

Lecture 3:

Error Handling

CSE 123: Computer Networks
Alex C. Snoeren





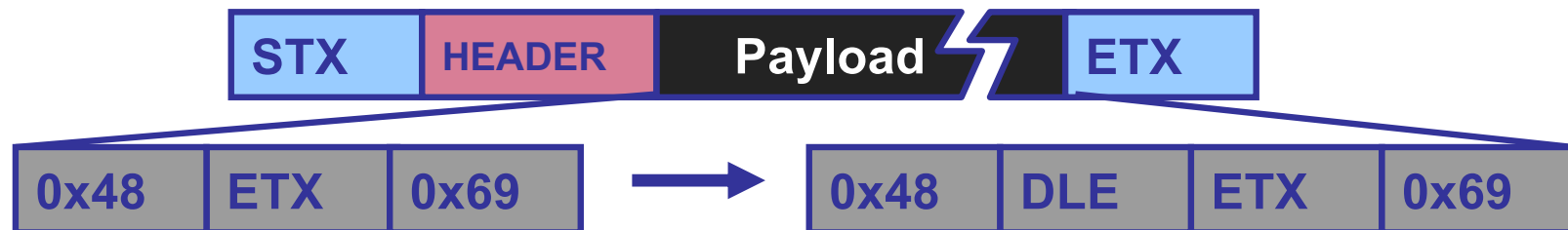
Lecture 3 Overview

- Framing wrap-up
 - ◆ Clock-based framing
- Error handling through redundancy
- Hamming Distance
 - ◆ When we can detect
 - ◆ When we can correct
- Parity-based schemes

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
 - ◆ Kind of like Phy-layer signaling: sample at specific intervals
 - ◆ 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



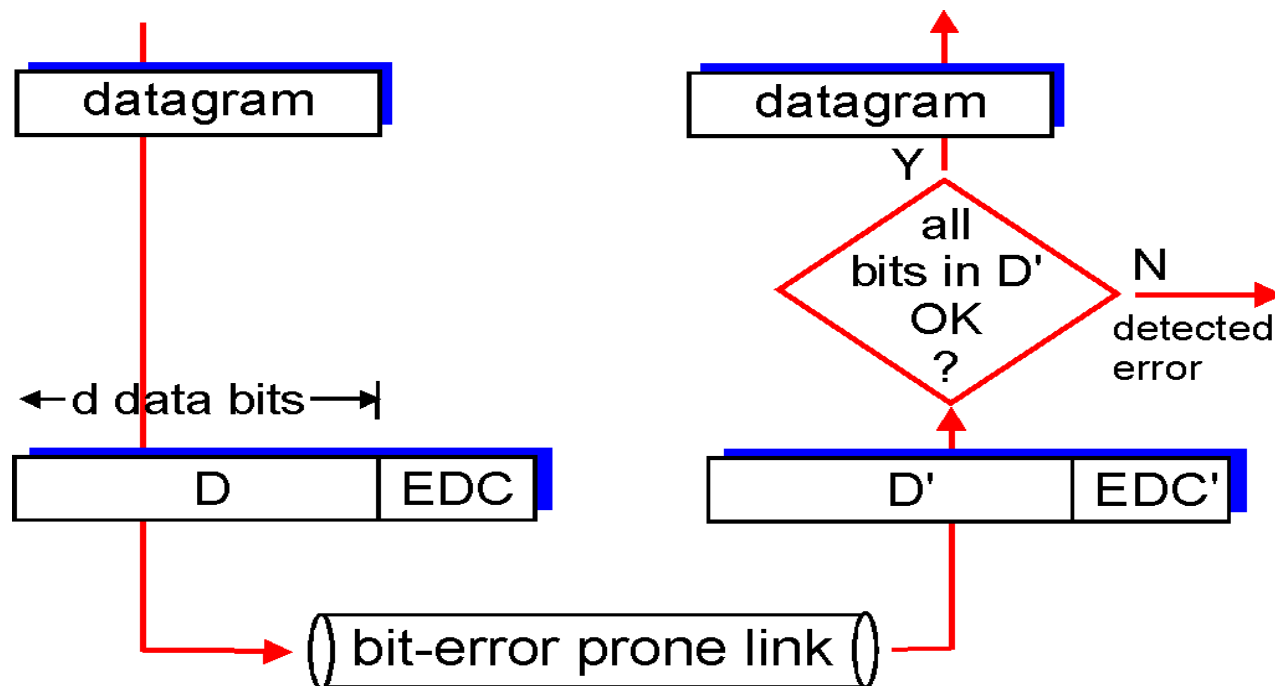
When Things Go Wrong

- 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

Error Detection



- Implemented at many layers
 - ◆ We'll mainly focus on link-layer techniques today



Basic Idea



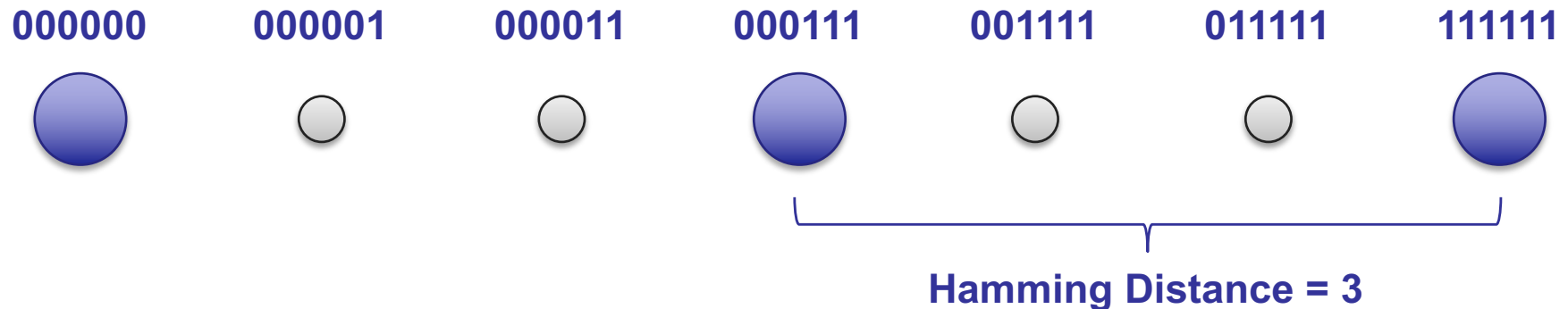
- The problem is data itself is not self-verifying
 - ◆ Every string of bits is potentially legitimate
 - ◆ Hence, any errors/changes in a set of bits are equally legit
- The solution is to reduce the set of potential bitstrings
 - ◆ Not every string of bits is allowable
 - ◆ Receipt of a disallowed string of bits means the original bits were garbled in transit
- Key question: which bitstrings are allowed?

Codewords



- 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

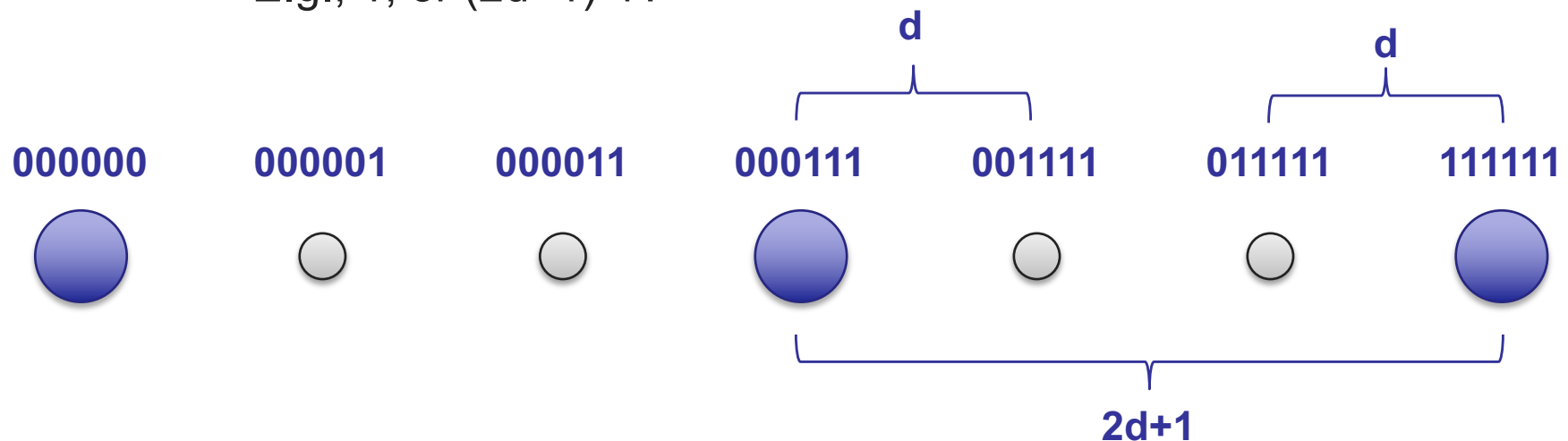


- Distance between legal codewords
 - ◆ Measured in terms of number of bit flips
- **Efficient** codes are of uniform Hamming Distance
 - ◆ All codewords are equidistant from their neighbors



$2d+1$ 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 $(2d+1)-1$?

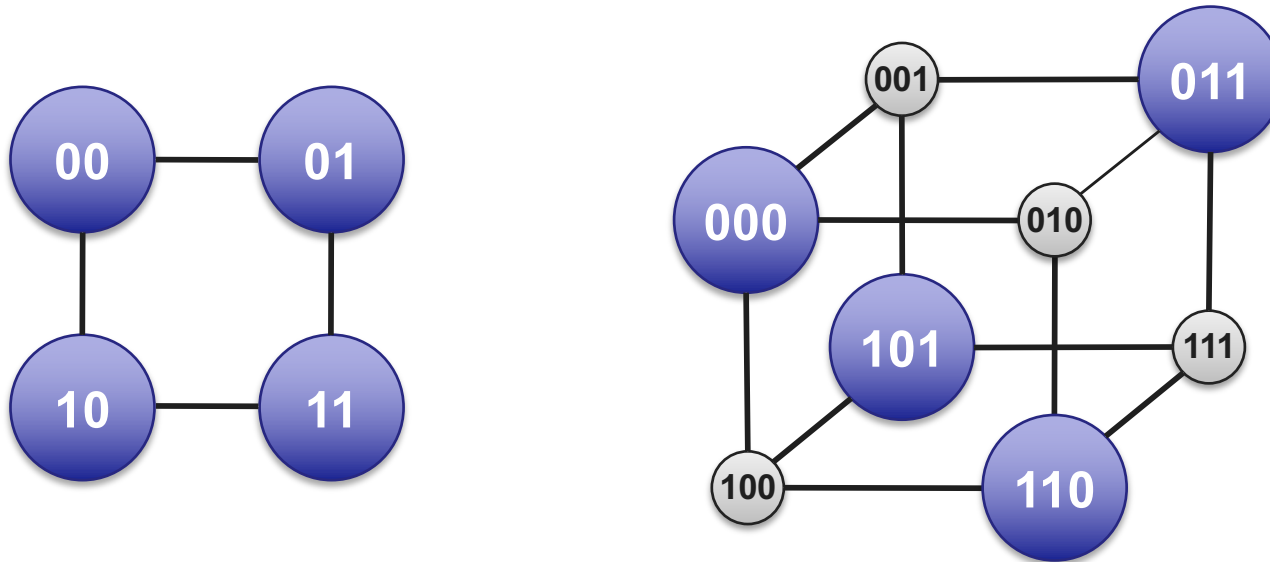


Encoding



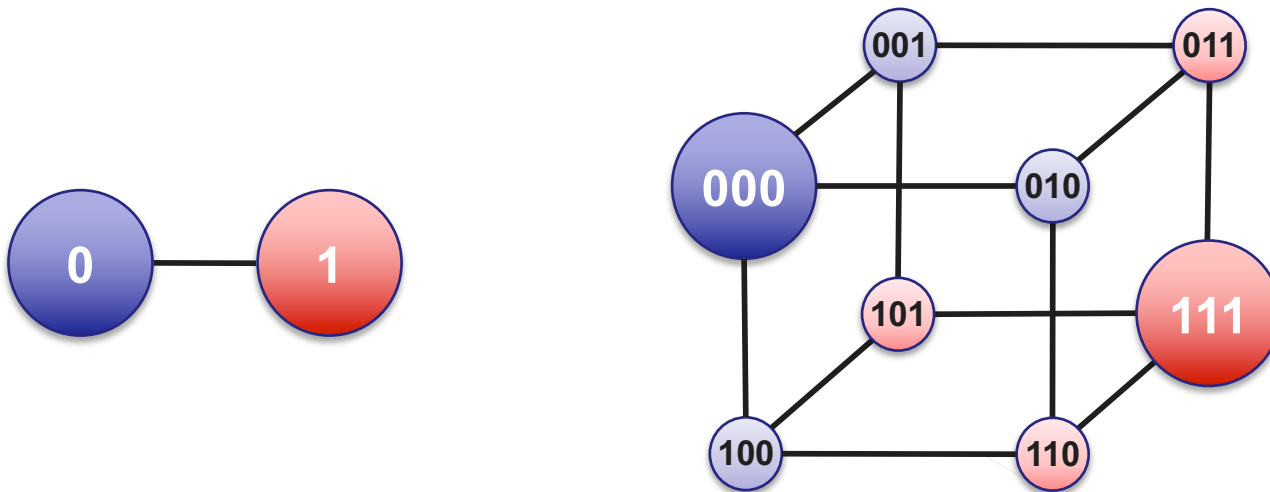
- We're going to send only codewords
 - ◆ Non-codewords indicate errors to receiver
- But we *want* to send any set of strings
 - ◆ Need to embed arbitrary input into sequence of codewords

Simple 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



		Parity Bits
Data	0101001	1
	1101001	0
	1011110	1
	0001110	1
	0110100	1
	1011111	0
Parity Byte	1111011	0

- 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?

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

For Next Class



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