Lecture 3 Overview

- Framing wrap-up
  - Clock-based framing

- Error handling through redundancy
  - When we can detect
  - When we can correct

- Parity-based schemes
Given the above received string of bits (HDLC framing sequences highlighted), how many stuffed bits are there?

A. 0
B. 3
C. 5
D. 7
Framing/Bit Stuffing (HDLC) Review

Given the above received string of bits (HLDC framing sequences highlighted), how many stuffed bits are there?

A. 0
B. 3
C. 5 (highlighted in yellow above)
D. 7
Byte Stuffing

- Same as bit stuffing, except at byte (character) level
  - Generally have two different flags, **STX** and **ETX**
  - Found in PPP, DDCMP, BISYNC, etc.

- Need to stuff if either appears in the payload
  - Prefix with another special character, **DLE** (data-link escape)
  - New problem: what if DLE appears in payload?

- Stuff DLE with DLE!
  - Could be as bad as 50% efficient to send all DLEs
Clock-Based Framing

- So far, we’ve based framing on what’s on the wire
  - Any bit errors may throw off our framing
  - What happens with missed flag? Spurious flag?

- An alternative is to base framing on external clock
  - This is what SONET does, among others

- Significant engineering tradeoffs
  - No extra bits needed in the data stream itself, but…
  - Need tight clock synchronization between sender and receiver
When Things Go Wrong

- Clock drift may confuse frame boundaries
  - Read the end of one frame and beginning of the next

- What happens if there are bit errors on channel?
  - We might misinterpret sentinels as data or vice versa
  - What will the frames look like?

- In general, need some way to make sure we’re OK
  - Error detection—and perhaps correction
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*
  - Counter example: satellite communication
Error-Detecting Codes

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

- 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
- In practice, we likely want to do it for a whole frame
- For now, let’s reduce our discussion to $n$-bit substrings (e.g., 8 bits 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

- **Hamming Distance** = 3

---

**Is this code efficient?**

A. Yes
B. No
C. I’m not sure
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
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
Simple Detection: Parity

- Add extra bit to ensure odd(even) number of ones
  - Code has a rate of 2/3 (need three bits to encode two)
  - Note: Even parity is simply XOR

What is the Hamming Distance of parity?
A. 0
B. 1
C. 2
D. I’m not sure
Simple Correction: Voting

- Simply send each bit $n$ (3 in this example) times
  - Code with Hamming Distance 3 ($d=1$)

- Straightforward duplication is extremely inefficient
  - We can be much smarter about this

How many bit flips can this code correct?
A. 0
B. 1
C. 3
D. I’m not sure
For Next Class

- We’ll finish error detection and talk about reliable transport on Friday
- Read 2.5 in P&D