UNIT-III

Objectives:

- Study the design principles of the second layer that is Data link Layer. This study deals with algorithms for achieving the reliable, efficient communication of whole units of information called frames between two adjacent machines.
- Understanding of the How the Data link Layer Perform Flow and Error control methods.
- Basic Knowledge of Elementary Protocols Supported by Data link Layer.

Syllabus:

Data Link Layer Design issues, Framing - fixed size framing, variable size framing,Error control, Error detection, Error correction, CRC, Checksum, hamming code. Elementary Data link Layer protocols - Unrestricted Simplex protocol, Simplex Stop-and-Wait Protocol, Simplex protocol for Noisy Channel. Sliding window protocol-one bit, Go back N, selective repeat, data link in HDLC, PPP.

Outcomes:

Students will be able to

- ➢ Different Framing Methods used by data Link Layer
- ➢ Error Detection by Simple parity check, 2-Dimensional parity check, CRC, Checksum.
- ➢ Error Correction by Codes like Hamming Code
- ➢ What type of Services provided to network layer
- ➢ Types of Noisy channels protocols

Learning Material

What is DLL (Data Link Layer)?

The Data Link Layer is the second layer in the OSI model, above the Physical Layer, which ensures that the error free data is transferred between the adjacent nodes in the network. It breaks the datagram passed down by above layers and converts them into frames ready for transfer. This is called **Framing.**

It provides two main functionalities

- Reliable data transfer service between two peer network layers
- Flow Control mechanism which regulates the flow of frames such that data congestion is not there at slow receivers due to fast senders.

THE DATA LINK LAYER DESIGN ISSUES

(1) **Framing.**

(2) **Error Control.**

(3) **Flow Control.**

(1) FRAMING

The frame contains

1. Frame header
2. Payload field for holding packet
3. Frame trailer

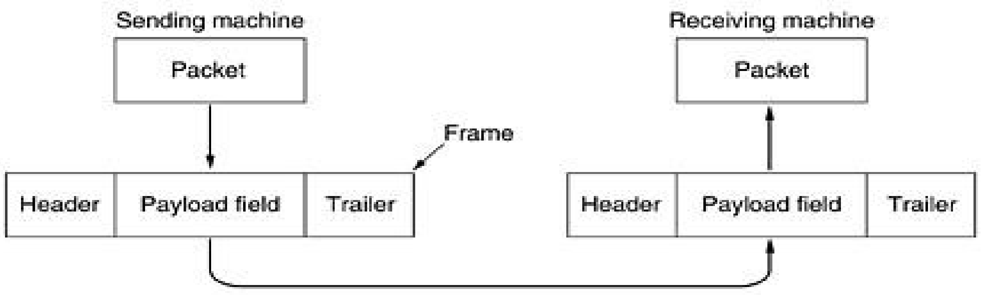


Figure 1.1 Relationships between Packets and Frames

Services provided to the network layer

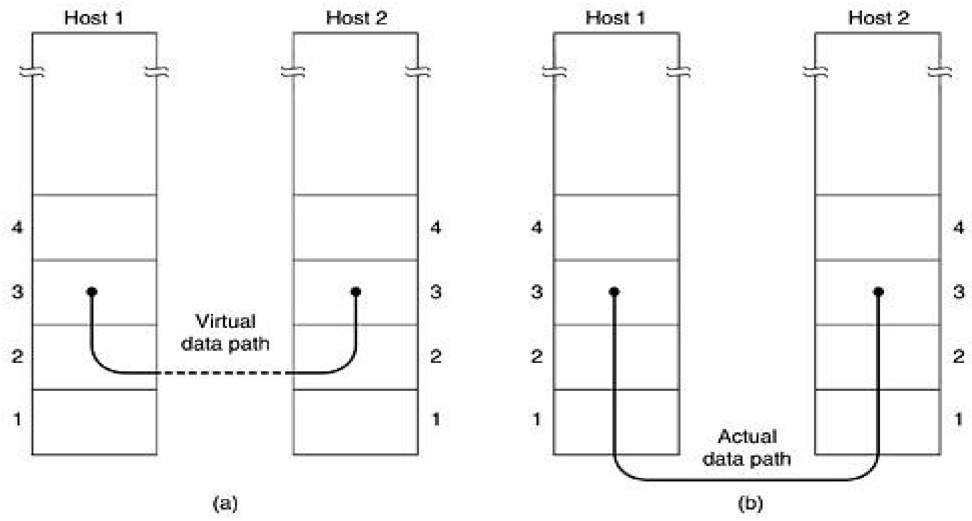


Figure 1.2 (a) Virtual communication. (b) Actual communication

Transferring data from the network layer on the source machine to the network layer on the destination machine. The data link layer can be designed to offer various services. The actual services offered can vary from system to system. Three reasonable possibilities that are commonly provided are

1. **Unacknowledged connectionless service**

   - Source machine sends independent frames to destination machine having destination machine acknowledge them
   - No logical connection
   - Used when error rate is very low
   - Good for real-time traffic (voice)

2. **Acknowledged connectionless service**

   - No logical connection
   - Each frame sent is individually acknowledged
   - Useful over unreliable channels (i.e. wireless systems)

3. Acknowledged connection-oriented service

- Source and destination machines establish a connection before any data are transferred
- Each frame is numbered
- DLL guarantees that...
    - Each frame is received
    - Each frame is received exactly once
    - Each frame is received in the right order

3 PHASES

When connection-oriented service is used, transfers go through three distinct phases

1. Connection established
2. Frames are transmitted
3. Connection released

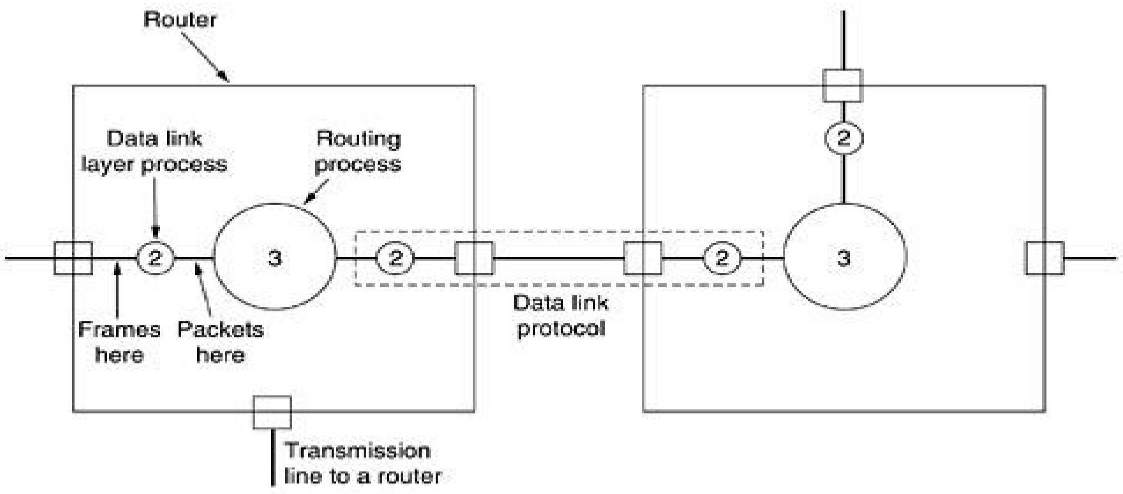


Figure 1.3 Placement of the data link Protocol

- Consider a typical example: a WAN subnet consisting of routers connected by point-to-point leased telephone lines.
- When a frame arrives at a router, the hardware checks it for errors, and then passes the frame to the data link layer software.

- The data link layer software checks to see if this is the frame expected, and if so, gives the packet contained in the payload field to the routing software.
- The routing software then chooses the appropriate outgoing line and passes the packet back down to the data link layer software, which then transmits it. The flow over two routers is shown in **Fig. 1-3.**

FRAMING

Breaking the bit stream up into frames is more difficult than it at first appears. One way to achieve this framing is to insert time gaps between frames, much like the spaces between words in ordinary text. However, networks rarely make any guarantees about timing, so it is possible these gaps might be squeezed out or other gaps might be inserted during transmission.

There are four methods:

1. Character count.
2. Flag bytes with byte stuffing.
3. Starting and ending flags, with bit stuffing.
4. Physical layer coding violations.

Character count:

The first framing method uses a field in the header to specify the number of characters in the frame. When the data link layer at the destination sees the character count, it knows how many characters follow and hence where the end of the frame is. This technique is shown in Fig. 3-4(a) for four frames of sizes 5, 5, 8, and 8 characters, respectively.

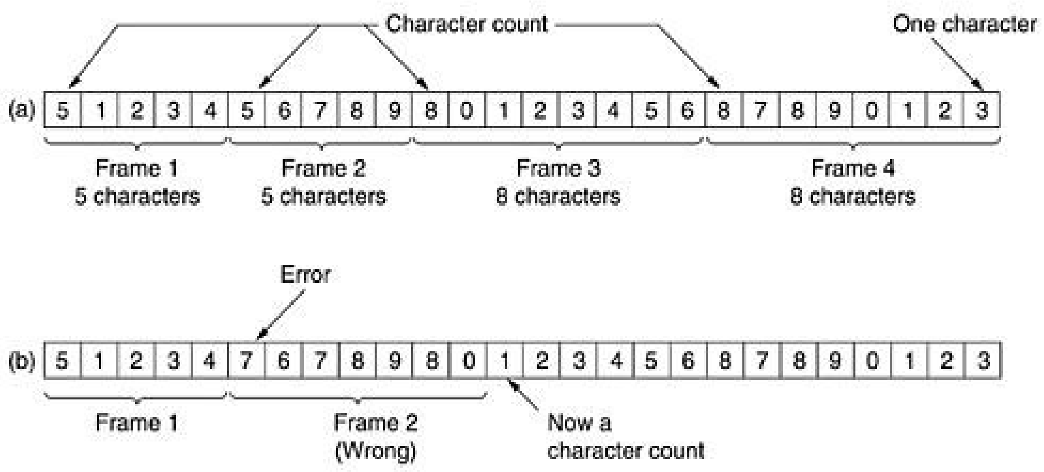


Figure 3-4. A character stream. (a) Without errors. (b) With one error.

Explanation (Figure 3-4.(a) A character stream Without errors.)

- The first framing method uses a field in the header to specify the number of characters in the frame.
- When the data link layer at the destination sees the character count, it knows how many characters follow and hence where the end of the frame is.
- This technique is shown in Fig. 3-4(a) for four frames of sizes 5, 5, 8, and 8 characters, respectively.
- The trouble with this algorithm is that the count can be garbled by a transmission error.

Explanation (Figure 3-4.(b) A character stream with errors.)

- For example, if the character count of 5 in the second frame of Fig. 3-4(b) becomes a 7, the destination will get out of synchronization and will be unable to locate the start of the next frame.
- Even if the checksum is incorrect so the destination knows that the frame is bad, it still has no way of telling where the next frame starts.
- Sending a frame back to the source asking for a retransmission does not help either, since the destination does not know how many characters to skip over to get to the start of the retransmission. For this reason, the character count method is rarely used

anymore.

Flag bytes with byte stuffing:

Character-oriented framing approach

- In a character-oriented approach, data to be carried are 8-bit characters.
- The header, which normally carries the source and destination addresses and other control information.
- Trailer carries error detection or error correction redundant bits, are also multiples of 8 bits.
- To separate one frame from the next, an 8-bit (1-byte) flag is added at the beginning and the end of a frame.
- The flag, composed of protocol-dependent special characters, signals the start or end of a frame.

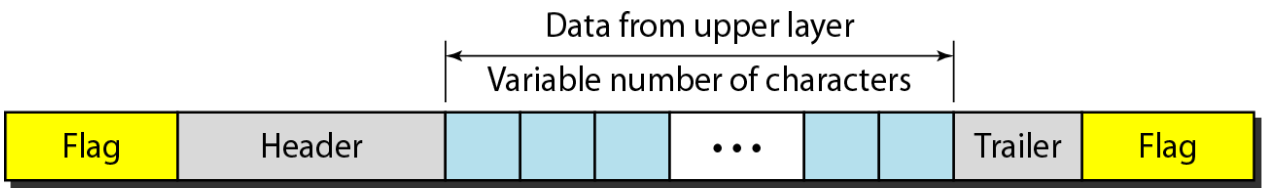


Figure: shows the format of a frame in a character-oriented protocol

Advantage:

1. Simple framing method.
2. Character-oriented framing was popular when only text was exchanged by the data Link layers.
3. The flag could be selected to be any character not used for text communication.

Disadvantage:

1. Even if with checksum, the receiver knows that the frame is bad there is no way to tell where the next frame starts.
2. Asking for retransmission doesn't help either because the start of the retransmitted frame is not known.
3. Hence No longer used.

Starting and ending character with byte stuffing

Byte stuffing is the process of adding 1 extra byte whenever there is a flag or escape character in the text.

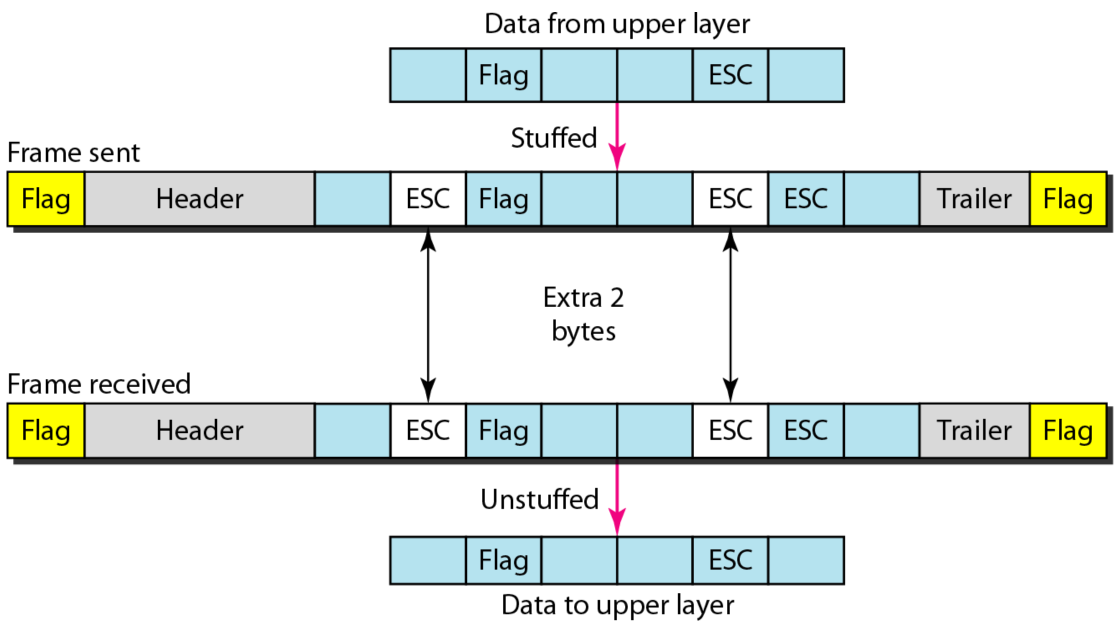


Figure : Byte stuffing and unstuffing

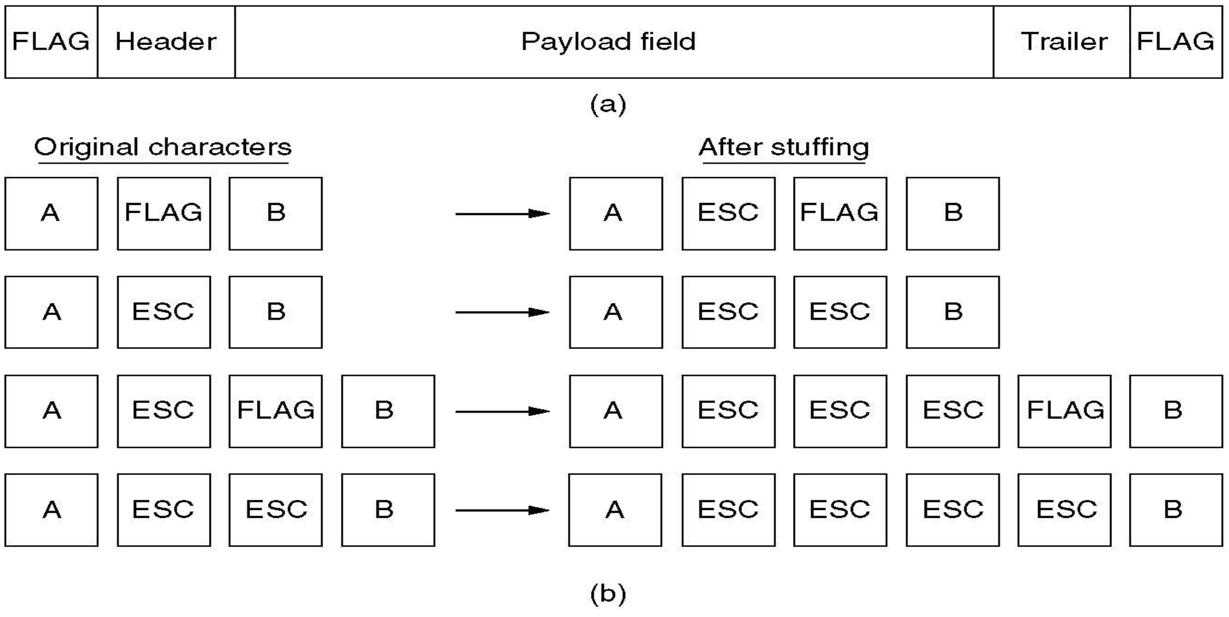


Fig: Framing with byte stuffing

Problem: fixed character size: assumes character size to be 8 bits: can't handle heterogeneous environment.

Bit-Oriented framing approach

- ➢ Bit stuffing is the process of adding one extra 0 whenever five consecutive 1's follow a 0 in the data, so that the receiver does not mistake the pattern 0111110 for a flag.
- ➢ Most protocols use a special 8-bit pattern flag 01111110 as the delimiter to define the beginning and the end of the frame, as shown in Figure below
- ➢ This flag can create the same type of problem. That is, if the flag pattern appears in the data, we need to somehow inform the receiver that this is not the end of the frame.
- ➢ We do this by stuffing 1 single bit (instead of I byte) to prevent the pattern from looking like a flag. The strategy is called bit stuffing.

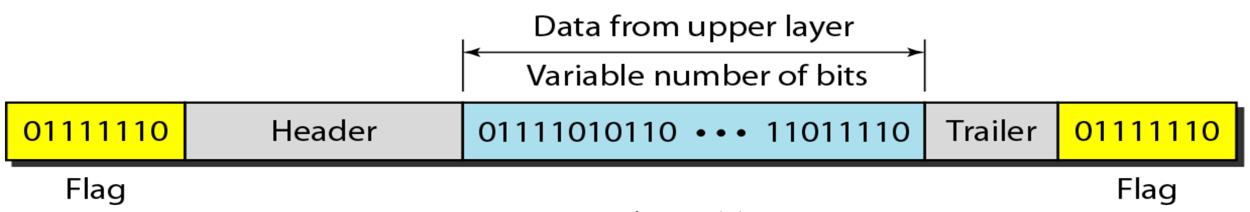
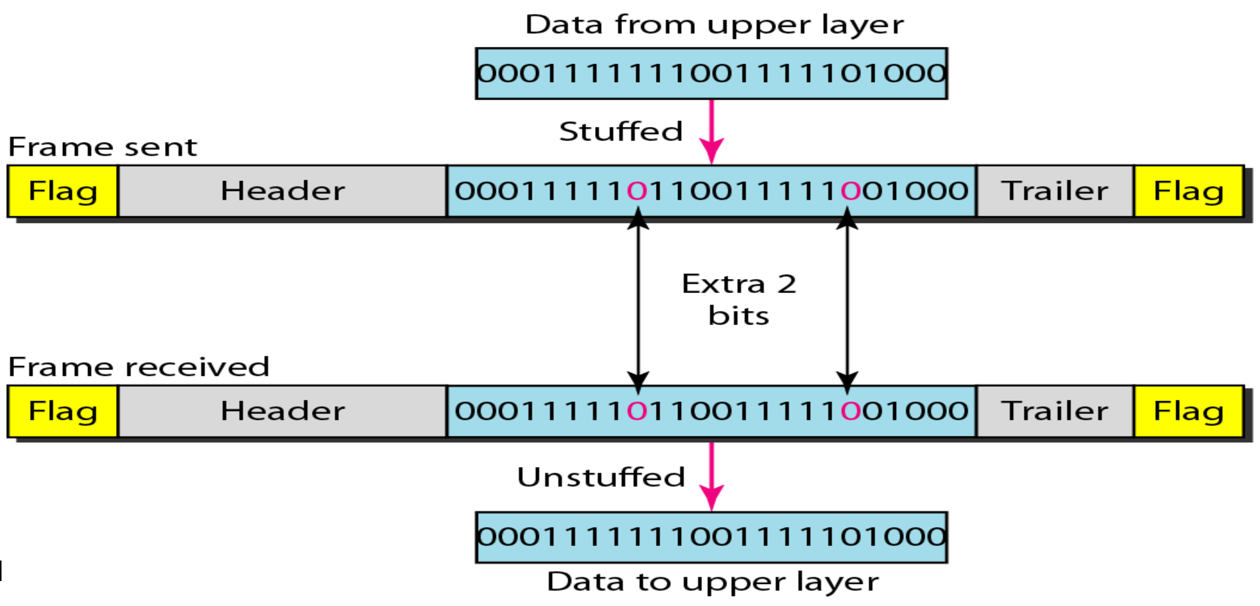


Figure (a)

Bit stuffing is the process of adding one extra 0 whenever five consecutive 1s follow a 0 in the data, so that the receiver does not mistake the pattern 0111110 for a flag.



III

Figure (b)

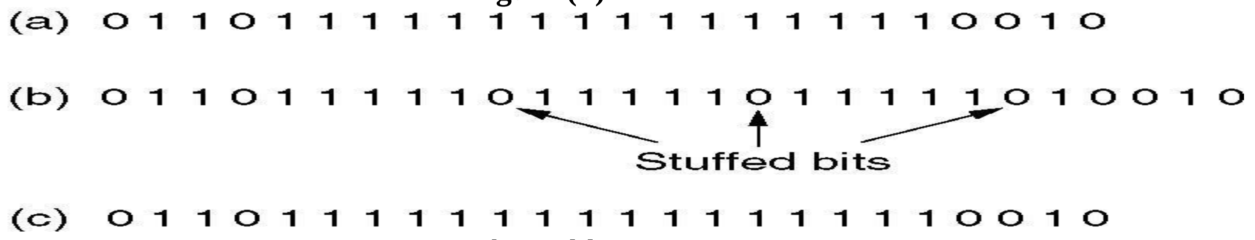

(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0
(b) 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0
Stuffed bits
(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Figure (c)

(a) The original data.
(b) The data as they appear on the line.
(c) The data as they are stored in receiver's memory after destuffing.

Physical layer coding violation:

The last method of framing is only applicable to networks in which the encoding on the physical medium contains some redundancy

. For example, some LANs encode 1 bit of data by using 2 physical bits. Normally, a 1 bit is a high- low pair and a 0 bit is a low-high pair. The scheme means that every data bit has a transition in the middle, making it easy for the receiver to locate the bit boundaries. The combinations high-high and low-low are not used for data but are used for delimiting frames in some protocols.

As a final note on framing, many data link protocols use a combination of a character count with one of the other methods for extra safety. When a frame arrives, the count field is used to locate the end of the frame. Only if the appropriate delimiter is present at that position and the checksum is correct is the frame accepted as valid. Otherwise, the input stream is scanned for the next delimiter.

(2) ERROR CONTROL

- How do we make sure that all frames are eventually delivered to the network layer at the destination and in the proper order?
- Provide sender with some acknowledgement about what is happening with the receiver
- Sender could wait for acknowledgement

Disadvantages

- If a frame vanishes, the receiver will not send an acknowledgement thus, sender will wait forever

- Dealt with by timers and sequence numbers – important part of DLL
- Sender transmits a frame, starts a timer.
- Timer set to expire after interval long enough for frame to reach destination, be processed, and have acknowledgement sent to sender
- Is a danger of frame being transmitted several times, however dealt with by assigning sequence numbers to outgoing frames, so that receiver can distinguish retransmissions from originals.

(3) FLOW CONTROL

What do we do when a sender transmits frames faster than the receiver can accept them?

- **Feedback-based flow control** – receiver sends back information to the sender, giving it permission to send more data or at least telling the sender how the receiver is doing
- **Rate-based flow control** – the protocol has a built-in mechanism that limits the rate at which the sender may transmit data, using feedback from the receiver.

ERROR DETECTION AND CORRECTION METHODS

- Because of Attenuation, distortion, noise and interferences, errors during transmission are inevitable, leading to corruption transmitted bits.
- Longer the frame size and higher the probability of single bit error, lower is the probability receiving a frame without error.

ERROR

- When data is being transmitted from one machine to another, it may possible that data become corrupted on its way. Some of the bits may be altered, damaged or lost during transmission. Such a condition is known as **error**.

TYPES OF ERRORS

- **Single bit error**: Only one bit gets corrupted. Common in Parallel transmission.
- **Burst error:** More than one bit gets corrupted very common in serial transmission of data occurs when the duration of noise is longer than the duration of one bit.

Single bit error:

- The term single-bit error means that only one bit of given data unit (such as a byte, character, or data unit) is changed from 1 to 0 or from 0 to 1 as shown in Fig. 3.2.1.
- Single bit errors are least likely type of errors in serial data transmission.
- For example, if 16 wires are used to send all 16 bits of a word at the same time and one of the wires is noisy, one bit is corrupted in each word.

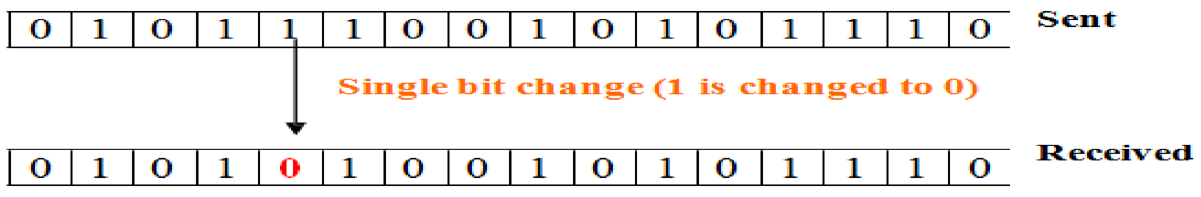


Figure 3.2.1 Single bit error

Burst error:

- More than one bit gets corrupted very common in serial transmission of data occurs when the duration of noise is longer than the duration of one bit.
- The noise affects data; it affects a set of bits.
- The number of bits affected depends on the data rate and duration of noise.

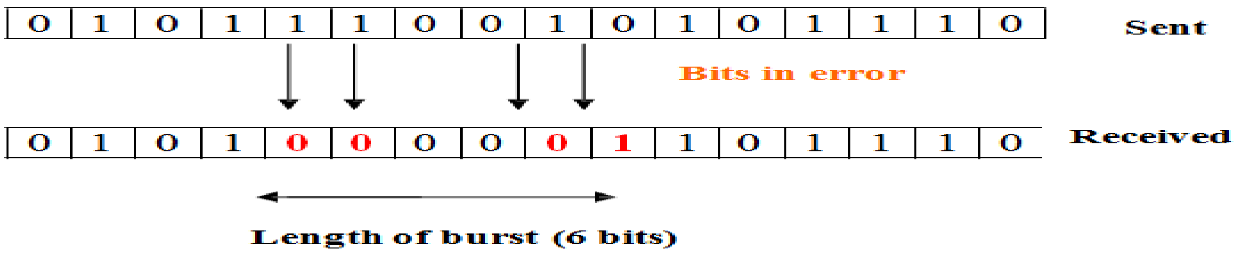


Figure 3.2.2 Burst Error

<u>ERROR DETECTION TECHNIQUES</u>

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

- Simple Parity check
- Two-dimensional Parity check
- Checksum
- Cyclic redundancy check

Redundancy is the method in which some extra bits are added to the data so as to check whether the data contain error or not.

m - data bits (i.e., message bits) r - redundant bits (or check bits). n - total number of bits

n= (m + r).

An n-bit unit containing data and check-bits is often referred to as an n-bit codeword.

SIMPLE PARITY CHECK

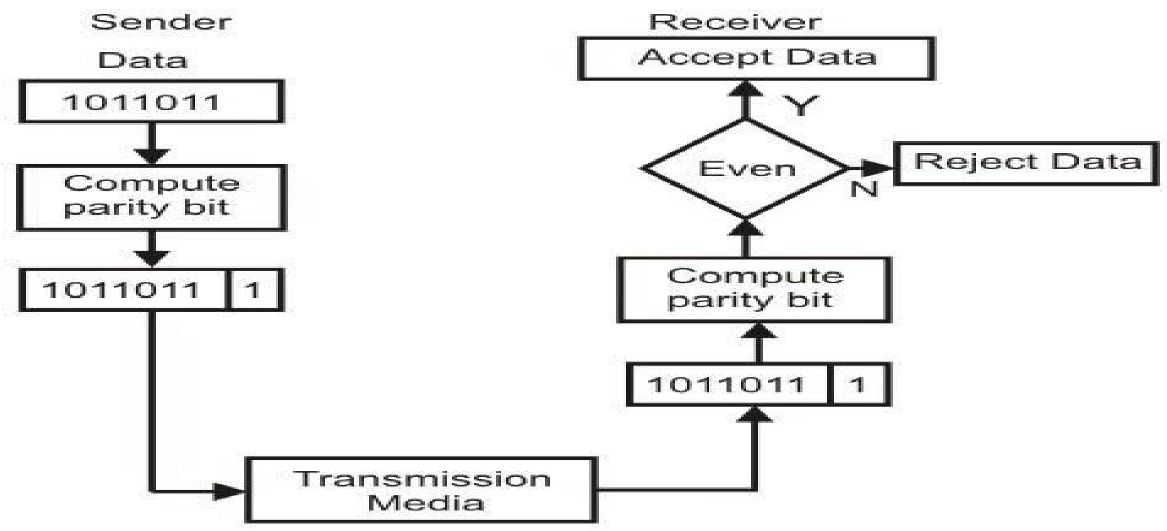
The simplest and most popular error detection scheme. Appends a Parity bit to the end of the data.

Even Parity: **01000001 –** Number of ones in the group of bits is even

Odd Parity: **11000001 -** Number of ones in the group of bits is odd

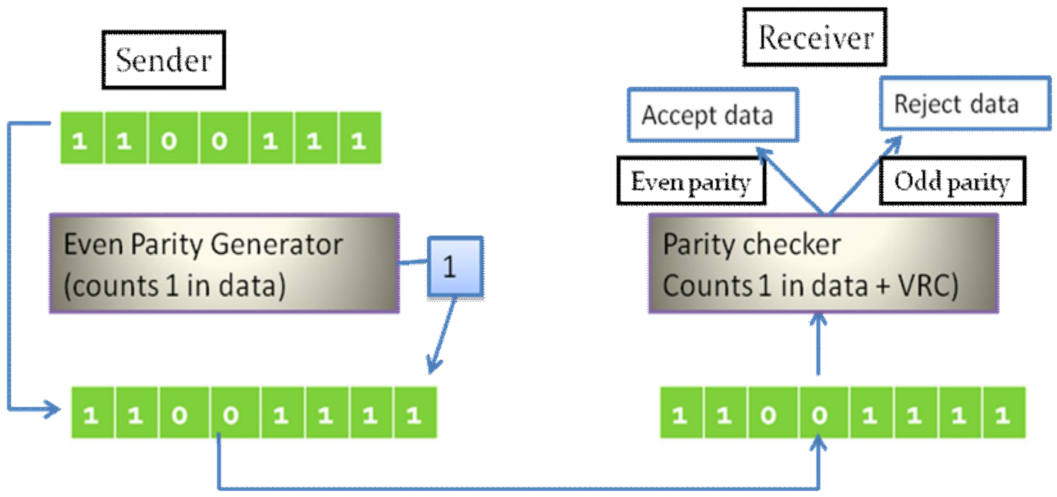
A parity of 1 is added to the block if it contains an odd number of 1's (ON bits) and 0 is added if it contains an even number of 1's. At the receiving end the parity bit is computed from the received data bits and compared with the received parity bit.

This scheme makes the total number of 1's even, that is why it is called *even parity checking*. Considering a 4-bit word, different combinations of the data words and the corresponding code words are given in Table 3.2.1.



Decimal value	Data Block	Parity bit	Code word
0	0000	0	00000
1	0001	1	00011
2	0010	1	00101
3	0011	0	00110
4	0100	1	01001
5	0101	0	01010
6	0110	0	01100
7	0111	1	01111
8	1000	1	10001
9	1001	0	10010
10	1010	0	10100
11	1011	1	10111
12	1100	0	11000
13	1101	1	11011
14	1110	1	11101
15	1111	0	11110

Example:



PERFORMANCE OF SIMPLE PARITY CHECK

- Simple parity check can **detect all single-bit error**
- It can also detect burst error, if the number of bits in **even or odd.**
- The technique is not foolproof against burst errors that **invert more than one bit**. If an even number of bits is inverted due to error, the **error is not detected.**

TWO-DIMENSION PARITY CHECKING

- Performance can be improved by using two dimensional parity check, which **organizes the block of bits in the form of table.**
- Parity check bits are **calculated from each row**, which is equivalent to a simple parity check.
- Parity check bits are also **calculated for all columns.**
- Both are sent along with the data.
- At the receiving end these are compared with the parity bits calculated on the received data.

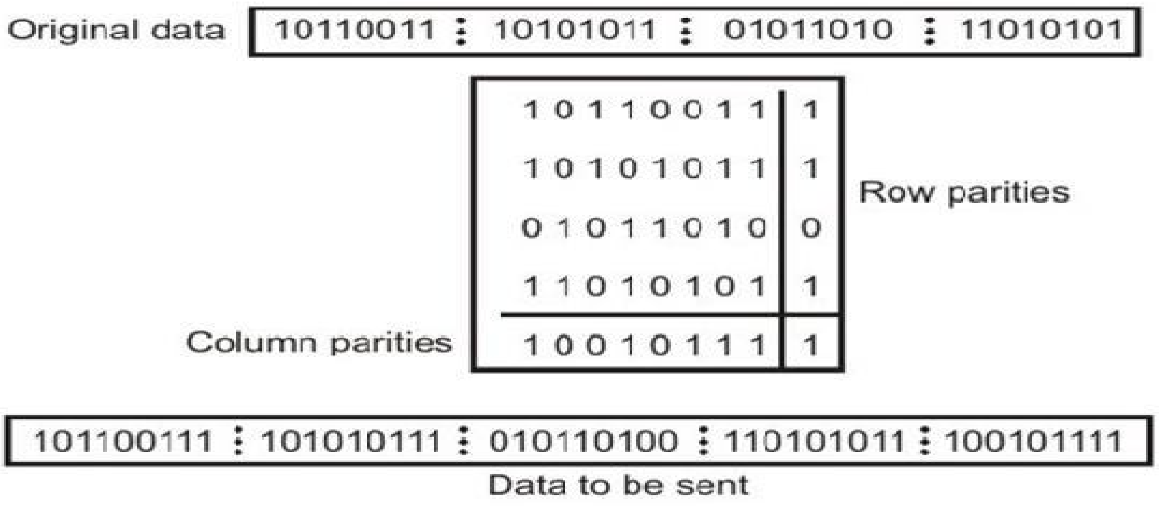


Figure 3.2.4 Two-dimension Parity Checking

Performance:

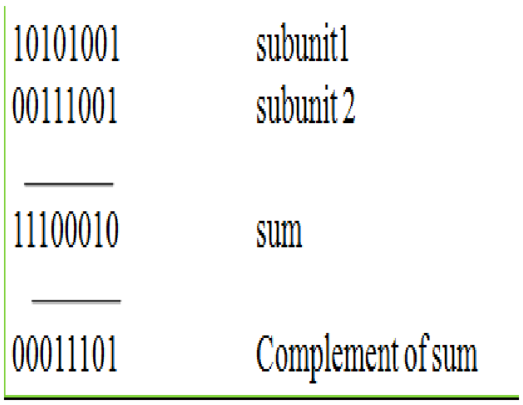
- If two bits in one data unit are damaged and two bits in exactly same position in another data unit are also damaged, The 2-D Parity check **checker will not detect an error**.
- For example, if two data units: **11001100 and 10101100.**
- If first and second from last bits in each of them is changed, making the data units as **01001110 and 00101110**, the error cannot be detected by 2-D Parity check.

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.**
- The sum is complemented to get the checksum. The **checksum** segment is sent **along with the data segments**

Example 1:

Sender **Reciever:**



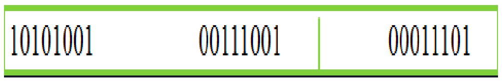
```
10101001        subunit1
00111001        subunit 2
________
11100010        sum
________
00011101        Complement of sum
```

```
10101001        subunit1
00111001        subunit2
00011101        Checksum

11111111        sum

00000000        complement
```

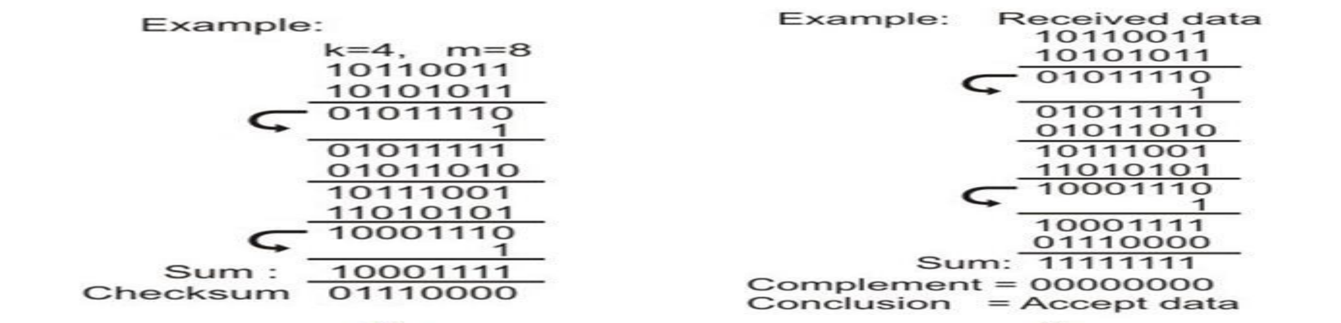
Conclusion = Accept data.



10101001	00111001	00011101

Data checksum

Example 2: K= 10110011, 10101011, 01011111, 11010101



```
Example:
          k=4,   m=8
          10110011
          10101011
          --------
          01011110
                 1
          --------
          01011111
          01011010
          --------
          10111001
          11010101
          --------
          10001110
                 1
          --------
Sum :     10001111
Checksum  01110000
            (a)
```

```
Example:     Received data
             10110011
             10101011
             --------
             01011110
                    1
             --------
             01011111
             01011010
             --------
             10111001
             11010101
             --------
             10001110
                    1
             --------
             10001111
             01110000
             --------
Sum:         11111111
Complement = 00000000
Conclusion = Accept data
               (b)
```

Figure 3.2.5 (a) Sender's end for the calculation of the checksum, (b) Receiving end for checking the checksum

<u>**CYCLIC REDUNDANCY CHECK**</u>

- One of the most powerful and commonly used error detecting codes.

Basic approach:

- Given a m-bit block of bit sequence, the sender generates an n-bit sequence known as **frame sequence check(FCS),** so that the resulting frame, consisting of m+n bits exactly divisible by **same predetermined number.**

- The receiver divides the incoming frame by that number and, if there is **no reminder, assumes there was no error.**

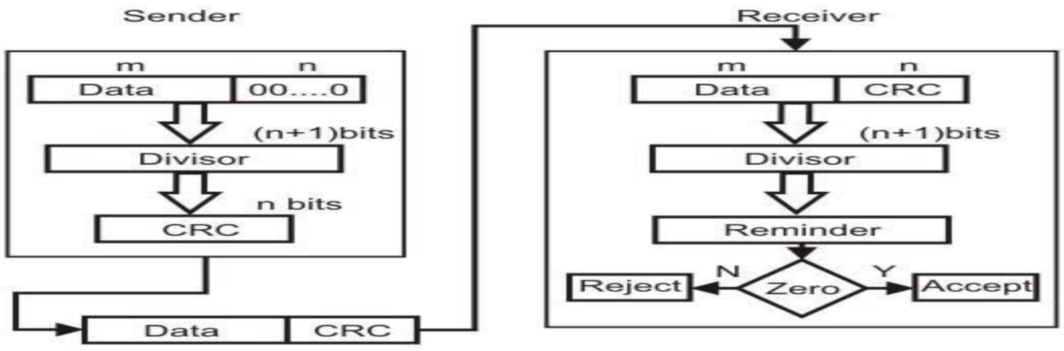


Fig. 3.2.7 by dividing a sample 4- bit number by the coefficient of the generator polynomial x^3+x+1, which is 1011, using the modulo-2 arithmetic.
Modulo-2 arithmetic is a binary addition process without any carry over, which is just the Exclusive-OR operation.
Consider the case where k=1101. Hence 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 oferrors.

Sender: Sender transmit the data along with remainder (**CRC**)

Data: 1101

Divisor: 1011

```
      ______________
     | 1111  Quotient
1011 |                      K
       1101000  <--------
       -------
       1011

        1100
       ------
        1011

         1110
        ------
         1011
```

1010 **Data to be sent: 1 1 0 1 | 0 0 1**

1011 **Data CRC**

001 Reminder (r)

Receiver:

1111 Quotient **Data: 1101001**

1011 | 1101001 **Divisor: 1011**

1011

1100

1011

1110

1011

1011

1011

000 Reminder

Note: Remainder is zero, no error. Receiver can accept the data.

Performance of CRC

- CRC can detect all single-bit errors.
- CRC can detect all double-bit errors(three1's)
- CRC can detect any odd number of errors of less than the degree of the polynomial.
- CRC detects most of the larger burst errors with a high probability.

3 Data Link Layer Design Issues:

- Services Provided to the Network Layer
- Framing
- Error Control
- Flow Control

3.1 FRAMING:

- Data transmission in the physical layer means moving bits in the form of a signal from the source to the destination.
- The physical layer provides bit synchronization to ensure that the sender and receiver use the same bit durations and timing.
- The data link layer, on the other hand, needs to pack bits into frames, so that each frame is distinguishable from another. Our postal system practices a type of framing.
- The simple act of inserting a letter into an envelope separates one piece of information from another; the envelope serves as the delimiter. In addition, each envelope defines the sender and receiver addresses since the postal system is a many-to-many carrier facility.
- Framing in the data link layer separates a message from one source to a destination, or from other messages to other destinations, by adding a sender address and a destination address. The destination address defines where the packet is to go; the sender address helps the recipient acknowledge the receipt.

3.1.1 Fixed-Size Framing: Frames can be of fixed or variable size. In fixed-size framing, there is no need for defining the boundaries of the frames; the size itself can be used as a delimiter. An example of this type of framing is the ATM wide-area network, which uses frames of fixed size called cells.

3.1.2 Variable-Size Framing : In variable-size framing, we need a way to define the end of the frame and the beginning of the next. Historically, two approaches were used for this purpose: a character-oriented approach and a bit-oriented approach.

3.1.2.1 *Character-Oriented Protocols*

- In a character-oriented protocol, data to be carried are 8-bit characters from a coding system such as The header, which normally carries the source and destination addresses and other control information, and the trailer, which carries error detection or error correction redundant bits, are also multiples of 8 bits.

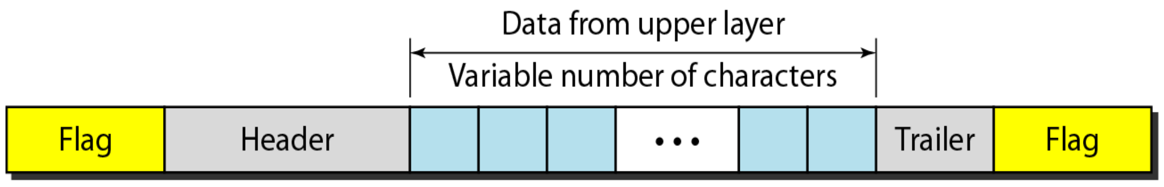


Figure: *A frame in a character-oriented protocol*

- To separate one frame from the next, an 8-bit (1-byte) flag is added at the beginning and the end of a frame. The flag, composed of protocol-dependent special characters, signals the start or end of a frame. Below figure shows the format of a frame in a character-oriented protocol.
- Character-oriented framing was popular when only text was exchanged by the data link layers. The flag could be selected to be any character not used for text communication. Now, however, we send other types of information such as graphs, audio, and video. Any pattern used for the flag could also be part of the information. If this happens, the receiver, when it encounters this pattern in the middle of the data, thinks it has reached the end of the frame. To fix this problem, a byte-stuffing strategy was added to character-oriented framing. In byte stuffing (or character stuffing), a special byte is added to the data section of the frame when there is a character with the same pattern as the flag. The data section is stuffed with an extra byte. This byte is usually called the escape character (ESC), which has a predefined bit pattern. Whenever the receiver encounters the ESC character, it removes it from the data section and treats the next character as data, not a delimiting flag.

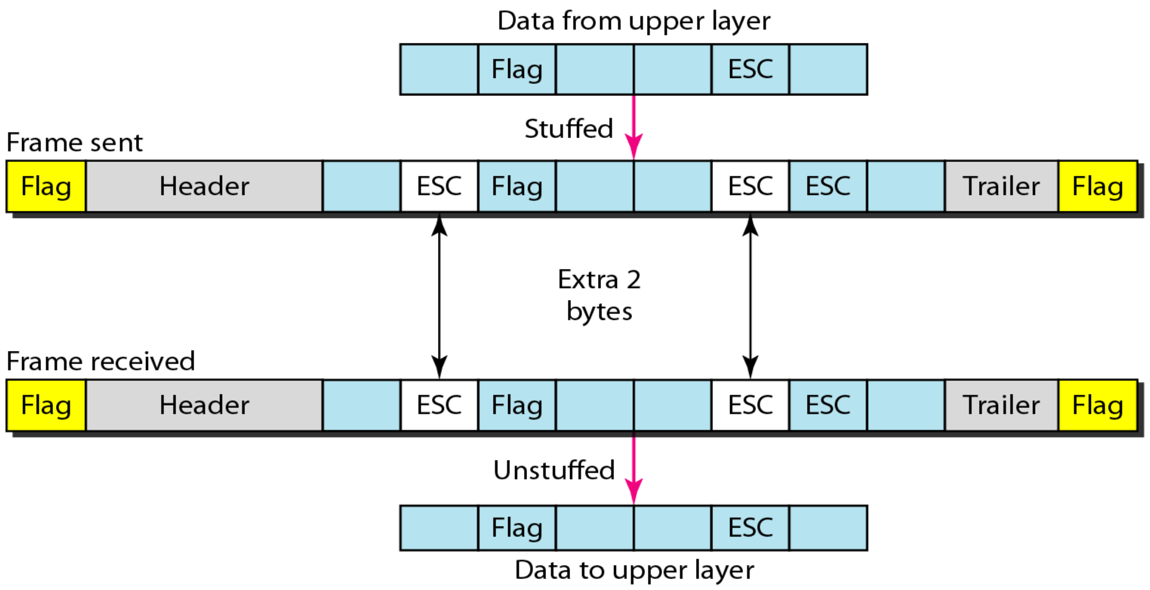


Figure : *Byte stuffing and unstuffing*

- Byte stuffing by the escape character allows the presence of the flag in the data section of the frame, but it creates another problem. What happens if the text contains one or more escape characters followed by a flag? The receiver removes the escape character, but keeps the flag, which is incorrectly interpreted as the end of the frame. To solve this problem, the escape characters that are part of the text must also be marked by another escape character. In other words, if the escape character is part of the text, an extra one is added to show that the second one is part of the text. Below figure shows the situation.

3.1.2.2Bit-Oriented Protocols

- In a bit-oriented protocol, the data section of a frame is a sequence of bits to be interpreted by the upper layer as text, graphic, audio, video, and so on. However, in addition to headers (and possible trailers), we still need a delimiter to separate one frame from the other. Most protocols use a special 8-bit pattern flag 01111110 as the delimiter to define the beginning and the end of the frame, as shown in figure.

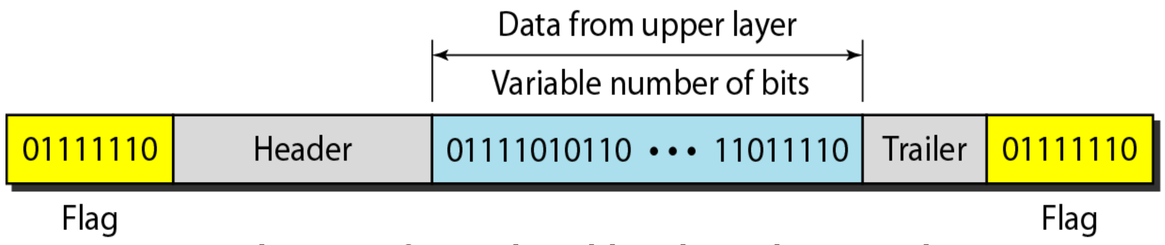


Figure: *A frame in a bit-oriented protocol*

- This flag can create the same type of problem we saw in the byte-oriented protocols.
- That is, if the flag pattern appears in the data, we need to somehow inform the receiver that this is not the end of the frame. We do this by stuffing 1 single bit (instead of 1 byte) to prevent the pattern from looking like a flag. The strategy is called bit stuffing. In bit stuffing, if a 0 and five consecutive 1 bits are encountered, an extra 0 is added.
- This extra stuffed bit is eventually removed from the data by the receiver. Note that the extra bit is added after one 0 followed by five 1s regardless of the value of the next bit. This guarantees that the flag field sequence does not inadvertently appear in the frame.
- Below figure shows bit stuffing at the sender and bit removal at the receiver. Note that even if we have a 0 after five 1s, we still stuff a 0. The 0 will be removed by the receiver.

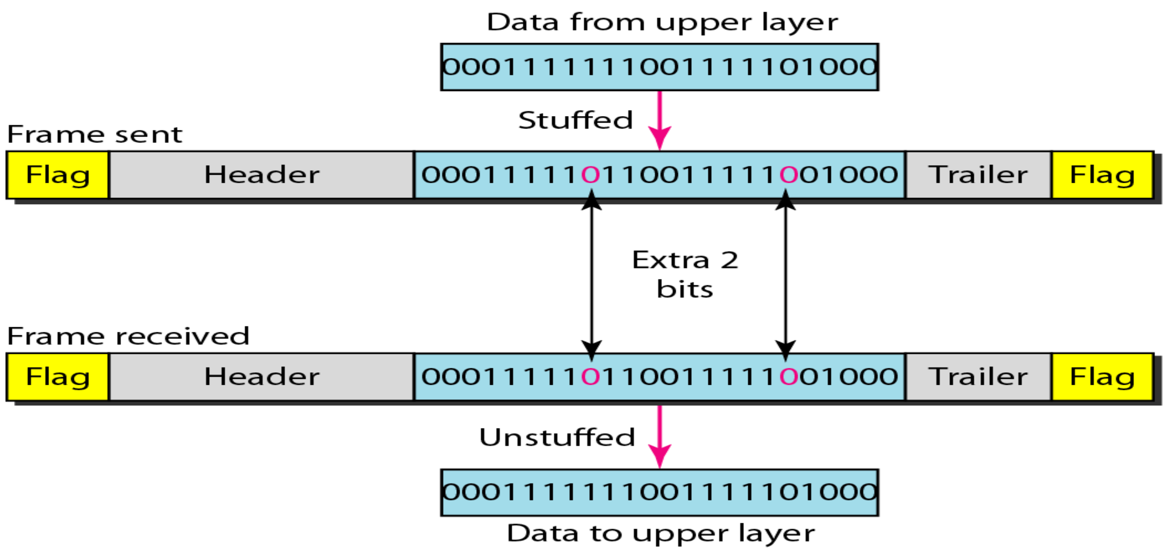


Figure:*Bit stuffing and unstuffing*

- This means that if the flaglike pattern 01111110 appears in the data, it will change to 011111010 (stuffed) and is not mistaken as a flag by the receiver. The real flag 01111110 is not stuffed by the sender and is recognized by the receiver.

3.2FLOW AND ERROR CONTROL

Data communication requires at least two devices working together, one to send and the other to receive. Even such a basic arrangement requires a great deal of coordination for an intelligible exchange to occur. The most important responsibilities of the data link layer are flow control and error control. Collectively, these functions are known as data link control.

3.2.1Flow Control

- Flow control coordinates the amount of data that can be sent before receiving an acknowledgment and is one of the most important duties of the data link layer. In most protocols,
- Flow control is a set of procedures that tells the sender how much data it can transmit before it must wait for an acknowledgment from the receiver. The flow of data must not be allowed to overwhelm the receiver.
- Any receiving device has a limited speed at which it can process incoming data and a limited amount of memory in which to store incoming data. The receiving device must be able to inform the sending device before those limits are reached and to request that the transmitting device send fewer frames or stop temporarily.
- Incoming data must be checked and processed before they can be used. The rate of such processing is often slower than the rate of transmission.
- For this reason, each receiving device has a block of memory, called a *buffer,* reserved for storing incoming data until they are processed.

- If the buffer begins to fill up, the receiver must be able to tell the sender to halt transmission until it is once again able to receive.

3.2.2 Error Control

- Error control is both error detection and error correction. It allows the receiver to inform the sender of any frames lost or damaged in transmission and coordinates the retransmission of those frames by the sender.
- In the data link layer, the term *error control* refers primarily to methods of error detection and retransmission.
- Error control in the data link layer is often implemented simply: Any time an error is detected in an exchange, specified frames are retransmitted. This process is called automatic repeat request (ARQ).

3.3 *Error Detection and Correction*

3.3.1 Introduction:

3.3.1.1Types 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.

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.

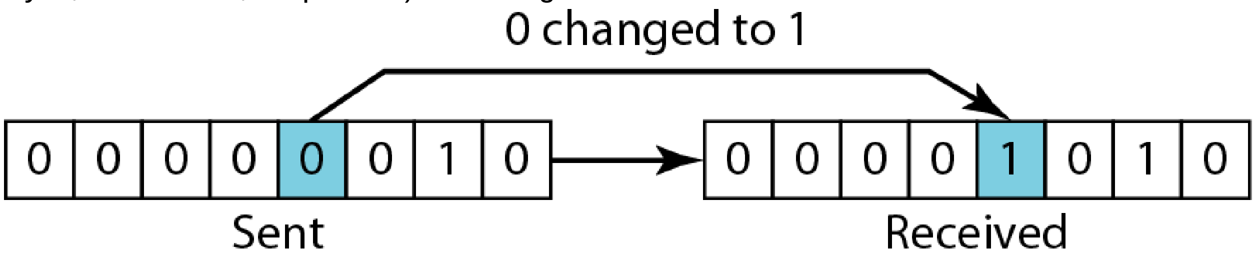


Figure :*Single-bit error*

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 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.

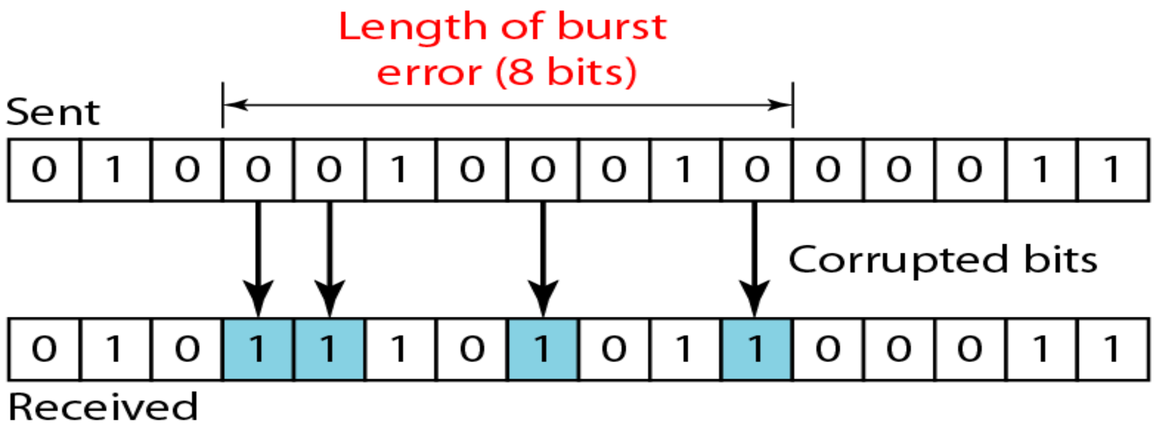


Figure:Burst error of length 8

3.3.1.2Redundancy

- 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.

3.3.1.3 Coding

- 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 shows the general idea of coding.

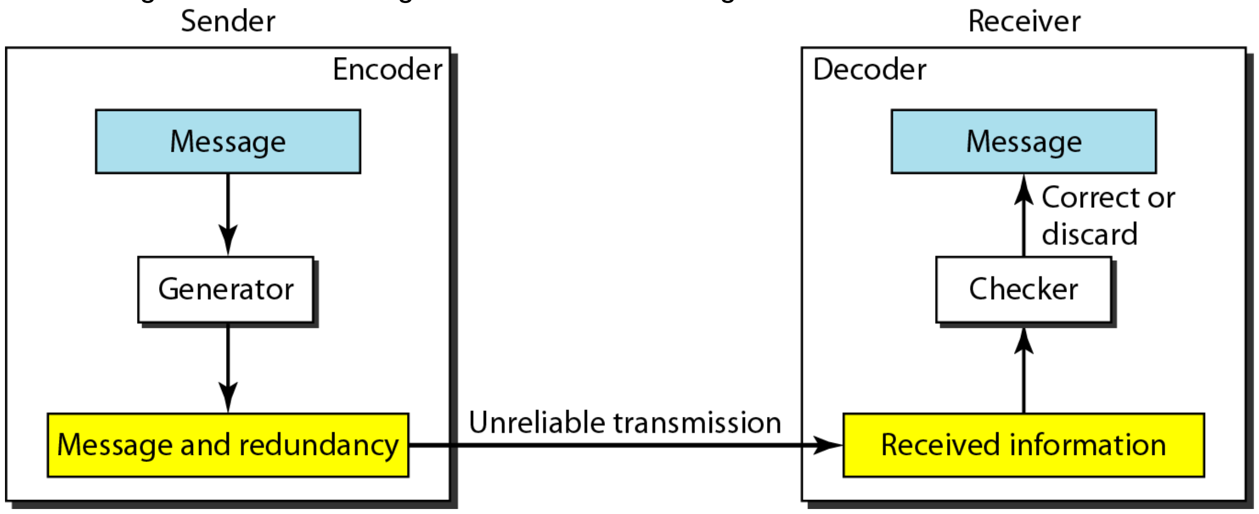


Figure :*The structure of encoder and decoder*

Coding schemes are classified into two broad categories: block coding and convolution coding.

3.3.2 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 2^n - 2^k codewords that are not used. We call these codewords invalid or illegal. Figure shows the situation.

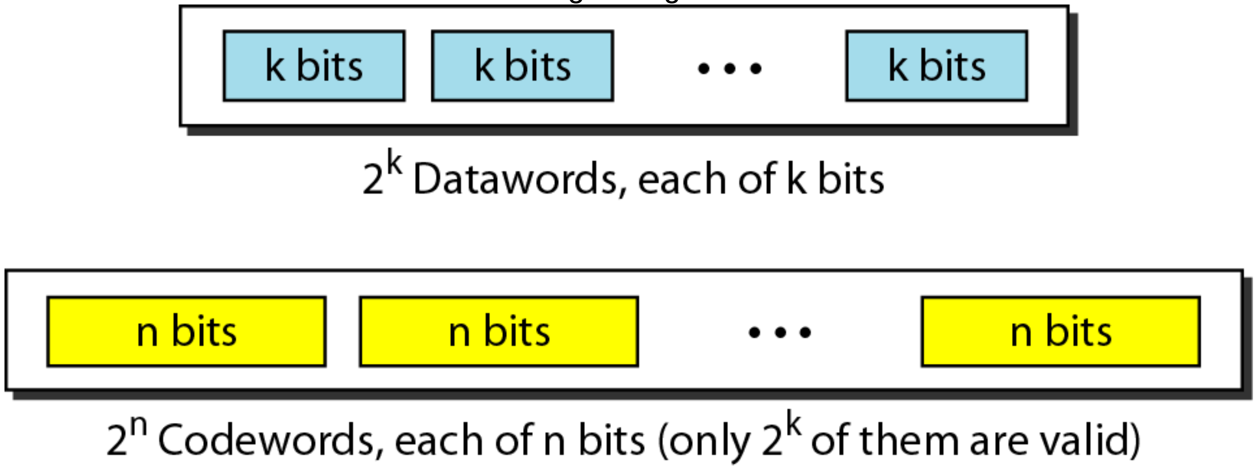

k bits
k bits
• • •
k bits
2^k Datawords, each of k bits
n bits
n bits
• • •
n bits
2^n Codewords, each of n bits (only 2^k of them are valid)

Figure: Datawords and Codewords in Block Coding

3.3.2.1 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.

i. The receiver has (or can find) a list of valid codewords.

ii. The original codeword has changed to an invalid one.

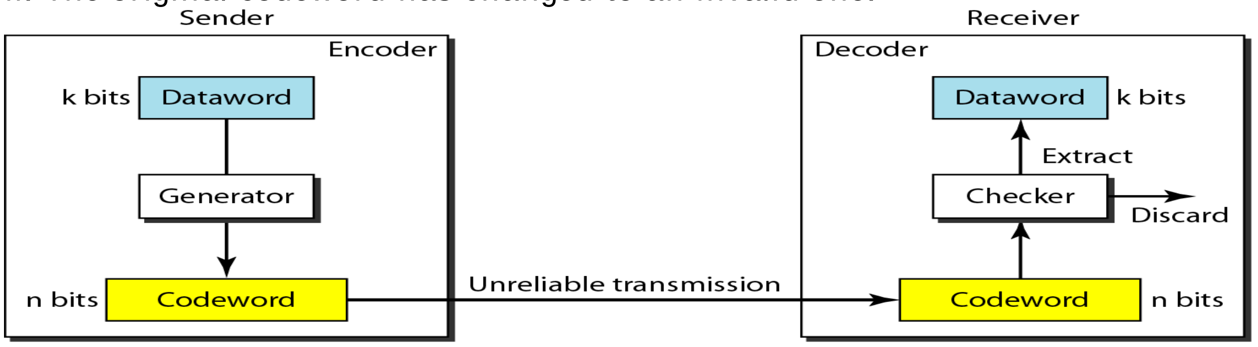


Figure :*Process of error detection in block coding*

- The sender creates codewords out of datawords by using a generator that applies the rules and procedures of encoding .Each codeword sent to the receiver may change during transmission.
- If the received codeword is the same as one of the valid codewords, the word is accepted; the corresponding dataword 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.

3.3.2.2 Error Correction

- 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. Figure shows the role of block coding in error correction.

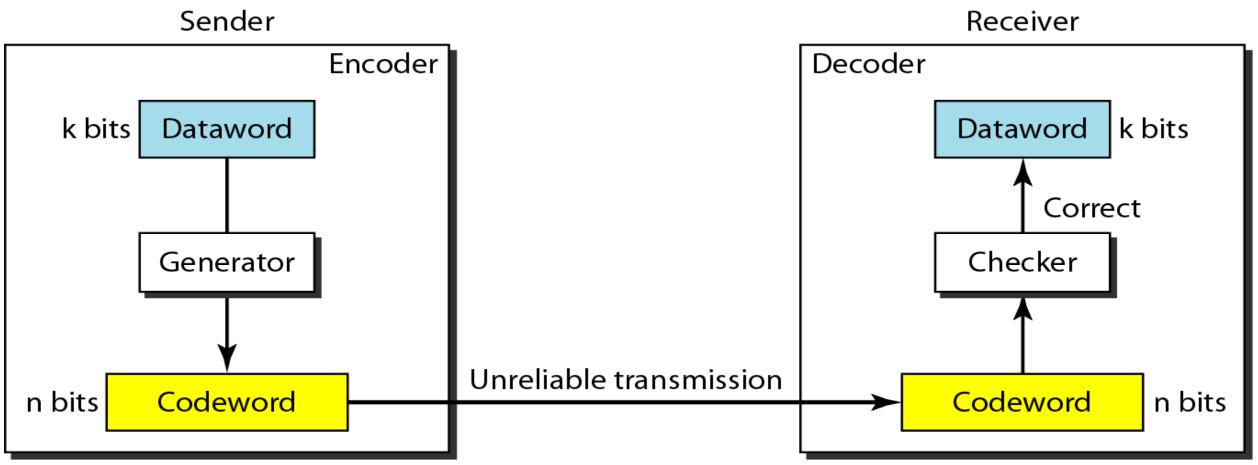


Figure :*Structure of encoder and decoder in error correction*

3.3.3 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.
- The **minimum Hamming distance** is the smallest Hamming distance between all possible pairs in a set of words

3.4 LINEAR BLOCK CODES

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

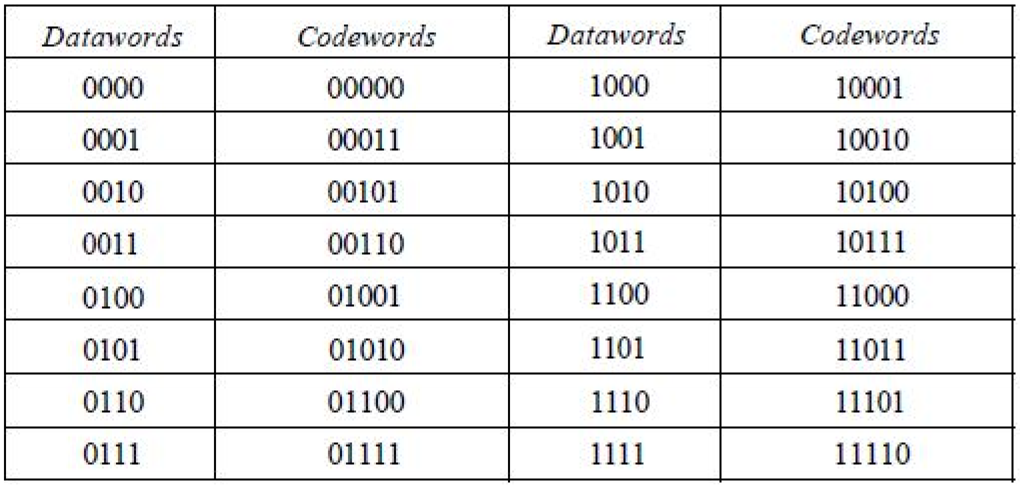
3.4.1 Minimum Distance for Linear Block Codes

It is simple to find the minimum Hamming distance for a linear block code. The minimum
Hamming distance is the number of Is in the nonzero valid codeword with the smallest number of 1s.

3.4.2 Simple Parity-Check Code

- The most familiar error-detecting code is the simple parity-check code. In this code, a *k-bit* dataword is changed to an n-bit codeword where *n* = *k* + 1.
- The extra bit, called the parity bit, is selected 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.
- Our first code (Table) is a parity-check code with *k* = 2 and *n* =3. The code in table is also a parity-check code with *k* =4 and *n* =5.

Simple parity-check code C(5, 4)



Datawords	*Codewords*	*Datawords*	*Codewords*
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11011
0110	01100	1110	11101
0111	01111	1111	11110

- Figure shows a possible structure of an encoder (at the sender) and a decoder (at the receiver).

Encoder and decoderfor simple parity-check code

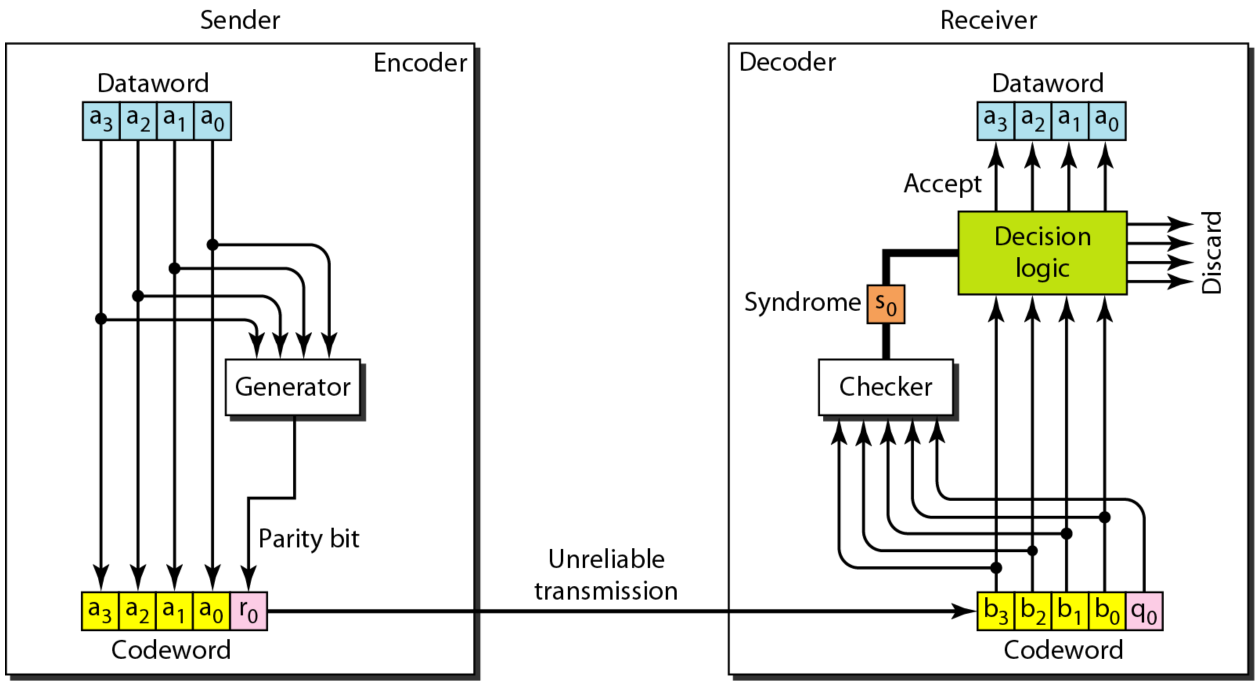


Figure :*Encoder and decoder for simple parity-check code*

- The encoder uses a generator that takes a copy of a 4-bit dataword *(a_0, a_1,a_2,*and *a_3)* and generates a parity bit *r_0* .The dataword bits and the parity bit create the 5-bit codeword. The parity bit that is added makes the number of 1s in the codeword even.
- This is normally done by adding the 4 bits of the dataword (modulo-2); the result is the parity bit. In other words,

$$r_0=a_3+a_2+a_1+a_0 \text{ (modulo-2)}$$

- If the number of 1s is even, the result is 0; if the number of 1s is odd, the result is 1. In both cases, the total number of 1s in the codeword is even. The sender sends the codeword which may be corrupted during transmission. The receiver receives a 5-bit word. The checker at the receiver does the same thing as the generator in the sender with one exception: The addition is done over all 5 bits. The result, which is called the syndrome, is just 1 bit. The syndrome is 0 when the number of 1s in the received codeword is even; otherwise, it is 1.

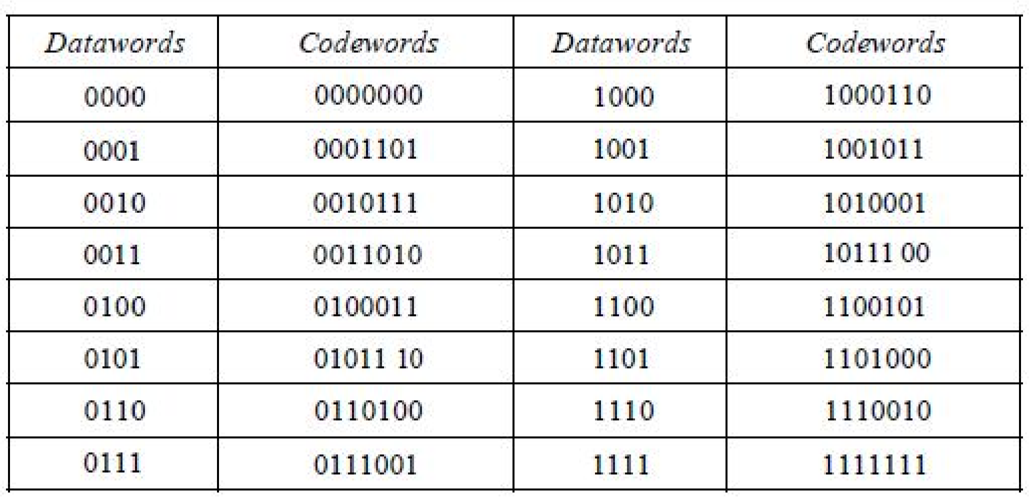
$$S_0=b_3+b_2+b_1+b_0+q_0 \text{ (modulo-2)}$$

- The syndrome is passed to the decision logic analyzer. If the syndrome is 0, there is no error in the received codeword; the data portion of the received codeword is accepted as the dataword; if the syndrome is 1, the data portion of the received codeword is discarded. The dataword is not created.

3.4.3 Hamming Codes

- Now let us discuss a category of error-correcting codes called Hamming codes. These codes were originally designed with $d_{min} = 3$, 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 nand *k* are then calculated from m as $n = 2^m - 1$ 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 $d_{min} = 3$. Below table shows the datawords and codewords for this code.



Datawords	*Codewords*	*Datawords*	*Codewords*
0000	0000000	1000	1000110
0001	0001101	1001	1001011
0010	0010111	1010	1010001
0011	0011010	1011	10111 00
0100	0100011	1100	1100101
0101	01011 10	1101	1101000
0110	0110100	1110	1110010
0111	0111001	1111	1111111

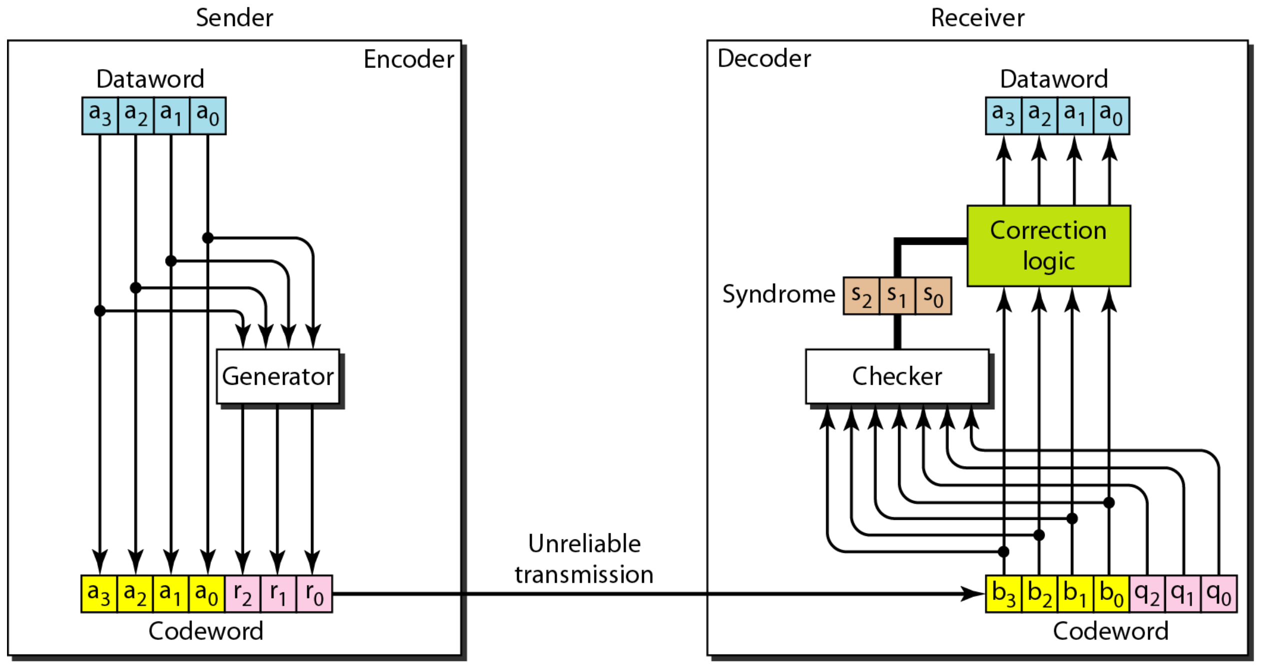


Figure: The structure of Encoder and Decoder for a Hamming code

- A copy of a 4-bit dataword is fed into the generator that creates three parity checks r_0, r_1 and r_2 as shown below:

r0=a2+a1+a0 modul02
r1=a3+a2+a1 modul02
r2=a1+a0+a3 modul02

- The checker in the decoder creates a 3-bit syndrome ($s_2s_1s_0$) in which each bit is the parity check for 4 out of the 7 bits in the received codeword:

S0=b2+b1+b0+q0 modulo2
S1=b3+b2+b1+q1 modulo2
S2=b1+b0+b3+q2 modulo2

Syndrome	000	001	010	011	100	101	110	111
Error	None	q_0	q_1	b_2	q_2	b_0	b_3	b_1

Table :*Logical decision made by the correction logic analyzer*

3.4.4Performance

- 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, as shown in figure.
- The key is to split a burst error between several codewords, 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* codewords out of our frame. Then, instead of sending one codeword at a

time, we arrange the codewords in a table and send the bits in the table a column at a time.

- In figure, the bits are sent column by column (from the left). In each column, the bits are sent from the bottom to the top. In this way, a frame is made out of the four codewords and sent to the receiver. Figure shows that when a burst error of size 4 corrupts the frame, only 1 bit from each codeword is corrupted. The corrupted bit in each codeword can then easily be corrected at the receiver.

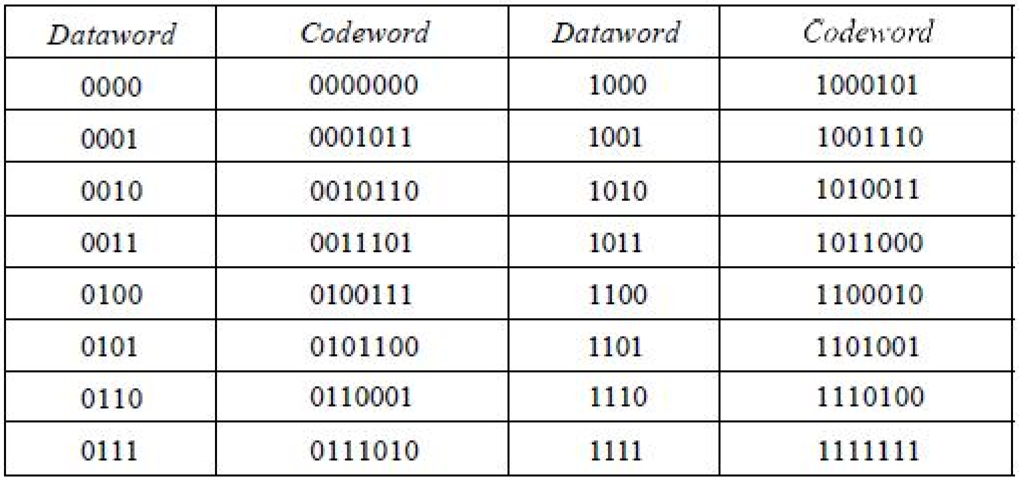
3.5CYCLIC CODES

- Cyclic codes are special linear block codes with one extra property. In a cyclic code, if a codeword is cyclically shifted (rotated), the result is another codeword.
- For example, if 1011000 is a codeword and we cyclically left-shift, then 0110001 is also a codeword.
- In this case, if we call the bits in the first word a_0 to a_6 ,and the bits in the second word b_0 to b_6, we can shift the bits by using the following:
- In the rightmost equation, the last bit of the first word is wrapped around and becomes the first bit of the second word.

3.5.1Cyclic Redundancy Check

- We can create cyclic codes to correct errors. However, the theoretical background required is beyond the scope of this book. In this section, we simply discuss a category of cyclic codes called the cyclic redundancy check (CRC) that is used in networks such as LANs and WANs.
- Table shows an example of a CRC code. We can see both the linear and cyclic properties of this code.

A CRC code with C(7, 4)



Dataword	*Codeword*	*Dataword*	*Codeword*
0000	0000000	1000	1000101
0001	0001011	1001	1001110
0010	0010110	1010	1010011
0011	0011101	1011	1011000
0100	0100111	1100	1100010
0101	0101100	1101	1101001
0110	0110001	1110	1110100
0111	0111010	1111	1111111

- In the encoder, the dataword has *k* bits (4 here); the codeword has *n* bits (7 here). The size of the dataword is augmented by adding *n* - *k* (3 here) Os to the right-hand side of the word. The n-bit result is fed into the generator.

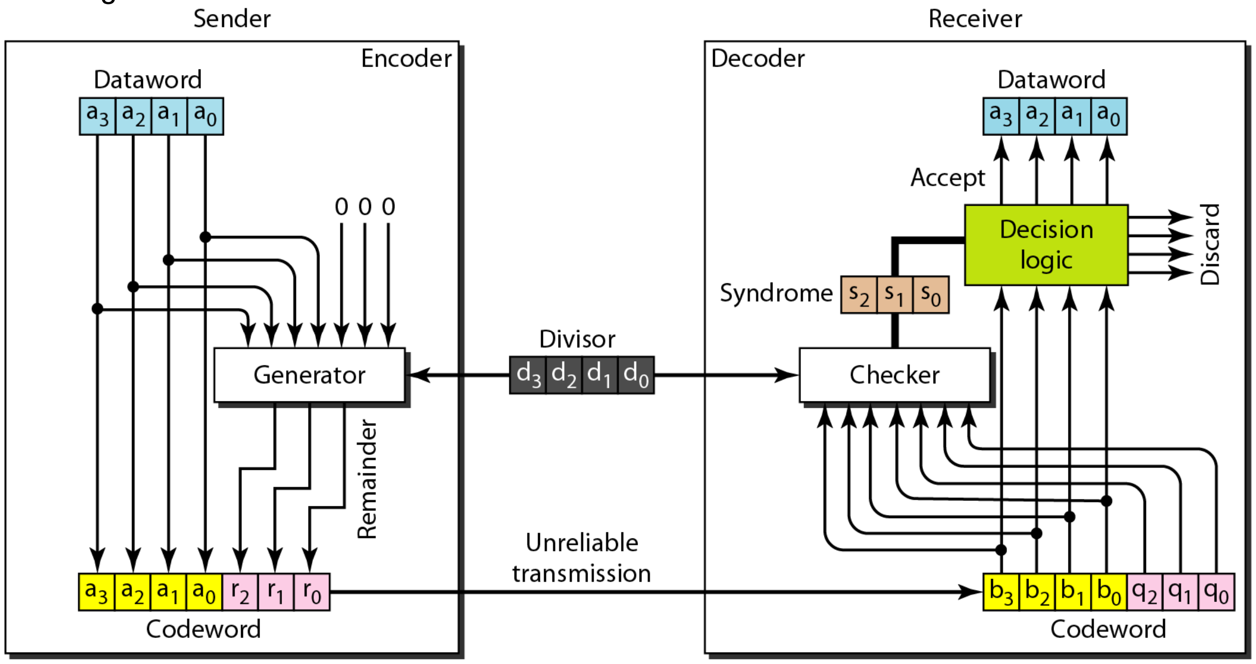


Figure:*CRC encoder and decoder*

- The generator uses a divisor of size *n* - *k* + 1(4 here), predefined and agreed upon. The generator divides the augmented dataword by the divisor (modulo-2 division). The quotient of the division is discarded; the remainder *($r_2r_1r_0$)* is appended to the dataword to create the codeword.
- The decoder receives the possibly corrupted codeword. A copy of all *n* bits is fed to the checker which is a replica of the generator. The remainder produced by the checker is a syndrome of *n* - *k* (3 here) bits, which is fed to the decision logic analyzer. The analyzer has a simple function. If the syndrome bits are all as, the 4 leftmost bits of the codeword are accepted as the dataword (interpreted as no error); otherwise, the 4 bits are discarded (error).

Encoder

- Let us take a closer look at the encoder. The encoder takes the dataword and augments it with *n* - *k* number of 0s. It then divides the augmented dataword by the divisor, as shown in figure.

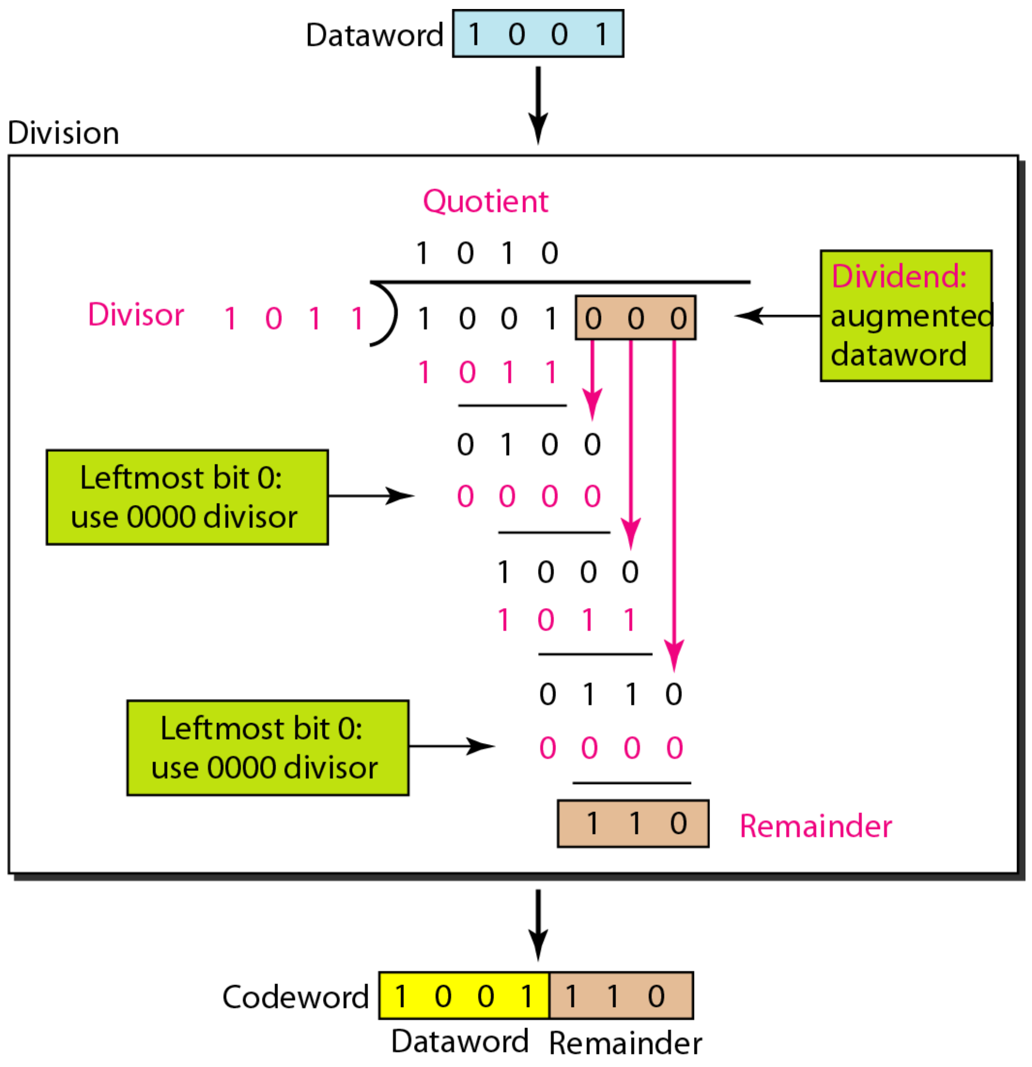


Figure: *Division in CRC encoder*

- The process of modulo-2 binary division is the same as the familiar division process
- As in decimal division, the process is done step by step. In each step, a copy of the divisor is XORed with the 4 bits of the dividend. The result of the XOR operation (remainder) is 3 bits (in this case), which is used for the next step after 1 extra bit is pulled down to make it 4 bits long.
- There is one important point we need to remember in this type of division. If the leftmost bit of the dividend (or the part used in each step) is 0, the step cannot use the regular divisor; we need to use an all-0s divisor. When there are no bits left to pull down, we have a result. The 3-bit remainder forms the check bits *(r2' rl'* and *ro).* They are appended to the dataword to create the codeword.

Decoder

- The codeword can change during transmission. The decoder does the same division process as the encoder. The remainder of the division is the syndrome. If the syndrome is all 0s, there is no error; the dataword is separated from the received codeword and accepted. Otherwise, everything is discarded. Figure shows two cases: The lefthand figure shows the value of syndrome when no error has occurred; the syndrome is 000. The right-hand part of the figure shows the case in which there is one single error.
  The syndrome is not all 0s (it is 011).

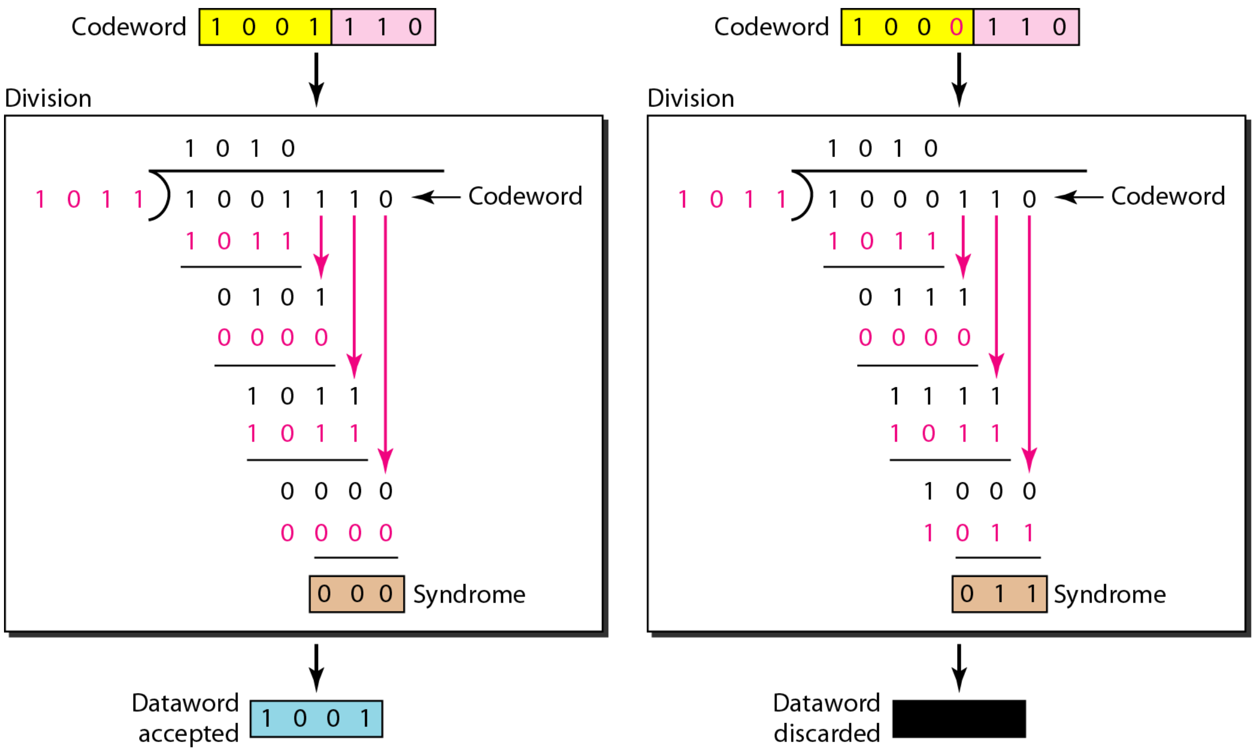
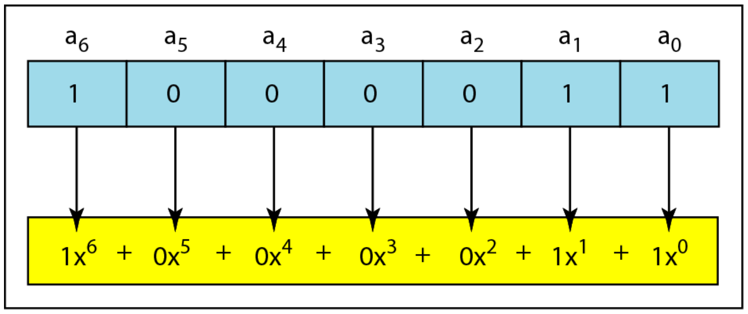


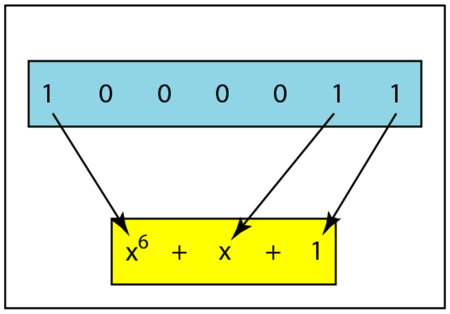
Figure: *Division in the CRC decoder for two cases*

3.5.2 Polynomials

- A better way to understand cyclic codes and how they can be analyzed is to represent them as polynomials.
- A pattern of 0s and 1s can be represented as a **polynomial** with coefficients of 0 and 1. The power of each term shows the position of the bit; the coefficient shows the value of the bit. Figure(a)a shows a binary pattern and its polynomial representation.
- In figure(b) can be shortened by removing all terms with zero coefficients and replacing x1 by *x* and *x0* by 1.


a_6 a_5 a_4 a_3 a_2 a_1 a_0
1 0 0 0 0 1 1
$1x^6 + 0x^5 + 0x^4 + 0x^3 + 0x^2 + 1x^1 + 1x^0$

a. Binary pattern and polynomial



b. Short form

Figure :*A polynomial to represent a binary word*

Degree of a Polynomial

- The degree of a polynomial is the highest power in the polynomial. For example, the degree of the polynomial $x^6 + x + 1$ is 6. Note that the degree of a polynomial is 1 less that the number of bits in the pattern. The bit pattern in this case has 7 bits.

Adding and Subtracting Polynomials

- Adding and subtracting polynomials in mathematics are done by adding or subtracting the coefficients of terms with the same power. In our case, the coefficients are only 0 and 1, and adding is in modulo-2. This has two consequences. First, addition and subtraction are the same.
- Second, adding or subtracting is done by combining terms and deleting pairs of identical terms. For example, adding $x^5 + x^4 + x^2$ and $x^6 + x^4 + x^2$ gives just $x^6 + x^5$. The terms x^4 and x^2 are deleted. However, note that if we add, for example, three polynomials and we get x^2 three times, we delete a pair of them and keep the third.

Cyclic Code Encoder Using Polynomials

- The dataword 1001 is represented as $x^3 + 1$. The divisor 1011 is represented as $x^3 + x + 1$. To find the augmented dataword, we have left-shifted the dataword 3 bits (multiplying by x^3) The result is $x^6 + x^3$. Division is straightforward.
- We divide the first term of the dividend, x^6, by the first term of the divisor, x^3. The first term of the quotient is then x^6/x^3, or x^3. Then we multiply $x3$ by the divisor and subtract, the result from the dividend. The result is x^4, with a degree greater than the divisor's degree; we continue to divide until the degree of the remainder is less than the degree of the divisor.

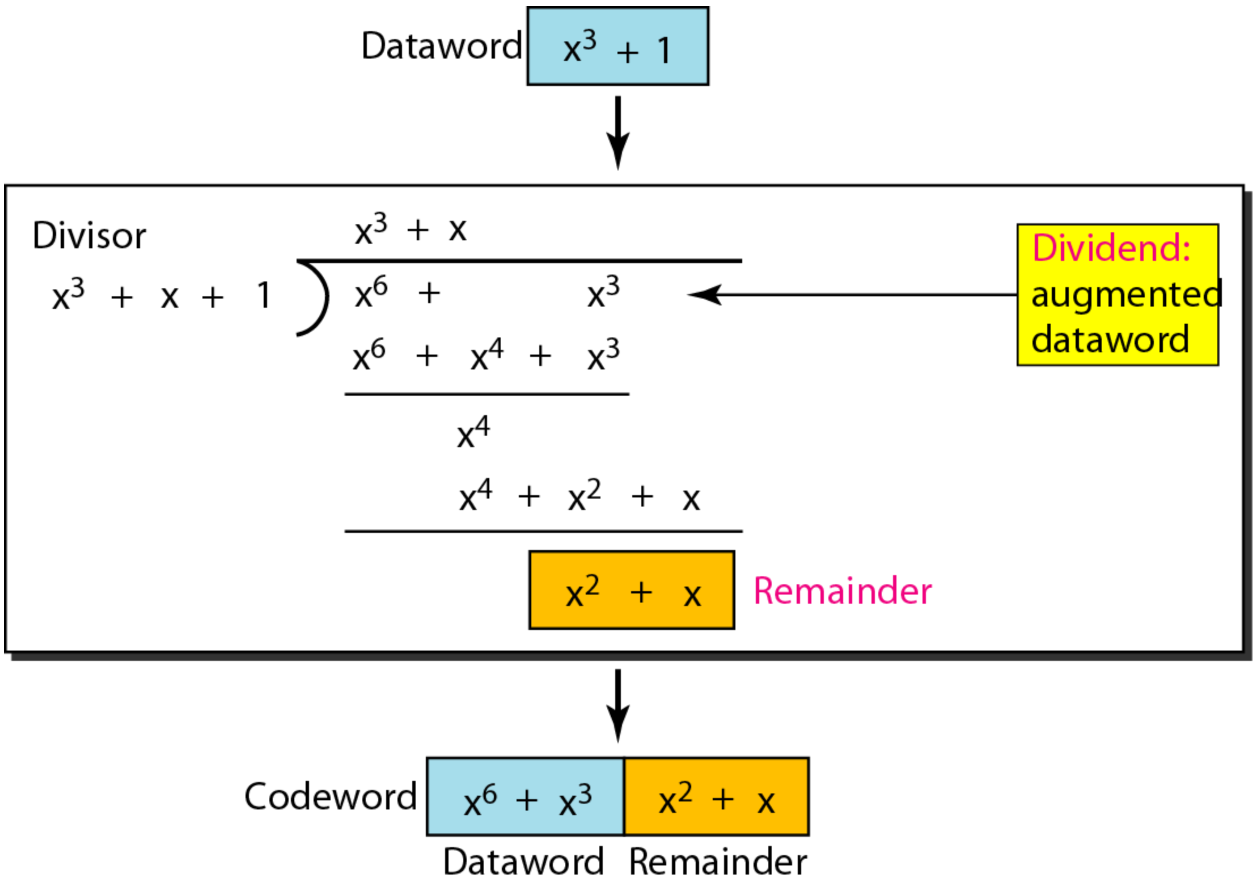


Figure: CRC division using polynomials

Cyclic Code Analysis

- We can analyze a cyclic code to find its capabilities by using polynomials. We define the following, *wheref(x)* is a polynomial with binary coefficients.
  Dataword: *d(x)* Syndrome: *s(x)* Codeword: *c(x)* Error: *e(x)* Generator: *g(x)*
- If *s(x)* is not zero, then one or more bits is corrupted. However, if *s(x)* is zero, either no bit is corrupted or the decoder failed to detect any errors.
- In our analysis we want to find the criteria that must be imposed on the generator, *g(x)* to detect the type of error we especially want to be detected. Let us first find the relationship among the sent codeword, error, received codeword, and the generator. We can say

$$\text{Received codeword} = c(x) + e(x)$$

- In other words, the received codeword is the sum of the sent codeword and the error. The receiver divides the received codeword by *g(x)* to get the syndrome. We can write this as

$$\frac{\text{Received codeword}}{g(x)} = \frac{c(x)}{g(x)} + \frac{e(x)}{g(x)}$$

- The first term at the right-hand side of the equality does not have a remainder (according to the definition of codeword). So the syndrome is actually the remainder of the second term on the right-hand side. If this term does not have a remainder (syndrome =0), either *e(x)* is 0 or *e(x)* is divisible by *g(x).*

3.6 CHECKSUM

Like linear and cyclic codes, the checksum is based on the concept of redundancy. Several protocols still use the checksum for error detection as we will see in future chapters, although the tendency is to replace it with a CRC. This means that the CRC is also used in layers other than the data link layer.

Idea

The concept of the checksum is not difficult. Let us illustrate it with a few examples.

Case1:

Suppose our data is a list of five 4-bit numbers that we want to send to a destination. In addition to sending these numbers, we send the sum of the numbers. For example, if the set of numbers is (7, 11, 12, 0, 6), we send (7, 11, 12, 0, 6, 36), where 36 is the sum of the original numbers. The receiver adds the five numbers and compares the result with the sum. If the two are the same, the receiver assumes no error, accepts the five numbers, and discards the sum. Otherwise, there is an error somewhere and the data are not accepted.

Case2:

We can make the job of the receiver easier if we send the negative (complement) of the sum, called the checksum. In this case, we send (7, 11, 12, 0, 6, −36). The receiver can add all the numbers received (including the checksum). If the result is 0, it assumes no error; otherwise, there is an error.

One's Complement

- All of our data can be written as a 4-bit word (they are less than 15) except for the checksum. One solution is to use one's complement arithmetic. In this arithmetic, we can represent unsigned numbers between 0 and 2^n - 1 using only *n* bits.
- If the number has more than *n* bits, the extra leftmost bits need to be added to the *n* rightmost bits (wrapping). In one's complement arithmetic, a negative number can be represented by inverting all bits (changing a 0 to a 1 and a 1 to a 0). This is the same as subtracting the number from 2^n - 1.
- **Example:** How can we represent the number 21 in one's complement arithmetic using only four bits?

The number 21 in binary is 10101 (it needs five bits). We can wrap the leftmost bit and add it to
the four rightmost bits. We have (0101 + 1) = 0110 or 6.

- How can we represent the number -6 in one's complement arithmetic using only four bits?

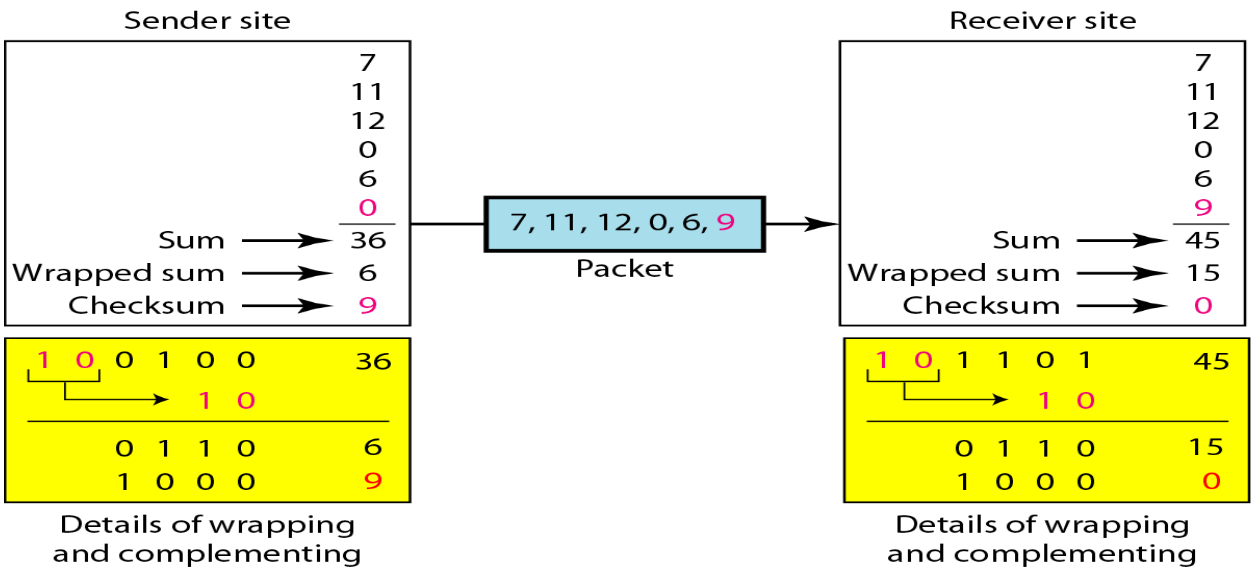
Solution

In one's complement arithmetic, the negative or complement of a number is found by inverting all bits. Positive 6 is 0110; negative 6 is 100I. If we consider only unsigned numbers, this is 9. In other words, the complement of 6 is 9. Another way to find the complement of a number in one's complement arithmetic is to subtract the number from 2^n - 1 (16 - 1 in this case).

Example:

Figure shows the process at the sender and at the receiver. The sender initializes the checksum to 0 and adds all data items and the checksum (the checksum is considered as one data item and is shown in color). The result is 36. However, 36 cannot be expressed in 4 bits. The extra two bits are wrapped and added with the sum to create the wrapped sum value 6. In the figure, we have shown the details in binary. The sum is then complemented, resulting in the checksum value 9 (15 − 6 = 9). The sender now sends six data items to the receiver including the checksum 9.

The receiver follows the same procedure as the sender. It adds all data items (including the checksum); the result is 45. The sum is wrapped and becomes 15. The wrapped sum is complemented and becomes 0. Since the value of the checksum is 0, this means that the data is not corrupted. The receiver drops the checksum and keeps the other data items. If the checksum is not zero, the entire packet is dropped.



Internet Checksum

Traditionally, the Internet has been using a 16-bit checksum. The sender calculates the
checksum by following these steps.

Sender site:

- The message is divided into 16-bit words.
- The value of the checksum word is set to O.
- All words including the checksum are added using one's complement addition.
- The sum is complemented and becomes the checksum.
- The checksum is sent with the data.

Receiver site:

- The message (including checksum) is divided into 16-bit words.
- All words are added using one's complement addition.
- The sum is complemented and becomes the new checksum.
- If the value of checksum is 0, the message is accepted; otherwise, it is rejected

---

3.7 TYPES OF SERVICES PROVIDED TO THE NETWORK LAYER:

- Unacknowledged Connectionless service
- Acknowledged Connectionless service
- Acknowledged Connection-Oriented service

Unacknowledged Connectionless service:

- no recovering of lost or corrupted frame
  - when the error rate is very low
  - real-time traffic, like speech or video
- Losses are taken care of at higher layers
- Used on reliable medium like coax cables or optical fiber, where the error rate is low.
- Appropriate for voice, where delay is worse than bad data.
- It consists of having the source machine send independent frames to the destination machine without having the destination machine acknowledge them.
- Example: Ethernet, Voice over IP, etc. in all the communication channel were real time operation is more important that quality of transmission.

Acknowledged Connectionless service:

- returns information a frame has safely arrived.
  - time-out, resend, frames received twice
  - unreliable channels, such as wireless systems. Useful on unreliable medium like wireless.
- Acknowledgements add delays.

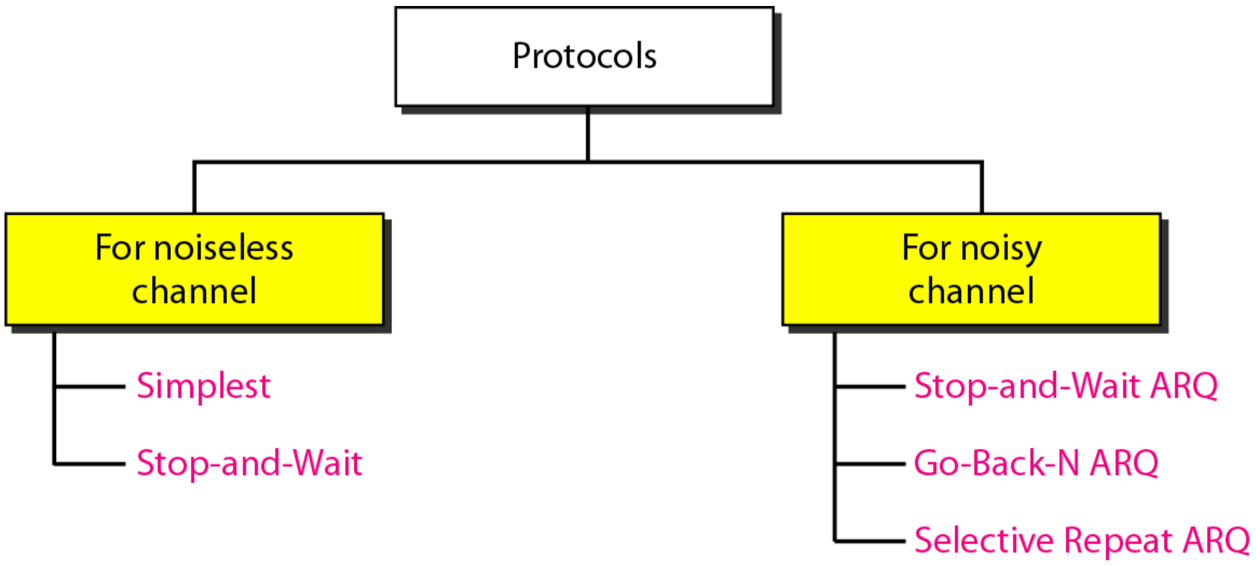
- Adding ack in the DLL rather than in the NL is just an optimization and not a requirement. Leaving it for the NL is inefficient as a large message (packet) has to be resent in that case in contrast to small frames here.
- On reliable channels, like fiber, the overhead associated with the ack is not justified.
- Each frame send by the Data Link layer is acknowledged and the sender knows if a specific frame has been received or lost.
- Typically the protocol uses a specific time period that if has passed without getting acknowledgment it will re-send the frame.
- This service is useful for commutation when an unreliable channel is being utilized (e.g., 802.11 WiFi).
- Network layer does not know frame size of the packets and other restriction of the data link layer. Hence it becomes necessary for data link layer to have some mechanism to optimize the transmission.

<u>Acknowledged Connection-oriented service</u>

- established connection before any data is sent.
- provides the network layer with a reliable bit stream.
- Most reliable, Guaranteed service –
  - Each frame sent is indeed received
  - Each frame is received exactly once
  - Frames are received in order
- Special care has to be taken to ensure this in connectionless services
- Source and Destination establish a connection first.
- Each frame sent is numbered
  - Data link layer guarantees that each frame sent is indeed received.
  - It guarantees that each frame is received only once and that all frames are received in the correct order.
- Exs: Satellite channel communication, Long-distance telephone communication, etc.

3.8 Elementary DLL protocols

- The data link layer can combine framing, flow control, and error control to achieve the delivery of data from one node to another. The protocols are normally implemented in software by using one of the common programming languages.



- Protocols are classified into various types depending on different channels.
  They are:
  1) noiseless channel
  2) noisy channel

3.8.1 Noiseless channel:

- Protocols for noiseless channel are further classified into two
  i) simplest protocol
  ii) stop-and-wait protocol

3.8.1.1Simplest Protocol

- It is a unidirectional protocol in which data frames are traveling in only one direction-from the sender to receiver. We assume that the receiver can immediately handle any frame it receives with a processing time that is small enough to be negligible.
- The data link layer of the receiver immediately removes the header from the frame and hands the data packet to its network layer, which can also accept the packet immediately.
- In other words, the receiver can never be overwhelmed with incoming frames.

Design

- There is no need for flow control in this scheme. The data link layer at the sender site gets data from its network layer, makes a frame out of the data, and sends it. The data link layer at the receiver site receives a frame from its physical layer, extracts data from the frame, and delivers the data to its network layer.
- The data link layers of the sender and receiver provide transmission services for their network layers. The data link layers use the services provided by their physical layers (such as signaling, multiplexing, and so on) for the physical transmission of bits.

- Figure shows a design.

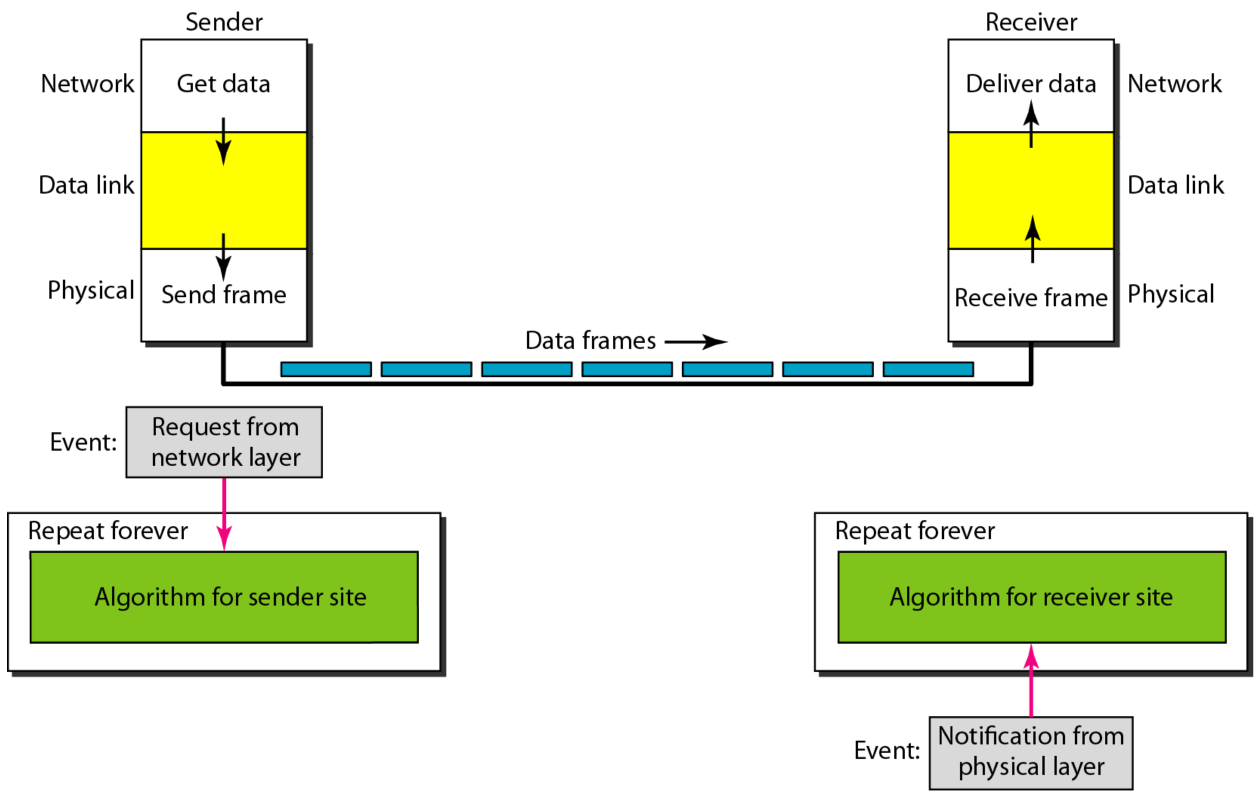


Figure: *The design of the simplest protocol with no flow or error control*

- The sender site cannot send a frame until its network layer has a data packet to send. The receiver site cannot deliver a data packet to its network layer until a frame arrives.
- If the protocol is implemented as a procedure, we need to introduce the idea of events in the protocol. The procedure at the sender site is constantly running; there is no action until there is a request from the network layer.
- The procedure at the receiver site is also constantly running, but there is no action until notification from the physical layer arrives.
- Both procedures are constantly running because they do not know when the corresponding events will occur.
- Figure shows an example of communication using this protocol. It is very simple. The sender sends a sequence of frames without even thinking about the receiver. To send three frames, three events occur at the sender site and three events at the receiver site. Note that the data frames are shown by tilted boxes; the height of the box defines the transmission time difference between the first bit and the last bit in the frame.

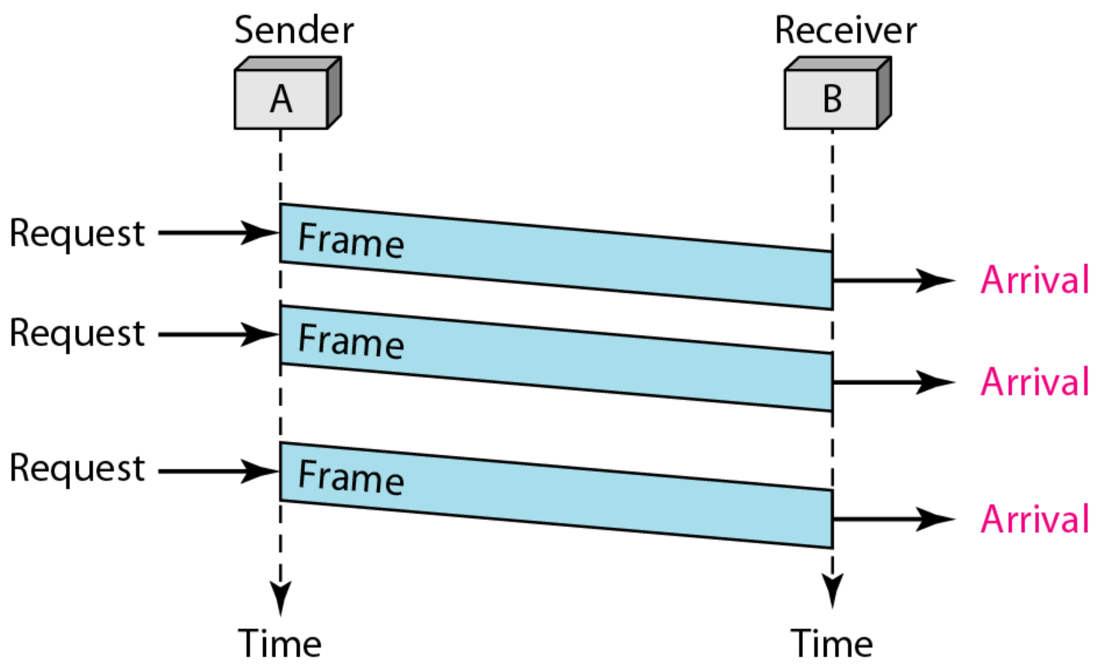


Figure: Flow diagram for above design

<u>3.8.1.2 Stop-and-Wait Protocol</u>

- If data frames arrive at the receiver site faster than they can be processed, the frames must be stored until their use.
- Normally, the receiver does not have enough storage space, especially if it is receiving data from many sources.
- This may result in either the discarding of frames or denial of service. To prevent the receiver from becoming overwhelmed with frames, we somehow need to tell the sender to slow down. There must be feedback from the receiver to the sender.
- The protocol we discuss now is called the Stop-and-Wait Protocol because the sender sends one frame, stops until it receives confirmation from the receiver (okay to go ahead), and then sends the next frame.
- We still have unidirectional communication for data frames, but auxiliary ACK frames (simple tokens of acknowledgment) travel from the other direction.

Design

- Figure illustrates the mechanism. Comparing this figure with above figure, we can see the traffic on the forward channel (from sender to receiver) and the reverse channel. At any time, there is either one data frame on the forward channel or one ACK frame on the reverse channel. We therefore need a half-duplex link.

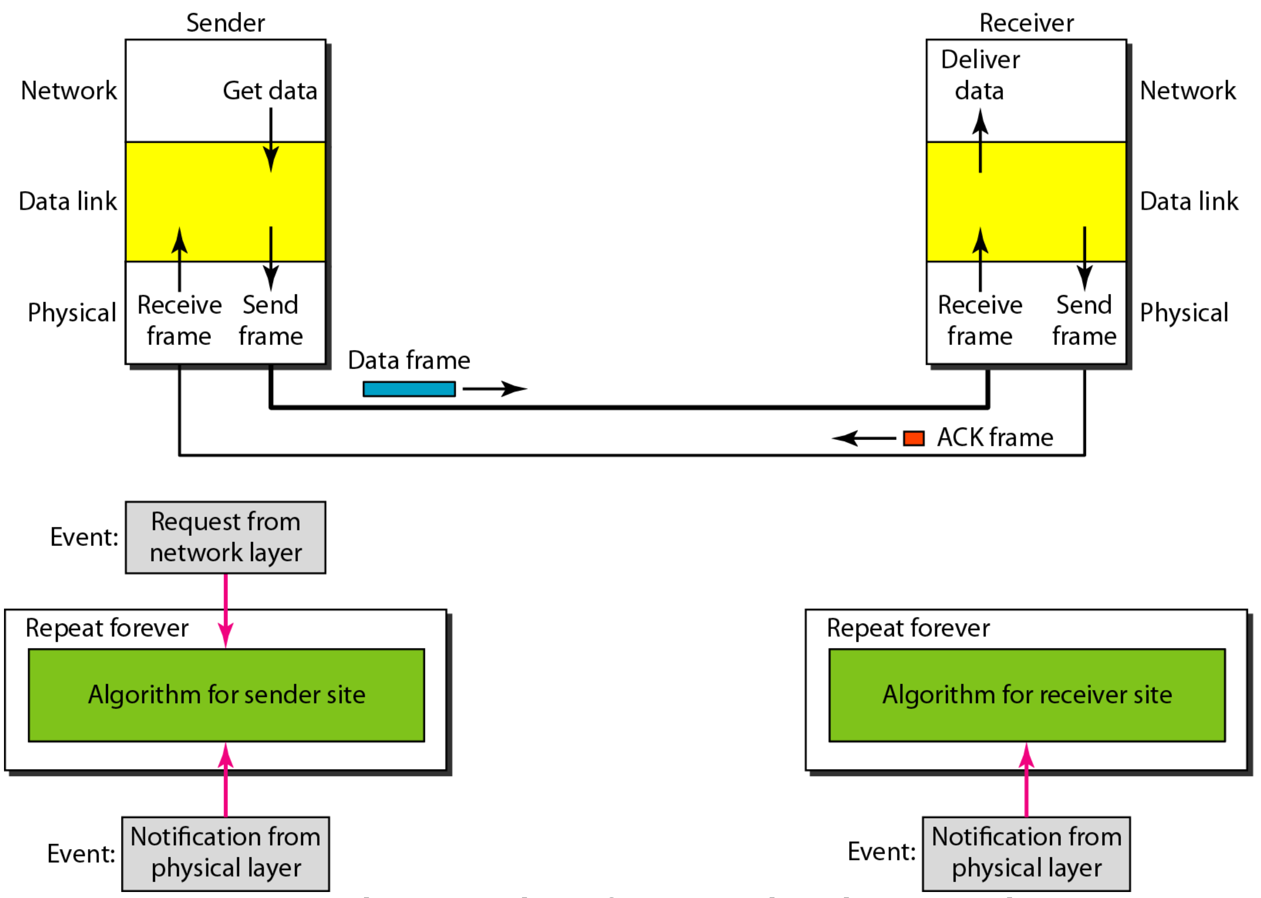


Figure:*Design of Stop-and-Wait Protocol*

Analysis

- Here two events can occur: a request from the network layer or an arrival notification from the physical layer. The responses to these events must alternate. In other words, after a frame is sent, the algorithm must ignore another network layer request until that frame is acknowledged.
- We know that two arrival events cannot happen one after another because the channel is error-free and does not duplicate the frames. The requests from the network layer, however, may happen one after another without an arrival event in between.
- We need somehow to prevent the immediate sending of the data frame. Although there are several methods, we have used a simple *canSend* variable that can either be true or false.
- When a frame is sent, the variable is set to false to indicate that a new network request cannot be sent until *canSend* is true. When an ACK is received, canSend is set to true to allow the sending of the next frame.
- Figure shows an example of communication using this protocol. It is still very simple. The sender sends one frame and waits for feedback from the receiver. When the ACK arrives, the sender sends the next frame. Note

that sending two frames in the protocol involves the sender in four events and the receiver in two events.

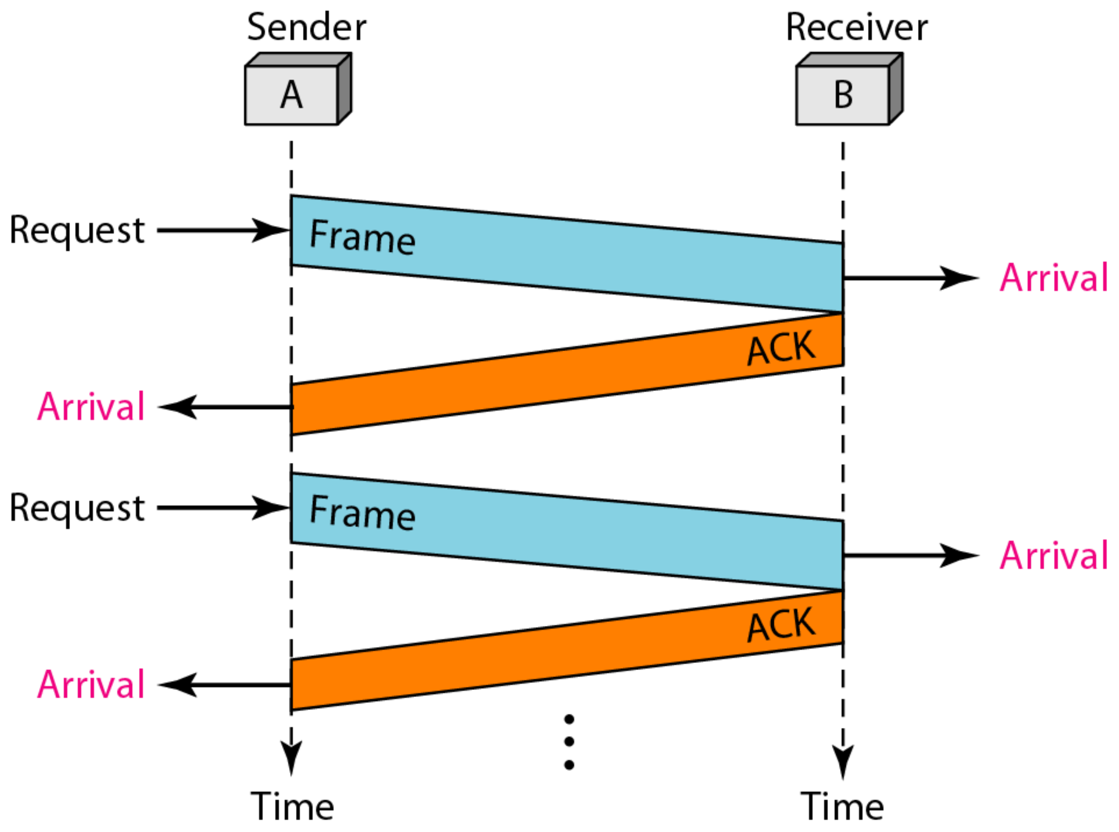


Figure:*Flow diagram for above design*

3.8.2 NOISY CHANNELS

Although the Stop-and-Wait Protocol gives us an idea of how to add flow control to its predecessor, noiseless channels are nonexistent.

3.8.2.1 Stop-and-Wait Automatic Repeat Request

- The Stop-and-Wait Automatic Repeat Request (Stop-andWait ARQ), adds a simple error control mechanism to the Stop-and-Wait Protocol.
- To detect and correct corrupted frames, we need to add redundancy bits to our data frame.When the frame arrives at the receiver site, it is checked and if it is corrupted, it is silently discarded. The detection of errors in this protocol is manifested by the silence of the receiver.
- Lost frames are more difficult to handle than corrupted ones. In our previous protocols,
  there was no way to identify a frame. The received frame could be the correct one, or a duplicate, or a frame out of order. The solution is to number the frames. When the receiver receives a data frame that is out of order, this means that frames were either

lost or duplicated.

- The completed and lost frames need to be resent in this protocol. If the receiver does
  not respond when there is an error, how can the sender know which frame to resend? To remedy this problem, the sender keeps a copy of the sent frame. At the same time, it starts a timer. If the timer expires and there is no ACK for the sent frame, the frame is resent, the copy is held, and the timer is restarted. Since the protocol uses the stop-and-wait mechanism, there is only one specific frame that needs an ACK even though several copies of the same frame can be in the network.
- Since an ACK frame can also be corrupted and lost, it too needs redundancy bits and a sequence number. The ACK frame for this protocol has a sequence number field.
  In this protocol, the sender simply discards a corrupted ACK frame or ignores an out-of-order one.

Sequence Numbers

The protocol specifies that frames need to be numbered. This is done by using sequence numbers. A field is added to the data frame to hold the sequence number of that frame.

One important consideration is the range of the sequence numbers. Since we want to minimize the frame size, we look for the smallest range that provides unambiguous communication. The sequence numbers of course can wrap around. For example, if we decide that the field is *m* bits long, the sequence numbers start from 0, go to 2^m - 1, and then are repeated.

Let us reason out the range of sequence numbers we need. Assume we have used *x* as a sequence number; we only need to use *x* + 1 after that. There is no need for *x* + 2. To show this, assume that the sender has sent the frame numbered *x.* Three things can happen.

1. The frame arrives safe and sound at the receiver site; the receiver sends an acknowledgment.

The acknowledgment arrives at the sender site, causing the sender to send the next frame numbered *x* + 1.

2. The frame arrives safe and sound at the receiver site; the receiver sends an acknowledgment,

but the acknowledgment is corrupted or lost. The sender resends the frame (numbered *x)* after the time-out. Note that the frame here is a duplicate. The receiver can recognize this fact because it expects frame *x* + I but frame *x* was received.

3. The frame is corrupted or never arrives at the receiver site; the sender resends the frame (numbered *x)* after the time-out.

We can see that there is a need for sequence numbers *x* and *x* + I because the receiver needs to distinguish between case 1 and case 2. But there is no need for a frame to be numbered *x* + 2. In case 1, the frame can be numbered *x* again because frames *x* and *x* + 1 are acknowledged and there is no ambiguity at either site. In cases 2 and 3, the new frame is x + I, not x + 2. If only x and x + 1 are needed, we can let x = 0 and x +1 == 1. This means that the sequence is 0, 1, 0, 1, 0, and so on.

In Stop-and-Wait ARQ we use sequence numbers to number the frames. The sequence numbers are based on modul0-2 arithmetic.

Acknowledgment Numbers

Since the sequence numbers must be suitable for both data frames and ACK frames, we use this convention: The acknowledgment numbers always announce the sequence number of the next frame expected by the receiver. For example, if frame 0 has arrived safe and sound, the receiver sends an ACK frame with acknowledgment 1 (meaning frame 1 is expected next). If frame 1 has arrived safe and sound, the receiver sends an ACK frame with acknowledgment 0 (meaning frame 0 is expected).

In Stop-and-Wait ARQ the acknowledgment number always announces in modul0-2 arithmetic the sequence number of the next frame expected.

Design

Figure shows the design of the Stop-and-WaitARQ Protocol. The sending device keeps a copy of the last frame transmitted until it receives an acknowledgment for that frame. A data frames uses a seqNo (sequence number); an ACK frame uses an ackNo (acknowledgment number). The sender has a control variable, which we call *Sn* (sender, next frame to send), that holds the sequence number for the next frame to be sent (0 or 1).

The receiver has a control variable, which we call *Rn* (receiver, next frame expected), that holds the number of the next frame expected. When a frame is sent, the value of *Sn* is incremented (modulo-2), which means if it is 0, it becomes 1 and vice versa. When a frame is received, the value of *Rn* is incremented (modulo-2), which means if it is 0, it becomes 1 and vice versa. Three events can happen at the sender site; one event can happen at the receiver site. Variable *Sn* points to the slot that matches the sequence number of the frame that has been sent, but not acknowledged; *Rn* points to the slot that matches the sequence number of the expected frame.

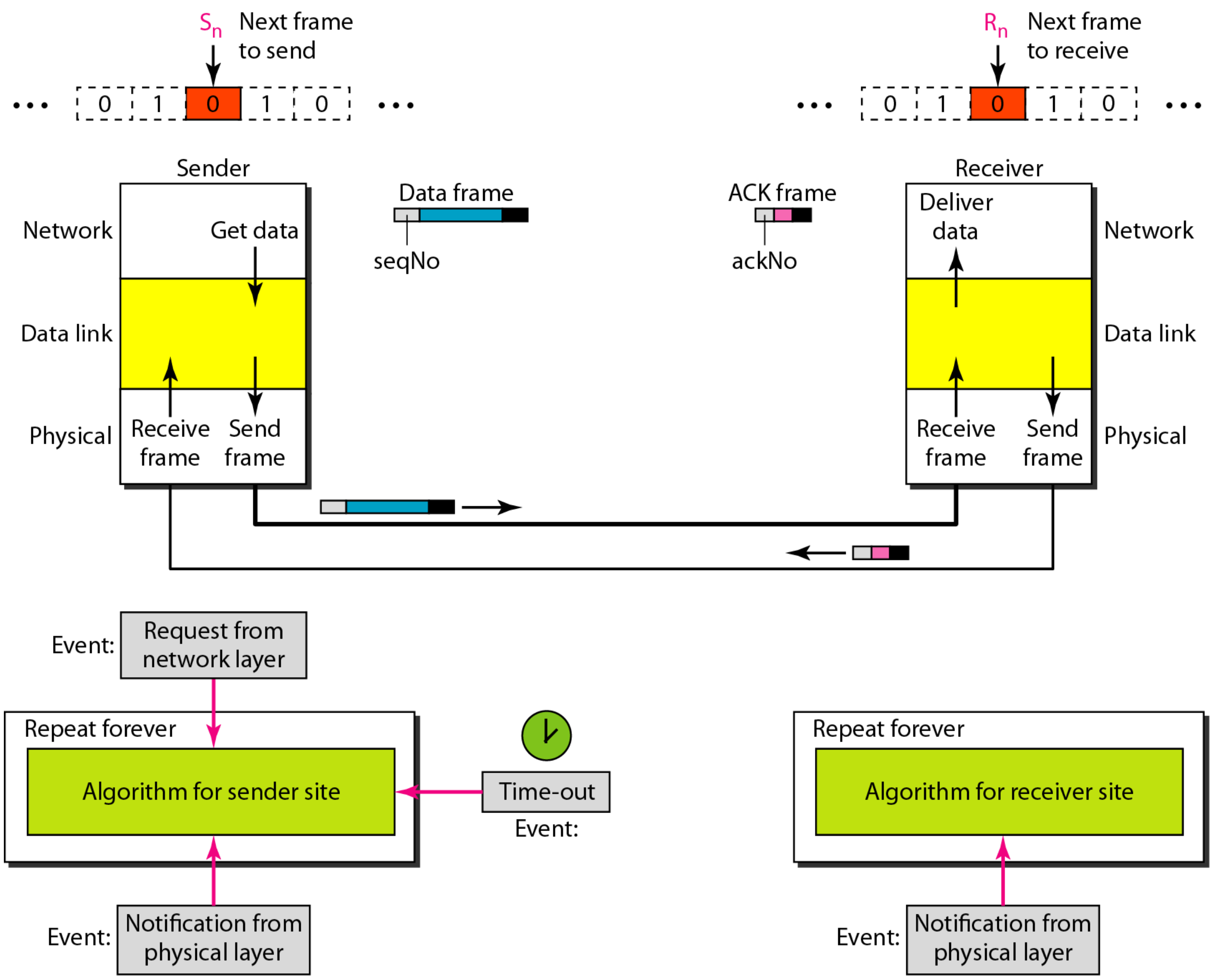


Figure :*Design of the Stop-and-Wait ARQ Protocol*

Below figure shows an example of Stop-and-Wait ARQ. Frame 0 is sent and acknowledged. Frame 1 is lost and resent after the time-out. The resent frame 1 is acknowledged and the timer stops. Frame 0 is sent and acknowledged, but the acknowledgment is lost. The sender has no idea if the frame or the acknowledgment is lost, so after the time-out, it resends frame 0, which is acknowledged.

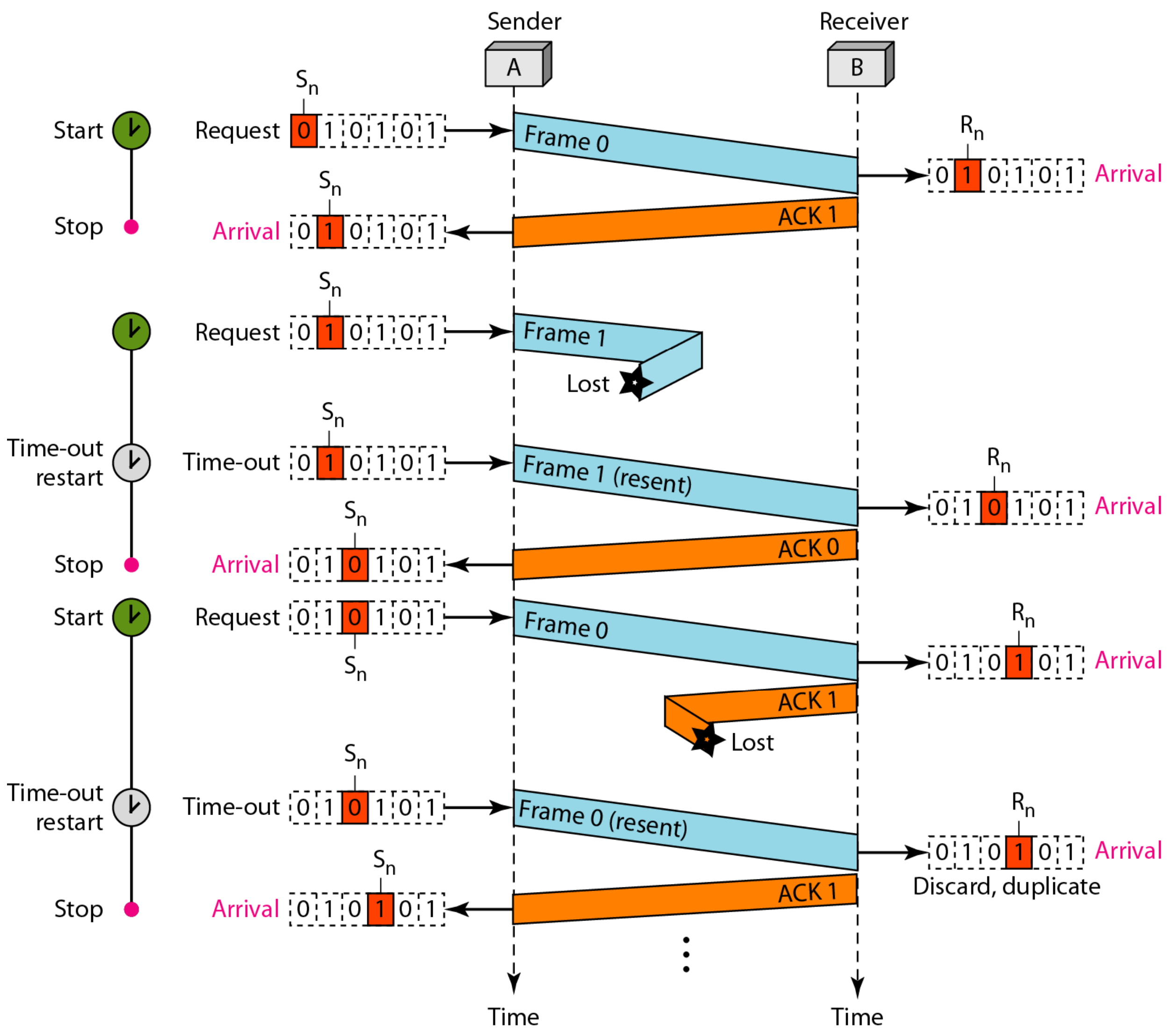


Figure :*Flow diagram for above design*

Data Link Protocols in Use:

PPP: The Internet uses PPP (Point-to-Point) as the primary data link protocol over point-to-point lines between routers and for dialup connections.The PPP frames are byte oriented.

HDLC: In LAN networks HDLC (High-level Data Link Control) protocols are used. These protocols originate from IBM mainframe world.
The HDLC frames are bite oriented.

Assignment-Cum-Tutorial Questions

Section A

Objective Questions

1. When 2 or more bits in a data unit has been changed during the transmission, the error is called [ ]
   a. random error b. burst error
   c. inverted error d. none of the mentioned
2. CRC stands for [ ]
   a. cyclic redundancy check b. code repeat check
   c. code redundancy check d. cyclic repeat check
3. The .... layer provides a well defined service interface to the network layer, determining how the bits of the physical layer are grouped into frames.

a. Data Link b. Physical c. Network d. Session [ ]

4. The different types of services provided by data link layer is/are ...
   a. Unacknowledged connectionless service [ ]

   b. Acknowledged connectionless service

   c. Acknowledged connection oriented service

   d. All of the above.
5. .... is used to indicate to the network layer that an event has happened, for example, establishment or release of a connection. [ ]

a. Request b. Indication c. Response d. Confirm

6. In .... we need to know the exact number of bits that are corrected and more importantly, their location in the message. [ ]

a. error searching b. error detection
c. error correction d. error transmission

7. .... is the process in which the receiver tries to guess the message by using redundant bits. [ ]

a. Forward error correction

b. Backward error correction

c. Transmission

d. Retransmission

8. In block coding, we divide our message into blocks, each of k bits, called .... [ ]

a. Dataword b. Generator c. Codeword d. Checker

9. .... in the data link layer separates a message from one source to a destination, or from other messages to other destinations, by adding a sender address and a destination address. [ ]

a. Transforming b. Framing c. Separating d. Messaging

10. In ...., there is no need for defining the boundaries of the frames; the size itself can be used a delimiter. [ ]

a. Standard Size Framing b. Fixed Size Framing

c. Variable Size Framing d. Constant Size Framing

11.In ...., the sender sends one frame, stops until it receives confirmation from the receiver, and then sends the next frame. [ ]

a. Stop and wait protocol b. Simplest protocol

c. Sliding window protocol d. high level data link control protocol(HDLC)

12. Which of the following is/are the methods used for carrying out framing.

a. Character count [ ]

b. Starting and ending characters, with character stuffing.

c. Starting and ending flags with bit stuffing.

d. All of the above

Section-B

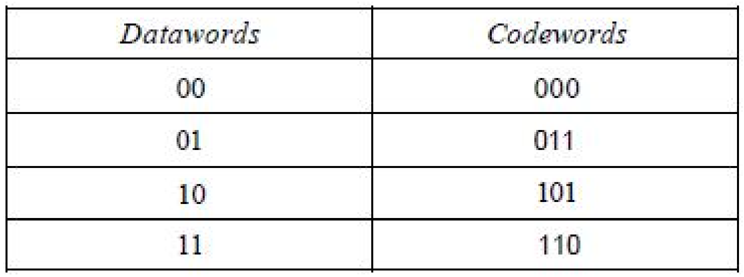
Subjective Questions

1 .What is the Hamming distance? What is the minimum Hamming distance?
2. Define CRC. Describe how CRC works for error detection with an example
3. Define framing and the reason for its need
4. Describe the two protocols in noiseless channels with their algorithms
5. Compare and constrast byte-stuffing and bit-stuffing. Which technique is used in byte-oriented protocols? which technique is used in bit-oriented protocols?
6. Briefly describe the services provided by the data link layer
7. Explain the working of checksum.
8. Explain the working of simple parity checksum with encoder and decoder.

Section-C

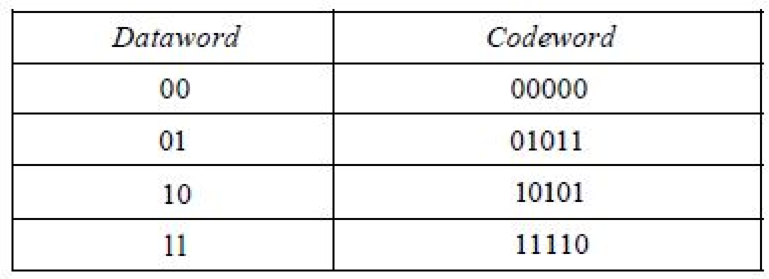
Problems

1.Find the minimum Hamming distance of the coding scheme in the table



Datawords	*Codewords*
00	000
01	011
10	101
11	110

2. Find the minimum Hamming distance of the coding scheme in the table



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

3. A code scheme has a Hamming distance d_{min} == 4. What is the error detection and correction

capability of this scheme?

4. We need a dataword of at least 7 bits. Calculate values of *k* and *n* that satisfy this requirement.

5. *Example 10.15*

Which of the following *g(x)* values guarantees that a single-bit error is caught? For each case,
what is the error that cannot be caught?

a. $x + 1$

b. x^3

c. 1

6. Find the status of the following generators related to two isolated, single-bit errors.

a. $x + 1$

b. $x^4 + l$

c. $x^7 + x^6 + 1$

d. $x^{15} + x^{l4} + 1$

7. Find the suitability of the following generators in relation to burst errors of different lengths.

a. $x^6 + 1$

b. $x^{18} + x^7 + x + 1$

c. $x^{32}+x^3+x^7+1$

8. How can we represent the number 21 in one's complement arithmetic using only four bits?

9. How can we represent the number -6 in one's complement arithmetic using only four bits?

10. Apply the following operations on the corresponding polynomials:

a. $(x^3 + x^2 + x + 1) + (x^4 + x^2 + x + 1)$

b. $(x^3 + x^2 + x + 1) - (x^4 + x^2 + x + 1)$

c. $(x^3 + x^2) \times (x^4 + x^2 + x + 1)$

d. $(x^3 + x^2 + x + 1) \ / \ (x^2 + 1)$

10. Given the dataword 1010011110 and the divisor 10111,

a. Show the generation of the codeword at the sender site (using binary division).

b. Show the checking of the codeword at the receiver site (assume no error).