**Batch: D - 1 Roll No.: 16010122096**

**Experiment / assignment / tutorial No. 04**

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

**Signature of the Staff In-charge with date**

**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:**

To learn and implement CRC & checksum error checking algorithms.

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

**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 –

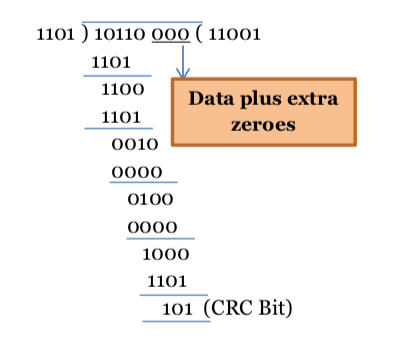


Figure 1: CRC calculation by sender

**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.

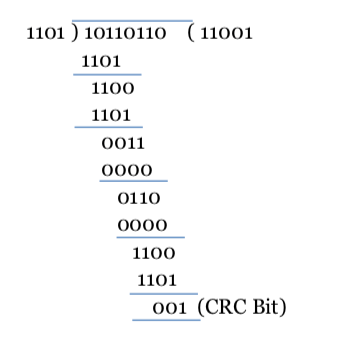


Figure 2: CRC calculation by receiver

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

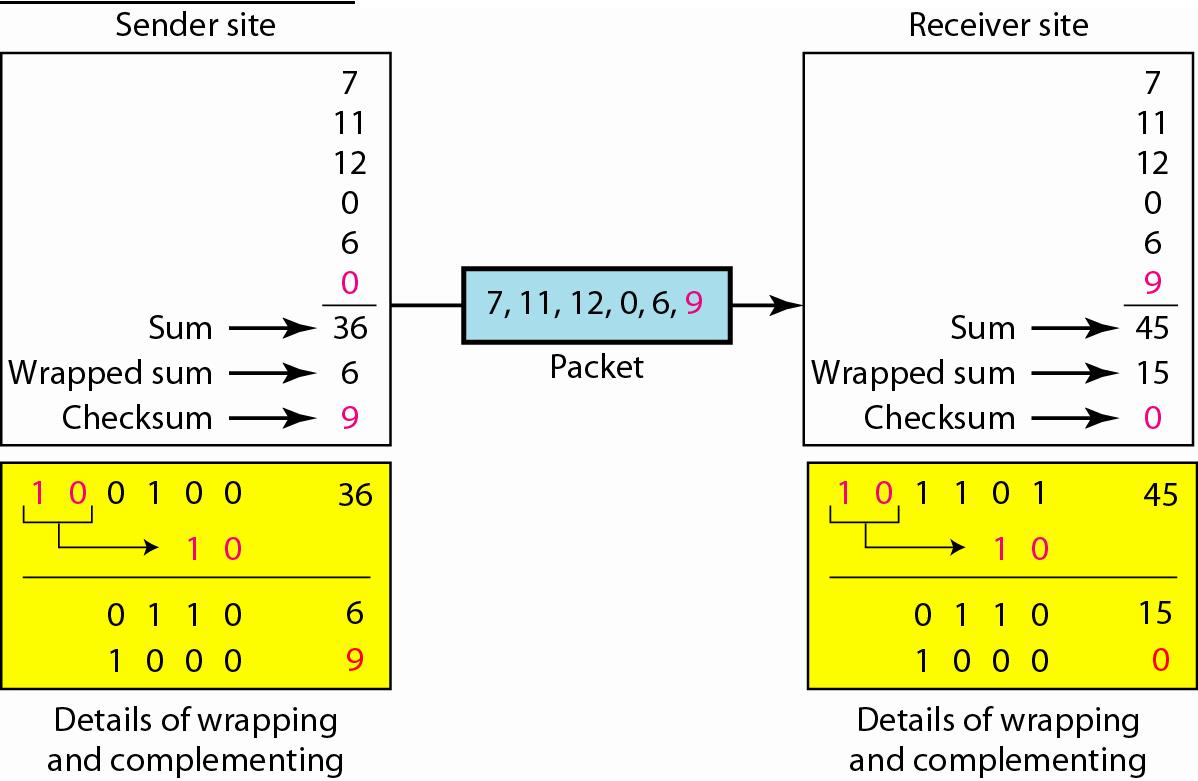
**Internet Checksum :**

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.

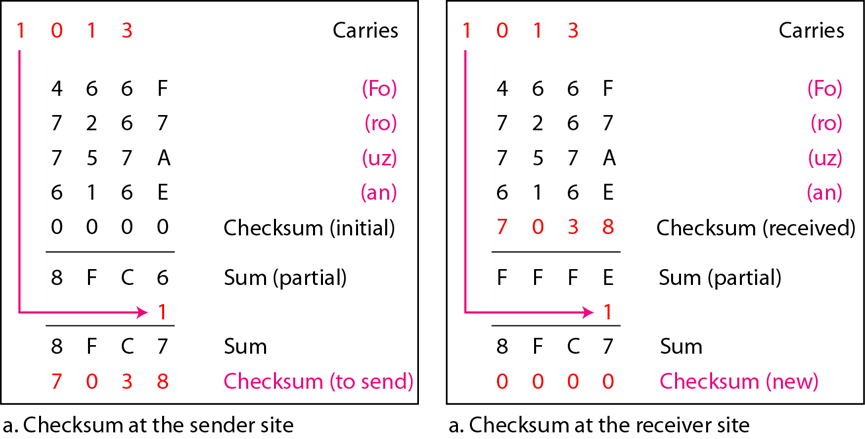
0100 + F203 + F4F5 + F6F7 + 210E = 0002 FFFD

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:**

**A] CRC:**

#include <iostream>

#include <string>

using namespace std;

string appendRedBits(const string &data, int g\_size)

{

   return data + string(g\_size - 1, '0');

}

string xor1(const string &dividend, const string &g)

{

   string result;

   for (int i = 0; i < g.size(); ++i)

   {

      result += (dividend[i] == g[i]) ? '0' : '1';

   }

   return result;

}

string findSyndrome(string dividend, const string &g)

{

   int g\_size = g.size();

   for (int i = 0; i <= dividend.size() - g\_size; ++i)

   {

      if (dividend[i] == '1')

      {

         string part = dividend.substr(i, g\_size);

         string res = xor1(part, g);

         dividend.replace(i, g\_size, res);

      }

   }

   return dividend.substr(dividend.size() - g\_size + 1);

}

string crc(const string &data, const string &g)

{

   int g\_size = g.size();

   string dividend = appendRedBits(data, g\_size);

   cout << "Initial Codeword: " << dividend << endl;

   string syndrome = findSyndrome(dividend, g);

   cout << "Remainder: " << syndrome << endl;

   return data + syndrome;

}

string introduceError(string c, int a)

{

   if (a < 1 || a > c.size())

   {

      cerr << "Index out of bounds!" << endl;

      return c;

   }

   c[a - 1] = (c[a - 1] == '1') ? '0' : '1';

   return c;

}

void checkError(const string &codeword, const string &generator)

{

   cout << "Do you want to introduce an error and check (enter 1) or not (enter -1): ";

   int input, a;

   cin >> input;

   string errorcode = codeword;

   string rem = "";

   switch (input)

   {

   case -1:

      cout << "No Error" << endl;

      cout << "Dataword: " + codeword.substr(0, codeword.size() - generator.size() + 1) << endl;

      break;

   case 1:

      cout << "Enter the index (1-based) to make the codeword erroneous: ";

      cin >> a;

      errorcode = introduceError(errorcode, a);

      rem = findSyndrome(errorcode, generator);

      if (rem.find('1') != string::npos)

      {

         cout << "Codeword with error: " << errorcode << endl;

         cout << "The remainder after the error is: " << rem << endl;

         cout << "Error Detected!" << endl;

      }

      else

      {

         cout << "No Error" << endl;

      }

      break;

   default:

      cout << "Invalid input!" << endl;

      break;

   }

}

int main()

{

   string dataWord, generator;

   cout << "Enter the dataword and generator in binary (without spaces): " << "\n";

   cin >> dataWord >> generator;

   if (generator.find\_first\_not\_of("01") != string::npos)

   {

      cout << "Generator must be in binary format!" << endl;

      return 1;

   }

   if (dataWord.find\_first\_not\_of("01") != string::npos)

   {

      cout << "Dataword must be in binary format!" << endl;

      return 1;

   }

   string codeword = crc(dataWord, generator);

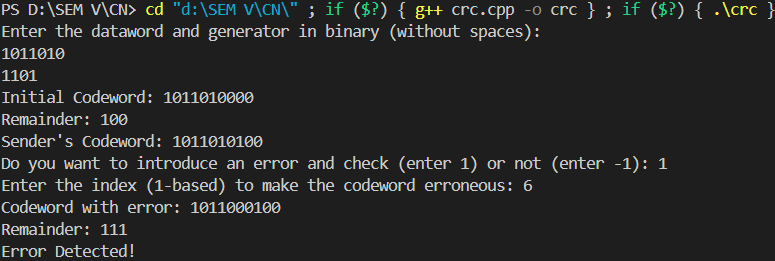
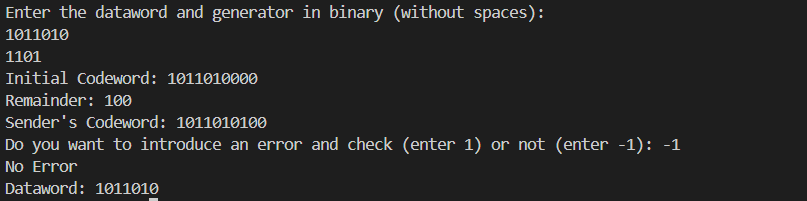
   cout << "Codeword: " << codeword << endl;

   checkError(codeword, generator);

   return 0;

}

**Output:**



**B] CHECKSUM**

#include <bits/stdc++.h>

using namespace std;

vector<string> packets;

string stringToBinary(const string &dw)

{

   string binaryWord = "";

   for (char c : dw)

   {

      binaryWord += bitset<8>(c).to\_string();

   }

   return binaryWord;

}

string addRedBits(string bdw)

{

   string redBits = "00000000";

   bdw += redBits;

   return bdw;

}

vector<string> formPackets(string &bdw, const string &dw)

{

   int noOfChars = dw.length();

   if (noOfChars & 1)

      bdw = addRedBits(bdw);

   int noOfPackets = ceil(bdw.length() / 16.0);

   for (int i = 0; i < noOfPackets; i++)

   {

      string packet = bdw.substr(i \* 16, 16);

      packets.push\_back(packet);

   }

   return packets;

}

string addingBinaryPackets(string a, string b)

{

   string ans = "";

   char carry = '0';

   for (int i = 15; i >= 0; --i)

   {

      char sum = a[i] ^ b[i] ^ carry;

      ans = sum + ans;

      carry = (a[i] & b[i]) | (a[i] & carry) | (b[i] & carry);

   }

   if (carry == '1')

      ans = carry + ans;

   return ans;

}

string addPackets(vector<string> &packets)

{

   string result = packets[0];

   for (int i = 1; i < packets.size(); i++)

   {

      result = addingBinaryPackets(result, packets[i]);

   }

   for (char &c : result)

      c = (c == '1') ? '0' : '1';

   return result;

}

string checkSum(const string &dataWord)

{

   string binaryDataWord;

   binaryDataWord = stringToBinary(dataWord);

   vector<string> packets = formPackets(binaryDataWord, dataWord);

   cout << "The transmitted data is: " + binaryDataWord << endl;

   return addPackets(packets);

}

string checkError(const string &checksum)

{

   string result = packets[0];

   for (int i = 1; i < packets.size(); i++)

   {

      result = addingBinaryPackets(result, packets[i]);

   }

   result = addingBinaryPackets(result, checksum);

   for (char &c : result)

      c = (c == '1') ? '0' : '1';

   return result;

}

int main()

{

   string dataWord;

   cout << "Enter the dataword input (text) without any spaces: " << endl;

   cin >> dataWord;

   string checksum = checkSum(dataWord);

   cout << "Checksum: " << checksum << endl;

   string errorCode = checkError(checksum);

   if (errorCode == "0000000000000000")

      cout << "No error in transmission, result after complement is " + errorCode << endl;

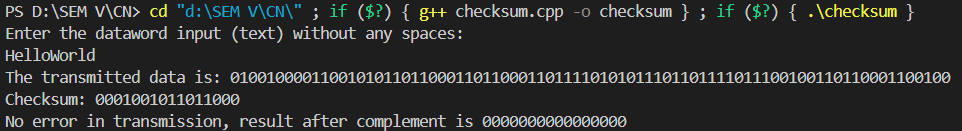
   else

      cout << "Error in transmission detected, erroneous code is " + errorCode << endl;

   return 0;

}

**Output:**



**CONCLUSION:** Implemented CRC and Checksum algorithms, successfully detecting and correcting transmission errors at the Data Link Layer.

**Post Lab Questions**

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

When choosing a CRC (Cyclic Redundancy Check) generator polynomial, certain rules and guidelines ensure that the CRC is effective in detecting errors:

* **Polynomial Length**: The length of the generator polynomial (degree + 1) determines the number of bits in the CRC. A longer polynomial provides better error detection.
* **Non-trivial**: The polynomial should be non-trivial (not all zeros or all ones) to ensure it can detect a variety of errors.
* **Highest Degree Term**: The generator polynomial should include the highest degree term (e.g., xnx^nxn) to ensure it can detect burst errors.
* **Lowest Degree Term**: The polynomial should include the constant term (i.e., 1) to detect errors involving bits in the middle of the message.
* **Primitive Polynomial**: A primitive polynomial of degree nnn generates a maximum-length sequence and can detect all single-bit errors, double-bit errors, and odd numbers of errors.
* **Well-known Standards**: Many standardized polynomials, such as CRC-32, have been thoroughly tested and are widely used. Using these can simplify implementation and increase reliability.

1. State the advantages and disadvantages of Internet Checksum.

**Advantages:**

* **Simplicity:** The Internet checksum algorithm is simple to implement, involving only basic addition and bit manipulation, making it computationally inexpensive.
* **Speed:** Due to its simplicity, the Internet checksum can be computed very quickly, which is beneficial for real-time applications like packet switching and network routing.
* **Compatibility:** It is widely supported and used in many legacy systems, ensuring broad compatibility across different platforms.

**Disadvantages:**

* **Limited Error Detection:** The Internet checksum is not as robust as other error detection methods like CRC. It may fail to detect some types of errors, such as those that cause an even number of bit inversions.
* **Less Effective for Longer Data:** As the length of the data increases, the likelihood of undetected errors also increases, making it less effective for larger messages.
* **Vulnerable to Specific Error Patterns:** The Internet checksum is particularly vulnerable to errors that are symmetric (e.g., swapping two bytes) or where the sum of changes equals zero.

**Date: 27 / 08 / 2024 Signature of Faculty In-charge**