4. Error correction and link control

Contents

a. Types of errors
b. Error detection and correction
c. Flow control
d. Error control

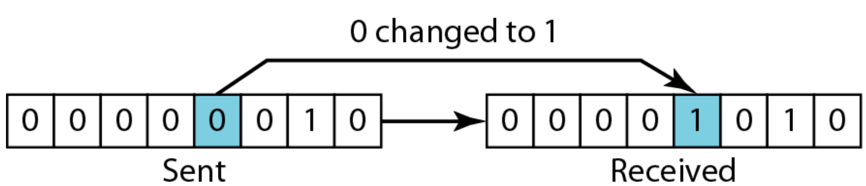
a. Types of errors

Data can be corrupted during transmission.

Some applications require that errors be detected and corrected.

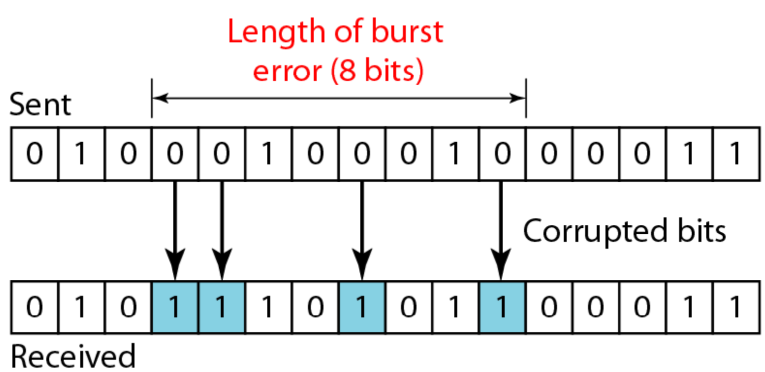
In a single-bit error, only 1 bit in the data unit has changed.

Single-bit error



A burst error means that 2 or more bits in the data unit have changed.

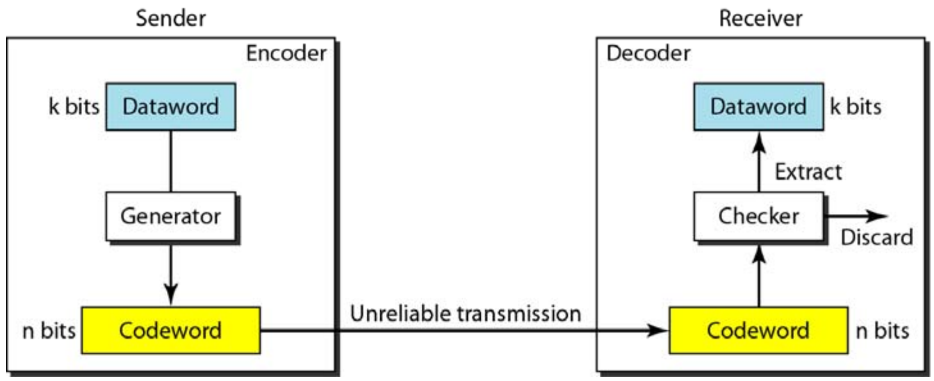
Burst error of length 8



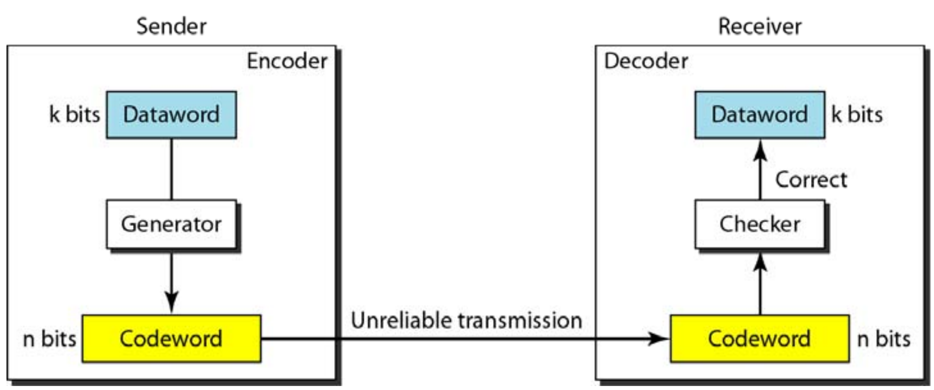
To detect or correct errors, we need to send extra (redundant) bits with data.

b. Error detection and correction

Process of error detection in block coding



Structure of encoder and decoder in error correction



c. Flow control

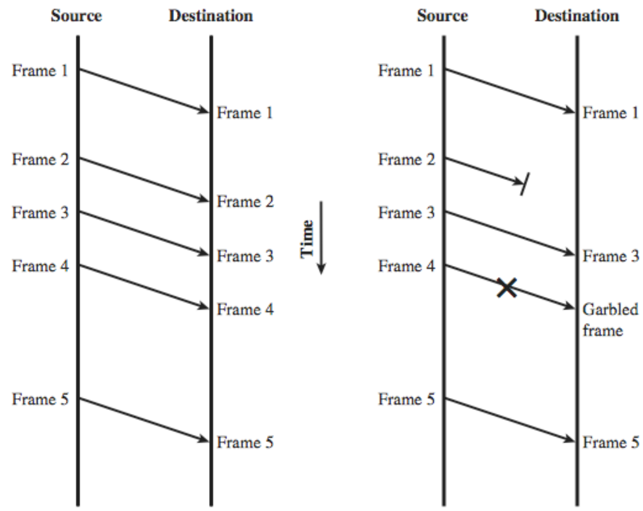
Flow Control

Ensure sending entity does not overwhelm receiving entity by preventing buffer overflow influenced by:

- transmission time
  - time taken to emit all bits into medium
- propagation time
  - time for a bit to traverse the link

assume here no errors but varying delays

Model of Frame Transmission



(a) Error-free transmission

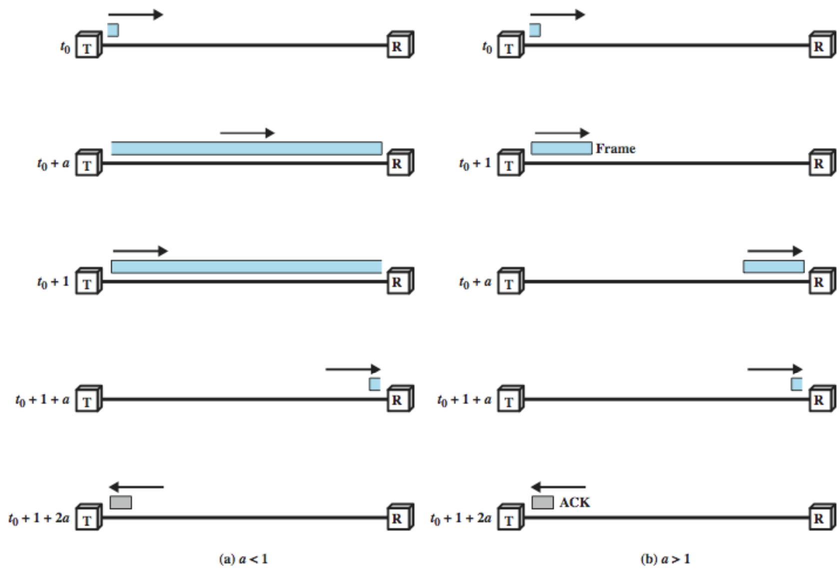
(b) Transmission with losses and errors

Stop and Wait

- source transmits frame
- destination receives frame and replies with acknowledgement (ACK)
- source waits for ACK before sending next
- destination can stop flow by not send ACK

- This procedure works well for a few large frames
- Stop and wait becomes inadequate if large block of data is split into small frames

Stop and Wait Link Utilization



Sliding Windows Flow Control

- Allows multiple numbered frames to be in transit
  - receiver has buffer W long
  - transmitter sends up to W frames without ACK
- ACK includes number of next frame expected
- Sequence number is bounded by size of field (k)
  - frames are numbered modulo 2^k
  - giving max window size of up to $2^k - 1$
- Receiver can ack frames without permitting further transmission (Receive Not Ready)
- Must send a normal acknowledge to resume
- If have full-duplex link, can piggyback ACKs

d. Error control

Error Control

1. Detection and correction of errors such as:
   - lost frames
   - damaged frames

- Common techniques use:
  - error detection
  - positive acknowledgment
  - retransmission after timeout
  - negative acknowledgement & retransmission

Automatic Repeat Request (ARQ)

Collective name for such error control mechanisms, including:

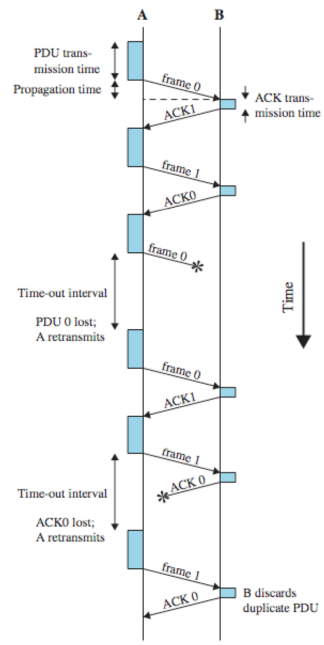
- stop and wait
- go back N
- selective reject (selective retransmission)

Stop and Wait

- Source transmits single frame
  - wait for ACK
- if received frame damaged, discard it
  - transmitter has timeout
  - if no ACK within timeout, retransmit
- if ACK damaged,transmitter will not recognize it
  - transmitter will retransmit
  - receive gets two copies of frame
  - use alternate numbering and ACK0 / ACK1

Stop and Wait

- Example with both types of errors
- pros and cons
  - simple
  - inefficient



Go Back N

- Based on sliding window
  - if no error, ACK as usual
- Use window to control number of outstanding frames
- If error, reply with rejection
  - discard that frame and all future frames until error frame received correctly
  - transmitter must go back and retransmit that frame and all subsequent frames

Go Back N - Handling

- Damaged Frame
  - error in frame *i* so receiver rejects frame *i*
  - transmitter retransmits frames from frame *i*
- Lost Frame
  - frame *i* lost and either
    - transmitter sends *i+1* and receiver gets frame *i+1* out of seq and rejects frame *i*
    - or transmitter times out and send ACK with P bit set which receiver responds to with ACK *i*
  - transmitter then retransmits frames from *i*

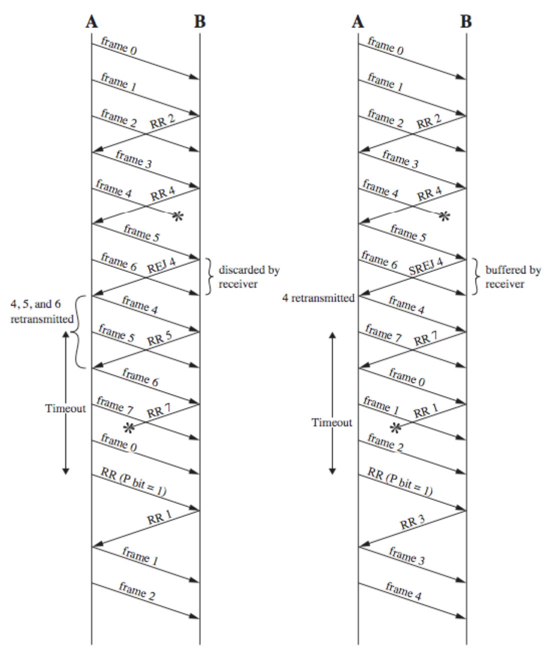
Go Back N - Handling

- Damaged Acknowledgement
  - receiver gets frame *i*, sends ack (*i+1*) which is lost
  - acks are cumulative, so next ack (*i+n*) may arrive before transmitter times out on frame *i*
  - if transmitter times out, it sends ack with P bit set
  - can be repeated a number of times before a reset procedure is initiated
- Damaged Rejection
  - reject for damaged frame is lost
  - handled as for lost frame when transmitter times out

Selective Reject

- Also called selective retransmission
- Only rejected frames are retransmitted
- Subsequent frames are accepted by the receiver and buffered
- Minimizes retransmission
- Receiver must maintain large enough buffer
- More complex logic in transmitter
- Hence less widely used
- Useful for satellite links with long propagation delays

Go Back N vs Selective Reject



(a) Go-back-N ARQ (b) Selective-reject ARQ

Error control

There are several error detection methods, and their application depends on the type of errors encountered on the line.

There are random 1-bit or 2-bit errors, and there are burst errors.

Error control

- **PARITY**
- **CRC**
- **VRC**
- **LRC**
- **CHECKSUM**

Error control by parity

This method identifies only errors in asynchronous transmissions and byte-oriented protocols.

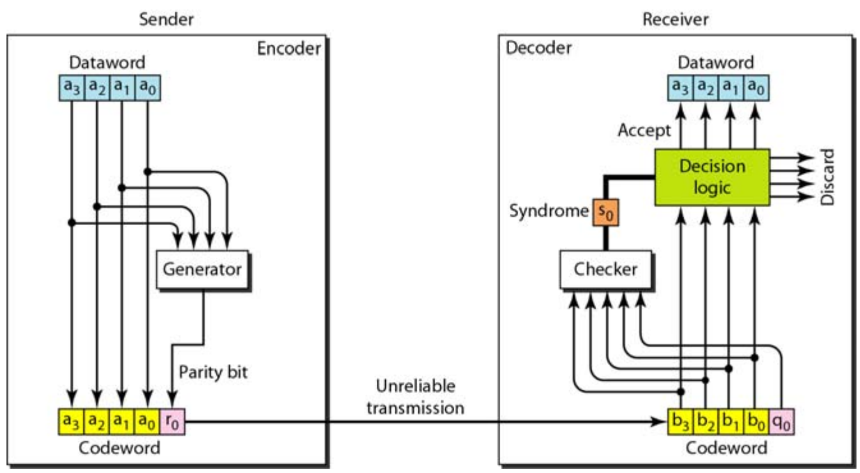
In this method, an extra bit (the parity bit) is added to each character before being sent.

The receiver performs an opposite operation, and if:

- The result is the same, it is assumed that there has been no error.
- If it differs, it is assumed that there has been an error.

However, if there are two errors, this character may go through with these errors undetected.

Encoder and decoder for simple parity-check code



A simple parity-check code can detect an odd number of errors.

Checksum

When character blocks are sent, there is an increased possibility that a character and, thus, the block, may contain an error.

In this method, characters are placed in a 2D block. A parity bit is added to each character, using the parity error check method.

A parity bit is also added for each bit position in all characters.

In other words, an additional character is created, in which its *i-th* bit is the parity bit for the *i-th* bits of the characters.

This may be expressed with an OR-EX operation.

Checksum

So the parity bit at the end of each character is the row parity bit and is:

$$R_j = b_{1j} \oplus b_{2j} \oplus .... b_{nj}$$

where:

R_j = parity bit of the *j-th* character

B_{ij} = *i-th* bit of the *j-th* character

n = number of bits in a character

Parity bits generated at the end of each character are known as Vertical Redundancy Check – VRC.

Checksum

There is a formula to generate the parity-check character:

$$Ci = b_{i1} \oplus b_{i2} \oplus \; ... \oplus b_{in}$$

where:

C_i = *i-th* bit of the parity-check character

m = number of characters in a frame

This character is called Longitudinal Redundancy Check (LRC)

Checksum

	Bit 1	Bit 2		Bit n	Bit de paridad
Carácter 1	b_{11}	b_{21}	…	b_{n1}	R_1
Carácter 2	b_{12}	b_{22}	…	b_{n2}	R_2
…	…	…	…	…	…
Carácter m	b_{1m}	b_{2m}	…	b_{nm}	R_m
Carácter de control de paridad LRC	C_1	C_2	…	C_n	C_{n+1}

VRC

Cyclic redundancy check - CRC

A more powerful error control technique is the Cyclic Redundancy Check (CRC).

Given a k-bit-long message or frame, the sender generates a sequence of n bits, known as Frame Check Sequence (FCS), in such a way that the resulting frame, consisting of k + n bits, is exactly divisible by a predetermined number.

Cyclic redundancy check - CRC

The receiver divides the incoming frame by the same predetermined number, and, if there is no remainder, it is assumed that the frame has arrived without errors.

This procedure may be applied in several ways: using modulo-2 arithmetic, polynomials and OR-EX gates with displacement records.

Cyclic redundancy check - CRC

We will work with binary numbers and modulo-2 arithmetic.

Modulo-2 arithmetic uses the binary sum with no carries as an OR-EX gate operation.

For example:

$$\begin{array}{rr} & 1\,1\,1\,1 \\ + & \underline{1\,0\,1\,0} \\ & 0\,1\,0\,1 \end{array} \qquad \begin{array}{rr} & 1\,1\,0\,0\,1 \\ \times & \underline{1\,1} \\ & 1\,1\,0\,0\,1 \\ & \underline{1\,1\,0\,0\,1\;\;} \\ & 1\,0\,1\,0\,1\,1 \end{array}$$

Cyclic redundancy check - CRC

Now we define:

T = (k+n) bit frame to be sent with $n < k$

M = k-bit message. They are the first bits of frame T

F = Frame Check Sequence (FCS). They are the last bits of frame T

P = n+1 bit pattern. This is the aforementioned predetermined divisor.

Cyclic redundancy check - CRC

The desired outcome is that the division between frame T to be sent and the n + 1 bit pattern will have no remainder.

So:

$$T = 2^n M + F \qquad (1)$$

That is, by multiplying M by 2^n, n bits will be displaced to the left and the displaced bits will be completed with 0’s.

The addition of F gives us the concatenation of M and F, which results in T.

Cyclic redundancy check - CRC

Now we want T to be exactly divisible by P. If we divide 2^nM by P:

$$\frac{2^nM}{P} = Q + \frac{R}{P} \qquad (2)$$

we obtain a quotient and a remainder.

Since it is a binary division, the remainder is always one bit less than the divisor.

We use this remainder as our FCS.

So:

$$T = 2^nM + R \qquad (3)$$

Cyclic redundancy check - CRC

Now we will show that R meets our condition. Let us consider:

$$\frac{T}{P} = \frac{2^nM + R}{P} = \frac{2^nM}{P} + \frac{R}{P} \qquad (4)$$

By substituting equation (2) in (4), we have that:

$$\frac{T}{P} = Q + \frac{R}{P} + \frac{R}{P} \qquad (5)$$

Cyclic redundancy check - CRC

Any binary number added to itself in modulo 2 is equal to zero. So:

$$\frac{T}{P} = Q + \frac{R + R}{P} = Q$$

It thus demonstrates that there is no remainder, so T is exactly divisible by P.

We thus generate the FCS.

$2^n M$ is divided by P and the remainder is used as the FCS.

In the receiving side, the receiver will divide T by P, and if there is no remainder, it means that the frame has been received with no errors.

Example of cyclic redundancy check

Given:

Message M = 1010001101 (10 bits)

Pattern P = 110101 (6 bits)

FCS (R) = to be calculated (5 bits)

The message is multiplied by 2^5, resulting in:

$101000110100000 = 2^n M = 2^5 M$

Example of cyclic redundancy check

This product 2^5M is divided by P:

```
2^5M     101000110100000 | 110101        P
         110101            1101010110    Q
          111011
          110101
             111010
             110101
                111110
                110101
                   101100
                   110101
                     110010
                     110101
                         1110            R
```

Example of cyclic redundancy check

The remainder is added to the value of 2^nM to obtain:

T = 101000110101110,

which is sent.

If there are no errors, the receiver will receive T intact.

Example of cyclic redundancy check

The frame received is divided by P:

```
101000110101110 | 110101
110101            110101
 111011
 110101
   111010
   110101
     111110
     110101
       101111
       110101
        110101
        110101
            00
```

Cyclic redundancy check - CRC

Since there is no remainder, it is assumed that the frame received has no errors.

There are 4 versions of CRC broadly used:

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

Block line codes

A block code requires that a message be partitioned in bit blocks.

A data word is a block that has k bits.

Therefore, the number of data words is: 2^k

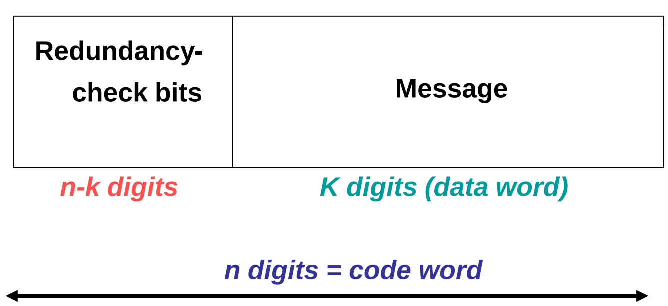
Data words are coded within code words.

Each code word is a block of **n** bits.

The possible number of code words is 2^n, but only 2^k are used and transmitted.

The additional n-k bits are called parity-check bits.

Systematic code words



Code rate

The code rate r_c is defined as:

$$r_c = \frac{k}{n}$$

The code annotation is (n, k).

For example, a code containing a 4-bit data word in a 7-bit code word will be a code:

(k = 4, n =7,)

Its code rate will be: $r_c = 4/7$

Repetition code

A repetition code has some of the properties of the block code.

In a repetition code, each bit is a data word:

K = 1

For a redundancy coding **n**, the encoder output will be **n** bits identical to the input bit.

Considering the case of: **n** = 3

If 1 is sent, the output is the code word 111

If 0 is sent, the code word will be 000

Hamming distance

The Hamming distance between two code words is the number of positions in which these words differ.

The smaller the distance between two code words, the better the code.

In this code, the minimum distance is 2.

Dataword	Modulo-2 addition of dataword	Codeword
000	0	0000
001	1	0011
010	1	0101
011	0	0110
100	1	1001
101	0	1010
110	0	1100
111	1	1111

Matrices used in codes

A k-bit data word is represented by a row vector **d**.

For example, the sixth data word in the table is row vector

d6 = [101] and its code word is row vector **c6** = [1010]

Dataword	Modulo-2 addition of dataword	Codeword
000	0	0000
001	1	0011
010	1	0101
011	0	0110
100	1	1001
101	0	1010
110	0	1100
111	1	1111

Matrices used in codes

In general, a code word **c** is generated by multiplying the data word **d** by a generating matrix **G**, where:

c = **dG**

An example of a generating matrix for code (7,4) is:

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Generation of a code word

The first 4 columns form an identity matrix.

This sub-matrix allows the first 4 bits of the code word to correspond to the data word.

The remaining bits generate the parity bits from the data bits.

Generation of a code word

As an example, we will generate the code word from the data word [1010].

$$C = \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Sub-matrix P and its transposed

The last 3 columns form sub-matrix **P** (parity bits):

$$P = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Where, taking its transposed, P^T:

$$P^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

H parity-check matrix

Then we form matrix H, inserting the identity matrix in matrix P^T.

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

The number of rows in H is equal to the number of n-k parity bits, and the number of columns is n.

Thus, the parity-check matrix is (n-k, n).

A fundamental property of code matrices is the product:

$$GH^T = 0$$

H parity-check matrix

When a code word is received, it may be checked by multiplying it by $\mathbf{H}^\mathbf{T}$.

The product $\mathbf{cH}^\mathbf{T}$ must be equal to **0**.

This is because **c** = **dG**.

If we multiply both terms by $\mathbf{H}^\mathbf{T}$, we get: $\mathbf{cH}^\mathbf{T} = \mathbf{dGH}^\mathbf{T}$, which is equal to **0**

If the result is not zero, it means that an error has been received.

In general terms, the product $c\mathbf{H}^\mathbf{T}$ provides what is known as a syndrome, and is represented by **s**:

$$\mathbf{s} = c\mathbf{H}^\mathbf{T}$$

The s-syndrome

In general, if a code word $\mathbf{c_R}$ is received, and the code word sent is $\mathbf{c_T}$ and the error vector is **e**, using the modulo-2 addition:

$$\mathbf{c_R} = \mathbf{c_T} + \mathbf{e}$$

By substituting **c** in the $\mathbf{s} = \mathbf{cH^T}$ equation, we have:

$\mathbf{s} = (\mathbf{c_T} + \mathbf{e})\, \mathbf{H^T} = \mathbf{c_T}\, \mathbf{H^T} + \mathbf{e}\, \mathbf{H^T}$ but $\mathbf{c_T}\, \mathbf{H^T} = 0$, so:

$$\mathbf{s} = \mathbf{eH^T}$$

This result shows that the syndrome depends only on the vector error and is independent from the code word sent.

Since the error vector has n bits, it is possible to get 2^n error vectors.

The s-syndrome

The syndrome has only n-k bits (determined by the number of rows in the H matrix), which gives us 2^{n-k} syndromes.

Since one of these is the zero syndrome, the number of errors that can be detected is 2^{n-k} -1.

In practice, decoders are designed to correct the most likely errors.

Example: error of a single bit.

The received syndrome is compared to the values in the tabulation table of known error patterns, and the most likely error is found. This is known as maximum likelihood decoding.

Example of a single-error correction

Let us suppose that the code word [1010010] is sent. But an error occurs in the fifth bit (from the left), so that the code word received is [1010110]. Applying the equation: **s** = c**H**^{T}

$$s = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$s = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

The syndrome [100] corresponds to the fifth row of matrix H, indicating an error in the fifth bit. This bit is changed and the error is corrected.

Cyclic codes

These codes are a sub-class of block codes. Their property is that a cyclic displacement of a code word is also a code word.

Example:

A code word is $\{c_1\ c_2\ c_3\ c_4\ c_5\ c_6\ c_7\}$, then the word $\{c_2\ c_3\ c_4\ c_5\ c_6\ c_7\ c_1\}$ is also a code word.

Cyclic codes are implemented with displacement records and are used in satellite transmission.

Since every code is an invention, codes are named after their inventor.

Cyclic codes and linear codes

- Cyclic codes:
  - Hamming codes
  - BCH (Bose, Chaudhuri and Hocquenghen) code
  - R-S (Reed-Solomon) codes
- Linear codes:
  - Convolution codes
    - Decoding of convolution codes
- Interleaving
- Concatenated codes

Cyclic codes: Hamming codes

It is defined that for an integer m greater or equal to 2, the values of k and n are related as follows:

$$n = 2^m - 1 \quad \text{and} \quad k = n - m$$

As the code rate $r_c = k/n$ approaches 1, m increases, making it more efficient. However, only one error can be corrected with this type of code. Some combinations that are allowed are:

m	n	k
2	3	1
3	7	4
4	15	11
5	31	26
6	63	57
7	127	120

Cyclic code: BCH (Bose, Chaudhuri and Hocquenghen)

These codes correct up to t errors, and m can be any positive integer. The allowed values include:

$n = 2^m-1$ and k greater/equal = $(n – mt)$

Integers m and t are arbitrary, which gives flexibility to the code designer.

n	k	t
7	4	1
15	11	1
15	7	2
15	5	3
31	26	1
31	21	2
31	16	3
31	11	5
31	6	7

Cyclic codes: R-S (Reed-Solomon) codes

They are used with burst errors, that is, errors that are grouped together.

With these codes, instead of coding directly in bits, bits are first grouped in symbols.

Then, data words and code words are coded in these symbols.

Errors affecting a group of bits are more likely to affect only one symbol, which may be corrected by the R-S code.

Forward error correction - FEC

Forward Error Correction (FEC) is a type of error correction mechanism that allows for correction of the error in the receiver, with no retransmission of the original information.

It is used in no-return or real-time systems where it is not possible to wait for the retransmission to show the data.

Forward error correction - Operation

The possibility of correcting errors is obtained by adding some redundancy bits to the original message.

The source sends the data sequence to the encoder, which is tasked with adding the redundancy bits.

The so-called code word is obtained at the output of the encoder.

This code word is sent to the receiver, who, using the appropriate decoder and the error correction algorithms, will obtain the original data sequence.

Main types of FEC coding

- Convolution codes

Convolution codes

Bits are coded as they arrive to the encoder. It should be noted that the encoding of one bit is affected by the encoding of its predecessors.

The decoding for this type of code is complex and a large amount of memory is required to estimate the most likely data sequence for the bits received.

The Viterbi algorithm is currently used for decoding this type of code because of its high efficiency in the use of resources.

Linear codes: Convolution codes

A convolution encoder consists of:

– A displacement registry that temporarily stores and permits the displacement of incoming bits; and
– A circuitry of OR-Ex gates that will generate a coded output from the bits stored in the displacement registry.

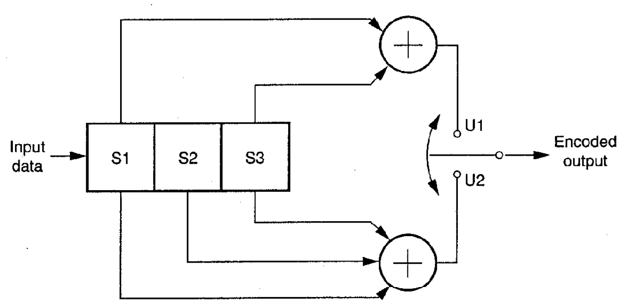
In general, k data bits can be displaced in the registry at the same time, and n code bits will be generated.

Example: k = 1 and n = 2, resulting in a code r_c=1/2.

The registry starts loaded with 0-bits.

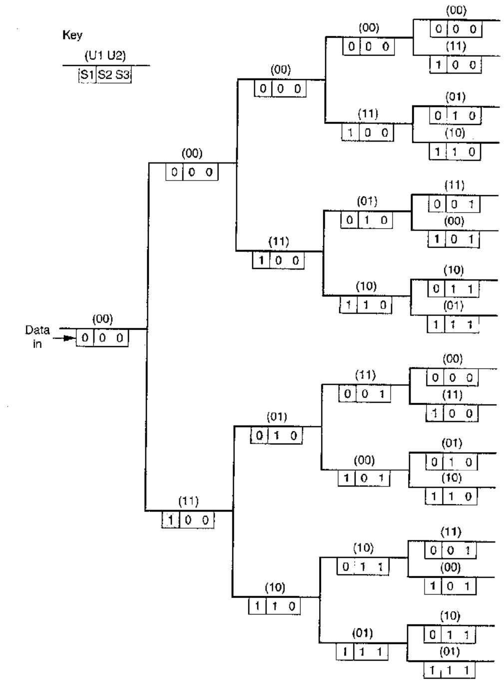
The input is R_b bits.

Convolution encoder with an r_c = 1/2



Encoder tree diagram
r_c = ½

Input: 1001



Encoder tree diagram
r_c = ½

Input: 1001

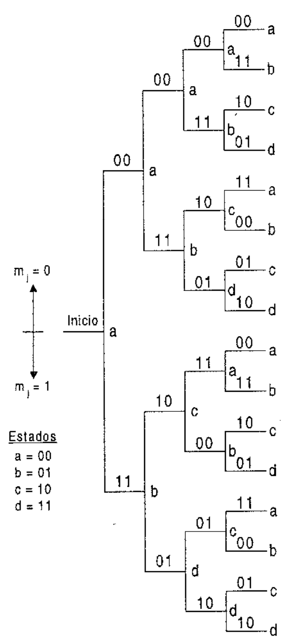


Figura B.3. Diagrama de árbol para el codificador (2, 1, 2)

Encoder trellis diagram
$r_c = ½$

Input: 1001

(n, k, L)
(2, 1, 2)

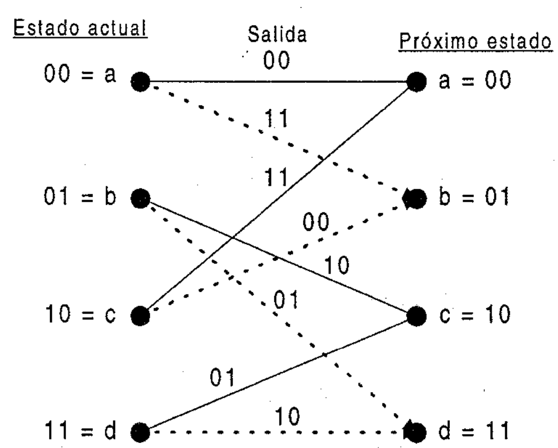
L = Number of status bits

k = data bits

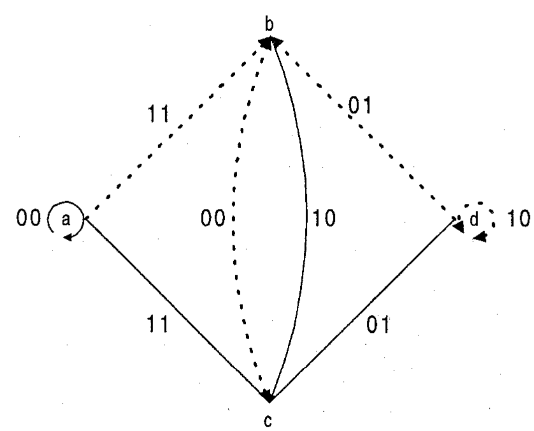
n = code bits

——— = 0

.... = 1



Encoder trellis diagram r_c = ½ Input: 1001 (n, k, L) (2, 1, 2)
L = Number of status bits



——— = 0

.... = 1

Entrada	1	1	0	1	1	1	0	0	1	0	0	0
Estado	a	b	d	c	b	d	d	c	a	b	c	a	a
Salida	11	01	01	00	01	10	01	11	11	10	11	00

Decoding

There are three types of decoding:

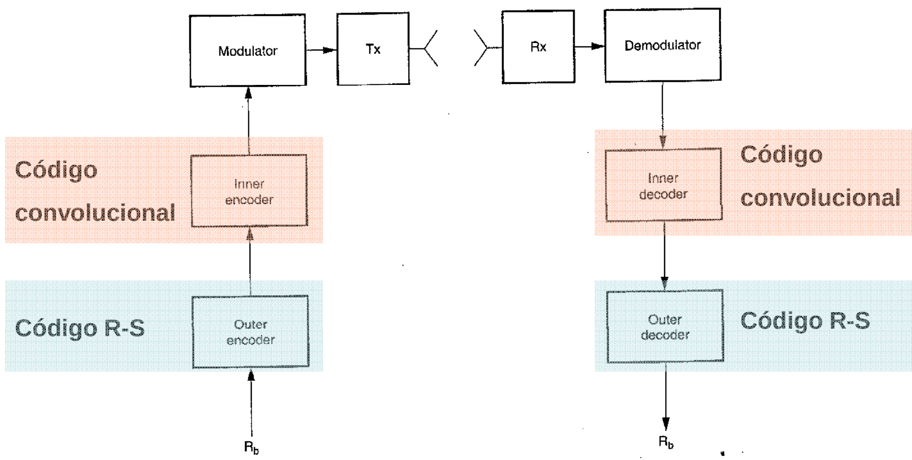
- Viterbi decoding (maximum likelihood)
- Sequential decoding
- Feedback decoding

Concatenated codes to improve performance

Classic block codes and convolution codes are frequently combined in concatenated schemes.

In these schemes, convolution codes correct random errors, while the block code (usually Reed-Solomon) corrects burst errors.

Concatenated codes

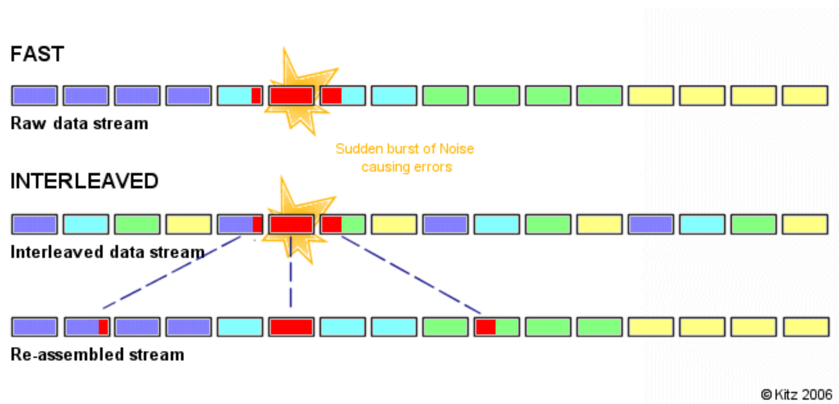


Interleaving

The idea is to change the order in which bits are sent, in such a way that error bursts are distributed in several code words instead of being concentrated in a single code word.

It is used together with block and convolution encoders.

Interleaving



Generation of interleaving