• 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:

The boundary between “0” and “1” regions is called the threshold voltage.

If received voltage between V0 & \( \frac{V1 + V0}{2} \) \( \rightarrow \) ‘0’, else ‘1’
Packaging Messages for Transmission and Reception

Original source

Digitize (if needed)

Source binary digits ("message bits")

Source coding

Bit stream

RECEIVING APP/USER

Render/display, etc.

Source decoding

Bit stream

COMMUNICATION NETWORK

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

Single Link Communication Model

Original source

Digitize (if needed)

Source binary digits ("message bits")

Source coding

Bit stream

RECEIVING APP/USER

Render/display, etc.

Source decoding

Bit stream

CHANNEL CODING (bit error correction)

MAPPER

BIT STREAM

RECV SAMPLES + DEMAPPER

BIT STREAM

CHANNEL DECODING (reducing or removing bit errors)

Network Communication Model

Three Abstraction Layers: Packets, Bits, Signals

Original source

Digitize (if needed)

Source binary digits ("message bits")

Source coding

Bit stream

PACKETS

SWITCH

LINK

BUFFER + STREAM

RECEIVING APP/USER

Render/display, etc.

Source decoding

Bit stream

Error Model: Binary Symmetric Channel

Suppose we wanted to reliably transmit the result of a single coin flip:

Heads: "0"

Tails: "1"

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 \( \varepsilon \).

This is a binary symmetric channel (BSC).

"heads" ~ 0

"tails" ~ 1 with prob \( \varepsilon \)
Performance of Replication Code

Prob(decoding error) over BSC w/ p=0.01

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!

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.

Each 0abcd connected to 1abcd (not shown)
### Hamming Distance of Code v. Detection & Correction Capabilities

If $D$ is the minimum Hamming distance between codewords, we can detect all patterns of $\leq (D-1)$ 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 satisfies the triangle inequality.

### How to Construct Codes?

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

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

<table>
<thead>
<tr>
<th>Codeword</th>
<th>Parity Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>010101</td>
<td></td>
</tr>
<tr>
<td>101001</td>
<td></td>
</tr>
<tr>
<td>111100</td>
<td></td>
</tr>
<tr>
<td>011100</td>
<td></td>
</tr>
<tr>
<td>100101</td>
<td></td>
</tr>
<tr>
<td>111000</td>
<td></td>
</tr>
<tr>
<td>110011</td>
<td></td>
</tr>
<tr>
<td>101101</td>
<td></td>
</tr>
<tr>
<td>001011</td>
<td></td>
</tr>
<tr>
<td>011101</td>
<td></td>
</tr>
<tr>
<td>100101</td>
<td></td>
</tr>
<tr>
<td>111111</td>
<td></td>
</tr>
</tbody>
</table>

### 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 has been an error.

\[
\begin{align*}
011001010011 & \rightarrow \text{original word with parity bit} \\
011000010011 & \rightarrow \text{single-bit error [detected] bit} \\
0110000110011 & \rightarrow \text{2-bit error [not detected] bit}
\end{align*}
\]

- 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 \( \rightarrow \) 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?

\[
\begin{align*}
\{000, 111\} & \quad (3,1,3). \text{ Rate= } 1/3. \\
\{0000, 1100, 0011, 1111\} & \quad (4,2,2). \text{ Rate = } \frac{1}{2}. \\
\{00000\} & \quad (5,0,\_). \text{ Rate = } 0! \\
\{1111, 0000, 0001\} & \quad \text{Not linear codes!} \\
\{1111, 0000, 0010, 1100\} & \quad \text{The HD of a linear code is the number of “1”s in the non-zero codeword with the smallest # of “1”s}
\end{align*}
\]

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

\( (n,k) \) 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

\[
\begin{align*}
\text{Message bits} & \quad \text{Parity bits} \\
\text{n-k} & \quad \text{n}
\end{align*}
\]

The entire block is the called the “code word in systematic form”

- Every linear code can be represented in systematic form