Error detection and correction

When data received is not same as it was sent then that means error has occurred during transmission.

Errors are of 2 types

a) Single bit error (single bit is corrupted)
b) Bust error (multiple bits are corrupted)

A burst error is more likely to occur than a single-bit error. The duration of noise is normally longer than the duration of 1 bit, which means that when noise affects data, it affects a set of bits. The number of bits affected depends on the data rate and duration of noise. For example, if we are sending data at I kbps, a noise of 10 ms can affect 10 bits; if we are sending data at 1 Mbps, the same noise can affect 10,000 bits.

Detection Versus Correction

The correction of errors is more difficult than the detection. In error detection, we are looking only to see if any error has occurred. The answer is a simple yes or no. We are not even interested in the number of errors. A single-bit error is the same for us as a burst error.

In error correction, we need to know the exact number of bits that are corrupted and more importantly, their location in the message. The number of the errors and the size of the message are important factors. If we need to correct one single error in an 8-bit data unit, we need to consider eight possible error locations; if we need to correct two errors in a data unit of the same size, we need to consider 28 possibilities. You can imagine the receiver's difficulty in finding 10 errors in a data unit of 1000 bits.

Hence, we can say that retransmission is better option than error correction.

BLOCK CODING

In block coding, we divide or chop our message into blocks, each of k bits, called datawords. We then add *r* redundant bits to each block to make the length $n=k+r$. The resulting n-bit blocks are called codewords (which is actually sent from sender to receiver i.e in place of sending dataword we send codeword corresponding to that). It is important to know that we have a set of datawords, each of size k, and a set of codewords, each of size of n.

Error Detection

With *k* bits, we can create a combination of 2^k datawords; with n bits, we can create a combination of 2^n codewords. Since n>k, the number of possible codewords is larger than the number of possible datawords. The block coding process is one-to-one; the same dataword is always encoded as the same codeword. This means that we have 2^n-2^k codewords that are not used. We call these codewords invalid or illegal. This is the basic logic behind the error detection.

Let us assume that k =2 and n =3. The following table shows the list of datawords and codewords. Later, we will see how to derive a codeword from a dataword

Datawords	Codewords
00	000
01	011
10	101
11	110

Assume the sender encodes the dataword 01 as 011 and sends it to the receiver. Consider the following cases:

1. The receiver receives 011. It is a valid codeword. The receiver extracts the dataword 01 form it.
2. The codeword is corrupted during transmission, and 111 is received (the leftmost bit is corrupted).
   This is not a valid codeword and is discarded.

3. The codeword is corrupted during transmission, and 000 is received (the right two bits are corrupted). This is a valid codeword. The receiver

incorrectly extracts the dataword 00. Two corrupted bits have made the error undetectable.

Hence, we can say that an error-detecting code can detect only the types of errors for which it is designed; other types of errors may remain undetected.

Hamming Distance

One of the central concepts in coding for error control is the idea of the Hamming distance. **The Hamming distance between two words (of the same size) is the number of differences between the corresponding bits.**

We show the Hamming distance between two words x and y as d(x, y).

The Hamming distance can easily be found if we apply the XOR operation on the two words and count the number of 1s in the result.

Note that the Hamming distance is a value greater than zero.

Let us find the Hamming distance between two pairs of words.

The Hamming distance d(000, 011) is 2 because 000 xor 011 is 011 (two 1s).

The Hamming distance d(10101, 11110) is 3 because 10101 xor 11110 is 01011 (three 1s).

Minimum Hamming Distance

Although the concept of the Hamming distance is the central point in dealing with error detection and correction codes, the measurement that is used for designing a code is the minimum Hamming distance. In a set of words, the minimum Hamming distance is the smallest Hamming distance between all possible pairs. We use d_{min} to define the minimum Hamming distance in a coding scheme.

To find the minimum hamming distance, we find the Hamming distances between all words and select the smallest one.

Hence, we can say that to find the min. hamming distance b/w 'n' codewords we must have to check ${}^{n}C_{2}$ combinations

In order to detect 't' bit error min. hamming distance b/w codewords must be t+1.

LINEAR BLOCK CODES

Almost all block codes used today belong to a subset called linear block codes. The use of nonlinear block codes for error detection and correction is not as widespread because their structure makes theoretical analysis and implementation difficult. We therefore concentrate on linear block codes.

A linear block code is a code in which the XOR (addition modulo-2) of two valid codewords creates another valid codeword.

The above 2 examples (used in error detection and correction) are of linear block codes

These codes have special property that Minimum Distance for Linear Block Code is the number of 1s in the nonzero valid codeword with the smallest number of 1s.

Some examples of linear block codes

1. Simple Parity Check Code
2. 2-d Pariety check code
3. Hamming codes

The most familiar error-detecting code is the simple parity-check code. One extra bit, called the parity bit, is added to make the total number of 1s in the codeword even. Although some implementations specify an odd number of 1s, we discuss the even case. The minimum Hamming distance for this category is d_{min} =2, which means that the code is a single-bit error-detecting code; it cannot correct any error.

This code may detect all odd numbers of error but can't detect any even number of errors.

2-D Parity Check Code

A better approach is the two-dimensional parity check. In this method, the dataword is organized in a table (rows and columns).

The data to be sent, five 7-bit data, are put in separate rows. For each row and each column, 1 parity-check bit is calculated. The whole table is then sent to the receiver, which again finds the parity for each row and each column.

The two-dimensional parity check guarantees to correct only single bit error at any position The two-dimensional parity check can detect up to three errors that occur anywhere in the table. However, errors affecting 4 bits may not be detected.

Hamming Codes

It can also correct one bit error only. Here also parity bits are added in data.

Number of parity bits added according to the relation $2^k \geq n+k+1$ here k=number of parity bits and n is the number of data bits

Normally used n is 4 so according. To relation k is 3 hence data word of 4 bit is converted into code word of 7 bits now we need to find the position of these parity bits i.e., normally used n is 4 so acc. To relation k is 3

Hence data word of 4 bit is converted into code word of 7 bits now we need to find the position of these pariety bits i.e.

$d_7d_6d_5p_4d_3p_2p_1$

So, the pariety bits are added at position power of 2.

Now we need to find the value of these pariety bits

$D_7= P_4 +P_2 +P_1$

$D_6= P_4 +P_2$

$D_5=P_4 +P_1$

$D_3= P_2 +P_1$

Hence, we can say that

$P_4=d_7, d_6, d_5$

$P_2= d_7, d_6, d_3$

$P_1=d_7, d_5, d_3$

For e.g. Data word is 1010

Now add pariety bit as

$101p_40p_2p_1$

Assuming even pariety (by default we have to consider even pariety only)

P_4= 1, 0, 1 hence P_4 is 0

P_2=1,0,0 hance P_2 is 1

P_1= 110 hence P_1 is 0

So, the codeword sent is 1010010

Now if received is 1011010

Then we want to find position of error so for that again calculate the pariety bits and add correction bits where ever there is error i.e.

P_4= 1, 0, 1 and here p_4 is 1 but it must be 0 so C_3=1

P_2= 1, 0, 0 and here p_2 is 1 and it must be 1 so C_2=0

P_1 = 1, 1, 0 and here p_1 is 0 and it must be 0 so C_2=0

Error position is $C_3C_2C_1$ i.e., 100 i.e., 4th position so change it to 0.