## Computer Networks(ARM-205)

### **Assignment-1**

# **Parity XOR Check:**

This technique works by XORing together all of the data bits and storing the result as the parity bit. The sender can then transmit the data along with the parity bit, and the receiver can XOR the received data bits and compare the result with the received parity bit to detect any errors.

# **Time Complexity:**

The time complexity is O(n), where n is the number of bits in the data, since the parity must be calculated and compared for each bit of the data.

# **Space Complexity:**

The space complexity of this technique is O(1), since only a single bit is needed to store the parity.

#### Code:

```
def send_data(data):
    # Calculate parity of data
    parity = calculate_parity(data)

# Send data and parity
    send_packet(data, parity)

def calculate_parity(data):
    # XOR together all data bits
    parity = 0
    for bit in data:
        parity = parity ^ bit

return parity
```

```
def receive_data():
    # Receive data and parity
    data, parity = receive_packet()

# Calculate parity of received data
    calculated_parity = calculate_parity(data)

# Check if parities match
    if parity != calculated_parity:
        return None, ERROR

return data, NO_ERROR
```

## **Comparison:**

Parity XOR Check is a technique for detecting errors in transmitted data by calculating a single parity bit using the XOR operation and comparing it with the received parity bit. Longitudinal Redundancy Check (LCR) and Variable Redundancy Check (VRC) are similar techniques that involve calculating a set of redundant bits based on the values of the data bits. These techniques can detect a larger number of errors compared to Parity XOR Check, but they also have higher space and time complexity due to the need to store and process the redundant bits. All three techniques have a time complexity of O(n), where n is the number of bits in the data.

Submitted to:

Dr Rahul Johari

Priyanshu Singh

AI-ML(B-1)

(00719011621)