Lecture 5 Overview

- Error handling through redundancy
  - Adding extra bits to the frame

- Hamming Distance
  - When we can detect
  - When we can correct

- Checksum
Error Detection

- Implemented at many layers
  - We’ll mainly focus on link-layer techniques today
Basic Idea

- The problem is data itself is not self-verifying
  - Every string of bits is potentially legitimate
  - Hence, any errors/changes in a set of bits are equally legit

- The solution is to reduce the set of potential bitstrings
  - Not every string of bits is allowable
  - Receipt of a disallowed string of bits means the original bits were garbled in transit

- Key question: which bitstrings are allowed?
Let’s start simple, and consider fixed-length bitstrings
- Reduce our discussion to $n$-bit substrings
- E.g., 7-bits at a time, or 4 bits at a time (4B/5B)
- Or even a frame at a time

We call an allowable sequence of $n$ bits a codeword
- Not all strings of $n$ bits are codewords!
- The remaining $n$-bit strings are “space” between codewords

Rephrasing previous question: how many codewords with how much space between them?
Hamming Distance

- Distance between legal codewords
  - Measured in terms of number of bit flips

- Efficient codes are of uniform Hamming Distance
  - All codewords are equidistant from their neighbors
2d+1 Hamming Distance

- Can **detect** up to 2d bit flips
  - The next codeword is always 2d+1 bit flips away
  - Any fewer is guaranteed to land in the middle

- Can **correct** up to d bit flips
  - We just move to the closest codeword
  - Unfortunately, no way to tell how many bit flips
    » E.g., 1, or (2d+1)-1?

CSE 123 – Lecture 5: Error Handling
Encoding

- We’re going to send only codewords
  - Non-codewords indicate errors to receiver

- But we want to send any set of strings
  - Need to embed arbitrary input into sequence of codewords

- We’ve seen this before: 4B/5B
  - We want more general schemes
Simple Embedding: Parity

- Code with Hamming Distance 2
  - Can detect one bit flip (no correction capability)
- Add extra bit to ensure odd(even) number of ones
  - Code is 66% efficient (need three bits to encode two)
  - Note: Even parity is simply XOR
Simple Correction: Voting

- Simply send each bit $n$ (3 in this example) times
  - Code with Hamming Distance 3 ($d=1$)
  - Can detect 2 bit flips and correct 1
- Straightforward duplication is extremely inefficient
  - We can be much smarter about this
### Two-Dimensional Parity

- **Start with normal parity**
  - $n$ data bits, 1 one parity bit
- **Do the same across rows**
  - $m$ data bytes, 1 parity byte
- **Can detect up to 3 bit errors**
  - Even most 4-bit errors
- **Can correct any 1 bit error**
  - Why?

<table>
<thead>
<tr>
<th>Data</th>
<th>Parity Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0101001</td>
<td>1</td>
</tr>
<tr>
<td>1101001</td>
<td>0</td>
</tr>
<tr>
<td>1011110</td>
<td>1</td>
</tr>
<tr>
<td>0001110</td>
<td>1</td>
</tr>
<tr>
<td>0110100</td>
<td>1</td>
</tr>
<tr>
<td>1011111</td>
<td>0</td>
</tr>
<tr>
<td>1111011</td>
<td>0</td>
</tr>
</tbody>
</table>
Per-Frame Detection Codes

- Want to add an error detection code per frame
  - Frame is unit of transmission; all or nothing.
  - Computed over the entire frame—including header! Why?
- Receiver checks EDC to make sure frame is valid
  - If frame fails check, throw it away
- We could use error-correcting codes
  - But they are less efficient, and we expect errors to be rare
Checksums

- Simply sum up all of the data in the frame
  - Transmit that sum as the EDC

- Extremely lightweight
  - Easy to compute fast in hardware
  - Fragile: Hamming Distance of 2

- Also easy to modify if frame is modified in flight
  - Happens a lot to packets on the Internet

- IP packets include a 1’s compliment checksum
IP Checksum Example

- 1’s compliment of sum of words (not bytes)
  - Final 1’s compliment means all-zero frame is not valid

```c
u_short cksum(u_short *buf, int count) {
    register u_long sum = 0;
    while (count--) {
        sum += *buf++;
        if (sum & 0xFFFF0000) {
            /* carry occurred, so wrap around */
            sum &= 0xFFFF;
            sum++;
        }
    }
    return ~(sum & 0xFFFF);
}
```
Checksum in Hardware

- Compute checksum in Modulo-2 Arithmetic
  - Addition/subtraction is simply XOR operation
  - Equivalent to vertical parity computation

- Need only a word-length shift register and XOR gate
  - Assuming data arrives serially
  - All registers are initially 0
Checksum Example

010100111101001010111101000111010110100111011111011110110

Data

Parity Byte

CSE 123 – Lecture 5: Error Handling
Checksum Example

Data

Parity Byte

0101001111010010101011110100111011011111011110110

0 0 0 0 0 0 0 0 + 0101...
Checksum Example

01010011110100101011110100011101011010011011111011110110

Data 0
Checksum Example

010100111101001010111101000111010110100110111110111101101110

Data 01

CSE 123 – Lecture 5: Error Handling
Checksum Example

Data 010

01010011101001010111101001110101101001101111101110110

0 0 0 0 0 1 0 + 1001...

CSE 123 – Lecture 5: Error Handling
Checksum Example

01010011110100101011110100011101011010011011111011110110

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

Data ↑ 0101

\[ + \] 0011…
Checksum Example

01010011110100101011110100011101011010011011111011110110

01001
01
1
0
1
0
1
1
+
1101...

Data \(\uparrow\) 01010011
Checksum Example

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

Data: 01010011
Parity Byte: 1

Parity: \( \sum \) 1010...
Checksum Example

0101001110100101011101000111010110100111011110111101110110

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

Data 01010011
Parity Byte 10

Parity byte:

01010011
11

+ 0100...
Checksum Example

0101001110100101011110100011101011010011011111011110110

1 0 0 0 0 0 0 1 + 1011...

Data

Parity Byte

CSE 123 – Lecture 5: Error Handling
Checksum Example

0101001110100101011110100011101011010011011111011110110

0101001111010010101110100011101011010011011111011110110

Data

Parity Byte

01010011
11010010
1
Parity Byte

0
Checksum Example

```
0101001111010010101111010001110101101001101111101110110
```

```
01010011  1  1  1  1  0  1  1  0
```

```
11110110 Parity Byte
```

```
0101001111010010101111010001110101101001101111101110110
```

```
01110110
```

```
01010011  1  1  1  1  0  1  1  0
```

Data

Parity Byte
For Next Class

- Read 2.5 in P&D
- Discussion section today!