push_back(f[i]);
    }
    for (int i = 8; i < 12; i++) {
        Rf3.push_back(f[i]);
    }
    cout << "\nRf1: ";
    for (int i = 0; i < Rf1.size(); i++) {
        cout << Rf1[i];
    }
    cout << "\nRf2: ";
    for (int i = 0; i < Rf2.size(); i++) {
        cout << Rf2[i];
    }
    cout << "\nRf3: ";
    for (int i = 0; i < Rf3.size(); i++) {
        cout << Rf3[i];
    }
    binaryAddition(Rf1, Rf2, Bin_result);
    binaryAddition(Bin_result, Rf3, RBin_result);
    cout << "\nBinaryAddition: ";
    for (int i = 0; i < RBin_result.size(); i++) {
        cout << RBin_result[i];
    }
    int count = 0;
    int i = 0;
    for( ; i< RBin_result.size(); i++){
        if(RBin_result[i] == 1){
            count++;
        }
    }
    cout<<"\n\nFinal Output: ";
    if((count%2) == 0){
        cout<<"No error\n";
    }
    else{
        cout<<"Error\n";
    }
}

// Check Sum main function
void checkSum(vector<int>& data) {
     cout<<"Check Sum technique\n";
    vector<int> f(8);
```

```
    vector<int> f1;
    vector<int> f2;
    vector<int> Comp;
    accept_Bits(f); //accepts the 8 bits binary message
    divide(f, f1, f2); //Divides main binary message to 4 bits two frames
    vector<int> Bin_result(4, 0);
    cout<<"\n\nSender's Side calculations =>\n";
    binaryAddition(f1, f2, Bin_result);
    cout << "\nBinaryAddition: ";
    for (int i = 0; i < Bin_result.size(); i++) {
        cout << Bin_result[i];
    }
    binaryComplement(Bin_result, Comp);
    cout<<"\n";
    f.insert(f.end(), Comp.begin(), Comp.end());
    cout<<"Data + Complement: ";
    for(int i = 0; i<f.size(); i++){
        cout<<f[i];
    }
    cout<<"\n\nReciever's Side calculations =>\n";
    CheckSumRecivers(f, Bin_result);
}
//--------------------------------------------------------------------------------
------------------------
// CRC function
std::vector<int> xor_division(const std::vector<int> &dividend, const
std::vector<int> &divisor) {
    std::vector<int> remainder = dividend;

    for (int i = 0; i <= remainder.size() - divisor.size(); ++i) {
        if (remainder[i] == 1) {
            for (int j = 0; j < divisor.size(); ++j) {
                remainder[i + j] ^= divisor[j];
            }
        }
    }

    // Return the remainder after division
    return std::vector<int>(remainder.begin() + remainder.size() -
divisor.size(), remainder.end());
}

void printdata(vector<int>data)
```

```
{
    int n=data.size();
    for(int i=0;i<n;i++)
    {
        cout<<data[i];
    }
    cout<<endl;
}
void sender_crc(vector<int>&data,vector<int>&gx)
{
    //accepting data in binary form only
    cout<<"enter the data to transmit........"<<endl;
    int n;
    cout<<"enter the number of bits in data:";
    cin>>n;
    //taking input
    cout<<"start entering data bitwise"<<endl;
    for(int i=0;i<n;i++)
    {
        int temp;
        cin>>temp;
        data.push_back(temp);
    }

    //taking g(x) as input
    int n2;
    cout<<"enter number of bits in g(x):";
    cin>>n2;
    //taking input
    cout<<"start entering data bitwise"<<endl;
    for(int i=0;i<n2;i++)
    {
        int temp;
        cin>>temp;
        gx.push_back(temp);
    }

    //calculating number of zeros to be appended
    int nu_zero=n2;
    nu_zero--;

    //appending n zero's to the data(bits of gx)
    vector<int>datacopy;
```

```
    datacopy=data;
    for(int i=0;i<nu_zero;i++)
    {
        datacopy.push_back(0);
    }

    //xor division datacopy/gx
    vector<int> remainder = xor_division(datacopy, gx);
    //append remainder to data and then send that data
    for(int i=0;i<remainder.size();i++)
    {
        data.push_back(remainder[i]);
    }
}

void reciever_crc(vector<int>&data,vector<int>&gx)
{
    //xor division datacopy/gx
    vector<int> remainder = xor_division(data,gx);

    //calculating that remainder is zero or not
    bool iszero=true;

    for(int i=0;i<remainder.size();i++)
    {
        if(remainder[i]==1)
        {
            iszero=false;
        }
    }
    cout<<"printing remainder........."<<endl;
    printdata(remainder);

    if(iszero)
    {
        cout<<"data send.........(without error)";
    }else{
        cout<<"data send.........(with error)";
    }
}


int main() {
```

```
    int C;
    while (true) {
        cout << "\nMenu\n1. Parity check\n2. CheckSum\n3. CRC\n";
        cout << "\nEnter the choice: ";
        cin >> C;

        vector<int> data;
        vector<int> gx;

        switch (C) {
            case 1:
                sender_code_parity_check(data);
                transmit_P(data);
                reciever_code_parity_check(data);
                break;
            case 2:
                checkSum(data);
                break;
            case 3:
                sender_crc(data, gx);
                transmit_P(data);
                reciever_crc(data, gx);
                break;
            default:
                break;
        }
    }

    return 0;
}
```

Output:

```
Menu
1. Parity check
2. CheckSum
3. CRC

Enter the choice: 2
Check Sum technique

Enter the 8 bit binary: 1 0 1 1 1 0 0 1

f: 10111001
f1: 1011
f2: 1001

Sender's Side calculations =>

BinaryAddition: 0100
Complement of Binary Addition: 1011
Data + Complement: 101110011011

Reciever's Side calculations =>

Rf1: 1011
Rf2: 1001
Rf3: 1011
BinaryAddition: 1111

Final Output: No error

Menu
1. Parity check
2. CheckSum
3. CRC
```

Conclusion:

In this lab, we explored three important methods for error detection in digital communication: Parity Check, Checksum, and Cyclic Redundancy Check (CRC). Each of these methods plays a crucial role in ensuring data integrity during transmission. Here are the key takeaways from this lab:

- **Parity Check** is a straightforward method that involves adding an extra bit to data to make the total number of 1s (either even or odd) in the data and parity bit even or odd. It can detect single-bit errors but cannot correct them.
- **Checksums** are values calculated from data sets to verify data integrity. A checksum is appended to the data, and the recipient can recalculate it to check for errors. Checksums are commonly used in networking protocols to detect errors during data transmission.
- **Cyclic Redundancy Check (CRC)** is an advanced error-checking method that uses polynomial division. A generator polynomial is agreed upon by both sender and receiver. The sender appends a remainder obtained through polynomial division to the data. The receiver performs the same division and checks if the remainder is zero. CRCs are powerful and can detect a wide range of errors, including burst errors