## 3.2 Error correction and detection at the data link layer.

### 3.2.1 Error-Correcting Codes:

Network designers have developed two basic strategies for dealing with errors. One way is to include enough redundant information along with each block of data sent, to enable the receiver to deduce what the transmitted data must have been. The other way is to include only enough redundancy to allow the receiver to deduce that an error occurred, but not which error, and have it request a retransmission. The former strategy uses error-correcting codes and the latter uses error-detecting codes. The use of error-correcting codes is often referred to as forward error correction.

Each of these techniques occupies a different ecological niche. On channels that are highly reliable, such as fiber, it is cheaper to use an error detecting code and just retransmit the occasional block found to be faulty. However, on channels such as wireless links that make many errors, it is better to add enough redundancy to each block for the receiver to be able to figure out what the original block was, rather than relying on a retransmission, which itself may be in error.

To understand how errors can be handled, it is necessary to look closely at what an error really is. Normally, a frame consists of m data (i.e., message) bits and r redundant, or check, bits. Let the total length be n (i.e., $n = m + r$). An n-bit unit containing data and check bits is often referred to as an n-bit codeword.

Given any two code words, say, 10001001 and 10110001, it is possible to determine how many corresponding bits differ. In this case, 3 bits differ. To determine how many bits differ, just exclusive OR the two code words and count the number of 1 bits in the result, for example:

The number of bit positions in which two code words differ is called the Hamming distance. Its significance is that if two codewords are a Hamming distance d apart, it will require d single-bit errors to convert one into the other.

In most data transmission applications, all 2m possible data messages are legal, but due to the way the check bits are computed, not all of the 2n possible codewords are used. Given the algorithm for computing the check bits, it is possible to construct a complete list of the legal codewords, and from this list find the two codewords whose Hamming distance is minimum. This distance is the Hamming distance of the complete code. The error-detecting and error correcting properties of a code depend on its Hamming distance. To detect d errors, you need a distance d + 1 code because with such a code there is no way that d single-bit errors can change a valid codeword into another valid codeword. When the receiver sees an invalid codeword, it can tell that a transmission error has occurred. Similarly, to correct d errors, you need a distance 2d + 1 code because that way the legal codewords are so far apart that even with d changes, the original codeword is still closer than any other codeword, so it can be uniquely determined. As a simple example of an error-detecting code, consider a code in which a single parity b appended to the data. The parity bit is chosen so that the number of 1 bits in the codeword is even (or odd). For example, when 1011010 is sent in even parity, a bit is added to the end to make it 10110100. With odd parity 1011010 becomes 10110101. A code with a single parity bit has a distance 2, since any single-bit error produces a codeword with the wrong parity. It can be used to detect single errors.

As a simple example of an error-correcting code, consider a code with only four valid codewords: 0000000000, 0000011111, 1111100000, and 1111111111

This code has a distance 5, which means that it can correct double errors. If the codeword 0000000111 arrives, the receiver knows that the original must have been 0000011111. If, however, a triple error changes 0000000000 into 0000000111, the error will not be corrected properly.

Imagine that we want to design a code with m message bits and r check bits that will allow all single errors to be corrected. Each of the 2m legal messages has n illegal codewords at a distance 1 from it. These are formed by systematically inverting each of the n bits in the n-bit codeword formed from it. Thus, each of the 2m legal messages requires n + 1 bit patterns dedicated to it. Since the total number of bit patterns is 2n, we must have $(n + 1)2m \leq 2n$.

Using $n = m + r$, this requirement becomes $(m + r + 1) \leq 2r$. Given m, this puts a lower limit on the number of check bits needed to correct single errors. This theoretical lower limit can, in fact, be achieved using a method due to Hamming (1950).

The bits of the codeword are numbered consecutively, starting with bit 1 at the left end, bit 2 to its immediate right, and so on. The bits that are powers of 2 (1, 2, 4, 8, 16, etc.) are check bits. The rest (3, 5, 6, 7, 9, etc.) are filled up with the m data bits. Each check bit forces the parity of some collection of bits, including itself, to be even (or odd). A bit may be included in several parity computations. To see which check bits the data bit in position k contributes to, rewrite k as a sum of powers of 2.For example, $11 = 1 + 2 + 8$ and $29 = 1 + 4 + 8 + 16$. A bit is checked by just those check bits occurring in its expansion (e.g., bit 11 is checked by bits 1, 2, and 8). When a codeword arrives, the receiver initializes a counter to zero. It then examines each check bit, k ($k = 1, 2, 4, 8 \ldots$), to see if it has the correct parity. If not, the receiver adds k to the counter. If the counter is zero after all the check bits have been examined (i.e., if they were all correct), the codeword is accepted as valid. If the counter is nonzero, it contains the number of the incorrect bit. For example, if check bits 1, 2, and 8 are in error, the inverted bit is 11, because it is the only one checked by bits 1, 2, and 8. Figure 4.1 shows some 7-bit ASCII characters encoded as 11-bit codewords using a Hamming code. Remember that the data are found in bit positions 3, 5, 6, 7, 9, 10, and 11.

| Char. | ASCII | Check bits |
|---|---|---|
| H | 1001000 | 00110010000 |
| a | 1100001 | 10111001001 |
| m | 1101101 | 11101010101 |
| m | 1101101 | 11101010101 |
| i | 1101001 | 01101011001 |
| n | 1101110 | 01101010110 |
| g | 1100111 | 01111001111 |
| | 0100000 | 10011000000 |
| c | 1100011 | 11111000011 |
| o | 1101111 | 10101011111 |
| d | 1100100 | 11111001100 |
| e | 1100101 | 00111000101 |

Order of bit transmission

**Fig.4.1. Use of a Hamming code to correct burst errors**



Hamming codes can only correct single errors. However, there is a trick that can be used to permit Hamming codes to correct burst errors. A sequence of k consecutive code words is arranged as a matrix, one codeword per row. Normally, the data would be transmitted one codeword at a time, from left to right. To correct burst errors, the data should be transmitted one column at a time, starting with the leftmost column. When all k bits have been sent, the second column is sent, and so on, as indicated in Fig.4.1. When the frame arrives at the receiver, the matrix is reconstructed, one column at a time. If a burst error of length k occurs, at most 1 bit in each of the k codewords will have been affected, but the Hamming code can correct one error per codeword, so the entire block can be restored. This method uses kr check bits to make blocks of km data bits immune to a single burst error of length k or less.