

Batch: D-2 **Roll No.:** 16010122151

Experiment / assignment / tutorial No. _____

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:

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 –

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

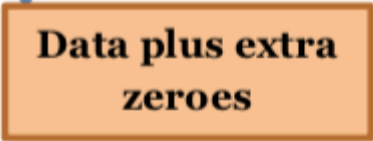


Figure 1: CRC calculation by sender

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.

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

Figure 2: CRC calculation by receiver

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

Internet Checksum :

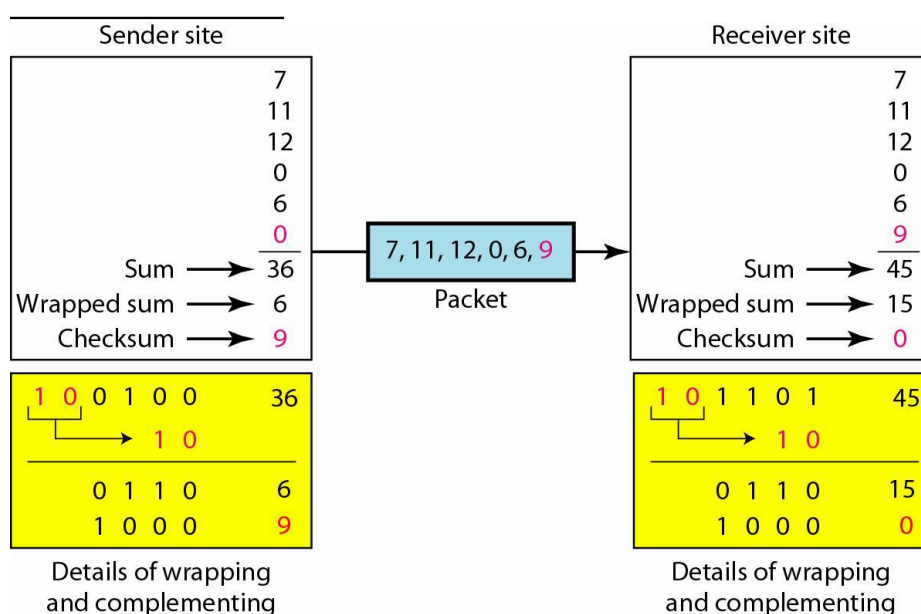
A checksum is a simple type of redundancy check that is used to detect errors in data.

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)
 $\sim \text{DEF1} = 210\text{E}$

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

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

Hydex Pascual

Date: / /

Rel no: 181

$= 145+1$

$= 146$

$g(x) = x^3 + 1x^2 + 1$

$d = 1110$

$d(x) = x^3 + x^2 + x$

$P(x) = \text{Rem} \int \frac{x^{145} d(x)}{g(x)} dx$

$= x \int \frac{x^{144} (x^3 + x^2 + x)}{x^3 + x^2 + x} dx$

$= x \int \frac{x^{144} (x^3 + x^2 + x)}{x^3 + x^2 + x} dx$

$= x \int x^{144} dx$

$= x \cdot \frac{x^{145}}{145} + C$

$= \frac{x^{146}}{145} + C$

$\therefore P(x) = \frac{x^{146}}{145} + C$

Date: 10/07/21

$$\begin{aligned}
 (x) &= x^{n-h} \cdot d(x) + p(x) \\
 &= x^{n-h} (x^3 + x^2 + x) + x \\
 &= x^7 + x^6 + x^5 + x^4 + x \\
 &= 1110010 \\
 &\quad x^6 \quad x^5 \quad x^4 \quad x^3 \quad x^2 \quad x^1 \quad x^0
 \end{aligned}$$

$\begin{array}{cc} 1110 & 010 \\ \downarrow & \downarrow \\ \text{Message} & \text{Redundant bit} \end{array}$

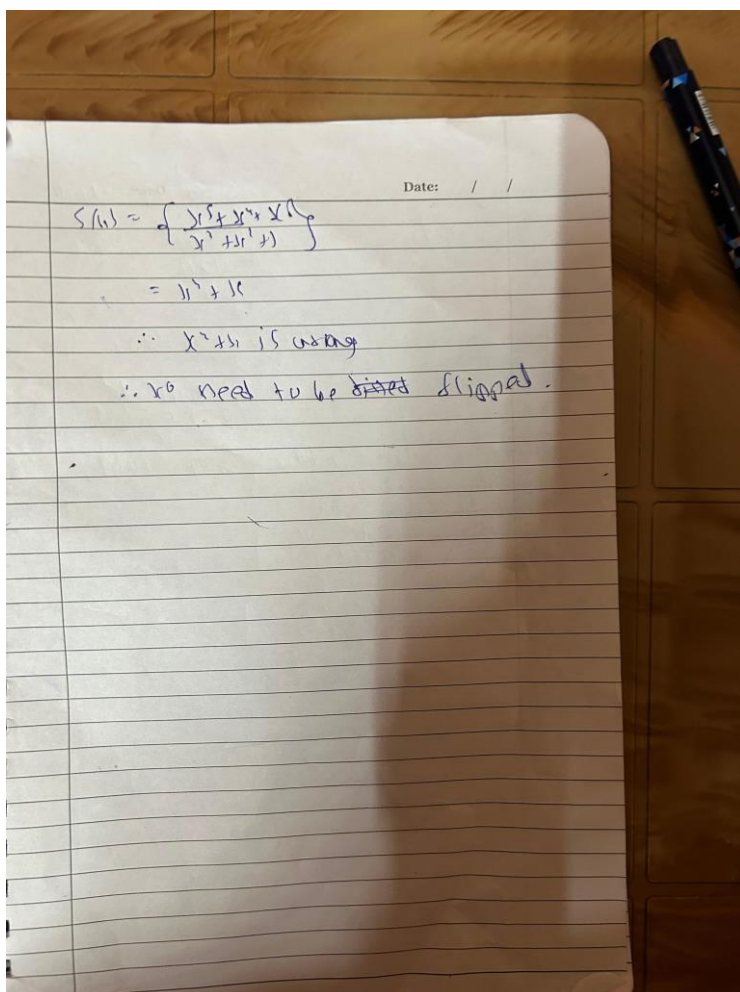
Received code add is 0110 010

$$R(x) = x^5 + x^4 + x$$

e	e(x)	s(x)	s
1000 000	x^6	$x^6 + x$	010
0100 000	x^5	x^5	011
0010 000	x^4	$x^4 + x$	101
0001 000	x^3	$x^3 + x$	100
0000 100	x^2	x^2	010
0000 010	x^1	x	001
0000 001	x^0	1	000

$$S(x) = \text{Rem} \left\{ \frac{e(x)}{g(x)} \right\}$$

$$\begin{array}{r}
 x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \\
 x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \\
 \hline
 0
 \end{array}$$



IMPLEMENTATION: (printout of codes)

```
#include <iostream>
using namespace std;

// Function to XOR two characters ('0' and '1')
char xors(char a, char b) {
    return (a == b) ? '0' : '1';
}

// Function to check if the first bit of the string is '0'
bool checkF(string s) {
    return s[0] == '0';
}

// Function to perform polynomial division (CRC)
string division(string div1, string div2) {
    string start = div1.substr(0, div2.size());
    string dividend = start;

    int k = div2.size();

    while (k <= div1.size()) {
        string ans = "";

        if (checkF(dividend)) {
            // If the first bit is '0', do XOR with all '0's
            for (int i = 0; i < div2.size(); i++) {
                ans += xors(dividend[i], '0');
            }
        } else {
            // Perform XOR with the divisor
            for (int i = 0; i < div2.size(); i++) {
                ans += xors(dividend[i], div2[i]);
            }
        }

        dividend = ans;

        // Shift the dividend
        for (int i = 0; i < dividend.size() - 1; i++) {
```

```

        dividend[i] = dividend[i + 1];
    }

    // Append the next bit of the dividend
    dividend[dividend.size() - 1] = div1[k];
    k++;
}

return dividend;
}

// Function to check if the remainder has no '1's (indicating no error)
bool isError(string s) {
    for (int i = 0; i < s.size(); i++) {
        if (s[i] != '0') {
            return false; // Error found
        }
    }
    return true; // No error
}

int main() {
    string input;
    cout << "Enter the data to send: ";
    cin >> input; // Input data (message)

    string key;
    cout << "Enter the CRC key: ";
    cin >> key; // CRC polynomial (key)

    int keyLen = key.size();

    // Append keyLen-1 zeros to the data for division
    string newDividend = input + string(keyLen - 1, '0');
    cout << "The Dividend in sender: " << newDividend << "\n";

    // Perform division to get the remainder
    string rem = division(newDividend, key);
    cout << "The Remainder is: " << rem << "\n";

    // Create the transmitted message by appending the remainder to the
    original data
    string receiver1 = input + rem;
    cout << "Transmitted message: " << receiver1 << "\n";
}

```



```
// Simulate the receiver's input
cout << "Enter what receiver has received: ";
string receiver;
cin >> receiver;

// Receiver checks the received message for errors
string rec = division(receiver + rem.substr(0, rem.size() - 1), key);
cout << "Remainder in receiver: " << rec << "\n";

// Check if there are any errors based on the remainder
if (isError(rec.substr(0, rec.size() - 1))) {
    cout << "No Errors Found\n";
} else {
    cout << "Errors Found\n";
}

return 0;
}
```

```
hyder@HyderPresswala MINGW64 ~/Downloads/Codewithharry
• $ g++ -o hyder hyder.cpp

hyder@HyderPresswala MINGW64 ~/Downloads/Codewithharry
• $ ./hyder.exe
Enter the data to send: 100100
Enter the CRC key: 1101
The Dividend in sender: 100100000
The Remainder is: 001
Transmitted message: 100100001
Enter what receiver has received: 100100
Remainder in receiver: 000
No Errors Found

hyder@HyderPresswala MINGW64 ~/Downloads/Codewithharry
• $ ./hyder.exe
Enter the data to send: 100100
Enter the CRC key: 1101
The Dividend in sender: 100100000
The Remainder is: 001
Transmitted message: 100100001
Enter what receiver has received: 111101
Remainder in receiver: 010
Errors Found

hyder@HyderPresswala MINGW64 ~/Downloads/Codewithharry
$
```

Checksum :-

```
#include <iostream>
#include <string>

using namespace std;

// Function to calculate checksum by summing byte values and applying modulo
256
unsigned char calculateChecksum(const string &data) {
    unsigned int sum = 0;

    // Sum the ASCII values of each character in the string
    for (char ch : data) {
        sum += static_cast<unsigned char>(ch);
    }

    // Return the checksum as the sum modulo 256 (1 byte value)
    return static_cast<unsigned char>(sum % 256);
}

int main() {
    string data;

    // Get input data from the user
```

```
cout << "Enter data to calculate checksum: ";
getline(cin, data);

// Calculate checksum
unsigned char checksum = calculateChecksum(data);

// Output the result
cout << "Checksum of the given data: " << (int)checksum << endl;

// Optionally: Simulate checking the data with checksum verification
string receivedData;
cout << "Enter received data to verify checksum: ";
getline(cin, receivedData);

unsigned char receivedChecksum = calculateChecksum(receivedData);

// Check if received checksum matches the expected checksum
if (receivedChecksum == checksum) {
    cout << "Data is correct. No errors detected." << endl;
} else {
    cout << "Error detected in received data." << endl;
}

return 0;
}
```

```
hyder@HyderPresswala MINGW64 ~/Downloads/Codewithharry
• $ g++ -o hyder hyder.cpp

hyder@HyderPresswala MINGW64 ~/Downloads/Codewithharry
• $ ./hyder.exe
Enter data to calculate checksum: Hello
Checksum of the given data: 244
Enter received data to verify checksum: Hello
Data is correct. No errors detected.

hyder@HyderPresswala MINGW64 ~/Downloads/Codewithharry
• $ ./hyder.exe
Enter data to calculate checksum: Hello
Checksum of the given data: 244
Enter received data to verify checksum: Hola
Error detected in received data.

hyder@HyderPresswala MINGW64 ~/Downloads/Codewithharry
○ $
```

Conclusion :-

We learned what is Cyclic Redundancy Check and Check sum and learned to implement it

Post Lab Questions

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

- The generator polynomial must have a degree of n , producing an $n+1$ bit CRC.
- The polynomial should not be divisible by x to avoid a leading zero.
- It must be irreducible, meaning it cannot be factored into smaller polynomials.
- The polynomial should detect common error types like single-bit, double-bit, and burst errors.
- The degree of the generator polynomial affects the CRC length and, thus, the error detection capability.
- Common generator polynomials include CRC-16 and CRC-32, which are widely used in networking and file integrity checks.

2. State the advantages and disadvantages of Internet Checksum.

Advantages:

- Simple algorithm and easy to implement in hardware or software.
- Minimal computational overhead, making it efficient for low-power or constrained devices.
- Suitable for detecting random single-bit errors.
- Commonly used in networking protocols like TCP/UDP for error-checking purposes.
- Faster than more complex algorithms like CRC due to reduced operations.
- Requires less memory, making it ideal for smaller systems.

Disadvantages:

- Less effective for detecting multiple-bit errors.



- Cannot detect errors when bit order is swapped.
- Weaker detection compared to CRC, especially for burst errors.
- Limited to specific networking applications, not suitable for critical data integrity verification.
- Vulnerable to specific patterns of errors that may pass undetected.
- Not robust for use in high-error environments like storage systems or files.

Date :- 11-11-2024

Signature of Faculty In-charge