

**Activity based**

**Project Report on**

**Computer Networks**

**Submitted to Vishwakarma University, Pune**

**Under the Initiative of**

**Contemporary Curriculum, Pedagogy, and Practice (C2P2)**

**By**

**Merwin Pinto**

**SRN No : 202100102**

**Roll No : 01**

**Diya Oswal**

**SRN No : 202101718**

**Roll No : 40**

**Div : E**

**Third Year Engineering**

**Department of Computer Engineering**

**Faculty of Science and Technology**

**Academic Year**

**2023-2024**

**Project Description:**

Parity Bit Checker for Error detection and Correction:

The parity bit checker is a network method designed to detect errors and check the integrity of the data received at the receiver side by the sender side. The parity check method adds a bit to the original data for checking errors at the receiver end.

There are mainly two types of Parity that is **Even Parity and Odd Parity**

**This Parity Checker is further divided into 3 parts**

1. Message representation into Binary bits and frames
2. Parity bit checking using Even and odd parity concepts
3. comparing performances with other error detection Techniques

**PROJECT MODULE 2 :**

**Parity bit checking using Even and odd parity concepts**

**Even Parity :**

The total number of 1n’s in the code , including parity bit should be even

**Example : 1**

1 0 1 1 0 0 1    X

As the total number of 1ns in this is four its an even number of 1n’s so we add a parity 0 to it making it Even parity , ie even number of 1ns maintained

1 0 1 1 0 0 1    0

**Example : 2**

1 0 1 1 0 1 1    X

As the total number of 1ns in this is five its an odd number of 1n’s so we add a parity 1 making it an even parity ie Even number of 1ns maintained

ie

1 0 1 1 0 1 1    1

**Odd parity:**

The total number of 1n’s in the code , including parity bit should be odd

**Example : 1**

1 0 1 1 0 0 1    X

As the total number of 1ns in this is four its an even number of 1n’s so we add a parity 1 to it making it odd parity , ie odd number of 1ns maintained

1 0 1 1 0 0 1    1

**Example : 2**

1 0 1 1 0 1 1    X

As the total number of 1ns in this is five it's an odd number of 1n’s so we add a parity 0 making it an odd parity  ie odd number of 1ns maintained

ie

1 0 1 1 0 1 1    1

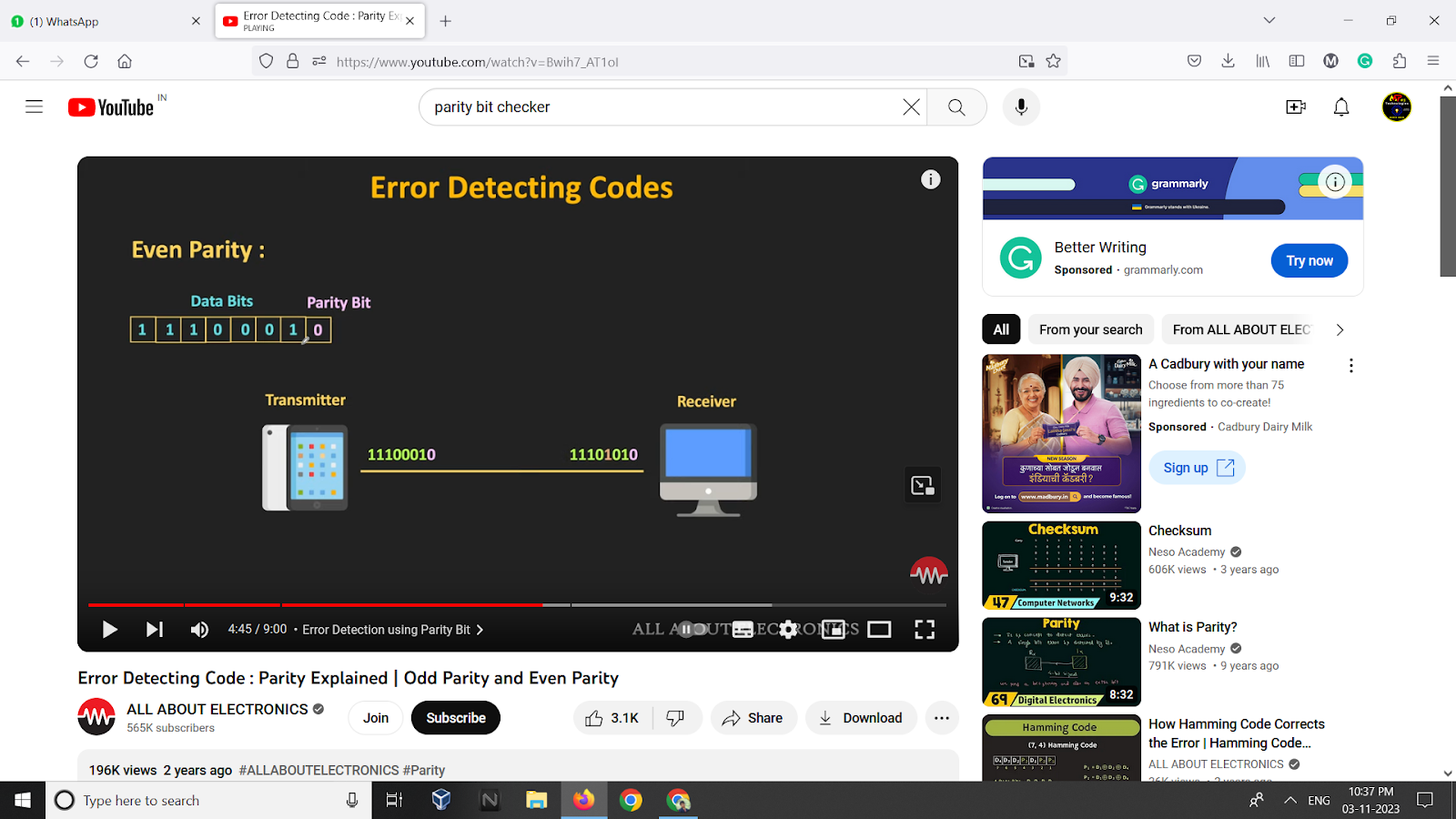
Thus from the Above example if code is sent to receiver from transmitter

**Suppose**

**Where X = 0 is a parity bit to maintain even parity**

 if Even number of 1ns were transmitted ( 1110**0**010 X even number of 1ns) and due to some noise one of the 1ns turn zero ie ( 1110**1**010 X ) during transmission. when parity checker checks  at the receiver's end, it found a change and detected an error in the data code sent.

Since the total number of 1ns was 5

****

**Example of Data word being sent and checking parity of a word**

In the scenario if the number of 1ns were (1110**00**10 X even number of 1ns) while sending, but due to noise after reaching receiver end it was (1110**11**10 X) it is wont throw error since the parity check the code has even number of 1ns and would treat it as a valid code

 but will be invalid code.

But Reciever cant find the exact location of the error in the Dataword. Thus the Receiver asks for retransmission of data .

Data word : 10101

* If we are checking using Even parity Bit checker
* We shall add 1 to the end of the Data word making it Even number of 1ns sent to Reciever
* Parity check for even parity at receivers end If error found Parity checker asks to retransmit the data and vice versa for odd parity

**Implementation :**

**Implementation procedure :**

Counts the number of '1's in the frame (using count('1')).

Checks if the number of '1's is even. If it is, it prints a message stating that the frame is an even parity frame.

Continues to the next iteration.

Counts the number of '1's in the corresponding received frame (using received\_frames[i].count('1')).

Checks if the number of '1's in the received frame is even. If it is, it prints a message stating that an even parity frame was received.

Continues to the next iteration.

**Code :**

**Code contains Parity functions with other functions as well used by parity checker**

**PSEUDO CODE**

def Parity\_checker(frames,received\_frames):

  for i in frames,received\_frame:

    count\_ones1 = frames.count('1')

    if count\_ones1 % 2 == 0:

      print("{frames} It is an even Parity Frame which was sent ")

    continue

    count\_ones2 = received\_frames[i].count('1')

    if count\_ones2 % 2 == 0:

      print("It is an even Parity Frame Received ")

    continue

    if frames[i] == received\_frame[i]:

      print("Frames matched !")

    else :

      print("Frame at receiver end Corrupted ")

Return

def check\_parity(frame):

    count\_ones = frame.count('1')

    is\_even\_parity = count\_ones % 2 == 0

    frame\_type = "Even" if is\_even\_parity else "Odd"

    print

return frame\_type

def Error\_randomizer(frame, error\_rate, skip\_probability):

  if error\_rate == 0:

    return frame

  corrupted\_frame = ""

  for bit in frame:

    if random.random() < skip\_probability:

      continue

    if random.random() < error\_rate:

      corrupted\_frame += "0" if bit == "1" else "1"

    else:

      corrupted\_frame += bit

  if len(corrupted\_frame) < len(frame):

    corrupted\_frame += "0" \* (len(frame) - len(corrupted\_frame))

  return corrupted\_frame

def simulate\_frame\_corruption(frames, error\_rate, skip\_probability):

  corrupted\_frames = []

  for frame in frames:

    if random.random() < skip\_probability:

      corrupted\_frames.append(frame)

      continue

    corrupted\_frame = Error\_randomizer(frame, error\_rate, skip\_probability)

    corrupted\_frames.append(corrupted\_frame)

  return corrupted\_frames

def server():

    server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    port\_no = ('localhost', 8080)

    server\_socket.bind(port\_no)

    server\_socket.listen(1)

    print("Waiting for a connection...")

    connection, client\_address = server\_socket.accept()

    print("Connection accepted ! ")

    received\_data = []

    parity\_type\_holder2 = []

    while True:

        data = connection.recv(1024)

        if not data:

            break

        received\_frame = data.decode()

        received\_data.append(received\_frame)

    print(received\_data)

    print("for Receiver side ")

    print(" RECEIVER MAC : ",receiver\_MAC())

    for frame in received\_data:

        frame\_type = check\_parity(frame)

        parity\_type\_holder2.append(frame\_type)

    connection.close()

    server\_socket.close()

def client():

    client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    port\_no = ('localhost', 8080)

    client\_socket.connect(port\_no)

    string = input("\n\nEnter a message > > ")

    frame\_size = 32

    char\_bit = 4

    error\_rate = 1

    skip\_probability = 0.5

    frames = Frame\_creation(string,frame\_size,char\_bit)

    print(f"\nFrames divided into {frame\_size} bit frames \n",frames)

    parity\_type\_holder1 = []

    print("\n\nfor Sender side ")

    print(" SENDER MAC : ")

    sender\_MAC()

    print("Message sent > > ",string)

    for fr in frames:

        frame\_type = check\_parity(fr)

        parity\_type\_holder1.append(frame\_type)

*# with open("p1.txt", 'w') as file:*

*#   for i in parity\_type\_holder1:*

*#       file.write(i + '\n')*

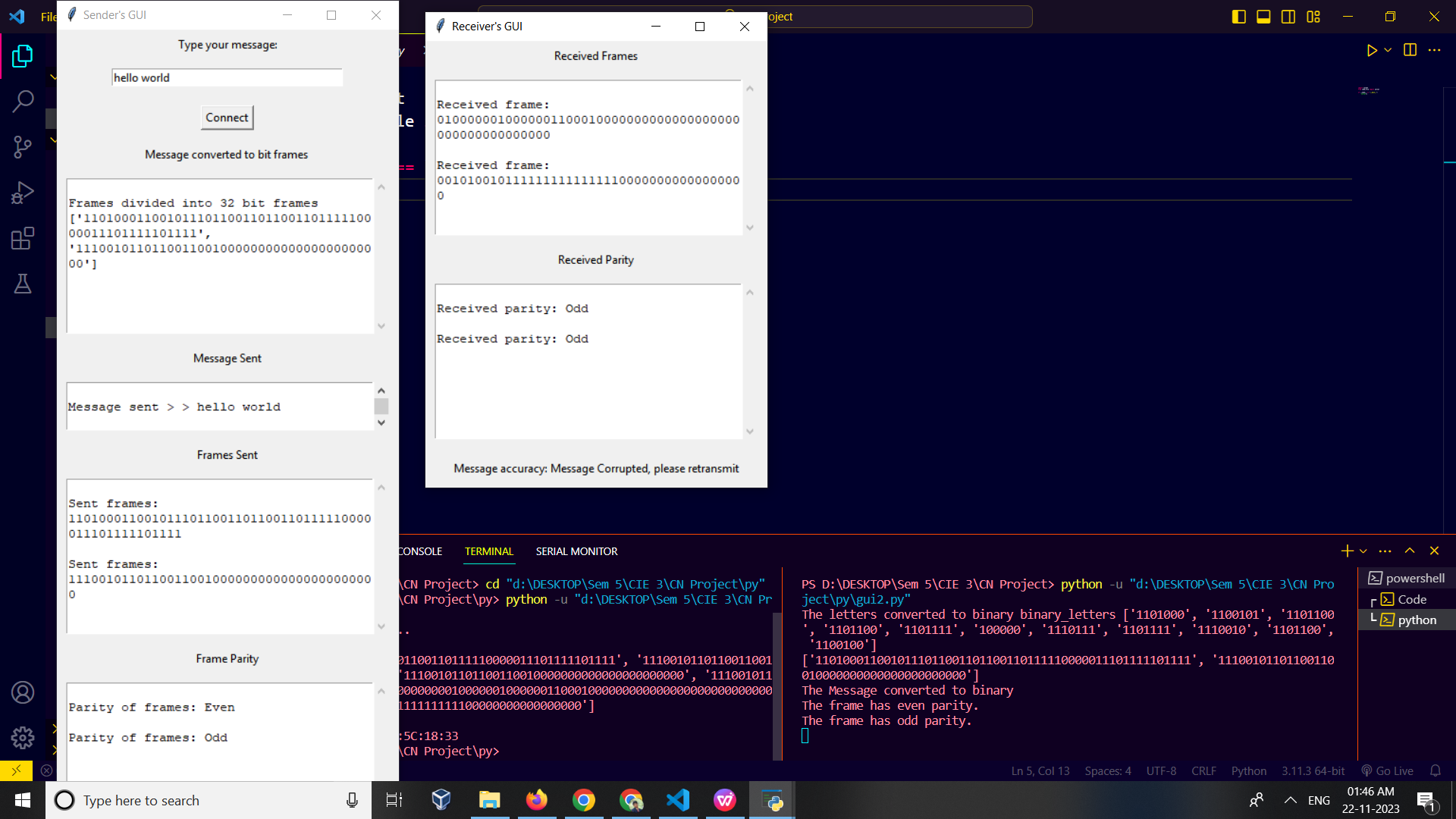
    C\_received\_frames = simulate\_frame\_corruption(frames, error\_rate, skip\_probability)

    for frame in C\_received\_frames:

        client\_socket.send(frame.encode())

    client\_socket.close()

**Output:**



**Conclusion :**

The overall conclusion of this is that it is designed to check the integrity of binary data frames by checking their parity and comparing them to their corresponding received frames.