UNIT 3: ERROR AND THEIR TYPES

Error: An error is the change or the mismatching take place between the data unit sent by transmitter and the data unit received by the receiver e.g. 10101010 sent by sender 10101011 received by receiver. Here is an error of 1 bit.

Types of error

Single bit error

Burst error

A **single bit error** is an error condition that alters(modifies) one bit but does not affect nearby bits.

A **burst error** is a contiguous sequence of bits in which the first and last bits and any number of intermediate bits are received in error.

Error detection

Regardless of the design of the transmission system, there will be errors, resulting in the change of one or more bits in a transmitted frame.

When a code word is transmitted one or more number of transmitted bits will be reversed due to transmission impairments. Thus error will be introduced.

It is possible to detect these errors if the received code word is not one of the valid code words. To detect the errors at the receiver, the valid code words should be separated by a distance of more than 1.

Error Correction

It is a mechanism for the receiver to locate and correct the error without resorting to retransmission. Send additional information so incorrect data can be corrected and accepted. Error correction is the additional ability to reconstruct the original, error free data.

Error Control

Error control refers to mechanisms to detect and correct errors that occur in the transmission of frames. The most common techniques for error control are based on some or all of the following:

1. Error detection
2. Positive acknowledgement

3. Retransmission after time-out

4. Negative acknowledgement and retransmission.

These mechanisms are also referred as **automatic repeat request (ARQ).**

Three types of redundancy checks used in data communication

Error detection uses the concept of redundancy, which means adding extra bits for detecting errors at the destination. There are three types of redundancy checks are common in data communication:

(a) Parity check

(h) Cyclic Redundancy check (CRC)

(c) Checksum.

Parity bit

In this technique, **a redundant bit called a parity bit**, is added to every data unit so that the total number of 1's in the unit becomes even (or odd).

In the example, we can understand that

Suppose we want to transmit **1100001.**

Adding the number of 1's gives us 3, an odd number.

Before transmitting, we pass the data unit through a **parity generator**. The parity generator counts the 1's and appends the parity bit to the end (al in this case).

Difference between even parity and odd parity

In case of redundancy check method we have to append the data unit with some extra bits. These extra bits are called **parity.**

This parity or parity hit can be even or odd.

In case of even parity we have to make number of 1's even, including the parity hit.

e.g. **1110001** is the data unit where the no. of l's is already even then we will insert 0 at the next to data unit it', 1110001.

In case of odd parity we have to make no. of l's odd, including the parity bit. e.g. 1111000 is the data unit, where the no. of 1's is even then we will insert 1 at the next to data unit i.e. 11110001.

CRC (Cyclic Redundancy Check) method of Error Detection

Cyclic Redundancy Check (CRC): Cyclic Redundancy check method is most powerful mechanism of error detecting. Unlike the parity check which is based on addition, CRC is based on **binary division.**

In CRC, instead of adding bits to achieve a desired parity, a sequence of redundant bits, called the CRC or the CRC remainder, is appended to the end of a data unit so that the resulting data unit becomes exactly divisible by a second predetermined binary number.

At its destination the incoming data unit is divided by the same number. If at this step there is no remainder, the data unit is assumed to be intact and is therefore accepted.

A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.

The redundancy bits used by CRC are derived by dividing the data unit by a predetermined divisor, the remainder is the CRC.

A CRC must have **two qualities**:

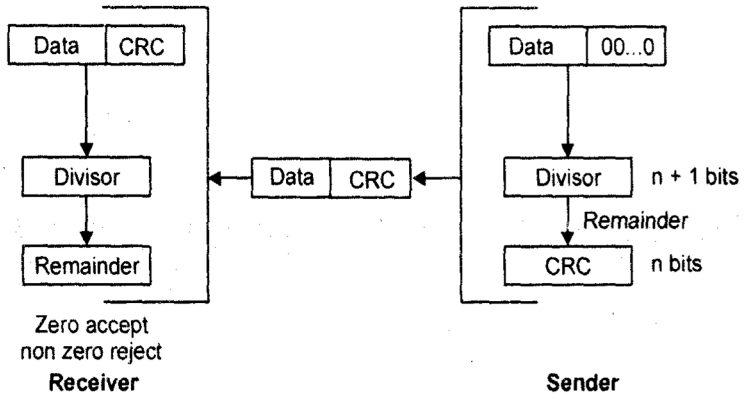
1. It must have exactly one less bit than the divisor.
2. Appending it to the end of the data string must make the resulting bit sequence exactly divisible by the divisor.

CRC generator and checker

First, a string of n 0's is appended to the data unit. The number n is less than the number of bits in the predetermined divisor, which are n + 1 bits.

Second, the newly formed data unit is divided by the divisor, using a process called binary division the remainder resulting from this division is the CRC.

Third, the CRC of n bits derived in step 2 replaces the appended Os at the end of the data unit. The data unit arrives at the receiver data first followed by the CRC. The receiver treats the whole string as a unit and divides it by the same divisor that was used to find the CRC remainder.



If the string arrives without error, the CRC checker yields a remainder of zero and the data unit passes. If the string has been changed in transit the division yields a non-zero remainder and the data unit does not pass.

Start with the message to be encoded:

```
11010011101100
```

This is first padded with zeroes corresponding to the bit length *n* of the CRC. Here is the first calculation for computing a 3-bit CRC:

```
11010011101100 000 <--- input left shifted by 3 bits
1011            <--- divisor (4 bits)
-----------------
01100011101100 000 <--- result
```

If the input bit above the leftmost divisor bit is 0, do nothing and move the divisor to the right by one bit. If the input bit above the leftmost divisor bit is 1, the divisor is XORed into the input. The divisor is then shifted one bit to the right, and the process is repeated until the divisor reaches the right-hand end of the input row. Here is the entire calculation:

```
11010011101100 000 <--- input left shifted by 3 bits
1011            <--- divisor
01100011101100 000 <--- result
 1011           <--- divisor
00111011101100 000
  1011
```

```
00010111101100 000
  1011
00000001101100 000
     1011
00000000110100 000
      1011
00000000011000 000
       1011
00000000001110 000
        1011
00000000000101 000
         101 1
----------------
00000000000000 100 <---remainder (3 bits)
```

Since the leftmost divisor bit zeroed every input bit it touched, when this process ends the only bits in the input row that can be nonzero are the n bits at the right-hand end of the row. These *n* bits are the remainder of the division step, and will also be the value of the CRC function.

The validity of a received message can easily be verified by performing the above calculation again, this time with the check value added instead of zeroes. The remainder should equal zero if there are no detectable errors.

```
11010011101100 100 <--- input with check value
1011           <--- divisor
01100011101100 100 <--- result
 1011          <--- divisor ...
00111011101100 100
```

and so on until:

```
00000000001110 100
       1011
0000000000 101 100
        101 1
------------------
            0 <--- remainder
```