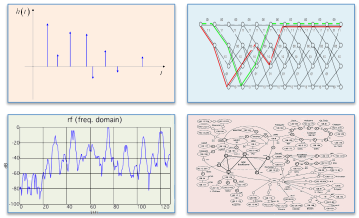


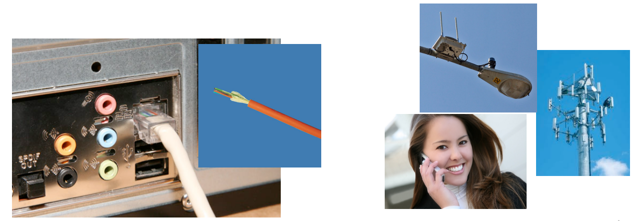
INTRODUCTION TO EECS II

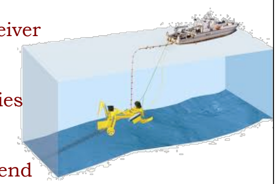
DIGITAL COMMUNICATION SYSTEMS

6.02 Spring 2012
Lecture #3

- Errors in communication
- Redundancy with coding (intro)

Physical Links are Inherently *Analog*





Wire: Send signals of different voltages; receiver measures voltage

Optical: send signals with different intensities (possibly at different wavelengths)

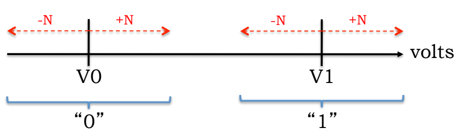
Radio/Acoustic: A bit trickier, but we can send at different amplitudes

Digital Signaling: Map Bits to Signals

To ensure we can distinguish signal from noise, we'll *map bits to signals* using a fixed set of discrete values. For example, in a *bipolar signaling* (or *bipolar mapping*) scheme we use two voltages:

V0 is the binary value "0"
V1 is the binary value "1"

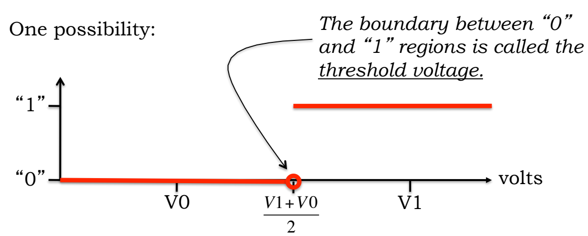
At the receiver,
- Voltages near V0 would be interpreted as representing "0"
- Voltages near V1 would be interpreted as representing "1"
- If we space V0 and V1 far enough apart, we can tolerate some degree of noise, N



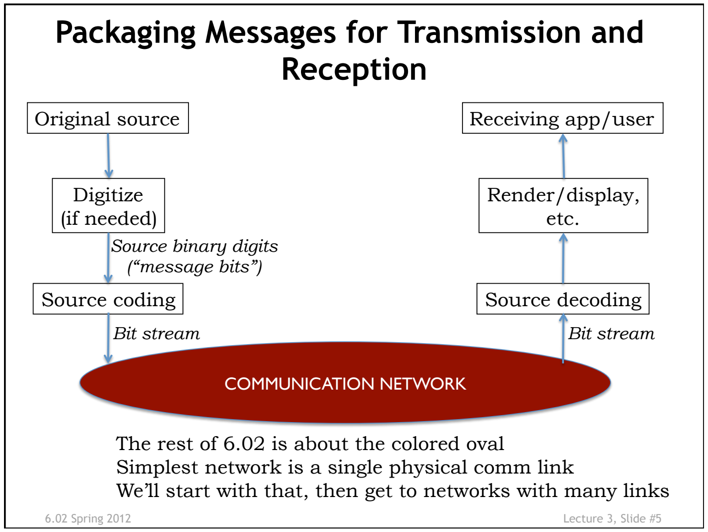
Digital Signaling: Receiving

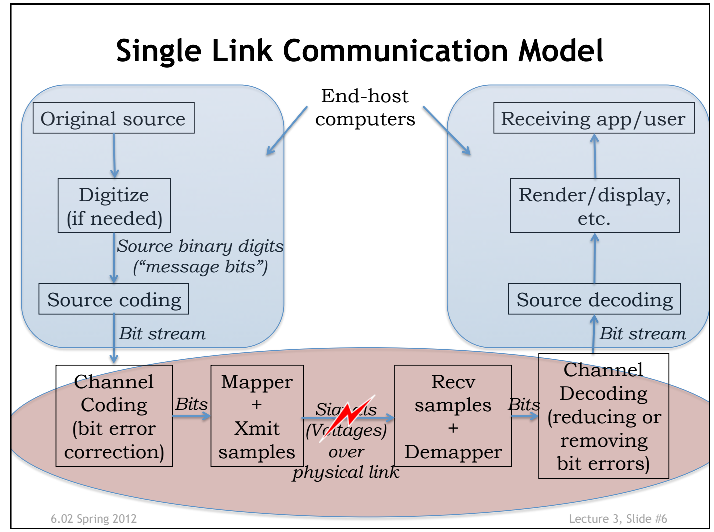
We can specify the behavior of the receiver with a graph that shows how incoming voltages are mapped to "0" and "1".

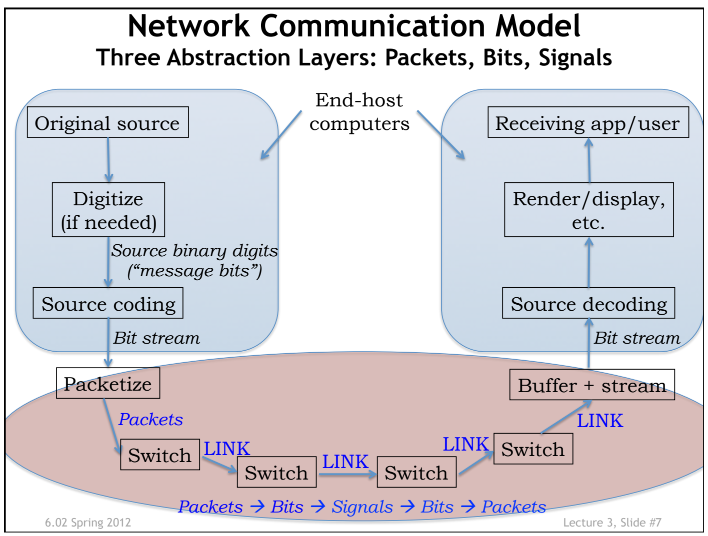
One possibility:

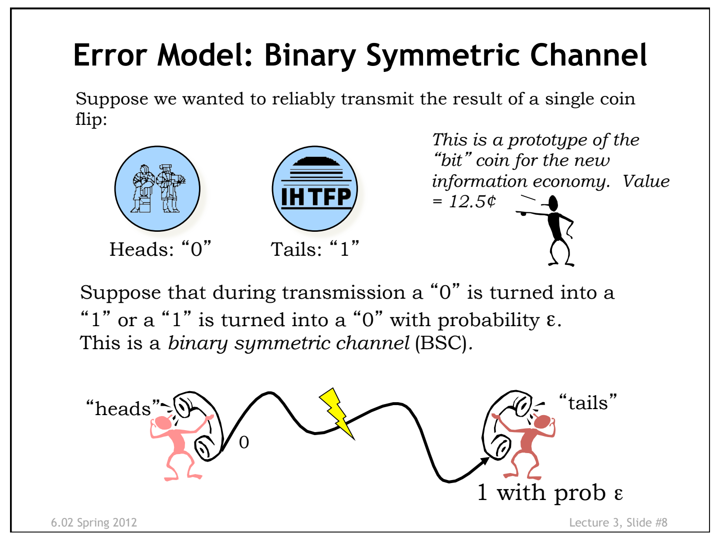


If received voltage between V0 & $\frac{V1+V0}{2}$ → '0', else '1'

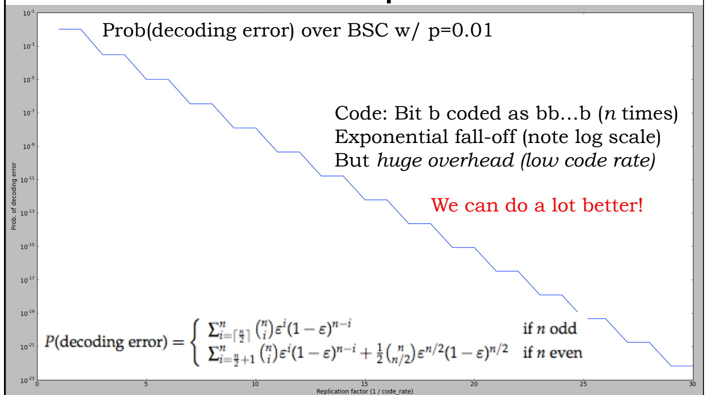

Packaging Messages for Transmission and Reception
Original source
Digitize (if needed)
Source binary digits (“message bits”)
Source coding
Bit stream
COMMUNICATION NETWORK
Bit stream
Source decoding
Render/display, etc.
Receiving app/user
The rest of 6.02 is about the colored oval
Simplest network is a single physical comm link
We’ll start with that, then get to networks with many links
6.02 Spring 2012
Lecture 3, Slide #5


Single Link Communication Model
End-host computers
Original source
Digitize (if needed)
Source binary digits (“message bits”)
Source coding
Bit stream
Channel Coding (bit error correction)
Bits
Mapper + Xmit samples
over physical link
Recv samples + Demapper
Bits
Channel Decoding (reducing or removing bit errors)
Bit stream
Source decoding
Render/display, etc.
Receiving app/user
6.02 Spring 2012
Lecture 3, Slide #6


Network Communication Model
Three Abstraction Layers: Packets, Bits, Signals
End-host computers
Original source
Digitize (if needed)
Source binary digits (“message bits”)
Source coding
Bit stream
Packetize
Packets
Switch
LINK
Switch
LINK
Switch
LINK
Switch
LINK
Buffer + stream
Bit stream
Source decoding
Render/display, etc.
Receiving app/user
Packets → Bits → Signals → Bits → Packets
6.02 Spring 2012
Lecture 3, Slide #7


Error Model: Binary Symmetric Channel
Suppose we wanted to reliably transmit the result of a single coin flip:
Heads: “0”
Tails: “1”
IHTFP
This is a prototype of the “bit” coin for the new information economy. Value = 12.5¢
Suppose that during transmission a “0” is turned into a “1” or a “1” is turned into a “0” with probability ε.
This is a binary symmetric channel (BSC).
“heads”
0
“tails”
1 with prob ε
6.02 Spring 2012
Lecture 3, Slide #8

Performance of Replication Code



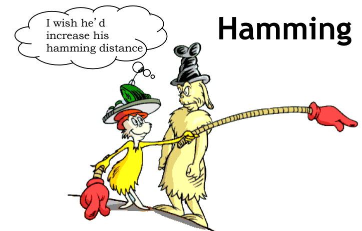
Code: Bit b coded as bb…b (n times)
Exponential fall-off (note log scale)
But *huge overhead (low code rate)*

We can do a lot better!

$$P(\text{decoding error}) = \begin{cases} \sum_{i=\lceil \frac{n}{2} \rceil}^{n} \binom{n}{i} \varepsilon^i (1-\varepsilon)^{n-i} & \text{if } n \text{ odd} \\ \sum_{i=\frac{n}{2}+1}^{n} \binom{n}{i} \varepsilon^i (1-\varepsilon)^{n-i} + \frac{1}{2}\binom{n}{n/2} \varepsilon^{n/2} (1-\varepsilon)^{n/2} & \text{if } n \text{ even} \end{cases}$$

Replication factor, n (1/code_rate)

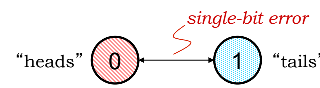
Hamming Distance



The number of bit positions in which the corresponding bits of two encodings of the same length are different

The Hamming Distance (HD) between a valid binary code word and the same code word with e errors is e.

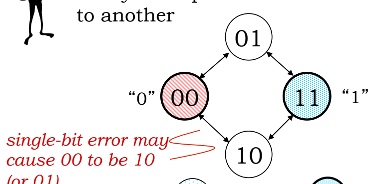
The problem with no coding is that the two valid code words ("0" and "1") also have a Hamming distance of 1. So a single-bit error changes a valid code word into another valid code word…



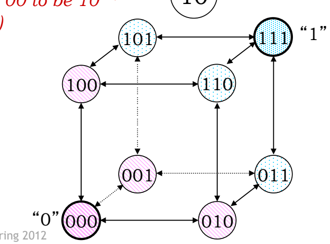
What is the Hamming Distance of the replication code?

Idea: Embedding for Structural Separation

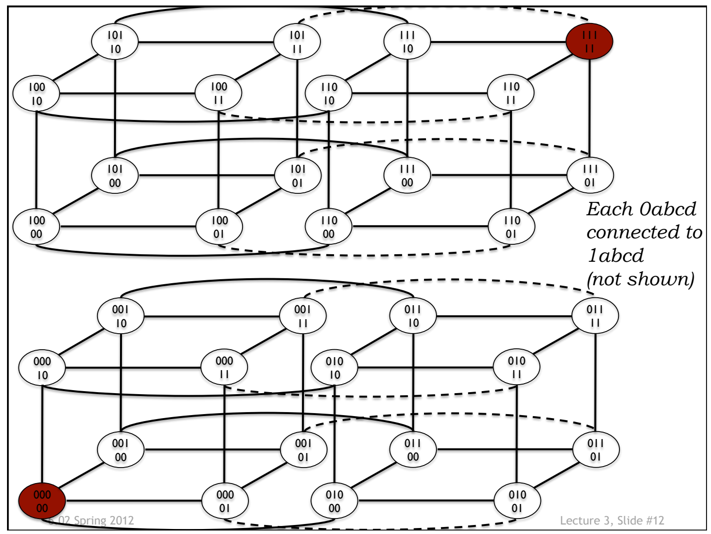
Encode so that the codewords are "far enough" from each other
Likely error patterns shouldn't transform one codeword to another

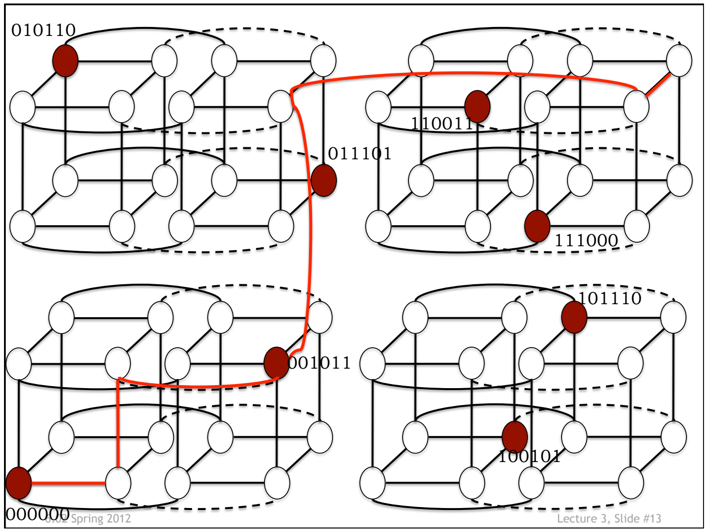


Code: nodes chosen in hypercube + mapping of message bits to nodes



If we choose 2^k out of 2^n nodes, it means we can map all k-bit message strings in a space of n-bit *codewords*. The *code rate* is k/n.





Hamming Distance of Code v. Detection & Correction Capabilities

If D is the minimum Hamming distance between codewords, we can detect **all** patterns of <= (D-1) bit errors

If D is the minimum Hamming distance between codewords, we can correct **all** patterns of $\left\lfloor \frac{D-1}{2} \right\rfloor$ or fewer bit errors

The Hamming distance is satisfies the triangle inequality.

How to Construct Codes?

0000000	1100001	1100110	0000111
0101010	1001011	1001100	0101101
1010010	0110011	0110100	1010101
1111000	0011001	0011110	1111111

Want: 4-bit messages with single error correction (min HD=3)

Quick, produce a code, i.e., a set of codewords, with this property!

Gaining Some Insight: Parity Calculations

We can add single-bit error detection to any length code word by adding a *parity bit* chosen to guarantee the Hamming distance between any two valid code words is at least 2.

Parity: addition in GF(2): 0+0=0, 1+0=0+1=1, 1+1=0
multiplication: 0*0=0*1=1*0 =0, 1*1=1

GF(2) arithmetic: Can count by summing the bits in the word modulo 2 (equivalent to XOR'ing the bits together).

A Simple Code: Parity Check

- Add a parity bit to message of length k to make the total number of "1" bits even (aka "even parity").
- If the number of "1"s in the received word is *odd*, there there has been an error.

0 1 1 0 0 1 0 1 0 0 1 1 → original word with parity bit
0 1 1 0 0 0 0 1 0 0 1 1 → single-bit error (detected) bit
0 1 1 0 0 0 1 1 0 0 1 1 → 2-bit error (not detected) bit

- Hamming distance of parity check code is 2
  - Can detect all single-bit errors
  - In fact, can detect all odd number of errors
  - But cannot detect even number of errors
  - And cannot correct any errors

Linear Block Codes

Block code: k message bits encoded to n code bits
I.e., each of 2^k messages encoded into a unique n-bit combination via a *linear transformation.*
Set of parity equations (in GF(2)) represents code.

Key property: Sum of any two codewords is *also* a codeword → necessary and sufficient for code to be linear.

(n,k) code has rate k/n.
Sometime written as (n,k,d), where d is the Hamming Distance of the code.

Examples: What are n, k, d here?

{000, 111} (3,1,3). Rate= 1/3.

{0000, 1100, 0011, 1111} (4,2,2). Rate = ½.

{00000} {5,0,_). Rate = 0!

{1111, 0000, 0001}
{1111, 0000, 0010, 1100} → Not linear codes!

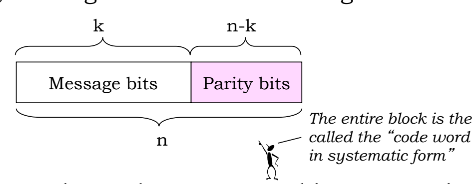
0000000	1100001	1100110	0000111
0101010	1001011	1001100	0101101
1010010	0110011	0110100	1010101
1111000	0011001	0011110	1111111

(7,4,3) code. Rate = 4/7.

The HD of a linear code is the number of "1"s in the non-zero codeword with the smallest # of "1"s

(n,k) Systematic Linear Block Codes

- Split data into k-bit blocks
- Add (n-k) parity bits to each block using (n-k) linear equations, making each block n bits long



- Every linear code can be represented in systematic form