

Lecture 3:

Framing and Error Handling

CSE 123: Computer Networks

Aaron Schulman



DISCUSSION TODAY@2pm

Intro to Project 1



UCSDCSE

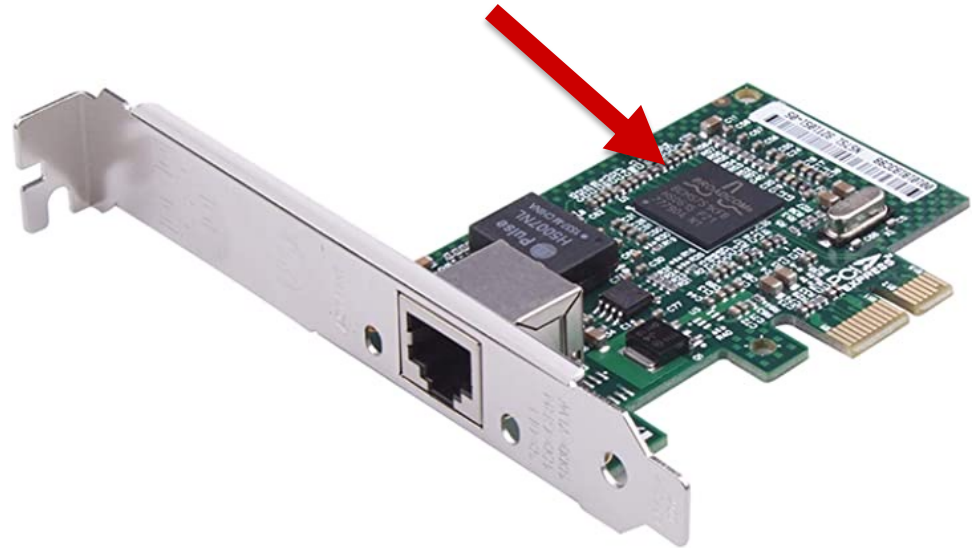


Project 1 is out

- Implement simple link layer protocol (Layer 2!)
- **Framing and error checking**

Why?

- Help you understand how it works.
- Link layers change and update frequently, you may be working on one!





Lecture 3 Overview

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



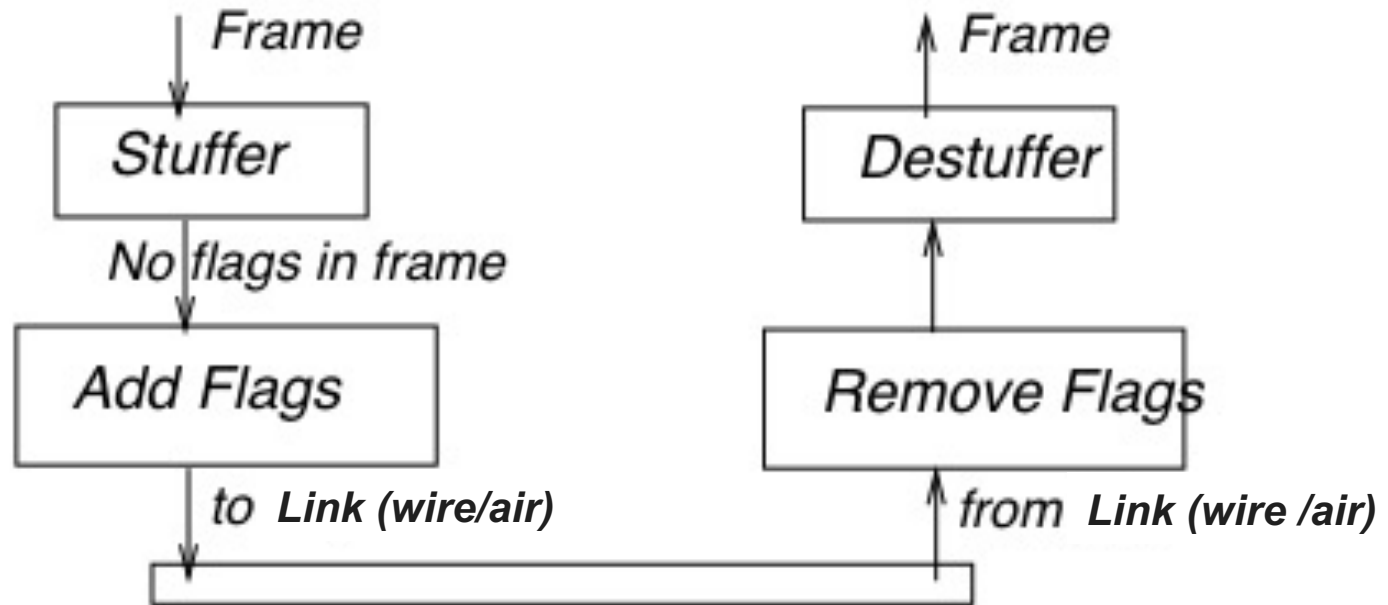
Sentinel-based Framing

- Allow for variable length frames
- Idea: mark start/end of frame with special “marker”
 - ◆ Byte pattern, bit pattern, signal pattern
- But... must make sure marker doesn't appear in data
- Two solutions
 - ◆ Special non-data physical-layer symbol
 - » Impact on efficiency (can't use symbol for data) of code
 - ◆ Stuffing
 - » Dynamically remove marker bit patterns from data stream
 - » Receiver “unstuffs” data stream to reconstruct original data

Stuffing



- Insert bytes/bits into data stream to make sure that sentinel (flag) does not appear in payload





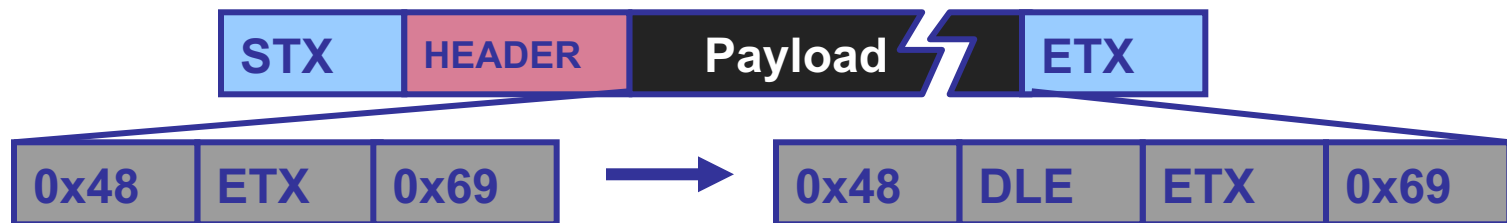
- Stuffed bits** 01111110000111011101111011111001
011111010000111011101111100111110001

- ## CSE 123 – Lecture 3: Framing and Error Handling



Byte Stuffing

- Same as bit stuffing, except at byte (character) level
 - ◆ Generally have two different flags, **STX** and **ETX**
 - ◆ Found in PPP, DDCMP, BISYNC, Embedded Systems etc.
- Need to stuff if either appears in the payload
 - ◆ Prefix byte with 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 signal-Based Framing



- So far, we've based framing on what's on the wire
 - ◆ Any bit errors may throw off our framing (dropped bits!)
 - ◆ What happens with missed delimiter? Spurious delimiter?
- An alternative is to base framing on external clock signal
 - ◆ Use some signal to indicate beginning of frame, and wait fixed time until frame ends
 - ◆ This is what SONET (metro optical network) does, among others
- Significant engineering tradeoffs
 - ◆ No extra bits needed in the data stream itself, but...
 - ◆ Need tight clock synchronization between sender and receiver

Clock-based example: SONET



- Synchronous Optical NETwork
 - ◆ Historically used to backhaul telephone calls and Internet traffic
 - ◆ Engineering goal is to reduce delay and buffering
- All frames take same amount of time
 - ◆ Independent of how fast bits in frame are sent (bit rate)!
 - ◆ Can have atomic clock at each end of the link so always in sync
- Each frame starts with special signal bits
 - ◆ Can sync clock—look for special periodic signal bits
 - ◆ No need to stuff; signal pattern is unlikely, so won't be periodic in data

Even with clock-based framing things go wrong!

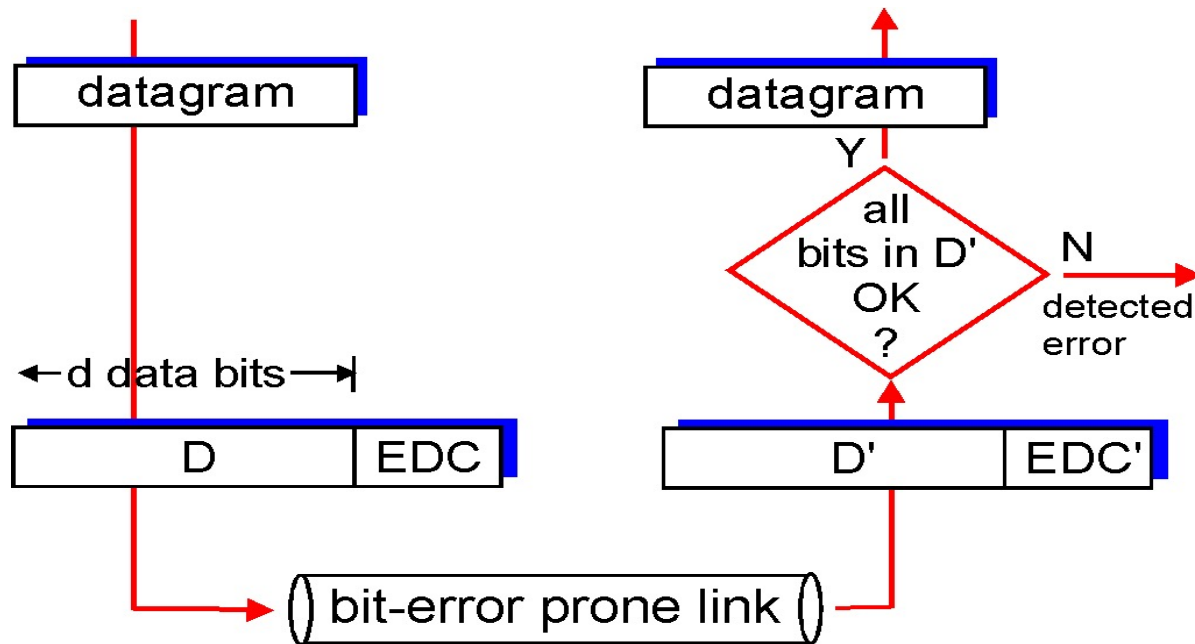


- Clock drift may confuse frame boundaries
 - ◆ May not have atomic clocks so clocks can get out of sync
 - ◆ Result: read the end of one frame and beginning of the next ☹
- What happens if there are **bit errors** on channel?
 - ◆ We might misinterpret clock sync signals (sentinels) as data or vice versa
- In general, need some way to make sure we're OK
 - ◆ Error detection—and perhaps correction



Error Detection

- Implemented at many layers (end-to-end argument)
 - ◆ We'll mainly focus on link-layer techniques today





Basic Idea

- The problem is headers/payload 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?

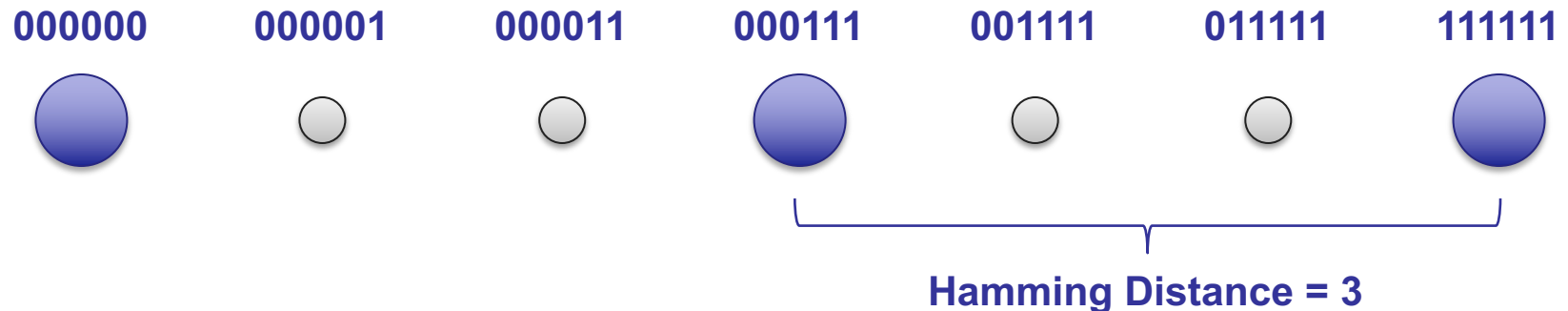
Codewords



- 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
 - ◆ Codewords represent shorter $< n$ bit sequences (add redundancy)
- What bit strings are allowed? → How many codewords with how much space between them?



Hamming Distance

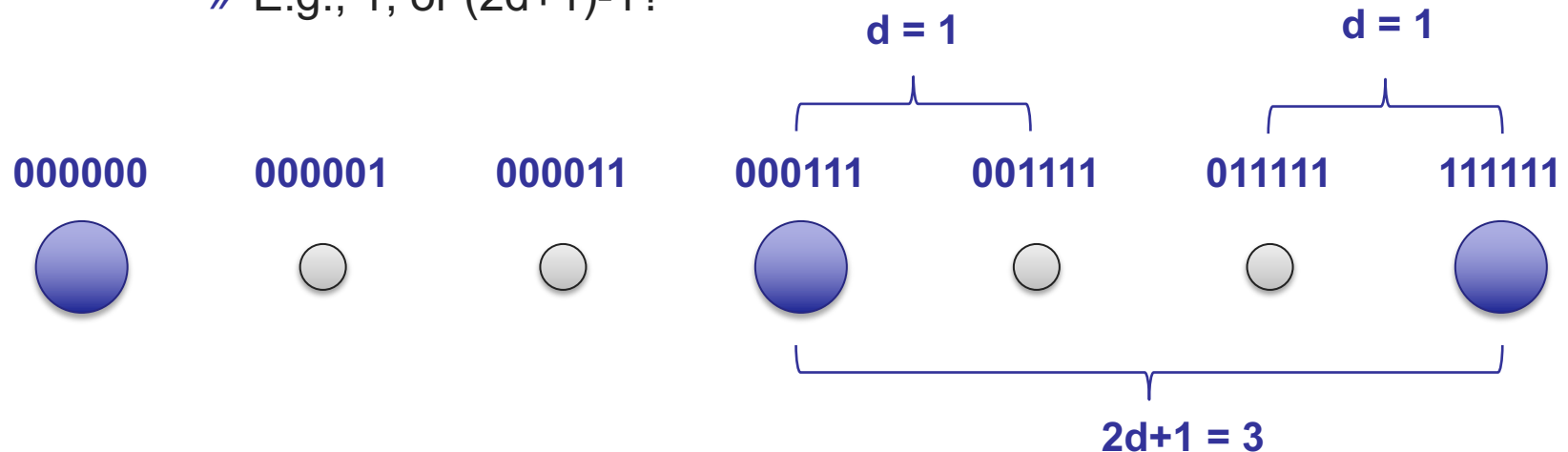


- Distance between legal codewords
 - ◆ Measured in terms of number of bit flips
- **Efficient** codes are of uniform Hamming Distance
 - ◆ All codewords are equidistant from their neighbors



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





For Next Class

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