**Homework – II**

**CSE 963: Computer Networks – I**

**Name: P. Sivachandran**

**Question – 1:**

Write a program (in any language) that generates an n-bit frame for transmission from a k-bit data block D and a (n – k + 1) bit CRC divisor P. Compile and run the program with atleast two set of inputs to confirm that this program is generating CRC patterns correctly.

Now, modify the program that performs the following steps:

1. Generates a message of k = 10 bits
2. Uses the previous code with P = 110101 to generate the corresponding 15-bit frame T for transmission
3. Generates transmission errors at any bit positions of T
4. Applies CRC to the received frame (i.e. frame T after introducing errors) to determine if the frame should be accepted or discarded

Attach your code with your report, and also put comments stating the instructions of how to run it.

**Answer:**

Prepared python code to generate an n-bit frame for transmission from the k-bit data block D and a (n – k + 1) bit CRC divisor P.

Please refer “Source\_Code\_1.ipynb” to get the source code prepared.

**Procedure to execute:** Please Open above file with “Anaconda – Jupyter” environment and then press “Run” button to execute the portion of program selected

**Referring (a) & (b) part of above question following inputs provided and respectively outputs are received:**

**Test input – 1** provided (k & P):

data = 1101001110

divisor = 110101

Output\_1 received:

Original Data: 1101001110

Transmitted Frame (Data + CRC): 110100111011011

Frame received correctly.

Please refer report “Exe\_report\_1\_1.html” to get more details

**Test input – 2** provided (k & P):

data = 1101000000

divisor = 110110

Output\_2 received:

Original Data: 1101000000

Transmitted Frame (Data + CRC): 110100000010000

Frame received correctly.

Please refer report “Exe\_report\_1\_2.html” to get more details

**Referring (c) & (d) part of above question following inputs provided and respectively outputs are received:**

**Test – 1,**

Selection of error bit as Random bit position, and found bit #12 introduced error

Original Data: 1101001110

Transmitted Frame (Data + CRC): 110100111011011

Received Frame after introducing error at position 12: 110100111011111

Error detected in received frame, so this frame should be discarded

Please refer report “Exe\_report\_1\_1.html” to get more details

Since error detected in the received frame, then I think it should be discarded

**Test – 2,**

Selection of error bit as Random bit position, and found bit #0 introduced error

Original Data: 1101001110

Transmitted Frame (Data + CRC): 110100111011011

Received Frame after introducing error at position 0: 010100111011011

Error detected in received frame, so this frame should be discarded

Please refer report “Exe\_report\_1\_2.html” to get more details

Since error detected in the received frame, then I think it should be discarded

Sharing below the source code as well just for reference,

(It also simulates error in the transmitted frame and applies CRC to the received frame to determine if the frame should be accepted or discarded)

**Step – 1:** Program to Basic CRC generator:

**# XOR operation between two strings**

def xor(a, b):

result = []

for i in range(1, len(b)):

result.append('0' if a[i] == b[i] else '1')

return ''.join(result)

**# Perform modulo-2 division and return remainder**

def mod2div(dividend, divisor):

pick = len(divisor)

tmp = dividend[0:pick]

while pick < len(dividend):

if tmp[0] == '1':

tmp = xor(divisor, tmp) + dividend[pick]

else:

tmp = xor('0' \* pick, tmp) + dividend[pick]

pick += 1

if tmp[0] == '1':

tmp = xor(divisor, tmp)

else:

tmp = xor('0' \* pick, tmp)

return tmp

def generate\_crc(data, divisor):

n = len(divisor) - 1

augmented\_data = data + '0' \* n

remainder = mod2div(augmented\_data, divisor)

crc = remainder

return data + crc # Return the complete frame (data + CRC)

def check\_crc(received\_frame, divisor):

remainder = mod2div(received\_frame, divisor)

if '1' in remainder:

return False # Error detected

return True # No error

**# Example - 10-bit data & CRC – 6 polynomial (divisor)**

data = "1101001110"

divisor = "110101"

print("Original Data: ", data)

transmitted\_frame = generate\_crc(data, divisor)

print("Transmitted Frame (Data + CRC): ", transmitted\_frame)

received\_frame = transmitted\_frame # Simulate no error

if check\_crc(received\_frame, divisor):

print("Frame received correctly.")

else:

print("Error detected in received frame, so this frame should be discarded")

**Step – 2:** Modify the Program to introduce errors and check CRC:

import random

**# Introduce an error at a specific bit position**

def introduce\_error(frame, error\_position):

error\_frame = list(frame)

error\_frame[error\_position] = '1' if frame[error\_position] == '0' else '0'

return ''.join(error\_frame)

**# Example – 10-bit data & CRC-6 polynomial (divisor)**

data = "1101001110"

divisor = "110101"

print("Original Data: ", data)

transmitted\_frame = generate\_crc(data, divisor)

print("Transmitted Frame (Data + CRC): ", transmitted\_frame)

**# Introduce error in the transmitted frame at a random position**

error\_position = random.randint(0, len(transmitted\_frame) - 1)

received\_frame = introduce\_error(transmitted\_frame, error\_position)

print(f"Received Frame after introducing error at position {error\_position}: ", received\_frame)

if check\_crc(received\_frame, divisor):

print("Frame received correctly.")

else:

print("Error detected in received frame, so this frame should be discarded")

**Question – 2:**

1. In a CRC error-detecting scheme, choose P(x)=X (pow 4)+X+1, Encode the bits 10010011011
2. Suppose the channel introduces an error pattern 100010000000000 (i.e., a flip from 1 to 0 or from 0 to 1 in position 1 and 5). What is received? Can the error be detected?
3. Repeat part (b) with error pattern 100110000000000

**Answer:**

1. **In a CRC error-detecting scheme, choose P(x)=X (pow 4) +X+1, Encode the bits 10010011011**

**Encode the bits using using CRC:**

Sharing below the step-wise approach to be followed to perform this task, and please refer source code ‘Source\_Code\_2.ipynb‘ to get furthermore details,

Step – 1: Determine the divisor P

The polynomial P(x) = X (pow 4) + X + 1, corresponds to the binary string ‘10011’

Step – 2: Appends zeros to the data bits:

Since the degree of the divisor P(x) is 4, we append 4 zeros to the data bits

Augmented date: ‘10010011011**0000**’

Step – 3: Perform modulo-2 divison:

Use the divisor ‘10011’ to divide the augmented data ‘100100110110000’ using modulo-2 division

Step – 4: Obtain the remainder and append to the original data

Append the remainder to the original data bits to form the encoded message

Sharing below the output received after executed source code:

Original Data: 10010011011

Encoded Message (Data + CRC): 100100110111100

Please refer “Exe\_report\_2\_1.html” to get some more details

1. **Suppose the channel introduces an error pattern 100010000000000 (i.e., a flip from 1 to 0 or from 0 to 1 in position 1 and 5). What is received? Can the error be detected?**

**Error Pattern 100010000000000**

Sharing below the step-wise approach to be followed to perform this task, and please refer source code ‘Source\_Code\_2.ipynb ‘ to get furthermore details,

Step – 1: Apply the error pattern to the encoded message:

Flip the bits in the encoded message at the positions where the error pattern has ‘1’

This results in the received message

Step – 2: Check if the error can be detected using the CRC

Perform modulo-2 divison on the received message using the divisor ‘10011’

If the remainder is non-zero, an error is detected. If the remainder is zero, no error is detected

Sharing below the output after executed source code:

Received Message after applying error pattern 100010000000000: 000110110111100

Error detected for error pattern 100010000000000.

Accordingly, **Error is detected** (with Error pattern 100010000000000),

Please refer “Exe\_report\_2\_1.html” to get some more details

**(c)Repeat part (b) with error pattern 100110000000000**

**Error pattern 100110000000000**

Sharing below the step-wise approach to be followed to perform this task, and please refer source code ‘Source\_Code\_2.ipynb‘ to get furthermore details,

Step – 1: Apply the error pattern to the encoded message:

Flip the bits in the encoded message at the positions where the error pattern has ‘1’

This results in the received message

Step – 2: Check if the error can be detected using the CRC

Perform modulo-2 division on the received message using the divisor ‘10011’

If the remainder is non-zero, an error is detected. If the remainder is zero, no error is detected

Sharing below the output after executed source code:

Received Message after applying error pattern 100110000000000: 000010110111100

No error detected for error pattern 100110000000000.

Accordingly, output is **No Error detected** (with Error pattern 100110000000000)

Please refer “Exe\_report\_2\_1.html” to get some more details,

**Brief explanation of functionalities used in source code:**

**Encoding the data:**

The function ‘generate\_crc( )’ generates the CRC code and appends it to the original data to form the encoded message

**Introducing Errors:**

The function ‘apply\_error( )’ applies an error pattern to the encoded message by flipping bits at the positions indicated by the error pattern

**Checking for Errors:**

The function ‘check\_crc( )’ performs modulo-2 division on the received message to check for errors

**Procedure to execute below source code:** Please Open above file with “Anaconda – Jupyter” environment and then press “Run” button to execute the portion of program selected

Copying below the source code here as well just for reference:

**# XOR operation between two strings**

def xor(a, b):

result = []

for i in range(1, len(b)):

result.append('0' if a[i] == b[i] else '1')

return ''.join(result)

**# Perform modulo-2 division and return remainder**

def mod2div(dividend, divisor):

pick = len(divisor)

tmp = dividend[0:pick]

while pick < len(dividend):

if tmp[0] == '1':

tmp = xor(divisor, tmp) + dividend[pick]

else:

tmp = xor('0' \* pick, tmp) + dividend[pick]

pick += 1

if tmp[0] == '1':

tmp = xor(divisor, tmp)

else:

tmp = xor('0' \* pick, tmp)

return tmp

def generate\_crc(data, divisor):

n = len(divisor) - 1

augmented\_data = data + '0' \* n

remainder = mod2div(augmented\_data, divisor)

crc = remainder

return data + crc **# Return the complete frame (data + CRC)**

**# Apply the error pattern to the frame**

def apply\_error(frame, error\_pattern):

error\_frame = list(frame)

for i in range(len(error\_pattern)):

if error\_pattern[i] == '1':

error\_frame[i] = '1' if frame[i] == '0' else '0'

return ''.join(error\_frame)

def check\_crc(received\_frame, divisor):

remainder = mod2div(received\_frame, divisor)

if '1' in remainder:

return False # Error detected

return True # No error

**# Part (a): Encoding**

data = "10010011011" **# Data bits**

divisor = "10011" **# CRC-4 polynomial**

print("Original Data: ", data)

encoded\_message = generate\_crc(data, divisor)

print("Encoded Message (Data + CRC): ", encoded\_message)

**# Part (b): Error Pattern 100010000000000**

error\_pattern\_b = "100010000000000"

received\_message\_b = apply\_error(encoded\_message, error\_pattern\_b)

print("Received Message after applying error pattern 100010000000000: ", received\_message\_b)

if check\_crc(received\_message\_b, divisor):

print("No error detected for error pattern 100010000000000.")

else:

print("Error detected for error pattern 100010000000000.")

**# Part (c): Error Pattern 100110000000000**

error\_pattern\_c = "100110000000000"

received\_message\_c = apply\_error(encoded\_message, error\_pattern\_c)

print("Received Message after applying error pattern 100110000000000: ", received\_message\_c)

if check\_crc(received\_message\_c, divisor):

print("No error detected for error pattern 100110000000000.")

else:

print("Error detected for error pattern 100110000000000.")