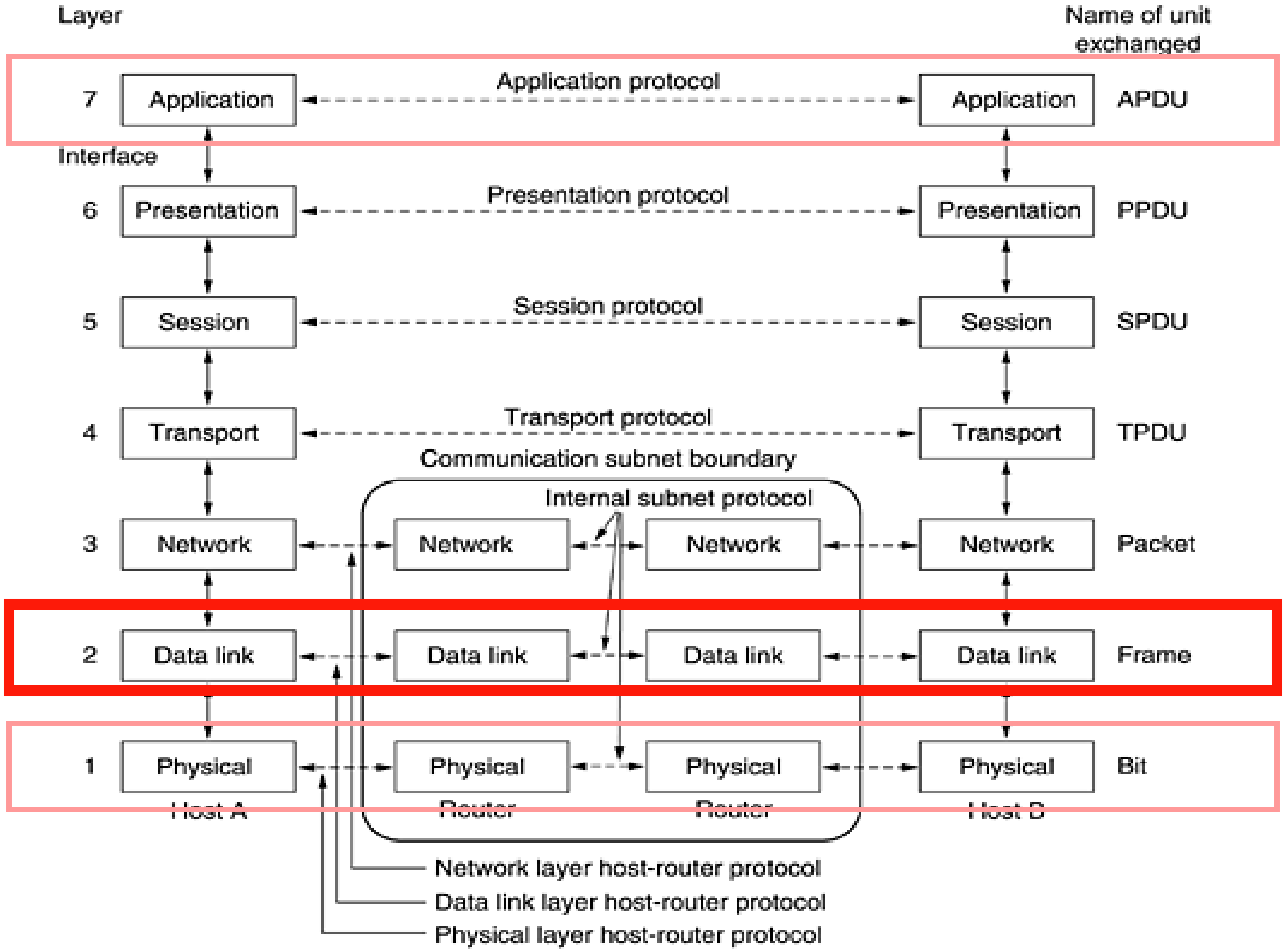
Computer Networks

Error Detection and Correction & Media Access Control

The Data Link Layer

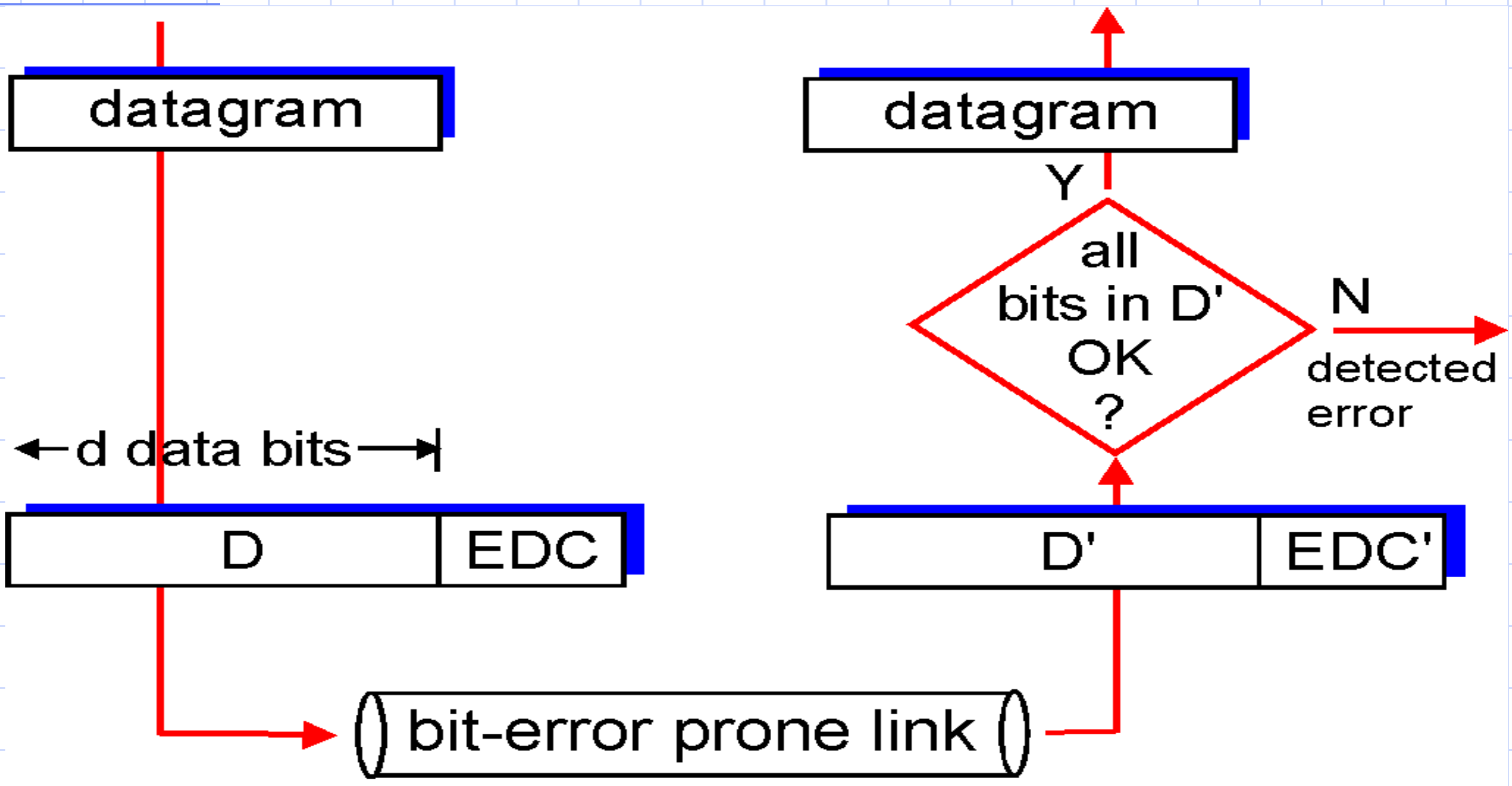


All **P**eople **S**eem **T**o **N**eed **D**ata **P**rocessing

Handling Errors

- Data can be corrupted during transmission
  - Bits lost
  - Bits changed
  - Bits added
- Frame additional data to protect
  - Link-level addressing, seq no., etc
- **Handling ? – add redundant bits (data)**

Error detection/correction scenario



Send **D+EDC**

Receive **D’+EDC’**

Detection vs Correction

- Error Detection Techniques
  - Allow for error detection but no possible correction
  - Require frame re-transmission
  - Used in low error rate transmission medias (fiber optics).
- Error correction techniques (FEC)
  - Involve more redundant data and processing power
  - Used in high error transmission medias (radio)

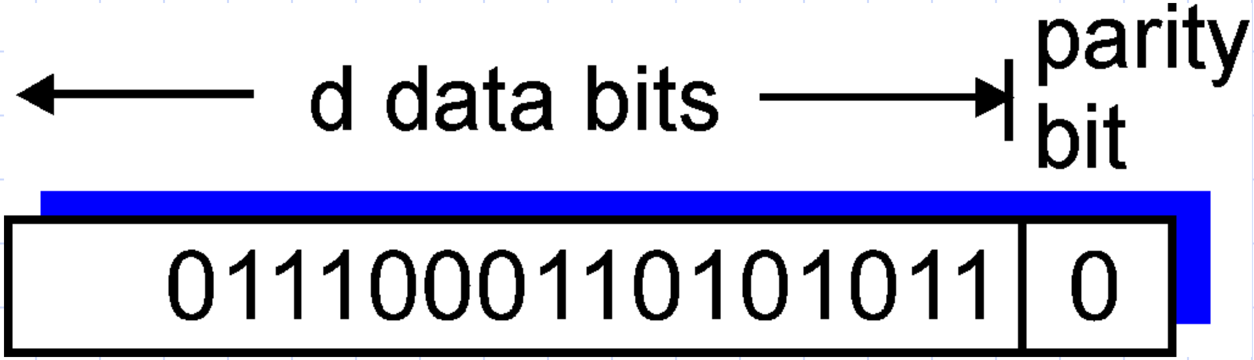
Advantages/disadvantages of Error detection/correction

- Error detection/correction techniques allow the receiver to sometimes *but no always* detect errors.
- *Undetected errors* might still remain => corrupted packets delivered to the network layer.
- **Goal** – have techniques that minimize the number of undetected errors

Detection/Correction Techniques

- Parity Checks
- Checksumming methods
- Cyclic redundancy checks

Parity Checks



- Parity Bit (PB)
  - One additional bit per character
  - *Even parity*
  - *Odd Parity*

How many bit errors can PB detect ?

10001110 ---→ 10101110 => error !

10001110 ---→ 10100110 => **No error detected** !!!

Conclusion – 1 PB can only detect an odd number of errors !

Hamming Distance

- **Hamming distance** = the number of bit positions in which two code-words differ.
- How to calculate ?(Exclusive OR=XOR):

10001001
10110001

00111000

=> The number of 1's give the number of different bits.

Hamming and error detection

- Error detection of *d* single-bit errors needs a *d*+1 distance code.

- Example:
  - BP has a distance of 2 => can detect single bit errors.

Bit Parity – YES or NO ?

Suppose a channel with BER: $p=10^{-4}$ =>

1) P(sb error)=p
2) P(no sb error)=1-p
3) P(no error in 8 bits)=$(1-p)^8$
4) P(undetected error in 8 bits)=$1-(1-p)^8$

P(undetected error in 8 bits)=**7.9×10^{-4}**

Bit Parity – YES or NO ? (2)

- After adding a <u>parity bit</u> :

P(no sb error)=1-p

P(no error in 9 bits)=$(1-p)^9$

P(sb error in 9 bits)=9xP(sb error)xP(no error in 8 bits) = $9p(1-p)^8$

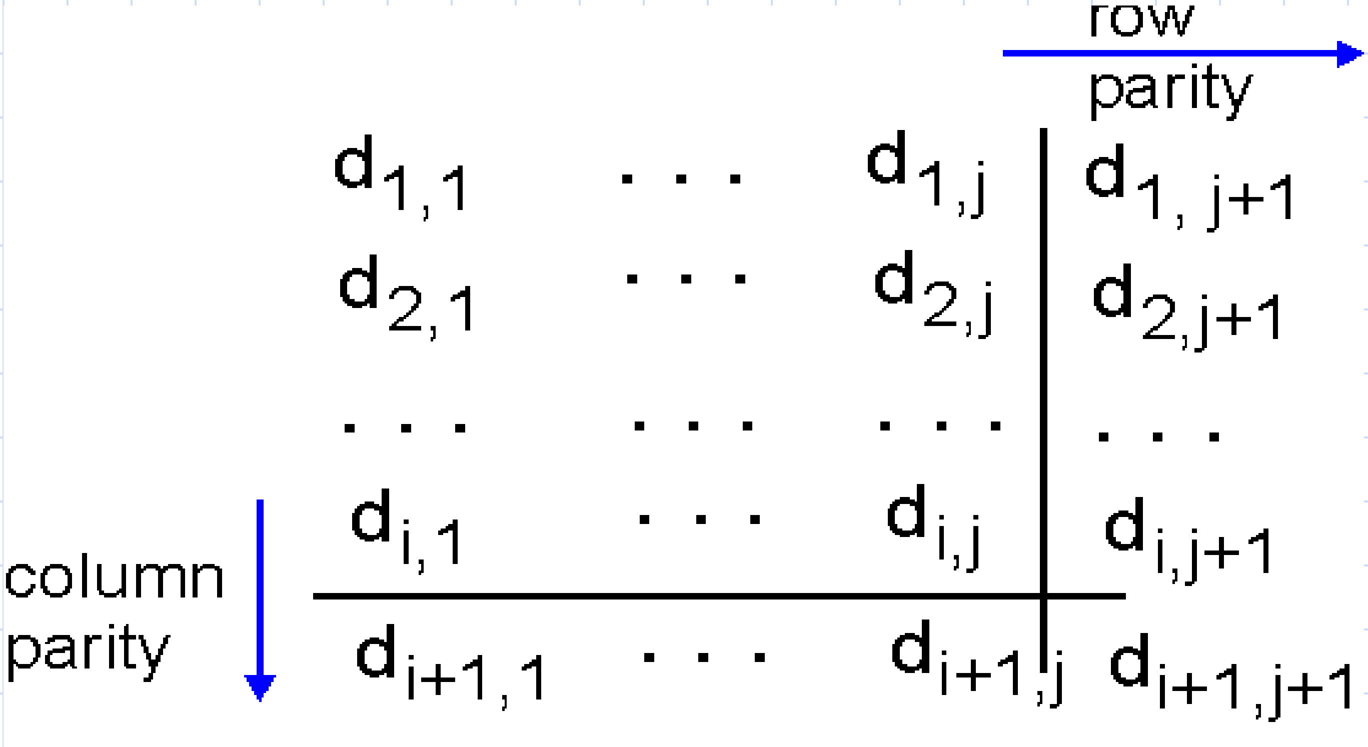
P(undetected error in 9 bits)=1-P(no error in 9 bits)-P(sb error in 9 bits)

$$= 1-(1-p)^9 - 9p(1-p)^8$$

⇒P(undetected error in 9 bits)=**3.6×10^{-7}**

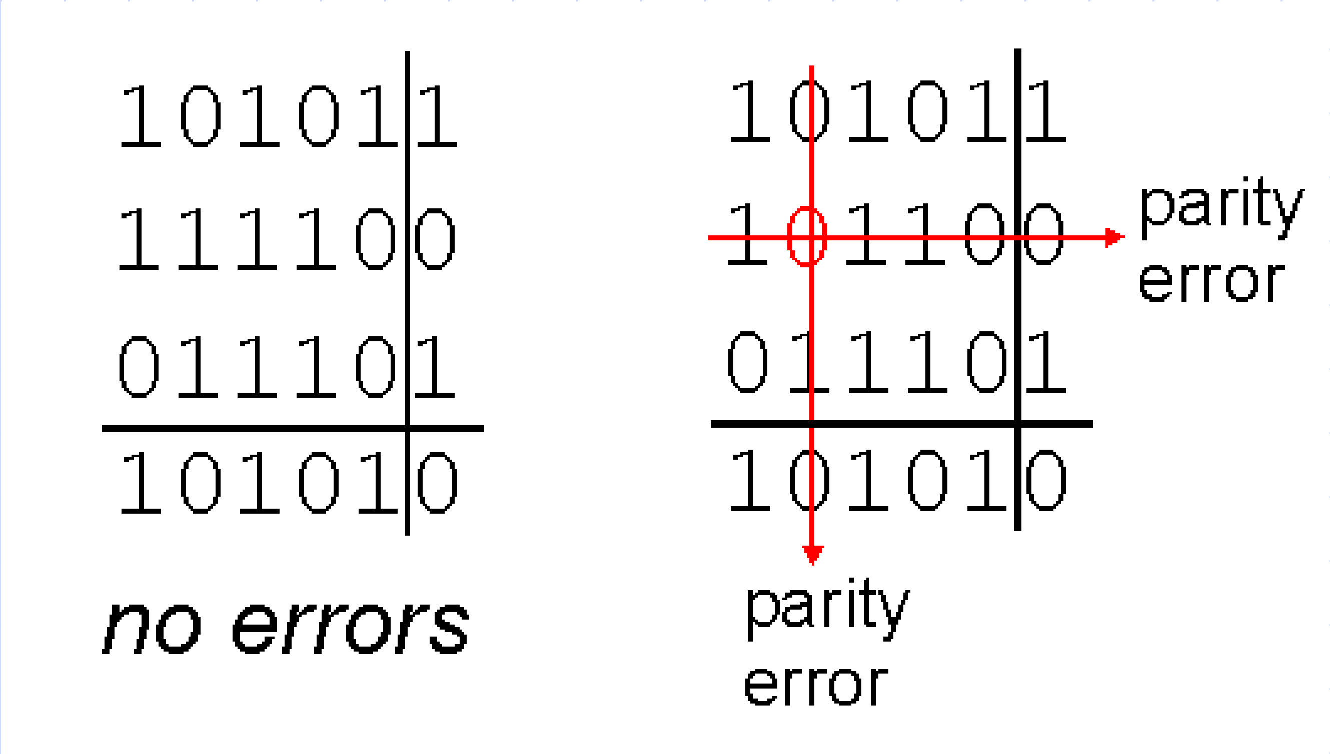
$$\frac{\text{P(undetected error in 8 bits)} \quad 7.9\times10^{-4}}{\text{P(undetected error in 9 bits)} \quad 3.6\times10^{-7}} \sim= 10^3$$

Single Bit Error Correction



Parity for each character(byte=line) + parity for each column (set of data bytes sent)

Example - Single Bit Error Correction



Hamming - Correctable single bit error

Correction vs Detection - Practice

- Detection techniques
  - For detecting d errors we need $d+1$ distance code.
- Correction Techniques
  - For correcting d errors we need $2d+1$ distance code.

Error Correction

- Valid codewords: 0000000000, 0000011111, 1111100000, and 1111111111
- The Hamming distance of the code=5 => we can correct 2d+1=5 => d=2 bit errors.

a) 0000000000---→0000000011 => the closest code is still 0000000000 !
b) 0000000000---→0000000111 => the closest code is not correctly determined anymore !!

Hamming correcting code

- Bits numbered from lsb to msb 1…n
- Positions power of 2 = check bits => bits 1,2,4,8… etc check bits
- Bit *k* from the sequence is checked by the positions from its binary decomposition k=11=1+2+8 => bits 1,2,8 are check bits

Hamming correcting code

In order to send 7 data bits we need 4 check bits.

Data: 1001101

Check bits 4 : 1,2,4,8

11	10	9	8	7	6	5	4	3	2	1	
1	0	0	x	1	1	0	x	1	x	x	sent as
1	0	0	1	1	1	0	0	1	0	1	

Hamming correcting code

10011100101 is sent as 00011100101

=>the error bit is given by the indices that are in error

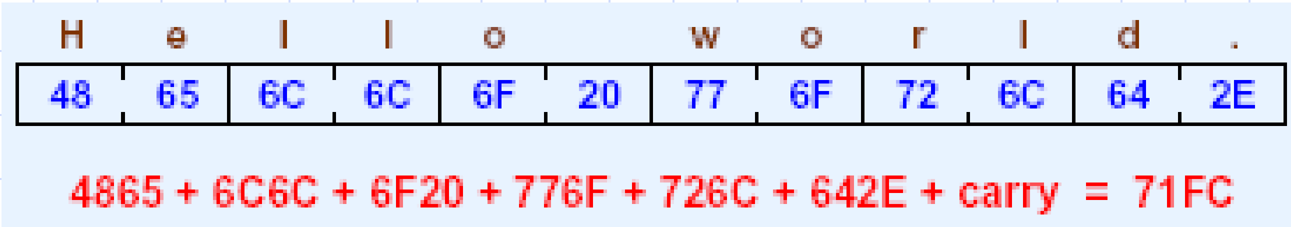
8=[11]+[10]+[9] – error =>k=8

4=[7]+[6]+[5] – ok =>k=8

2=[11]+[10]+[7]+[6]+[3]–error=>k=10

1=[11]+[9]+[7]+[5]+[3]-error =**<u>k=11</u>**

Checksum Codes



soh	data	eot	Checksum

- Byte stream interpreted as series of numbers (16 bit integers)
- Integers are added =>checksum appended to the frame.
- Receiver calculates again the checksum and discovers the errors.

Errors Checksum fails to detect

Data Item Binary	Checksum Value
0001	1
0010	2
0011	3
0001	1
Total	**7**

Data Item Binary	Checksum Value
0011	3
0000	0
0001	1
0011	3
Total	**7**

- Second bit inverted for each value
- Checksum is the same

Cyclic Redundancy Check (CRC)

- Bit strings represented as polynomials with coef. 0 and 1.
- K bit frame =>$x^{k-1}+\ldots+1$ (first and last coef must be 1)
- Example
  - 110001 => x^5+x^4+1
- Polynomial arithmetic is done module 2 i.e. ⬄ addition/subtraction = XOR operation

CRC (2)

- Sender (S) and Receiver (R) agree on a *generator polynomial* G(x)
- Frame – m bits => M(X) – the checksum of m is the remaining of R(x)=M(x)/G(x)
- Checksum added to frame.
- (R) Gets the frame M'(x)=[M(x)-R(x)]
  - If M'(x)/G(x) has remainder => **error**

CRC (3)

- Frame *m* bits. Generator *r* bits.
- Calculate: $x^r M(x)$ – m+r bits
- $x^r M(x) / G(x)$ – take remainder R(x)
- Send: $T(x) = x^r M(x) - R(x)$

soh	data	eot	CRC

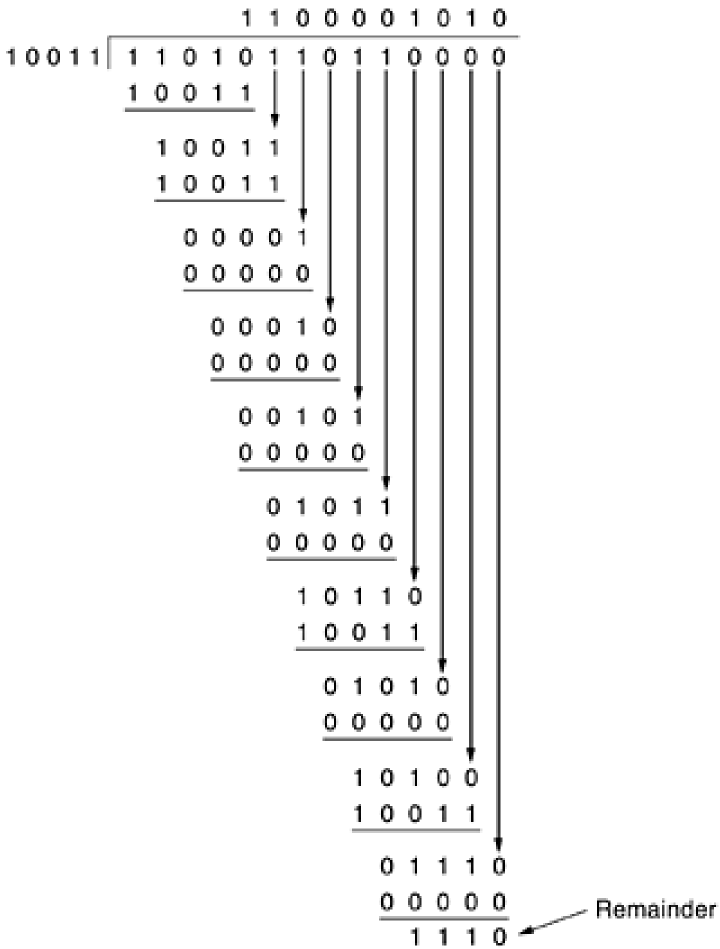
- Receiver: T(x) should be divisible with G(x). If not we have transmission errors.

CRC - Example

Frame : 1 1 0 1 0 1 1 0 1 1

Generator: 1 0 0 1 1

Message after 4 zero bits are appended: 1 1 0 1 0 1 1 0 1 1 0 0 0 0



Transmitted frame: 1 1 0 1 0 1 1 0 1 1 1 1 1 0

Frame – 1101011011

$G(x)=x^4+x+1$

<u>Transmitted frame:</u>

11010110110000 –

00000000001110

11010110111110

CRC (5)

- IEEE 802 uses:

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

Catches all $\leq$ 32 bit error bursts and all odd length error bursts.