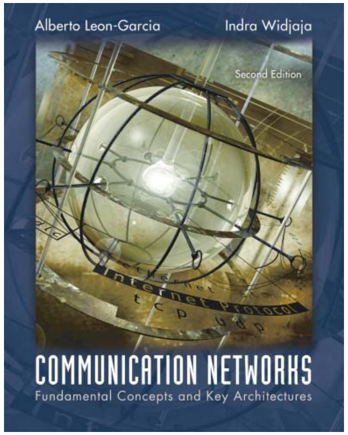
Chapter 3
Digital Transmission Fundamentals



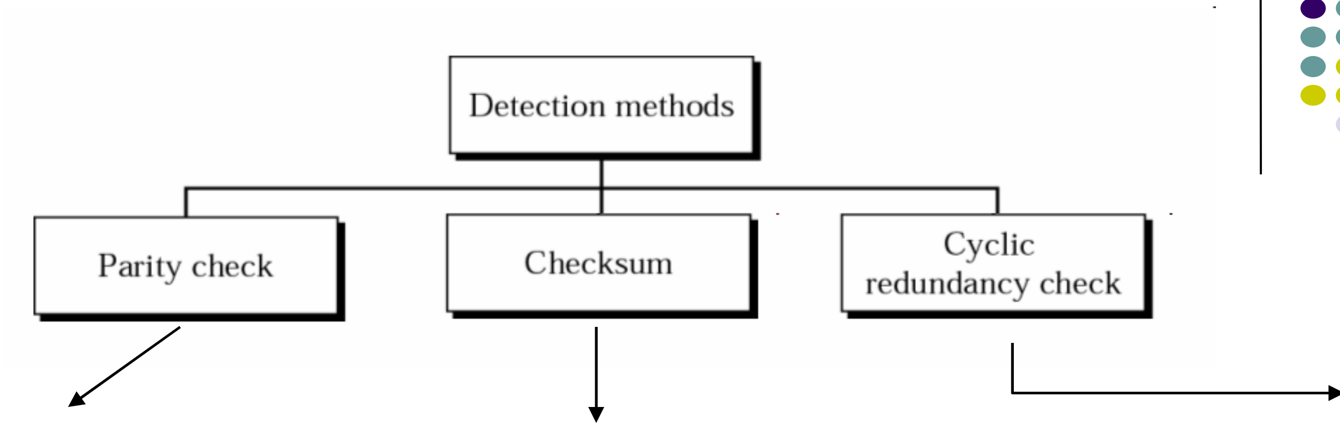
Error Detection and Correction

CSE 3213, Winter 2010

Instructor: Foroohar Foroozan

Other Error Detection Codes

- Many applications require very low error rate
- Need codes that detect the vast majority of errors
- Single parity check codes do not detect enough errors
- Two-dimensional codes require too many check bits
- The following error detecting codes used in practice:
  - Internet Check Sums
  - CRC Polynomial Codes



Single Parity

- **detects all error involving an odd # of errors**

2-D Parity

- **detects & <u>corrects</u> 1-bit errors**
- **detects all 2- and 3- bit errors**
- **detects some 4-bit errors**

Internet Checksum

- **detects all errors involving an odd # of bits**
- **detects most errors involving an even # of bits**

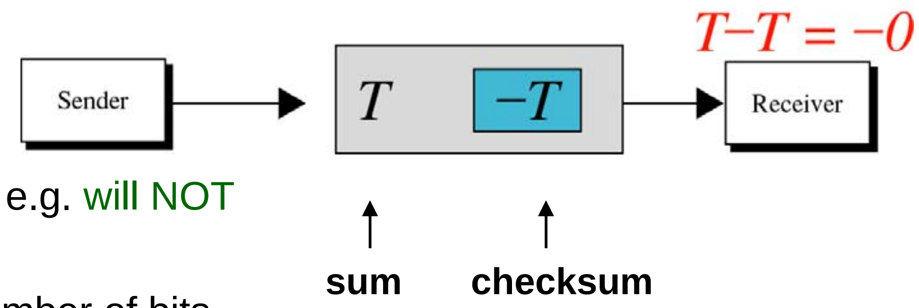
Error Detection: Internet Checksum

- Several Internet protocols (e.g. IP, TCP, UDP) use check bits to detect errors in the *IP header*
- A checksum is calculated for header contents and included in a special field.
- Checksum recalculated at every router, so algorithm selected for ease of implementation in software
- Let header consist of L, 16-bit words,

  $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{L-1}$
- The algorithm appends a 16-bit checksum $\mathbf{b}_L$

Checksum Calculation

- checksum calculation:
  - IP/TCP/UDP packet is divided into n-bit sections
  - n-bit sections are added using "1-s complement arithmetic" – the sum is also n-bits long!
  - the sum is complemented to produce checksum (complement of a number in 1-s arithmetic is the negative of the number)
- advantages:
  - relatively little packet overhead is required – n bits regardless of packet size
  - easy / fast to implement in software
- disadvantages:
  - weak protection compared to CRC – e.g. will NOT detect misordered bytes/words !!!
  - <u>detects</u> <u>all errors involving an odd number of bits</u> and <u>most errors involving an even number of bits</u>



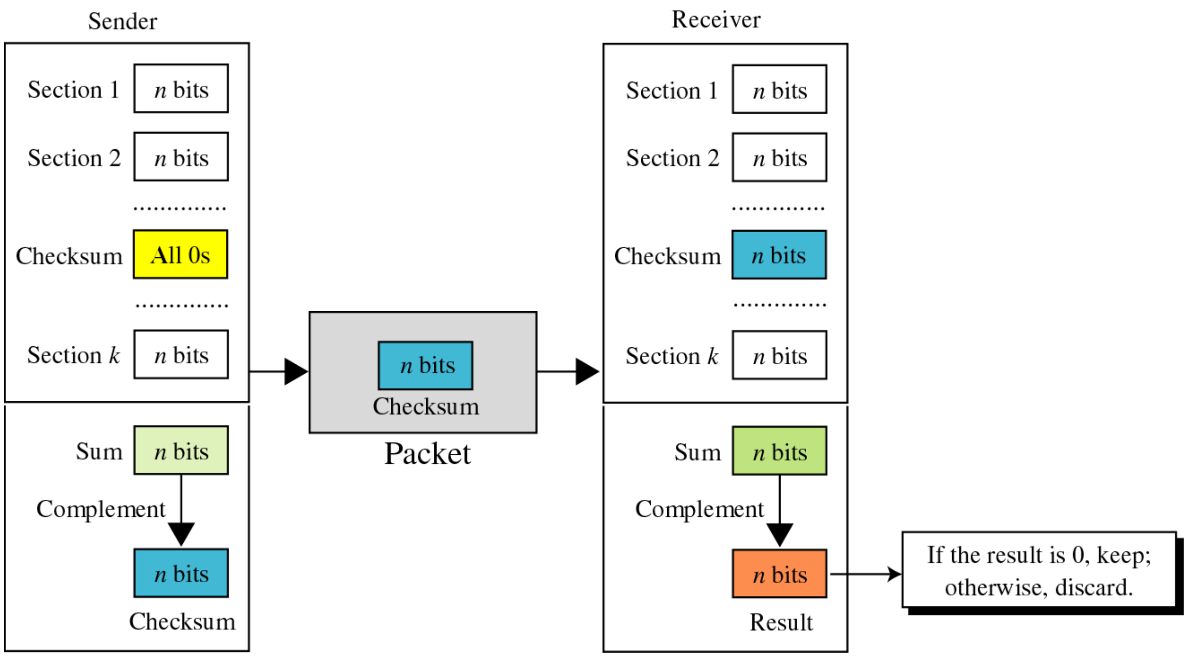
Checksum Calculation

Sender:

- data is divided into k sections each n bits long
- all sections are added using 1-s complement to get the sum
- the sum is bit-wise complemented and becomes the checksum
- the checksum is sent with the data

Receiver:

- data is divided into k sections each n bits long
- all sections are added using 1-s complement to get the sum
- the sum is bit-wise complemented
- if the result is zero, the data is accepted, otherwise it is rejected



Checksum Calculation

Example [Internet Checksum]

Suppose the following block of 8 bits is to be sent using a checksum of 4 bits: 1100 1010. Find the checksum of the given bit sequence.

1100
1010
0000

sum: 10110

0110
1
1-s complement addition: 0111 (7)

1-s complement addition:
Perform standard binary addition.
If a carry-out (>nth) bit it produced, swing that bits around and add it back into the summation.

checksum: 1000 (-7)

Negative binary numbers:
Negative binary numbers are bit-wise complement of corresponding positive numbers.

Checksum Calculation

Suppose the receiver receives the bit sequence and the checksum with no error.

	1100
	1010
	1000
sum:	**11110**
1-s complement addition:	**1111**
bit-wise complement:	**0000**

When the receiver adds the three blocks, it will get all 1s, which, after complementing, is all 0s and shows that there is no error.

If one or more bits of a segment are damaged, <u>and the corresponding bit of opposite value in a second segment is also damaged</u>, the sums of those columns will not change and the receiver will not detect the problem. ☹

Checksum Calculation

Example [Internet Checksum]

Suppose the following block of 16 bits is to be sent using a checksum of 8 bits.
10101001 00111001. The numbers are added using one's complement:

	10101001
	00111001
	00000000

Sum	**11100010**
Checksum	**00011101**

The pattern sent is 10101001 00111001 00011101.

Now suppose the receiver receives the pattern with no error.
10101001 00111001 00011101
When the receiver adds the three blocks, it will get all 1s, which, after complementing, is all 0s and shows that there is no error.

	10101001
	00111001
	00011101
Sum	**11111111**
Complement	**00000000**

means that the pattern is OK.

Checksum Calculation

Example [Internet Checksum]

Now suppose that in the previous example, there was a burst error of length 5 that affected 4 bits.

10101111 11111001 00011101

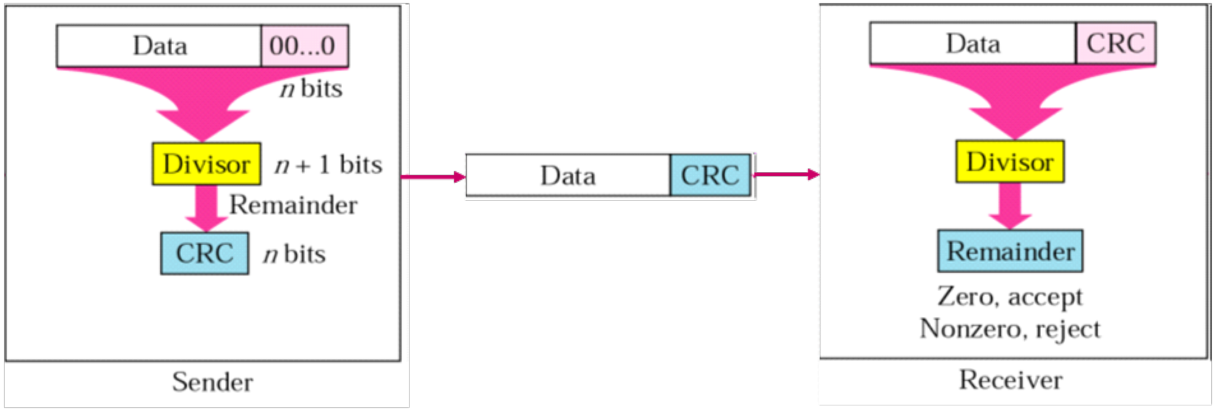
When the receiver added the three sections, it got

	10101111
	11111001
	00011101
Partial Sum	**1 11000101**
Checksum	**11000110**
Complement	**00111001**

the pattern is corrupted.

CRC (Polynomial Codes)

- Polynomials instead of vectors for codewords
- Polynomial arithmetic instead of check sums
- Implemented using shift-register circuits
- Also called *cyclic redundancy check (CRC)* codes
- Most data communications standards use polynomial codes for error detection
- Polynomial codes also basis for powerful error-correction methods



Binary Polynomial Arithmetic

- Binary vectors map to polynomials

$$(i_{k-1}, i_{k-2}, \ldots, i_2, i_1, i_0) \rightarrow i_{k-1}x^{k-1} + i_{k-2}x^{k-2} + \ldots + i_2x^2 + i_1x + i_0$$

Addition:

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

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

$$= x^7 + x^5 + 1 \quad \text{since } 1+1=0 \text{ mod2}$$

Multiplication:

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

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

$$= x^3 + 1$$

Binary Polynomial Division

- Division with Decimal Numbers

$$
\begin{array}{r l}
 & 34 \quad \leftarrow \text{quotient} \\
35\ ) & 1222 \quad \leftarrow \text{dividend} \\
 & 105 \\
\hline
 & 172 \\
 & 140 \\
\hline
 & 32 \quad \leftarrow \text{remainder}
\end{array}
$$

divisor

dividend = quotient x divisor +remainder

1222 = 34 x 35 + 32

- Polynomial Division

$$
\begin{array}{r l}
 & x^3 + x^2 + x \qquad = q(x) \text{ quotient} \\
\hline
x^3 + x + 1\ ) & x^6 + x^5 \qquad \leftarrow \text{dividend} \\
 & x^6 + \quad x^4 + x^3 \\
\hline
 & x^5 + x^4 + x^3 \\
 & x^5 + \quad x^3 + x^2 \\
\hline
 & x^4 + \quad x^2 \\
 & x^4 + \quad x^2 + x \\
\hline
 & x \qquad = r(x) \text{ remainder}
\end{array}
$$

divisor

Note: Degree of r(x) is less than degree of divisor

Polynomial Coding

- Code has binary *generating polynomial* of degree $n\text{–}k$

$$g(x) = x^{n-k} + g_{n-k-1}x^{n-k-1} + \ldots + g_2x^2 + g_1x + 1$$

- *k information bits* define polynomial of degree $k - 1$

$$i(x) = i_{k-1}x^{k-1} + i_{k-2}x^{k-2} + \ldots + i_2x^2 + i_1x + i_0$$

- Find *remainder polynomial* of at most degree $n - k - 1$

$$g(x)\ \overline{\big)\ x^{n-k}\, i(x)}^{\;q(x)} \qquad r(x)$$

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

- Define the *codeword polynomial* of degree n -1

$$\boxed{\underbrace{b(x)}_{n\text{ bits}} = \underbrace{x^{n-k}i(x)}_{k\text{ bits}} + \underbrace{r(x)}_{n-k\text{ bits}}}$$

Polynomial example: $k = 4$, $n-k = 3$

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

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

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

$$\begin{array}{r l}
 & x^3 + x^2 + x \\ \hline
x^3 + x + 1\,) & x^6 + x^5 \\
 & x^6 + x^4 + x^3 \\ \hline
 & x^5 + x^4 + x^3 \\
 & x^5 + x^3 + x^2 \\ \hline
 & x^4 + x^2 \\
 & x^4 + x^2 + x \\ \hline
 & x
\end{array}$$

$$\begin{array}{r l}
 & 1110 \\ \hline
1011\,) & 1100000 \\
 & 1011 \\ \hline
 & 1110 \\
 & 1011 \\ \hline
 & 1010 \\
 & 1011 \\ \hline
 & 010
\end{array}$$

Transmitted codeword:

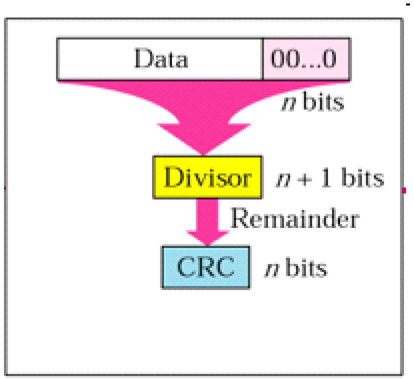
$b(x) = x^6 + x^5 + x$

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

Polynomial Coding (Cont.)

CRC Polynomial Arithmetic (cont.)

divisor / generator polynomial:	G	(n-k+1 bits)
information:	I	(k bits, k<n)
CRC remainder:	R	(≤ n-k bits)
transmitted frame – I+R:	B	(n bits)



Sender

- CRC process can now be described as:

step 1) $$\frac{\boxed{X^{n-k}\cdot I(X)}}{G(X)} = Q(X) + \frac{R(X)}{G(X)}$$

step 2) $$B(X) = X^{n-k}\cdot I(X) + R(X)$$ ← - - - - transmitted frame

- note, from step 2) and 1)

$$B(X) = \boxed{X^{n-k}\cdot I(X)} + R(X) = [G(X)\cdot Q(X) + R(X)] + R(X)$$

and in modulo-2 arithmetic

$$B(X) = G(X)\cdot Q(X)$$ ← - - - - - - - - - - - - - transmitted frames, i.e. all valid codewords are multiples of the generator polynomial

$$\boxed{B(X)/G(X) = Q(X),\ \text{no remainder}}$$

Polynomial Coding (Cont.)

Error Detection with CRC Polynomial Arithmetic

– receiver can check whether there have been any transmission errors by dividing the received polynomial (B'(X)) by G(X)

- if there are no errors, remainder = 0

$$B'(X) = B(X): \qquad \frac{G(X) \cdot Q(X)}{G(X)} = Q(X), \quad \text{no remainder}$$

- if remainder $\neq 0$, an error is detected

$$B'(X) = B(X) + E(X): \qquad \frac{G(X) \cdot Q(X) + E(X)}{G(X)} = Q(X) + \boxed{\frac{E(X)}{G(X)}}$$

- note: if error polynomial E(X) is divisible by G(X), error pattern will be undetectable !!!

- design of polynomial codes involves:

  1) identifying error polynomials we want to be able to detect

  2) synthesizing a generator polynomial that will not divide the given error polynomials without remainder

Polynomial Coding

Designing Good Polynomial Codes – G(X)

(1) Codes that Detect Single Errors

- codeword of n bits $\Rightarrow$ $E_{single} = (0,0,0,1,0,0, \ldots, 0)$ $\Rightarrow$ $E(X) = X^i$, $0 \le i < n$
- if G(X) has more than one term, it cannot divide E(X) without remainder

(2) Codes that Detect Double Errors

- codeword n bits $\Rightarrow$ $E_{double} = (0,0,0,1,0,1, \ldots, 0)$ $\Rightarrow$

$$\Rightarrow \quad E(X) = X^i + X^j, \quad 0 \le i < j \le n$$

$$\Rightarrow \quad E(X) = X^i\,(1+ X^{i-j}), \quad 0 \le i < j \le n$$

- from (1), we have picked G(X) such that it has more than one term and cannot divide X^i $\Rightarrow$ E(X) will be divisible by G(X) only if G(X) divides $(1 + X^{i-j})$ $\Rightarrow$ so we are interested in G(X) that does NOT divide $(1 + X^{i-j})$ without remainder
- if G(x) is a *primitive polynomial* of degree N, it cannot divide X^m+1 for all $m<2^N-1$ $\Rightarrow$ need to keep codeword length less than 2^N-1

Polynomial Coding

Example [primitive polynomial]

Primitive polynomial – cannot be factorized!

$X^2 + 1 = (X+1)(X+1)$ - is NOT a primitive polynomial

$X^2 + X + 1$ - is a primitive polynomial ◄------------------ Primitive polynomials can be found by consulting coding theory books!

Example [CRC-16 polynomial generator / code]

$G(X) = X^{16} + X^{15} + X^2 + 1 = (X + 1)*(X^{15} + X + 1)$

$\Rightarrow$ $(X^{15} + X + 1)$ is a primitive polynomial of degree N=15

$\Rightarrow$ $(X^{15} + X + 1)$ cannot divide (X^m+1) for all $m < 2^N-1 = 32{,}767$

$\Rightarrow$ G(X) will detect all double errors as long as codeword length < 32,767

Example [CRC-12 polynomial generator / code]

$$\mathbf{G(X) = X^{12} + X^{11} + X^3 + X^2 + X + 1 = (1 + X)*(X^{11} + X^2 + 1)}$$

Polynomial Coding

Designing Good Polynomial Codes – G(X) (cont.)

(3) Codes that Detect Odd Number of Errors

- we want to make sure that CRC performs as good as single parity check
- E(X) has an odd number of terms, hence at X=1 $\Rightarrow$ E(1) = 1
- G(X) must have a factor (X+1), since there is no polynomial E(X) with an odd number of terms that has (1+X) as a factor
  - PROOF: assume such a polynomial, E(X), exists, then

$$E(X) = (1+X)\,Q(X) \quad \Rightarrow \quad E(1) = (1+1)*Q(1) = 0$$

    and this contradicts the fact that E(1) = 1, due to an odd number of terms

- pick $G(X)=(X+1)*P_{primitive}(X)$ to be able to detect all single, double, and odd-number of errors

Polynomial Coding

Example [CRC error control]

Let $G(x) = (x^3+x^2+1)$. Consider the information bits (1,1,0,1,1,0).

(a) Find the codeword corresponding to these information bits if G(x) is used as the generating polynomial.

(b) Can G(x) detect single errors? Double errors?

$G(X) = X^3 + X^2 + 1$, n=4

$I(X) = X^5 + X^4 + X^2 + X$

$X^3 * I(X) = X^8 + X^7 + X^5 + X^4$

$R(X) = X^2 + X + 1 \quad \Rightarrow \quad R = (1,1,1)$

B = (1,1,0,1,1,0,1,1,1)

$$
\begin{array}{r|l}
 & x^5 + x + 1 \\
\hline
x^3 + x^2 + 1 & x^8 + x^7 + x^5 + x^4 \\
 & x^8 + x^7 + x^5 \\
\hline
 & x^4 \\
 & x^4 + x^3 + x \\
\hline
 & x^3 + x \\
 & x^3 + x^2 + 1 \\
\hline
 & x^2 + x + 1
\end{array}
$$

- Single errors can be detected since G(X) has more than one term.
- Double errors cannot be detected even though G(X) is primitive, because the codeword length exceeds $2^3 - 1 = 7$.

Polynomial Coding (Exercise)

1. Which error detection method uses ones complement arithmetic?
   (a) single parity check
   (b) 2-D parity check
   (c) CRC
   (d) checksum

2. In cyclic redundancy checking, the divisor is ______________________ the CRC.
   (a) the same size as
   (b) 1 bit less than
   (c) 1 bit more than
   (d) 2 bits more than

3. In CRC there is no error if the remainder at the receiver is ________________.
   (a) equal to the remainder at the sender
   (b) zero
   (c) nonzero
   (d) the quotient at the sender

4. Which error detection method can detect a burst error?
   (a) the parity check
   (b) 2-D parity check
   (c) CRC
   (d) (b) and (c)