MODULE-II

The Data Link Layer

The data link layer uses the services of the physical layer to send and receive bits over communication channels. It has a number of functions, including:

1. Providing a well-defined service interface to the network layer.
2. Dealing with transmission errors.
3. Regulating the flow of data so that slow receivers are not swamped by fast senders.

To accomplish these goals, the data link layer takes the packets it gets from the network layer and encapsulates them into frames for transmission. Each frame contains a frame header, a payload field for holding the packet, and a frame trailer.

Types of Errors

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.
- ✓ Single-bit errors are the least likely type of error in serial data transmission
- ✓ For a single-bit error to occur the noise must have a duration of only 1 micro second 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.
- ✓ 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.
- ✓ 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

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.

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 (usually, not all errors can be detected).

Error Detection

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 code words.
2. The original codeword has changed to an invalid one.

The sender creates code words out of data words by using a generator that applies the rules and procedures of encoding (discussed later). Each codeword sent to the receiver may change during transmission. If the received codeword is the same as one of the valid code words, the word is accepted; the corresponding data word is extracted for use. If the received codeword is not valid, it is discarded. However, if the codeword is corrupted during transmission but the received word still matches a valid codeword, the error remains undetected. This type of coding can detect only single errors. Two or more errors may remain undetected.

Example

Let us assume that *k* =2 and *n* =3. Table 10.1 shows the list of data words and code words. Later, we will see how to derive a codeword from a data word.

A code for error detection

Datawords	Codewords
00	000
01	011
10	101
11	111

Assume the sender encodes the data word 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 data word 01 from 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 data word 00. Two corrupted bits have made the error undetectable.

An error-detecting code can detect only the types of errors for which it is designed; other types of errors may remain undetected.

Error Correction

Error correction is much more difficult than error detection. In error detection, the receiver needs to know only that the received codeword is invalid; in error correction the receiver needs to find (or guess) the original codeword sent. We can say that we need more redundant bits for error correction than for error detection.

Example
Assume the data word is 01. The sender consults the table (or uses an algorithm) to create the Code word 01011. The codeword is corrupted during transmission, and 01001 is received (error in the second bit from the right). First, the receiver finds that the received codeword is not in the table. This means an error has occurred. (Detection must come before correction.) The receiver, assuming that there is only 1 bit corrupted, uses the following strategy to guess the correct data word.
A code for error correction

Datawords	Codewords
00	00000
01	01011
10	10101
11	11110

1. Comparing the received codeword with the first codeword in the table (01001 versus 00000), the receiver decides that the first codeword is not the one that was sent because there are two different bits.
2. By the same reasoning, the original codeword cannot be the third or fourth one in the table.

The original codeword must be the second one in the table because this is the only one that differs from the received codeword by 1 bit. The receiver replaces 01001 with 01011 and consults the table to find the dataword 01.

There are two basic strategies for dealing with errors.

Error-detecting Codes

Include only enough redundancy to allow the receiver to deduce that error has occurred, but not which error has occurred and the receiver asks for a retransmission.

Error-Correcting Codes

Include enough redundant information (extra bits are introduced into the data stream at the transmitter on a regular and logical basis) along with each block of data sent to enable the receiver to deduce what the transmitted character must have been.

To understand how errors can be handled, it is necessary to look closely at what error really is. Normally, a frame consists of m-data bits (i.e., message bits) and r-redundant bits (or check bits). Let the total number of bits be 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 10010101 and 11010100, it is possible to determine how many corresponding bits differ, just EXCLUSIVE OR the two code-words, and count the number of 1's in the result. The number of bits position in which code words differ is called the **Hamming distance**. If two code words are a Hamming distance d-apart, it will require d single-bit errors to convert one codeword to other.

The error detecting and correcting properties depends 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 code word into another valid code word. Whenever receiver sees an invalid code word, 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 code words are so far apart that even with d changes, the original codeword is still closer than any other code-word, so it can be uniquely determined.

Error Detecting Codes

Basic approach used for error detection is the use of redundancy, where additional bits are added to facilitate detection and correction of errors. Popular techniques are:

1. Simple Parity check
2. Two-dimensional Parity check
3. Checksum
4. Cyclic redundancy check

1. Simple Parity Checking or One-dimension Parity Check

- The most common and least expensive mechanism for error- detection is the simple parity check.
- In this technique, a redundant bit called parity bit, is appended to every data unit so that the number of 1s in the unit (including the parity becomes even).
- Blocks of data from the source are subjected to a check bit or Parity bit generator form, where a parity of 1 is added to the block if it contains an odd number of 1's and 0 is added if it contains an even number of 1's.
- At the receiving end the parity bit is computed from the received data bits and compared with the received parity bit, as shown in Figure. This scheme makes the total number of 1's even, that is why it is called even parity checking.

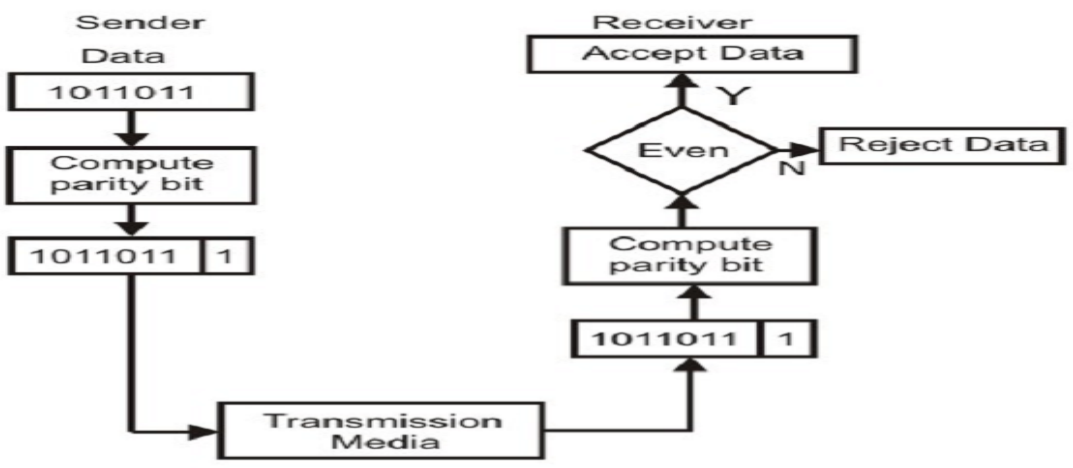


Figure 1 Even parity checking

NOTE: It is also possible to use odd-parity checking, where the number of 1's should be odd.

Parity Checking is divided into following types

A. **Vertical Redundancy check**
   Ex- Actual Data= 1110 1111 1001 0011
   After adding parity bits 1110**1** 1111**0** 1001**0** 0011**0** (Bold Number indicate parity bit)
B. **Longitudinal redundancy check** or **Horizontal redundancy check**
   Blocks of bit are arranged in table form and redundant row of bit is calculated as follows

Ex- Actual Data= 1110 1111 1001 0011 Arrange in a table form	1110 1111 1001 0011 **1011** (Parity bits)

Data to be send = 1110 1111 1001 0011 **1011**

2. **Two Dimensional Parity Check**
   Here a block of bits is divided into rows (each row typically represents a character with a single parity bit) and redundant row of nits is added to the whole block of bit.

Ex- Actual Data= 1110 1111 1001 0011 Arrange in a table form	1110 1111 1001 0011	**1** **0** **0** **0**
	1011	**1**
Data to be send =1110**1** 1111**0** 1001**0** 0011**0** **10111**		

- Two- Dimension Parity Checking increases the likelihood of detecting burst errors.
- 2-D Parity check of n bits can detect a burst error of n bits.
- A burst error of more than n bits is also detected by 2-D Parity check with a high probability.
- There is, however, one pattern of error that remains elusive. If two bits in one data unit are damaged and two bits in exactly same position in another data unit are also damaged, the 2-D Parity check checker will not detect an error. For example, if two data units: 11001100 and 10101100. If first and second from last bits in each of them is changed, making the data units as 01001110 and 00101110, the error cannot be detected by 2-D Parity check.

3. **Checksum**

- In checksum error detection scheme, the data is divided into k segments each of m bits. In the sender's end the segments are added using 1's complement arithmetic to get the sum.

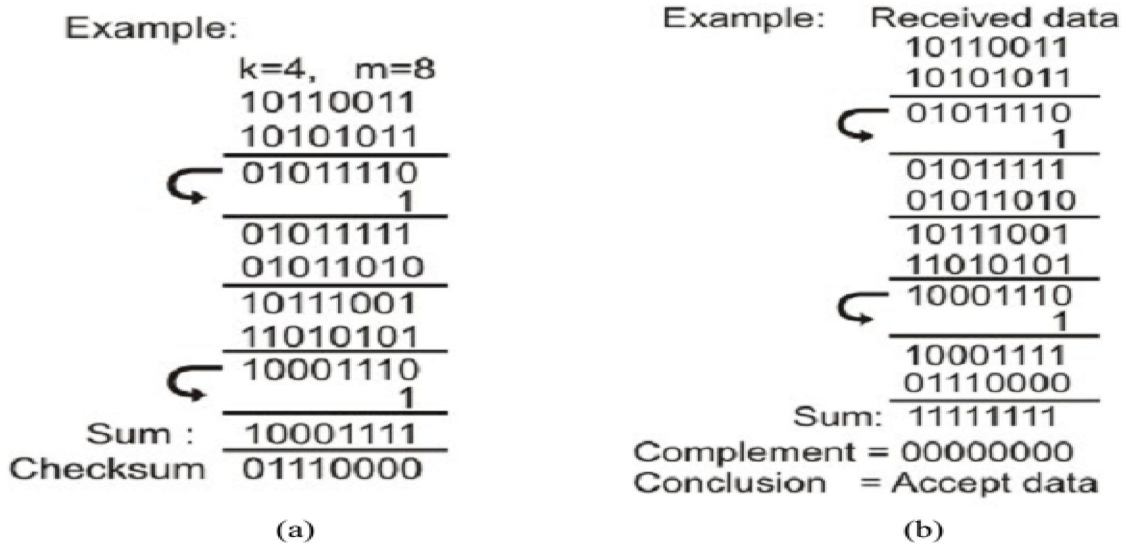


Figure 2 (a) Sender's end for the calculation of the checksum, (b) Receiving end for checking the checksum

- The sum is complemented to get the checksum. The checksum segment is sent along with the data segments as shown in Figure.
- At the receiver's end, all received segments are added using 1's complement arithmetic to get the sum. The sum is complemented. If the result is zero, the received data is accepted; otherwise discarded.

4. **Cyclic Redundancy Check**

- Cyclic Redundancy Check is the most powerful and easy to implement technique. Unlike checksum scheme, which is based on addition, CRC is based on binary division.
- In CRC, a sequence of redundant bits, called cyclic redundancy check bits, are appended to the end of data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number.

- At the 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 correct and is therefore accepted. A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.
- The generalized technique can be explained as follows.
  - If a k bit message is to be transmitted, the transmitter generates an r-bit sequence, known as Frame Check Sequence (FCS) so that the (k+r) bits are actually being transmitted.
  - Now this r-bit FCS is generated by dividing the original number, appended by r zeros, by a predetermined number. This number, which is (r+1) bit in length, can also be considered as the coefficients of a polynomial, called Generator Polynomial.
  - The remainder of this division process generates the r-bit FCS. On receiving the packet, the receiver divides the (k+r) bit frame by the same predetermined number and if it produces no remainder, it can be assumed that no error has occurred during the transmission. Operations at both the sender and receiver end are shown in Figure.

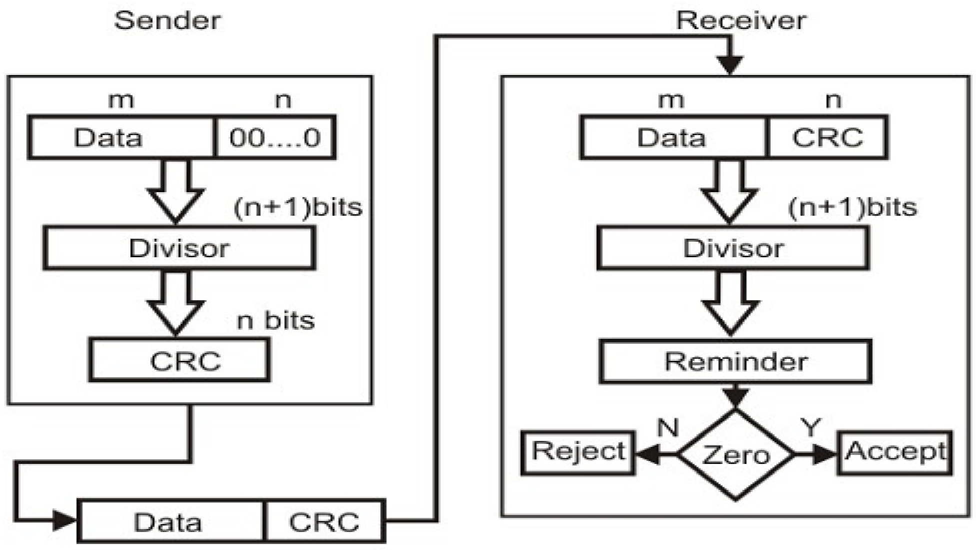
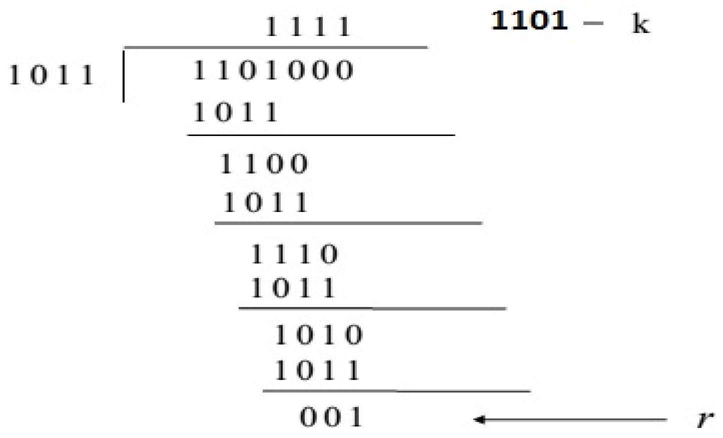


Figure 3 Cyclic Redundancy Checking

Dividing a sample 4-bit number by the coefficient of the generator polynomial x^3+x+1, which is 1011, using the modulo-2 arithmetic.

Modulo-2 arithmetic is a binary addition process without any carry over, which is just the Exclusive-OR operation. Consider the case where k=1101. Hence we have to divide 1101000 (i.e. k appended by 3 zeros) by 1011, which produces the remainder r=001, so that the bit frame (k+r) =1101001 is actually being transmitted through the communication channel. At the receiving end, if the received number, i.e., 1101001 is divided by the same generator polynomial 1011 to get the remainder as 000, it can be assumed that the data is free of errors.



- Problem

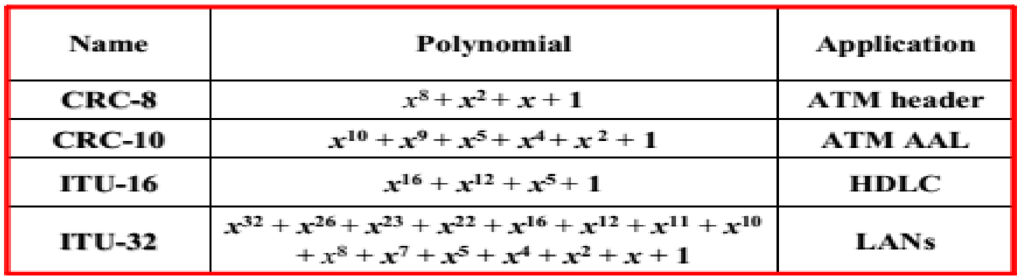
Data = 1101011011

Polynomial X4+X+1

Sender side CRC is 1110

- All the values can be expressed as polynomials of a dummy variable X. For example, for P = 11001 the corresponding polynomial is X^4+X^3+1.
- A polynomial is selected to have at least the following properties:
  - It should not be divisible by X.
  - It should not be divisible by (X+1).
- The first condition guarantees that all burst errors of a length equal to the degree of polynomial are detected. The second condition guarantees that all burst errors affecting an odd number of bits are detected.

Commonly used divisor polynomials are:



Name	Polynomial	Application
CRC-8	$x^8 + x^2 + x + 1$	**ATM header**
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	**ATM AAL**
ITU-16	$x^{16} + x^{12} + x^5 + 1$	**HDLC**
ITU-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	**LANs**

CRC is a very effective error detection technique. If the divisor is chosen according to the previously mentioned rules, its performance can be summarized as follows:

- CRC can detect all single-bit errors
- CRC can detect all double-bit errors (three 1's)

- CRC can detect any odd number of errors (X+1)
- CRC can detect all burst errors of less than the degree of the polynomial.
- CRC detects most of the larger burst errors with a high probability.
- For example CRC-12 detects 99.97% of errors with a length 12 or more.

Error Correcting Codes

The techniques that we have discussed so far can detect errors, but do not correct them.

Error Correction can be handled in two ways.

- One is when an error is discovered; the receiver can have the sender retransmit the entire data unit. This is known **as backward error correction**.
- In the other, receiver can use an error-correcting code, which automatically corrects certain errors. This is known as **forward error correction**.

In theory it is possible to correct any number of errors atomically. Error-correcting codes are more sophisticated than error detecting codes and require more redundant bits. The number of bits required to correct multiple-bit or burst error is so high that in most of the cases it is inefficient to do so. For this reason, most error correction is limited to one, two or at the most three-bit errors.

Forward error correction

Single-bit error correction

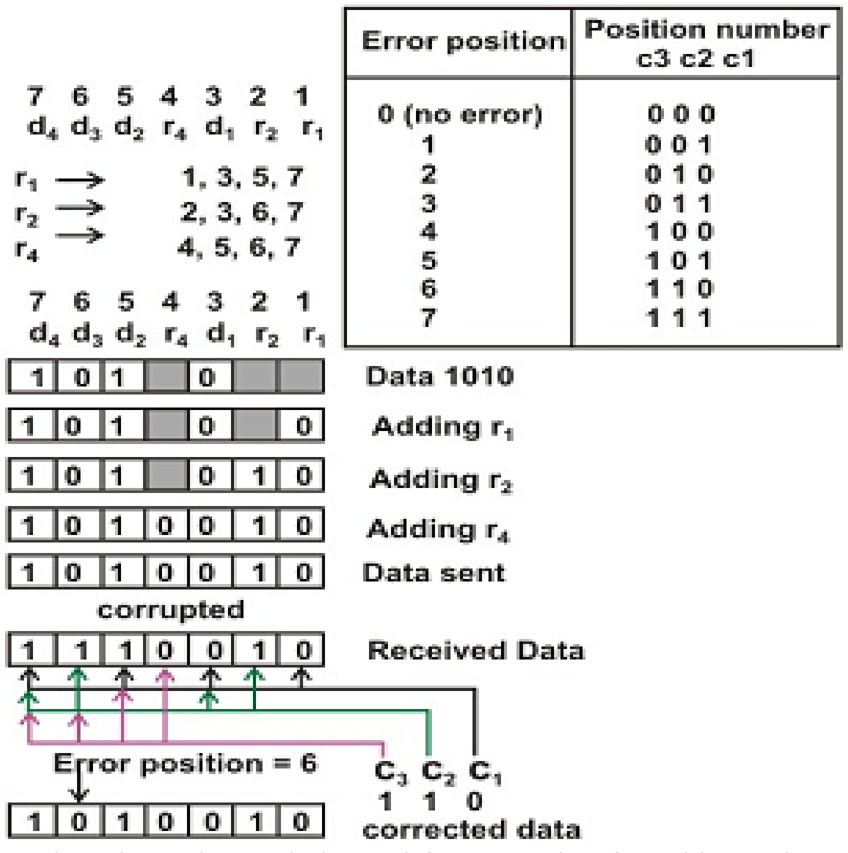
- Concept of error-correction can be easily understood by examining the simplest case of single-bit errors. As we have already seen that a single-bit error can be detected by addition of a parity bit (VRC) with the data, which needed to be send.
- A single additional bit can detect error, but it's not sufficient enough to correct that error too. For correcting an error one has to know the exact position of error, i.e. exactly which bit is in error (to locate the invalid bits). For example, to correct a single-bit error in an ASCII character, the error correction must determine which one of the seven bits is in error. To this, we have to add some additional redundant bits.
- To calculate the numbers of redundant bits (r) required to correct d data bits, let us find out the relationship between the two. So we have (d+r) as the total number of bits, which are to be transmitted; then r must be able to indicate at least d+r+1 different value. Of these, one value means no error, and remaining d+r values indicate error location of error in each of d+r locations. So, d+r+1 states must be distinguishable by r bits, and r bits can indicates 2^r states. Hence, 2^r must be greater than d+r+1. ($2^r >= d+r+1$)
- The value of r must be determined by putting in the value of d in the relation. For example, if d is 7, then the smallest value of r that satisfies the above relation is 4. So the total bits, which are to be transmitted is 11 bits (d+r = 7+4 =11).

Hamming Code

Now let us examine how we can manipulate these bits to discover which bit is in error. A technique developed by R.W.Hamming provides a practical solution. The solution or coding scheme he developed is commonly known as Hamming Code. Hamming code can be applied to data units of any length and uses the relationship between the data bits and redundant bits as discussed.

Basic approach for error detection by using Hamming code is as follows:

- To each group of m information bits k parity bits are added to form (m+k) bit code as shown in Figure.
- Location of each of the (m+k) digits is assigned a decimal value.
- The k parity bits are placed in positions 1, 2, …, 2 $^{k-1}$ positions.–K parity checks are performed on selected digits of each codeword.
- At the receiving end the parity bits are recalculated. The decimal value of the k parity bits provides the bit-position in error, if any.



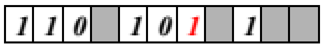
Error position	Position number c3 c2 c1
0 (no error)	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

Above figure shows how hamming code is used for correction for 4-bit numbers (d4 d3 d2 d1) with the help of three redundant bits (r4 r2 r1). For the example data 1010, first r1 (0) is calculated considering the parityof the bit positions, 1, 3, 5 and 7. Then the parity bits r2 is calculated considering bit positions 2, 3, 6 and 7. Finally, the parity bits r4 is calculated considering bit positions 4, 5, 6 and 7 as shown. If any corruption occurs in any of the transmitted code 1010010, the bit position in error can be found outby calculating r4 r2 r1 at the receiving end. For example, if the received code word is

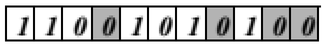
1110010, the recalculated value of r4 r2 r1 is 110, which indicates that bit position in error is 6, the decimal value of 110.

Example:

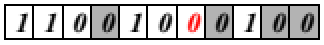
Let us consider an example for 5-bit data. Here 4 parity bits are required. Assume that during transmission bit 5 has been changed from 1 to 0 as shown in Figure. The receiver receives the code wordand recalculates the four new parity bits using the same set of bits used by the sender plus the relevantparity (r) bit for each set (as shown in Figure). Then it assembles the new parity values into a binary number in order of r positions (r8, r4, r2, r1).



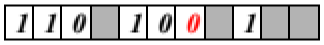
Data to be send



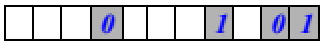
Data to be send along with redundant bits



Data Received



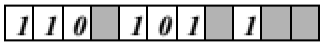
Data Received Minus Parity Bits



Parity bits recalculated

Parity recalculated (r8, r4, r2, r1) = 01012 = 510. Hence, bit 5
th
is in error i.e. d5 is in error.

So, correct code-word which was transmitted is:



Data Link Layer Error Detection and correction: Types of Errors, Detection, Error Correction Data Link Control and Protocols: Flow and Error Control, Stop-and-wait ARQ. Go-Back-N ARQ, Selective Repeat ARQ, HDLC. Point-to –Point Access: PPP Point –to- Point Protocol, PPP Stack, Multiple Access Random Access, Controlled Access, Channelization. Local area Network: Ethernet. Traditional Ethernet, Fast Ethernet, Gigabit Ethernet. Token bus, token ring Wireless LANs: IEEE 802.11, Bluetooth virtual circuits: Frame Relay and ATM