

**Experiment No.:4**

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

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

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

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**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 –

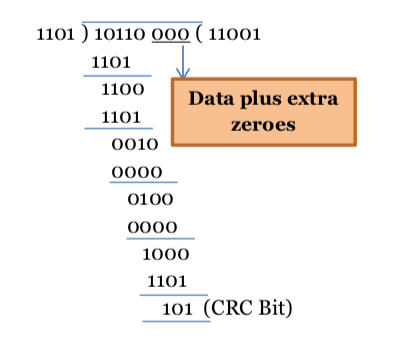
(a) data is 10110.

(b) code generator is 1101.

(Code generator can also be mentioned in polynomial : x3+x2+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 –



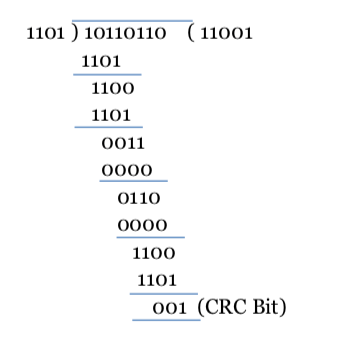
**Figure 1: CRC calculation by sender**

**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.



**Figure 2: CRC calculation by receiver**

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

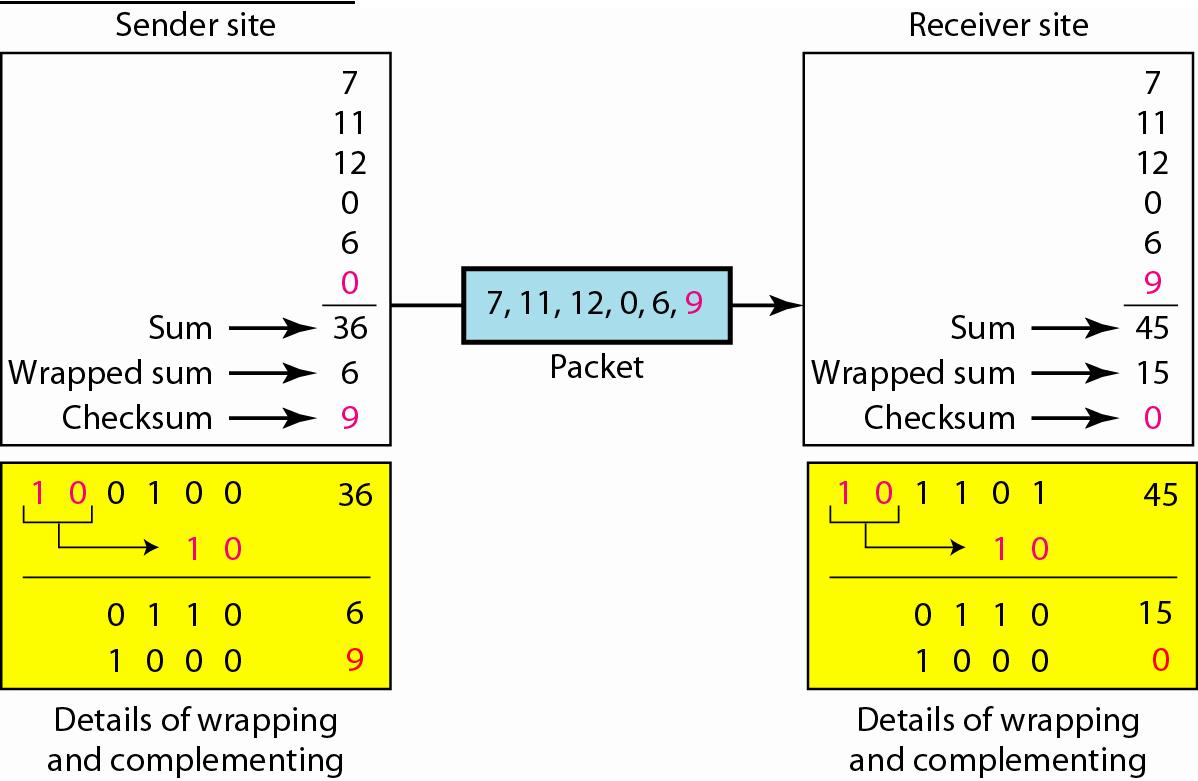
**Internet Checksum :**

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)

~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.

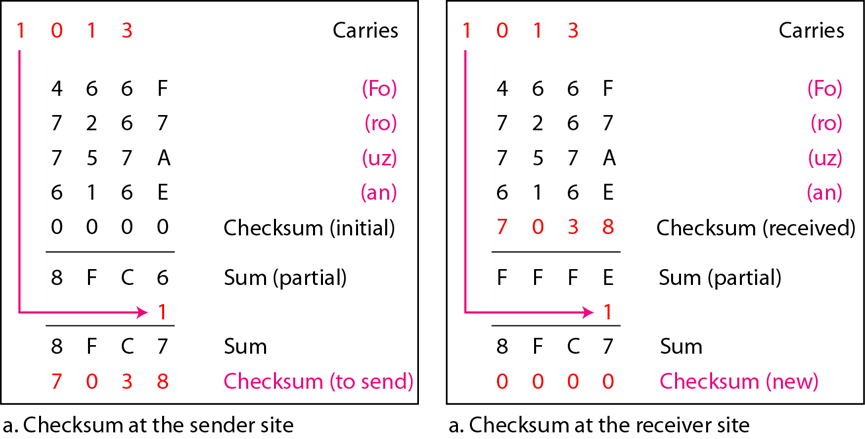
0100 + F203 + F4F5 + F6F7 + 210E = 0002 FFFD

The third step is to add the carries (0002) to the 16-bit sum.

FFFD + 0002 = FFFF which means that no error was detected.

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

**Example:**



**IMPLEMENTATION:** (printout of codes)

**1) CRC:**

**def xor(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 mod2div(dividend, divisor):**

**flag = len(divisor)**

**tmp = dividend[0: flag]**

**while flag < len(dividend):**

**if tmp[0] == '1':**

**tmp = xor(divisor, tmp) + dividend[flag]**

**else:**

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

**flag += 1**

**if tmp[0] == '1':**

**tmp = xor(divisor, tmp)**

**else:**

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

**return tmp**

**def encodeData(data, key):**

**l\_key = len(key)**

**appended\_data = data + '0'\*(l\_key-1)**

**remainder = mod2div(appended\_data, key)**

**codeword = data + remainder**

**return codeword**

**def checkError(received\_data, key):**

**remainder = mod2div(received\_data, key)**

**return remainder, remainder == '0' \* (len(key) - 1)**

**def binary\_to\_int(binary\_str):**

**return int(binary\_str, 2) if binary\_str else 0**

**if \_\_name\_\_ == "\_\_main\_\_":**

**print("Sender's side : ")**

**data = input("Enter the data: ")**

**key = input("Enter the key: ")**

**encoded\_data = encodeData(data, key)**

**print("Initial Dataword (Dividend) : " ,(data+'000'))**

**print("Encoded Data (Data + Remainder) of sender :", encoded\_data)**

**print(" ")**

**print("Receiver's side : ")**

**received\_data = input("Enter the received data: ")**

**remainder, is\_check= checkError(received\_data, key)**

**print("Remainder of received data :", remainder)**

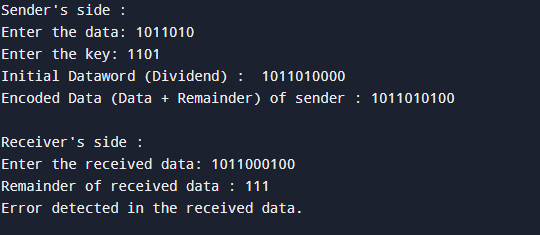
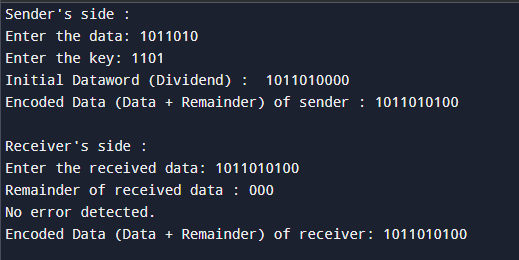
**if is\_check:**

**print("No error detected.")**

**print("Encoded Data (Data + Remainder) of receiver:", encoded\_data)**

**else:**

**print("Error detected in the received data.")**

**  
**

**2) Internet Check Sum :**

def calculate\_checksum(data):

words = [int(data[i:i+4], 16) for i in range(0, len(data), 4)]

sum\_value = sum(words)

sum\_value = (sum\_value & 0xFFFF) + (sum\_value >> 16)

checksum = ~sum\_value & 0xFFFF

return format(checksum, '04X')

def verify\_checksum(data):

words = [int(data[i:i+4], 16) for i in range(0, len(data), 4)]

sum\_value = sum(words)

sum\_value = (sum\_value & 0xFFFF) + (sum\_value >> 16)

return (sum\_value & 0xFFFF) == 0xFFFF

def print\_words(data):

print("16-bit Words:")

for i in range(0, len(data), 4):

print(data[i:i+4])

def main():

print("Sender's side:")

data = input("Enter the hexadecimal data (without spaces): ").strip().upper()

if len(data) % 4 != 0:

data = data.zfill(len(data) + (4 - len(data) % 4))

print\_words(data)

sum\_value = sum(int(data[i:i+4], 16) for i in range(0, len(data), 4))

print(f"Sum of 16-bit words: {format(sum\_value, '04X')}")

sum\_value = (sum\_value & 0xFFFF) + (sum\_value >> 16)

print(f"Adjusted Sum after adding carry: {format(sum\_value, '04X')}")

checksum = calculate\_checksum(data)

print(f"Calculated Checksum: {checksum}")

full\_packet = data + checksum

print(f"\nFull Packet: {full\_packet}")

print("\nReceiver's side:")

received\_packet = input("Enter the received packet (without spaces): ").strip().upper()

print\_words(received\_packet)

sum\_value = sum(int(received\_packet[i:i+4], 16) for i in range(0, len(received\_packet), 4))

sum\_value = (sum\_value & 0xFFFF) + (sum\_value >> 16)

final\_result = ~sum\_value & 0xFFFF

print(f"1's Complement: {format(final\_result, '04X')}")

if verify\_checksum(received\_packet):

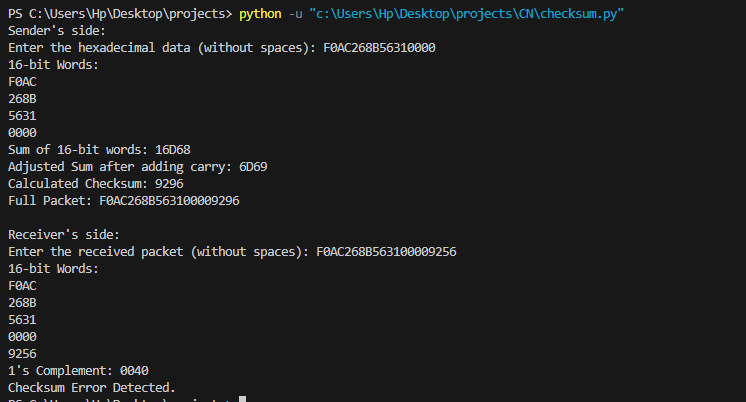
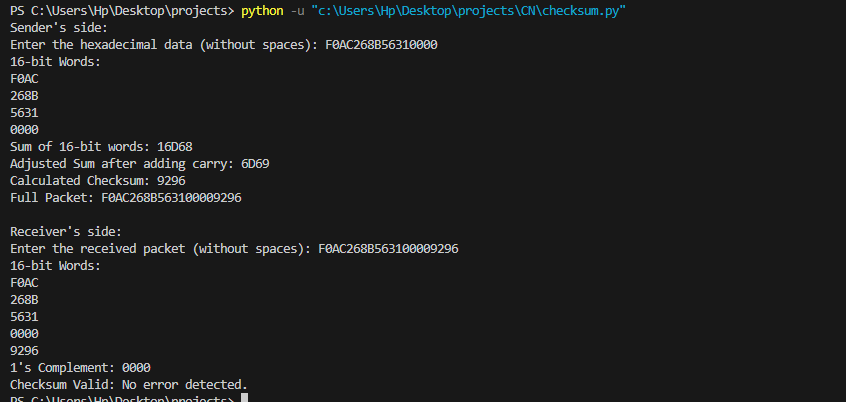
print("Checksum Valid: No error detected.")

else:

print("Checksum Error Detected.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**CONCLUSION:**

Learned CRC(Cyclic Redundancy Check) and Internet Check Sum error algorithms.

**Post Lab Questions**

1. Discuss about the rules for choosing a CRC generator.  
      
   1. It should have at least two terms.
   2. The coefficient of the term x0 should be 1.
   3. It should not be divisible by x.
   4. It should be divisible by x+ 1.
   5. There are several different standard polynomials used by popular protocols for CRC generation.

1. State the advantages and disadvantages of Internet Checksum.  
     
   **Advantages:**

* Easy to implement and understand.
* Fast to compute, especially in software.
* Small checksum size relative to the data.

**Disadvantages:**

* Less robust than CRCs may not catch all types of errors (e.g., multiple-bit errors).
* Only detects errors but cannot correct them.
* May not meet the robustness requirements for high-integrity data communications.

**Date : 26/8/24 Signature of Faculty In-charge**