Module 3

ERROR-DETECTION AND CORRECTION

When bits flow from 1 point to another, they are subject to unpredictable-changes because of interference. The interference can change the shape of the signal.

Types of Errors

- When bits flow from 1 point to another, they are subject to unpredictable-changes '.' of interference.
- The interference can change the shape of the signal.
- Two types of errors:
    1) Single-bit error
    2) Burst-error.

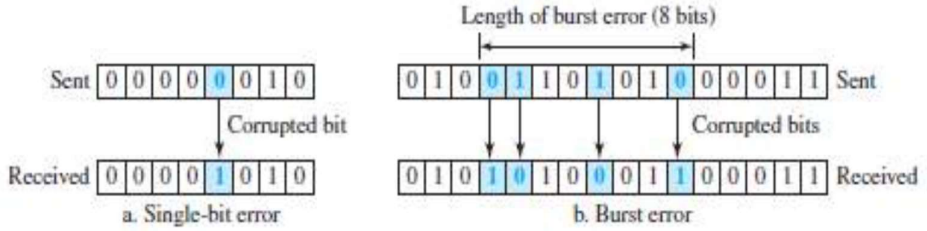


Figure 10.1 *Single-bit and burst error*

1) Single-Bit Error

Only 1 bit of a given data is changed

→ from 1 to 0 or

→ from 0 to 1 (Figure 10.1a).

2) Burst Error

Two or more bits in the data have changed

→ from 1 to 0 or

→ from 0 to 1 (Figure 10.1b).

A burst-error occurs more than a single-bit error. This is because:

- Normally, the duration of noise is longer than the duration of 1-bit.
- When noise affects data, the noise also affects the bits.

The no. of corrupted-bits depends on

→ data-rate and

→ duration of noise.

Redundancy

- The central concept in detecting/correcting errors is *redundancy*.
- Some extra-bits along with the data have to be sent to detect/correct errors. These extra bits are called redundant-bits.
- The redundant-bits are
  → added by the sender and
  → removed by the receiver.
- The presence of redundant-bits allows the receiver to detect/correct errors.

Error Detection vs. Error Correction

- Error-correction is more difficult than error-detection.

1) Error Detection

- Here, we are checking whether any error has occurred or not.
- The answer is a simple YES or NO.
- We are not interested in the number of corrupted-bits.

2) Error Correction

- Here, we need to know
  → exact number of corrupted-bits and
  → location of bits in the message.
- Two important factors to be considered:
  1) Number of errors and
  2) Message-size.

Coding

- Redundancy is achieved through various coding-schemes.
  1) Sender adds redundant-bits to the data-bits. This process creates a relationship between
  → redundant-bits and
  → data-bits.
  2) Receiver checks the relationship between redundant-bits & data-bits to detect/correct errors.
- Two important factors to be considered:

1) Ratio of redundant-bits to the data-bits and

2) Robustness of the process.

- Two broad categories of coding schemes: 1) Block-coding and 2) Convolution coding.

Block Coding

- The message is divided into *k*-bit blocks. These blocks are called data-words.
- Here, r-redundant-bits are added to each block to make the length $n=k+r$.
- The resulting *n*-bit blocks are called code-words.
- Since $n>k$, the number of possible code-words is larger than the number of possible data-words.
- Block-coding process is 1-to-1; the same data-word is always encoded as the same code-word.
- Thus, we have 2^n-2^k code-words that are not used. These code-words are invalid or illegal.

Error Detection

- If the following 2 conditions are met, the receiver can detect a change in the original code-word:

  1) The receiver has a list of valid code-words.

  2) The original code-word has changed to an invalid code-words.

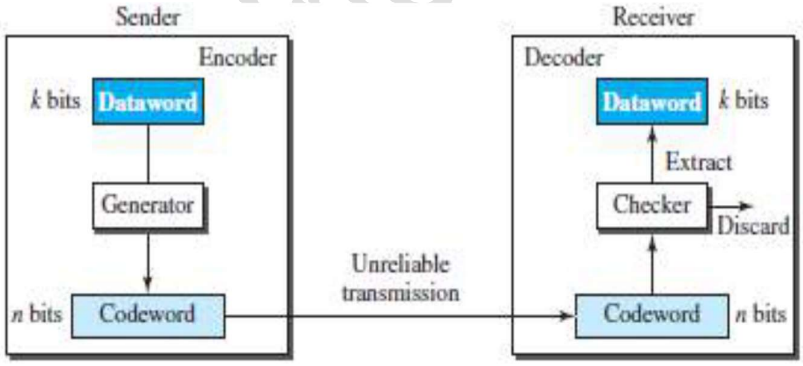


Figure 10.2 *Process of error detection in block coding*

Here is how it works (Figure 10.2):

1) At Sender

i) The sender creates code-words out of data-words by using a generator.

The generator applies the rules and procedures of encoding.

ii) During transmission, each code-word sent to the receiver may change.

2) At Receiver

i) a) If the received code-word is the same as one of the valid code-words, the code-word is accepted; the corresponding data-word is extracted for use.

b) If the received code-word is invalid, the code-word is discarded.

ii) However, if the code-word is corrupted but the received code-word still matches a valid codeword, the error remains undetected.

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

Example 10.1

Let us assume that k = 2 and n = 3. Table 10.1 shows the list of datawords and codewords. Later, we will see how to derive a codeword from a dataword.

Table 10.1 *A code for error detection in Example 10.1*

Dataword	*Codeword*	*Dataword*	*Codeword*
00	000	10	101
01	011	11	110

Assume the sender encodes the dataword 01 as 011 and sends it to the receiver. Consider the following cases:

1. The receiver receives 011. It is a valid codeword. The receiver extracts the dataword 01 from it.
2. The codeword is corrupted during transmission, and 111 is received (the leftmost bit is corrupted). This is not a valid codeword and is discarded.
3. The codeword is corrupted during transmission, and 000 is received (the right two bits are corrupted). This is a valid codeword. The receiver incorrectly extracts the dataword 00. Two corrupted bits have made the error undetectable.

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

Hamming Distance

• The main concept for error-control: Hamming distance.

• **The Hamming distance between 2 words is the number of differences between the corresponding bits**.

• Let ***d(x , y)*** = Hamming distance between 2 words x and y.

• Hamming distance can be found by

→ applying the XOR operation on the 2 codewords and

→ counting the number of 1s in the result.

• For example:

1) The Hamming distance d(000, 011) is 2 because 000⊕011= 011 (two 1s).

2) The Hamming distance d(10101, 11110) is 3 because 10101⊕11110 = 01011 (three 1s).

Hamming Distance and Error

- Hamming distance between the received word and the sent code-word is the number of bits that are corrupted during transmission.

  For example: Let Sent code-word = 00000

  Received word = 01101

  Hamming distance = d(00000, 01101) =3. Thus, 3 bits are in error.

Minimum Hamming Distance

• Minimum Hamming distance is the smallest Hamming distance between all possible pairs of code-words.

• Let $\mathbf{d_{min}}$ = Minimum Hamming distance.

• To find $\mathbf{d_{min}}$ value, we find the Hamming distances between all words and select the smallest one.

Minimum Hamming distance for Error-detection

- If '*s*' errors occur during transmission, the Hamming distance b/w the sent code-word and received code-word is 's' (Figure 10.3).
- If code has to detect upto 's' errors, the minimum-distance b/w the valid codes must be 's+1' i.e. $\mathbf{d_{min}=s+1.}$
- We use a geometric approach to define dmin=s+1.

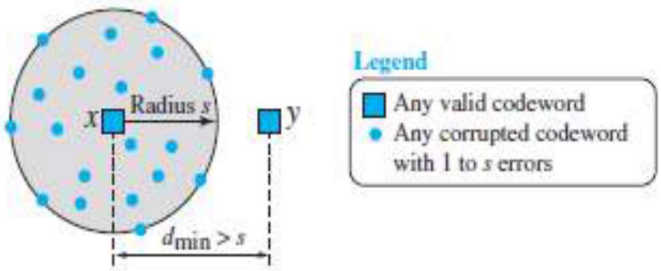


Figure 10.3 *Geometric concept explaining d_{min} in error detection*

- Let us assume that the sent code-word x is at the center of a circle with radius s.
- All received code-words that are created by 0 to s errors are points inside the circle or on the perimeter of the circle.
- All other valid code-words must be outside the circle

  For example: A code scheme has a Hamming distance $d_{min} = 4$.

  This code guarantees the detection of upto 3 errors (d = s + 1 or s = 3).

Linear Block Codes

• Almost all block codes belong to a subset of block codes called linear block codes.

• A linear block code is a code in which the XOR of 2 valid code-words creates another valid code-word.

(XOR →Addition modulo-2)

Table 10.1 *A code for error detection*

Dataword	Codeword	Dataword	Codeword
00	000	10	101
01	011	11	110

• The code in Table 10.1 is a linear block code because the result of XORing any code-word with any other code-word is a valid code-word.

For example, the XORing of the 2nd and 3rd code-words creates the 4th one.

Minimum Distance for Linear Block Codes

• Minimum Hamming distance is no. of 1s in the nonzero valid code-word with the smallest no. of 1s.

• In Table 10.1,

The numbers of 1s in the nonzero code-words are 2, 2, and 2.

So the minimum Hamming distance is $d_{min} = 2$.

Parity Check Code

• This code is a linear block code. This code can detect an odd number of errors.

• A k-bit data-word is changed to an n-bit code-word where n=k+1.

• One extra bit is called the parity-bit.

• The parity-bit is selected to make the total number of 1s in the code-word even.

• Minimum hamming distance $d_{min} = 2$. This means the code is a single-bit error-detecting code.

Table 10.2 *Simple parity-check code C(5, 4)*

Dataword	*Codeword*	*Dataword*	*Codeword*
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**

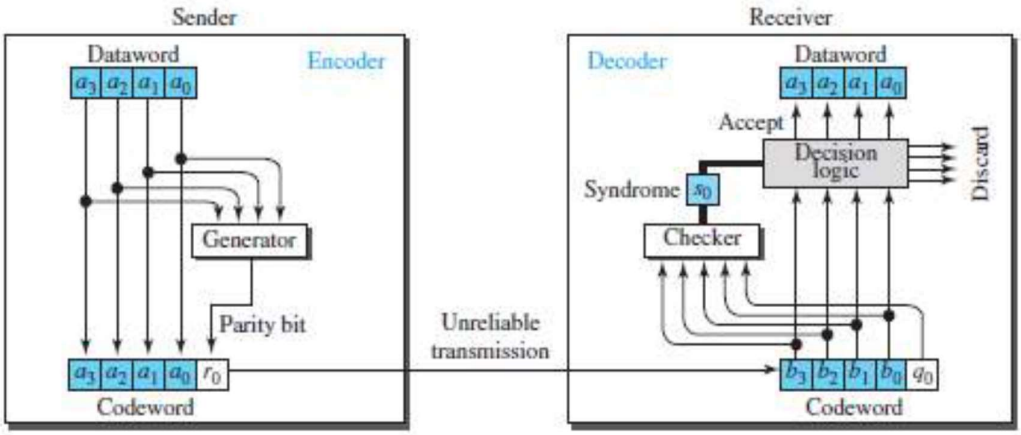


Figure 10.4 *Encoder and decoder for simple parity-check code*

• Here is how it works (Figure 10.4):

1) At Sender

- The encoder uses a generator that takes a copy of a 4-bit data-word (a0, a1, a2, and a3) and generates a parity-bit r0.
- The encoder

  → accepts a copy of a 4-bit data-word (a0, a1, a2, and a3) and

  → generates a parity-bit r0 using a generator

  → generates a 5-bit code-word
- The parity-bit & 4-bit data-word are added to make the number of 1s in the code-word even.

- The addition is done by using the following:

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

- The result of addition is the parity-bit.

  1) If the no. of 1s in data-word = even, result = 0. (r_0=0)

  2) If the no. of 1s in data-word = odd, result = 1. (r_0=1)

  3) In both cases, the total number of 1s in the code-word is even.

- The sender sends the code-word, which may be corrupted during transmission.

2) At Receiver

- The receiver receives a 5-bit word.
- The checker performs the same operation as the generator with one exception:

The addition is done over all 5 bits.

$$s_0 = b_3 + b_2 + b_1 + b_0 + q_0 \quad \text{(modulo-2)}$$

- The result is called the syndrome bit (so).
- Syndrome bit = 0 when the no. of 1s in the received code-word is even; otherwise, it is 1.
- The syndrome is passed to the decision logic analyzer.

  1) If s0=0, there is no error in the received code-word. The data portion of the received code-word is accepted as the data-word.

  2) If s0=1, there is error in the received code-word. The data portion of the received code-word is discarded. The data-word is not created.

Cyclic Codes

• Cyclic codes are special linear block codes with one extra property:

If a code-word is cyclically shifted (rotated), the result is another code-word.

For ex: if code-word = 1011000 and we cyclically left-shift, then another code-word = 0110001.

• Let First-word = a0 to a6 and Second-word = b0 to b6, we can shift the bits by using the following:

$$b_1 = a_0 \quad b_2 = a_1 \quad b_3 = a_2 \quad b_4 = a_3 \quad b_5 = a_4 \quad b_6 = a_5 \quad b_0 = a_6$$

Cyclic Redundancy Check (CRC)

• CRC is a cyclic code that is used in networks such as LANs and WANs.

Table 10.3 *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

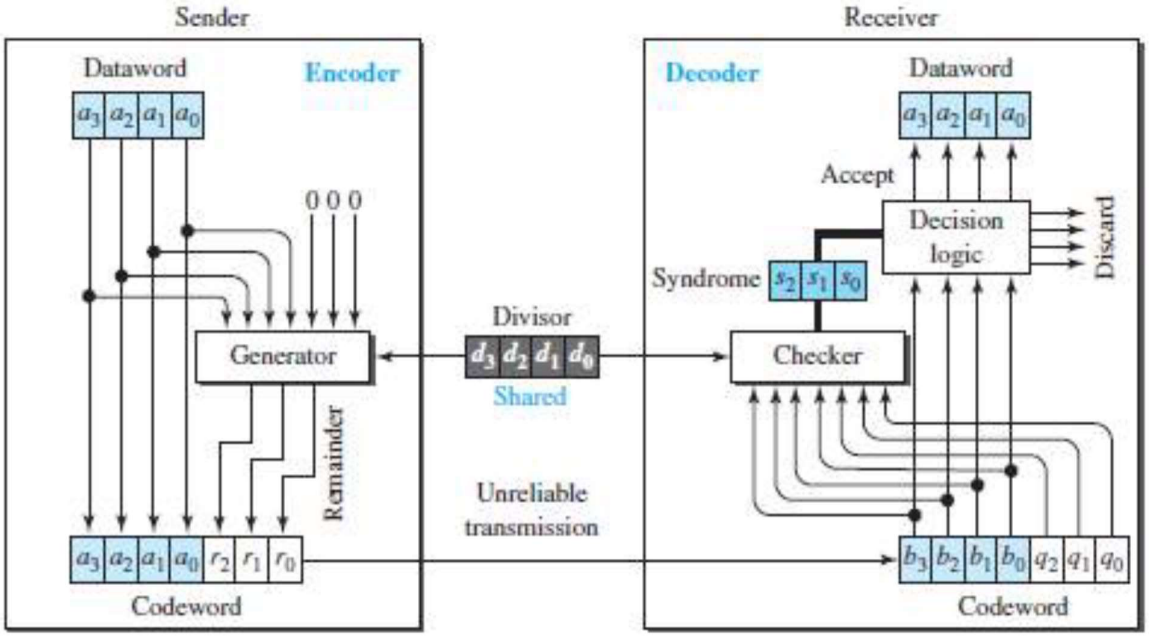


Figure 10.5 *CRC encoder and decoder*

• Let Size of data-word = k bits (here k=4).

Size of code-word = n bits (here n=7).

Size of divisor = n-k+1 bits (here n-k+1=4). (Augmented → increased)

• Here is how it works (Figure 10.5):

1) At Sender

- n-k 0s is appended to the data-word to create augmented data-word. (here n-k=3).
- The augmented data-word is fed into the generator (Figure 10.6).
- The generator divides the augmented data-word by the divisor.
- The remainder is called check-bits ($r_2r_1r_0$).
- The check-bits ($r_2r_1r_0$) are appended to the data-word to create the code-word.

2) At Receiver

- The possibly corrupted code-word is fed into the checker.
- The checker is a replica of the generator.
- The checker divides the code-word by the divisor.
- The remainder is called syndrome bits ($r_2r_1r_0$).
- The syndrome bits are fed to the decision-logic-analyzer.
- The decision-logic-analyzer performs following functions:

i) For No Error

- If all syndrome-bits are 0s, the received code-word is accepted.
- Data-word is extracted from received code-word (Figure 10.7a).

ii) For Error

- If all syndrome-bits are not 0s, the received code-word is discarded (Figure 10.7b).

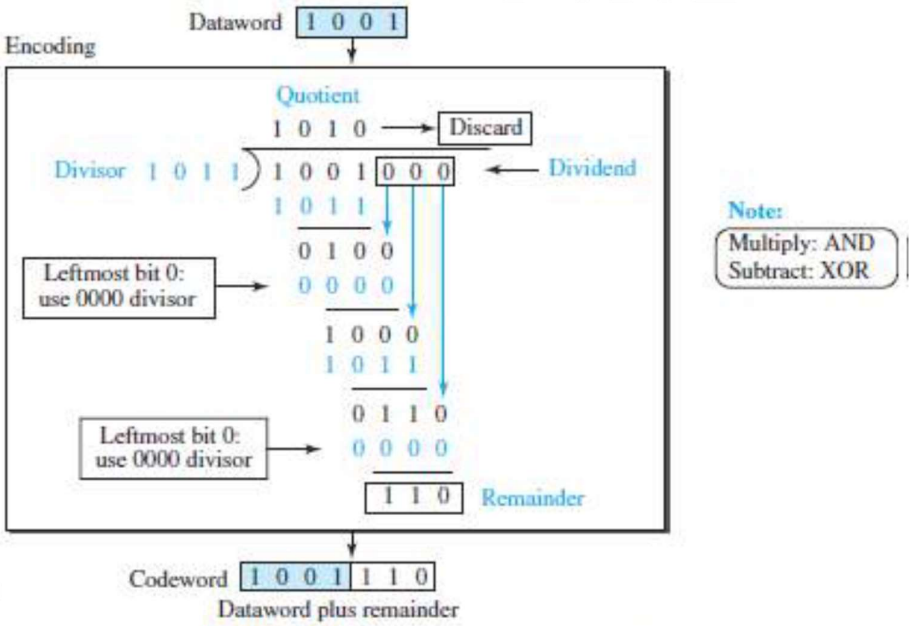


Figure 10.6 *Division in CRC encoder*

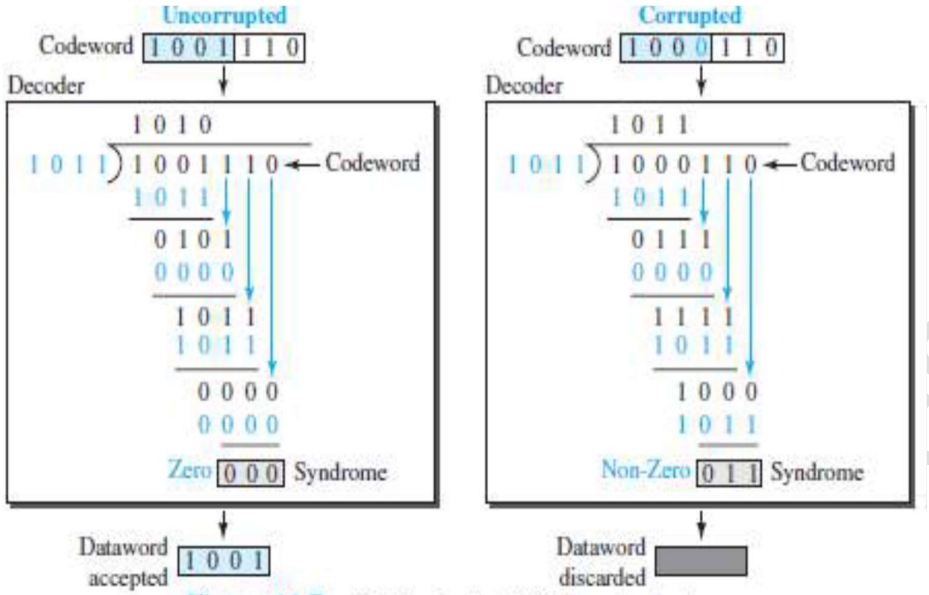
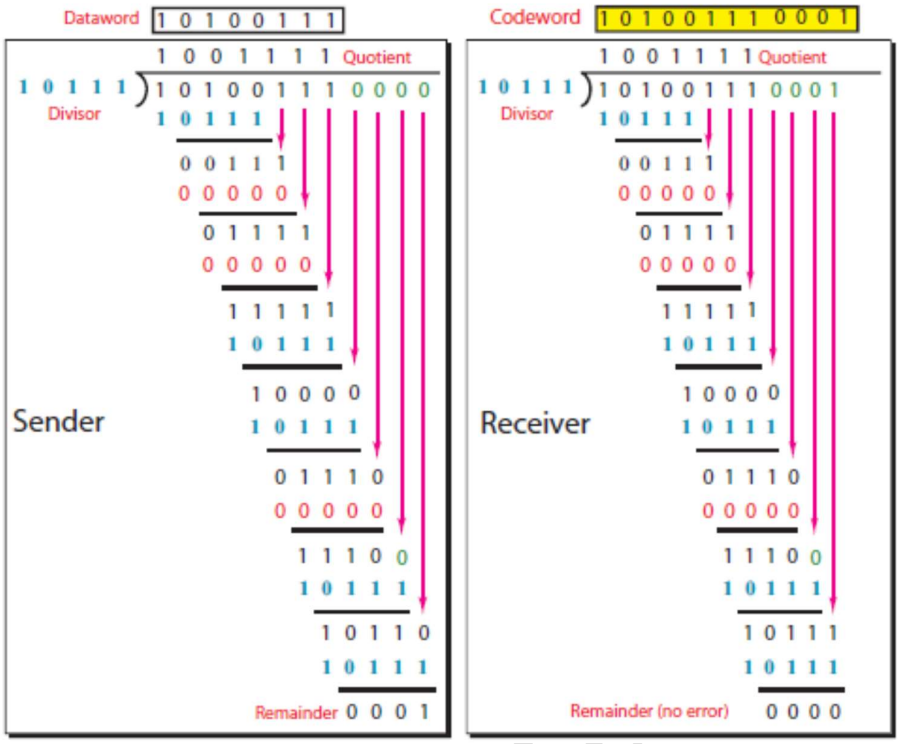


Figure 10.7 *Division in the CRC decoder for two cases*

Example:

Given the dataword 101001111 and the divisor 10111, show the generation of the CRC codeword at the sender site (using binary division).



Polynomials

• A pattern of 0s and 1s can be represented as a polynomial with coefficients of 0 and 1 (Figure 10.8).

• The power of each term shows the position of the bit; the coefficient shows the value of the bit.

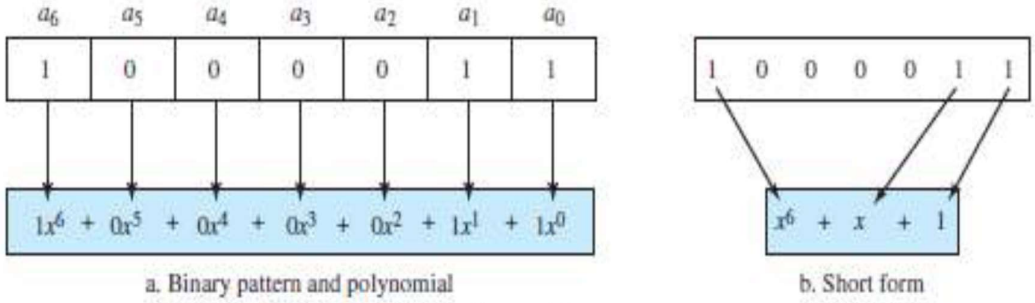


Figure 10.8 *A polynomial to represent a binary word*

Cyclic Code Encoder Using Polynomials

- Let Data-word = 1001 = x^3+1.
- Divisor = 1011 = x^3+x+1.

• In polynomial representation, the divisor is referred to as generator polynomial g(x) (Figure 10.9).

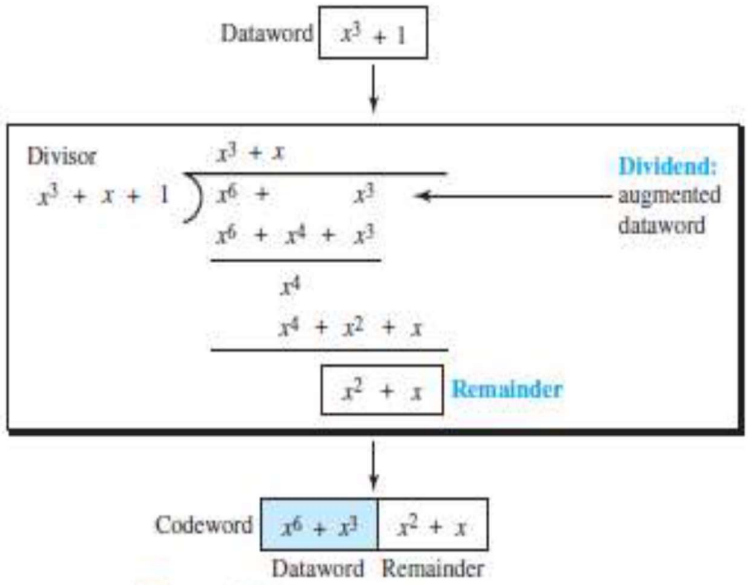


Figure 10.9 *CRC division using polynomials*

Cyclic Code Analysis

• We define the following, where $f(x)$ is a polynomial with binary coefficients:

Dataword: $d(x)$ **Codeword: $c(x)$** **Generator: $g(x)$** **Syndrome: $s(x)$** **Error: $e(x)$**

In a cyclic code,

1. If $s(x) \neq 0$, one or more bits is corrupted.
2. If $s(x) = 0$, either
   a. No bit is corrupted, or
   b. Some bits are corrupted, but the decoder failed to detect them.

Single Bit Error

• If the generator has more than one term and the coefficient of x^0 is 1, all single-bit errors can be caught.

Two Isolated Single-Bit Errors

• If a generator cannot divide x^i+1 (t between 0 & n-1), then all isolated double errors can be detected (Figure 10.10).

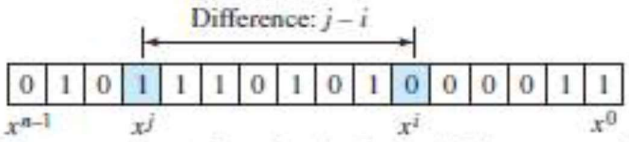


Figure 10.10 *Representation of two isolated single-bit errors using polynomials*

Odd Numbers of Errors

• A generator that contains a factor of x+1 can detect all odd-numbered errors.

A good polynomial generator needs to have the following characteristics:

1. **It should have at least two terms.**
2. **The coefficient of the term x^0 should be 1.**
3. **It should not divide $x^t + 1$, for t between 2 and $n - 1$.**
4. **It should have the factor $x + 1$.**

Burst Error

Now let us extend our analysis to the burst error, which is the most important of all. A burst error is of the form $e(x) = (x^j + \ldots + x^i)$. Note the difference between a burst error and two isolated single-bit errors. The first can have two terms or more; the second can only have two terms. We can factor out x^i and write the error as $x^i(x^{j-i} + \ldots + 1)$. If our generator can detect a single error (minimum condition for a generator), then it cannot divide x^i. What we should worry about are those generators that divide $x^{j-i} + \ldots + 1$. In other words, the remainder of $(x^{j-i} + \ldots + 1)/(x^r + \ldots + 1)$ must not be zero. If $j\text{-}i < r$, the remainder can never be zero. We can write $j - i = L\text{-}1$, where L is the length of the error. So $L - 1 < r$ or $L < r + 1$ or $L = r$.

- **All burst errors with $L \leq r$ will be detected.**
- **All burst errors with $L = r + 1$ will be detected with probability $1 - (1/2)^{r-1}$.**
- **All burst errors with $L > r + 1$ will be detected with probability $1 - (1/2)^r$.**

Table 10.4 *Standard polynomials*

Name	*Polynomial*	*Used in*
CRC-8	$x^8 + x^2 + x + 1$ **100000111**	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$ **11000110101**	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$ **10001000000100001**	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ **10000010011000001000111011011011**	LANs

Advantages of Cyclic Codes

• The cyclic codes have a very good performance in detecting

→ single-bit errors

→ double errors

→ odd number of errors and

→ burst-errors.

• They can easily be implemented in hardware and software. They are fast when implemented in hardware.

Checksum

• Checksum is an error-detecting technique.

• In the Internet,

→ The checksum is mostly used at the network and transport layer.

→ The checksum is not used in the data link layer.

• Like linear and cyclic codes, the checksum is based on the concept of redundancy.

• Here is how it works (Figure 10.15):

1) At Source

- Firstly the message is divided into m-bit units.
- Then, the generator creates an extra m-bit unit called the checksum.
- The checksum is sent with the message.

2) At Destination

- The checker creates a new checksum from the combination of the message and sent checksum.

  i) If the new checksum is all 0s, the message is accepted.

  ii) If the new checksum is not all 0s, the message is discarded.

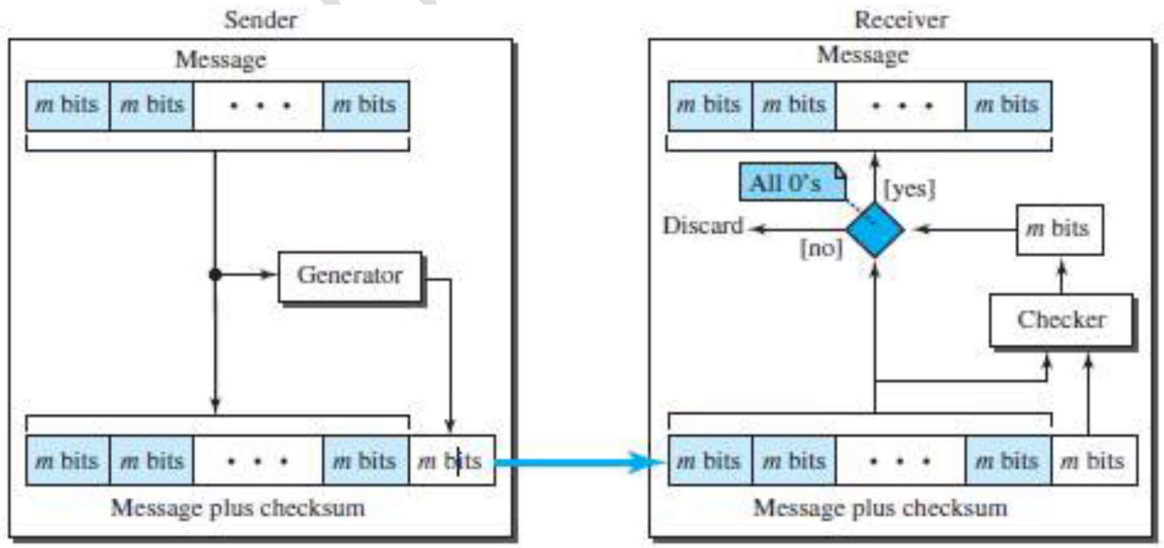


Figure 10.15 *Checksum*

Concept of Checksum

Consider the following example:

Example 3.4

• 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:

Let set of numbers = (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 result & the sum 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.

Example 3.5

• To make the job of the receiver easy 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

• The previous example has one major drawback.

All of our data can be written as a 4-bit word (they are less than 15) except for the checksum.

• Solution: Use one's complement arithmetic.

- We can represent unsigned numbers between 0 and 2n-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).
- A negative number can be represented by inverting all bits (changing 0 to 1 and 1 to 0).
- This is the same as subtracting the number from 2n-1.

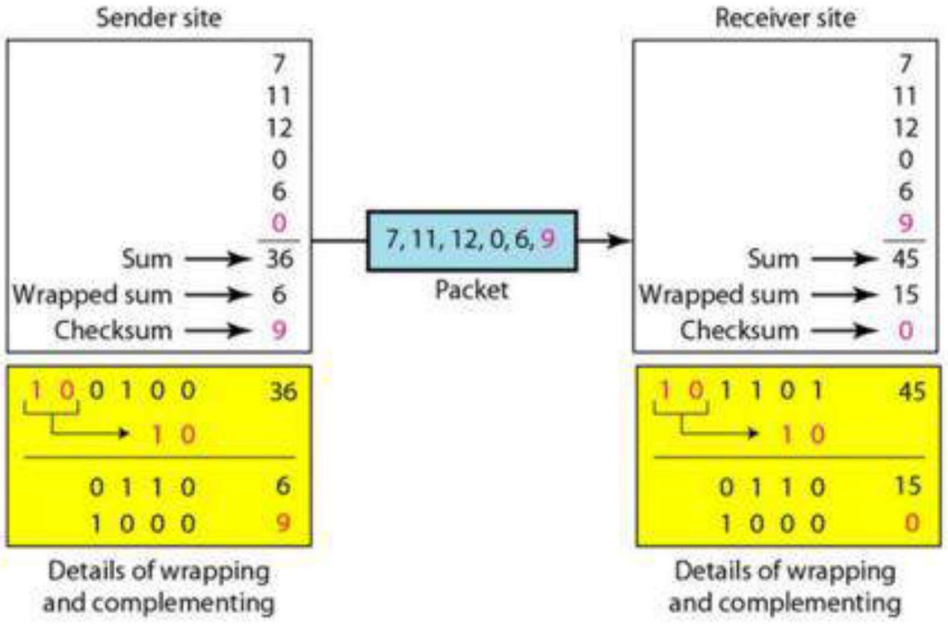


Figure 10.16

• Here is how it works (Figure 10.16):

1) At Sender

- The sender initializes the checksum to 0 and adds all data items and the checksum.
- 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.
- 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.

2) At Receiver

- 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 or the receiver uses five steps.

Table 10.5 *Procedure to calculate the traditional checksum*

Sender	*Receiver*
1. The message is divided into 16-bit words.	1. The message and the checksum are received.
2. The value of the checksum word is initially set to zero.	2. The message is divided into 16-bit words.
3. All words including the checksum are added using one's complement addition.	3. All words are added using one's complement addition.
4. The sum is complemented and becomes the checksum.	4. The sum is complemented and becomes the new checksum.
5. The checksum is sent with the data.	5. If the value of the checksum is 0, the message is accepted; otherwise, it is rejected.

Algorithm:

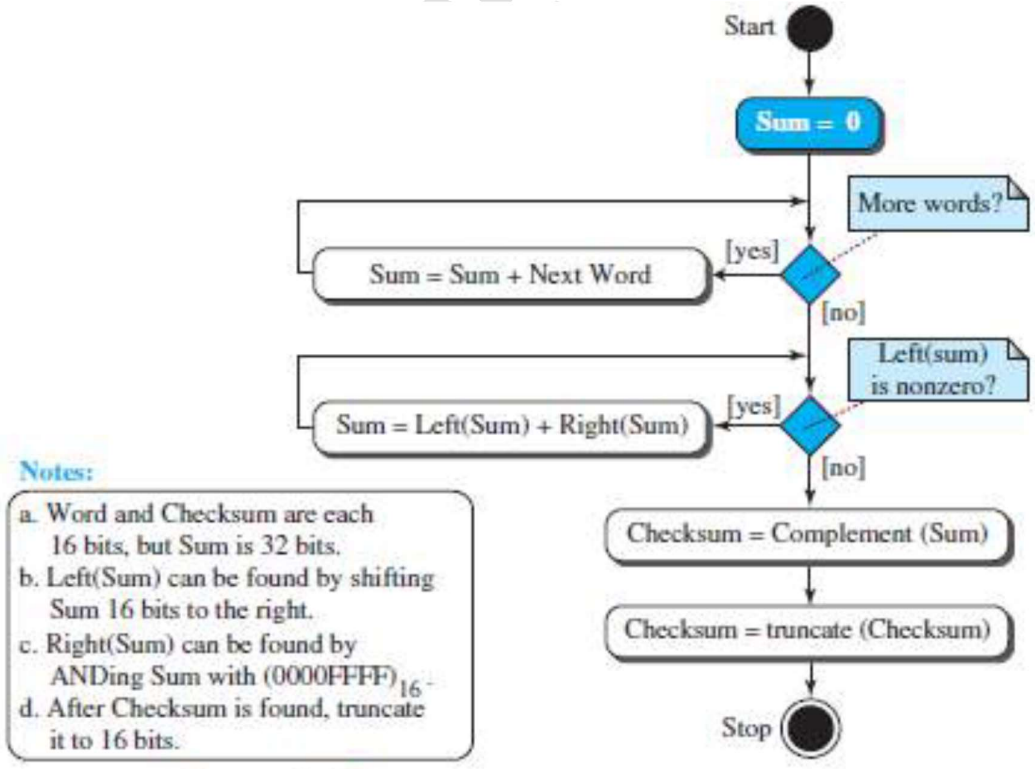


Figure 10.17 *Algorithm to calculate a traditional checksum*

Other Approaches to the Checksum

• If two 16-bit items are transposed in transmission, the checksum cannot catch this error.

• The reason is that the traditional checksum is not weighted: it treats each data item equally.

• In other words, the order of data items is immaterial to the calculation.

• Two approaches have been used to prevent this problem: 1) Fletcher and 2) Adler

Fletcher Checksum

• The Fletcher checksum was devised to weight each data item according to its position.

• Fletcher has proposed two algorithms: 8-bit and 16-bit (Figure 10.18).

• The first, 8-bit Fletcher, calculates on 8-bit data items and creates a 16-bit checksum. The second, 16-bit Fletcher, calculates on 16-bit data items and creates a 32-bit checksum.

• The 8-bit Fletcher is calculated over data octets (bytes) and creates a 16-bit checksum.

• The calculation is done modulo 256 (2^8), which means the intermediate results are divided by 256 and the remainder is kept.

• The algorithm uses two accumulators, L and R.

• The first simply adds data items together;

The second adds a weight to the calculation.

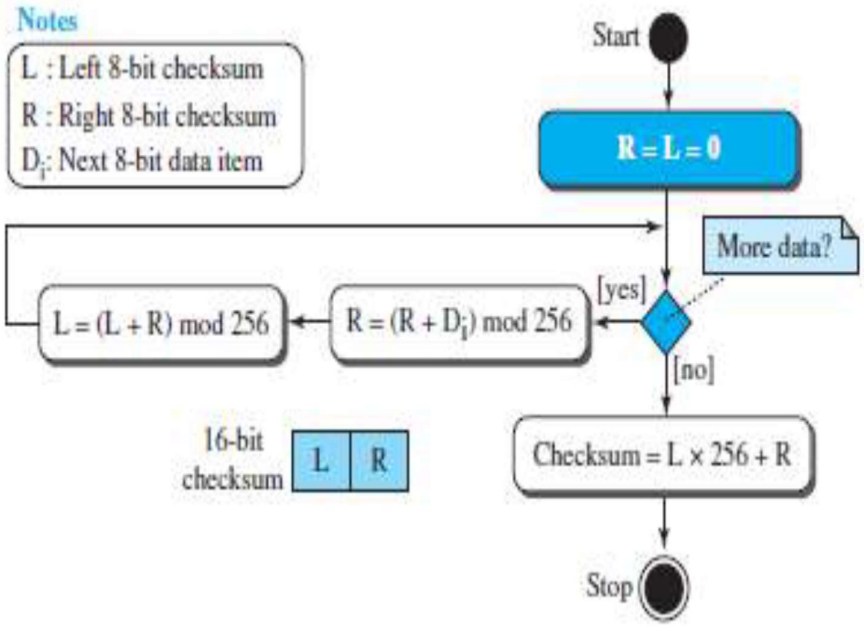


Figure 10.18 *Algorithm to calculate an 8-bit Fletcher checksum*

Adler Checksum

• The Adler checksum is a 32-bit checksum.

• It is similar to the 16-bit Fletcher with three differences (Figure 10.19).

1) Calculation is done on single bytes instead of 2 bytes at a time.

2) The modulus is a prime number (65,521) instead of 65,536.

3) L is initialized to 1 instead of 0.

• A prime modulo has a better detecting capability in some combinations of data.

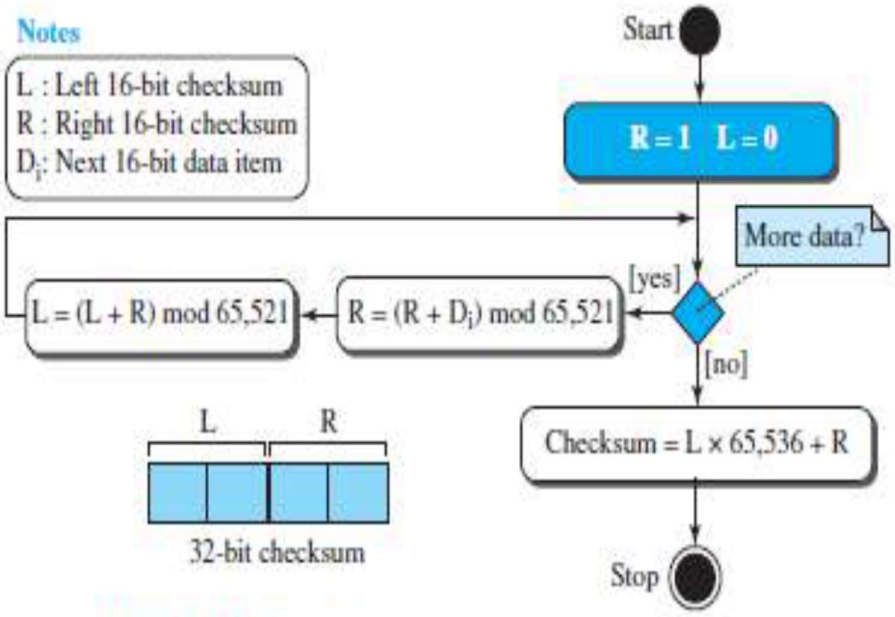


Figure 10.19 *Algorithm to calculate an Adler checksum*

Forward Error Correction

• Retransmission of corrupted and lost packets is not useful for real-time multimedia transmission because it creates an unacceptable delay in reproducing: we need to wait until the lost or corrupted packet is resent.

• We need to correct the error or reproduce the packet immediately.

• Several schemes have been designed and used that are collectively referred to as Forward Error Correction (FEC) techniques.

Using Hamming Distance

• To detect t errors, we need to have $d_{min} = 2t + 1$ (Figure 10.20).

• In other words, if we want to correct 10 bits in a packet, we need to make the minimum hamming distance 21 bits, which means a lot of redundant bits need to be sent with the data.

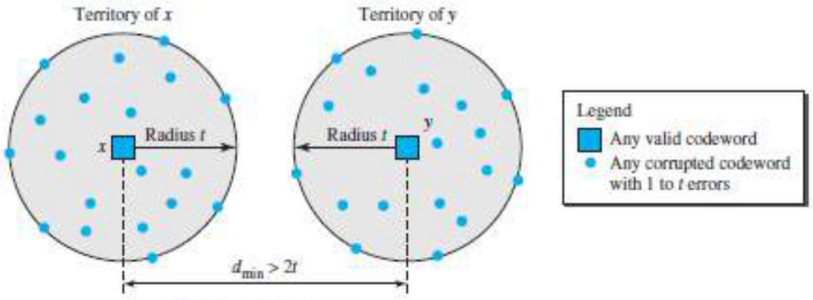


Figure 10.20 *Hamming distance for error correction*

Using XOR

• Use the property of the exclusive OR operation as shown below.

$$R = P_1 \oplus P_2 \oplus \ldots \oplus P_i \oplus \ldots \oplus P_N \quad \rightarrow \quad P_i = P_1 \oplus P_2 \oplus \ldots \oplus R \oplus \ldots \oplus P_N$$

• We divide a packet into N chunks, create the exclusive OR of all the chunks and send $N + 1$ chunks.

• If any chunk is lost or corrupted, it can be created at the receiver site.

• If N = 4, it means that we need to send 25 percent extra data and be able to correct the data if only one out of four chunks is lost.

Chunk Interleaving

• Another way to achieve FEC in multimedia is to allow some small chunks to be missing at the receiver.

• We cannot afford to let all the chunks belonging to the same packet be missing.

However, we can afford to let one chunk be missing in each packet.

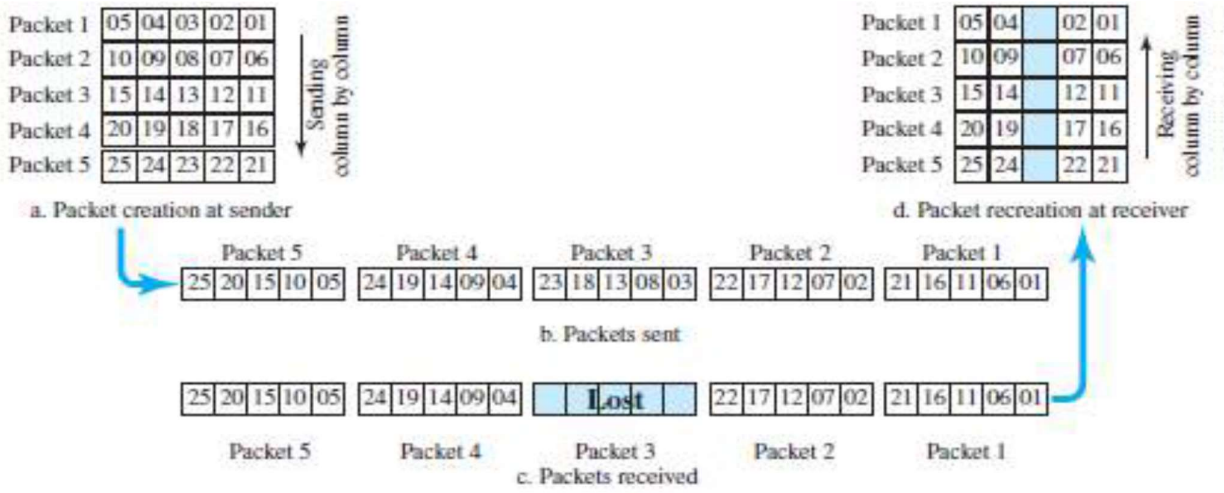


Figure 10.21 *Interleaving*

• In Figure 10.21, each packet is divided into 5 chunks (normally the number is much larger).

• Then, we can create data chunk-by-chunk (horizontally), but combine the chunks into packets vertically.

• In this case, each packet sent carries a chunk from several original packets.

• If the packet is lost, we miss only one chunk in each packet.

• Normally, missing of a chunk is acceptable in multimedia communication.

Combining Hamming Distance and Interleaving

• Hamming distance and interleaving can be combined.

• Firstly, we can create n-bit packets that can correct t-bit errors.

• Then, we interleave m rows and send the bits column-by-column.

• In this way, we can automatically correct burst-errors up to m×t bit errors.

Compounding High- and Low-Resolution Packets

• Another solution is to create a duplicate of each packet with a low-resolution redundancy and combine the redundant version with the next packet.

• For example (Figure 10.22):

We can create 4 low-resolution packets out of 5 high-resolution packets and send them (Fig 10.22).

• If a packet is lost, we can use the low-resolution version from the next packet.

• In this method, if the last packet is lost, it cannot be recovered, but we use the low-resolution version of a packet if the lost packet is not the last one.

• The audio and video reproduction does not have the same quality, but the lack of quality is not recognized most of the time.

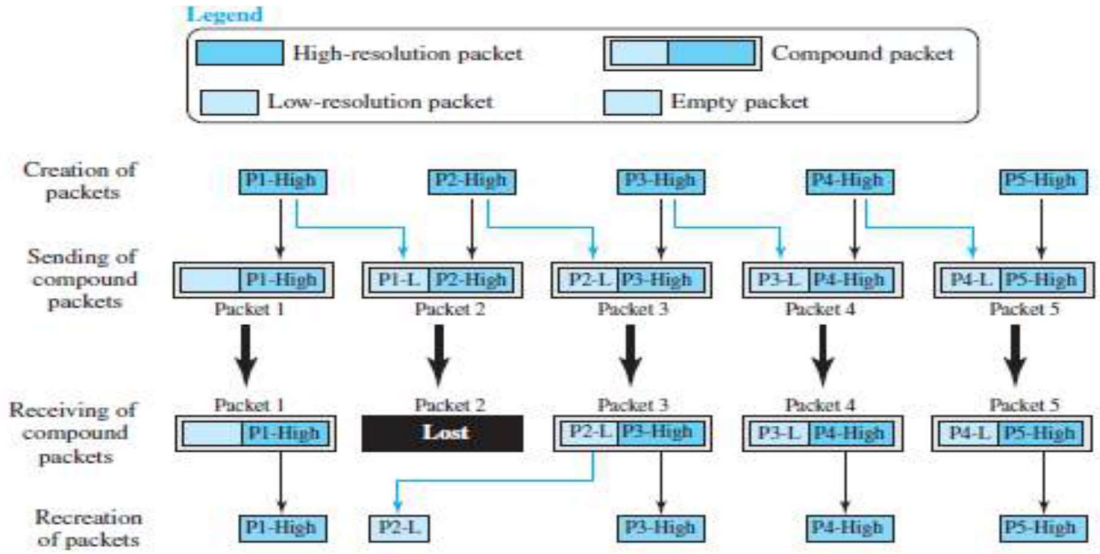


Figure 10.22 *Compounding high- and low-resolution packets*

Data Link Control (DLC)

The **data link control (DLC)** deals with procedures for communication between two adjacent nodes.Data link control functions include framing and flow and error control.

DLC SERVICES

Framing

The data-link layer, needs to pack bits into frames, so that each frame is distinguishable from another. **Framing** in the data-link layer separates a message from one source to a destination by adding a sender address and a destination address. The destination address defines where the packet is to go and the sender address helps the recipient acknowledge the receipt.

A message is divided into smaller frames, Since packing the entire message into one very large frame makes flow and error control very inefficient.

Frame Size

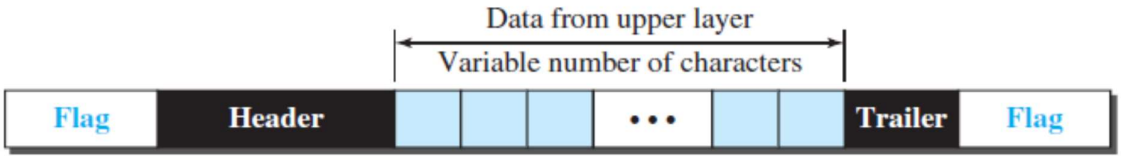
Frames can be of fixed or variable size. In fixed-size framing, there is no need for defining the boundaries of the frames; the size itself can be used as a delimiter. An example of this type of framing is the ATM WAN, which uses frames of fixed size called cells.

variable-size framing needs a way to define the end of one frame and the beginning of the next. Two approaches used for this purpose are: A character-oriented approach and A bit-oriented approach.

Character-Oriented Framing

In character-oriented (or byte-oriented) framing, data to be carried are 8-bit characters from a coding system such as ASCII . The header, carries the source and destination addresses and other control information, and the trailer, carries error detection 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 11.1 shows the format of a frame in a character-oriented protocol.

Figure 11.1 *A frame in a character-oriented protocol*

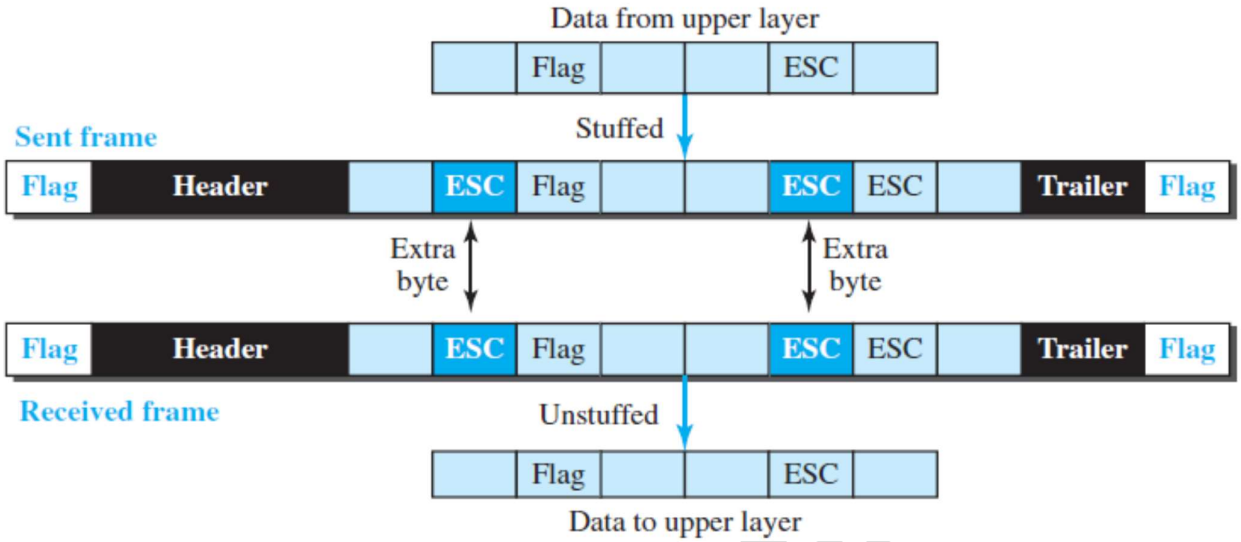


If data also contains the flag pattern then the receiver when it it encounters the flag pattern in the middle of data would conclude that it has reached the end of the frame. To fix this issue Byte stuffing strategy is added to character oriented framing.

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)* and 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 as a delimiting flag. Figure 11.2 shows the situation. If the text contains one or more escape characters followed by a byte with the same pattern as the flag, Then the escape characters that are part of the text must also be marked by another escape character.

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

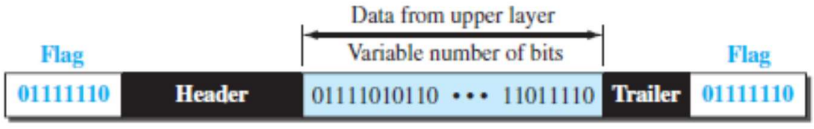
Figure 11.2 *Byte stuffing and unstuffing*



Bit-Oriented Framing

- The data-section of a frame is a sequence of bits to be interpreted by the upper layer as text, audio, video, and so on.
- However, in addition to headers and trailers, we 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 (Figure 11.3).

Figure 11.3 *A frame in a bit-oriented protocol*



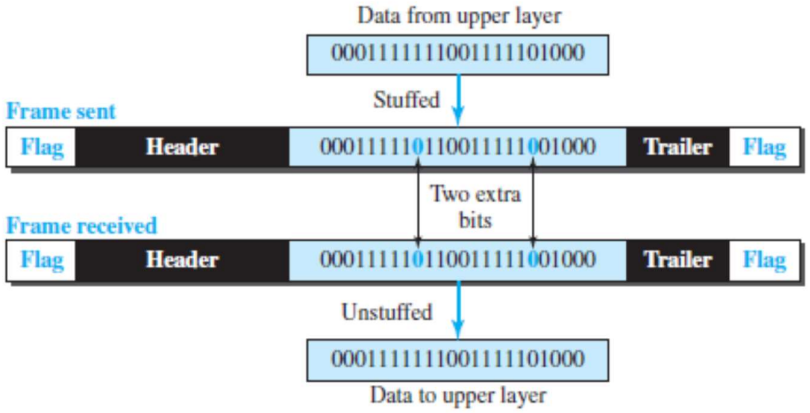
Problem:

If the flag-pattern appears in the data-section, the receiver might think that it has reached the end of the frame.

Solution: A bit-stuffing is used.

- **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. (Figure 11.4). This guarantees that the flag field sequence does not inadvertently appear in the frame.
- In short, 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.

Figure 11.4 *Bit stuffing and unstuffing*



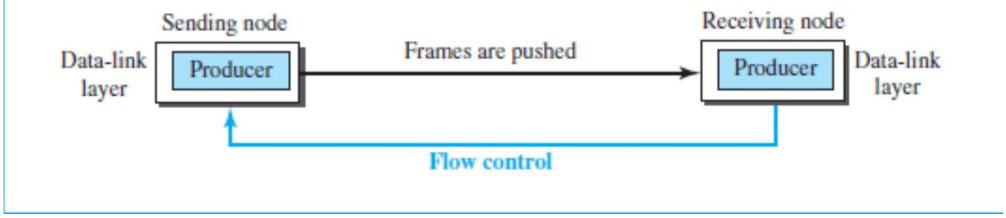
Flow Control and Error Control

One of the responsibilities of the DLC sublayer is flow and error control at the data-link layer.

Flow Control

- Whenever an entity produces items and another entity consumes them, there should be a balance between production and consumption rates.
- If the items are produced faster than they can be consumed, the consumer can be overwhelmed and may need to discard some items.We need to prevent losing the data items at the consumer site.
- At the sending node, the data-link layer tries to push frames toward the data-link layer at the receiving node (Figure 11.5).
- If the receiving node cannot process and deliver the packet to its network at the same rate that the frames arrive, it becomes overwhelmed with frames.
- Here, flow control can be feedback from the receiving node to the sending node to stop or slow down pushing frames.

Figure 11.5 *Flow control at the data-link layer*



Buffers

- Flow control can be implemented by using buffer.
- A buffer is a set of memory locations that can hold packets at the sender and receiver.
- Normally, two buffers can be used.
- First buffer at the sender.
- Second buffer at the receiver.
- The flow control communication can occur by sending signals from the consumer to the producer.
- When the buffer of the receiver is full, it informs the sender to stop pushing frames.

Error Control

- Error-control includes both error-detection and error-correction.
- Error-control allows the receiver to inform the sender of any frames lost/damaged in transmission.
- A CRC is
  → added to the frame header by the sender and
  → checked by the receiver.
- At the data-link layer, error control is normally implemented using one of the following two methods.
- First method: If the frame is corrupted, it is discarded.If the frame is not corrupted, the packet is delivered to the network layer.This method is used mostly in wired LANs such as Ethernet.
- Second method: If the frame is corrupted, it is discarded. If the frame is not corrupted, an acknowledgment is sent to the sender.Acknowledgment is used for the purpose of both flow and error control.

Combination of Flow and Error Control

- Flow and error control can be combined.
- The acknowledgment that is sent for flow control can also be used for error control to tell the sender the packet has arrived uncorrupted.
- The lack of acknowledgment means that there is a problem in the sent frame.
- A frame that carries an acknowledgment is normally called an ACK to distinguish it from the data frame.

Connectionless and Connection-Oriented

A DLC protocol can be either connectionless or connection-oriented.

Connectionless Protocol

- Frames are sent from one node to the next without any relationship between the frames.each frame is independent.
- The term connectionless does not mean that there is no physical connection (transmission medium) between the nodes; it means that there is no connection between frames.
- The frames are not numbered and there is no sense of ordering.
- Most of the data-link protocols for LANs are connectionless protocols.

Connection-Oriented Protocol

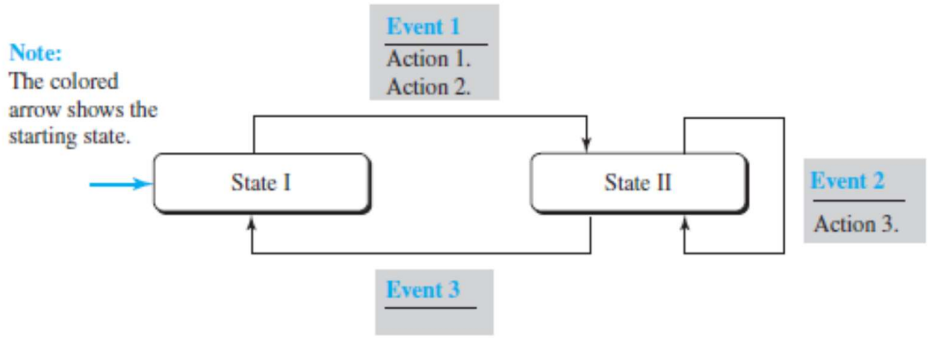
- A logical connection should first be established between the two nodes (setup phase).
- After all frames that are somehow related to each other are transmitted (transfer phase), the logical connection is terminated (teardown phase).
- The frames are numbered and sent in order. If the frames are not received in order, the receiver needs to wait until all frames belonging to the same set are received and then deliver them in order to the network layer.
- Connection oriented protocols are rare in wired LANs, but we can see them in some point-topoint protocols, some wireless LANs, and some WANs.

DATA LINK LAYER PROTOCOLS

- Traditionally 2 protocols have been defined for the data-link layer to deal with flow and error control:
- 1)Simple Protocol and 2) Stop-and-Wait Protocol.

- The behavior of a data-link-layer protocol can be better shown as a finite state machine (FSM). An FSM is a machine with a finite number of states (Figure 11.6).
- The machine is always in one of the states until an event occurs.
- Each event is associated with 2 reactions:
  Defining the list (possibly empty) of actions to be performed.
  Determining the next state (which can be the same as the current state).
- One of the states must be defined as the initial state, the state in which the machine starts when it turns on.

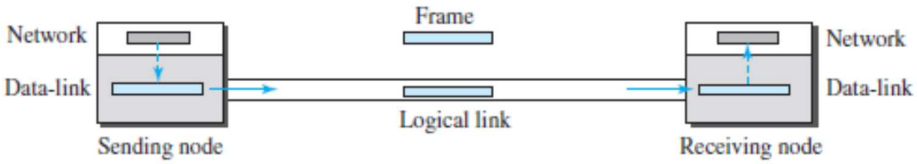
Figure 11.6 *Connectionless and connection-oriented service represented as FSMs*



Simple Protocol

- The protocol has no flow-control or error-control. The protocol is a unidirectional protocol, The receiver can immediately handle any frame it receives.

Figure 11.7 *Simple protocol*



1) At Sender the data-link-layer gets data from its network-layer, makes a frame out of the data and sends the frame.

2) At Receiver the data-link-layer receives a frame from its physical layer extracts data from the frame and delivers the data to its network-layer.

FSMs for simple protocol

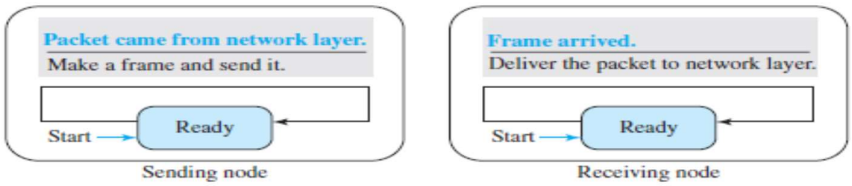
Two major requirements:

1) The sender-site cannot send a frame until its network-layer has a data packet to send.

2) The receiver-site cannot deliver a data packet to its network-layer until a frame arrives.

- These 2 requirements are shown using two FSMs. Each FSM has only one state, the ready state.

Figure 11.8 *FSMs for the simple protocol*



1) At Sending Machine

The sending machine remains in the ready state until a request comes from the process in the network layer. When this event occurs, the sending machine encapsulates the message in a frame and sends it to the receiving machine.

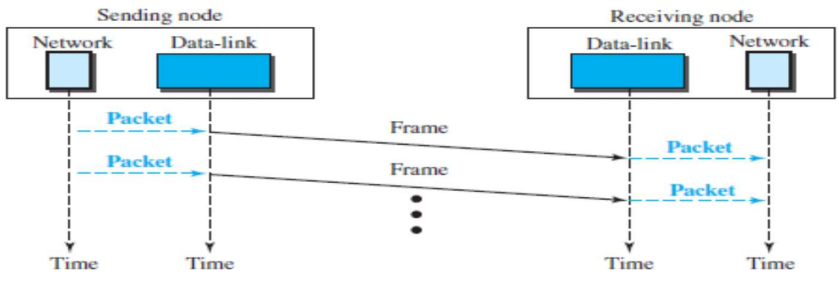
2) At Receiving Machine

The receiving machine remains in the ready state until a frame arrives from the sending machine. When this event occurs, the receiving machine decapsulates the message out of the frame and delivers it to the process at the network layer.

Example 11.2

Figure 11.9 shows an example of communication using this protocol. It is very simple. The sender sends frames one after another without even thinking about the receiver.

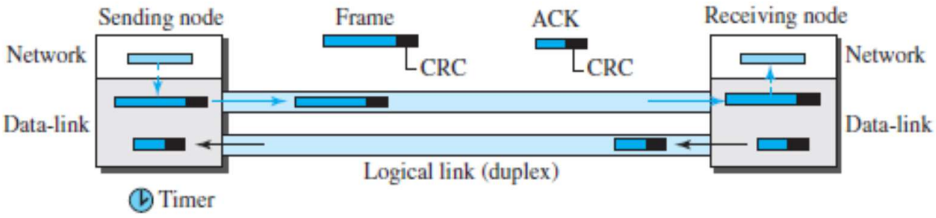
Figure 11.9 *Flow diagram for Example 11.2*



Stop & Wait Protocol

- This uses both flow and error control.
- Normally, the receiver has limited storage-space.
- If the receiver is receiving data from many sources, the receiver may
  → be overloaded with frames &
  → discard the frames.
- To prevent the receiver from being overloaded with frames, we need to tell the sender to slow down.

Figure 11.10 *Stop-and-Wait protocol*



1) At Sender

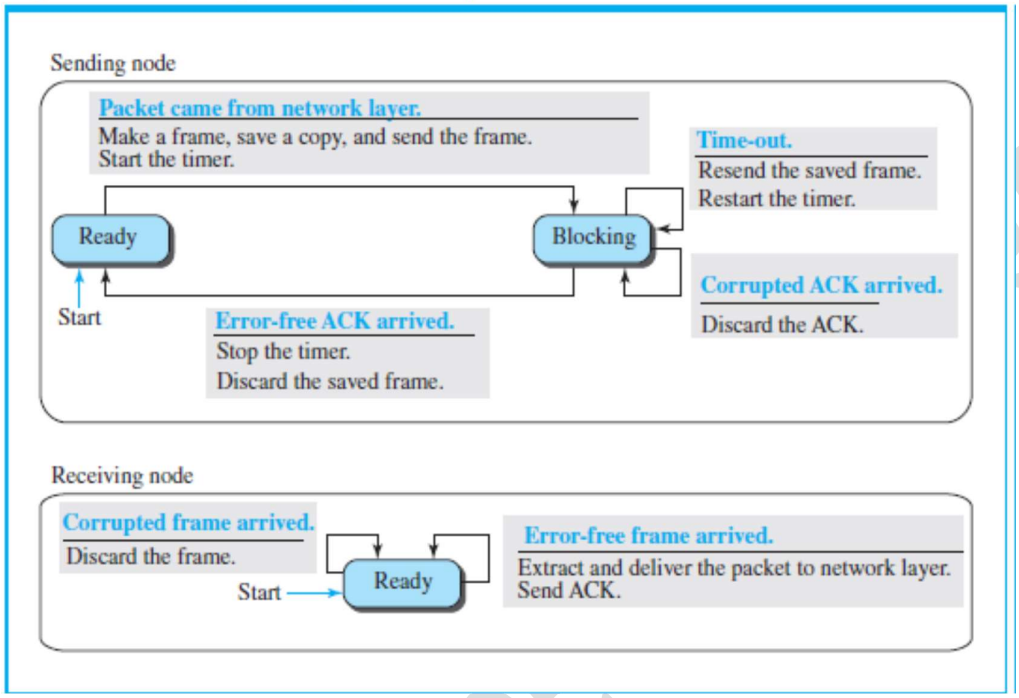
- The sender sends one frame & starts a timer, keeps a copy of the sent-frame and waits for ACK-frame from the receiver (okay to go ahead).
- Then, If an ACK-frame arrives before the timer expires, the timer is stopped and the sender sends the next frame. Also, the sender discards the copy of the previous frame.
- If the timer expires before ACK-frame arrives, the sender resends the previous frame and restarts the timer

2) At Receiver

- To detect corrupted frames, a CRC is added to each data frame. When a frame arrives at the receiver-site, the frame is checked. If frame's CRC is incorrect, the frame is corrupted and discarded. The silence of the receiver is a signal for the sender that a frame was either corrupted or lost.

FSMs

Figure 11.11 *FSM for the Stop-and-Wait protocol*



Sender States

Sender is initially in the ready state, but it can move between the ready and blocking state.

- **Ready State:** When the sender is in this state, it is only waiting for a packet from the network layer. If a packet comes from the network layer, the sender creates a frame, saves a copy of the frame, starts the only timer and sends the frame. The sender then moves to the blocking state.
- **Blocking State:** When the sender is in this state, three events can occur:
  - If a time-out occurs, the sender resends the saved copy of the frame and restarts the timer.
  - If a corrupted ACK arrives, it is discarded.
  - If an error-free ACK arrives, the sender stops the timer and discards the saved copy of the frame. It then moves to the ready state.

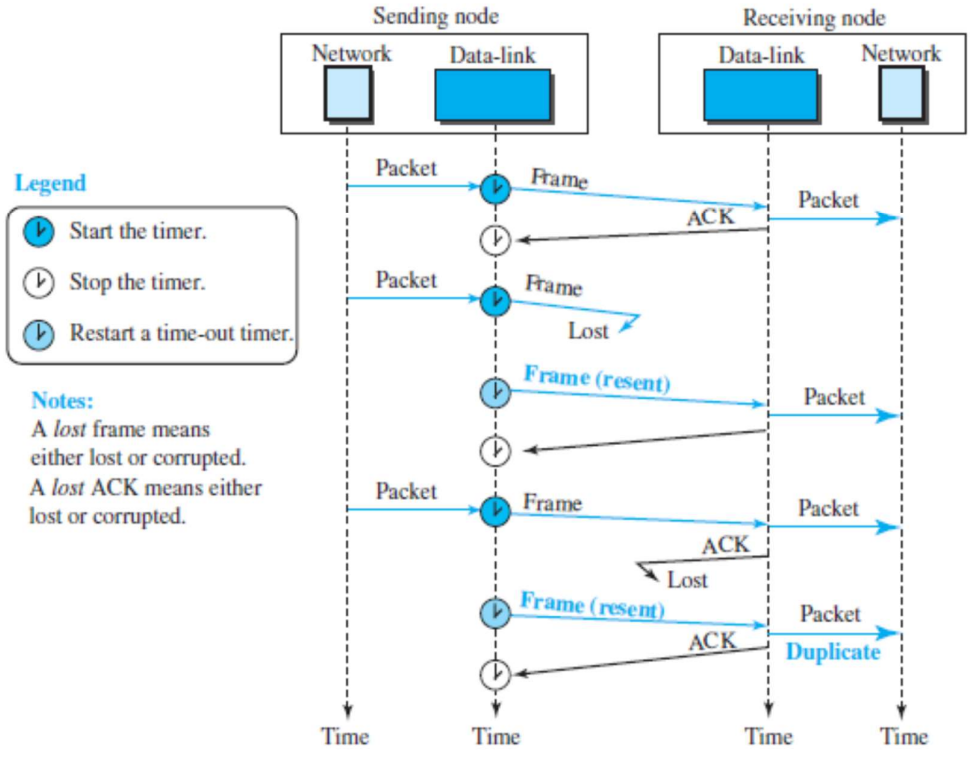
Receiver Satates

- The receiver is always in the ready state. Two events may occur:
  - If an error-free frame arrives, the message in the frame is delivered to the network layer and an ACK is sent.
  - If a corrupted frame arrives, the frame is discarded.

Example 11.3

Figure 11.12 shows an example. The first frame is sent and acknowledged. The second frame is sent, but lost. After time-out, it is resent. The third frame is sent and acknowledged, but the acknowledgment is lost. The frame is resent. However, there is a problem with this scheme. The network layer at the receiver site receives two copies of the third packet, which is not right. In the next section, we will see how we can correct this problem using sequence numbers and acknowledgment numbers.

Figure 11.12 *Flow diagram for Example 11.3*



Sequence and Acknowledgment Numbers

- If the corrupted-frame arrives at the receiver-site, then the frame is simply discarded.
- If the receiver receives out-of-order data-frame, then it means that frames were lost. The lost-frames need to be resent.

Problem in Stop and Wait protocols:

1) There is no way to identify a frame.

2) The received-frame could be the correct one, or a duplicate, or a frame out of order.

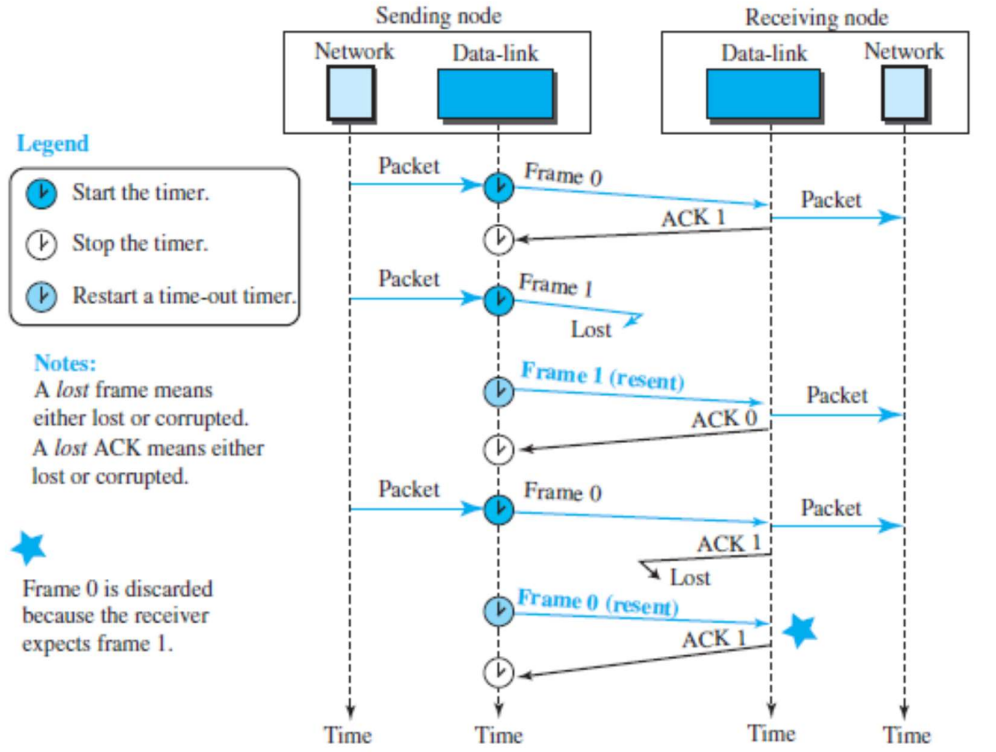
Solution: 1) Use sequence-number for each data frame. 2) Use Acknowledgment-number for each ACK frame.

- **Sequence Numbers-** Frames need to be numbered. This is done by using sequence-numbers. A sequence-number field is added to the data-frame.
- **Acknowledgment Numbers -**An acknowledgment-number field is added to the ACK-frame.
- Sequence numbers are 0, 1, 0, 1, 0, 1, . . .
- The acknowledgment numbers can also be 1, 0, 1, 0, 1, 0, …
- The acknowledgment-numbers always announce the sequence-number of the next frame expected by the receiver.
- For example, If frame-0 has arrived safely, the receiver sends an ACK-frame with acknowledgment-1 (meaning frame-1 is expected next).

Example 11.4

Figure 11.13 shows how adding sequence numbers and acknowledgment numbers can prevent duplicates. The first frame is sent and acknowledged. The second frame is sent, but lost. After time-out, it is resent. The third frame is sent and acknowledged, but the acknowledgment is lost. The frame is resent.

Figure 11.13 *Flow diagram for Example 11.4*



Piggybacking

- A technique called piggybacking is used to improve the efficiency of the bidirectional protocols.
- The data in one direction is piggybacked with the acknowledgment in the other direction.
- In other words, when node A is sending data to node B, Node A also acknowledges the data received from node B.

High-Level Data Link Control (HDLC)

- HDLC is a bit-oriented protocol for communication over point-to-point and multipoint links.
- HDLC implements the ARQ mechanisms.

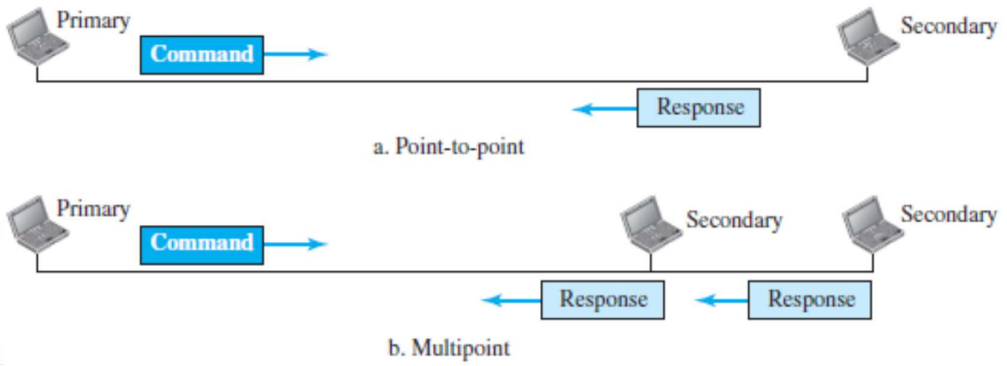
Configurations and Transfer Modes

- HDLC provides 2 common transfer modes that can be used in different configurations:
    - Normal response mode (NRM)
    - Asynchronous balanced mode (ABM).

NRM(Normal Response Mode)

- The station configuration is unbalanced (Figure 11.14).
- We have one primary station and multiple secondary stations.
- A primary station can send commands, a secondary station can only respond.
- The NRM is used for both point-to-point and multiple-point links.

Figure 11.14 *Normal response mode*



ABM(Asychronous Mode)

- The configuration is balanced (Figure 11.15).
- Link is point-to-point, and each station can function as a primary and a secondary (acting as peers).
- This is the common mode today.

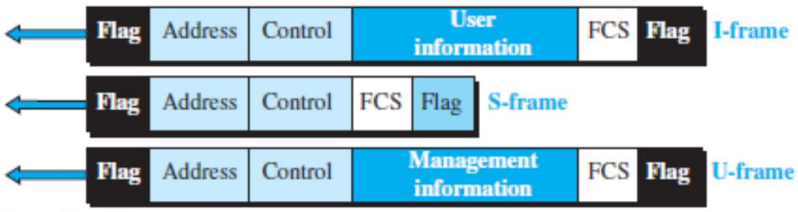
Figure 11.15 *Asynchronous balanced mode*

Framing

- To provide the flexibility necessary to support all the options possible in the modes and configurations, HDLC defines three types of frames:
    1) Information frames (I-frames): are used to transport user data and control information relating to user data (piggybacking).
    2) Supervisory frames (S-frames): are used only to transport control information.
    3) Unnumbered frames (U-frames): are reserved for system management.
- Information carried by U-frames is intended for managing the link itself.
- Each type of frame serves as an envelope for the transmission of a different type of message.

Frame Format

Figure 11.16 *HDLC frames*



1) Flag Field

This field has a synchronization pattern 01111110. This field identifies both the beginning and the end of a frame.

2) Address Field

This field contains the address of the secondary station.If a primary station created the frame, it contains a to-address. If a secondary creates the frame, it contains a from-address. This field can be 1 byte or several bytes long, depending on the needs of the network.

3) Control Field This field is one or two bytes used for flow and error control.

4) Information Field

This field contains the user's data from the network-layer or management information. Its length can vary from one network to another.

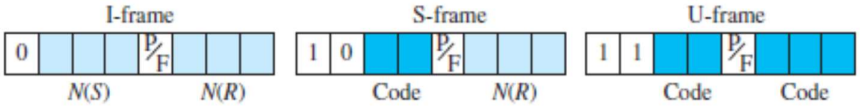
5) FCS Field

This field is the error-detection field. (FCS-Frame Check Sequence) .This field can contain either a 2- or 4-byte standard CRC.

Control Fields of HDLC Frames

The control field determines the type of frame and defines its functionality (Figure 11.17).

Figure 11.17 *Control field format for the different frame types*



Control Field for I-Frames

- I-frames are designed to carry user data from the network-layer. In addition, they can include flow and error-control information (piggybacking).
- The subfields in the control field are:
  - The first bit defines the type. If the first bit of the control field is 0, this means the frame is an I-frame.
  - The next 3 bits N(S) define the sequence-number of the frame. With 3 bits, we can define a sequence-number between 0 and 7
  - The last 3 bits N(R) correspond to the acknowledgment-number when piggybacking is used.
  - The single bit between N(S) and N(R) is called the P/F bit.
  - The P/F field is a single bit with a dual purpose. It can mean poll or final. It means poll when the frame is sent by a primary station to a secondary and It means final when the frame is sent by a secondary to a primary.

Control Field for S-Frames

- Supervisory frames are used for flow and error-control whenever piggybacking is either impossible or inappropriate (e.g., when the station either has no data of its own to send or needs to send a command or response other than an acknowledgment).S-frames do not have information fields.
- The subfields in the control field are:
  - If the first 2 bits of the control field is 10, this means the frame is an S-frame.
  - The last 3 bits N(R) corresponds to the acknowledgment-number (ACK) or negative acknowledgment-number (NAK).
  - The 2 bits called code is used to define the type of S-frame itself. With 2 bits, we can have four types of S-frames:
    - **Receive ready (RR) = 00**

      This acknowledges the receipt of frame or group of frames. The value of N(R) is the acknowledgment-number.
    - **Receive not ready (RNR) = 10**

      This is an RR frame it announces that the receiver is busy and cannot receive more frames. It acts as congestion control mechanism by asking the sender to slow down. The value of N(R) is the acknowledgment-number.
    - **Reject (REJ) = 01**

      It is a NAK frame used in Go-Back-N ARQ to improve the efficiency of the process. It informs the sender, before the sender time expires, that the last frame is lost or damaged. The value of N(R) is the negative acknowledgment-number.
    - **Selective reject (SREJ) = 11**

      This is a NAK frame used in Selective Repeat ARQ. The value of N(R) is the negative acknowledgment-number.

Control Field for U-Frames

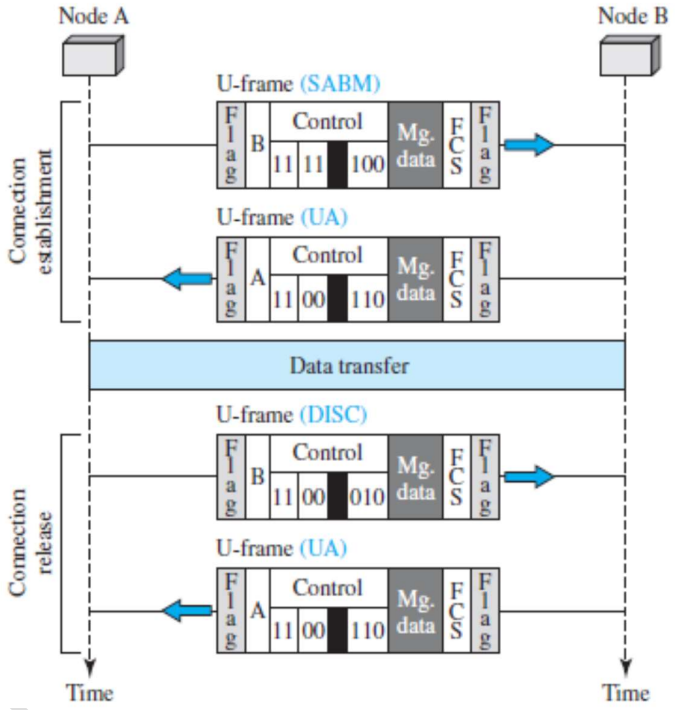
- Unnumbered frames are used to exchange session management and control information between connected devices. U-frames contain an information field used for system management information, but not user data. Much of the information carried by U-frames is contained in codes included in the control field.

- U-frame codes are divided into 2 sections: i) A 2-bit prefix before the P/F bit ii) A 3-bit suffix after the P/F bit. Together, these two segments (5 bits) can be used to create up to 32 different types of U-frames.

Example 11.5

Figure 11.18 shows how U-frames can be used for connection establishment and connection release. Node A asks for a connection with a set asynchronous balanced mode (SABM) frame; node B gives a positive response with an unnumbered acknowledgment (UA) frame. After these two exchanges, data can be transferred between the two nodes (not shown in the figure). After data transfer, node A sends a DISC (disconnect) frame to release the connection; it is confirmed by node B responding with a UA (unnumbered acknowledgment).

Figure 11.18 *Example of connection and disconnection*



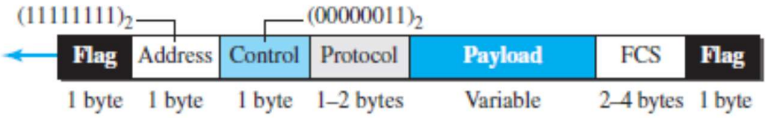
POINT-TO-POINT PROTOCOL (PPP)

PPP is one of the most common protocols for point-to-point access. Today, millions of Internet users who connect their home computers to the server of an ISP use PPP.

Framing

• PPP uses a character-oriented (or byte-oriented) frame (Figure 11.20).

Figure 11.20 *PPP frame format*



Various fields of PPP frame are:

1) Flag

This field has a synchronization pattern 01111110. This field identifies both the beginning and the end of a frame.

2) Address

This field is set to the constant value 11111111 (broadcast address).

3) Control

This field is set to the constant value 00000011 (imitating unnumbered frames in HDLC). PPP does not provide any flow control. Error control is also limited to error detection.

4) Protocol

This field defines what is being carried in the payload field. Payload field carries either i) user data or ii) other control information. By default, size of this field = 2 bytes.

5) Payload field

This field carries either i) user data or ii) other control information. By default, maximum size of this field is 1500 bytes. This field is byte-stuffed if the flag-byte pattern appears in this field. Padding is needed if the payload-size is less than the maximum size.

6) FCS

This field is the PPP error-detection field. This field can contain either a 2- or 4-byte standard CRC.

Byte Stuffing

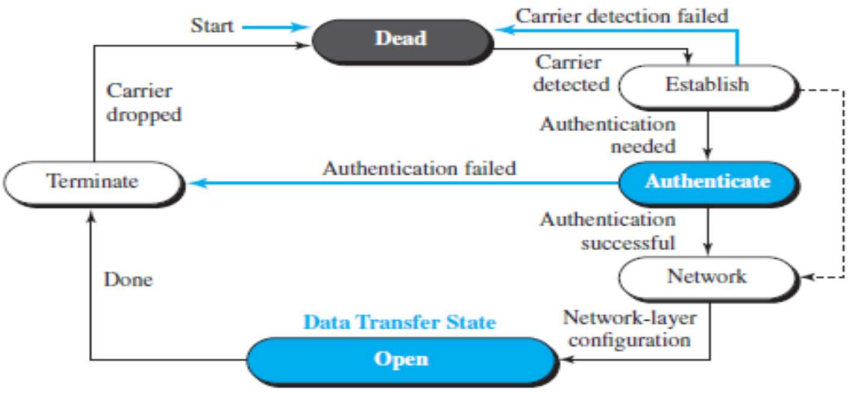
- Since PPP is a byte-oriented protocol, the flag in PPP is a byte that needs to be escaped whenever it appears in the data section of the frame.

- The escape byte is 01111101, which means that every time the flag like pattern appears in the data, this extra byte is stuffed to tell the receiver that the next byte is not a flag. Obviously, the escape byte itself should be stuffed with another escape byte.

Transition Phases

The transition diagram starts with the dead state (Figure 11.21).

Figure 11.21 *Transition phases*



1) Dead State

In dead state, there is no active carrier and the line is quiet.

2) Establish State

When 1 of the 2 nodes starts communication, the connection goes into the establish state. In establish state, options are negotiated between the two parties.

3) Authenticate State

If the 2 parties agree that they need authentication, Then the system needs to do authentication. Otherwise, the parties can simply start communication.

4) Open State

Data transfer takes place in the open state.

5) Terminate State

When 1 of the endpoints wants to terminate connection, the system goes to terminate state.