10.1 INTRODUCTION

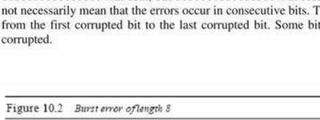
Types of Errors

Whenever bits flow from one point to another, they are subject to unpredictable changes because of interference. This interference can change the shape of the signal. In a single-bit error, a 0 is changed to a 1 or a 1 to a O. In a burst error, multiple bits are changed. For example, a 11100 s burst of impulse noise on a transmission with a data rate of 1200 bps might change all or some of the12 bits of information.

Single-Bit Error

The term *single-bit error* means that only 1 bit of a given data unit (such as a byte, character, or packet) is changed from 1 to 0 or from 0 to 1. Figure 10.1 shows the effect of a single-bit error on a data unit. To understand the impact of the change, imagine that each group of 8 bits is an ASCII character with a 0 bit added to the left. In Figure 10.1,00000010 (ASCII *STX)* was sent, meaning *start of text,* but 00001010 (ASCII *LF)* was received, meaning *line feed.*

Figure 10.1 *Single-bit error*

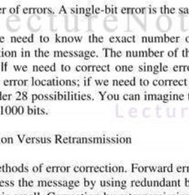


Single-bit errors are the least likely type of error in serial data transmission. To understand why, imagine data sent at 1 Mbps. This means that each bit lasts only 1/1,000,000 s, or 1)ls. For a single-bit error to occur, the noise must have a duration of only 1)ls, which is very rare; noise normally lasts much longer than this.

Burst Error

The term *burst error* means that 2 or more bits in the data unit have changed from 1 to 0 or from 0 to 1. Figure 10.2 shows the effect of a burst error on a data unit. In this case, 0100010001000011 was sent, but 0101110101100011 was received. Note that a burst error does not necessarily mean that the errors occur in consecutive bits. The length of the burst is measured from the first corrupted bit to the last corrupted bit. Some bits in between may not have been corrupted.

Figure 10.2 *Burst error of length 8*



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 11100 s can affect 10 bits; if we are sending data at I Mbps, the same noise can affect 10.000 bits.

Redundancy

The central concept in detecting or correcting errors is redundancy. To be able to detect or correct errors, we need to send some extra bits with our data. These redundant bits are added by the sender and removed by the receiver. Their presence allows the receiver to detect or correct corrupted 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.

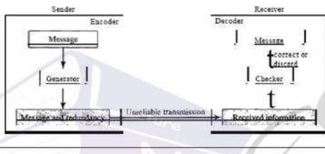
Forward Error Correction Versus Retransmission

There are two main methods of error correction. Forward error correction is the process in which the receiver tries to guess the message by using redundant bits. This is possible, as we see later, if the number of errors is small. Correction by retransmission is a technique in which the receiver detects the occurrence of an error and asks the sender to resend the message. Resending is repeated until a message arrives that the receiver believes is error-free

Redundancy is achieved through various coding schemes. The sender adds redundant bits through a process that creates a relationship between the redundant bits and the actual data bits. The receiver checks the relationships between the two sets of bits to detect or correct the errors. The ratio of redundant bits to the data bits and the robustness of the process are important factors in any coding scheme. Figure 10.3 shows the general idea of coding.

We can divide coding schemes into two broad categories:
block coding and convolution coding..

Figure 10.3 *The structure of encoder and decoder*



Modular Arithmetic

In modular arithmetic, use only a limited range of integers. An upper limit, called a modulus *N* then use only the integers 0 to *N* - I, inclusive. This is *modulo-N* arithmetic.

For example, if the modulus is 12, we use only the integers 0 to 11, inclusive. An example of modulo arithmetic is our clock system. It is based on modulo-12 arithmetic, substituting the number 12 for O. In a *modulo-N* system, if a number is greater than *N,* it is divided by *N* and the remainder is the result. If it is negative, as many *Ns* as needed are added to make it positive.

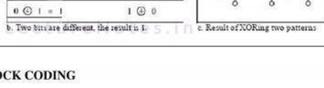
Modulo-2 Arithmetic

Of particular interest is modulo-2 arithmetic. In this arithmetic, the modulus *N* is 2. We can use only 0 and 1. Operations in this

Adding:	0+0=0	0+1=1	1+0=1	1+1=0
Subtracting:	0-0=0	0-1=1	1-0=1	1-1=0

Notice particularly that addition and subtraction give the same results. In this arithmetic we use the XOR (exclusive OR) operation for both addition and subtraction. The result of an XOR operation is 0 if two bits are the same; the result is 1 if two bits are different. Figure 10.4 shows this operation.

Figure 10.4 *XORing of two single bits or two words*

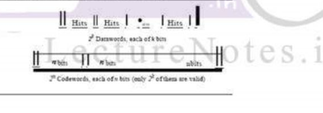


10.2 BLOCK CODING

In block coding, we divide our message into blocks, each of *k* bits, called datawords. We add *r* redundant bits to each block to make the length *n = k + r.* The resulting *n-bit* blocks are called codewords. How the extra *r* bits is chosen or calculated is something we will discuss later. For the moment, 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.* With *k* bits, we can create a combination of *2k* datawords; with *n* bits, we can create a combination of *2n* 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 *2n - 2k* codewords that are not used. We call these codewords invalid or illegal. Figure 10.5 shows the situation.

Figure 10.5 *Datawords and codewords in block coding*



Error Detection

How can errors be detected by using block coding? If the following two conditions are met, the receiver can detect a change in the original codeword.

1. The receiver has (or can find) a list of valid codewords.
2. The original codeword has changed to an invalid one.

Figure 10.6 shows the role of block coding in error detection.