Transmission Errors

Error Detection and Correction

Transmission Errors

- Transmission errors are caused by:
  - thermal noise {Shannon}
  - impulse noise (e..g, arcing relays)
  - signal distortion during transmission (attenuation)
  - crosstalk
  - voice amplitude signal compression (companding)
  - quantization noise (PCM)
  - jitter (variations in signal timings)
  - receiver and transmitter out of synch

Error Detection and Correction

- *error detection* :: adding enough "extra" bits to deduce that there is an error but not enough bits to correct the error.
- If only error detection is employed in a network transmission → retransmission is necessary to recover the frame (data link layer) or the packet (network layer).
  - *At the data link layer, this is referred to as* ARQ (Automatic Repeat reQuest).

Error Detection and Correction

- *error correction* :: requires enough additional (redundant) bits to deduce what the correct bits must have been.

Examples

Hamming Codes

FEC = Forward Error Correction *found in MPEG-4.*

Hamming Codes

codeword :: a legal dataword consisting of *m* data bits and *r* redundant bits.

Error detection involves determining if the received message matches one of the legal codewords.

Hamming distance :: the number of bit positions in which two bit patterns differ.

Starting with a complete list of legal codewords, we need to find the two codewords whose Hamming distance is the smallest. This determines the Hamming distance of the code.

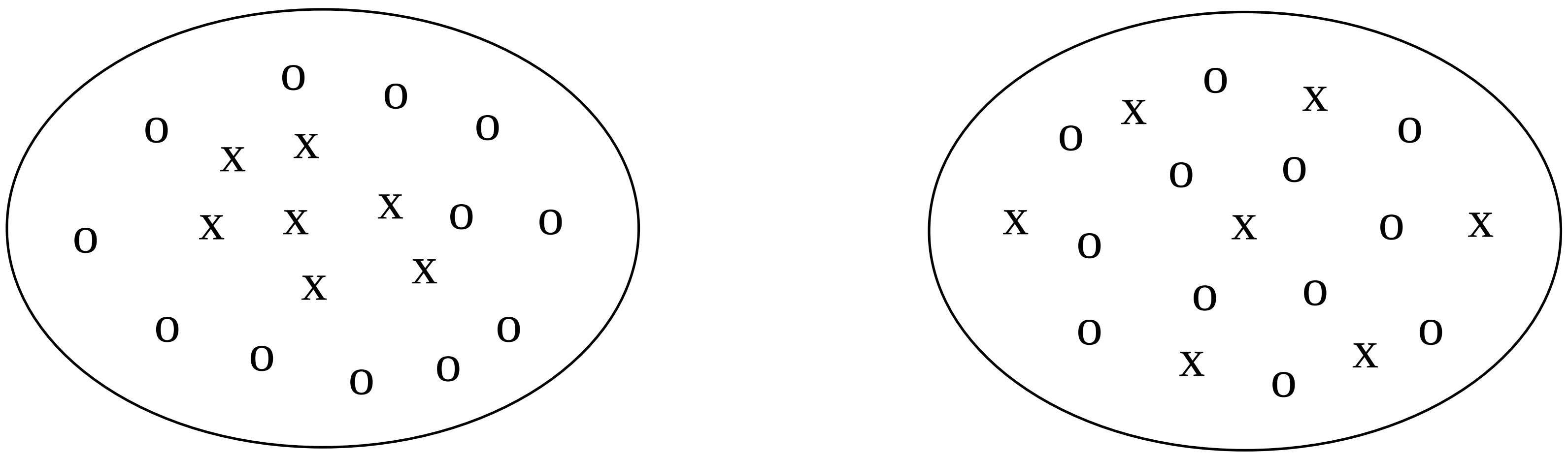
Error-Correcting Codes

Char.	ASCII	Check bits
H	1001000	00110010000
a	1100001	10111001001
m	1101101	11101010101
m	1101101	11101010101
i	1101001	01101011001
n	1101110	01101010110
g	1100111	01111001111
	0100000	10011000000
c	1100011	11111000011
o	1101111	10101011111
d	1100100	11111001100
e	1100101	00111000101

Order of bit transmission

Figure 3-7. Use of a Hamming code to correct burst errors.

(a) A code with poor distance properties (b) A code with good distance properties



x = codewords o = **non-codewords**

 Leon-Garcia & Widjaja: *Communication Networks* Figure 3.51

Hamming Codes

- To detect ***d*** single bit errors, you need a ***d+1*** code distance.
- To correct ***d*** single bit errors, you need a ***2d+1*** code distance.

➔In general, the price for redundant bits is too expensive to do error correction for network messages.

➔ use error detection and ARQ.

Error Detection

Remember – errors in network transmissions are <u>bursty.</u>

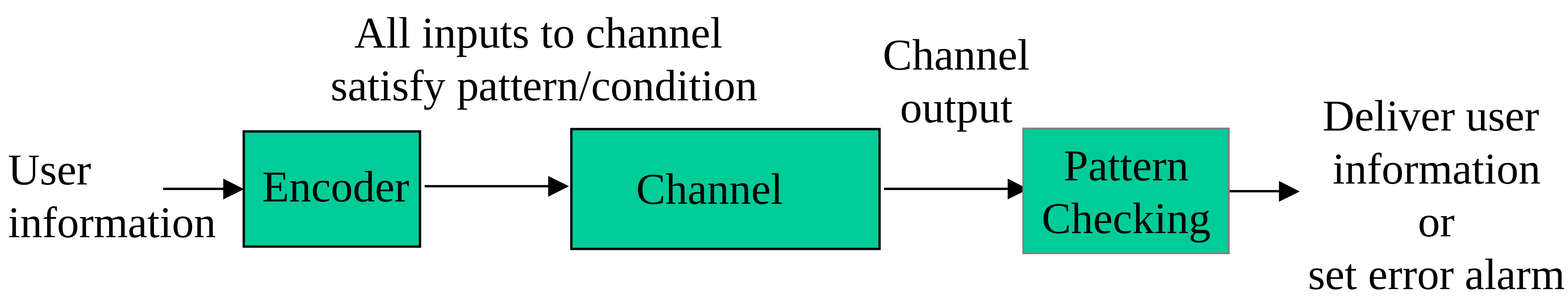
➔ *The percentage of damage due to errors is lower.*

➔ *It is harder to detect and correct network errors.*

- Linear codes
  - Single parity check code :: take *k* information bits and appends a single **check bit** to form a codeword.
  - Two-dimensional parity checks
- IP Checksum
- Polynomial Codes

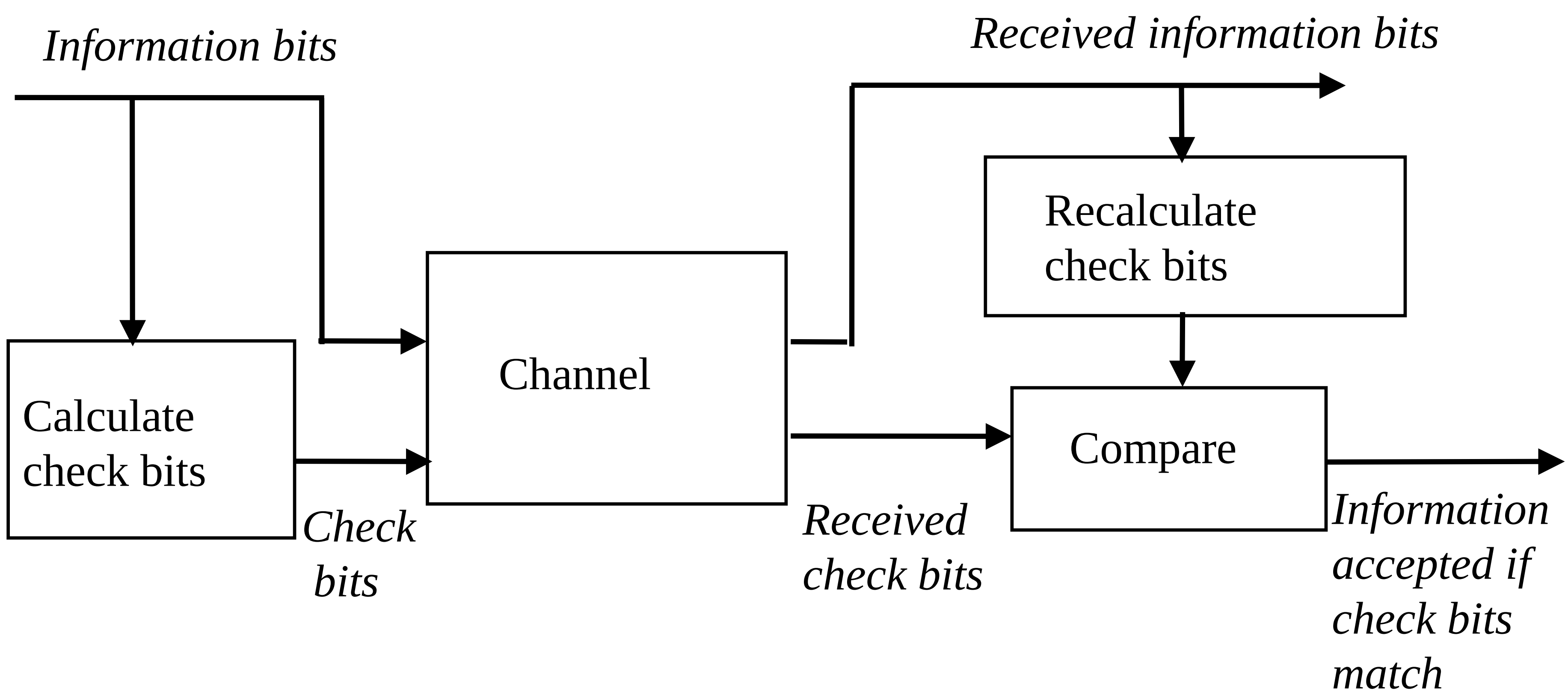
Example: **CRC** (Cyclic Redundancy Checking)

General Error-Detection System



 Leon-Garcia & Widjaja: *Communication Networks* Figure 3.49

Error-Detection System using Check Bits



Two-dimensional parity check code

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

Last column consists of check bits for each row

Bottom row consists of check bit for each column

Figure 3.52

One error

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

Two errors

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

Three errors

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

Four errors

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

Arrows indicate failed check bits

Figure 3.53

```
unsigned short cksum(unsigned short *addr, int count)
    {
        /*Compute Internet Checksum for “count” bytes
         *  beginning at location “addr”.
        */
    register long sum = 0;
    while ( count > 1 ) {
        /* This is the inner loop*/
             sum += *addr++;
             count -=2;
        }

        /*  Add left-over byte, if any  */
    if ( count > 0 )
        sum += *addr;

        /* Fold 32-bit sum to 16 bits */
    while (sum >>16)
        sum = (sum & 0xffff) + (sum >> 16) ;

    return ~sum;
}
```

Polynomial Codes [LG&W pp. 161-167]

- Used extensively.
- Implemented using <u>shift-register circuits</u> for speed advantages.
- Also called CRC (cyclic redundancy checking) because these codes generate check bits.
- Polynomial codes :: bit strings are treated as representations of polynomials with ONLY binary coefficients (0's and 1's).

Polynomial Codes

- The *k bits* of a message are regarded as the coefficient list for an information polynomial of degree *k-1*.

$$\mathrm{I} :: i(x) = i_{k-1}\, x^{k-1} + i_{k-2}\, x^{k-2} + \ldots + i_1\, x + i_0$$

Example

1 0 1 1 0 0 0

$$i(x) = x^6 + x^4 + x^3$$

Notation

- Encoding process takes *i(x)* produces a codeword polynomial *b(x)* that contains information bits and additional check bits that satisfy a pattern.
- Let the codeword have *n* bits with *k* information bits and *n-k* check bits.
- We need a *generator polynomial* of degree *n-k* of the form

$$G = g(x) = x^{n-k} + g_{n-k-1} x^{n-k-1} + \ldots + g_1 x + 1$$

Note – the first and last coefficient are always 1.

Polynomial Arithmetic

Addition:

$$(x^7 + x^6 + 1) + (x^6 + x^5) = x^7 + (1+1)x^6 + x^5 + 1$$
$$= x^7 + x^5 + 1$$

Multiplication:

$$(x+1)(x^2 + x + 1) = x^3 + x^2 + x + x^2 + x + 1 = x^3 + 1$$

Division:

$x^3 + x^2 + x$ = $q(x)$ quotient

divisor: $x^3 + x + 1$) $x^6 + x^5$:dividend

$$
\begin{array}{r l}
 & x^3 + x^2 + x \\
x^3 + x + 1 \,\big) & x^6 + x^5 \\
 & \underline{x^6 + \quad\; x^4 + x^3} \\
 & \quad x^5 + x^4 + x^3 \\
 & \quad \underline{x^5 + \quad\; x^3 + x^2} \\
 & \qquad x^4 + \quad\; x^2 \\
 & \qquad \underline{x^4 + \quad\; x^2 + x} \\
 & \qquad\qquad\qquad x
\end{array}
$$

x = $r(x)$ remainder

$$
\begin{array}{r r}
 & 3 \\
35\,\big) & 122 \\
 & \underline{105} \\
 & 17
\end{array}
$$

CRC Algorithm

Steps:

1) Multiply $i(x)$ by x^{n-k} (puts zeros in $(n\text{-}k)$ low order positions)

2) Divide $x^{n-k}\, i(x)$ by $g(x)$

quotient remainder

$$x^{n-k}i(x) = g(x)\, q(x) + r(x)$$

3) Add remainder $r(x)$ to $x^{n-k}\, i(x)$

(puts check bits in the n-k low order positions):

$$b(x) = x^{n-k}i(x) + r(x)$$ transmitted codeword

 Leon-Garcia & Widjaja: *Communication Networks* Figure 3.56

Information: (1,1,0,0) $\Longrightarrow$ $i(x) = x^3 + x^2$

Generator polynomial: $g(x)= x^3 + x + 1$

Encoding: $x^3 i(x) = x^6 + x^5$

$$
\begin{array}{r l}
 & x^3 + x^2 + x \\
\cline{2-2}
x^3 + x + 1 \,) & x^6 + x^5 \\
 & x^6 + \quad x^4 + x^3 \\
\cline{2-2}
 & \quad x^5 + x^4 + x^3 \\
 & \quad x^5 + \quad x^3 + x^2 \\
\cline{2-2}
 & \qquad x^4 + \quad x^2 \\
 & \qquad x^4 + \quad x^2 + x \\
\cline{2-2}
 & \qquad\qquad x
\end{array}
\qquad\qquad
\begin{array}{r l}
 & 1110 \\
\cline{2-2}
1011 \,) & 1100000 \\
 & 1011 \\
\cline{2-2}
 & \ 1110 \\
 & \ 1011 \\
\cline{2-2}
 & \ \ 1010 \\
 & \ \ 1011 \\
\cline{2-2}
 & \ \ \ \boxed{010}
\end{array}
$$

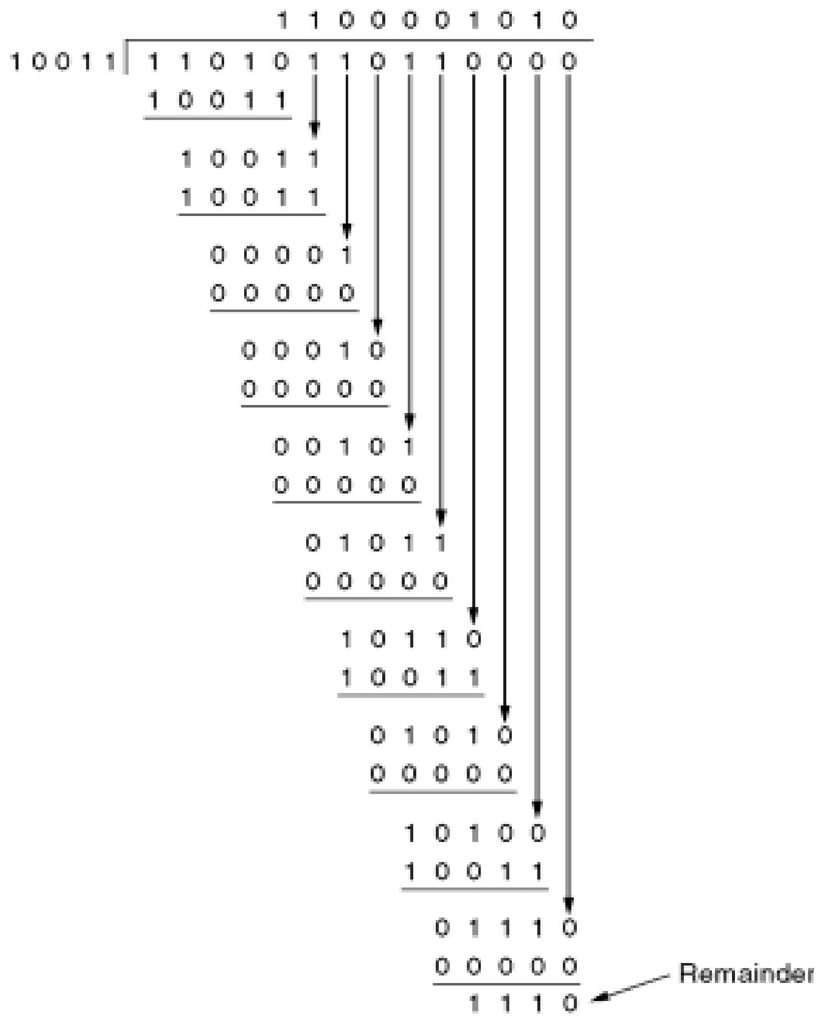
Transmitted codeword:

$b(x) = x^6 + x^5 + x$

$\Longrightarrow$ $\underline{b} = (1,1,0,0,0,1,0)$

 Leon-Garcia & Widjaja: *Communication Networks* Figure 3.57

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

Cyclic Redundancy Checking

Figure 3-8. Calculation of the polynomial code checksum.

Generator Polynomial Properties for Detecting Errors

1. **Single bit errors:** $e(x) = x^i \qquad 0 \le i \le n\text{-}1$

   If ***g(x)* has more than one term**, it cannot divide *e(x)*

2. **Double bit errors:** $e(x) = x^i + x^j \quad 0 \le i < j \le n\text{-}1$

   $= x^i (1 + x^{j-i})$

   If ***g(x)* is primitive**, it will not divide $(1 + x^{j-i})$ for $j\text{-}i \le 2^{n-k}-1$

3. **Odd number of bit errors:** $e(1) = 1$ If number of errors is odd.

   If ***g(x)* has (*x*+1) as a factor**, then $g(1) = 0$ and all codewords have an even number of 1s.

Generator Polynomial Properties for Detecting Errors

*i*th position

L

4. Error bursts of length b: 0000**110 • • • •0001101**100 • • • 0

error pattern $d(x)$

$$e(x) = x^i \ d(x) \quad \text{where } \deg(d(x)) = L\text{-}1$$

$g(x)$ has degree n-k;

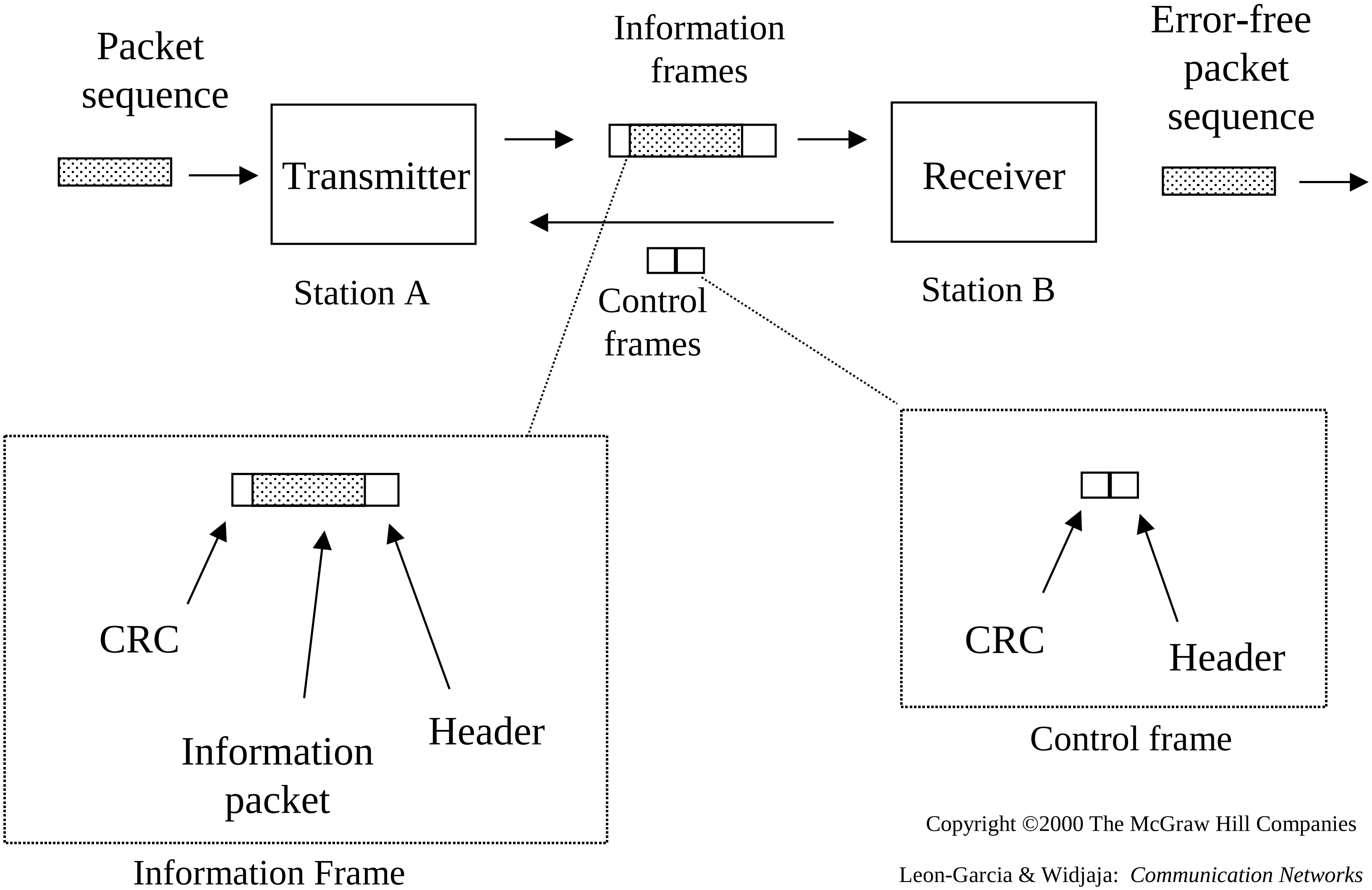
$g(x)$ cannot divide $d(x)$ if $\deg(g(x)) > \deg(d(x))$

- ***L = (n-k)*** or less: all errors will be detected
- ***L = (n-k+1)***: $\deg(d(x)) = \deg(g(x))$

  i.e. d(x) = g(x) is the only undetectable error pattern,

  fraction of bursts which are undetectable = $1/2^{L-2}$
- ***L > (n-k+1)***: fraction of bursts which are undetectable = $1/2^{n-k}$

Basic ARQ with CRC



Leon-Garcia & Widjaja: *Communication Networks*

Figure 5.8