Overview: Physical & DataLink Layers

- How are frames assembled and transmitted ?
- Dealing with transmission errors
    - Error detection
    - Error correction

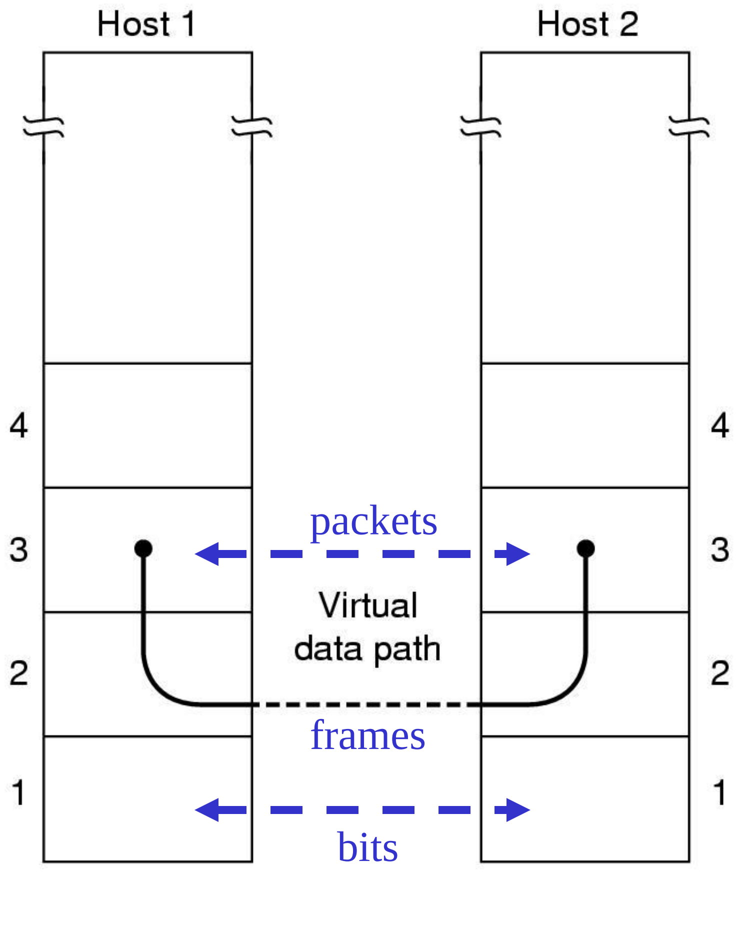
Datalink layer

- how to achieve (reasonably) reliable communication between 2 adjacent computers
  - adjacent = physically connected by a communication channel, assumed to deliver bits in their transmitted order

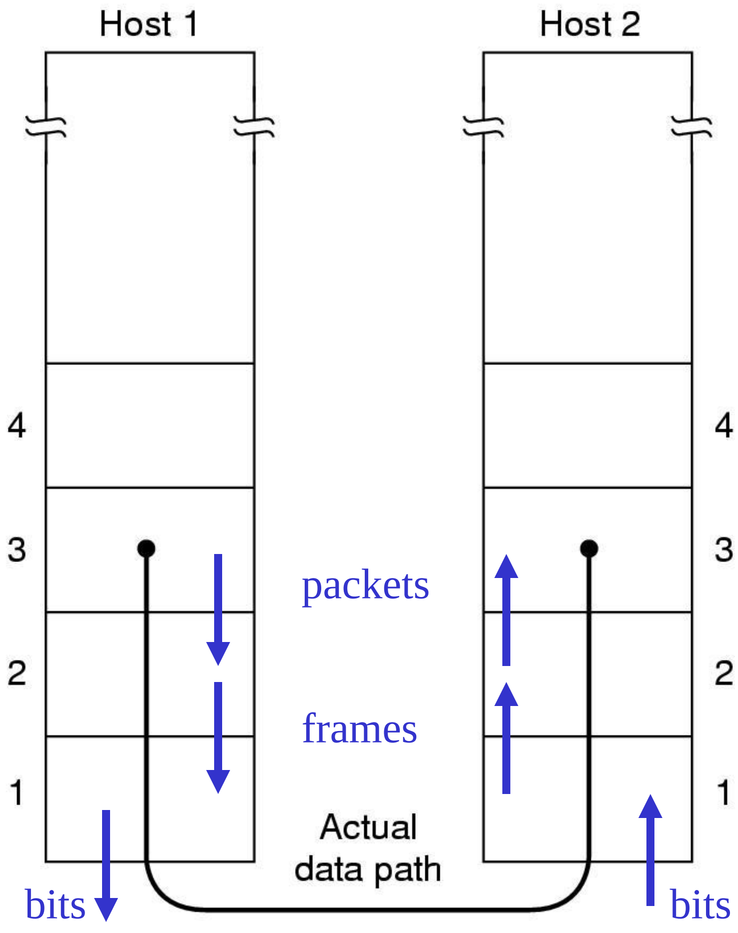
- what makes this problem difficult ?
  - errors can occur (transmit 0/1, receive 1/0)
  - transmission delay
  - propagation delay

- functions of the datalink layer include:
  - determining how bits in physical layer are framed
  - dealing with transmission errors
  - controlling the flow of frames so that the receiver is not overwhelmed by the sender

Datalink layer (cont.)



(a)
datalink layer virtual communication

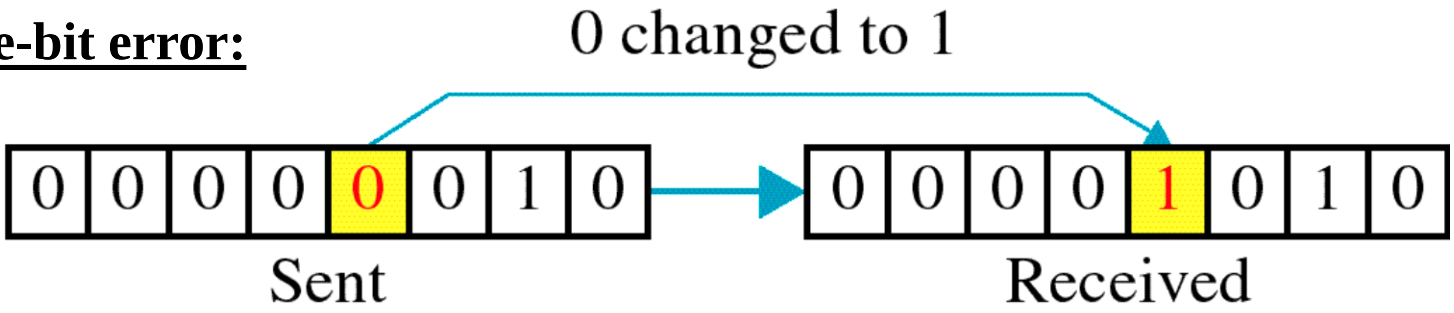


(b)
datalink layer actual communication

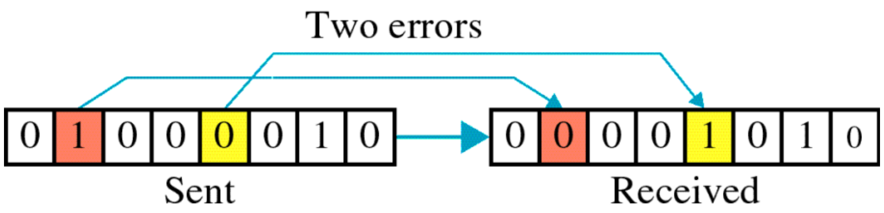
Datalink layer: Framing

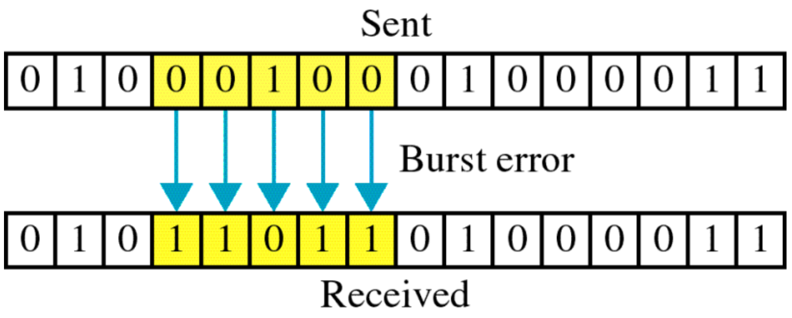
- physical layer offers ***unreliable bit pipe*** service

single-bit error:



burst errors:





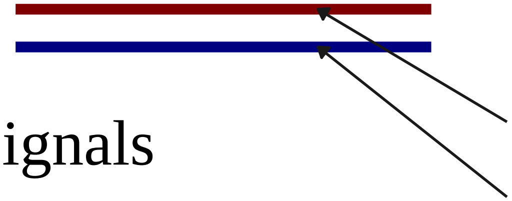
- usual approach: datalink layer entities exchange frames which contain extra information that allows them to detect/correct transmission errors ⇒ need to define what a frame is

Errors and Media Types

- As signals propagate along a Transmission Line they suffer from Attenuation and Distortion, the extent of which are determined by:
  - Transmission Media Type
  - Bit Rate
  - Distance between the Communicating Devices
- Often the received signals will be so badly attenuated and distorted they will be incorrectly interpreted by the receiver, resulting in a Bit Error. Bit Error Rate (BER) is the main performance metric.
- There are a number of Transmission media types used for Data Communications. The choice of medium depends on:
  - Distance to be covered
  - Desired Bite Rate (in bits per second, b/s)
  - Cost Considerations

Two-Wire Open Lines / Twister Pair

- Used mainly for directly connecting devices over small distances (<50m) and at moderate bit rates
- Signals get distorted due to:
  - Cross-talk between the two signals
  - Susceptibility to Noise Signals



Signal Wire

Reference Wire

- **Twister Pair**
- The proximity of signal and reference lines means noise signals are picked up by both wires
- Also have shielded twisted pairs, giving even better noise immunity.
- Even with no shielding can get high bit rate at very low cost.
- Offer higher bit rates over 10 Mb/s, over short distances (<100 m) or lower rates over longer distances.

Coaxial Cable

- This is made up of an inner conductor surrounded by an insulator and that surrounded again by the outer conductor.
- Use of the dielectric material and outer conductor effectively isolate the core conductor from external noise interference.
- Coaxial cables can be used at rates from over 100 Mbps, over distances of several hundred meters, up to higher bit rates of Gb/s over shorter distances ~100 meters.

Optical Fibre

- Optic fiber does not use electrical signals to transmit the data, rather it uses light. It transmits these signals down a thin piece of glass.
- Because data is transferred using light beams significantly higher bit rates, maybe from 100's of Mb/s to many Gb/s or even Tb/s can be achieved.
- Have no electrical components and are immune to electromagnetic interference and cross-talk.
- Can use Hard Core Silica (HSC) or plastic for lower speeds or distances

Satellite

- The first satellites were launched in the 1970’s and now the most common are the Geo-stationary ones at 36,000 km above the earth.
- Satellites are using direct line of sight with the transmitters and receivers.
- Data transmitted using electromagnetic waves propagating through the atmosphere at > 4GHz.
- Typically many signals will be multiplexed onto a single satellite channel utilizing a high bit rate.

Terrestrial Microwave/Radio

- Again these provide direct line of sight between the transmitter and receiver where possible.
- Provide communication links when it is impractical or too costly to install a physical link.
- Suffer from factors such as bad weather conditions and obstruction by man-made objects.
- Use radio waves (150kHz - 1GHz) for long distances and microwaves (>1GHz) for shorter distances but limited by the curvature of the earth.

Attenuation & Distortion

- Signal “Attenuation” is the phenomenon whereby the Amplitude of a signal decreases as it propagates along a transmission line.
- Sometimes sets of amplifiers, also known as repeaters, are placed at intervals along the line to ensure acceptable signal amplitude is maintained.
- Signal “Distortion” involves the shape of the signal becoming altered as it propagates along the line. This is more complex to explain.

Propagation Distortion

- The propagation rate of a pure sinusoidal signal depends on it’s frequency.
- The components of a data signal will propagate at different speeds, arriving at the receiver with varying delays, resulting in Delay Distortion.
- If the frequency components of one bit are delayed sufficiently they will overlap with the next bit resulting in Inter Symbol Interference (ISI).
- ISI is an important effect as it causes the bit transition instants of the signal to change.

Noise

- On any line, even in the absence of a data signal, random perturbations of the line voltage and current will occur.
- This effect is known as Line Noise Level, or simply Background Noise.
- There are three main causes of this noise:
    - Cross-talk
    - Impulse Noise
    - Thermal Noise

Cross-talk & Impulse Noise

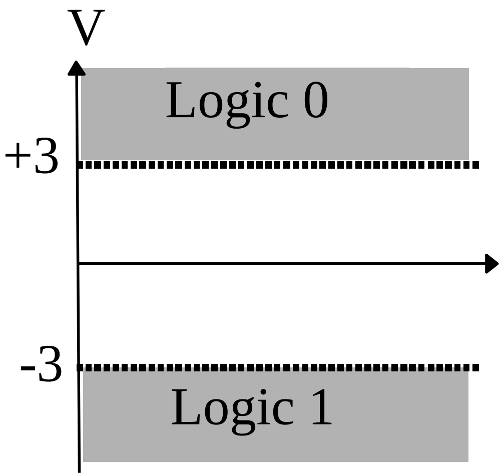
- Cross-talk - Occurs when a signal on one line is picked up by adjacent lines as a small noise signal.
- Particularly troublesome is Near-End Cross-talk (NEXT) caused when a strong transmitter output signal interferes with a much weaker incoming receiver signal.
- Impulse Noise - Caused by external activity or equipment. Generally takes the form of electrical impulses on the line which cause large signal distortion for their duration.

Thermal Noise

- Thermal noise is always present because it is caused by the thermal agitation of electrons associated with each atom in the device or transmission line material.
- It consists of random frequency components of continuously varying amplitude.
- For this reason it is also known as White Noise.
- One of the main aims of transmission is to contain the effects of the different type of noise.

Noise & Voltage Levels

- The RS-232C was originally defined for connecting a device to a modem, over a short physical length using a relatively low bit rate.
- Has been adapted as the interface for connecting any character-oriented peripheral to a computer.



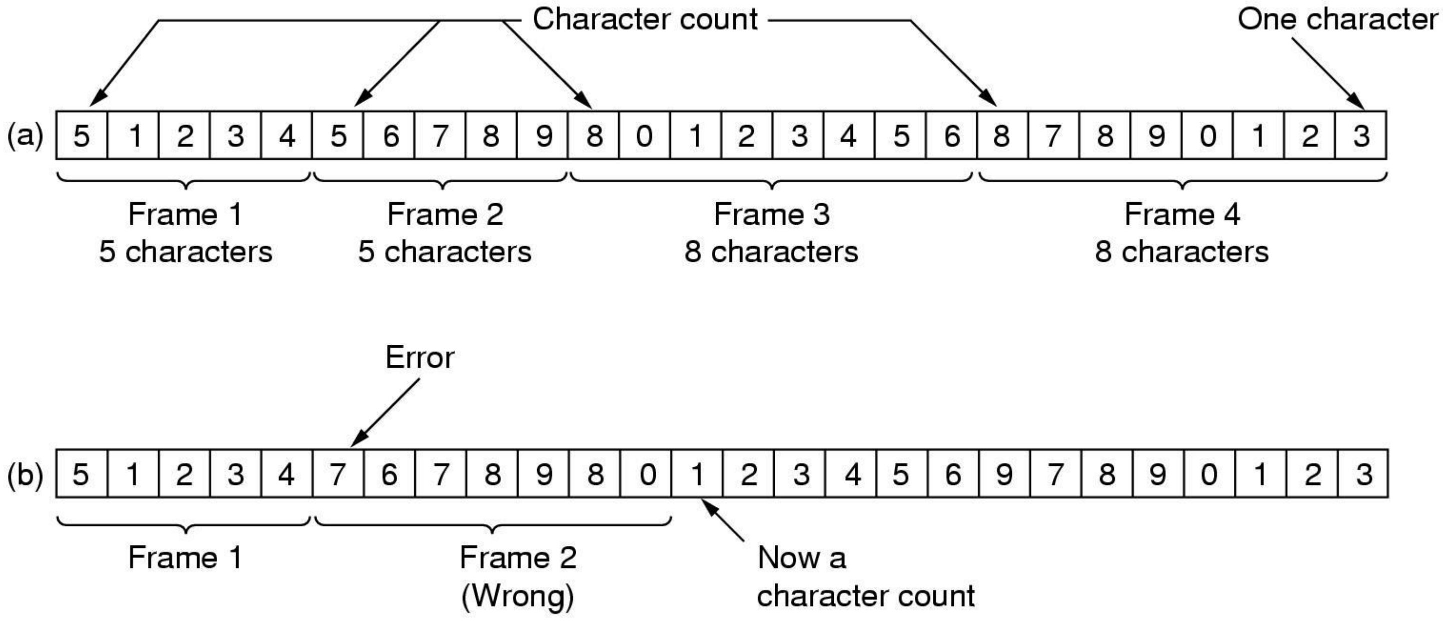
Actual voltage levels used are typically 12V or 15V.

Datalink layer: Framing (cont.)

- how can frames be defined ?
  - time intervals between frames
    - these inter-frame gaps may be changed (shorter/longer) during transmission
  - character count – a field in the frame header specifies the number of characters in the frame
    - problems if the character count field is corrupted during transmission

OK

?!?



(a) 5 1 2 3 4 5 6 7 8 9 8 0 1 2 3 4 5 6 8 7 8 9 0 1 2 3

(b) 5 1 2 3 4 7 6 7 8 9 8 0 1 2 3 4 5 6 9 7 8 9 0 1 2 3

Datalink layer: Framing (cont.)

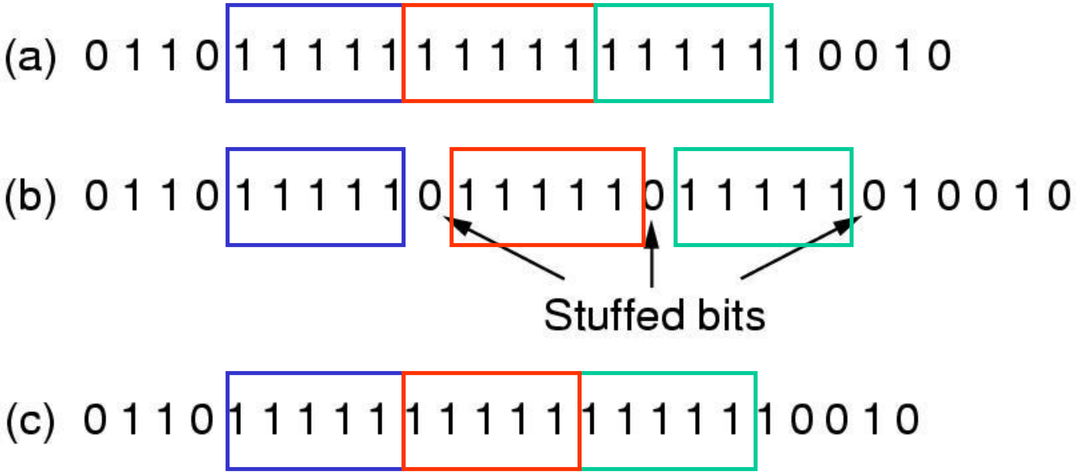
- starting and ending flag, with bit-stuffing
  - flag = 0111 1110 at start and end of each frame
  - have to make sure flag doesn’t occur in the data part of the frame: whenever sender finds 5 consecutive 1’s in the data, it automatically inserts a 0 (***bit-stuffing***); when receiver sees 5 consecutive 1’s followed by a 0, it removes the 0 (***destuffing***)

original data

transmitted data

(start and end flags not shown)

destuffed data


(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

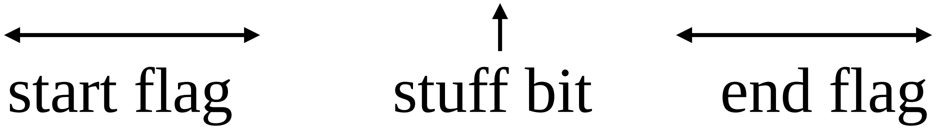
- the start and end flags are not bit-stuffed, so the receiver can scan the incoming bitstream for frame boundaries

Datalink layer: Framing (cont.)

- Note: even if the data contains a 0 after 5 consecutive 1's, the sender still stuffs a 0

  e.g. data = 001111101000

  transmitted = 011111100011111001000001111110



start flag stuff bit end flag

  destuffed = 001111101000, which is correct

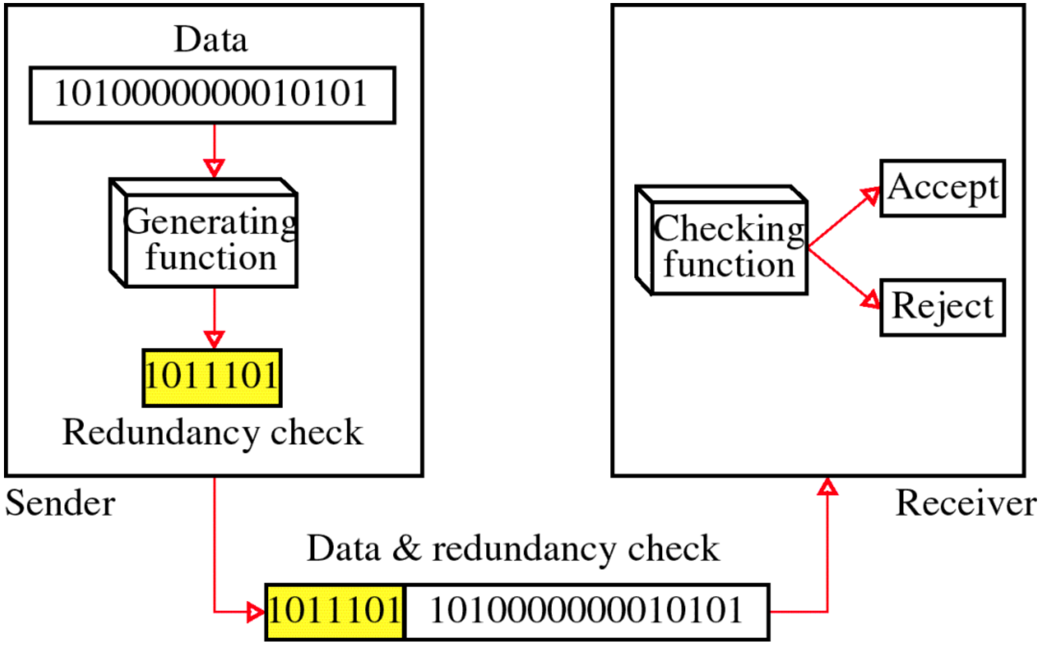
  - why ? Because if it didn't do this, how would the receiver know which 0's were stuff bits and which were data ?!

- other methods also used
  - e.g. special character sequences used as start and end flags, which means the data needs to be "character-stuffed" to prevent the flags occurring in the data
- in practice, use a combination of these methods
  - e.g. character count + 0111 1110 flags

Datalink layer: Dealing with transmission errors

- at the receiver, data contained in frame is an arbitrary bit string
  - this means that the receiver is not allowed to know which bit strings correspond to "legal" data and which don't (in accordance with layered encapsulation principles)
- therefore some additional bits must be added to the frame to allow errors to be detected (and possibly corrected)

Example:



"redundancy" because the extra bits are only used to check the accuracy of the transmission, and are discarded by the receiver as soon as this has been done

Datalink layer: Dealing with transmission errors (cont.)

- let pB = the probability that a single bit is received incorrectly
  - e.g. pB = 10^{-10} on optical fibre, 10^{-7} on coaxial cable
- if a frame contains N bits, and bit errors occur independently of each other, what is pF = the probability that the frame is received incorrectly (in this case, meaning: 1 or more of the N bits are in error) ?
  - pF approximately = N×pB (under usual circumstances)
  - if pB = 10^{-7} and N = 10000, then pF = 10^{-3} *…very high*
- solution 1: send each frame three times
  - receiver takes majority value for each bit as correct
  - now pF approximately = 3×N×pB^2 (under usual circumstances)
  - if pB = 10^{-7} and N = 10000, then pF = 3×10^{-10} *…ok*
  - but: transmission rate effectively reduced by a factor of 3, plus additional delays to examine 3 copies of each bit
- solution 2: send each frame twice
  - receiver can compare 2 copies: if identical, assume correct; if different, know error(s) occurred, ask for frame (& copy) to be retransmitted
  - but: transmission rate effectively cut in half, plus additional delays to compare frame copies bit-by-bit

Datalink layer: Dealing with transmission errors (cont.)

- frame consists of m "data" bits and r redundancy bits (also called *check bits*) for error detection/correction
  - total length n = m+r bits
  - the n-bit string comprising a frame is called a ***codeword***

- solution 3: r = 1, choose check bit so that **total number of 1's** (including the check bit itself) in every valid codeword is **even**
  - this is the ***single parity check***; the check bit is often called the ***parity bit***
  - equivalent definition: parity bit = 1 if total number of 1's in the data is odd; parity bit = 0 if total number of 1's in the data is even

- if receiver gets a codeword with an odd number of 1's, it knows error(s) occurred ⇒ can **detect** any odd number of bit errors
  - but: can't tell how many errors, or which bits are in error
  - even worse: any even number of bit errors is *undetectable*

Datalink layer: Single Parity Check (cont.)

- Example 1: data to be transmitted = 10110101
  - 5 1's in the data ⇒ parity bit is 1
  - transmitted codeword = 101101011
  - if receiver gets 101101011, parity check ok ⇒ accept (OK)
  - if receiver gets 101100011, parity check fails ⇒ reject (OK), ask for frame to be re-transmitted
  - if receiver gets 101110011, parity check ok ⇒ accept (NOT OK: even number of errors undetected)
  - if receiver gets 001100011, parity check ok ⇒ accept (NOT OK: even number of errors undetected)

- Example 2: data to be transmitted = 10110001
  - 4 1's in the data ⇒ parity bit is 0
  - transmitted codeword = 101100010

Datalink layer: <u>**solution 4:**</u> 2-Dimensional Parity Check

- form data into a 2-dimensional array; add single parity check bits to each row and each column; transmit row-by-row
- Example: data = 1110001 1000111 0011001
  - form 3×7 array and add row and column parity bits:

1 1 1 0 0 0 1 0
1 0 0 0 1 1 1 0
0 0 1 1 0 0 1 1
0 1 0 1 1 1 1 1

data bits in black
parity bits in blue

  - transmitted: 11100010 10001110 00110011 01011111
  - receiver knows to form received bit string into 4×8 array, then check the row and column parity bits…
  - can **<u>detect</u>** any odd number of bit errors in a row or column, and can detect an even number of bit errors if they're in a single row (using the column parity checks) or in a single column (using the row parity checks); and can **<u>correct</u>** any single bit error

Datalink layer: 2-Dimensional Parity Check (cont.)

- Example (cont.): suppose bit in position (1,3) is received in error (in other words, 1 bit error)

1 1 0 0 0 0 1 0	row 1 parity check fails
1 0 0 0 1 1 1 0	row 2 parity check ok
0 0 1 1 0 0 1 1	row 3 parity check ok
0 1 0 1 1 1 1 1	row 4 parity check ok

column 1 parity check ok
column 2 parity check ok
column 3 parity check fails
column 4 parity check ok
column 5 parity check ok
column 6 parity check ok
column 7 parity check ok
column 8 parity check ok

therefore the receiver can detect that the bit in position (1,3) was in error, so this bit can be corrected

Datalink layer: 2-Dimensional Parity Check (cont.)

- Example (cont.): suppose bits in positions (1,1) and (1,3) are received in error (in other words, 2 bit errors)

0 1 0 0 0 0 1 0	row 1 parity check ok
1 0 0 0 1 1 1 0	row 2 parity check ok
0 0 1 1 0 0 1 1	row 3 parity check ok
0 1 0 1 1 1 1 1	row 4 parity check ok

column 1 parity check fails
column 2 parity check ok
column 3 parity check fails
column 4 parity check ok
column 5 parity check ok
column 6 parity check ok
column 7 parity check ok
column 8 parity check ok

therefore the receiver can detect that bit errors occurred, but it cannot correct them (here, if the bit errors were e.g. in positions (3,1) and (3,3) instead, the receiver parity checks would be the same)

Datalink layer: 2-Dimensional Parity Check (cont.)

- Example (cont.): suppose bits in positions (1,1) and (2,3) are received in error (in other words, 2 bit errors)

0 1 1 0 0 0 1 0	row 1 parity check fails
1 0 1 0 1 1 1 0	row 2 parity check fails
0 0 1 1 0 0 1 1	row 3 parity check ok
0 1 0 1 1 1 1 1	row 4 parity check ok

column 1 parity check fails
column 2 parity check ok
column 3 parity check fails
column 4 parity check ok
column 5 parity check ok
column 6 parity check ok
column 7 parity check ok
column 8 parity check ok

therefore the receiver can detect that bit errors occurred, but it cannot correct them (here, if the bit errors were in positions (1,3) and (2,1) instead, the receiver parity checks would be the same)

Datalink layer: 2-Dimensional Parity Check (cont.)

- Example (cont.): suppose bits in positions (1,1), (1,2) and (1,3) are received in error (in other words, 3 bit errors)

0 0 0 0 0 0 1 0	row 1 parity check fails
1 0 0 0 1 1 1 0	row 2 parity check ok
0 0 1 1 0 0 1 1	row 3 parity check ok
0 1 0 1 1 1 1 1	row 4 parity check ok

column 1 parity check fails
column 2 parity check fails
column 3 parity check fails
column 4 parity check ok
column 5 parity check ok
column 6 parity check ok
column 7 parity check ok
column 8 parity check ok

therefore the receiver can detect that bit errors occurred, but it cannot correct them (here, if the bit errors were e.g. in positions (1,1), (2,2) and (2,3) instead, the receiver parity checks would be the same)

Datalink layer: 2-Dimensional Parity Check (cont.)

- Example (cont.): suppose bits in positions (1,1), (1,3) and (2,3) are received in error (in other words, 3 bit errors)

0 1 0 0 0 0 1 0	row 1 parity check ok
1 0 1 0 1 1 1 0	row 2 parity check fails
0 0 1 1 0 0 1 1	row 3 parity check ok
0 1 0 1 1 1 1 1	row 4 parity check ok

column 1 parity check fails

column 2 parity check ok

column 3 parity check ok

column 4 parity check ok

column 5 parity check ok

column 6 parity check ok

column 7 parity check ok

column 8 parity check ok

therefore the receiver can detect that bit errors occurred, but it "corrects" them incorrectly by only changing the bit in position (2,1) from 1 to 0: this method cannot cope with 3 bit errors in this pattern…

Datalink layer: 2-Dimensional Parity Check (cont.)

- Example (cont.): suppose bits in positions (1,1), (1,3), (2,3) and (4,1) are received in error (in other words, 4 bit errors)

0 1 0 0 0 0 1 0	row 1 parity check ok
1 0 1 0 1 1 1 0	row 2 parity check fails
0 0 1 1 0 0 1 1	row 3 parity check ok
1 1 0 1 1 1 1 1	row 4 parity check fails

column 1 parity check ok
column 2 parity check ok
column 3 parity check ok
column 4 parity check ok
column 5 parity check ok
column 6 parity check ok
column 7 parity check ok
column 8 parity check ok

therefore the receiver can detect that bit errors occurred, but it cannot correct them (here, if the bit errors were e.g. in positions (1,1), (1,3), (2,1) and (4,3) instead, the receiver parity checks would be the same)

Datalink layer: 2-Dimensional Parity Check (cont.)

- Example (cont.): suppose bits in positions (1,1), (1,3), (2,1) and (2,3) are received in error (in other words, 4 bit errors)

0 1 0 0 0 0 1 0	row 1 parity check ok
0 0 1 0 1 1 1 0	row 2 parity check ok
0 0 1 1 0 0 1 1	row 3 parity check ok
0 1 0 1 1 1 1 1	row 4 parity check ok

column 1 parity check ok
column 2 parity check ok
column 3 parity check ok
column 4 parity check ok
column 5 parity check ok
column 6 parity check ok
column 7 parity check ok
column 8 parity check ok

therefore the receiver <u>cannot detect</u> that an even number of bit errors in this "rectangular" pattern occurred

(think about what happens if an even number of "rows" are hit by a burst error and have all their bits flipped…)

Datalink layer: solution 5: General Parity Check

- the **Hamming distance** between 2 codewords W_1 and W_2 : $d(W_1, W_2)$ = *number of bit positions in which W_1 and W_2 differ*

Example: W_1 =10001001, W_2 =10110001, then $d(W_1, W_2) = 3$

- for any error detection/correction scheme, we can define the minimum Hamming distance (or “minimum distance”) of the scheme as the ***smallest number*** of bit errors that changes one valid codeword into another
- remember: m data bits, r check bits, n = m+r total bits
- all 2^m possible data strings are (usually) valid; but since the check bits are determined from the data, not all 2^n possible codewords are valid

- if the way of computing the check bits is known, a list of all the valid codewords can be compiled and stored at the receiver
- when a word W is received, the receiver finds the ***closest valid codeword to W (in Hamming distance)*** and takes this codeword as the transmitted codeword

Datalink layer: General Parity Check (cont.)

- if the minimum distance of an error-handling scheme is D, this scheme can **detect** any combination of ≤ D-1 bit errors and **correct** any combination of strictly less than D/2 bit errors

- alternatively: if you want to detect B bit errors, use a scheme with minimum distance at least B+1; if you want to correct B bit errors, use a scheme with minimum distance at least 2B+1

- Examples
    - single parity check scheme has D=2, and we saw that it can ***detect*** any single bit error but ***cannot correct*** any bit errors

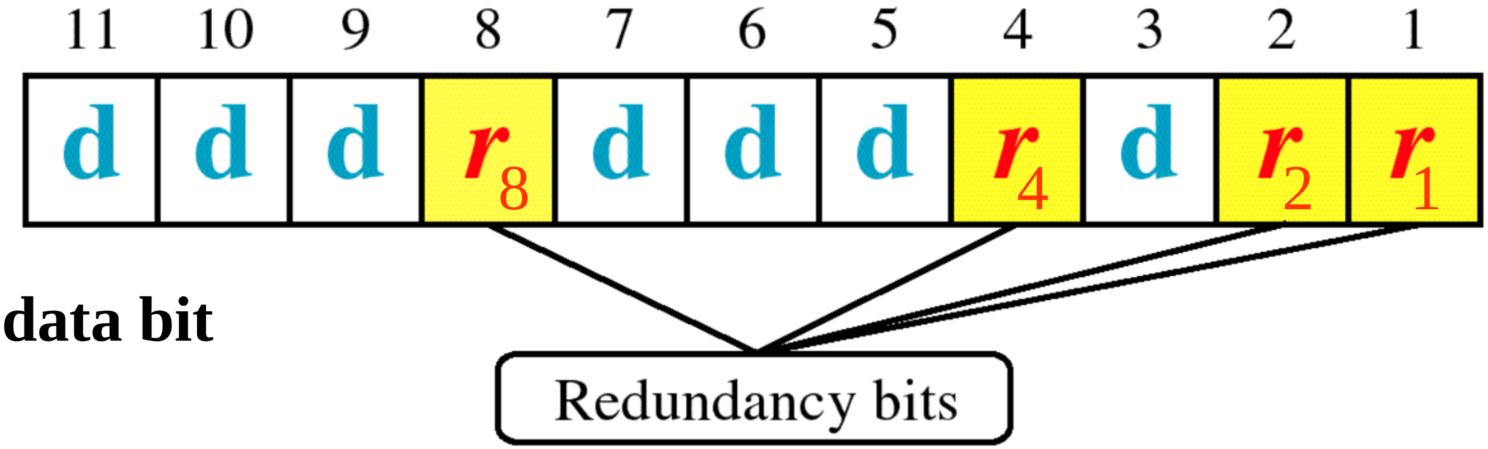
    - 2-dimensional parity check scheme has D=4, and we saw that it can ***detect*** any combination of ≤ 3 bit errors and can only ***correct*** any single bit error

Datalink layer: General Parity Check (cont.)

- Another Example: suppose only 4 valid codewords are 00000 00000, 00000 11111, 11111 00000, and 11111 11111
  - minimum distance is D=5, so any combination of ≤ 4 bit errors can be detected and any combination of ≤ 2 bit errors can be corrected, but 3 bit errors can't be properly corrected
  - if 00000 00000 transmitted, W=00000 00011 received: receiver knows received word is not a valid codeword ***(errors detected)*** and computes Hamming distance between W and all valid codewords: d(W, C_1)=2, d(W, C_2)=3, d(W, C_3)=7, and d(W, C_4)=8 ⇒ receiver takes C_1=00000 00000 as transmitted codeword ***(errors corrected)***
  - if 00000 00000 transmitted, X=00000 00111 received: receiver knows received word is not a valid codeword ***(errors detected)*** and computes Hamming distance between X and all valid codewords: d(X, C_1)=3, d(X, C_2)=2, d(X, C_3)=8, and d(X, C_4)=7 ⇒ receiver takes C_2=00000 11111 as transmitted codeword ***(in this case, error correction fails)***

Datalink layer: solution 6: Hamming code

- based on calculation of the ***minimum number of redundancy bits*** needed to correct any single bit error in the data
    - Result: with 7 data bits, need minimum of 4 redundancy bits
- redundancy bits in the Hamming code are placed in the codeword bit positions that are a power of 2
    - each redundancy bit is the parity bit for a different combination of data bits
    - each data bit may be included in more than one parity check



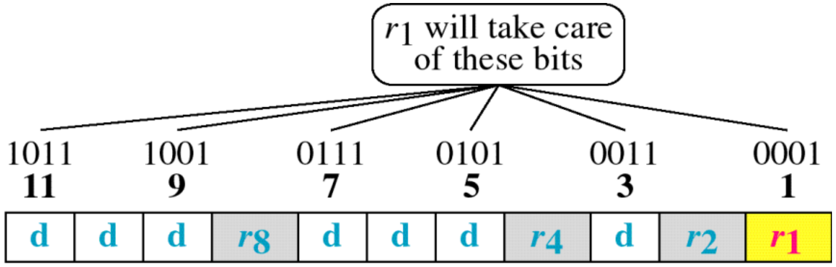
d: data bit

Datalink layer: <u>solution 6:</u> Hamming code (cont.)

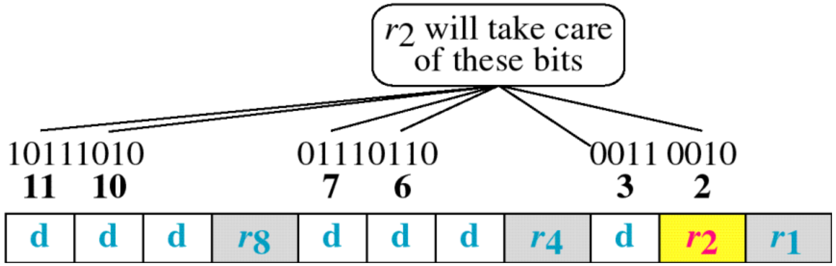
- which data bits does each redundancy bit check ?
  - r_1 checks data in positions whose binary representations have a 1 in the rightmost position
  - r_2 checks data in positions whose binary representations have a 1 in the 2nd-rightmost position
  - r_4 checks data in positions whose binary representations have a 1 in the 3rd-rightmost position
  - r_8 checks data in positions whose binary representations have a 1 in the 4th-rightmost (in other words, leftmost) position

  - equivalently: by writing a bit's position as a sum of powers of 2, can tell which redundancy bit(s) check this bit
    - bit in position 11 is checked by r_1 , r_2 , and r_8 because 11=1+2+8;
    - bit in position 6 is checked by r_2 and r_4 because 6=2+4; etc.

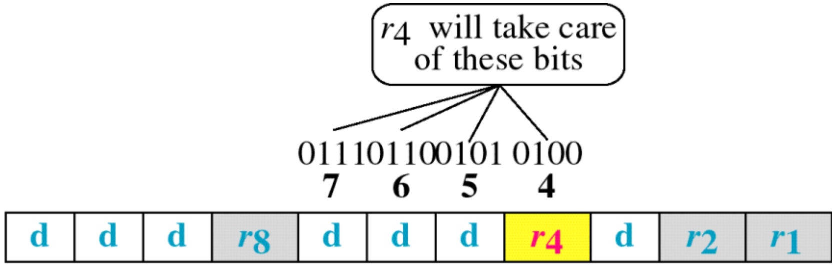
Datalink layer: solution 6: Hamming code (cont.)



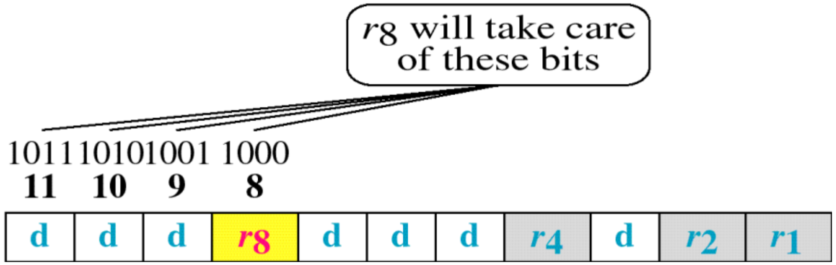
r_1 will take care of these bits
1011 1001 0111 0101 0011 0001
11 9 7 5 3 1
d d d r_8 d d d r_4 d r_2 r_1



r_2 will take care of these bits
1011 1010 0111 0110 0011 0010
11 10 7 6 3 2
d d d r_8 d d d r_4 d r_2 r_1



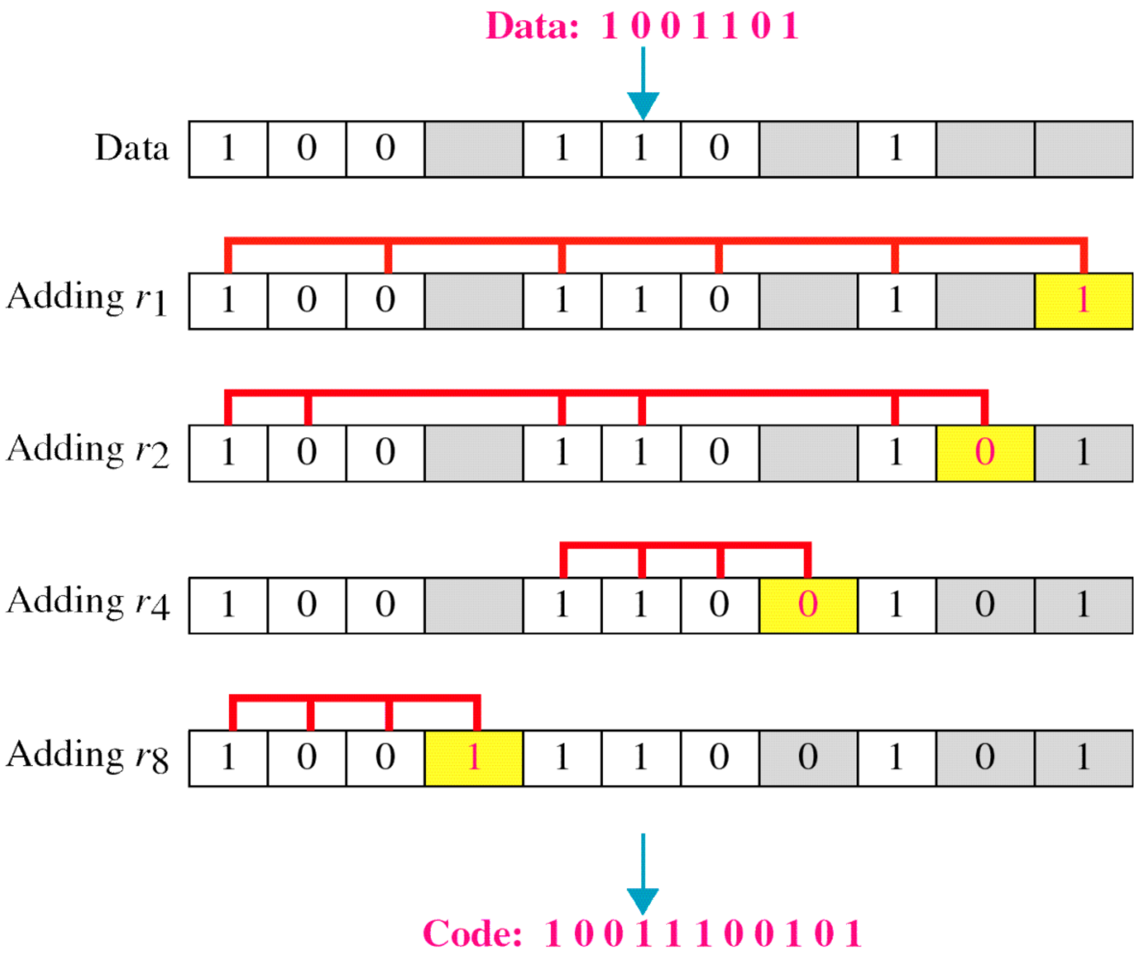
r_4 will take care of these bits
0111 0110 0101 0100
7 6 5 4
d d d r_8 d d d r_4 d r_2 r_1



r_8 will take care of these bits
1011 1010 1001 1000
11 10 9 8
d d d r_8 d d d r_4 d r_2 r_1

Datalink layer: solution 6: Hamming code (cont.)

- Example of redundancy bit calculations:



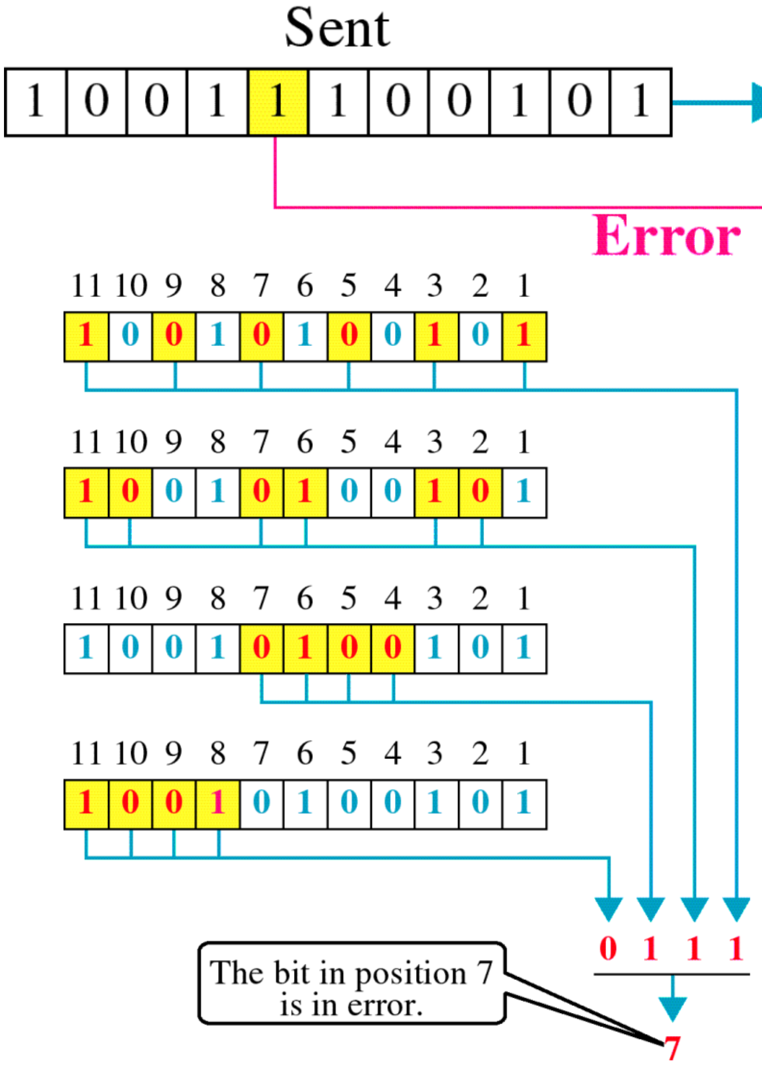
Data: 1 0 0 1 1 0 1

Data	1	0	0		1	1	0		1		
Adding r_1	1	0	0		1	1	0		1		1
Adding r_2	1	0	0		1	1	0		1	0	1
Adding r_4	1	0	0		1	1	0	0	1	0	1
Adding r_8	1	0	0	1	1	1	0	0	1	0	1

Code: 1 0 0 1 1 1 0 0 1 0 1

Datalink layer: solution 6: Hamming code (cont.)

- suppose that the bit in position 7 is received in error:



The bit in position 7 is in error.

receiver computes "new" parity bits for each redundancy bit position; if these "new" parity bits are not all 0, the binary value of the receiver's "new" parity bits tells the position of the single bit error, which can therefore be corrected

receiver's "new" parity bits

Datalink layer: <u>solution 6:</u> Hamming code (cont.)

- if the transmitted codeword is received error-free, the “new” parity bits the receiver computes will all be 0 (why ?) ⇒ the receiver knows no bit errors occurred

- this simple form of Hamming code can be used to provide some protection against burst errors, by transmitting 1st bit from every codeword to be transmitted, then 2nd bit from every one of these codewords, and so on…In some cases, burst errors can be corrected

- however, for better error protection it is necessary to add more redundancy bits, and the number of extra redundancy bits rises dramatically from that required to correct single bit errors

- in cases where better **error detection** is required, either ***checksum*** or ***CRC*** (Cyclic Redundancy Check) is used. If these mechanisms detect errors, re-transmission is requested (where it is feasible) – more efficient than error correction

Mathematical Notation

- The probability of bit error, p_b, at the Physical transmission layer leads on to a probability of frame error, p.
- This depends on the number of bits of information in the frame, l, and the number of bits of header in the frame, l' .
- To get a good frame all the bits have to be good.

=> $(1 - p) = (1 - p_b)^{(l+l')}$

=> $p = 1 - (1 - p_b)^{(l+l')}$

=> $p = (l+l')\, p_b$, iff $p << 1$

Example

Note:

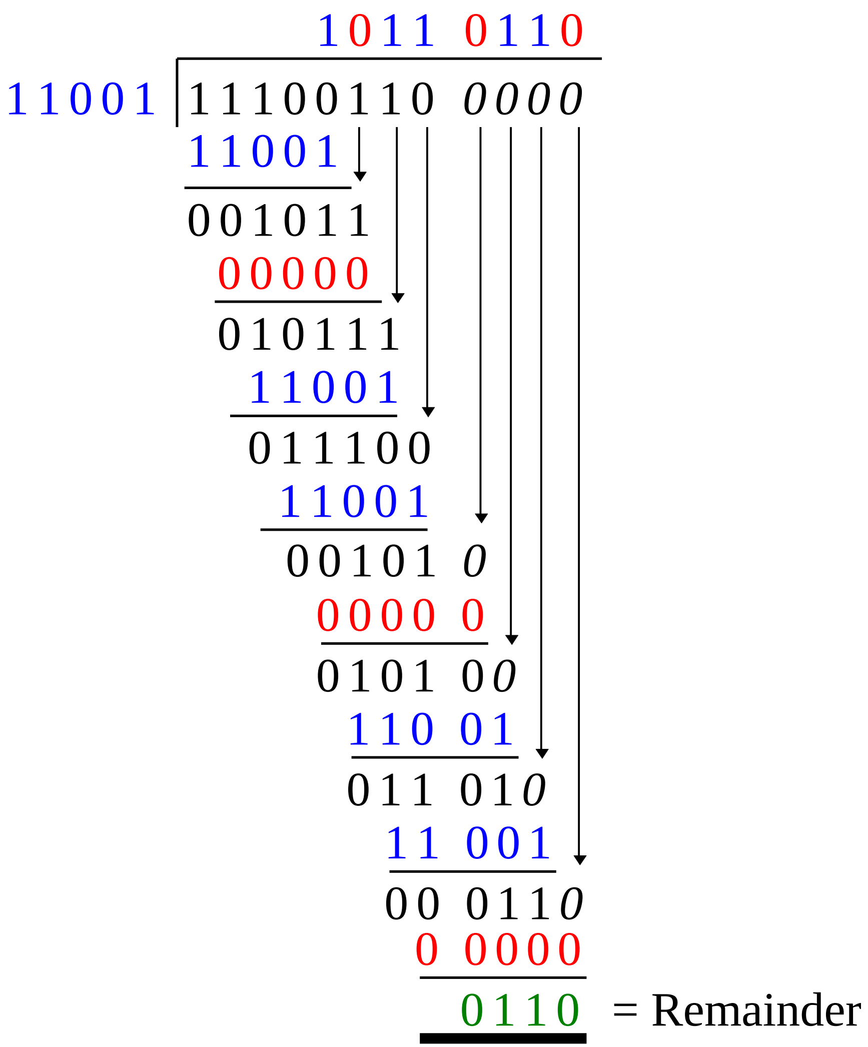
M: 11100110
(k=8)

M with zeros :
11100110 *0000*
(n=4)

G: 11001

CRC: 0110

Transmitted bits:
111001100110



```
                   1011 0110
          11001 | 11100110 0000
                  11001
                  001011
                   00000
                   010111
                    11001
                    011100
                     11001
                     00101 0
                      0000 0
                      0101 00
                       110 01
                       011 010
                        11 001
                        00 0110
                         0 0000
                           0110  = Remainder
```

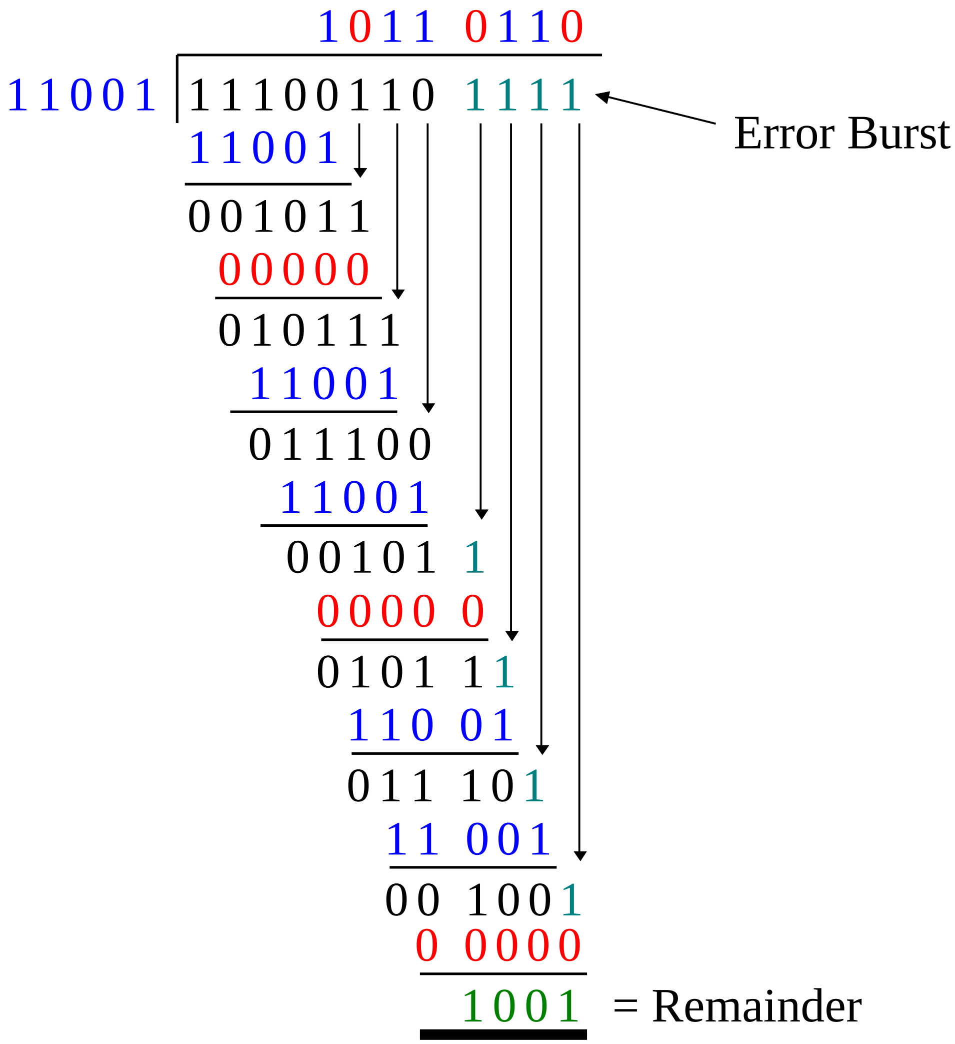
Cyclic Redundancy Check

- Let M be the frame contents that are to be protected, usually from everything except the flags and the bit stuffing. Assume it to be k bits long.
- Let the CRC be n bits long.
- Let G, the Generator Polynomial, be n+1 bits long, this is know in advance to both the transmitter and the receiver.
- Divide M (frame) with n zeros appended to it, by G (generator) and the remainder is the CRC.
- This CRC, which is n bits long, is sent along with the frame to the receiver.

CRC Transmit & Receive

- You transmit the bit sequence, M, appended by the CRC, and any other unprotected bits like the flags.
- The receiver takes the unprotected bits off, and then divides the k+n bits of the bit stream by the Generator G.
- If the remainder is zero then there are no detected errors. A non-zero remainder will indicate that errors have been detected.
- Look at the case where an error burst causes the CRC bits to be received incorrectly.

Example of CRC Check



```
           1011 0110
11001 | 11100110 1111  ← Error Burst
        11001
        001011
         00000
         010111
          11001
          011100
           11001
           00101 1
            0000 0
            0101 11
             110 01
             011 101
              11 001
              00 1001
               0 0000
                 1001  = Remainder
```

Remainder ≠ 0, ⇒ Error Detected

CRC Operation

- Only an error pattern that is identical, or has a factor identical to the generator polynomial, will produce the same CRC bits remaining undetected.
- For this reason a polynomial which is prime, in the modulo-2 sense, is chosen as the generator.
- The standard way of representing a generator polynomial is to show those bit positions in the number that are binary 1 as powers of X.
- A standard 16-bit CRC generator polynomials is:
- CRC-16 = X16 + X15 + X2 + 1
  = 1 1000 0000 0000 0101