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

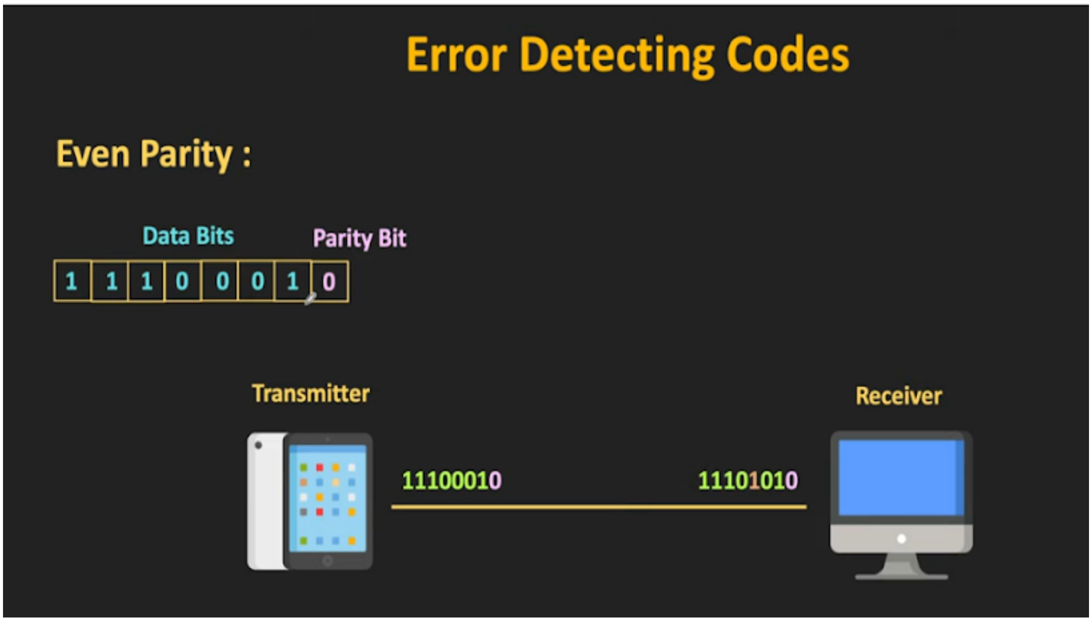
Thus from the Above example if code is sent to receiver from transmitter

Suppose

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

if Even number of 1ns were transmitted (11100010 X even number of 1ns) and due to some noise one of the 1ns turn zero ie (11101010 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


Error Detecting Codes

Even Parity :

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

In the scenario if the number of 1ns were (11100010 X even number of 1ns) while sending, but due to noise after reaching receiver end it was (11101110 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.

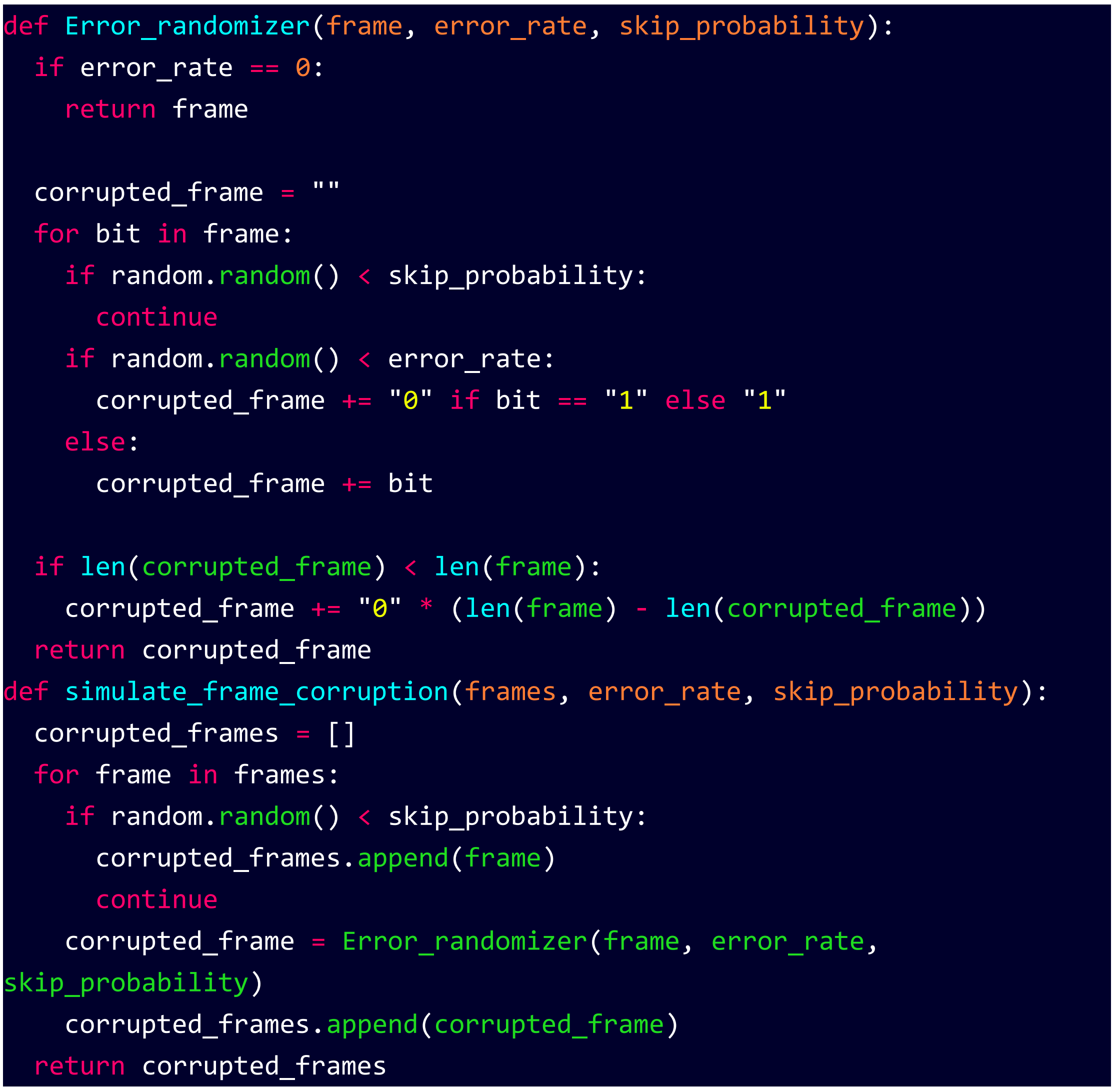
Continues to the next iteration.

Code :

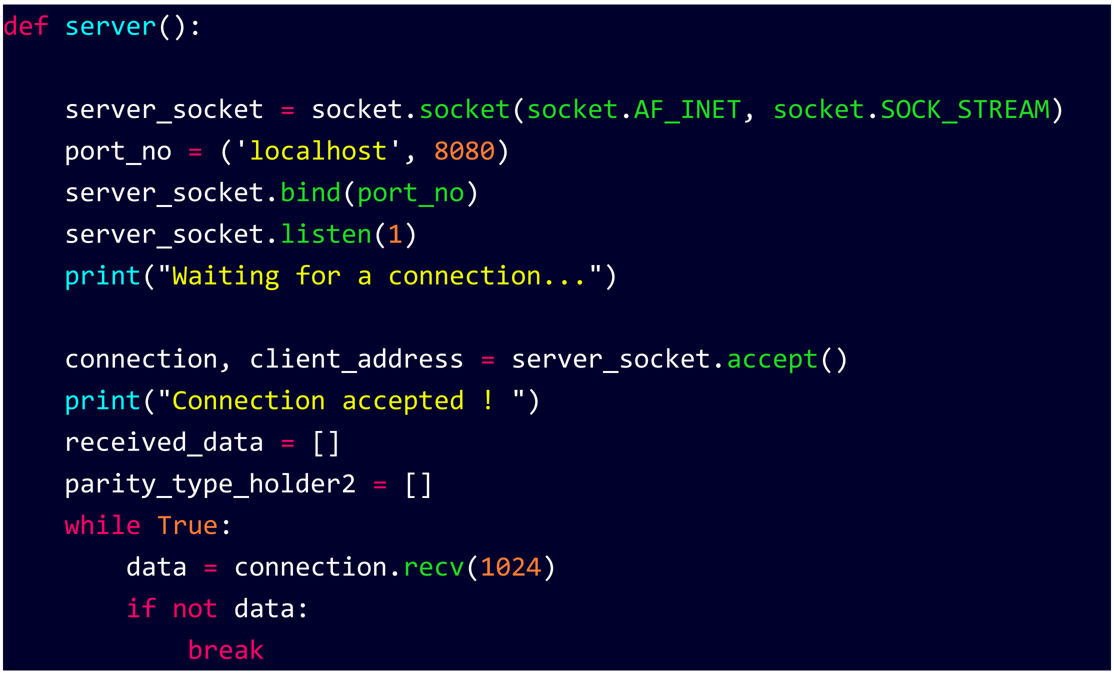
Code contains Parity functions with other functions as well used by parity checker

```
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
```

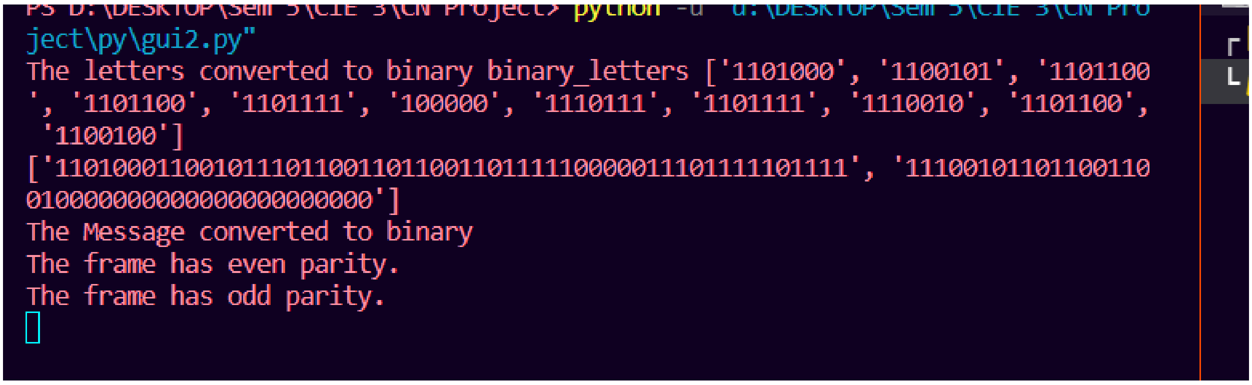


```python
def Error_randomizer(frame, error_rate, skip_probability):
  if error_rate == 0:
    return frame

  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
```



```python
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:



```
PS D:\DESKTOP\sem 3\CIE 3\CN Project> python -u "d:\DESKTOP\sem 3\CIE 3\CN Pro
ject\py\gui2.py"
The letters converted to binary binary_letters ['1101000', '1100101', '1101100
', '1101100', '1101111', '100000', '1110111', '1101111', '1110010', '1101100',
 '1100100']
['1101000110010111011001101100110111110000011101111101111', '11100101101100110
010000000000000000000000']
The Message converted to binary
The frame has even parity.
The frame has odd parity.
```

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.