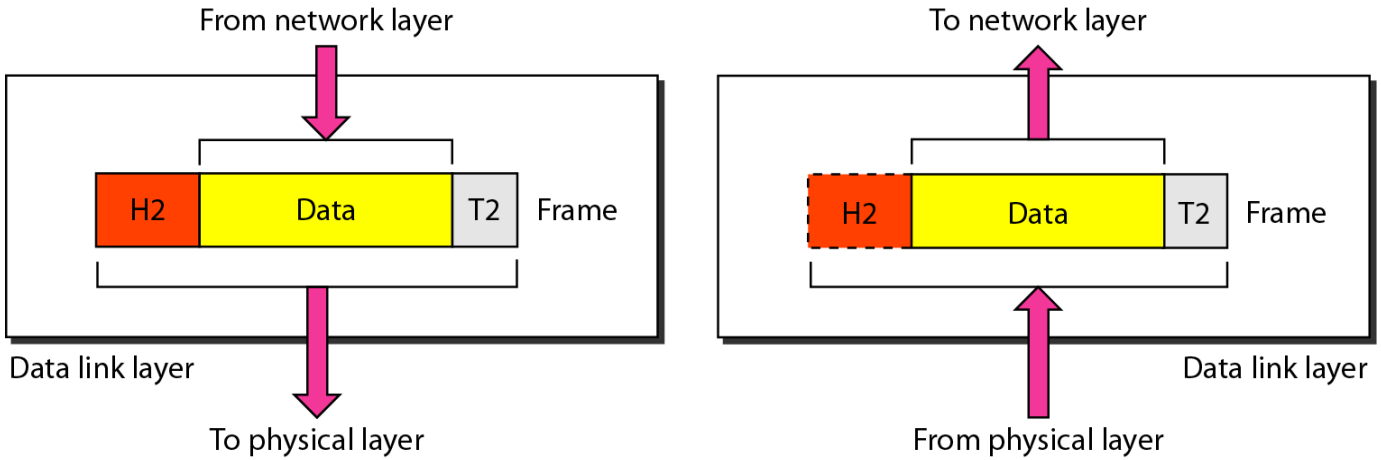
Data Communication

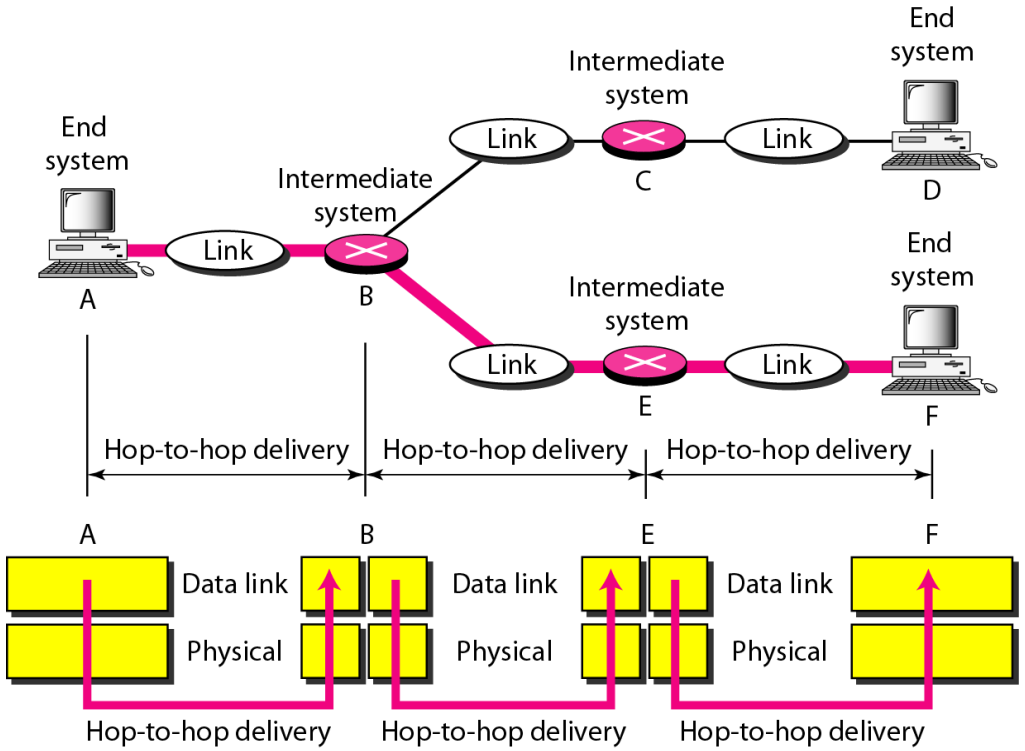
#6. Data Link Layer

Susmini I. Lestariningati, M.T

Introduction to Data Link Layer

- The data link layer transforms the physical layer, a raw transmission facility, to a link responsible for node-to-node (hop-to-hop) communication. Specific responsibilities of the data link layer include framing, addressing, flow control, error control, and media access control.
- The data link layer divides the stream of bits received from the network layer into manageable data units called frames. The data link layer adds a header to the frame to define the addresses of the sender and receiver of the frame.





- The data link layer also adds reliability to the physical layer by adding mechanisms to detect and retransmit damaged, duplicate, or lost frames. When two or more devices are connected to the same link, data link layer protocols are necessary to determine which device has control over the link at any given time.

Error Control

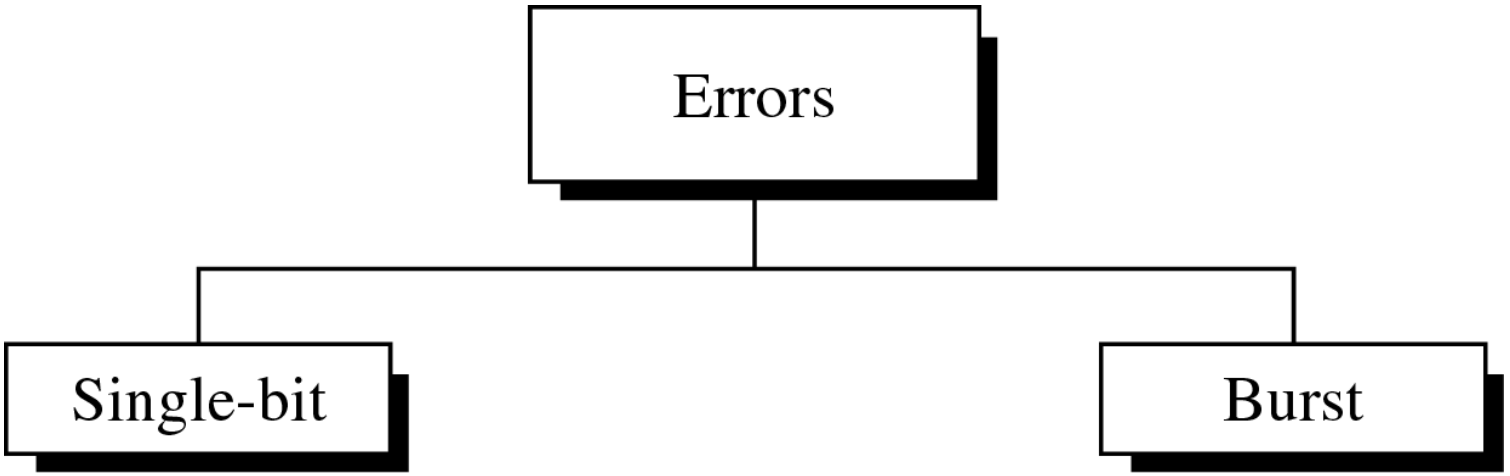
- Error detection and correction or error control are techniques that enable reliable delivery of digital data over unreliable communication channels.
- Many communication channels are subject to channel noise, and thus errors may be introduced during transmission from the source to a receiver.
- Error detection techniques allow detecting such errors, while error correction enables reconstruction of the original data.

Error Detection and Correction

- Data can be corrupted during transmission. For reliable communication, error must be detected and corrected
- are implemented either at the data link layer or the transport layer of the OSI model
- The general definitions of the terms are as follows:
  - **Error detection** is the detection of errors caused by noise or other impairments during transmission from the transmitter to the receiver.
  - **Error correction** is the detection of errors and reconstruction of the original, error-free data.

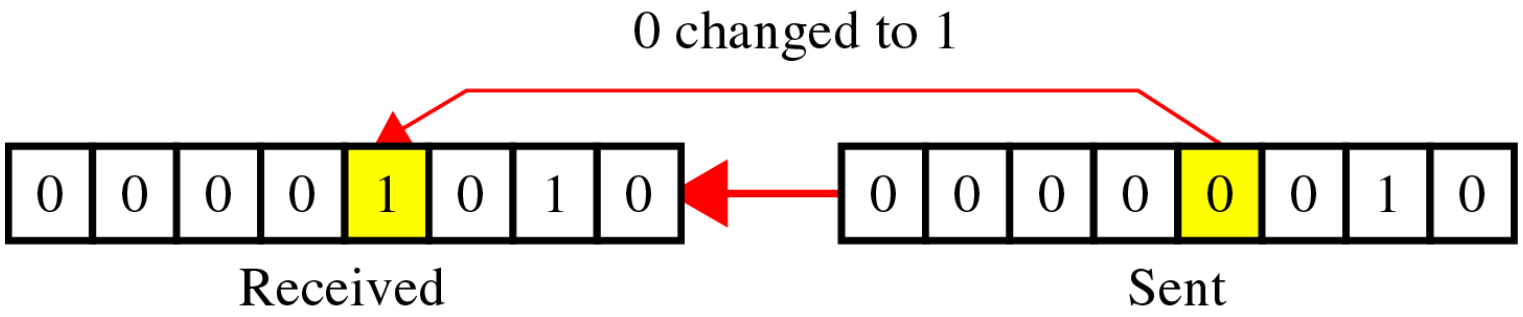
Types of Errors

- Single Bit Error
- Burst Error



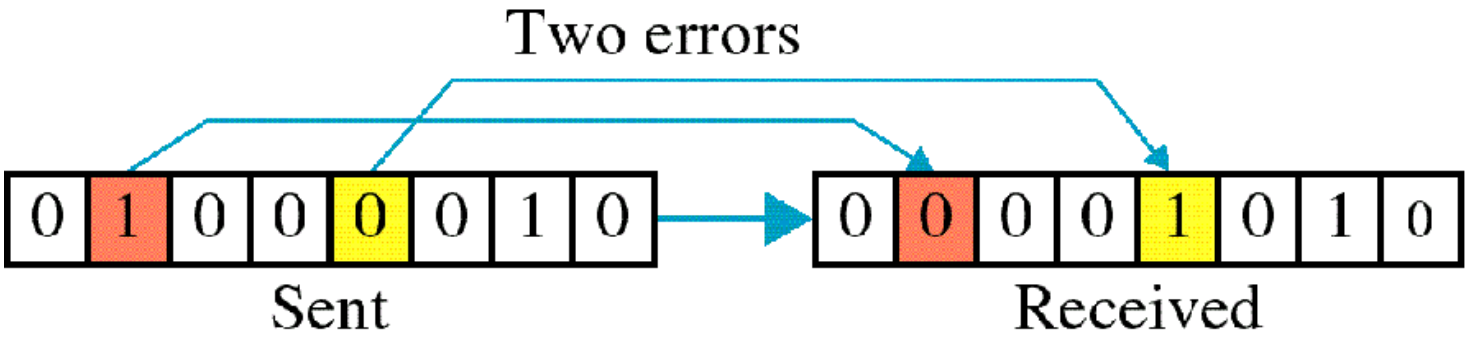
Single Bit Error

- Single-Bit Error
  - is when only one bit in the data unit has changed
  - ex : ASCII STX - ASCII LF



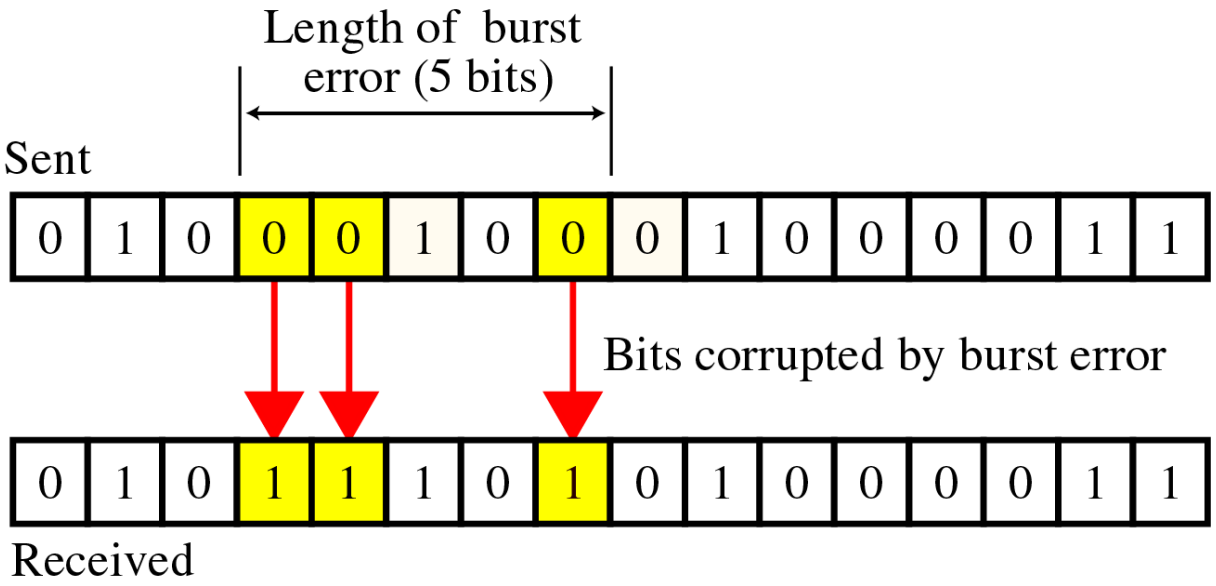
Multiple Bit Error

- Multiple-Bit Error
  - is when two or more nonconsecutive bits in the data unit have changed
  - ex : ASCII B - ASCII LF



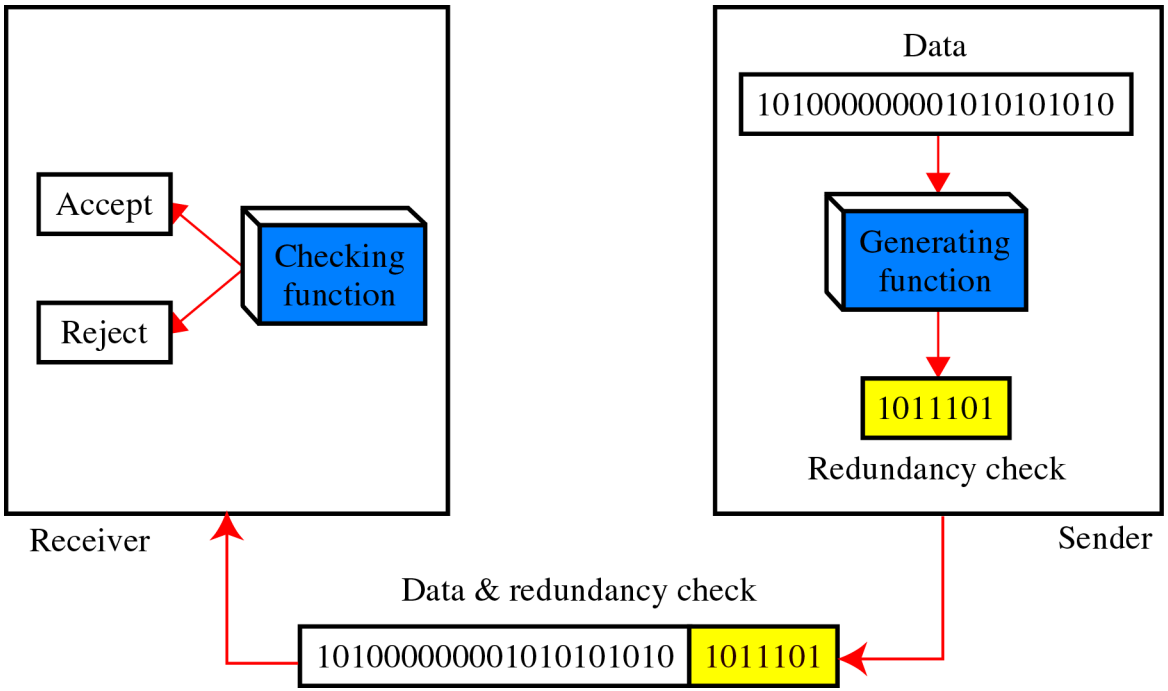
Burst Error

- Burst Error
  - means that two or more consecutive bits in the data unit have changed

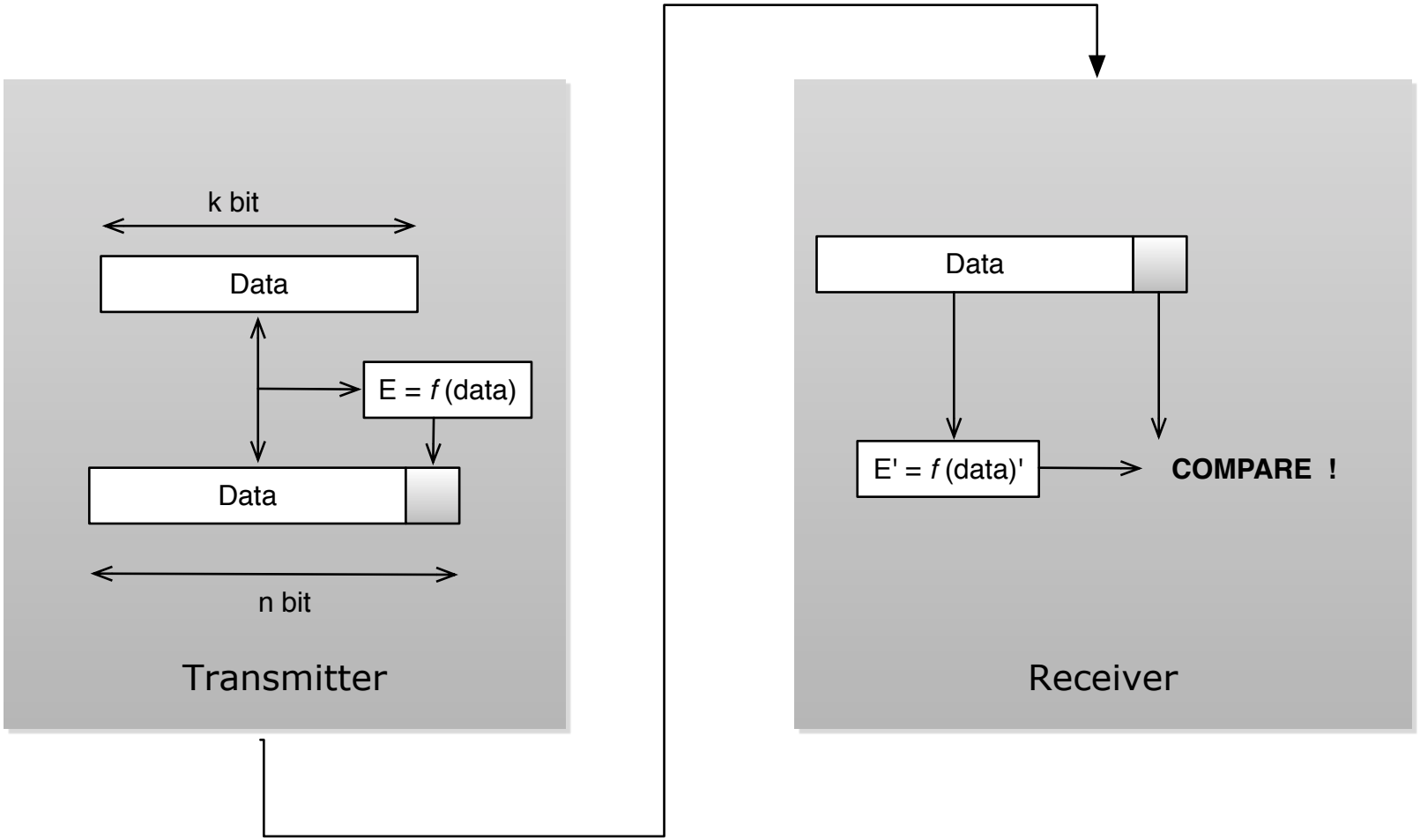


Redundancy Concepts

- The general idea for achieving error detection and correction is to add some redundancy (i.e., some extra data) to a message, which receivers can use to check consistency of the delivered message, and to recover data determined to be corrupted.

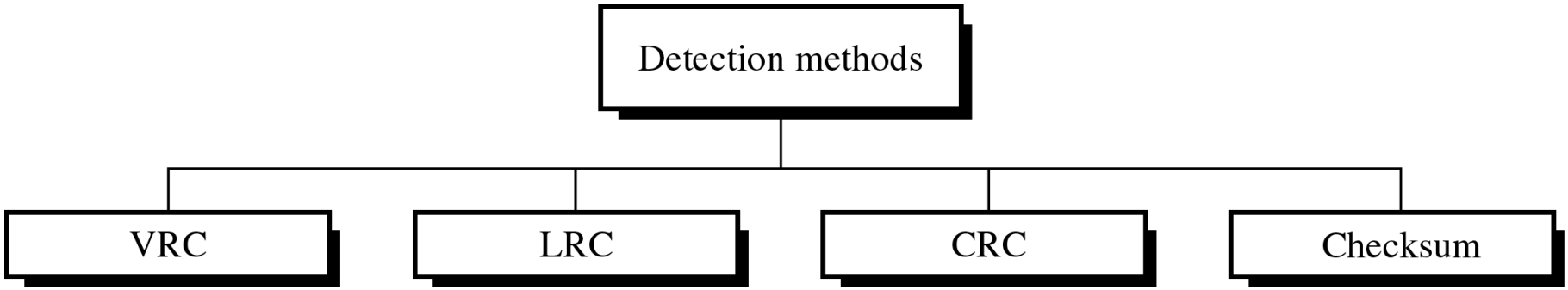


- Error-detection and correction schemes can be either systematic or non-systematic:
  - In a systematic scheme, the transmitter sends the original data, and attaches a fixed number of check bits (or parity data), which are derived from the data bits by some deterministic algorithm. If only error detection is required, a receiver can simply apply the same algorithm to the received data bits and compare its output with the received check bits; if the values do not match, an error has occurred at some point during the transmission.
  - In a system that uses a non-systematic code, the original message is transformed into an encoded message that has at least as many bits as the original message.


k bit
Data
E = f (data)
Data
n bit
Transmitter
Data
E' = f (data)'
COMPARE !
Receiver

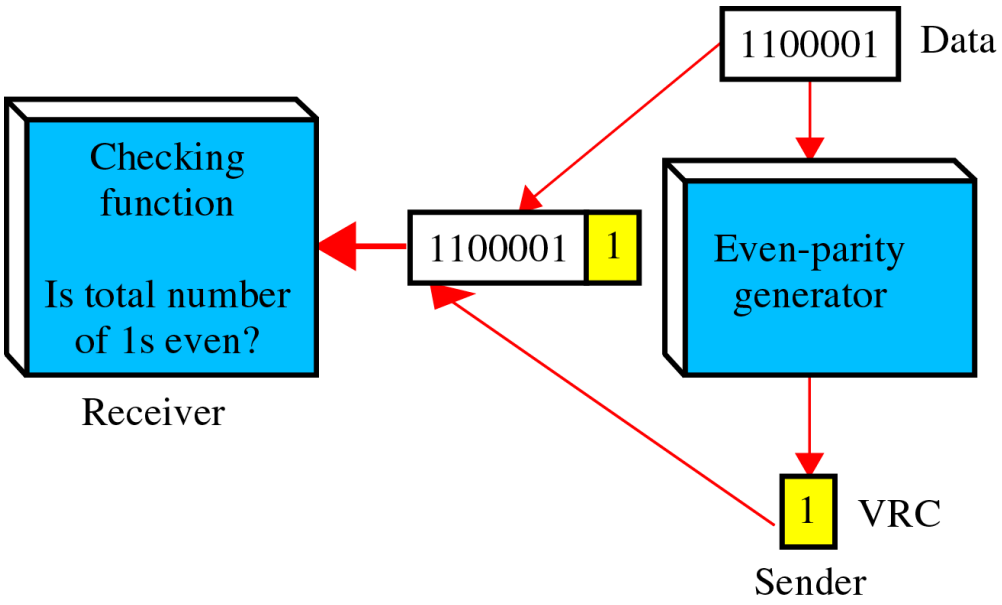
Error Detection Methods

- Detection methods
  - VRC (Vertical Redundancy Check)
  - LRC (Longitudinal Redundancy)
  - CRC (Cyclical redundancy Check)
  - Checksum



VRC (Vertical Redundancy Check)

- A parity bit is added to every data unit so that the total number of 1s(including the parity bit) becomes even for even-parity check or odd for odd-parity check
- VRC can detect all single-bit errors. It can detect multiple-bit or burst errors only the total number of errors is odd/even



- Example of Even Parity

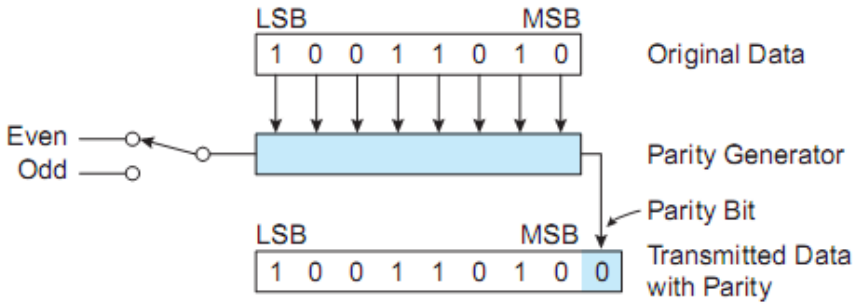
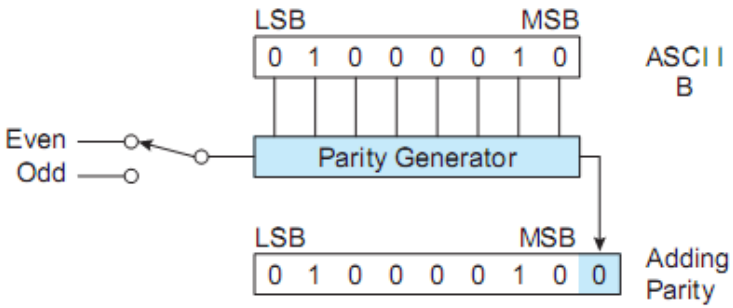
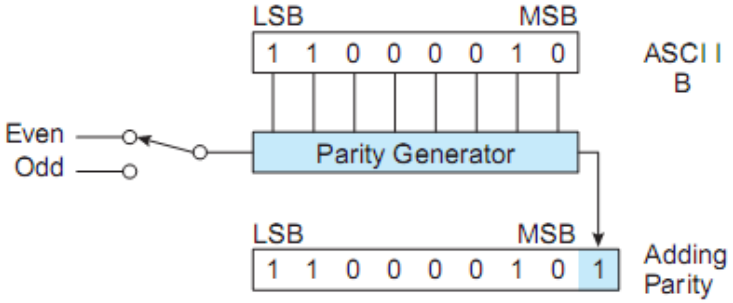


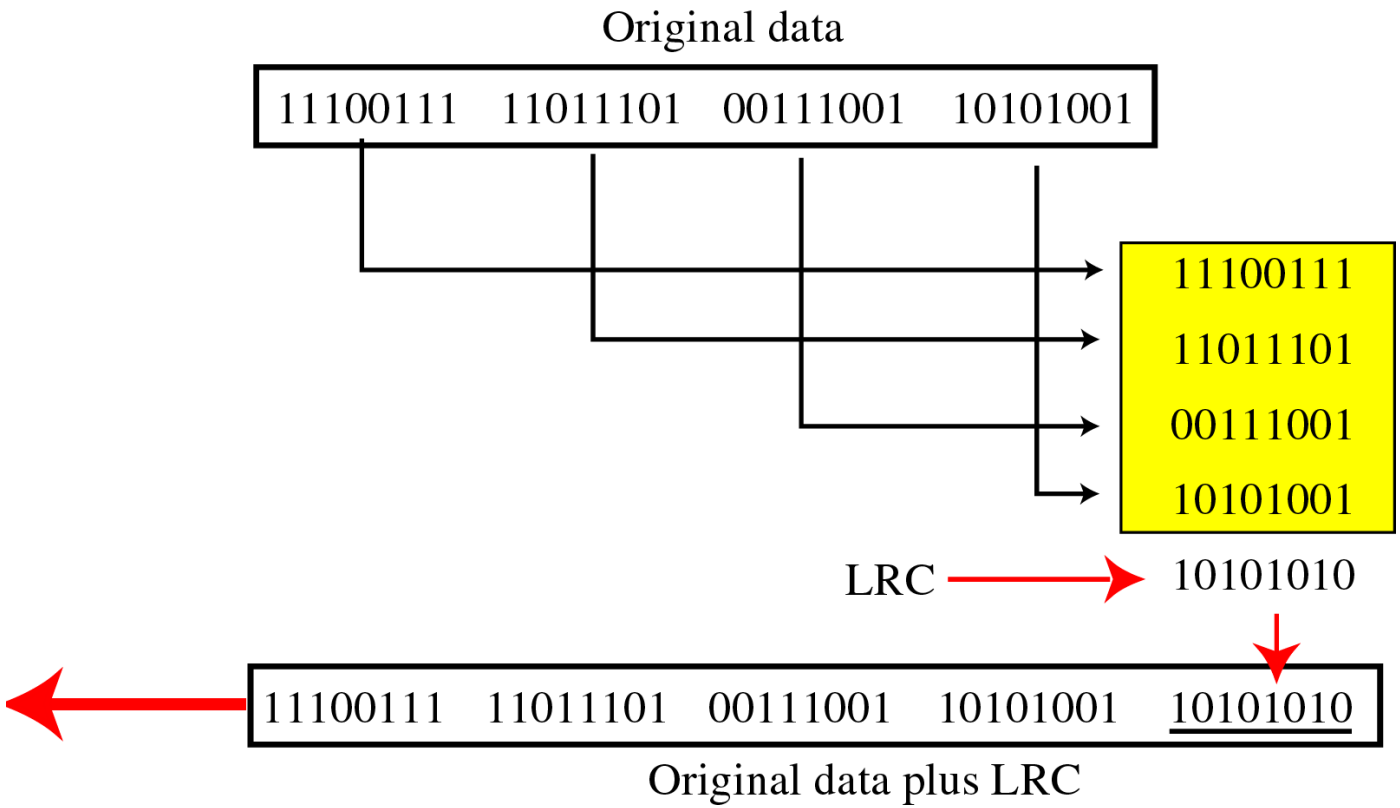
FIGURE 3-1 Appending Parity Bit





LRC (Longitudinal Redundancy Check)

- Parity bits of all the positions are assembled into a new data unit, which is added to the end of the data block



EXAMPLE 3-4

Determine the states of the LRC bits for the asynchronous ASCII message "Help!"

SOLUTION

The first step in understanding the process is to list each of the message's characters with their ASCII code and even VRC parity bit:

LSB						MSB	VRC	CHARACTER
0	0	0	1	0	0	1	0	H
1	0	1	0	0	1	1	0	e
0	0	1	1	0	1	1	0	l
0	0	0	0	1	1	1	1	p
1	0	0	0	0	1	0	0	!

Next, for each vertical column, find the LRC bit by applying the exclusive OR function. To make this process easier, you can consider the results of the exclusive OR process as being low or zero (0), if the number of ones (1) are even, and one (1) if the count is odd. For instance, in the LSB column, there are two 1's, so the LRC bit for that column is a 0. And for the rest:

LSB						MSB	VRC	CHARACTER
0	0	0	1	0	0	1	0	H
1	0	1	0	0	1	1	0	e
0	0	1	1	0	1	1	0	l
0	0	0	0	1	1	1	1	p
1	0	0	0	0	1	0	0	!
0	0	0	0	1	0	0	1	LRC

EXAMPLE 3-5

Show how a good message would produce an LRC of 0 at the receiver.

SOLUTION

Repeat the process as before, but include the LRC character this time. Note that the number of 1s in each column are always even if there are no errors present:

LSB						MSB	VRC	CHARACTER
0	0	0	1	0	0	1	0	H
1	0	1	0	0	1	1	0	e
0	0	1	1	0	1	1	0	l
0	0	0	0	1	1	1	1	p
1	0	0	0	0	1	0	0	!
0	0	0	0	1	0	0	1	LRC
0	0	0	0	0	0	0	0	Receiver LRC

EXAMPLE 3-6

Illustrate how LRC/VRC is used to correct a bad bit.

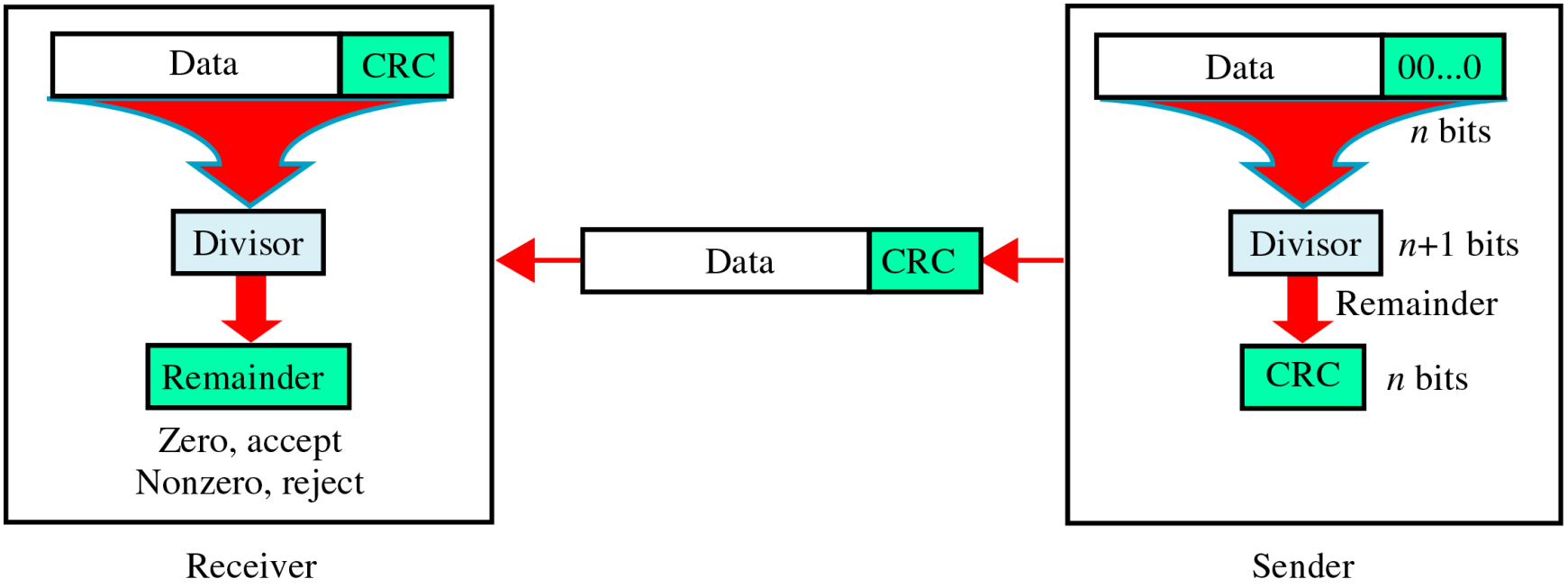
SOLUTION

We will use the same message, but by placing an error in the received data would cause the l character to print as an *h*. You can compare the data with the good example to satisfy yourself as to which bit is bad and confirm that the LRC process does indeed pick out the same bit.

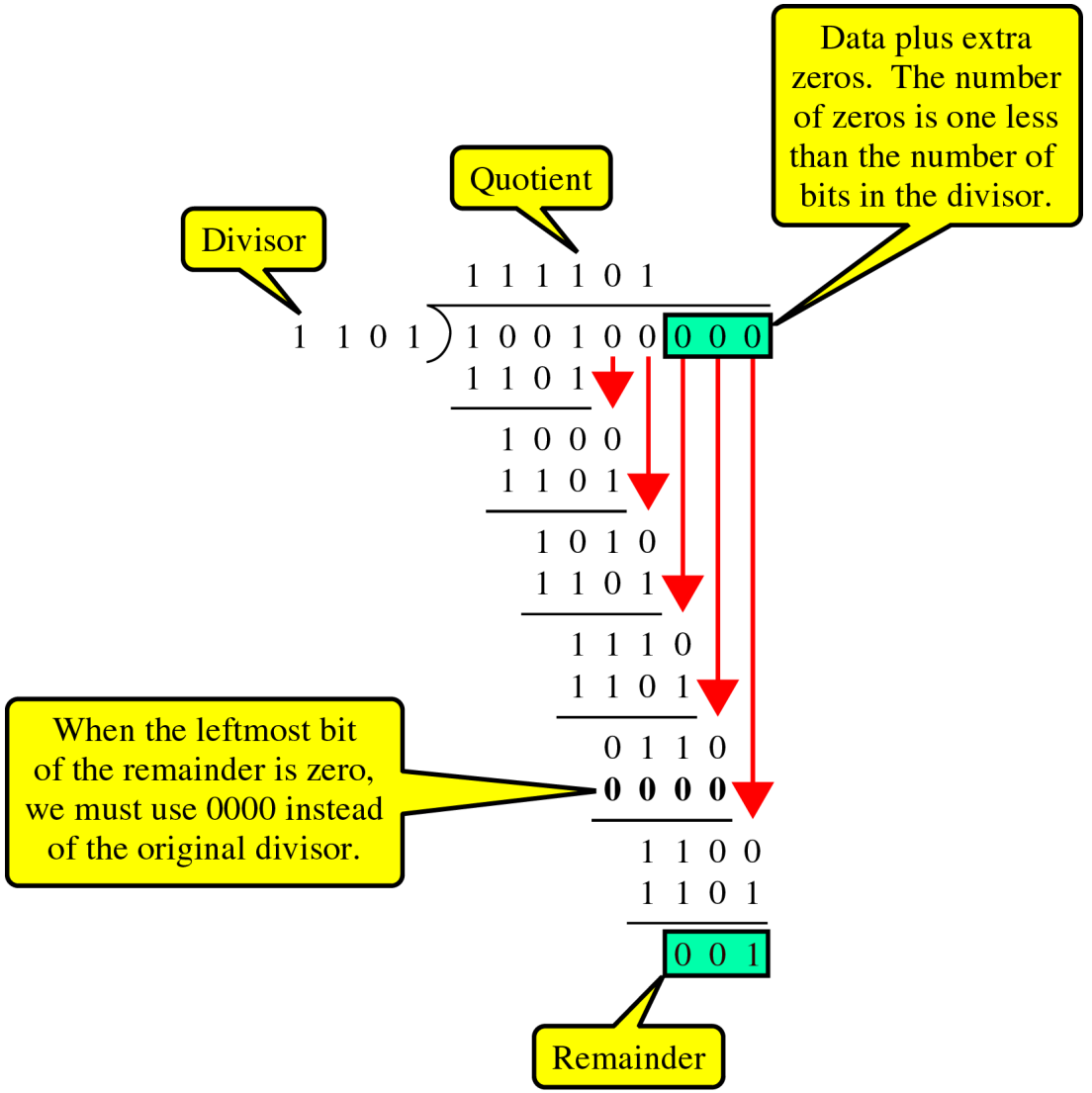
LSB						MSB	VRC	CHARACTER
0	0	0	1	0	0	1	0	H
1	0	1	0	0	1	1	0	e
0	0	0	1	0	1	1	0	h
0	0	0	0	1	1	1	1	p
1	0	0	0	0	1	0	0	!
0	0	0	0	1	0	0	1	LRC
0	0	1	0	0	0	0	1	Received LRC

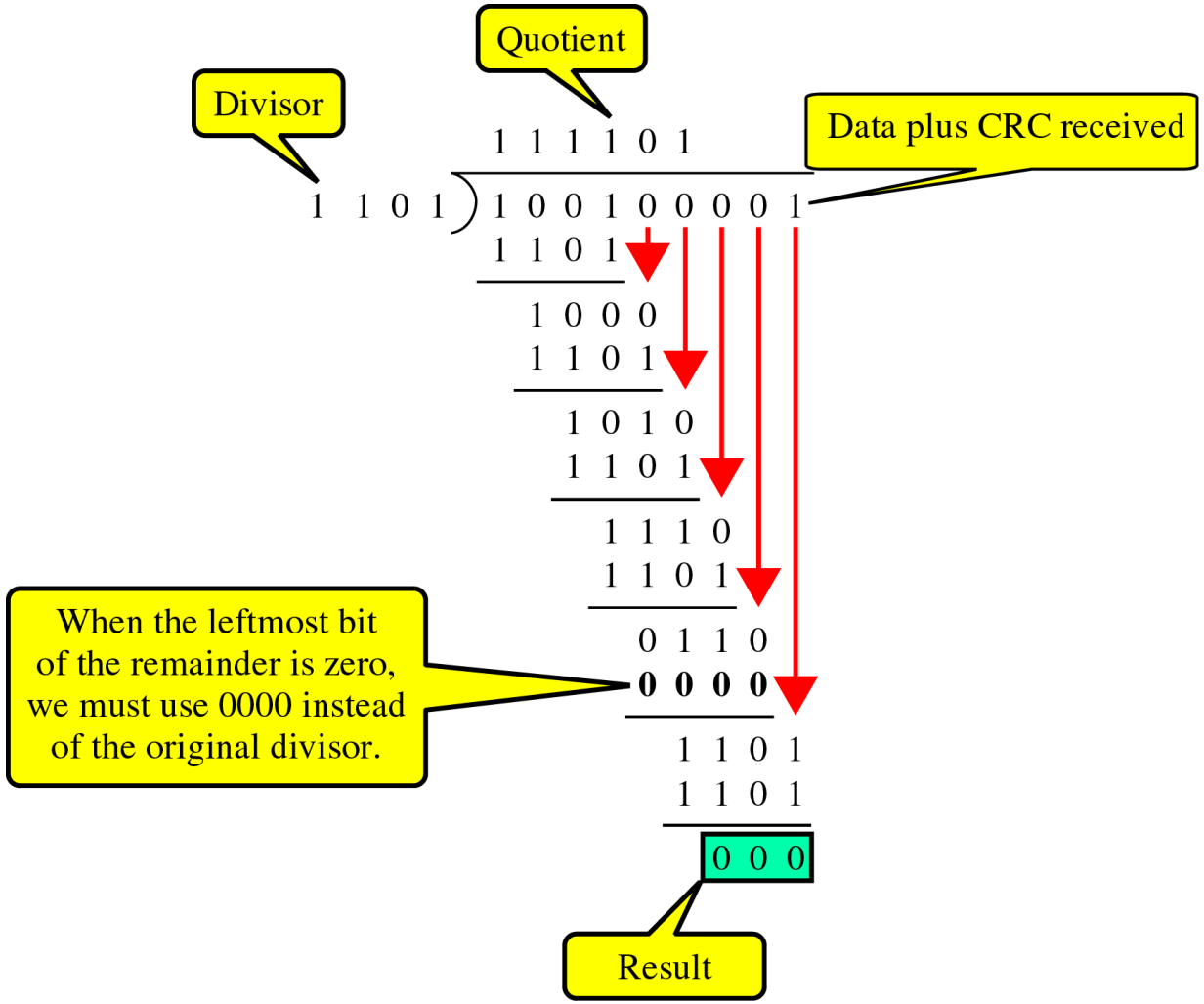
CRC (Cyclic Redundancy Check)

- is based on binary division.



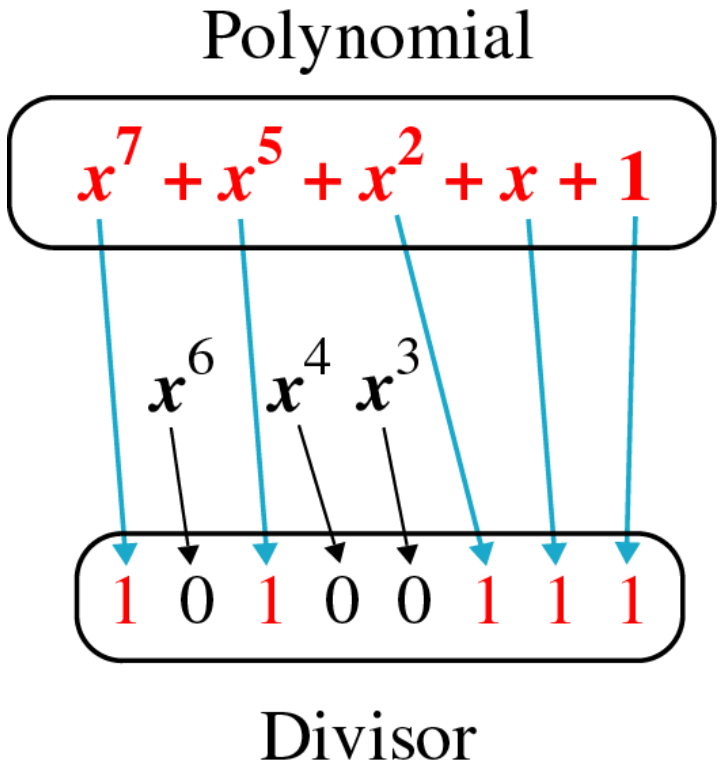
- Using Modular 2 Division




Quotient
Divisor
Data plus CRC received
1 1 1 1 0 1
1 1 0 1) 1 0 0 1 0 0 0 0 1
1 1 0 1
1 0 0 0
1 1 0 1
1 0 1 0
1 1 0 1
1 1 1 0
1 1 0 1
0 1 1 0
0 0 0 0
1 1 0 1
1 1 0 1
0 0 0
When the leftmost bit of the remainder is zero, we must use 0000 instead of the original divisor.
Result

Polynomials

- CRC generator (divisor) is most often represented not as a string of 1s and 0s, but as an algebraic polynomial.



Standard Polynomials

CRC-12

$$x^{12} + x^{11} + x^3 + x + 1$$

CRC-16

$$x^{16} + x^{15} + x^2 + 1$$

CRC-ITU-T

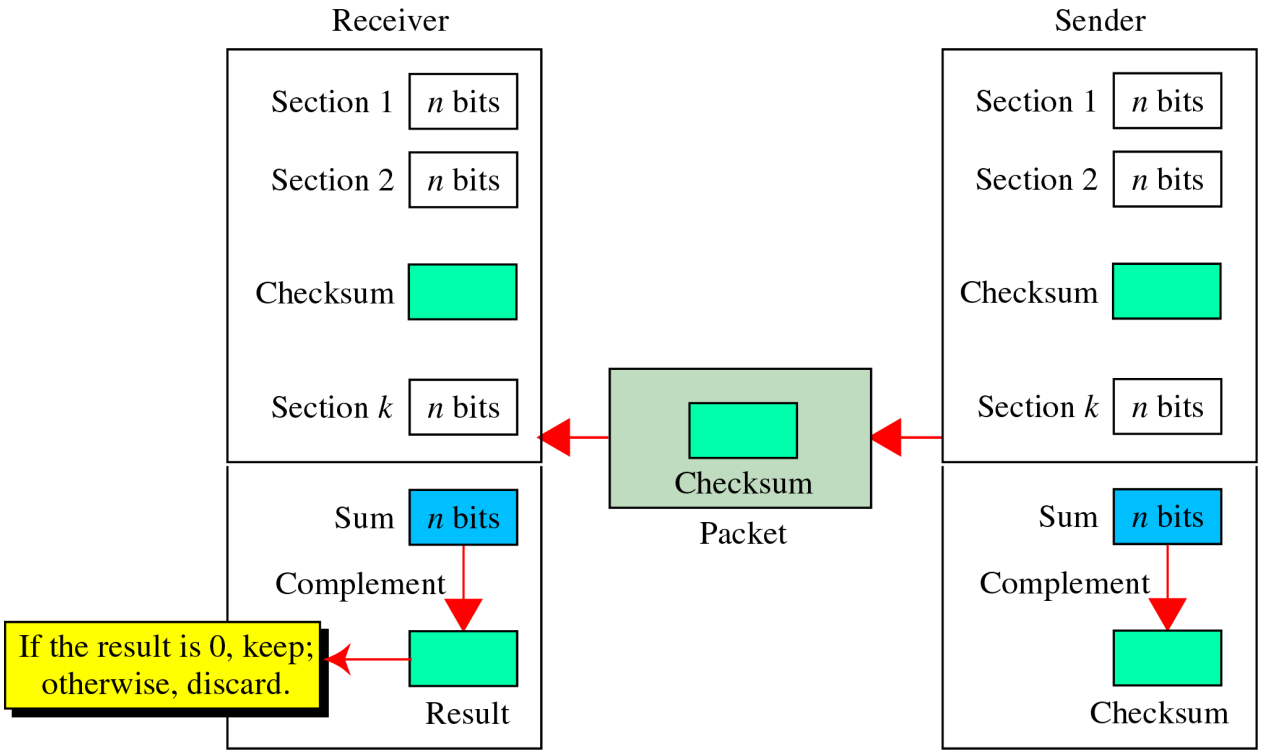
$$x^{16} + x^{12} + x^5 + 1$$

CRC-32

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

Checksum

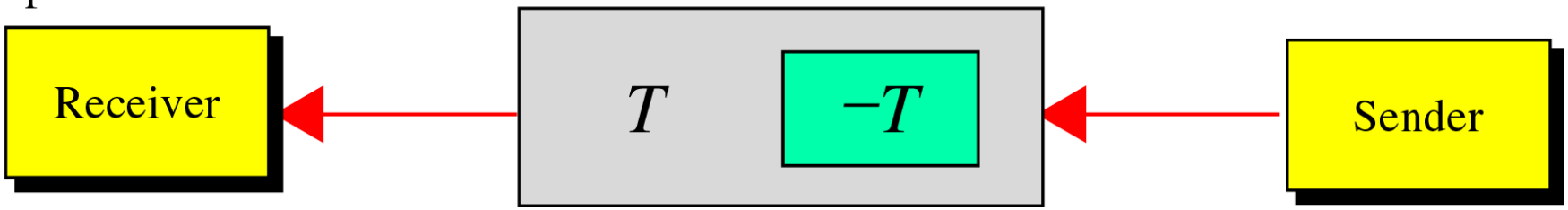
- Checksum
- used by the higher layer protocols
- is based on the concept of redundancy (VRC, LRC, CRC)

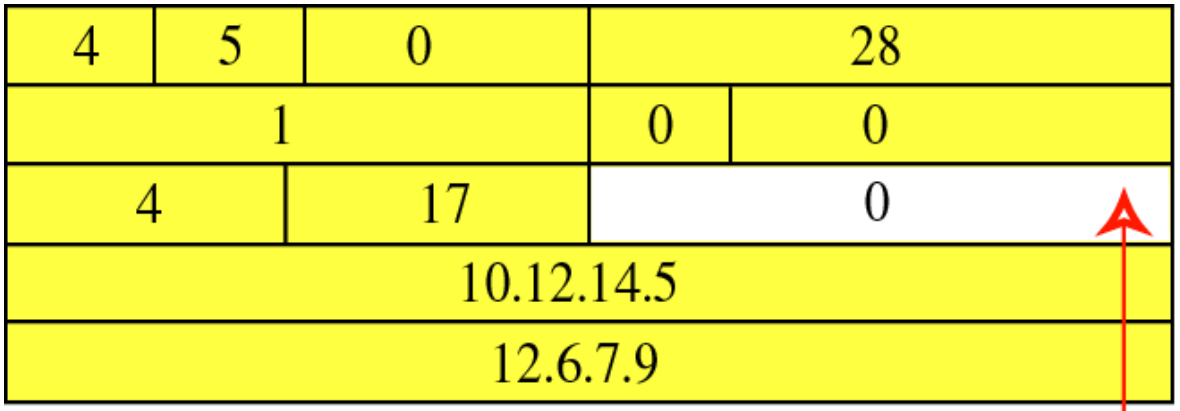


- To create the checksum the sender does the following:
  - The unit is divided into K sections, each of n bits.
  - Section 1 and 2 are added together using one's complement.
  - Section 3 is added to the result of the previous step.
  - Section 4 is added to the result of the previous step.
  - The process repeats until section k is added to the result of the previous step.
  - The final result is complemented to make the checksum.

The receiver adds the data unit and the checksum field. If the result is all 1s, the data unit is accepted; otherwise it is discarded.

$$
\begin{array}{rr}
 & T \\
 & -T \\
\hline
\text{Sum} & -0 \\
\text{Complement} & 0
\end{array}
$$





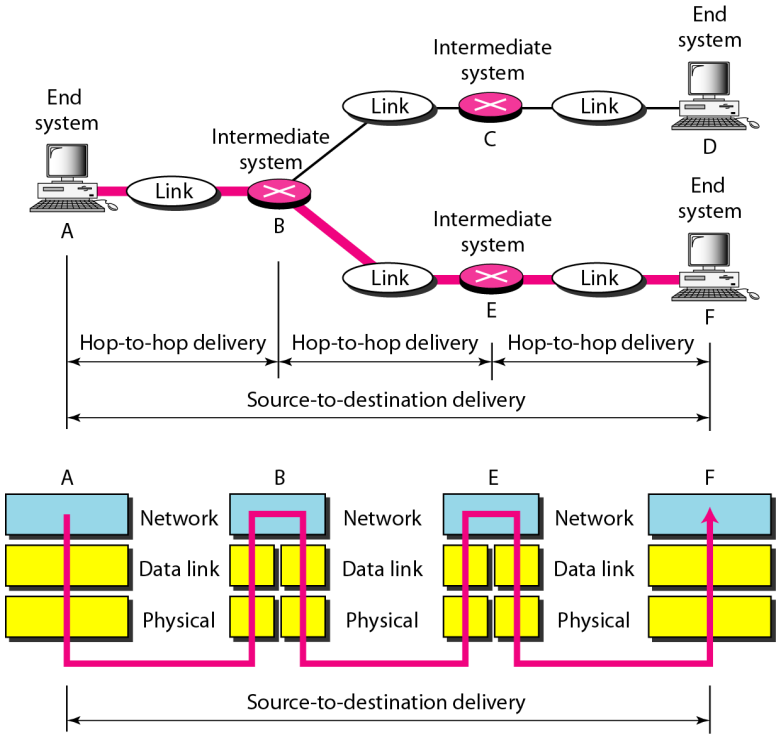
4, 5, and 0	→	01000101 00000000
28	→	00000000 00011100
1	→	00000000 00000001
0 and 0	→	00000000 00000000
4 and 17	→	00000100 00010001
0	→	00000000 00000000
10.12	→	00001010 00001100
14.5	→	00001110 00000101
12.6	→	00001100 00000110
7.9	→	00000111 00001001
Sum	→	01110100 01001110
Checksum	→	10001011 10110001

Network Layer

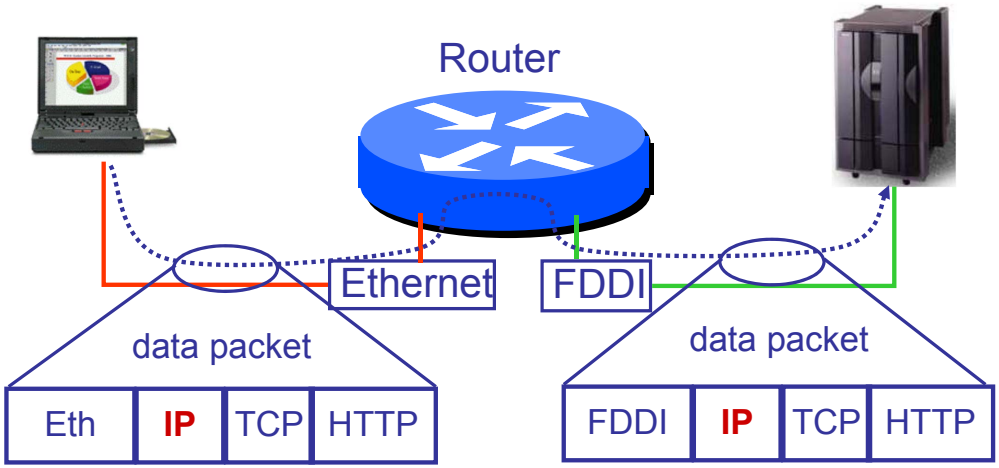
- The network layer is responsible for host to host delivery and for routing the packets through the routers.

- Routers forwards packets from source to destination
  - May cross many separate networks along the way

- All packets use a common Internet Protocol
  - Any underlying data link protocol
  - any higher layer transport protocol



IP Networking

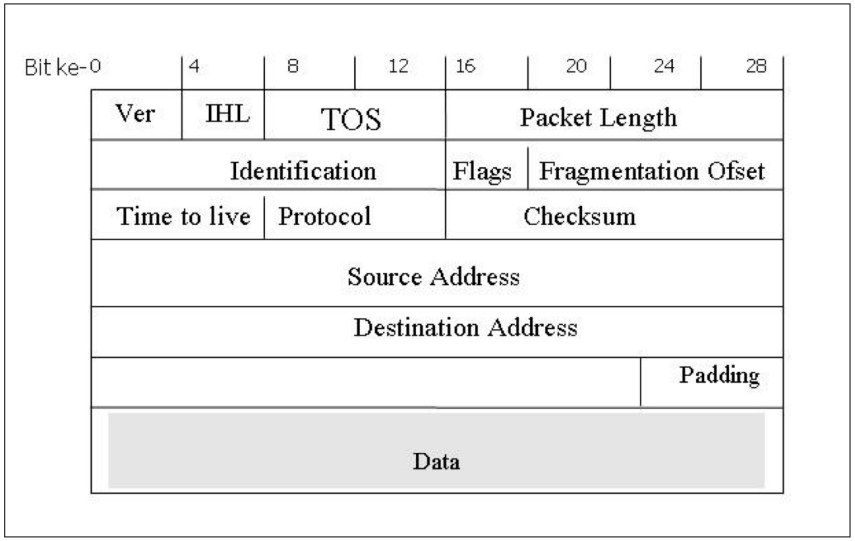


So what does IP do?

- Addressing
- Fragmentation
  - e.g. FDDI maximum packet is 4500 bytes while Ethernet is 1500 byte, how to manage this?
- Some error detection
- Routers only forward packets to the next hop
  - they do not:
    - detect packet loss, packet duplication
    - Reassemble or retransmit packets

IP Header

- **Ver** : Which version of IP is this?
- **HL** : Header Length
  - How big is IP header? (in bytes/octets)
- **TOS** : Type of Services
  - care/don't care for delay, throughput, reliability, cost
- **Length** : How long is whole packet in bytes/octets?
  - Includes header
  - Limits total packet to 64K
  - Redundant?
- **TTL**: Time To Live
  How many more routers can this packet pass through/
  - Designed to limit packet from looping forever
  - Each router decrements TTL fields
  - If TTL is 0 then router discards packet
- **Protocol**: Which transport protocol is the data using?
  - i.e how should a host interpret the data
- **Checksum**
  - Header contains simple checksum
    - validates content of header only
  - Recalculated at each hop
    - Routers need to update TTL
    - Hence straightforward to modify



EXAMPLE 3-10

What is the checksum value for the extended ASCII message "Help!"?

SOLUTION

The checksum value is found by adding up the bytes representing the Help! characters:

01001000	H
01100101	e
01101100	l
01110000	p
00100001	!
00010000	Checksum

Error Correction

- can be handled in two ways
  1. when an error is discovered, the receiver can have the sender retransmit the entire data unit.
  2. a receiver can use an error-correcting code, which automatically corrects certain errors.