International Journal of
Advances in Scientific Research and Engineering (ijasre)

E-ISSN : 2454-8006

DOI: http://dx.doi.org/10.7324/IJASRE.2018.32674

Volume 4, Issue 4
April - 2018

Analytical Review of Error Control in Data Link Layer

Ezeagwu C. O [1] Nwokoye C.S [2] , Omeje A [3] and Njonu U.J [1]

Department of Electronics and Computer Engineering

Faculty of Engineering

Awka, Anambra

Nigeria

ABSTRACT

Error control portrays how the system handles and identifies blunders particularly in the information interface layer. In this paper, we exhibit a review of blunder control with respect to mistake discovery and mistake redress. Error control occurs in data link layer. We chiefly examine the kind of error detection systems that is utilized to recognize the mistakes and how the mistakes will be redressed so the collector can remove the genuine information. For most applications, a framework must ensure that the information got are indistinguishable to the information transmitted. Transmission media are prone to error. In a network, the capacity of nodes is different and the rate at which the sender is sending data might not be the same rate at which the receiver accepts it. In this paper, we discuss on designing issues of data link layer. The primary focus is on various error detecting and controlling mechanisms.

Key Words: *Error Detection, Error Control, Data link layer, CRC, Checksum.*

1. INTRODUCTION

The Data Link Layer is the second layer of the OSI Model. It transforms the physical layer, a raw transmission facility, to a link responsible for node-to-node (hop-to-hop) communication. The data link layer divides the stream of bits received from the network layer into manageable data units called frames. The data link layer adds a header to the frame to define the addresses of the sender and receiver of the frame.
At the receiving end, Data link layer picks up data from hardware which are in the form of electrical signals assembles them in a recognizable frame format, and hands over to upper layer [1][2]. Specific responsibilities of the data link layer include framing, addressing, flow control, error control, synchronization and media access control.

Framing: Data-link layer takes packets from Network Layer and encapsulates them into Frames. Then sends each Frame bit-by-bit on the hardware. At receiver's end Data link layer picks up signals from hardware and assembles them into frames.

Addressing: Data-link layer provides layer-2 hardware addressing mechanism. Hardware address is assumed to be unique on the link. It is encoded into hardware at the time of manufacturing.

Flow Control: Stations on same link may have different speed or capacity. Data-link layer ensures flow control that enables both machines to exchange data on same speed.

Error Control: Sometimes signals may have encountered problem in transition and bits are flipped. These errors are detected and attempts are made to recover actual data bits. It also provides error reporting mechanism to the sender.

Synchronization: When data frames are sent on the link, both machines must be synchronized in order for transfer to take place.

Media-Access Control: Hosts on shared link when tries to transfer data, has great probability of collision. Data-link layer provides mechanism like CSMA/CD to equip capability of accessing a shared media among multiple Systems. [3]

2. ERROR CONTROL

Networks must be able to transfer data from one device to another with acceptable accuracy.
For most applications, a system must guarantee that the data received are identical to the data transmitted. Any time data are transmitted from one node to the next, they can become corrupted in passage. Many factors can alter one or more bits of a message. Some applications require a mechanism for detecting and correcting errors. This calls for error control. Error control includes both error detection and error correction. It allows the receiver to inform the sender if a frame is lost or damaged during transmission and coordinates the retransmission of those frames by the sender. Error control in the data link layer is based on automatic repeat request (ARQ). Whenever an error is detected, specified frames are retransmitted [4][5].

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. There are two types of error namely:

1. **Single-bit Error:** 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 1 µs, which is very rare because noise normally last much longer than 1 µs.

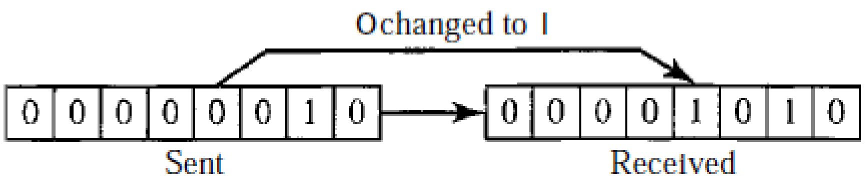

Figure 1: Single bit error [5]

2. **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. Burst errors are most likely to occur than single-bit error. 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. The number of bits that are corrupted always depends on the data rate and duration of noise [6].

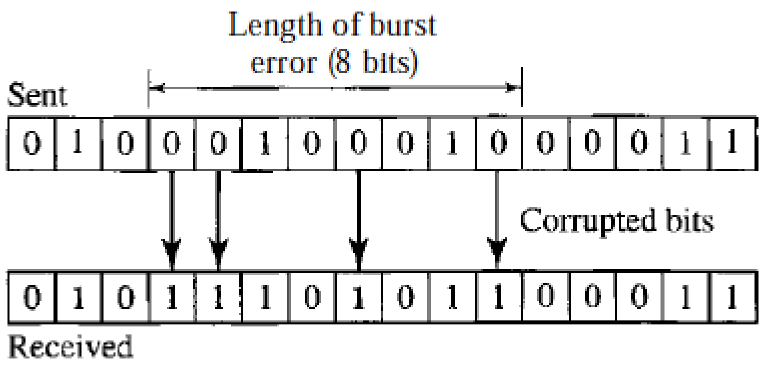

Figure 2: Burst Error [4]

2.1.0. Error Detection Versus Error Correction

Error correction 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. While 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. For instance, 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. This gets tedious as the number of error bits increases.

2.1.1. Error Detection

When transmitting a bit stream over a transmission line or channel a scheme is normally incorporated into the transmission control circuit to enable the presence of bits or transmission error in the receiving block to be detected. In general, this is done by the transmitter which computes a set of additional bits based on the contents of blocks of bits to be transmitted. This is known as error detection which is based on a block of extra bits which is transmitted together with the original bits in the block. The receiver uses the complete sets of received bits to determine whether the block contains any error to a high probability [7]. The two factors that determine the type of error detection scheme used are the bit error rate (BER) probability of the line and the type of error, that whether the errors occur as random single-bit errors or as groups of continuous of bit errors (burst error).

The different type of error detection schemes detects different type of errors. Also the number of bits used in some schemes determines the burst lengths that are detected. The three most widely used schemes are parity, cyclic redundancy checks (CRC) and checksum.

Parity Check: The most common method for detecting bits error with asynchronous character and character-oriented synchronous transmission is parity bit method. There are two types of parity check schemes: even and odd parity checks [8]. With the even parity check, the redundant bit is chosen so that an even number of bits are set to one in the transmitted bit string of N+r bits, where r is the bit that used to be the even parity check and N is the bit that is transmitted by the transmitter of the network. The receiver re-computes the parity of each received bits from the transmitter and discard the strings with the invalid parity. The parity scheme is always used if 7-bits character is exchanged. If there are 7-bits that are transmitted by the transmitter and parity check are used to detect the error, the eight bit is often the parity bit.

Table 1: Illustration of Parity bit detection scheme [4]

3 bits string	Odd parity	Even parity
000	1	0
001	0	1
010	0	1
100	0	1
110	1	0
111	1	0

Table 1 shows a table contain 3 bits string. The transmitter will add 0 or 1 to the bits string according to the parity check mechanism (even or odd). When the receiver receives the bits string, the receiver will use the same mechanism to count the 1's in the bit string to determine whether it matches the counted parity from the transmitter or not. For example if the bits string transmitted is 000, for odd parity check, the transmitter will add 1 at the end of bits string so it will transmitted 0001 to the receiver. Then the receiver will count every single bit using the same mechanism for parity check. If the 1's in the receive bits string is odd, the bits string will be accept, otherwise the bits string will be rejected.

Parity check mechanism can detect all single-bit errors. It can also detect burst errors only if the total number of errors in each data unit is odd/even (based on parity check used). For example, even parity check mechanism cannot detect errors where the total number of bits changed is even. If any two bits change in transmission, the changes cancel each other and the data unit will pass a parity check even though the data unit is damaged. The same holds true for any further even number of errors.

Cyclic Redundancy Check (CRC): CRC is based on the 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. On the destination side, the incoming data the binary data is divided by the same number to be compared on the source side. This means that, if the remainder of the division is same as the value on the added CRC when the data was transmitted, the data will be accepted, otherwise the unmatched reminder produced on the destination after the CRC indicates that the data unit has been damage during the transmission of data [5]. The redundancy bits used by CRC are derived by dividing the data unit by a predetermined divisor; the remainder is the CRC. To be valid, a CRC must satisfy two conditions: It must have exactly one less bit than the divisor and appending it to the end of the data string must make the resulting bit sequence exactly divisible by the divisor.

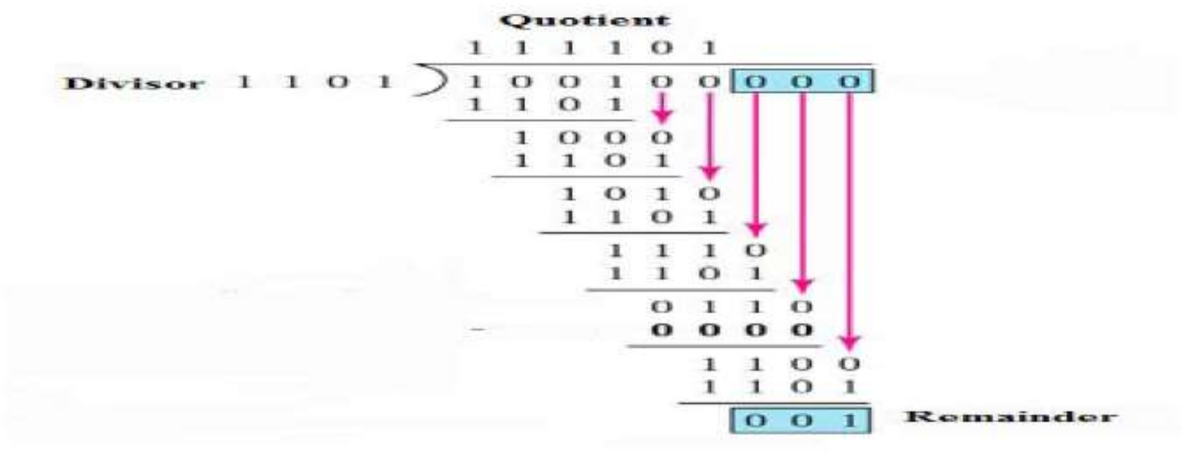

Figure 3: CRC operation at sender site

Figure 3, shows the outline basic operation of the CRC. It is as follows:

i. The string of n bits is added to the data unit.
ii. The newly binary data unit is divided by the divisor p, combination of n + 1 bits, called binary division. The remainders resulting from this division is the CRC (n bits).
iii. The CRC value resulting from the second step is replaced with the value of n string. Note that the CRC may consist of all 0’s. The binary data unit arrives to the receiver followed by the CRC. The receiver treats the incoming binary data as one data block and divides it with the same divisor that is used to get the CRC value. If the data arrived without an error, the CRC checks return all the value which is zero and the binary data unit is passed [5]. As shown in fig. 3, the data bits plus extra zero that is added to the data string is divided with the divisor. The remainder of the division will be the value of CRC that will replace the data plus extra zeros at the receiver side.

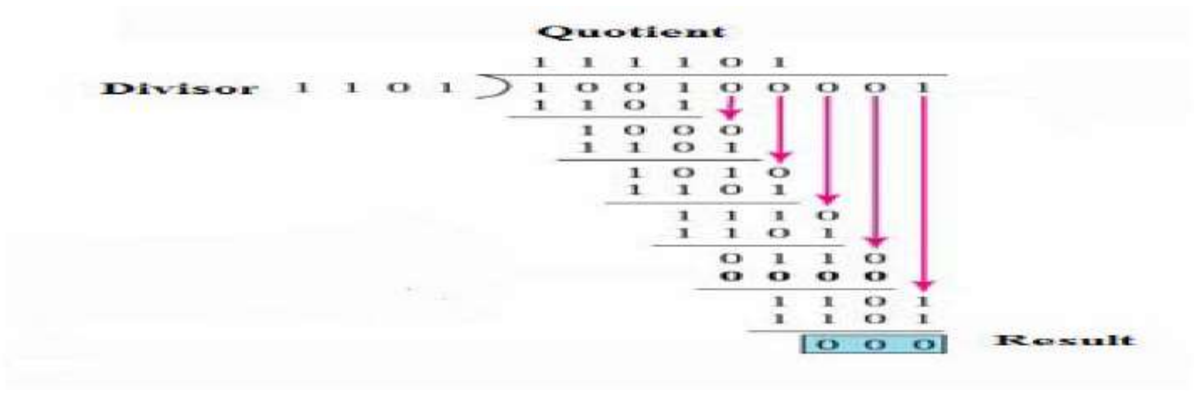

Figure 4: CRC operation at receiver site

Figure 4 shows the calculation for the CRC in the receiver side. At the receiver side, the data string and the CRC value is divided by the same value of divisor in the sender part. Then the remainder of this division determines whether the received data bit string will be accepted or not. If the remainder is zero, then the data will be accepted, else it will be rejected. CRC has a very good performance in detecting single-bit errors, double errors, an odd number of errors and burst errors. They can easily be implemented in hardware and software. They are especially fast when implemented in hardware. This has made CRC a good candidate for many networks [9].

CHECKSUM: The checksum is used in the Internet by several protocols although not at the data link layer. Like linear and cyclic codes, the checksum is based on the concept of redundancy. The checksum detects all errors involving an odd detection. The concept of the checksum is not difficult. As shown in fig. 5, in the sender, the checksum generator subdivides the data unit into equal segments of n bits (usually 16). These segments are added using ones complement arithmetic in such a way that the total is also n bits long. That total (sum) is then complemented and appended to the end of the original data unit as redundancy bits, called the checksum field. The extended data unit is transmitted across the network. So if the sum of the data segments is T, the checksum will be -T. The receiver performs the same calculation on the received data and compares the result with the received checksum. If the result is 0, the receiver keeps the transmitted data; otherwise, the receiver knows that an error occurred discards the transmitted data [8][3].

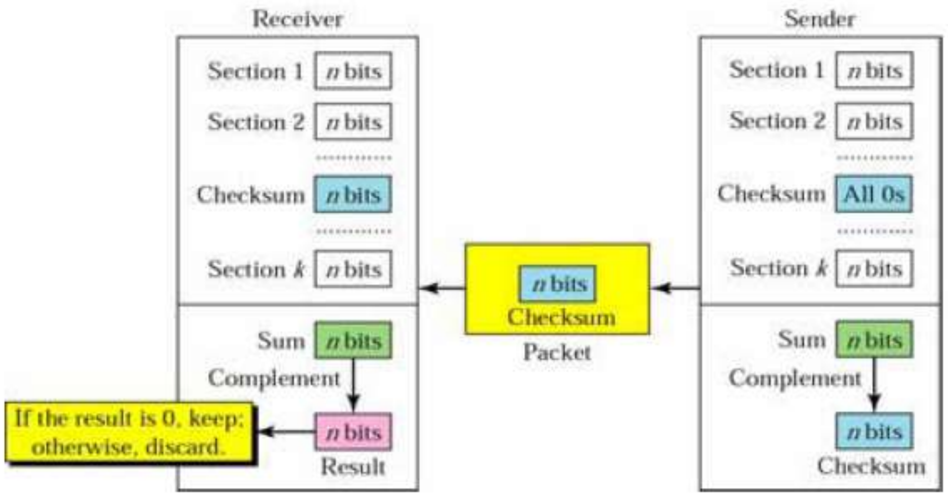

Figure 5: Checksum illustration

The traditional checksum is not as strong as the CRC in error-checking on capability. For example, if the value of one word is incremented and the value of another word is decremented by the same amount, the two errors cannot be detected because the sum and checksum remain the correcting and burst error locating codes, in same. Also if the values of several words are incremented but the total change is a multiple of the sum and the checksum does not change, which means the errors are not detected. Fletcher and Adler [9][10] have proposed some weighted checksums, in which each word is multiplied by a number (its weight) that is related to its position in the text. This will eliminate the first problem we mentioned. However, the tendency in the Internet, particularly in designing new protocols, is to replace the checksum with a CRC [11][3]. The Fletcher checksum and the later Adler checksum are both designed to give error detection properties almost as good as CRCs with significantly reduced computational cost. [3]

2.1.2. Error Correction

As earlier noted, 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. More redundant bits are needed for error correction than for error detection. In this section, various error correction schemes are used.

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 Is in the result. Note that the Hamming distance is a value greater than zero. For example, The Hamming distance between two pair of words *d(OOO,* 011) is 2 because 000 $\oplus$ 011 is 011 (two Is) [6].

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 dmin to define the minimum Hamming distance in a coding scheme. To find this value, we find the Hamming distances between all words and select the smallest one.

Table 2: Codes for illustrating error correction

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

For instance, the minimum Hamming distance of the coding scheme in Table 2 is calculated by finding
all the Hamming distances. They are:

d(OOOOO, 01011) = 3, *d(01011,* 10101) =4, *d(OOOOO,* 10101) =3
d(OlO11, 11110) = 3, *d(OOOOO,* 11110) = 4, d(10101, 11110) =3

The *dmin* in this case is 3.
When a codeword is corrupted during transmission, the Hamming distance between the sent and received codeword's is the number of bits affected by the error. In other words, the Hamming distance between the received codeword and the sent codeword is the number of bits that are corrupted during transmission. If s errors occur during transmission, the Hamming distance between the sent codeword and received codeword is s. If our code is to detect up to s errors, the minimum distance between the valid codes must be s + 1, so that the received codeword does not match a valid codeword. In other words, if the minimum distance between all valid codeword's is s + 1, the received codeword cannot be erroneously mistaken for another codeword. The distances are not enough (s + 1) for the receiver to accept it as valid. The error will be detected. We need to clarify a point here: Although a code with *dmin* =s + 1 may be able to detect more than s errors in some special cases, only s or fewer errors are guaranteed to be detected [6]
.

Hamming Codes
Now let us discuss a category of error-correcting codes called Hamming codes. These codes were originally designed with *dmin* = 3, which means that they can detect up to two errors or correct one single error. Although there are some Hamming codes that can correct more than one error, our discussion focuses on the single-bit error-correcting code. First let us find the relationship between *n* and *k* in a Hamming code. We need to choose an integer *m* >= 3. The values of n and *k* are then calculated from m as *n* = *2m* – 1 and *k* ::: *n* - *m.* The number of check bits *r* = *m.*
For example, if *m* = 3, then *n* ::: 7 and *k:::* 4. This is a Hamming code *C(7,* 4) with dmin =3.
Table 10.4 shows the data words and codeword's for this code [6].

A Hamming code can only correct a single error or detect a double error. However, there is a way to make it detect a burst error. The key is to split a burst error between several codeword's, one error for each codeword. In data communications, we normally send a packet or a frame of data. To make the Hamming code respond to a burst error of size *N*, we need to make *N* codeword's out of our frame. Then, instead of sending one codeword at a time, we arrange the codeword's in a table and send the bits in the table a column at a time [6].

3.0 CONCLUSION

Various methods of error correction and detection in the data link layer has been analyzed, not all the methods of error detection can detect error accurately and effectively. Each scheme discussed has their advantage and their own mechanism to detect error. Parity check is simple and can detect all single-bit error. CRC has a very good performance in detecting single-bit errors, double errors, an odd number of errors and burst errors while checksum is not efficient as the CRC in error detection when the two words are incremented with the same amount, the two errors cannot be detected because the sum and checksum remain the same.

REFERENCE

1. Chellis, J., C. Perkins and M. Strebe, 1999. MCSE: ed.: New Riders Publishing.
2. Kaise, T. and M. Kitakami, 2002 Single-bit error Information Theory, 2002. Proceedings. 2002 IEEE 65535 International Symposium on, pp: 117.
3. Monika Singh & Ruhi Saxena, “Data Link Layer Designing Issues: Error Control—A Roadmap”, Global Journal of Computer Science and Technology: C Software & Data Engineering Volume 14 Issue 8 Version 1.0 Year 2014
4. Wu, K., H. Tan, Y. Liu, J. Zhang, Q. Zhang and L.M. Ni, 2012: Side channel interference, Mobile Computing, IEEE Transactions 11: 1317-1330.
5. Behrouz, F. and M. Firouz, 2012. Computer Networks: A Top Down Approach, 1st International ed.: McGraw-Hill.
6. Behrouz A. Forouzan, 2007 “Data Communications and Networking”, Fourth ed.:McGraw-Hill.
7. Halsall, F., 2006. Computer Networking and the Internet, 5/e: Pearson Education India.
8. Bonaventure, O., 2011. Computer Networking: Principles, Protocols and Practice: The Saylor Foundation.
9. Fletcher, J., 1982. An arithmetic checksum for serial transmissions," Communications, IEEE Transactions on, 30: 247-252.
10. Gailly. J. and P. Deutsch, 1996. Zlib compressed data format specification version 3.3,Network Working Group Request for Comments (RFC), http://tools.ietf.org/html/rfc1950.
11. Hossein BerenjeianTabrizi, Ali Abbasi and HajarJahadianSarvestani, 2013. Comparing the Static and Dynamic Balances and Their Relationship with the Anthropometrical Characteristics in the Athletes of Selected Sports, Middle-East Journal of Scientific Research, 15(2): 216-221.

DOI: dx.doi.org/10.7324/IJASRE.2018.32674