

# Módulo 06

## Detección y Corrección de Errores (Pt. 1)



Organización de Computadoras  
Depto. Cs. e Ing. de la Comp.  
Universidad Nacional del Sur



# Copyright

- Copyright © **2011-2015** A. G. Stankevicius  
Copyright © **2016** Leonardo de - Matteis
- Se asegura la libertad para copiar, distribuir y modificar este documento de acuerdo a los términos de la GNU Free Documentation License, Versión 1.2 o cualquiera posterior publicada por la Free Software Foundation, sin secciones invariantes ni textos de cubierta delantera o trasera.
- Una copia de esta licencia está siempre disponible en la página <http://www.gnu.org/copyleft/fdl.html>.



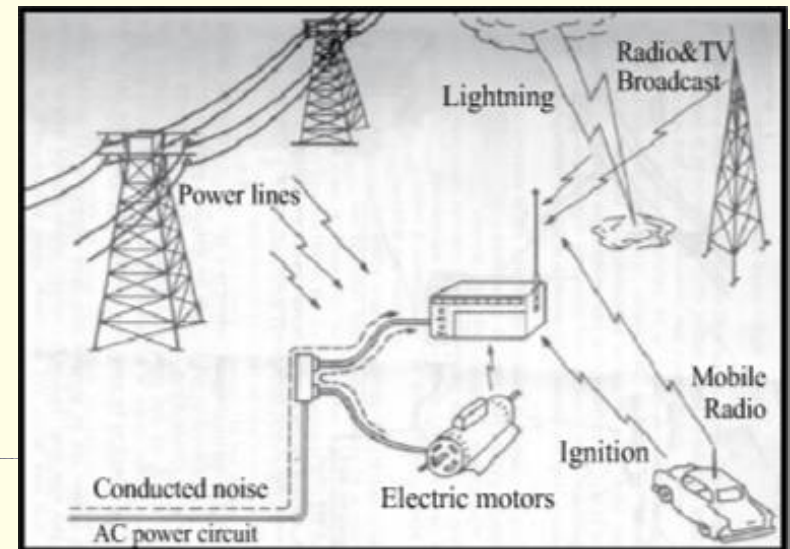
# Contenidos

- Concepto de error.
- Mínima distancia de un código.
- Mecanismos de detección de errores.
- Paridad aplicada en los códigos **VRC** y **LRC**.
- Generación y verificación de código **CRC**.
- Mecanismos de corrección de errores.
- Códigos correctores simples.
- Hamming mínima distancia 3 y 4.



# Concepto de error

- Toda vez que una pieza de información es transmitida existe la posibilidad de que **lo enviado no coincida con lo recibido**.
- El origen de estos errores suele depender del medio de transmisión utilizado:
  - Ruido y/o interferencia.
  - Atenuación de la señal.
  - Problemas de sincronización.
  - Propagación multicamino.



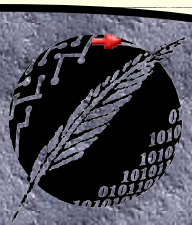
# Concepto de error

- ¿Qué actividades se verán afectadas?
  - La comunicación de información sobre grandes distancias (por caso, Internet).
  - La comunicación de información sobre cortas distancias (por caso, comunicación entre el **CPU** y la memoria o los dispositivos).
  - El almacenamiento de información en dispositivos no confiables (por caso, disquettes).
  - El almacenamiento de información en dispositivos lábiles (por caso, **CDs**, **DVDs** y **BRs**).



# Contramedidas

- Nosotros como humanos, ¿qué recaudos tomamos para contrarrestar los errores?
  - ➔ Al hablar por celular cuando entramos a un túnel o pasamos abajo de líneas de alta tensión, ¿qué hacemos si justo se corto lo que nos decían?
  - ➔ En un boliche con la música muy alta, al tratar de conversar con nuestra ocasional pareja de baile, ¿de qué manera hablamos?
  - ➔ Cuando un grupo de soldados solicita un ataque aéreo sobre posiciones enemigas en las cercanías, ¿de qué manera se transmiten las coordenadas?



# Contramedidas

- Para **contrarrestar el efecto de los errores** en la transmisión existen tres alternativas:
  - ➔ Aceptar que se produzcan: en ciertas circunstancias es posible que la información transmitida siga siendo relevante aún ante la presencia de un error (por caso, al transmitir un video).
  - ➔ Impedir que se produzcan: tomar todos los recaudos necesarios para asegurar que nunca se produzca un error. Esta alternativa suele tener un costo prohibitivo.
  - ➔ Contemplar que se produzcan: incorporar mecanismo que permitan atemperar el impacto de estos errores.



# Contramedidas

- En caso de adoptar la última alternativa la idea es incorporar un mecanismo que se encargue cancelar el impacto de los eventuales errores de transmisión.
- Estos mecanismos, cuyo propósito es lidiar con los errores, se clasifican en dos categorías:
  - ➔ Códigos detectores de error.
  - ➔ Códigos correctores de error.





# Detección vs. corrección

## ● Detección de errores:

- ➔ Un código detector incorpora información adicional junto con los datos transmitidos de manera que se pueda **determinar si se produjo o no un error** durante la transmisión.

## ● Corrección de errores:

- ➔ Un código corrector incorpora más información que uno detector, ya que la idea es, además de detectar si se produjo un error, tener la certeza de en qué lugar se produjo a fin de **poder corregirlo sin requerir la retransmisión** del dato en cuestión.



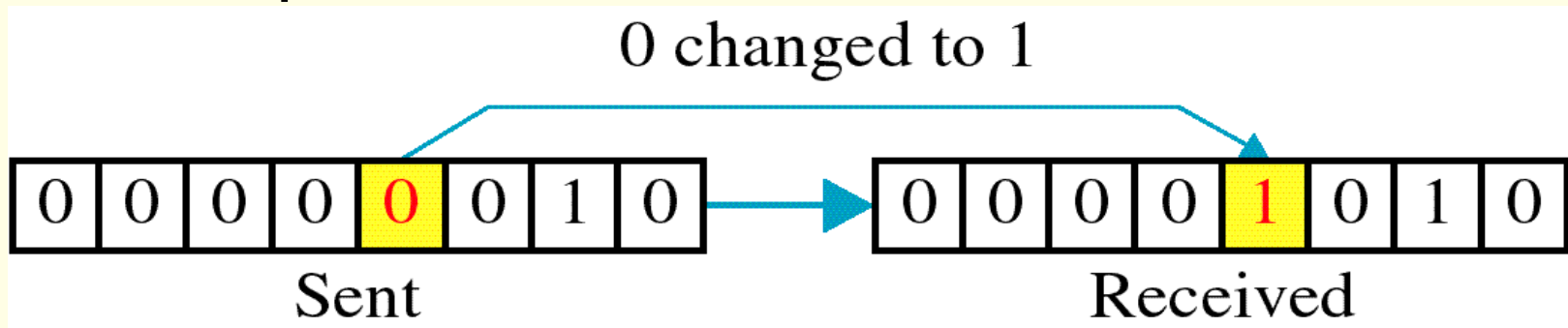
# Tipos de error

- Al trabajar con información binaria, el error se trata simplemente de un intercambio (*toggle*) del valor de uno o más bits.
- Los errores se clasifican de la siguiente manera:
  - Error a nivel de bit: afecta a **n** bits del dato transmitido. En función de **n**, hablamos de error simple, error doble, etc.
  - Error en ráfaga: afecta a **m** bits en un bloque de datos, estando el primero y el último en error (nótese que los bits entre medio pueden o no estar en error).

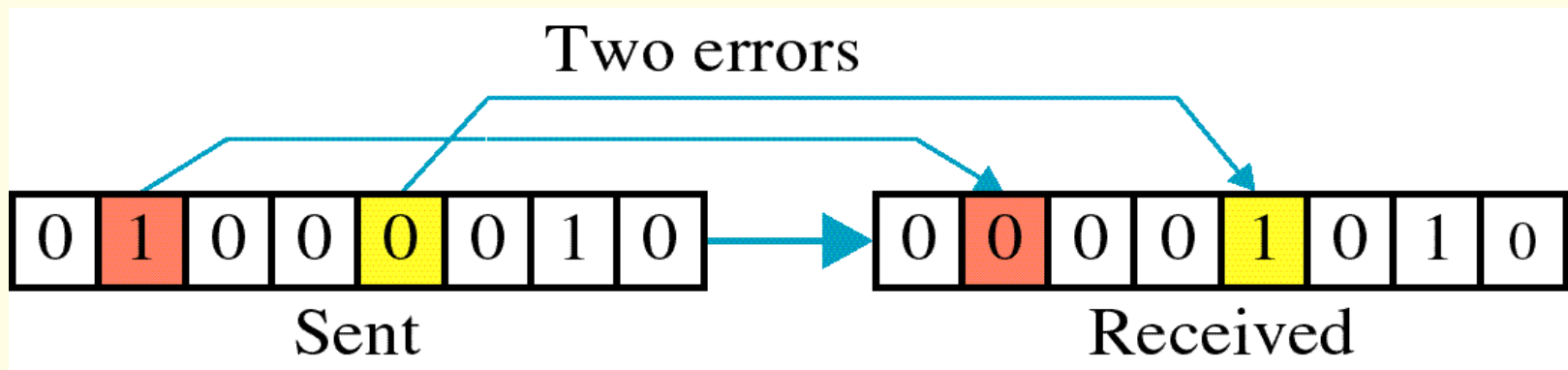


# Error a nivel de bit

- Error simple:



- Error doble:

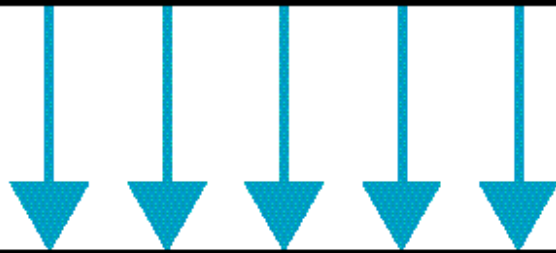


# Error en ráfaga

- Error en ráfaga (el ejemplo asume que la ráfaga alteró a la totalidad de los bits alcanzados):

Sent

0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



Burst error

0	1	0	1	1	0	1	1	0	1	0	0	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Received



# Definiciones

- Denominaremos **código** a un determinado conjunto de patrones de bits usualmente de longitud fija.
- Sean **p** y **q** dos patrones de bits, en este contexto denominaremos **distancia** entre **p** y **q**, notado  **$d(p, q)$**  a la cantidad de posiciones de bits en los cuales los patrones **p** y **q** difieren
- Sea **p** un patrón de bits, denominaremos **peso** de la palabra **p**, notado  **$w(p)$** , al número de bits puestos a **1** dentro de esa palabra.



# Ejemplo

- Sean  $p = 01001010$  y  $q = 10001011$ .
- En este contexto, calcular los siguientes pesos:
  - $w(p) = ?$
  - $w(q) = ?$
- Finalmente, determinar qué distancia existe entre los patrones  $p$  y  $q$ :
  - $d(p, q) = ?$



# Mínima distancia

- Sea **C** un código compuesto de patrones de bits de longitud fija.
- Denominaremos **mínima distancia** del código **C** a la menor distancia que se observe entre dos patrones de bits no idénticos tomados de **C**.
  - Tener en cuenta que para determinar la mínima distancia de un cierto código es necesario calcular un gran número de distancias, ya que se debe analizar la distancia entre cada patrón de bits con respecto a los restantes patrones de bits.



# Ejemplo

- Supongamos que un cierto código se compone de los siguientes patrones de bits los que a su vez codifican a cuatro caracteres:

**A: 0 0 0 0 0**

**B: 1 1 1 0 0**

**C: 0 0 1 1 1**

**D: 1 1 0 1 1**

- ¿Cuál es la mínima distancia de este código?





# Ejemplo


- Supongamos que el emisor envía el carácter **D** usando el código recién considerado (es decir, envía el patrón **11011**), pero el receptor recibe el patrón **11000**.
  - ¿Cuántos errores a nivel de bit se produjeron?
  - El patrón **11000** no será confundido con ningún otro patrón válido, es decir, este código permitió detectar el error.
  - ¿Hasta cuántos bits en error puede detectar un código cuya mínima distancia sea **m**?




# Análisis

- En general, podemos concluir lo siguiente:
  - Toda vez que se produzca una cantidad menor de errores que la mínima distancia del código adoptado, **el error será siempre detectado**.
  - Pero, si se produce una cantidad igual o mayor de errores, **a veces se detectará pero a veces no**.
  - Por caso, usando el código antes visto, tres errores pueden convertir una **A** en una **B**, o bien pueden convertir la **A** en un patrón inválido:

A: 0 0 0 0 0  
B: 1 1 1 0 0



A: 0 0 0 0 0  
1 1 0 0 1

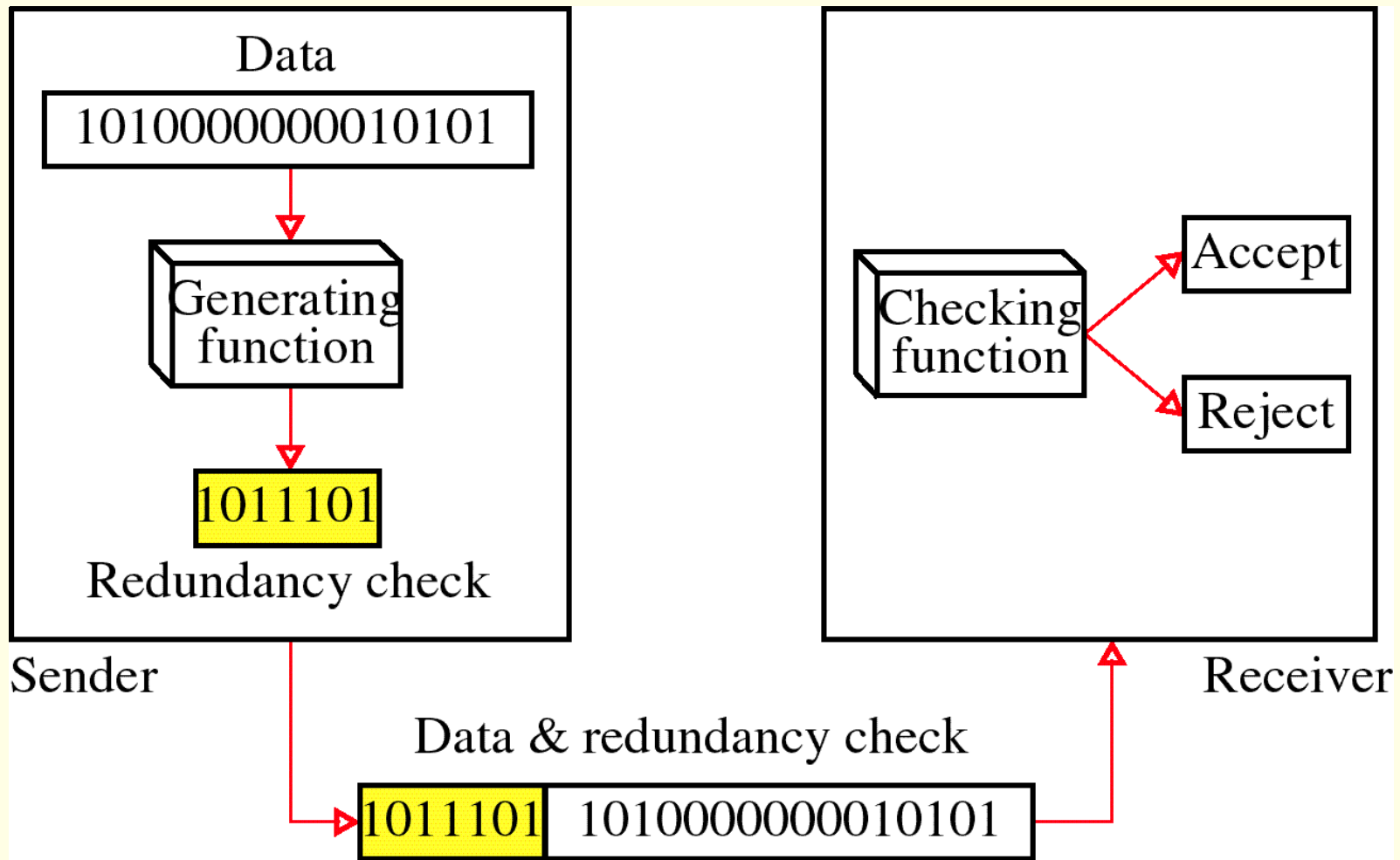


# Detección de errores

- Para detectar errores se debe incorporar alguna forma de redundancia al dato transmitido, a fin de que se pueda determinar si la información se recibió correctamente.
- Se han ensayado distintas alternativas:
  - **VRC**: verificación de redundancia vertical.
  - **LRC**: verificación de redundancia longitudinal.
  - **CRC**: verificación de redundancia cíclica.
  - **Checksum**: calculo de una suma de comprobación.



# Detección de errores



# Bit de paridad

- Una de las formas más elementales de detección de errores consiste en incorporar un **bit de paridad** al dato transmitido.
  - El bit de paridad es un bit que toma el valor **0** ó **1** con el objeto de satisfacer una cierta restricción sobre la paridad de un determinado patrón de bits.
- Es posible implementar un esquema de **paridad par** o **impar**, a saber:
  - Un patrón de bits se dice tener paridad par o impar si, y sólo si, tiene un peso par o impar respectivamente.



# Bit de paridad

- A partir de la paridad par o impar es posible definir sendos códigos de detección de errores:
  - ➔ Código de paridad par: a partir de un cierto dato se incorpora un bit adicional el cual adoptará el valor necesario para que el patrón dato + bit de paridad satisfaga una paridad par.
  - ➔ Código de paridad impar: definido de manera análoga, con la salvedad de que el patrón dato + bit de paridad ahora debe satisfacer una paridad impar.
- Por convención, el bit de paridad lo indicaremos **a la izquierda del patrón de bits original.**



# Ejemplo

- Supongamos que nuestro dato original es **p = 0111001**. El patrón de bits que resulta al incorporar un bit de paridad dependerá de la paridad que estemos usando:
  - **0 0111001**, al usar paridad par (pues **w(p)** es par).
  - **1 0111001**, al usar paridad impar (pues **w(p)** es par).
- Al recibir un cierto patrón de bits, se puede verificar si respeta la paridad adoptada:
  - Por caso, usando paridad par se acepta el patrón de bits **0 00000000**, pero se rechaza **1 11111111**.



# Análisis

- Paridad es un código con mínima distancia **2**, ¿qué capacidad de detección manifestará?
- Esta capacidad de detección, ¿depende del esquema de paridad elegido?
- ¿Será capaz de detectar errores en ráfaga?
- Este código no está en condiciones de corregir los errores detectados... ¿por qué razón?
- Resulta muy fácil de implementar en **HW**... ¿de qué manera?



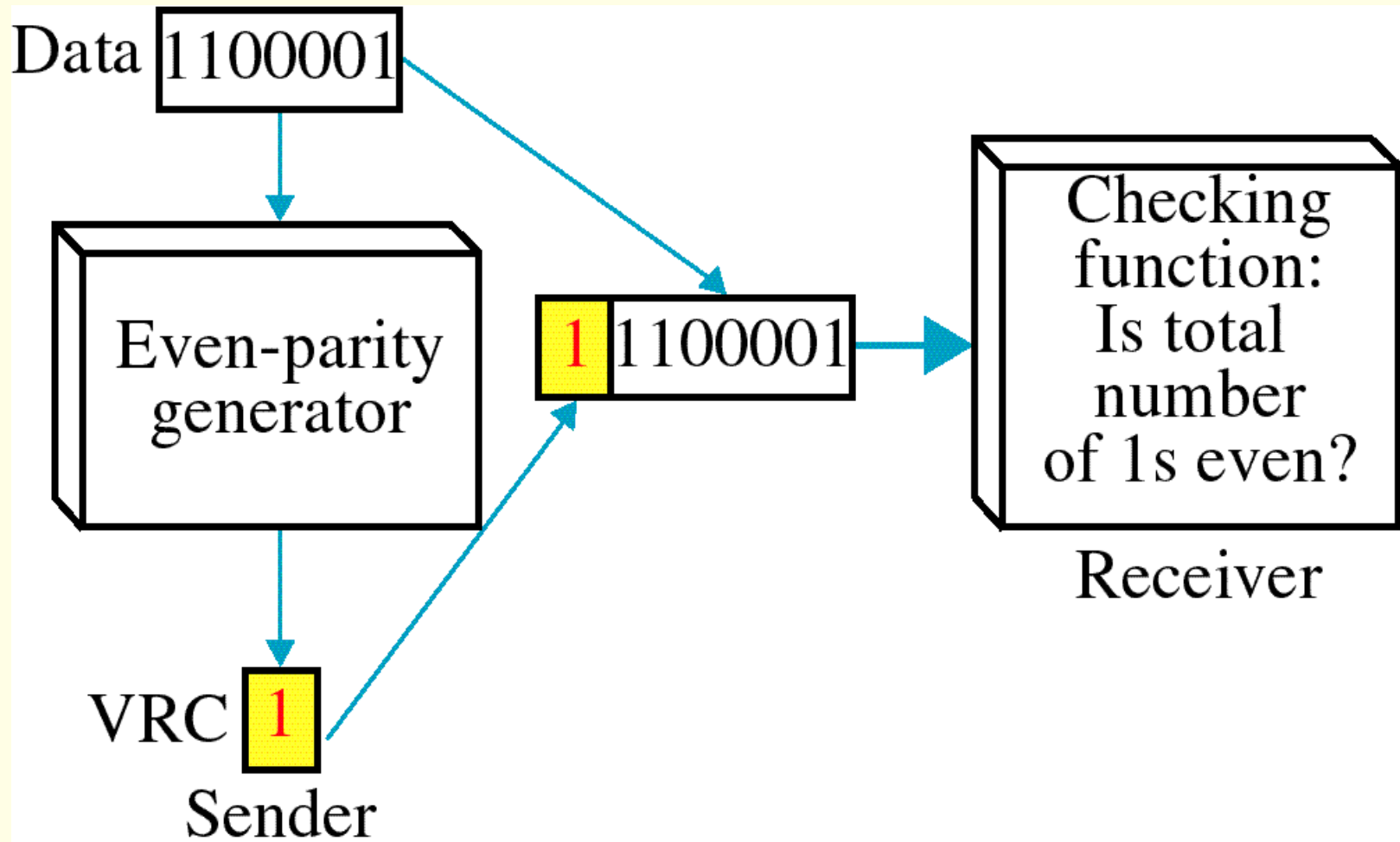


# Código VRC

- El código **VRC** (Vertical Redundancy Check) añade un bit de paridad a cada unidad de datos de manera que la cantidad de bits en **1** sea par o impar (en función de la paridad adoptada).
  - También se lo conoce como código **TRC** (Transverse Redundancy Check).
  - La idea es que se use un bit de paridad para cada unidad de datos, en vez de usar un único bit para la totalidad del mensaje.
  - Al igual que paridad, **detecta la totalidad de los errores simples** a nivel de unidad de datos.

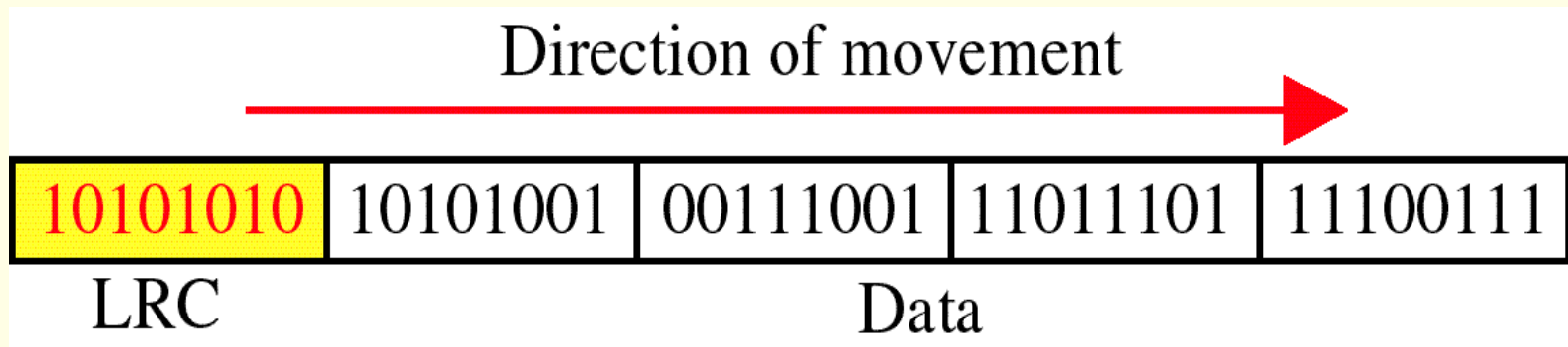


# Código VRC



# Código LRC

- El código **LRC** (Longitudinal Redundancy Check) aplica la misma idea que el código **VRC**, pero computa la paridad en sentido longitudinal:



- La idea es que un bloque de bits se divida en filas, para luego añadir una fila de bits de redundancia.
- La intención es permitir la **detección de errores en ráfaga**.



# Ejemplo

- Supongamos que se desea transmitir usando el código **LRC**, paridad par, al mensaje **HOLA** el cual está codificado en **ASCII** extendido:

H:	0	1	0	0	1	0	0	0
O:	0	1	0	0	1	1	1	1
L:	0	1	0	0	1	1	0	0
A:	0	1	0	0	0	0	0	1
<hr/>								
	0	0	0	0	1	0	1	0

- ¿En qué orden se deben transmitir estos bits?

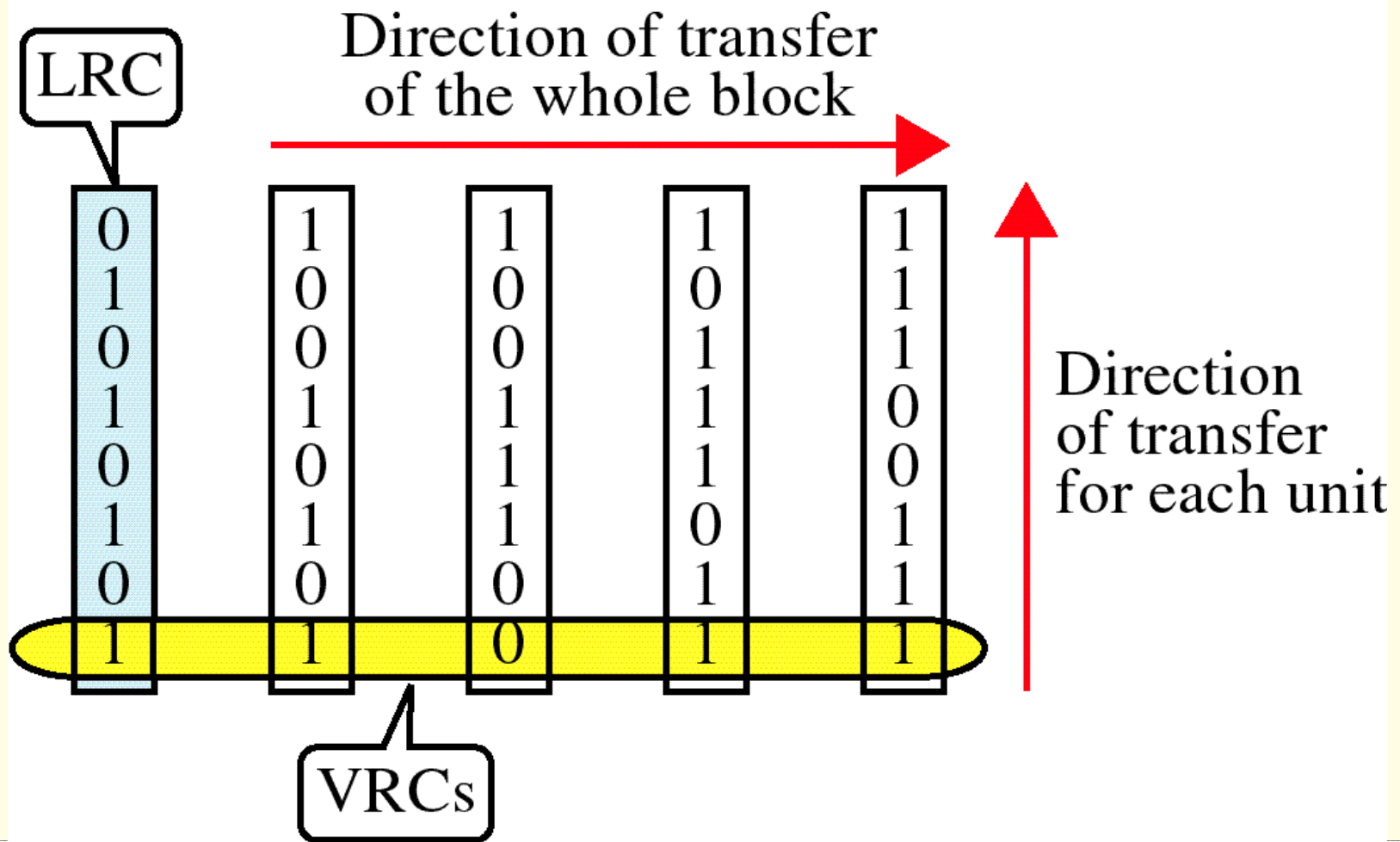


# Análisis

- El código **LRC** resulta más complicado de analizar producto de la manera en la que ordenan los bits del mensaje a ser transmitido.
- En relación a los errores a nivel de bit, este código sigue detectando correctamente a lo sumo errores simples.
- El principal beneficio es que ahora estamos en condiciones de **detectar errores en ráfaga**.
  - ➔ ¿Hasta qué tamaño de ráfaga podemos detectar correctamente?



# Combinando VRC y LRC



# Código CRC

- El **código CRC** (Cyclic Redundancy Check) se basa en ciertas propiedades matemáticas que satisface el cociente entre polinomios.
- ➔ Mecanismo: agregar al patrón de bits que compone el mensaje a ser enviado un conjunto de bits adicionales, de manera que el patrón resultante (al ser considerado como un polinomio binario), resulte divisible de manera exacta por un cierto polinomio denominado polinomio generador.



# Código CRC

- Para poder llevar adelante el cociente entre polinomios, se debe interpretar el mensaje original como si se tratara de un polinomio.
  - La clave está en pensar los  $n$  bits que componen al patrón original de bits como los coeficientes binarios de las primeras  $n - 1$  potencias de  $x$ .
  - Nótese que el polinomio resultante es de grado a lo sumo  $n - 1$ .
  - Por caso, el patrón **010011** denota al polinomio  $x^4 + x + 1$ , mientras que el patrón **110110** al polinomio  $x^5 + x^4 + x^2 + x$ .





# Código CRC

- Sea  $M(x)$  el polinomio binario representado el mensaje original, sea  $G(x)$  el polinomio generador que se esté usando y sea  $r$  su grado.
- En este contexto, el mensaje a ser transmitido  $T(x)$  es  $x^r M(x) + R(x)$ , donde  $R(x)$  es el resto de dividir  $x^r M(x)$  por  $G(x)$ .
  - $T(x)$  resulta divisible de manera exacta por  $G(x)$  puesto que al tratarse de polinomios binarios, se puede demostrar que  $x^r M(x) + R(x)$  equivale a  $x^r M(x) - R(x)$ .



# Algoritmo CRC

- Pasos para calcular los bits que deben ser agregados a un cierto mensaje  $M(x)$ :
  - Primero se añaden  $r$  bits en  $0$  a la derecha de  $M(x)$  (esto es, se añaden tantos ceros como grado tenga el polinomio generador).
  - Luego se divide el polinomio obtenido por el polinomio generador. Esta división se realiza en módulo dos, que es igual que la división binaria, con dos excepciones: no hay carries ni borrows.
  - Finalmente, para obtener  $T(x)$  se suma el resto  $R(x)$  al polinomio original  $M(x)$  (desplazado en  $r$  bits).



# Ejemplo

- Supongamos que el mensaje que se desea transmitir es  $M(x) = 11010110111$  y que el polinomio generador que se está usando es  $G(x) = 10011$ .

$$\begin{array}{r} x^r M(x) = 110101101110000 \\ \oplus 10011 \\ \hline 010011 \\ \oplus 10011 \\ \hline 0000010111 \\ \oplus 10011 \\ \hline 0010000 \\ \oplus 10011 \\ \hline 0001100 = R(x) \end{array}$$



# Ejemplo

- Verifiquemos que el mensaje a ser transmitido  $T(x) = x^r M(x) + R(x)$  es divisible de manera exacta por  $G(x)$ :

$$\begin{array}{r} x^r M(x) + R(x) = 110101101111100 \\ \oplus 10011 \\ \hline 010011 \\ \oplus 10011 \\ \hline 0000010111 \\ \oplus 10011 \\ \hline 0010011 \\ \oplus 10011 \\ \hline 00000000 \quad \checkmark \end{array}$$



# Roles del algoritmo

- El algoritmo **CRC** cumple dos roles:
  - Por un lado permite **determinar los bits que se deben agregar al mensaje original.**
  - A su vez, también permiten **verificar si el mensaje recibido contiene o no errores.**
- Nótese que el receptor puede ir dividiendo el mensaje a medida que va recibiendo los bits.
  - Es decir, **no hace falta recibirlo en su totalidad** para recién ahí comenzar a verificar el **CRC**.
  - Este código **es muy simple de implementar en HW.**



# Ejemplo

- Supongamos que se alteran un par de los bits del mensaje transmitido  $T(x)$ . En este contexto, verifiquemos nuevamente el cociente:

$$\begin{array}{r} 110101101111100 \rightarrow 100111101111100 \\ \oplus 10011 \\ \hline 0000011011 \\ \oplus 10011 \\ \hline 010001 \\ \oplus 10011 \\ \hline 00010110 \\ \oplus 10011 \\ \hline 001010 \quad \boxed{\times} \end{array}$$



# Polinomio generador

- El polinomio generador a ser usado **debe ser elegido con cuidado**, ya que la capacidad de detección de errores dependerá de las características del mismo.
- Existen varios polinomios actualmente en uso:
  - **CRC-4-ITU:**  $x^4 + x + 1$
  - **CRC-16-IBM:**  $x^{16} + x^{15} + x^2 + 1$
  - **CRC-CCITT:**  $x^{16} + x^{12} + x^5 + 1$
  - **CRC-32:**  $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$



# Códigos cíclicos

- Los polinomios generados más interesantes dan a lugar a **códigos cíclicos**.
- Un código se denomina cíclico cuando el mensaje  **$T(x)$**  puede ser corrido cíclicamente a derecha o izquierda un número arbitrario de lugares sin perder la propiedad de ser divisible exactamente por  **$G(x)$** .
- Sea  **$k$**  la longitud del mensaje original  **$M(x)$**  y sea  **$n$**  la del mensaje a transmitir  **$T(x)$** . Se puede demostrar que para obtener un código cíclico para el par  **$(n, k)$**  basta con tomar como polinomio generador de grado  **$n - k$**  a alguno de los factores del polinomio  **$x^n + 1$** .





# Ejemplo

- Se desea obtener un polinomio generador para  $k = 11$  y  $n = 15$ .

→ En primer lugar se debe factorizar  $x^{15} + 1$ :

$$x^{15} + 1 = (x^4 + x + 1) \times (x^4 + x^3 + 1) \times \\ (x^4 + x^3 + x^2 + x + 1) \times \\ (x^2 + x + 1) \times (x + 1)$$

- Luego, se puede tomar cualquiera de los factores de grado  $r = n - k = 4$ .
- Por caso, en el último ejemplo desarrollado usamos como polinomio generador  $(x^4 + x + 1)$ .



# Ejemplo

- Verifiquemos que al rotar tres lugares a derecha el mensaje **T(x)** del último ejemplo calculado se sigue verificando la propiedad:

$$\begin{array}{r} 110101101111100 \gg 3 = 100110101101111 \\ \oplus 10011 \\ \hline 00000010110 \\ \oplus 10011 \\ \hline 0010111 \\ \oplus 10011 \\ \hline 0010011 \\ \oplus 10011 \\ \hline 000000 \quad \checkmark \end{array}$$



# Relación entre CRC y paridad

- **CRC** se distingue de los métodos antes vistos de detección de errores ya que a priori parece no depender de la incorporación de bits de paridad entre medio de los bits del mensaje.
- No obstante, sigue existiendo un vínculo entre **CRC** y paridad:
  - ➔ Agregar un bit de paridad equivale a aplicar el código **CRC** usando  $x + 1$  como polinomio generador.
  - ➔ Verificar esta aseveración calculando el patrón de bits a transmitir para  $M(x) = 1101$  y  $G(x) = 11$ .



# Capacidad de detección

- Analizar la capacidad de detección de errores del código **CRC** no es del todo trivial, requiere un análisis pormenorizado.
- Sea  $T(x)$  el patrón de bits enviado y sea  $T(x) + E(x)$  el patrón recibido.
  - En caso de que  $E(x) = 0$  hablamos de una transmisión sin errores.
  - Caso contrario, cuando  $E(x) \neq 0$ , la mayor y menor potencia de  $x$  marcarán el comienzo y el fin de la ráfaga en error.



# Capacidad de detección

- Como por construcción  $G(x)$  divide a  $T(x)$ , la capacidad de detección del código **CRC** pivota en que  $G(x)$  no divida a  $E(x)$ .
  - Por caso, para un error en ráfaga de  $p$  bits comenzando a partir de la posición  $q$ , se verifica que  $E(x) = x^q(x^{p-1} + \dots + 1)$ .
  - Nótese que en el término de la derecha pueden faltar factores, salvo el primero y el último.
- Analicemos por casos lo que sucede para distintos valores de  $p$ .



# Capacidad de detección

## ● Error en ráfaga de longitud $p \leq r$ :

- En este caso no hay forma de que  $G(x)$  divida exactamente a  $E(x)$  puesto que no divide de manera exacta a ninguno de los dos términos que lo componen.
- Es decir, las ráfagas en error de una longitud menor o igual al grado del polinomio generador **son siempre detectadas.**



# Ejemplo

- Retomando el ejemplo anterior, toda ráfaga en error de longitud 4 o menor será detectada. Por caso, para  $E(x) = x^{10}(x^3 + 1)$  vimos que  $T(x) + E(x)$  no resulta divisible por  $G(x)$ :

$$\begin{array}{r} 110101101111100 \rightarrow 100111101111100 \\ \oplus 10011 \\ \hline 0000011011 \\ \oplus 10011 \\ \hline 010001 \\ \oplus 10011 \\ \hline 00010110 \\ \oplus 10011 \\ \hline 001010 \quad \boxed{\times} \end{array}$$



# Capacidad de detección

## ● Error en ráfaga de longitud $p = r + 1$ :

- En este caso, como estamos calculando el cociente entre polinomio de igual grado,  $G(x)$  dividirá a  $E(x)$  únicamente cuando  $G(x)$  coincida con el **factor** de la derecha de  $E(x)$ .
- En este escenario, ¿cuál será la probabilidad de no detectar el error? Pues bien, para que coincidan los  $r + 1$  en error, basta con que coincidan los  $r - 1$  bits internos (ya que en el polinomio generador, al igual que en  $E(x)$ , el primer y el último bit deben ser **1**).
- Es decir, **CRC** falla en sólo **1** de las  $2^{r-1}$  posibilidades.





# Ejemplo

- Analicemos el caso en que no se detecta la ráfaga en error de longitud 5. Por caso, considerando que  $E(x) = x^5(x^4 + x + 1)$ :

110101101111100 → 110100100011100

$\oplus 10011$

010010

$\oplus 10011$

000011000

$\oplus 10011$

010111

$\oplus 10011$

0010011

$\oplus 10011$

0000000 ✓



# Capacidad de detección

## • Error en ráfaga de longitud $p > r + 1$ :

- En este caso la situación es análoga, debemos analizar bajo qué condiciones  $G(x)$  divide exactamente al **factor** de la derecha de  $E(x)$ .
- Este análisis se reduce a considerar qué sucede con el último paso de la división: para que el resto sea  $0$  el último resto parcial debe coincidir con  $G(x)$ .
- Esto equivale a que coincidan sólo los primeros  $r$  bits de  $G(x)$  con ese último resto, ya que el bit más significativo de  $G(x)$  es necesariamente  $1$ .



# Ejemplo

- Analicemos el caso en que no se detecta una ráfaga en error de longitud mayor a 5. Por caso, sea  $E(x) = x(x^{12} + x^9 + x^2 + 1)$ :

$$\begin{array}{r} 110101101111100 \rightarrow 100111101110110 \\ \oplus 10011 \\ \hline 0000011011 \\ \oplus 10011 \\ \hline 010001 \\ \oplus 10011 \\ \hline 00010011 \\ \oplus 10011 \\ \hline 0000000 \quad \checkmark \end{array}$$



# Capacidad de detección

- En síntesis, la capacidad de detección de ráfagas en error de un código **CRC** que haga uso de un polinomio generador de grado **r** es:
  - Detecta con una probabilidad del **100%** a las ráfagas en error de longitud  **$k \leq r$** .
  - Detecta con una probabilidad de  **$1 - 2^{-(r-1)}$**  a las ráfagas en error de longitud  **$k = r + 1$** .
  - Detecta con una probabilidad de  **$1 - 2^{-r}$**  a las ráfagas en error de cualquier otra longitud.



# Ejemplo

● Consideremos la capacidad de detección de los códigos **CRC-12** y **CRC-CCITT**:

- ➔ **CRC-12** detecta el **100%** de las ráfagas en error de longitud **12** o menor, pero también detecta el **99.91%** de las ráfagas de longitud **13** y el **99.96%** de las restantes ráfagas en error.
- ➔ Pero adoptando **CRC-CCITT** en vez de **CRC-12**, la situación mejora notablemente, ahora se detecta el **100%** de la ráfagas en error de longitud **16** o menor, pero también el **99.994%** de las ráfagas de longitud **17** y el **99.997%** de las restantes.



# Funciones hash

- Si bien la capacidad de detección del código **CRC** es directamente proporcional al grado del polinomio generador que se esté usando, no se ha popularizado el uso de polinomios de grado mayor a 32.
  - ➔ El uso de polinomios generadores de grado tan alto **degrada el desempeño**.
- De hacer falta un mayor niveles de detección se puede hacer uso de **funciones hash** tales como **MD5** o **SHA-1**.

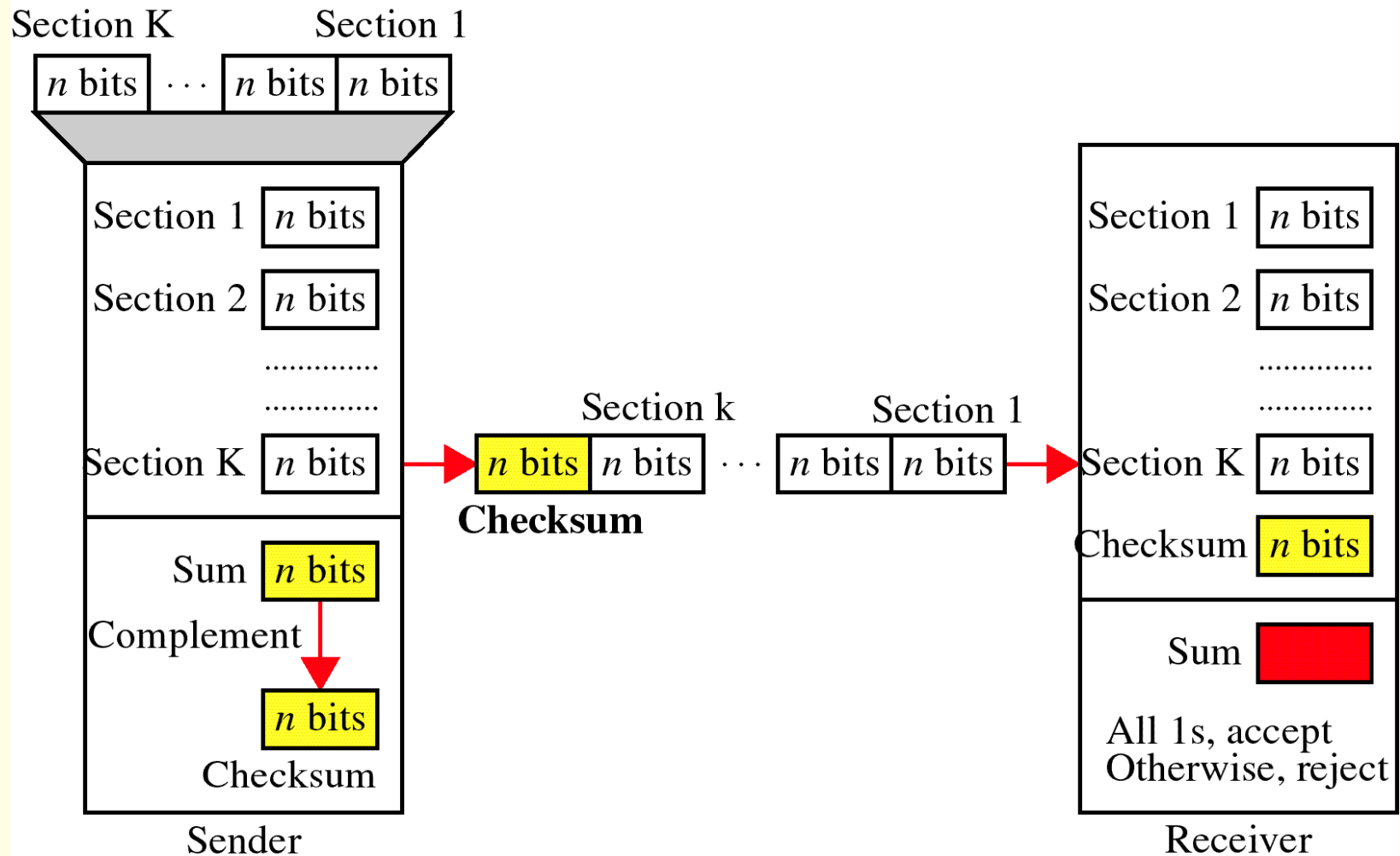


# Checksum

- El **checksum** (suma de comprobación) es un mecanismo relativamente sencillo para verificar la integridad de un mensaje.
  - ➔ La idea central consiste en ir sumando segmentos de datos de **n** bits en complemento a **1** de forma que la longitud de la suma sea también **n** bits, para luego complementar ese total obtenido antes de anexarlo al bloque de datos.
  - ➔ ¿Cuanto dará la suma de comprobación del bloque original al agregarle el complemento del resultado antes obtenido?



# Checksum





# Análisis

- Asumido un tamaño de segmento de datos de **8** bits, el checksum calculado no se altera en ninguno de los siguientes escenarios:
  - Al reordenar sin cambiar los bytes del mensaje.
  - Al insertar o eliminar bytes con sus bits todos en **0** o todos en **1**.
  - Al presentarse errores múltiples o en ráfaga pero que justo se cancelen unos a otros.
- En síntesis, checksum no constituye un mecanismo propicio de detección de errores.



# ¿Preguntas?

