Name:Het M. Thakar

Topic: Error Detection Techniques

Project Title: Bit Error Simulation using Python

Abstract / Executive Summary:

This project demonstrates the concept of Error Detection and Correction techniques used in digital communication systems to ensure data integrity during transmission.
Using Python programming, a simulation of three major techniques — Parity Check, Checksum, and Cyclic Redundancy Check (CRC) — has been implemented. These methods are essential to detect and sometimes correct errors that occur due to noise, interference, or distortion in transmission channels.
The simulation introduces random bit errors into data streams and verifies how each method identifies or corrects the corrupted bits.
Protocols such as TCP, ICMP, and ARP rely heavily on such mechanisms for reliable communication.
Tools/Technologies used: Python 3, socket, random, and binascii libraries.

Introduction:

Error detection and correction play a crucial role in ensuring reliable communication between devices over a network. Data transmitted from sender to receiver may get corrupted due to noise, attenuation, or other channel issues.

In networking, routing protocols (like TCP/IP) depend on these mechanisms to verify the integrity of packets. For instance, the TCP checksum validates packet correctness, while protocols like ICMP or ARP depend on similar integrity checks.

Wireshark is a widely used packet analyzer that helps visualize these errors, packet drops, and retransmissions. Packet analysis assists in troubleshooting, performance monitoring, and network learning.

Objectives of this Report:

- To simulate error detection using CRC, Parity, and Checksum methods.
- To demonstrate how data errors can be introduced and detected using Python.
- To understand how such mechanisms are applied in real-world network protocols.

Role of Programming in Networking:
Programming automates the detection, transmission, and analysis of data packets in networks.

Why Python?
Python is preferred because of its simplicity, strong standard libraries (`socket`, `struct`, `random`, `binascii`), and readability — making it ideal for network and data simulations.

Background & Theoretical Concepts

Sockets (TCP vs UDP)

- **TCP (Transmission Control Protocol):** Reliable, connection-oriented protocol that uses checksum for error detection.
- **UDP (User Datagram Protocol):** Faster, connectionless, and may tolerate packet loss without correction.

Client-Server Model

Communication over the internet follows a **client-server** model, where the client sends requests and the server responds, using sockets to transmit data.

OSI/TCP-IP Layers Involved

Error detection occurs mainly at the **Data Link** and **Transport** layers of the OSI model.

Protocols Implemented

The Python simulation mimics functionalities used in protocols like **TCP checksum** and **CRC used in Ethernet frames**.

Libraries and Functions Used

- **socket** – for network communication simulation.
- **random** – to introduce bit errors.
- **binascii** – for CRC and binary conversions.
- **struct** – for byte-level data handling.

Methodology / Simulation Setup

Tools Used

- **Python 3.11+**
- **VS Code / Jupyter Notebook**
- **Wireshark** (optional – to verify checksum mechanisms visually)

Simulation Setup

This project doesn't use routers or switches; instead, a **virtual data transmission** is simulated using Python scripts.

Network Topology Diagram

[Sender] → (Introduce Error) → [Receiver]

Step-by-Step Process

1. Input data message (binary or ASCII).
2. Compute **Parity / Checksum / CRC** at sender side.
3. Introduce random bit errors in transmission.
4. Receiver recomputes and verifies received data.
5. Display whether error is detected (and corrected, if possible).

Capture Filter vs Display Filter

While this simulation does not use real packet captures, Wireshark filters like tcp.checksum_bad == 1 could be used to observe corrupted packets.

Flowchart of Program Logic

Input Data from Sender
Compute CRC / Parity / Check
Introduce Bit Error Randomly
Receiver Recomputes Value
Compare Results → Detect Error

Implementation & Results

Python Code Snippet

```
import random, binascii

def introduce_error(data):
    pos = random.randint(0, len(data)-1)
    data = list(data)
    data[pos] = '1' if data[pos] == '0' else '0'
    return ''.join(data)

def parity_bit(data):
    count = data.count('1')
    return '0' if count % 2 == 0 else '1'

def checksum(data):
    return sum(ord(ch) for ch in data) % 256

def crc16(data):
    crc = binascii.crc_hqx(data.encode(), 0xffff)
    return format(crc, '04x')

data = "HELLO"
print("Original:", data)

# Parity
p = parity_bit(''.join(format(ord(c),'08b') for c in data))
print("Parity Bit:", p)

# Checksum
cs = checksum(data)
print("Checksum:", cs)

# CRC
crc = crc16(data)
print("CRC-16:", crc)

# Simulate error
err_data = introduce_error(''.join(format(ord(c),'08b') for c in data))
```

print("Erroneous data:", err_data)

Results Observed

Technique	Error Detection	Correction Possible	Remarks
Parity Bit	Simple	No	Detects single-bit errors
Checksum	Medium	No	Used in TCP, IP headers
CRC	Strong	Sometimes	Used in Ethernet, Storage systems

```
Original: HELLO
Parity Bit: 0
Checksum: 116
CRC-16: 49d6
Erroneous data: 010010000100010101011100010011000100111
```

Conclusion

This project successfully simulates the working of **error detection and correction techniques** using Python.
Parity, Checksum, and CRC were implemented to detect transmission errors, showing the reliability levels of each method.

Error detection ensures **data integrity**, making it a vital component of networking and communication systems.
Python's simplicity and extensive libraries make it an ideal choice for such simulations, helping students and engineers visualize low-level communication processes.

References

- Forouzan, B. A., *Data Communications and Networking*, McGraw Hill.
- Tanenbaum, A. S., *Computer Networks*.
- Python Documentation: https://docs.python.org/3/library
- Wireshark User Guide: https://www.wireshark.org