

Batch: A2

Roll No.: 16010121045

Experiment 4

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 –

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

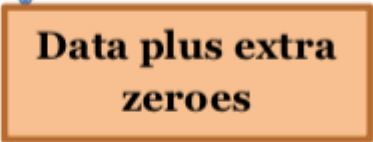


Figure 1: CRC calculation by sender

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.

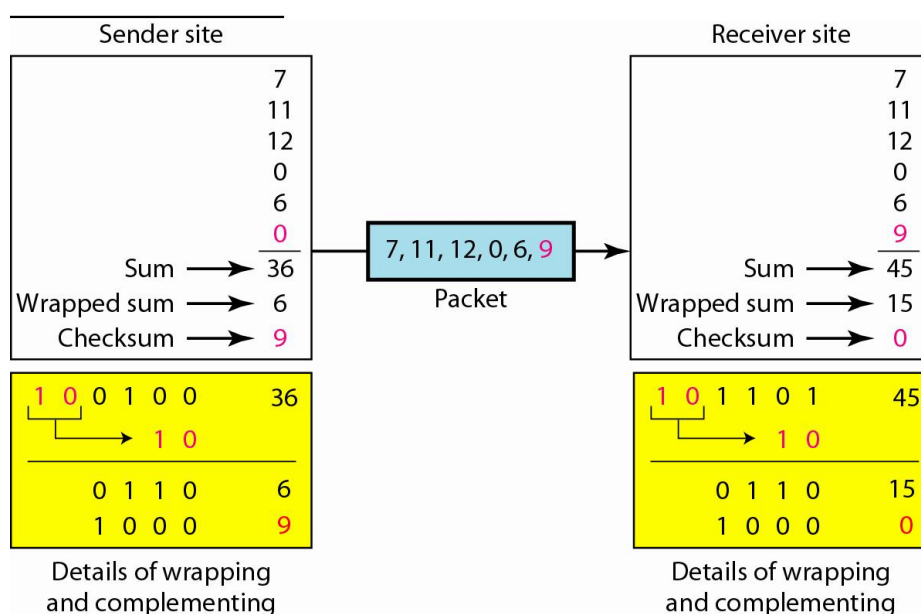
Internet Checksum :

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	1	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	1	Sum
0	0	0	0		Checksum (new)

a. Checksum at the receiver site

IMPLEMENTATION: (printout of codes)

```
Enter Data
100100
Enter Key
1101

Sender side...
Remainder : 001
Encoded Data (Data + Remainder) :100100001

Receiver side...
correct message received
```

```
import java.util.*;
import java.util.Arrays;
public class binkey {

    static String Xor(String a, String b)
    {

        String result = "";
        int n = b.length();

        for (int i = 1; i < n; i++) {
            if (a.charAt(i) == b.charAt(i))
                result += "0";
            else
                result += "1";
        }
        return result;
    }

    static String Mod2Div(String dividend, String divisor)
    {

        int pick = divisor.length();

        String tmp = dividend.substring(0, pick);
```

```
int n = dividend.length();

while (pick < n) {
    if (tmp.charAt(0) == '1')

        tmp = Xor(divisor, tmp)
            + dividend.charAt(pick);
    else

        tmp = Xor(new String(new char[pick])
            .replace("\0", "0"),
            tmp)
            + dividend.charAt(pick);

    pick += 1;
}

if (tmp.charAt(0) == '1')
    tmp = Xor(divisor, tmp);
else
    tmp = Xor(new String(new char[pick])
        .replace("\0", "0"),
        tmp);

return tmp;
}

static void EncodeData(String data, String key)
{
    int l_key = key.length();

    String appended_data
        = (data
            + new String(new char[l_key - 1])
                .replace("\0", "0"));

    String remainder = Mod2Div(appended_data, key);

    String codeword = data + remainder;
    System.out.println("Remainder : " + remainder);
    System.out.println(
        "Encoded Data (Data + Remainder) : " + codeword
        + "\n");
}
```



```
}

static void Receiver(String data, String key)
{
    String currxor
        = Mod2Div(data.substring(0, key.length()), key);
    int curr = key.length();
    while (curr != data.length()) {
        if (currxor.length() != key.length()) {
            currxor += data.charAt(curr++);
        }
        else {
            currxor = Mod2Div(currxor, key);
        }
    }
    if (currxor.length() == key.length()) {
        currxor = Mod2Div(currxor, key);
    }
    if (currxor.contains("1")) {
        System.out.println(
            "there is some error in data");
    }
    else {
        System.out.println("correct message received");
    }
}

public static void main(String[] args)
{
    Scanner ob=new Scanner(System.in);
    System.out.println("Enter Data");
    String data = ob.nextLine();
    System.out.println("Enter Key");
    String key = ob.nextLine();
    System.out.println("\nSender side...");
    EncodeData(data, key);

    System.out.println("Receiver side...");
    Receiver(data+Mod2Div(data+new String(new char[key.length() - 1])
        .replace("\0", "0"),key),key);
}
}
```

Checksum

```
S C:\Users\kjsce_comp4\Documents\Ath> & 'C:\Users\kjsce_comp4\AppData\Roaming\Code\User\work
Enter the input string:
Sample
a : 7361
p : 6d70
e : 6c65
The generated checksum is: b2c8
Enter the data for sending to the receiver:
Sample
Enter the checksum for sending to the receiver:
b2c8
a : 7361
p : 6d70
e : 6c65
The syndrome is 0
Data received without any errors
S C:\Users\kjsce_comp4\Documents\Ath> |
```

```
import java.util.*;

public class checksum {
    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the input string: ");
        String input = sc.next();

        int checkSum = generateCheckSum(input);
        System.out.println("The generated checksum is: " +
Integer.toHexString(checkSum));

        System.out.println("Enter the data for sending to the receiver: ");
        input = sc.next();

        System.out.println("Enter the checksum for sending to the receiver: ");

        checkSum = Integer.parseInt((sc.next()), 16);
        receive(input, checkSum);
    }
}
```

```
        sc.close();
    }

    static int generateChecksum(String s) {

        String hexadecimalValue = new String();
        int x, i, checksum = 0;

        for (i = 0; i < s.length() - 2; i = i + 2) {

            x = (int) (s.charAt(i));

            hexadecimalValue = Integer.toHexString(x);

            x = (int) (s.charAt(i + 1));

            hexadecimalValue = hexadecimalValue + Integer.toHexString(x);

            System.out.println(s.charAt(i) + "" + s.charAt(i + 1) + " : " +
hexadecimalValue);

            x = Integer.parseInt(hexadecimalValue, 16);
            checksum += x;
        }

        if (s.length() % 2 == 0) {
            x = (int) (s.charAt(i));
            hexadecimalValue = Integer.toHexString(x);

            x = (int) (s.charAt(i + 1));

            hexadecimalValue = hexadecimalValue + Integer.toHexString(x);

            System.out.println(s.charAt(i) + "" + s.charAt(i + 1) + " : " +
hexadecimalValue);

            x = Integer.parseInt(hexadecimalValue, 16);
        } else {
            x = (int) (s.charAt(i));

            hexadecimalValue = "00" + Integer.toHexString(x);

            x = Integer.parseInt(hexadecimalValue, 16);
        }
    }
}
```

```
        System.out.println(s.charAt(i) + " : " + hexadecimalValue);
    }
    checksum += x;

    hexadecimalValue = Integer.toHexString(checksum);

    if (hexadecimalValue.length() > 4) {
        int carry = Integer.parseInt("'" + hexadecimalValue.charAt(0)), 16);

        hexadecimalValue = hexadecimalValue.substring(1, 5);

        checksum = Integer.parseInt(hexadecimalValue, 16);

        checksum += carry;
    }
    checksum = generateComplement(checksum);

    return checksum;
}

static void receive(String s, int checksum) {

    int generatedChecksum = generateChecksum(s);

    generatedChecksum = generateComplement(generatedChecksum);

    int syndrome = generatedChecksum + checksum;

    syndrome = generateComplement(syndrome);

    System.out.println("The syndrome is " + Integer.toHexString(syndrome));

    if (syndrome == 0) {
        System.out.println("Data received without any errors");
    } else {
        System.out.println("Some error encountered in the received data");
    }
}

static int generateComplement(int checksum) {
```

```
        checksum = Integer.parseInt("FFFF", 16) - checksum;
        return checksum;
    }
}
```

CONCLUSION:

We were successfully able to implement the CRC code using XOR division in a Java code and were able to get correct output. We understood the concept, the uses and the advantages and disadvantages of CRC's.

Post Lab Questions

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

Determine the data length.

This is the payload being protected by the CRC, in bits. Decide if you need to allow for some headroom in this value in case your message has variable size and could be expanded beyond its current maximum. If you need to be able to make guarantees about probability of undetected error rates then you may need to ensure all possible data lengths are covered by your polynomial, but in some situations there can be an argument for accepting that larger messages will suffer a lower HD when the frequency or importance of those messages is low.

Find the smallest polynomial with the largest Hamming Distance that covers your data length.

While doing this, bear in mind these points

- ☐ HD=3 is barely an improvement over good checksum methods, so try to achieve HD=4 minimum to make the computational expense of CRC worthwhile. HD=3 means that all 2-bit errors are detected.
- ☐ aiming higher than HD=6 is probably overkill given that this value is considered sufficient for high-integrity systems, but you might be able to justify it in specialized cases.
- ☐ You might want to only choose between 8-bit, 16-bit, 24-bit and 32-bit polynomials because these fit your processor data word sizes or your message has space for a whole number of bytes for the FCS.

- ☐ If you need to steal a bit from your FCS field for other purposes, bear in mind the effect it has on the protected data length or HD, and it reduces the burst error detection length as well as the fraction of undetected errors.

Allow for expansion of data word size.

If you are close to a boundary value of data length for your chosen polynomial, think about whether it is a good idea to go up one step now to avoid problems later.

If two polynomials of the same size offer the same length protection at your chosen HD, choose the one where the length at HD+1 is better.

If those are the same, look up the Hamming Weights of the two polynomials at that length and HD and choose the one with the lowest HW.

2. State the advantages and disadvantages of Internet Checksum.

Checksum is the error detection method used by upper layer protocols and is considered to be more reliable than LRC, VRC and CRC. This method makes the use of Checksum Generator on Sender side and Checksum Checker on Receiver side.

Advantage:

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

- - ☐ The checksum detects all the errors involving an odd number of bits.
 - ☐ It detects most of the errors involving an even number of bits.
 - ☐ It also verifies the identity and file source by verifying the checksum values.

Disadvantage:

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.

- ☐ Using checksum can be a difficult process.
- ☐ It takes some extra time to verify checksum values.
- ☐ The checksum process can be fooled by the parties who can manipulate the checksum values.

Date: 29/08/2023

Signature of Faculty In-charge

Department of Computer Engineering