Error Control Techniques

Model of a conventional signaling system

1. random errors
2. burst errors

Two categories of error control techniques

1. ARQ (automatic-repeat-request)
   - buffering, error-detection-codes, acknowledgment channel, retransmission.
2. FEC (forward error control)
   - error-correction codes (put enough redundancy information for correction).

FEC is inferior to ARQ except when

1. an acknowledgment channel is not available or expensive, or, even dangerous!
   (e.g. deep space comm. such as Voyage II)
2. the small fraction of correctable error patterns has almost all the probability weight.
Arithmetic Checksum

Error detection at the higher layer is usually done by ordinary arithmetic operations. This is simpler in software but somewhat less effective than a CRC.

Standard technique is to view packet as sequence of k numbers of n bits each, say \( x_1, x_2, \ldots, x_k \).

Checksum is then the n bit number \( x_1 + x_2 + \ldots + x_k \) using ordinary arithmetic with no carry.

Alternatively, checksum might be 2n bits; first n bits is (sum) \( x_1 + x_2 + \ldots + x_k \) and second n bits is (sum of sum) \( x_1 + 2x_2 + 3x_3 + \ldots + kx_k \).

Example: In TCP, \( n=16 \), checksum is 16 bits and one complement of the sum. In ISBN, the data are radix 10 digits, checksum is radix 11 digit (with 10 represented as X) and is (sum of sum of all digits)/11.
Weighted code used in ISBN number for Error Detection

Our textbook has a ISBN number 0-13-162959-X.
To check that this number is a proper ISBN number we proceed as follows:

<table>
<thead>
<tr>
<th>Number</th>
<th>SUM</th>
<th>SUM of SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>21</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>34</td>
</tr>
<tr>
<td>9</td>
<td>22</td>
<td>56</td>
</tr>
<tr>
<td>5</td>
<td>27</td>
<td>83</td>
</tr>
<tr>
<td>9</td>
<td>36</td>
<td>119</td>
</tr>
<tr>
<td>10=X</td>
<td>46</td>
<td>165=11×15</td>
</tr>
</tbody>
</table>

165 mod 11 = 0

a correct ISBN number
A code consists of the rule/algorithm for computing check bits from data bits and for generating codewords from data bits and check bits.

The coding algorithm defines the legal or illegal codewords.

For fixed length codes,

For \( x, y \) the set of codewords, Hamming distance, \( \text{Hd}(x,y) \) is the no. of 1's and \( d=x-y \).

The Hamming distance of a code, \( C \), is \( \text{Hd}(C)=\min\{z \mid z = \text{Hd}(x,y) \text{ where } x, y \text{ are code-words of } C, \text{ and } x \neq y\} \).

To detect \( d \) (single) errors, we need a code \( C \) with \( \text{Hd}(C)=d+1 \).

To correct \( d \) errors, we need a code \( C \) with \( \text{Hd}(C)=2d+1 \).

Exercise: Prove the \( \text{Hd} \) (odd parity code) = 2.

For single error correcting code, where \( m(r) \) is the no. of data(check) bits,

How many check bits is required? Prove that \( r \) must satisfy \( (m+r+1) \leq 2^r \).

Each of \( 2^m \) msgs has \( n \) illegal codes at Hamming distance 1 from it.

Each of \( 2^m \) msgs requires \( n+1 \) bit patterns from \( 2^n \) bit patterns.

\[(n+1)2^m \leq 2^n \quad (m+r+1)2^m \leq 2^{m+r} \quad (m+r+1) \leq 2^r \]

For \( m=8, 8+r+1 \leq 2^r, \quad r=4 \).
Hamming's Single Error Correcting Code

3 5 6 7 9 1011 Data bit Positions
\[ C_0 C_1 D_1 C_2 D_2 D_3 C_3 D_3 D_4 D_5 C_4 D_4 D_5 D_6 C_5 D_6 D_7 C_6 D_7 D_8 C_7 D_8 D_9 C_8 D_9 D_{10} C_9 D_{10} D_{11} C_{10} D_{11} D_{12} C_{11} D_{12} D_{13} C_{12} D_{13} D_{14} C_{13} D_{14} D_{15} C_{14} D_{15} \]

Encode the check bits using even parity

X bit 7 Error

X X X Errors in the check bits

1+2+4=7 X position of error is bit 7.

Corrected Message

a 1 1 0 0 0 0 1 ASCII code

Exercise: Illustrate how the receiver corrects bit 6 error in a Hamming code of $\text{￥￥}$
Illustrate how the receiver correct bit 6 error in the Hamming code of $U = 101010$

Hamming's Single Error Correcting Code

- **C0** checks positions 1, 3, 5, 7, 9, 11, 13, 15,... (make it even parity.)
- **C1** checks positions 2, 3, 6, 7, 10, 11, 14, 15,...
- **C2** checks positions 4, 5, 6, 7, 12, 13, 14, 15,...
- **C3** checks positions 8, 9, 10, 11, 12, 13, 14, 15, 24, 25,...

$U = 1010101$ Message

- 1 0 1 0 1 0 1 Data bits
- 1 1 1 1 0 1 0 0 1 0 1 Encode
- **X** bit 6 Error
- 1 1 1 1 0 0 0 0 1 0 1 Code Received
- 1 0 0 0 1 0 1 Data bits Received
- 1 0 0 0 Regenerate Check bits Using Received Data
- **X** **X** Errors in the check bits,
- $2 + 4 = 6$ **X** Add the weight of error check it position = 6.
- 1 0 1 0 1 0 1 Reverse bit 6 Corrected Message
Burst Error Correction

Arrange $k$ Hamming codewords in a matrix

```
  0 0 1 1 0 0 1 0 0 0 0
  1 0 1 1 1 0 0 1 0 0 1
  0 0 1 1 0 0 1 0 0 0 0
```

send bits columnwise

Use $k^r$ check bits to correct a single burst error of length $k$.
Trade-off is the delay increases from $n/C$ to $k^*n/C$ where $C$ is the link capacity.

ECC vs. EDC

For error rate=$10^{-6}$, 1000-bit data per block, and a msg=1000 blocks is to be sent, use odd parity code (EDC),

- $\text{Msg}=10^6$ bits
- only one bit error
- only one error block needs retransmit
- the overhead is $(1001+1000x1)/1001x1001 = 0.002$;

use ECC,

- $1000+r+1 = 2^r$
- $r=10$
- each codeword has 1010 bits,
- the overhead is $10/1010 = 0.0099$. 

send bits columnwise
Polynomial Code, Cyclic Redundancy (CRC) Code

Polynomial Code, also called CRC code, is a class of Error Detecting Codes, it uses polynomial arithmetic to calculate the check bits. A bit string can be represented as a polynomial with coefficient 0 or 1.

110001 \( M(x) = x^5 + x^4 + 1 \), degree of \( M(x) \) = 5.

\( x^4 M(x) = x^9 + x^8 + x^4 \) 1100010000 multiply \( x^4 \) is equivalent to ¡§logical shift¡¨ \( M(x) \) to left by 4 bit. Polynomial arithmetic:

\[
\begin{array}{c|c}
10011011 & 10011011 \\
+ 11001010 & 11001010 \\
\hline
01010001 & 110101011 \\
\end{array}
\]

remainder

It sends checksumed code (frame) \( T(x) = x^r M(x) \) \( \forall \) \( r M(x) \% G(x) \) where \( M(x) \) is the msg and \( G(x) \) is the generator polynomial. \( T(x) \) is divisible by \( G(x) \). \( G(x) \) has degree of \( r \).

Assume that \( T(x) + E(x) \) is received, here \( E(x) \) represents the error bits,

If \( T(x) + E(x) \) is not divisible by \( G(x) \), then error is detected.

If \( T(x) + E(x) \) is divisible by \( G(x) \), then

case 1. \( E(x) = 0 \), no error.

case 2. \( E(x) \neq 0 \), and \( E(x) \) is divisible by \( G(x) \). What are these \( E(x) \)s?
Frame: 1101011011
Generator: 10011
Message after appending 4 zero bits: 110101100000

Transmitted frame: 110101101111110
Exercise on CRC code

Assume we use generator polynomial $G(x)=x^4+x+1$ for computing the checksum of a frame.

a) Given data=11111111, what is the checksum?

b) Transmission frame $T(x)=111111110100$

c) If not bit was corrupted, the Receiver receives $T(x)$

d) Receiver performs $T(X)/G(x)$ and remainder = 0 (Shown in the middle). That indicates no error.

c2) Assume bit 5 errors, receiving frame $R(x)=111101110100$

d2) The Receiver performs $R(x)/G(x)$ and get 1000 as remainder (Shown in the right) That indicates the frame got garbled.
CRC Code

It can be shown [PETE61] that all the following are not divisible by G(x).
1. All single-bit errors E(x)=x^i. If G(x) contains two terms. E(x)!= G(x)*H(x)
2. All double-bit errors E(x)=x^i+x^j=x^i(1+x^{j-i}), as long as G(x) has a factor with at least three terms.
3. Any odd number errors, i.e.,E(x=1)=1, as long as G(x) contains a factor (x+1).
   If G(x)=(x+1)*H(x), then G(x=1)=0*H(x)=0 -->E(x=1)=1 != G(x=1)
4. Any burst error for which the length of the burst is less than the length of checksum.
5. Most larger burst errors.
Four version of G(x) are widely used:

CRC-12= x^{12}+x^{11}+x^{3}+x^{2}+x^{1}+1=(x+1)(x^{11}+x^{2}+1)
CRC-16= x^{16}+x^{15}+x^{2}+1=(x+1)(x^{15}+x+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

They all contain x+1 as prime factor. CRC-12 is used for transmission of streams of 6-bit characters with 12-bit checksum. CRC-16 and CRC-CCITT are popular for 8-bit characters with 16-bit checksum. CRC-32 are used in IEEE802 standards with 32-bit checksum.
**CRC Generation Using Shift Registers**

Encoder for $g(x) = x^3 + x + 1$

- **$i(x)$**
  - $x^3$
  - $x^2$
  - $x$ (input)

- **$g(x)$**
  - $g_0 = 1$
  - $g_1 = 1$
  - $g_3 = 1$

- Shift registers:
  - **reg 0**
  - **reg 1**
  - **reg 2**

<table>
<thead>
<tr>
<th>Clock</th>
<th>Input</th>
<th>reg 0</th>
<th>reg 1</th>
<th>reg 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$1 = i_3$</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>$1 = i_2$</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>$0 = i_1$</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>$0 = i_0$</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Check bits:
- $r_0 = 0$
- $r_1 = 1$
- $r_2 = 0$