The Physical layer gives us a stream of bits. How do we interpret it as a sequence of frames?

...10110 ...

Um?
Framing Methods

• We’ll look at:
  – Byte count (motivation)
  – Byte stuffing
  – Bit stuffing

• In practice, the physical layer often helps to identify frame boundaries
  – E.g., Ethernet, 802.11
Byte Count

- First try:
  - Let’s start each frame with a length field!
  - It’s simple, and hopefully good enough ...
 Byte Count (2)

- How well do you think it works?
Byte Count (3)

- Difficult to re-synchronize after framing error
  - Want a way to scan for a start of frame
Byte Stuffing

- Better idea:
  - Have a special flag byte value that means start/end of frame
  - Replace ("stuff") the flag inside the frame with an escape code
  - Complication: have to escape the escape code too!
Byte Stuffing (2)

• Rules:
  – Replace each FLAG in data with ESC FLAG
  – Replace each ESC in data with ESC ESC
Byte Stuffing (3)

- Now any unescaped FLAG is the start/end of a frame
Bit Stuffing

• Can stuff at the bit level too
  – Call a flag six consecutive 1s
  – On transmit, after five 1s in the data, insert a 0
  – On receive, a 0 after five 1s is deleted
Bit Stuffing (2)

• Example:

Data bits

0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Transmitted bits with stuffing
Bit Stuffing (3)

- So how does it compare with byte stuffing?

Data bits: 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Transmitted bits with stuffing: 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 0 1 0 0 1 0

Stuffed bits
Topic

• Some bits will be received in error due to noise. What can we do?
  – Detect errors with codes »
  – Correct errors with codes »
  – Retransmit lost frames ← Later

• Reliability is a concern that cuts across the layers – we’ll see it again
Problem – Noise may flip received bits

<table>
<thead>
<tr>
<th>Signal</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slightly</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Noisy</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Very noisy</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Approach – Add Redundancy

- **Error detection codes**
  - Add check bits to the message bits to let some errors be detected

- **Error correction codes**
  - Add more check bits to let some errors be corrected

- Key issue is now to structure the code to detect many errors with few check bits and modest computation
Motivating Example

• A simple code to handle errors:
  – Send two copies! Error if different.

• How good is this code?
  – How many errors can it detect/correct?
  – How many errors will make it fail?
Motivating Example (2)

• We want to handle more errors with less overhead
  – Will look at better codes; they are applied mathematics
  – But, they can’t handle all errors
  – And they focus on accidental errors (will look at secure hashes later)
Using Error Codes

- Codeword consists of D data plus R check bits (=systematic block code)

\[
\begin{array}{|c|c|}
\hline
\text{Data bits} & \text{Check bits} \\
\hline
D & R=fn(D) \\
\hline
\end{array}
\]

- Sender:
  - Compute R check bits based on the D data bits; send the codeword of D+R bits
Using Error Codes (2)

- **Receiver:**
  - Receive D+R bits with unknown errors
  - Recompute R check bits based on the D data bits; error if R doesn’t match R’

![Diagram showing data bits and check bits with function R=fn(D) and comparison symbol]
Intuition for Error Codes

• For D data bits, R check bits:

  All codewords

  Correct codewords

• Randomly chosen codeword is unlikely to be correct; overhead is low
R.W. Hamming (1915-1998)

• Much early work on codes:
  – “Error Detecting and Error Correcting Codes”, BSTJ, 1950

• See also:
  – “You and Your Research”, 1986
Hamming Distance

- Distance is the number of bit flips needed to change $D_1$ to $D_2$

- **Hamming distance** of a code is the minimum distance between any pair of codewords
Hamming Distance (2)

• Error detection:
  – For a code of distance $d+1$, up to $d$ errors will always be detected
Hamming Distance (3)

- Error correction:
  - For a code of distance $2d+1$, up to $d$ errors can always be corrected by mapping to the closest codeword.