

Lecture 3:

Error Handling

CSE 123: Computer Networks
Alex C. Snoeren



Lecture 3 Overview



- Link performance
- Error handling through redundancy
- Hamming Distance
 - ♦ When we can detect
 - ♦ When we can correct
- Specific schemes

Link Performance

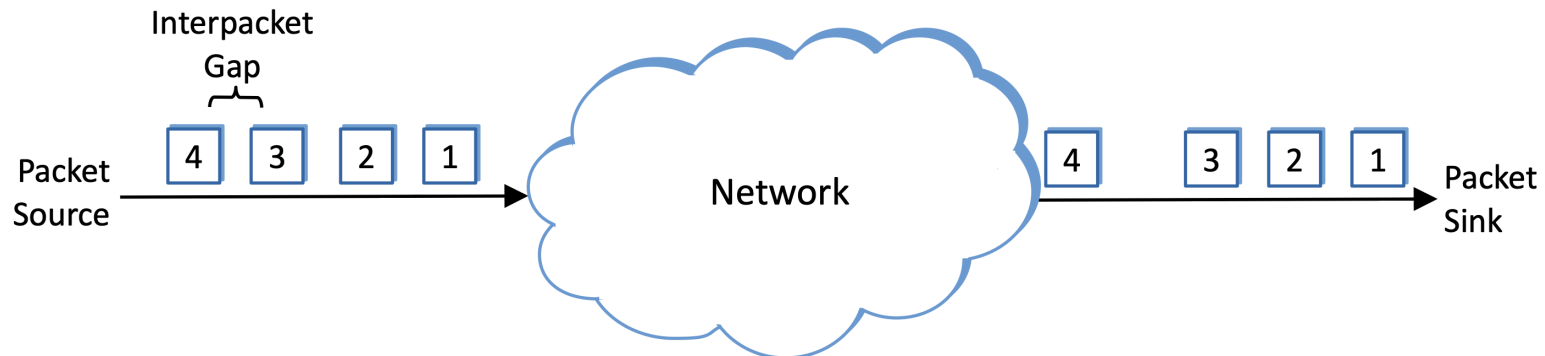


- Hosts take a certain amount of time to transmit (**serialization delay**)
 - ◆ Link capacity often measured in bits per second (**data rate**)
 - ◆ Sometimes colloquially referred to as “bandwidth”
- And then it takes a while to traverse the link (**propagation delay**)
 - ◆ If we want to hear back from the other host, takes a **round-trip time (RTT)**
- Ideal throughput requires hosts to constantly use the full data rate
 - ◆ Need to keep a certain amount of data “in flight”, the **bandwidth-delay product**

Under-driven Links



- Most links aren't "full", meaning there are gaps between frames



- Uneven **interpacket gaps** lead to **jitter** which can impact applications
 - ◆ Leads to "stuttering" in video playback
 - ◆ Hence, modern video players include a playback buffer to hide jitter

When Things Go Wrong



- What happens if there are **bit errors** on channel?
 - ♦ What will the frames look like?
 - ♦ We might mis-interpret header fields (like length!)
- In general, need some way to make sure we're OK
 - ♦ Error detection—and perhaps correction



Per-Frame Detection Codes

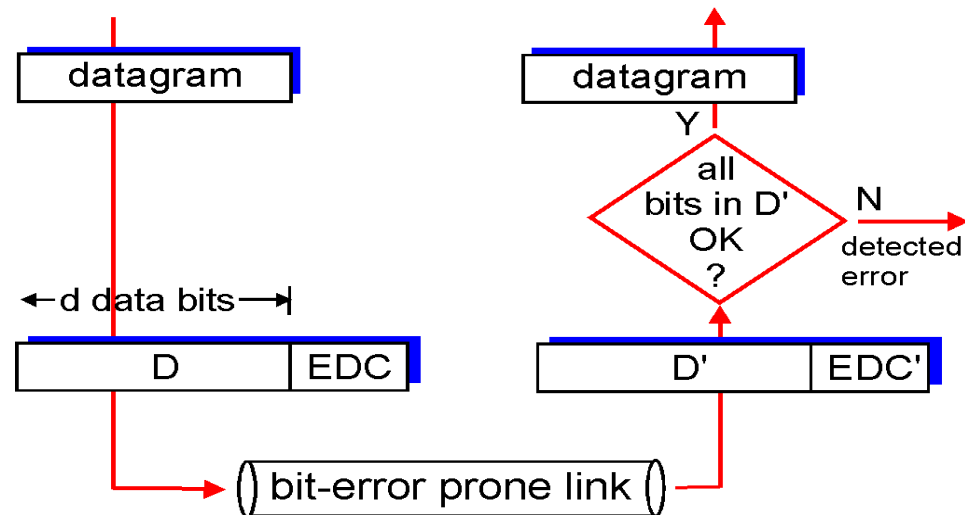


- Want to add an **error detection code** per frame
 - ♦ Frame is unit of transmission; all or nothing.
 - ♦ Computed over the entire frame—including header! Why?
- Receiver checks EDC to make sure frame is valid
 - ♦ If frame fails check, throw it away
- We *could* use error-correcting codes
 - ♦ But 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



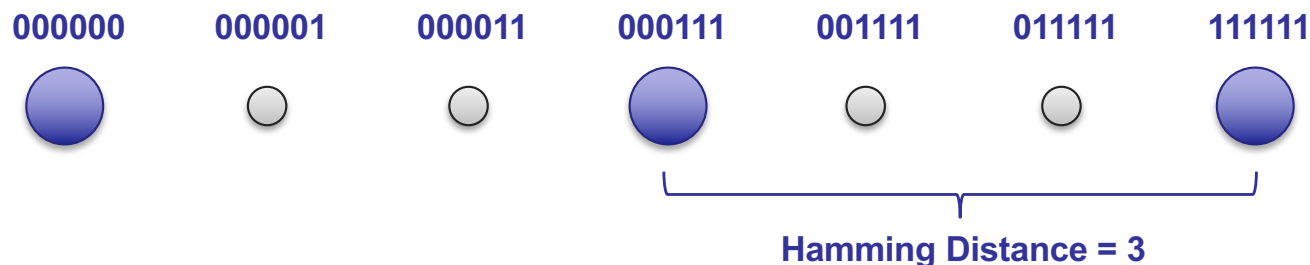
- The problem is data itself is not self-verifying
 - ◆ Every string of bits is potentially legitimate
 - ◆ Hence, any errors/changes in a set of bits are equally legit
- The solution is to reduce the set of potential bitstrings
 - ◆ Not every string of bits is allowable
 - ◆ Receipt of a disallowed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

Codewords



- 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

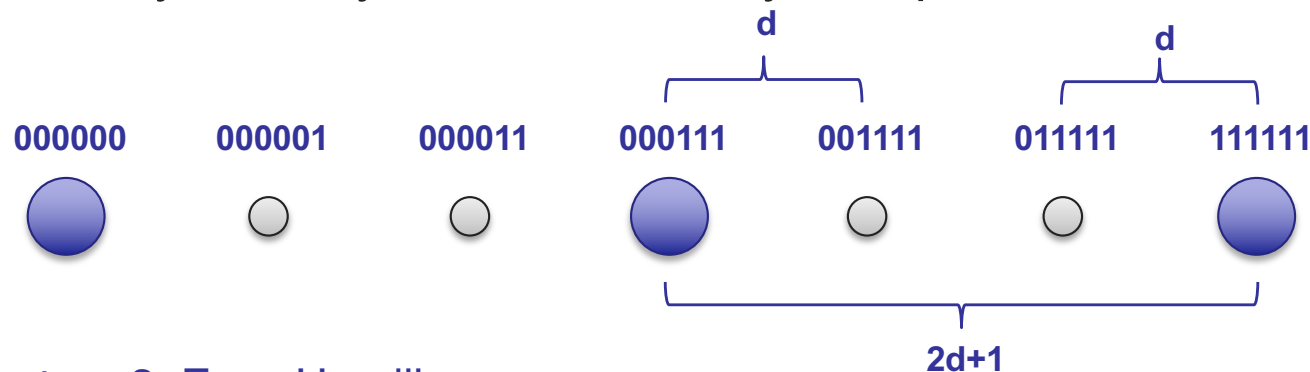


- Distance between legal codewords
 - ◆ Measured in terms of number of bit flips
- Distance of the code is minimum distance between any two codewords
 - ◆ Minimum means you can't reach any (other) codeword by flipping fewer bits
- **Efficient** codes are of uniform Hamming Distance
 - ◆ All codewords 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

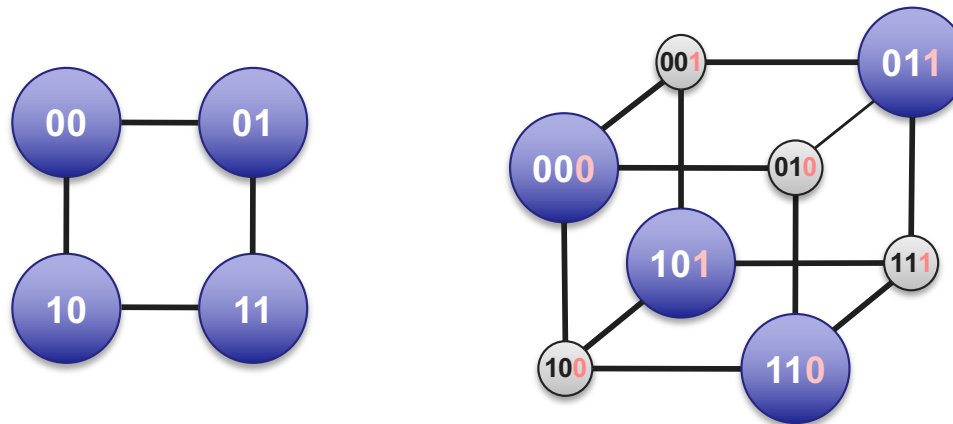


Encoding



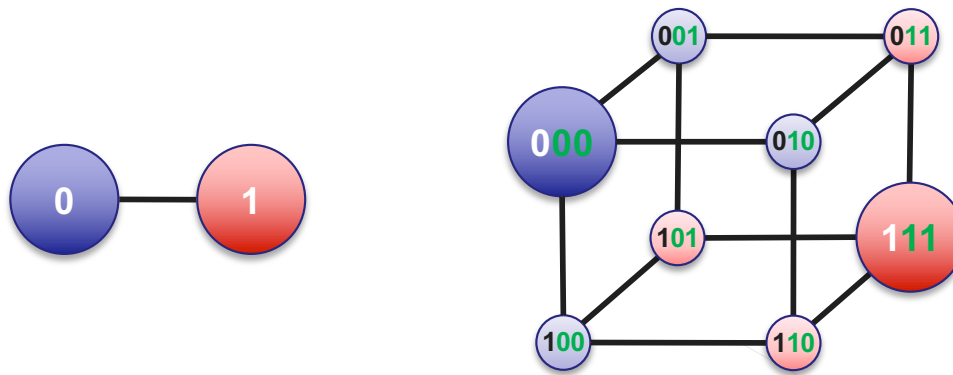
- We're going to send only codewords
 - ♦ Non-codewords indicate errors to receiver
- But we *want* to send any set of strings
 - ♦ Need to embed arbitrary input into sequence of codewords
- Basic idea is to "embed" arbitrary k -bit strings into $n > k$ bit code
 - ♦ There are 2^k codewords among the 2^n set of n -bit strings

Simple Detection: Parity



- EDC is just an extra bit computed 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

Simple Correction: Voting



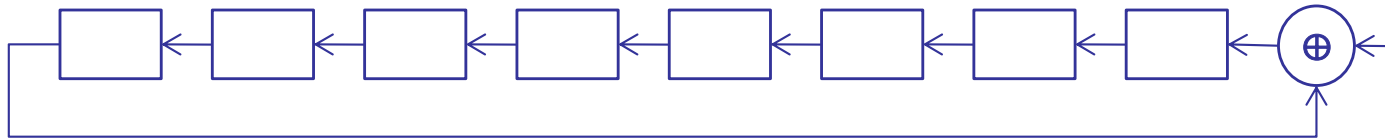
- EDC just duplicates data bit n (2 in this example) times
 - ◆ Code with Hamming Distance 3 ($d=1$)
- Straightforward duplication is extremely inefficient
 - ◆ We can be much smarter about this...

Checksums



- EDC is the sum of the data in the frame
- Extremely lightweight
- Also easy to modify if frame is modified in flight
 - ♦ Happens a lot to packets on the Internet
- IP packets include a 1's complement checksum

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



From Sums to Remainders

- Checksums are easy to compute, but very fragile
 - ◆ In particular, **burst** errors are frequently undetected
 - ◆ We'd rather have a scheme that "smears" parity
- Need to remain easy to implement in hardware
 - ◆ All we need are shift registers and an XOR gate
- We'll stick to Modulo-2 arithmetic
 - ◆ Multiplication and division are XOR-based as well



Cyclic Redundancy Check

- Idea is to *divide* the incoming data, D , rather than add
 - ♦ The divisor is called the **generator**, g
- We can make a CRC resilient to k -bit burst errors
 - ♦ Need a generator of $k+1$ bits
- Divide $2^k D$ by g to get remainder, r
 - ♦ Remainder is called **frame check sequence**
- Send $2^k D - r$ (i.e., $2^k D$ XOR r)
 - ♦ Note $2^k D$ is just D shifted left k bits
 - ♦ Remainder must be at most k bits
- Receiver checks that $(2^k D - r)/g = 0$

For Next Class



- Read Chapter 12-12.1 in P&D
- Get started on HW 1, due a week from today at 10am