L15: Error Detection and Correction

Sebastian Magierowski
York University

Outline

• Basic (channel) coding ideas
• Error detection (Backward error correction)
  – single parity
  – interleaved parity (2-D parity)
  – internet checksum
  – polynomial codes
• Effectiveness and Error Models
• (Forward) Error correction
  – cyclic codes, block codes, convolutional codes, iterative codes
Types of Coding

- Some options
  - Line Coding
    - spectrum control
    - timing
    - basic error detection
  - Channel Coding
    - error detection
    - error correction
    - error prevention (combined detection & decoding)

Channel Coding

- Add in redundancy
- Two basic ideas, used in combination
  - error detection: recognizes an error in a frame (request re-send)
    - ARQ
  - error correction: finds error and corrects it (no need to re-send)
    - more desirable, but requires greater overhead (FEC)
Basic Ideas and Nomenclature

- Data consists of \( k \) bits
  - \( 2^k \) possible messages
- Add \( m \) redundant bits to this
- **Codeword** of \( n = m + k \) bits
  - \( 2^{m+k} = 2^n > 2^k \) possible strings
  - But only \( 2^k \) are valid!
  - \((n,k)\) codes e.g.: \((2,1)\)
- Thus coded messages (**codewords**) are separated in signal space
  - **Hamming distance**, \( d \): # of bit positions that differ
- **Code rate**: \( r = k/n \)
  - \( \frac{1}{2}, \frac{3}{4} \)

Detection and Correction Basic Example

- To detect \( d \) errors: need Hamming distance of \( d+1 \)
  - \( d + 1 \)
- To correct \( d \) errors: need Hamming distance of \( 2d+1 \)
  - \( 2d + 1 \)
- e.g.
  - 0000000000
  - 0000011111
  - 1111100000
  - 1111111111
  - \( d_{\text{min}} = 5 \)
  - detect up to 4 errors
  - correct up to 2 errors
  - only one at a time
System-Level Implementation

Simple Detection: Single Parity Code

- Information (7 bits): \((0, 1, 0, 1, 1, 0, 0)\)
- Parity Bit: \(b_8 = 0 + 1 + 0 + 1 + 1 + 0 = 1\)
- Codeword (8 bits): \((0, 1, 0, 1, 1, 0, 0, 1)\)
- If single error in bit 3: \((0, 1, 1, 1, 1, 0, 0, 1)\)
  - \# of 1's = 5, odd
  - Error detected
- If errors in bits 3 and 5: \((0, 1, 1, 1, 0, 0, 0, 1)\)
  - \# of 1's = 4, even
  - Error not detected
Single Parity Check: Formally

- Append an overall parity check to \( k \) information bits
  - Info Bits: \( b_1, b_2, b_3, ..., b_k \)
  - Check Bit: \( b_{k+1} = b_1 + b_2 + b_3 + ... + b_k \mod 2 \)
  - Codeword: \( (b_1, b_2, b_3, ..., b_k, b_{k+1}) \)
- All codewords have even # of 1s
- Redundancy: Single parity check code adds 1 redundant bit per \( k \) information bits: overhead = \( 1/(k + 1) \)
- Coverage
  - All error patterns that change an odd # of bits are detectable
  - All even-numbered patterns are undetectable
- Parity bit used in ASCII code

Effectiveness: Random Error Vector Model

- Effectiveness: Probability system fails to detect error
- Dependent on error model
  - Random Error Vector
  - Random Bit Error
  - Burst
- Random Error Vector
  - \( n \)-bit vector, \( e \), represents error pattern
    - \( e_i = 0 \) -> no error in position \( i \)
    - \( e_i = 1 \) -> an error in position \( i \)
- \( 2^n \) possible combinations
  - Assumes all possibilities equally likely
  - 50% chance of even number of errors
  - Therefore...????
What If Bit Errors are Random?

- Many transmission channels introduce bit errors at random, independent of each other, with probability $p$
- Some error patterns are more probable than others:
  - For example, if $p = 0.1$
    \[
    P[10000000] = p(1-p)^7 = 0.0478 \\
    P[11000000] = p^2(1-p)^6 = 0.0053
    \]
- In any worthwhile channel $p < 0.5$, and so $p/(1-p) < 1$
- It follows (can you show this?) that patterns with 1 error are more likely than patterns with 2 errors and so forth
- What is the probability that an undetectable error pattern occurs?

Effectiveness: Random Bit Error Model

- Undetectable error pattern if even # of bit errors:
  \[
P[\text{error detection failure}] = P[\text{undetectable error pattern}] = P[\text{error patterns with even number of 1s}]
  = \binom{n}{2} p^2 (1-p)^{n-2} + \binom{n}{4} p^4 (1-p)^{n-4} + \ldots
  \]
- Example: Evaluate above for $n = 32$, $p=10^{-3}$
  \[
P[\text{undetectable error}] = \binom{32}{2} (10^{-3})^2 (1 - 10^{-3})^{30} + \binom{32}{4} (10^{-3})^4 (1 - 10^{-3})^{28}
  \approx 496 (10^{-6}) + 35960 (10^{-12}) \approx 4.96 (10^{-4})
  \]
- For this example, roughly 1 in 2000 error patterns is undetectable
Two-Dimensional Parity Check (Interleaving)

- More parity bits to improve coverage
- Arrange information as rows
- Add single parity bit to each row
- Add a final “parity” row
- Used in early error control systems

\[
\begin{array}{cccccc}
1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 1 \\
\end{array}
\]

Last column consists of check bits for each row

Bottom row consists of check bit for each column

Error-Detecting Capability

- 1, 2, or 3 errors can always be detected
- Not all patterns >4 errors can be detected

Arrows indicate failed check bits
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

Internet Checksum

- Several Internet protocols (e.g. IP, TCP, UDP) use check bits in IP header to detect errors (or in the header and data for TCP/UDP)
- 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, \( b_0, b_1, b_2, \ldots, b_{L-1} \)
- The algorithm appends a 16-bit checksum \( b_L \)
Checksum Calculation

The checksum $b_L$ is calculated as follows:

- Treating each 16-bit word as an integer, find
  \[ x = (b_0 + b_1 + b_2 + \ldots + b_{L-1}) \mod (2^{16} - 1) \]

- The checksum is then given by:
  \[ b_L = -x \mod (2^{16} - 1) \]

Thus, the headers must satisfy the following pattern:

\[ 0 = (b_0 + b_1 + b_2 + \ldots + b_{L-1} + b_L) \mod (2^{16} - 1) \]

- The checksum calculation is carried out in software using one's complement arithmetic.

Internet Checksum Example

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0100</td>
<td></td>
<td>In the receiver</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>18</td>
<td>10010</td>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-3</td>
<td>1100</td>
</tr>
<tr>
<td>18 \mod (2^4 - 1)</td>
<td>0010</td>
<td>15</td>
<td>11110</td>
</tr>
<tr>
<td>= 3</td>
<td>+ 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>15 \mod (2^4 - 1)</td>
<td>1110</td>
<td></td>
</tr>
</tbody>
</table>

- Make checksum: -3 = 0 + 1
- 1100

\[ 1111 \]
Polynomial Codes

• Polynomials instead of vectors for codewords
• Polynomial arithmetic instead of checksums
• 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

General Idea

• Choose a special code: \( G \) (generator code, \( n=m+k \) bits)
• Shift information by \( m \) bits, \( +G \), and find remainder, \( R \)
  \[
  \frac{2^m I}{G} = Q \oplus \frac{R}{G}
  \]
  
• Make \( n=m+k \) bit codeword
  \[
  B = 2^m I \oplus R
  \]
  
  m-bit redundancy

• At receiver if no error:
  \[
  \frac{B}{G} = \frac{2^m I \oplus R}{G} = \frac{Q \oplus \frac{R}{G} \oplus \frac{R}{G}}{G} = Q
  \]

• At receiver if have error:
  \[
  \frac{B \oplus E}{G} = \frac{2^m I \oplus R \oplus E}{G} = \frac{C \oplus \frac{S}{G}}{G} \neq Q
  \]
Cyclic Error Correction

- We can do more than just detect...
- If have error:
  \[ \frac{B \oplus E}{G} = \frac{2^m I \oplus R \oplus E}{G} = \frac{C \oplus S}{G} \]
- But note:
  \[ \frac{B \oplus E}{G} = \frac{Q \oplus E}{G} = \frac{C \oplus S}{G} \]
- **Rearranging:**
  \[ \frac{E}{G} = \left[ \frac{Q \oplus C}{G} \right] \oplus \frac{S}{G} \]
  - remainder (syndrome) depends only on the error (not on codeword B)
  - Syndrome can be used to identify error
  - As simple as LUT

Cyclic Code Types

- Cyclic codes are a type of **block code**
  - redundant bits are generated by some block of data (contrast with convolutional code)
- BCH codes are a specific example
  - \((n,k,d)\)
  - \((7,4,3): \) code rate = \(4/7 = 0.571\) (2 detect, 1 correct)
  - \((15,5,7): \) code rate = \(5/15 = 0.333\) (6 detect, 3 correct)
- Reed-Solomon
  - operate on \(k\)-bit symbols (rather than individual bits)
  - and \(2^k-1\) symbols at a time (e.g. 8-bit symbol & 255 symbols total)
  - typical: \((255,233,33), \) therefore can correct \((33 – 1)/2 = 16\) symbols
  - \(8 \times 16 = 128\) bits in a \(8 \times 255 = 2040\) bit sequence
  - very good for burst errors (DSL, cable, satellite, CDs)
Convolutional Codes

• Codes continuously
  – good for streaming, don’t have to pause to collect blocks of bits

• Data is shifted through registers
  – output depends on present and past inputs (state-machine)
  – this redundancy achieves the necessary coding

• NASA convolutional code (Voyager)
  – (2,1), \( r = \frac{1}{2} \)
  – constraint length = 7
  – GSM, 802.11

• Trellis decoding
  – Viterbi algorithm

Recent Iterative Codes

• Turbo codes, 1993
  – two codes generated and interleaved
  – two decoders work iteratively to decode message
  – close to Shannon limit

• Low Density Parity Check, 1962 & 2003
  – block code
  – each output bit formed from only a fraction of input bits
  – iteratively re-assembled
  – rapidly being incorporated (no IP issues)
    • digital video, 10 Gbps ethernet, power line, latest 802.11