ERROR DETECTION TECHNIQUES
COMPUTER NETWORK

Submitted By :

- Trishansh Sahane
- Vaidik Lotan

Under the Guidance of :

Mrs. Deepali Panwar

Acknowledgement :

We would like to express our sincere gratitude to **Deepali Panwar Mam** for her invaluable guidance and support throughout this project. Her mentorship helped us gain a deep understanding of error detection techniques and successfully complete this project.

Introduction :

This project focuses on exploring various error detection techniques used in computer networks to ensure the accurate transmission of data. These methods play a critical role in identifying and minimizing data errors during communication, enhancing data integrity, and reliability in networking systems.

In this project we have written the python code for various Error Detection Techniques used to detect errors.

Objective :

The objective of this project is to study, understand, and implement different **error detection techniques** and analyse their applications in real-world scenarios.

Github : https://github.com/trish-2610/Computer-Network

Error detection is the process of identifying mistakes or changes in data during transmission over a network. It ensures that the data received is same as data sent.

PARITY CHECKING

Definition : Parity checking is a simple error-detection technique used in digital communication and storage systems to ensure data integrity. It involves adding a **parity bit** to data to make the total number of 1s in the binary representation either even or odd, depending on the type of parity. In this an extra bit (Parity bit) is added to each word before transmitting.

Types of Parity Checking :

1. **Even Parity**:
    - Ensures the total number of 1s (including the parity bit) is even.
    - Example:
        - Data: `1010`
        - Add parity bit: `1` (to make the total 1s = 4, even)
        - Transmitted data: `10101`
2. **Odd Parity**:
    - Ensures the total number of 1s (including the parity bit) is odd.
    - Example:
        - Data: `1010`
        - Add parity bit: `0` (to keep the total 1s = 3, odd)
        - Transmitted data: `10100`

Working of Parity Checking :

1. **At the Sender's End** :
    - The sender calculates the parity bit and appends it to the data before transmission.
2. **At the Receiver's End** :
    - The receiver recalculates the parity from the received data.
    - If the recalculated parity matches the parity bit, the data is considered error-free.
    - If it doesn’t match, an error is detected.

Limitations of Parity Checking :

- **Cannot fix errors** : It only detects errors but cannot correct them.
- **Limited detection** : Detects errors affecting an odd number of bits,errors in an even number of bits remain undetected.

- **Not suitable for complex systems** : Advanced error detection methods like checksums or CRC (Cyclic Redundancy Check) are more reliable.

Applications of Parity Checking :

- **Memory Systems** : Detecting errors in RAM.
- **Communication Channels** : Ensuring data integrity in simple communication protocols.
- **Storage Devices** : Verifying data blocks in basic systems.

Parity checking is a straightforward technique used in systems where low-cost and simple error detection is sufficient.

Python code for Parity Checking :

```
try :
    data = int(input("Enter data : "))
    print("Actual Data = ",data)

    if(type(data) is int) :
        def dec_bin(data):
            data = bin(data)
            data = str(data)
            data = str.split(data,"b")[1]
            return data

        data = dec_bin(data)

        print(data)

        def even_parity(data):
            updated_even_data = "0" + str(data)
            print(updated_even_data)
            ones_count = 0
            for i in updated_even_data:
                if(i=="1"):
                    ones_count += 1
            if(ones_count % 2 == 0):
                print("Even Parity : NO ERROR")
            else :
                print("Even Parity : ERROR")
```

```
        def odd_parity(data):
            updated_odd_data = "1" + str(data)
            print(updated_odd_data)
            ones_count = 0
            for i in updated_odd_data:
                if(i=="1"):
                    ones_count += 1
            if(ones_count % 2 != 0):
                print("Odd Parity : NO ERROR")
            else :
                print("Odd parity : ERROR")

        even_parity(data)
        odd_parity(data)

    else :
        print("Exception : Not an Integer")

except Exception :
    print("Exception : Not an Integer Value")
```

Output :

```
Enter data : 36
Actual Data =  36
100100
0100100
Even Parity : NO ERROR
1100100
Odd Parity : NO ERROR
```

```
Enter data : 31
Actual Data =  31
11111
011111
Even Parity : ERROR
111111
Odd parity : ERROR
```

CHECKSUM

Definition : Checksum is an error-detection technique used in digital communication and data storage systems to verify data integrity. It involves generating a small fixed-size value (checksum) from a block of data and comparing it during transmission or storage to detect errors.

Example : Think of a checksum as a "quick test" to ensure that data hasn't been accidentally corrupted while being sent or saved. It's like verifying the total of a shopping bill to ensure nothing is missing or extra.

Working of Checksum :

1. **At the Sender's End**:
    - A mathematical algorithm (sum of all bytes) is applied to the data to calculate the checksum.
    - The checksum is appended to the data and sent to the receiver.
2. **At the Receiver's End**:
    - The same algorithm is applied to the received data to recalculate the checksum.
    - If the recalculated checksum matches the transmitted checksum, the data is considered error-free.
    - If it doesn't match, an error is detected.

Advantages of Checksum :

- **Simple to Implement**: Easy to calculate and verify using basic arithmetic.
- **Lightweight**: Adds minimal overhead to the data being transmitted or stored.
- **Detects Common Errors**: Effective at identifying errors like flipped bits or corrupted data during transmission.

Limitations of Checksum :

- **Cannot correct errors**: Like parity checking, it only detects errors, not fixes them.
- **Limited accuracy**: May fail to detect errors in certain cases, such as when multiple errors cancel each other out.
- **Not robust for complex systems**: Advanced error detection methods like CRC or Hamming Code are more reliable for high-reliability systems

Applications of Checksum :

- **Data Transmission**: Ensuring data integrity in networks (TCP/IP protocols).
- **File Transfers**: Verifying file integrity during downloads or data transfer.
- **Storage Systems**: Detecting errors in stored data blocks.

Checksum is a simple and effective technique for error detection, widely used in systems requiring lightweight and efficient data integrity verification.

Python code for Checksum :

```
def transmitter():
    transmitted_lst = []
    while(True):
            try:
                data = int(input("Enter Transmitted Data : "))
                if(data == 0):
                    break
                transmitted_lst.append(data)
            except:
                print("Not an Integer")
                break

    print(transmitted_lst)

    checksum_transmitted_data= 0
    for i in transmitted_lst:
        checksum_transmitted_data += i
    print(checksum_transmitted_data)
    return checksum_transmitted_data
```

```
def receiver():
    received_lst = []
    while(True):
            try:
                data = int(input("Enter Received Data : "))
                if(data == 0):
                    break
                received_lst.append(data)
            except:
                print("Not an Integer")
                break

    print(received_lst)

    checksum_received_data= 0
    for i in received_lst:
        checksum_received_data += i
    print(checksum_received_data)
    return checksum_received_data
```

```
checksum_trans_data = transmitter()
checksum_rec_data = receiver()

Total = checksum_trans_data - checksum_rec_data

if(Total == 0):
    print("No Error")
else :
    print("Error")
```

Output :

```
Enter Transmitted Data : 12
Enter Transmitted Data : 32
Enter Transmitted Data : 43
Enter Transmitted Data : 0
[12, 32, 43]
87
Enter Received Data : 12
Enter Received Data : 32
Enter Received Data : 43
Enter Received Data : 0
[12, 32, 43]
87
No Error
```

```
Enter Transmitted Data : 12
Enter Transmitted Data : 98
Enter Transmitted Data : 0
[12, 98]
110
Enter Received Data : 34
Enter Received Data : 5
Enter Received Data : 0
[34, 5]
39
Error
```