

**Batch: A1      Roll No.: 16010120015**

**Experiment / assignment / tutorial No.03**

**Grade: AA / AB / BB / BC / CC / CD / DD**

## Experiment No.:3

**TITLE:** Implementation of CRC & Checksum for Computer Networks

**AIM:** To implement Layer 2 Error Control schemes: CRC & Checksum.

### Expected Outcome of Experiment:

**CO:** Demonstrate Data Link Layer, MAC layer technologies & protocols and implement the functionalities like error control, flow control

### Books/ Journals/ Websites referred:

1. A. S. Tanenbaum, "Computer Networks", Pearson Education, Fourth Edition
2. B. A. Forouzan, "Data Communications and Networking", TMH, Fourth Edition

### Pre Lab/ Prior Concepts:

Data Link Layer, Error Correction/Detection, Types of Errors

### New Concepts to be learned: Checksum.

### CRC(Cyclic Redundancy Check):

Cyclic Redundancy Check (CRC) is another error detection technique to detect errors in data that has been transmitted on a communications link. A sending device applies a 16 or 32 bit polynomial to a block of data that is to be transmitted and appends the resulting cyclic redundancy check (CRC) to the block. The receiving end applies the same polynomial to the data and compares its result with the result appended by the sender. If they agree, the data has been received successfully. If not, the sender can be notified to resend the block of data.

### At Sender Side:

- Sender has a generator  $G(x)$  polynomial.
- Sender appends  $(n-1)$  zero bits to the data.

- Where,  $n$  = no of bits in generator
- Dividend appends the data with generator  $G(x)$  using modulo 2 division (arithmetic).
  - Remainder of  $(n-1)$  bits will be CRC.

**Codeword:** It is combined form of Data bits and CRC bits i.e. Codeword = Data bits + CRC bits.

### Example

Assume that –

(a) data is 10110.

(b) code generator is 1101.

(Code generator can also be mentioned in polynomial :  $x^3+x^2+1$  )

**Calculate CRC Bits:** While calculating the CRC bits, we pad  $(n-1)$  0's to the message bits, where 'n' = no of bits in the code generator.

Cyclic Redundancy check will be generated as shown below –

$$\begin{array}{r}
 1101 \overline{) 10110 \underline{000} ( 11001} \\
 \underline{1101} \phantom{000} \\
 1100 \phantom{000} \\
 \underline{1101} \phantom{000} \\
 0010 \phantom{000} \\
 \underline{0000} \phantom{000} \\
 0100 \phantom{000} \\
 \underline{0000} \phantom{000} \\
 1000 \phantom{000} \\
 \underline{1101} \phantom{000} \\
 101 \text{ (CRC Bit)}
 \end{array}$$

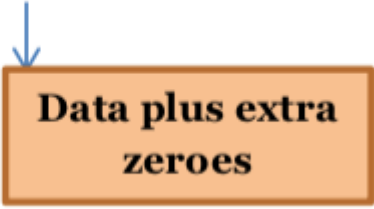


Figure 1: CRC calculation by sender

### At Receiver Side

- Receiver has same generator  $G(x)$ .
- Receiver divides received data (data + CRC) with generator.
- If remainder is zero, data is correctly received.

- Else, there is error.

Assume the received message is 10110110.

**Calculate CRC Bits:** It does not add any padding bits, rather calculates from the entire received code word.

$$\begin{array}{r}
 1101 \overline{) 10110110} \quad (11001 \\
 \underline{1101} \phantom{00000000} \\
 1100 \phantom{00000000} \\
 \underline{1101} \phantom{00000000} \\
 0011 \phantom{00000000} \\
 \underline{0000} \phantom{00000000} \\
 0110 \phantom{00000000} \\
 \underline{0000} \phantom{00000000} \\
 1100 \phantom{00000000} \\
 \underline{1101} \phantom{00000000} \\
 001 \phantom{00000000} \text{ (CRC Bit)}
 \end{array}$$

Figure 2: CRC calculation by receiver

The CRC bits are calculated to be different. Thus, there is an error detected.

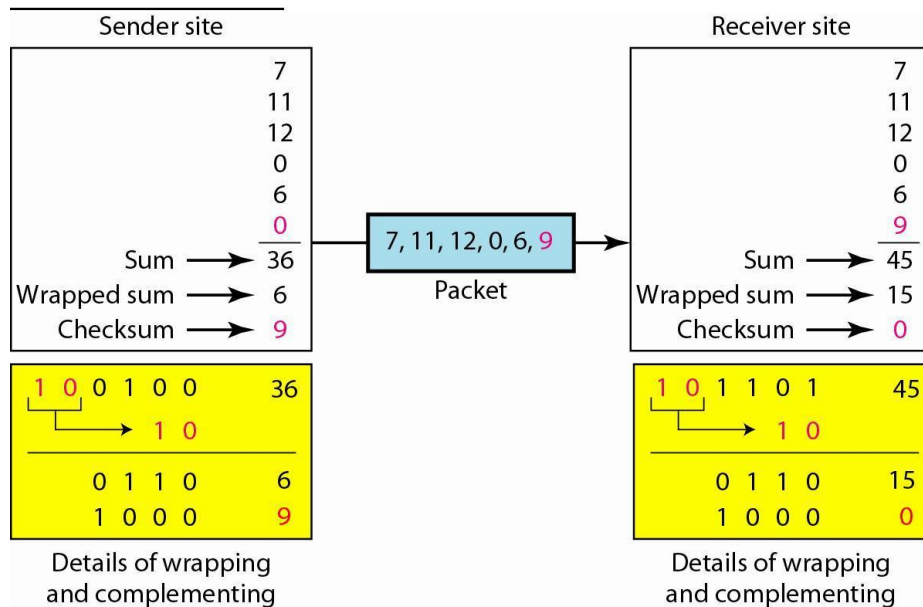
### Internet Checksum :

A checksum is a simple type of redundancy check that is used to detect errors in data.

Errors frequently occur in data when it is written to a disk, transmitted across a network or otherwise manipulated. The errors are typically very small, for example, a single incorrect bit, but even such small errors can greatly affect the quality of data, and even make it useless.

In its simplest form, a checksum is created by calculating the binary values in a packet or other block of data using some algorithm and storing the results with the data. When the data is retrieved from memory or received at the other end of a network, a new checksum is calculated and compared with the existing checksum. A non-match indicates an error; a match does not necessarily mean the absence of errors, but only that the simple algorithm was not able to detect any.

### Simple Checksum:



### Internet Checksum

The following process generates Internet Checksum

Assume the packet header is: 01 00 F2 03 F4 F5 F6 F7 00 00  
(00 00 is the checksum to be calculated)

The first step is to form 16-bit words.  
0100 F203 F4F5 F6F7

The second step is to calculate the sum using 32-bits.  
 $0100 + F203 + F4F5 + F6F7 = 0002 DEEF$

The third step is to add the carries (0002) to the 16-bit sum.  
 $DEEF + 002 = DEF1$

The fourth step is to take the complement. (1s becomes 0s and 0s become 1s)  
 $\sim DEF1 = 210E$

So the checksum is 21 0E.

The packet header is sent as: 01 00 F2 03 F4 F5 F6 F7 21 0E

\* At the receiver, the steps are repeated.

The first step is to form 16-bit words.  
0100 F203 F4F5 F6F7 210E

The second step is to calculate the sum using 32-bits.

$$0100 + F203 + F4F5 + F6F7 + 210E = 0002 \text{ FFFD}$$

The third step is to add the carries (0002) to the 16-bit sum.  
 $\text{FFFD} + 0002 = \text{FFFF}$  which means that no error was detected.

(In 1s complement, zero is 0000 or FFFF.)

**Example:**

1	0	1	3	Carries
4	6	6	F	(Fo)
7	2	6	7	(ro)
7	5	7	A	(uz)
6	1	6	E	(an)
0	0	0	0	Checksum (initial)
8	F	C	6	Sum (partial)
8	F	C	7	Sum
7	0	3	8	Checksum (to send)

a. Checksum at the sender site

1	0	1	3	Carries
4	6	6	F	(Fo)
7	2	6	7	(ro)
7	5	7	A	(uz)
6	1	6	E	(an)
7	0	3	8	Checksum (received)
F	F	F	E	Sum (partial)
8	F	C	7	Sum
0	0	0	0	Checksum (new)

a. Checksum at the receiver site

### IMPLEMENTATION: (printout of codes)

CRC:

```
class CRC:

    def __init__(self):
        self.cdw = ''

    def xor(self,a,b):
        result = []
        for i in range(1,len(b)):
            if a[i] == b[i]:
                result.append('0')
            else:
                result.append('1')

        return ''.join(result)

    def crc(self,message, key):
        pick = len(key)

        tmp = message[:pick]

        while pick < len(message):
            if tmp[0] == '1':
                tmp = self.xor(key,tmp)+message[pick]
            else:
                tmp = self.xor('0'*pick,tmp) + message[pick]

            pick+=1

        if tmp[0] == "1":
            tmp = self.xor(key,tmp)
        else:
            tmp = self.xor('0'*pick,tmp)

        checkword = tmp
        return checkword

    def EncodedData(self,data,key):
```

```
l_key = len(key)
append_data = data + '0'*(l_key-1)
remainder = self.crc(append_data,key)
codeword = data+remainder
self.cdw += codeword
print("Data Remainder is : " ,remainder)
print("Transmitted Value is : " ,codeword)

#have to edit this side
def ReciverSide(codeword,key,data):
    r = codeword.crc(data,key)
    size = len(key)
    print("remainder:", r)
    if r != 0:
        print("Transmitted Successfully ")
    else:
        print("Transmitted Error ")

def Divide(self,augdata,key):
    a = augdata
    b = key
    d = len(key)
    c = [0 for x in range(len(key))]

    for i in range(len(augdata)):
        if a[0]==1:
            ans = self.xor(a,b)
        else:
            ans = self.xor(a,c)
            ans.pop(0)
            if(len(augdata) == d):
                break
            else:
                ans.append(augdata[d])
                a = ans
                d += 1

    return ans

print('*****')
print('A1 16010120015 YASH ')
print('*****')
```

```
print('----- INPUT MESSAGE -----')

data = input("Input data : \n")
print('-----')
key = input("Input key : \n")

if len(key) > len(data):
    print("Invalid input")
else:
    parity = []
    for i in range(len(key)-1):
        parity.append(0)

    print('----- PARITY BITS -----')
    print("parity bits: ",parity)
    augdata = []
    augdata = str(data) + str(parity)
    print("Augmented dataword: ",augdata)

    print('----- SENDER SIDE -----')
    c = CRC()
    c.EncodedData(data,key)

    print('----- RECIEVER SIDE -----')
    c.ReceiverSide(c.cdw,key)

    print('-----')
    print("Reciver Get data: ",c.cdw)

    print('*****')
```

Output:



```
PS C:\Users\yashg\AppData\Local\Temp\Temp1_Sem5_Lab.zip\Sem5_Lab\ComputerNetworks\Codes\exp4\crc> & 'E:\SOFTWARES\PYTHON\python.exe' 'c:\Users\yashg\.vscode\extensions\ms-python.python-2022.16.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '55303' '--' 'c:\Users\yashg\AppData\Local\Temp\Temp1_Sem5_Lab.zip\Sem5_Lab\ComputerNetworks\Codes\exp4\crc\2.py'
*****
A1 16010120015 YASH
*****
----- INPUT MESSAGE -----
Input data :
10010011
-----
Input key :
111
----- PARITY BITS -----
parity bits: [0, 0]
Augmented dataword: 10010011[0, 0]
----- SENDER SIDE -----
Data Remainder is : 10
Transmitted Value is : 1001001110
----- RECIEVER SIDE -----
Reciver Get data: 1001001110
Correct codeword: 1001001110
█
```

### Internet Checksum:

```
import enum
class Hexadecimal(enum.Enum):
    a=10
    b=11
    c=12
    d=13
    e=14
    f=15
def summation(datablocks):
    sum = 0
    for i in datablocks: sum += int(i,16)
    sum = hex(sum)[2:]
    sum_length = len(sum)
    print(f"> Sum:{sum}")
    if sum_length > (step*2): slice_index = sum_length - (step*2)
    sum = hex(int(sum[int(slice_index):],16)+
int(sum[:int(slice_index)],16))[2:]
    print(f"> Final sum: {sum}")
    return sum
def complement(sum):
    complement = []
    for letter in sum:
        if letter.isnumeric():
            value = 15 - int(letter)
        else:
            value = 15 - Hexadecimal[letter].value
    if value >= 10:
        complement.append(Hexadecimal(value).name)
```

```

else:
    complement.append(value)
return complement
dataword = input("\n>>Enter the word to be sent: ")
dataword_length = len(dataword)
if dataword_length%2 != 0:
    dataword += ' '
    dataword_length = len(dataword)

print('*****')
print('A1 16010120015 YASH ')
print('*****')

print('\n-----Sender Site-----\n')
# Step 1: Breaking message into k number of blocks with n bits in each one
i = 2
while True:
    if dataword_length%i == 0:
        # print(f"{i}")
        break
    i += 1
step = i
datablock_length = dataword_length / i
print(f">> There are {datablock_length} Data Blocks having {step} Bits in each block.\n")
# Step 2: Summation of all datablocks
datablocks = []
for index in range(0,dataword_length,step):
    datablocks.append(dataword[index:index+step])
print(f"> DataBlocks: {datablocks}")
datablocks_hex = []
for datablock in datablocks:
    hexa_equi = []
    for letter in datablock: hexa_equi.append(hex(ord(letter))[2:])
    datablocks_hex.append(''.join(hexa_equi))
print(f"> Hexadecimal values of Datablocks: {datablocks_hex}")
sum = summation(datablocks_hex)
checksum = complement(sum)
print(f"> CheckSum at Sender Site: {''.join(map(str,checksum))}")
datablocks_hex.append("".join(map(str,checksum)))

```

```
print(f"> CheckSum appended to DataWord, sent to the receiver: {datablocks_hex}")
print('\n-----Receiver End-----\n')
print(f"> Received DataWord: {datablocks_hex}")
sum = summation(datablocks_hex)
checksum = complement(sum)
print(f"> CheckSum at Receiver End: {''.join(map(str,checksum))}")
print('\n-----Result-----\n')
if all(v==0 for v in checksum):
    print(">>There was no Error Found!")
else:
    print(">>Error Found!")
```

#### Output:

```
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\yashg\AppData\Local\Temp\Temp1_Sem5_Lab.zip\Sem5_Lab\ComputerNetworks\Codes\exp4\crc> & 'E:\SOFTWARES\Python\ms-python.python-2022.16.1\pythonFiles\lib\python\debugpy\adapter\..\debugpy\launcher' '65182' '--' 'c:\Users\yashg\AppData\Local\Temp\Temp1_Sem5_Lab.zip\Sem5_Lab\ComputerNetworks\Codes\exp4\crc\INCHCK.py'

>>Enter the word to be sent: 101FADE101
>> There are 5.0 Data Blocks having 2 Bits in each block.

> DataBlocks: ['10', '1F', 'AD', 'E1', '01']
> Hexadecimal values of Datablocks: ['3130', '3146', '4144', '4531', '3031']
> Sum:1191c
> Final sum: 191d
> CheckSum at Sender Site: 2
> CheckSum appended to DataWord, sent to the receiver: ['3130', '3146', '4144', '4531', '3031', '2']

-----Receiver End-----

> Received DataWord: ['3130', '3146', '4144', '4531', '3031', '2']
> Sum:1191e
> Final sum: 191f
> CheckSum at Receiver End: 0

-----Result-----

>>There was no Error Found!
PS C:\Users\yashg\AppData\Local\Temp\Temp1_Sem5_Lab.zip\Sem5_Lab\ComputerNetworks\Codes\exp4\crc> |
```

**CONCLUSION:** In this experiment we have successfully Implemented the concept of layer 2 error control schemes (CRC) and internet checksum which are used to check for errors in Data

### **Post Lab Questions**

1. Discuss about the rules for choosing a CRC generator.

**ANS :**

CRC generator is an algebraic polynomial represented as a bit pattern.

- Bit pattern is obtained from the CRC generator using the following rules are -

#### **Rule-01:**

- It should not be divisible by  $x$ .
- This condition guarantees that all the burst errors of length equal to the length of polynomial are detected.

#### **Rule-02:**

- It should be divisible by  $x+1$ .
- This condition guarantees that all the burst errors affecting an odd number of bits are detected.

2. State the advantages and disadvantages of Internet Checksum.

Ans.

#### **Advantages:**

- The checksum detects all the errors involving an odd number of bits as well as the error involving an even number of bits.

#### **Disadvantages:**

- The main problem is that the error goes undetected if one or more bits of a subunit is damaged and the corresponding bit or bits of a subunit are damaged and the corresponding bit or bits of opposite value in second subunit are also damaged. This is because the sum of those columns remains unchanged.

**Date : 13/09/2022**

**Signature of Faculty In-charge**